Л.М. Круподьорова
А.М. Пєтух

Технологія програмування мовою Сі
Частина 1

Міністерство освіти і науки України
Вінницький національний технічний університет

Л.М. Круподьорова
А.М.Пєтух

Технологія програмування мовою Сі

Частина 1

Затверджено Вченою радою Вінницького національного технічного університету як навчальний посібник для студентів напряму підготовки 0804 – "Комп'ютерні науки" спеціальностей "Інтелектуальні системи прийняття рішень" та "Програмне забезпечення автоматизованих систем". Протокол № 10 від 27 квітня 2006 р.

Вінниця ВНТУ 2006

УДК 681.31
К 84

Рецензенти:
С.М. Москвіна, кандидат технічних наук, доцент
О.Д. Азаров, доктор технічних наук, професор
О.М. Роік, доктор технічних наук, професор
О.В. Сілагін, к. т. н., директор ТОВ " І Т І "

Рекомендовано до видання Вченою радою Вінницького національного технічного університету Міністерства освіти і науки України

Круподьорова Л.М., Пєтух А.М.
К 84 **Технологія програмування мовою Сі. Частина 1.**
Навчальний посібник. – Вінниця: ВНТУ, 2006. – 206 с.

В навчальному посібнику розглянуті основи програмування мовою Сі, історія виникнення мови, технологія створення різних програм і робота в середовищі програмування Сі, наведено багато прикладів і рекомендації для їх засвоєння. У посібнику пропонується огляд стандартних бібліотечних функцій мови програмування Сі, що розміщені за алфавітом. Для кожної з функцій є опис прототипу та зміст її призначення. Посібник може бути корисним для розв'язування завдань з лабораторних та практичних вправ, а також стати в нагоді програмістам будь-якої кваліфікації, що використовують Сі.

Навчальний посібник призначений для студентів напряму підготовки 0804 "Комп'ютерні науки" спеціальностей "Інтелектуальні системи прийняття рішень" та "Програмне забезпечення автоматизованих систем".

УДК 681.31

Зміст

Введення

Вашій увазі пропонується теоретичний курс з програмування на мові C/C++, прочитаного у університеті. Даний курс включає три семестри. Цей посібник присвячений вивченню основ мови, основних навичок і знань, необхідних для програмування. Хоча викладення ведеться на прикладі мови C/C++, багато даних аспектів важливих як для повноцінного освоєння сучасної комп'ютерної техніки, так і для програмування взагалі, незалежно від конкретної мови програмування. Викладання матеріалу ведеться з використанням інтегрованого середовища розробника Borland C++ 3.1, хоча більшість програм може бути перенесена і в інші компілятори зовсім без змін або з мінімальними змінами.

Матеріал викладений у вигляді 13 розділів, відповідних програмі даного курсу. У кожному розділі розглядаються найважливіші аспекти і наводяться приклади. Також наведені найбільш використовувані функції мови і приклади їх використовування. У посібнику пропонується огляд близько 180 стандартних бібліотечних функцій алгоритмічної мови програмування Турбо Сі, що розміщені за алфавітом. Викладено основні, найбільш широко вживані функції мови Сі, можливі способи їхнього застосування до розв'язування ряду задач з практичного програмування. Для кожної з функцій є опис прототипу та зміст її призначення.

Автори сподіваються, що даний посібник допоможе студентам у вивченні мови C/C++ і стимулюватиме інтерес до самостійного вивчення основ програмування не тільки на даній мові.

При роботі рекомендується використовувати збірник задач, розроблений спеціально для цього курсу.

1 Чому C++?

1.1 Коротка історія мови Сі

Мова Сі була створена на початку 70-х років Денісом Річчі, який працював в компанії Bell Telephone Laboratories. Родовід мови Сі бере свій початок від мови Алгол і включає Паскаль.

Сі було розроблено як мову для програмування в новій на ті часи операційній системі Unix. ОС Unix була написана на мові асемблера для ЕОМ PDP-7 і перенесена потім на PDP-11. На мову Сі значно вплинув її попередник, мова Бі, створена Кеном Томпсоном, яка в свою чергу є послідовником мови BCPL. Мова BCPL була створена в 1969г. Мартіном Річардсом в рамках проекту "Комбінована мова програмування" в Кембріджському університеті в Лондоні. Незабаром Unix була переписана мовою Сі, і в 1974-75 роках ОС Unix фірми Bell Laboratories стала першим комерційним продуктом, реалізовуючим ідею про те, що операційна система

може бути успішно написана мовою високого рівня, якщо ця мова є достатньо могутньою і гнучкою.

У 1978г. Брайан Керніган і Деніс Річчі написали книгу "Мова програмування Сі" (видавництво Prentice-Hall). Ця робота, яка в своєму крузі називалася "білою книгою" і "K & R" в решті світу, стала стандартом опису мови Сі. На момент створення "K & R" існували компілятори мови Сі для ЕОМ PDP-11, Interdata 8/32, Honeywell 6000 і IBM 370. Надалі цей список був продовжений.

В кінці 70-х почали з'являтися транслятори Сі для МІКРОЕОМ на процесорах 8080 і Z80 з операційною системою CP/M. Скотт Газері і Джим Гібсон розробили і пустили в продаж Тіпу-С ("Крихітний Сі") – інтерпретатор, заснований на підмножині мови Сі. Його інтерактивне середовище програмування дуже схоже на ту, що має надзвичайно популярний транслятор Basic фірми Microsoft. У 1980р. Рон Кейн створив свій компілятор Small-C ("Малий Сі") для ОС CP/M і мікропроцесора 8080. Компілятор Small-C, заснований на підмножині мови Сі, був написаний на самому Small-C. Проблема "курки і яйця" була розв'язана, коли Кейн створив першу версію компілятора на основі інтерпретатора Тіпу-С. Потім Small-C методом розкручування створив самого себе, коли Кейн разом з іншими, використовуючи ранні версії компілятора, зробив досконаліший компілятор. Small-C компілює початковий модуль на мові Сі в модуль на мові асемблера процесора 8080. Слід зазначити, що Кейн надав свій компілятор і його початковий текст в суспільну власність.

Приблизно в цей же час Лео Золман представив свій компілятор BDS-C для CP/M, також заснований на підмножині мови Сі. Достоїнствами цього компілятора були висока швидкість і можливість сумісної компоновки переміщуваних об'єктних модулів в завантажувальному модулі.

Незабаром після BDS-C були створені компілятори, призначені для CP/M і засновані на повній множині мови Сі. Це дало імпульс розвитку програмування на Сі для МІКРОЕОМ. У 1981 р., у зв'язку із створенням IBM PC, в світі мікроЕОМ був зроблений значний стрибок вперед.

Після появи IBM PC стали з'являтися і компілятори Сі для цієї ПЕОМ. Деякі компілятори були одержані шляхом перетворення відповідних компіляторів для процесора 8080, інші були розроблені спеціально для IBM PC. В наш час на ринку є більше двох десятків компіляторів мови Сі для IBM PC.

У 1983 р. Американський Інститут Стандартів (ANSI) сформував Технічний Комітет X3J11, статут якого передбачає створення стандарту мови Сі. Стандартизація поширюватиметься не тільки на мову, але і на програмне середовище компілятора, а також на бібліотеку стандартних функцій. У

роботі комітету беруть участь представники основних фірм – постачальників компіляторів Сі, у тому числі і для IBM PC, а також багато інших світил зі світу програмування на мові Сі. Зусилля комітету X3J11 привернули увагу засобів масової інформації. Пропонований стандарт був опублікований для того, щоб всі зацікавлені сторони могли ознайомитися з ним і внести свої пропозиції. Оскільки більшість постачальників компіляторів для IBM PC бере участь в роботі комітету X3J11, то нові версії компіляторів, що розробляються ними, будуть в рамках цього стандарту, (компілятор Borland C++ 3.1 для IBM PC відповідає більшості вимог стандарту на мову і бібліотеку).

1.2 Особливості мови Сі

Сі є мовою функцій, типів даних, операторів присвоєння і керування послідовністю обчислень. Програмуючи на Сі, ви здійснюєте звернення до функцій, і більшість функцій повертає деякі значення. Значення, що повертається функцією, хай то значення змінної або константа, може використовуватися в операторі присвоювання, який змінює значення іншої змінної. Доповнений операторами керування послідовністю обчислень (while, for, do, switch), Сі перетворюється на мову високого рівня, яка сприяє гарному стилю програмування.

Сі має невеликий набір типів даних: цілі числа, числа з плаваючою комою, бітові поля і переліковний тип. У мові Сі ви можете описати змінну типу покажчик, який пов'язується з об'єктом, що належить до будь-якого типу даних. Адресна арифметика мови Сі є чутливою до типу даних того об'єкту, з яким пов'язаний використовуваний покажчик. Дозволені також покажчики до функцій. Ви можете розширити список типів даних шляхом створення структур з ієрархічною залежністю вхідних в неї типів даних. Кожен тип даних може належати або до основного типу, або до раніше описаного структурного типу. Об'єднання нагадують структури, але визначають різні види ієрархічних залежностей, в яких дані різних типів розташовуються в пам'яті.

Допускається опис масивів даних різних типів, включаючи структури і об'єднання. Масиви можуть бути багатовимірними.

Функції Сі є рекурсивними за замовчуванням. Ви можете, правда, створити функцію, яка не буде рекурсивною, але сама мова за своєю природою прагне підтримувати рекурсивність і вимагає мінімальних зусиль при програмуванні рекурсій.

Програма функції на мові Сі розбивається на блоки, в кожному з яких можуть бути визначені свої власні локальні змінні. Блоки можуть вибиратися для виконання за результатом виконання оператора керування послідовністю обчислень. Блоки можуть бути вкладеними один в одного.

Змінні і функції можуть бути глобальними для програми, глобальними для початкового модуля або локальними для блоку, в якому вони описані. Локальні змінні можуть бути описані таким чином, що вони зберігатимуть свої значення при всіх звертаннях усередині даного блоку (статичні змінні) або ж сприйматимуться як нові об'єкти при кожному звертанні (автоматичні змінні).

Сі дозволяє створювати програму у вигляді декількох початкових модулів, які транслюватимуться незалежно. Переміщувані об'єктні модулі, відповідні початковим модулям, компонуються в єдиний завантажувальний модуль. Ця особливість дозволяє компілятору підтримувати об'єктні бібліотеки багаторазово використовуваних функцій і створювати великі програми з безлічі невеликих початкових модулів.

У мові Сі немає операторів введення/виведення, все введення/виведення виконується за допомогою функцій. Унаслідок цієї особливості мови Сі розроблена стандартна бібліотека функцій. Існування цього стандарту і складає головну привабливість мови Сі, бо робить програми на Сі переносними.

1.3 Переваги мови Сі

Переносимість програм, написаних на Сі, є найбільш розрекламованою перевагою цієї мови. Якщо ви пишете програму на Сі і уникаєте при цьому використання розширень бібліотеки, залежних від конкретного компілятора, або машинно-залежних операцій, то ви одержуєте непогані шанси (значно більші, ніж при будь-якій іншій мові) на успішне перенесення вашої програми в інше програмно-апаратне середовище, включаючи зміну компілятора, операційної системи і ЕОМ.

Розширюваність мов програмування означає існування потенційної можливості внести додатки в мову. Сі розрахований на розширення за власним задумом, оскільки містить дуже невелике число операторів. Слід пам'ятати, що сама мова дозволяє набагато більше, ніж змінювати значення змінних і керувати послідовністю виконання програми. Найважливіше в програмах на Сі укладене у функціях, а мова сам по собі не має інших внутрішніх функції, окрім основної функції (функції main). Перша група розширень мови Сі розміщується в стандартній бібліотеці, інші нестандартні розширення підтримує конкретний компілятор і, нарешті, третя група розширень міститься в додаткових бібліотеках функцій Сі. Остання група розширень розробляється самим програмістом, який створює програми для багаторазового використання.

Окрім своєї функціональної розширюваності, Сі дозволяє розширювати стандартний набір типів даних шляхом визначення структур, об'єднань і використання операторів typedef.

Програмістам особливо подобається стисліcть виразів, якими в Сі кодуються алгоритми. Більшість операторів, чи то оператори присвоєння, чи умовні оператори, звернення до функцій або вирази, окрім операторів керування послідовністю виконання програми, повертають певні значення. Використання цієї особливості мови дозволяє подавати вирази в короткій формі.

Сі забезпечує формування ефективного машинного коду програми, що досягається прив'язкою мов програмування до структури пам'яті і регістрової архітектури ЕОМ, для яких вони створюються. Сі часто характеризується як переносна мова асемблера високого рівня. Сама природа мови дозволяє компілятору генерувати ефективний оптимізований машинний код.

Не повинно залишатися сумнівів з приводу переваг мови Сі в порівнянні з іншими мовами програмування. Програмісти люблять Сі з причин, вказаних вище. Компанії з розробки програмного забезпечення люблять її за те, що вона дозволяє писати програми, незалежні від конкретної апаратури і операційної системи. Сучасні менеджери, багато з яких у минулому програмісти, враховують обидві ці переваги. Сі є мовою, якою написано більшість найпопулярніших в світі пакетів програм.

1.4 Робота в Borland C++ 3.1

Інтегроване середовище Borland C++ 3.1 є текстовим редактором з вбудованим компілятором, компонувальником і іншими засобами налагодження програми. У верхній частині екрана розташовується меню. У меню File є такі опції:

- New – створення нового файлу;
- Load – завантаження раніше збереженого файлу;
- Save – збереження поточного файлу;
- Save as – зберегти під новим ім'ям;
- Quit – вихід.

Тут і далі перераховуються не всі опції, а тільки ті, які бувають особливо корисними програмістам-початківцям. При виборі пункту New, створюється нове вікно, показане на рисунку 1. Біля нього знаходяться певні атрибути. А саме: ім'я файлу, оброблюваного у вікні (для нового файлу буде написано NONAME00.CPP), порядковий номер вікна (вікно можна вибрати, натиснувши клавішу Alt + цей номер), місцерозташування курсора (номер рядка і стовпця). Також є вертикальна і горизонтальна смуги прокрутки. У лівому верхньому кутку – маркер закриття вікна (вікно також само можна закрити, натиснувши Alt + F3). Одночасно на екрані можуть знаходитися декілька вікон, але тільки одне з них може бути активним. Воно обрамляється подвійною рамочкою, а неактивні вікна – одинарною. Перемикання між вікнами відбувається при натисненні F6. За допомогою маніпулятора "миша"

можна змінювати розміри вікна (узявшися за правий нижній куток) і переміщати вікно по екрану (узявшися за подвійну рамочку), розвертати на весь екран (правий верхній маркер або клавіша F5) і відновлювати до початкового розміру повторним натисненням на маркер.

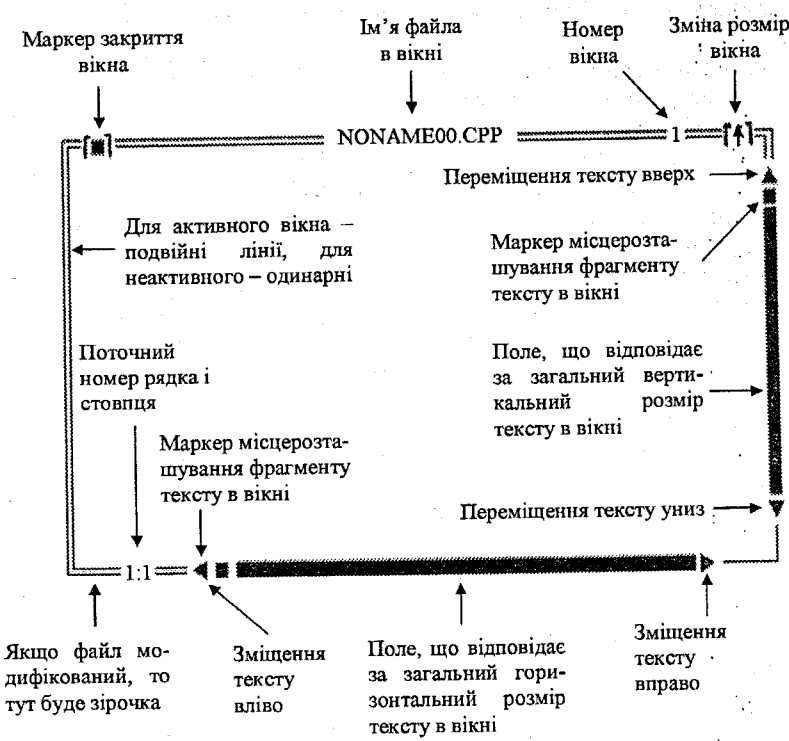


Рисунок 1 – Структура екрана інтегрованого середовища Borland C++ 3.1

Для запуску компілятора, компонувальника і (якщо вдало пройшли попередні дії) самої програми достатньо натиснути Ctrl + F9. З'явиться вікно, що відображає процес компіляції. Якщо помилок немає, то зразу ж буде запущена програма на виконання. Якщо є помилки, то в нижній частині екрану з'явиться ще одне вікно з переліком всіх помилок, що зустрілися, і вказанням місцезнаходження помилки.

Після виконання програми ми повертаємося назад, в інтегроване середовище. Для переглядання результату роботи програми потрібно натиснути Alt + F5.

Завантажити новий файл можна натисненням клавіші F3. Для збереження файлу (рекомендується це робити періодично) використовується клавіша F2. При першому збереженні Borland C++ 3.1 попросить вас вказати ім'я файлу і місце, куди ви хочете його зберегти.

Вбудований текстовий редактор дозволяє працювати з блоками інформації. Щоб задати блок, потрібно, утримуючи клавішу Shift, за допомогою клавіш керування курсором виділити потрібний текст. Далі з блоком можна робити таке (див. також меню Edit):

- Копіювати в буфер обміну (Ctrl + Ins);
- Вставляти з буфера (Shift + Ins);
- Видаляти з тексту програми (Ctrl + Del);
- Записати на диск у вигляді окремого файлу (Ctrl + K + W);
- Завантажити з диска (Ctrl + K + R);
- Вивести блок на принтер (Ctrl + K + P).

Щоб видалити рядок повністю необхідно натиснути Ctrl+Y. При неправильних діях (випадково щось було видалене) можна відмінити останні зміни за допомогою комбінації Alt + Backspace. Для повторення відмінених дій: Shift + Alt + Backspace.

1.5 Компіляція, компонування і виконання програми

Припустімо, що треба розробити програму з ім'ям myprog. Все що необхідно для цього можна подати у вигляді такої схеми:

myprog.cpp → myprog.obj →myprog.exe

Спочатку потрібно створити файл для початкового тексту програми на Сі myprog.cpp, потім у вікні редактора набрати текст відповідної програми і записати його у файл. Далі виконується компіляція програми і утворюється об'єктний файл (myprog.obj). Компілятор перекладає код програми з мови Сі на зрозумілу для машини мову двійкових кодів (команди для процесора). Оскільки в програмі ми використовуємо вже готові функції, то їх об'єктні файли також потрібно включити в програму. Цим займається компонувальник. На останній стадії після підключення компонувальника створюється виконуваний файл (myprog.exe). Його вже можна завантажувати і запускати на виконання на персональному комп'ютері. Більш детальна робота в середовищі буде розглянута в розділі 12.

2 Алгоритми

Алгоритм – це послідовність команд, що ведуть до якої-небудь мети. Можна сказати, що це чітко визначена процедура, що гарантує отримання результату за кінцеве число кроків, містить в собі дії, в результаті виконання послідовності яких відбувається перехід від початкових даних до шуканого

результату. Вказана послідовність дій називається **алгоритмічним процесом**, а кожна окрема дія – його кроком. Приклад: площа прямокутника S = a ×b.
Алгоритми умовно можна розділити на:

- обчислювальні,
- діалогові,
- графічні,
- обробки даних, керування об'єктами та процесами і ін.

Властивості алгоритмів – однозначність (і визначеність), результативність (і здійснимість), правильність (і зрозумілість), масовість або універсальність (тобто застосовність для цілого класу задач, до різних наборів початкових даних). Способи запису алгоритмів:

- У вигляді блок-схем,
- У вигляді програм,
- У вигляді текстових описів (рецепти, наприклад, рецепти приготування їжі, ліків і ін.).

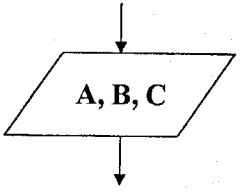
Ми з вами розглядатимемо алгоритми у вигляді блок-схем і у вигляді програм. Блок-схема алгоритму – це графічний опис алгоритму як послідовності дій.

Блок-схема складається з таких елементів:

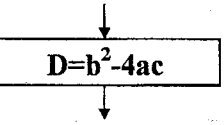
Для того, що б людині, яка розбиратиметься в вашому алгоритмі було зрозуміло звідки він починається, в графічному способі запису алгоритму існує спеціальний елемент блок-схеми, який символізує початок алгоритму – це овал, усередині якого написано "Початок". Елемент "початок" має один вихід (тобто стрілку, що виходить) і жодного входу (тобто вхідної стрілки), що говорить про те, що він розпочинає блок-схему, тобто перед ним нічого немає, а після нього відповідно до напряму стрілки йде алгоритм. Цей блок використовується в кожному алгоритмі.

Початок алгоритму

Далі, будь-який алгоритм вимагає вхідні дані, за допомогою яких і виконуються розрахунки. Для введення значень з клавіатури передбачений елемент блок-схеми у вигляді паралелепіпеда, з переліком змінних, що вводяться, усередені нього. Цей блок має вхідну і вихідну стрілки.



Блок введення інформації

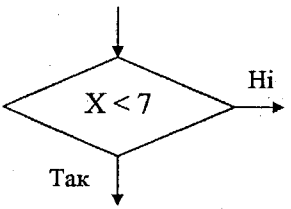


Блок обробки інформації

У прямокутнику звичайно записують арифметичні вирази, формули, які-небудь дії, які потрібно виконати для отримання необхідного результату. Лінійні алгоритми в основному будуються з таких елементів блок-схем.

Усередині блоку записуються формули, знаки операцій і функцій.

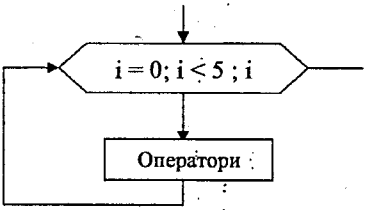
Блок умовного оператора. Зображається за допомогою ромба. Особливістю цього елементу є те, що до нього підходить одна стрілка, а виходять дві. Досягши блоку розгалуження, алгоритм далі може йти по одній з двох гілок, залежно від того, правильно записана умова усередині ромба на даний момент чи ні. Якщо правильно, то рухаємося за стрілкою з написом "Так", якщо ні, то за стрілкою з написом "Ні". Алгоритми, що містять блоки умови, називаються алгоритмами **з розгалуженнями**.



Блок умови

Усередині блоку записуються умови напряму дії алгоритму.

Іноді буває необхідно виконати одну і ту ж дію кілька разів підряд. Для того, щоб не записувати одні і ті ж елементи блок-схем кілька разів, використовують блок циклу. Він зображається подовгуватим шестигранником. За стрілкою зверху ми потрапляємо всередину циклу. Спочатку виконується первинна установка значення лічильника (змінна i = 0). Далі записується умова, при виконанні якої ми йдемо за стрілкою вниз і виконуємо дії, записані в операторі (іноді їх декілька). Потім ми підіймаємося за стрілкою вгору, змінюємо значення параметра і знов перевіряємо умову (i < 5). Якщо умова перестає працювати, то виходимо з циклу вправо. Детальніше алгоритми з використанням циклів ми розберемо трохи пізніше.



Блок циклу з параметром

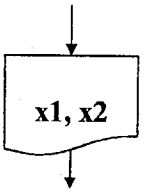
Блок циклу з параметром. Усередині блоку записуються через крапку з комою: початкове значення параметра лічильника;

умова виконання циклу;

зміна параметра на кожному кроці циклу.

Метою будь-якого алгоритму є отримання результату. І цей результат необхідно вивести на екран. Для виведення значень змінних

використовується блок-схема, усередині якої записуються значення, що виводяться.



Блок виведення інформації

В кінці алгоритму звичайно ставиться елемент блок-схеми "Кінець". До нього підходить лише одна стрілка і не виходить жодної.
Кінець алгоритму позначається таким блоком.

Блок закінчення алгоритму

Як ви вже здогадалися, алгоритми бувають декількох видів. А саме:

- лінійними,
- з розгалуженнями,
- циклічними, тобто що містять цикли,
- з підпрограмами,
- змішані (тобто що містять і цикли, і підпрограми, і розгалуження).

Лінійним називається алгоритм, в якому окремі оператори виконуються в природному порядку незалежно від значень початкових даних і проміжних результатів.

З розгалуженнями називається алгоритм, в якому залежно від результату перевірки деякої умови виконується один з декількох передбачених операторів.

Циклічним називається алгоритм, в якому зустрічаються цикли, що дозволяють кілька разів повторити одну і ту саму групу команд.

З підпрограмами називається алгоритм, в якому є виклик підпрограм (процедур, функцій).

В даному посібнику не будуть розглядатися правила складання алгоритмів. В разі необхідності читачу слід звернутися до навчального посібника автора Круподьорової Л.М. „Алгоритмізація і основи програмування", де докладно розглянуті правила складання алгоритмів і подано багато прикладів.

2.1 Створення першої програми

Давайте напишемо простеньку програму на мові Сі, яка виводитиме на екран вітання "Hello world !". Для цього запустіть інтегроване середовище Borland C++ 3.1, створіть новий файл, і наберіть такий текст:

```
/* Моя перша програма */
# include <stdio.h>
void  main () {
    printf("Hello world !"); }
```

Відкомпілюйте її і запустіть. На екрані з'явиться напис "Hello world !". Давайте спробуємо розібратися, що ж ми написали, і як це працює. Перший рядок взято в дужки вигляду /* */. Це так званий коментар. Коментарі ніяк не впливають на роботу програми і використовуються тільки для пояснення початкового коду. В даному випадку, в коментарі записана назва програми. Існує ще одна форма запису коментарів. Якщо в тексті програми зустрічаються підряд два символи //, то все, що розташовується до кінця рядка,вважається коментарем.

Наступний рядок буде виділений зеленим кольором. Слово include – означає "підключити". Далі, в програмі, ми використовуватимемо функцію printf(), яка описана в стандартній бібліотеці stdio.h. Для того, щоб ми могли користуватися цією функцією, необхідно підключити відповідну бібліотеку. Якщо потрібно підключити інші бібліотеки, наприклад, математичну, в якій містяться основні тригонометричні функції, операції піднесення до степеня, знаходження арифметичного квадратного кореня і т.п., то нам необхідно написати на наступному рядку: #include <math.h> , де math.h – ім'я файлу, в якому описана стандартна математична бібліотека.

Стандартних бібліотек в Ci/Ci++ дуже багато. Ми будемо знайомитись з ними при вивченні нових функцій. Трикутні дужки означають, що файл знаходиться в каталозі INCLUDE, де розміщені описи всіх стандартних бібліотек. Алфавітний перелік найбільш необхідних функцій подано в кінці посібника. Далі йде така конструкція:

```
void main 0
{
}
```

Це функція main0, яка присутня в будь-якій програмі на Ci/Ci++, і саме з неї починається виконання програми. Тіло функції main0 обмежено фігурними дужками. Усередині них записуються оператори програми. У нашому випадку записана функція виведення на екран – функція printf0. Відмінною рисою всіх функцій є наявність круглих дужок. Усередині них записуються передані функції параметри. Тут ми передаємо рядок тексту.
Рядок повинен бути взятий в подвійні лапки і може містити усередині себе будь-які символи (у тому числі і букви російського алфавіту). Зверніть увагу, що після функції ставиться крапка з комою.

Спробуйте замінити рядок тексту на інший. Наприклад, зробіть так, щоб програма вивела вітання вигляду: "Привіт, Андрію !".

2.2 Як зробити красиво

У файлі conio.h є досить багато корисних функцій, що дозволяють красиво оформити програму і надати їй дружній інтерфейс. Наприклад, для того, щоб

очистити екран достатньо записати clrscr(). Це скорочення від двох англійських слів clear – чистота і screen – екран.

Можна задати колір фону і колір символів, що виводяться:

textcolor(color) – встановлює колір тексту. Значення color може бути узяте будь-яке з діапазону 0..15 (див. таблицю кольорів далі).

textbackground(color) – задає колір фону символів, що виводяться. Якщо цю функцію викликати перед очищенням екрана, то весь екран буде кольору color.

gotoxy(x, y) – переміщає курсор на рядок y в стовпець x екрана. Сам екран в текстовому режимі має 25 рядків по 80 знакомісць на кожному рядку. Далі текст виводитиметься, починаючи з цього місця.

window(x1, y1, x2, y2) – створює вікно на екрані з верхнім лівим кутком в позиції (x1, y1) і правим нижнім в позиції (x2, y2). Текст виводитиметься усередині даного вікна.

3 Мова C++. Практичне знайомство

У попередній першій програмі використовуються то українські, то англійські букви, спеціальні символи. А якими взагалі символами можна користуватися при написанні програм? Тіло програми, написаної на мові Сі, складається з інструкцій. Кожна інструкція викликає певні дії на відповідному кроці виконання програми. При написанні інструкцій застосовуються певні символи, що становлять алфавіт мови.

Алфавіт мови Сі включає:

- букви латинського алфавіту: великі від A до Z і малі від a до z.
- арабські цифри: 0, 1, 2, 3, 4, 5, 6, 7, 8, 9.
- спеціальні символи:

Символ	Назва	Символ	Назва
+	Плюс	“	Лапки
-	Мінус	’	Апостроф
*	Зірочка	( )	Круглі дужки
/	Прямий слеш	[ ]	Квадратні дужки
=	Дорівнює	{ }	Фігурні дужки
<	Менше	%	Процент
>	Більше	&	Амперсент
!	Знак оклику	#	Решітка
?	Знак питання	~	Тильда
.	Крапка	|	Вертикальна риска
,	Кома	\	Зворотний слеш
:	Двокрапка	_	Підкреслення
;	Крапка з комою		Пропуск
^	Кришечка		

Використання українських букв (і інших специфічних символів) можливе тільки в рядкових виразах. Кількість пропусків і натиснень клавіші <Enter>, використовуваних для візуального форматування тексту, не має для компілятора значення. При набиранні початкового коду рядки не нумеруються.

Крапка з комою служить ознакою кінця оператора, і найчастіше перші помилки програміста пов'язані саме з цією обставиною. Компілятор розрізняє малі і великі букви, і в деяких випадках заміна малої букви на велику або навпаки може викликати помилку.

У базову множину мови входить набір операцій, операторів, функцій і визначених програмістом змінних, причому їх типи наперед визначені і використовуються для зберігання і обробки змінних певного типу, про це буде розказано пізніше. Також програміст може створювати свої власні типи даних, базуючись на основних. Як ім'я або ідентифікатор змінної або функції може служити набір букв латинського алфавіту, цифр і символу підкреслення, при цьому починатися це ім'я повинне обов'язково з букви і не повинне збігатися з будь-яким ключовим словом. Якщо з таким ім'ям в системі вже використовується змінна або функція, то залежно від ситуації, про що буде розказано пізніше, може виникнути або помилка збігу імені ще на стадії компіляції, або перевизначення змінної чи функції в межах частини програми. Довжина імені не фіксується (в розумних межах), але різні компілятори розрізняють імена за різною кількістю початкових символів, що теж може служити джерелом помилки. Як ім'я заборонено використовувати такі ключові слова:

asm, auto, break, case, char, const, continue, default, do, double, else, enum, extern, float, for, goto, if, int, long, register, return, short, signed, sizeof, static, struct, switch, typedef, union, unsigned, void, volatile, while, _ss, interrupt, _cs, cdecl, near, _ds, far, pascal, _es, huge, _export.

Змінні можуть використовуватися як аргументи або операнди для математичних або логічних операцій, таких як порівняння, присвоєння, додавання, побітові операції зсуву і додавання або множення, за допомогою яких можна виконувати перетворення даних і складати логічні вирази, істинність або помилковість яких може використовуватися як критерій виконання тих або інших умовних операторів, у тому числі і умовних циклів. Залежно від кількості операндів операції діляться на унарні, бінарні.

Змінна є будь-якими даними, що зберігаються в певних елементах пам'яті, і значення якої, на відмінність від константи, може бути змінене в ході виконання програми. Вид даних (цілі, дробові числа, рядки і так далі) і спосіб

їх подання і зберігання визначається типом змінної, який указується при оголошенні або визначенні змінної. Як вже наголошувалося вище, ідентифікатори змінної не можуть збігатися з ключовими словами, тому є невідомими компілятору. У зв'язку з цим до першого використання змінної її обов'язково потрібно оголосити або визначити, тобто надати їй початкове значення. Якщо змінна просто оголошується, то її початкове значення може бути або нулем, або будь-яким довільним, відповідним тим байтам, які були записані в елементах пам'яті, відведених під змінну. Це залежить від того, глобальною чи локальною була змінна, і від компілятора. Щоб уникнути невизначеності ми або визначатимемо змінні, або надаватимемо їм початкові значення до першого використання в програмі.

Загальна структура оголошення змінної може бути записана так:

тип_даних ідентифікатор;

Якщо потрібно оголосити декілька змінних одного типу, то їх ідентифікатори можна перераховувати через кому:

тип_даних ідентифікатор1, ідентифікатор2;

Загальна структура визначення змінної відрізняється тільки тим, що ідентифікатору зразу ж присвоюється початкове значення, причому можна одночасно визначати і оголошувати змінні однакового типу:

тип_даних ідентифікатор=значення;

або

тип_даних ідентифікатор1=значення1, ідентифікатор2=значення2;

або

тип_даних ідентифікатор1, ідентифікатор2=значення;

Як базові типи даних визначені цілі числа і дробові числа, а також посилальний тип, за допомогою якого, наприклад, можна зберігати рядки.

Для зберігання цілих чисел застосовуються типи char і int, причому перший може зберігати числа від -128 до 127, тобто об'єм цього типу складає один байт. Звичайно тип char використовують для зберігання символів в кодуванні ASCII, тому він так і називається. Довжина типу int дорівнює машинному слову і для 16-ти розрядних систем складає 2 байти, а для тридцятидворозрядних – 4 байти. Ми розглядатимемо 16-ти розрядні системи, наприклад, Borland C++ 3.1, тому з його допомогою можна зобразити числа від -32768 до 32767. Це так звані знакові типи, в них для зберігання числа використовуються не всі біти, наприклад, 7 і 15, а останній, старший біт дорівнює 0 для додатних чисел, і 1 для від'ємних. Кожне число за модулем розкладається на степені двійки, наприклад, $25_d = 1\times2^4 + 1\times2^3 + 0\times2^2 + 0\times2^1 + 1\times2^0 = 11001_b$.

Для зберігання від'ємних чисел використовується додаткова двійкова арифметика, тобто береться додатне число, всі його біти інвертуються, і до

нього додається 1, тоді при складанні однакових за модулем, але різних за знаком чисел з використанням правил двійкової арифметики одержимо 0. Наприклад, для типу char 127 записується в двійковій системі як 01111111, а -127 для типу integer як 10000001.

Для цілих чисел існує модифікатор unsigned, який дозволяє використовувати базові типи для зберігання беззнакових чисел, тобто від 0 до 255 і від 0 до 65535. Крім того, визначений модифікатор signed, але його використання не має сенсу, оскільки типи int і char є знаковими за замовчуванням.

Для зберігання дробових (дійсних) чисел використовуються типи float і double. Дійсні числа можуть бути подані або в звичному записі, наприклад, 3956.768, або в науковому: 0.3956768e+4, тобто 0.3956768×10^4. У другому випадку число подається у вигляді мантиси, тобто основи, і експоненти, в даному випадку – ступені десятки, на яку домножають мантису.

Для зберігання мантиси використовують такий факт: нескінченна геометрична прогресія сходиться до певного числа, наприклад, якщо складатимемо 1/2 +1/4+1/8+... або 1/21+1/22+1/23+..., то в кінці-кінців одержимо в сумі 1, причому якщо ми складатимемо не до безкінечності, а до якогось певного ступеня n (наприклад, 10), то ми одержимо число, відмінне від 1 не більше, ніж на $1/2^{n-1}$. Якщо тепер ми викидатимемо якісь ступені двійки, підставляючи у відповідні біти двійкового подання 0, то одержимо число, відмінне від 1. Таким чином, за допомогою n біт ми можемо подати будь-яку дробову частину дійсного числа з точністю $1/2^{n-1}$, розкладаючи його за від'ємними степенями двійки 2^{-1}, 2^{-2} і так далі. Типи float і double відрізняються кількістю бітів, що відводяться для зберігання мантиси, тобто точністю, і на зберігання експоненти, тобто діапазоном, від найменших до найбільших чисел.

До перерахованих типів можуть застосовуватися модифікатори short і long, що зменшують і збільшують кількість байтів, які відводяться для зберігання типу. Причому модифікатор short визначений тільки для типу int і потрібен в 32-розрядних системах для сумісності, в 16-ти розрядній системі він позбавлений значення, хоча і визначений, ми його не братимемо до уваги.

Відзначимо, що слово int в довгих поєднаннях з модифікаторами може опускатися, наприклад, unsigned і unsigned int означають одне і те ж.

Крім того, якщо потрібно запобігти зміні якої-небудь величини, її потрібно оголосити як константу з використанням модифікатора const. При спробі модифікувати значення константи в програмі виникне помилка компіляції.

Перерахуємо описані типи в таблиці 1.

Таблиця 1 – Основні типи даних в мові Сі

Тип	Довжина в байтах	Діапазон
char	1	-128 до 127
unsigned char	1	0 до 255
int	2	-32768 до 32767
unsigned int	2	0 до 65535
long int	4	-2147483648 до 2147483647
unsigned long int	4	0 до 4294967295
float	4	3.4e-38 до 3.4e+38
double	8	1.7e-308 до 1.7e+308
long double	10	3.4e-4932 до 1.1e+4932

Як вже згадувалося, змінні можуть оголошуватися або визначатися на самому початку програми, такі змінні називаються **глобальними**, оскільки вони можуть бути використані в будь-якій частині програми, наприклад, в будь-якій функції. Також змінні можуть оголошуватися або визначатися усередині якого-небудь блоку, взятого в операторні дужки {}, наприклад, усередині якої-небудь функції або складового оператора. В цьому випадку змінна існує тільки усередині цього блоку, а при спробі до неї звернутися поза цим блоком виникає помилка компіляції, поза цим блоком дана змінна є невизначеною. Такі змінні називаються **локальними**. Тобто для кожної змінної існує область видимості, в якій вона визначена. В межах області видимості ідентифікатори змінних не повинні збігатися, в той же час в різних областях видимості, наприклад, в різних блоках можуть існувати змінні з однаковими ідентифікаторами. Проте ідентифікатори глобальної і локальної змінної можуть збігатися. При цьому поза областю видимості локальної змінної завжди відбувається звернення до глобальної, а в межах області видимості – до локальної. Існує оператор розширення області видимості, який дозволяє звернутися до однойменної глобальної змінної в межах області видимості локальної, він позначається як дві двокрапки ::, тобто якщо існує глобальна змінна s і локальна – s, то в межах області видимості локальної s означає локальну, а ::s – глобальну.

Окрім області видимості для кожної змінної існує ще і час життя, тобто той час, який вона знаходиться в пам'яті. Звичайно глобальні змінні знаходяться в пам'яті весь час виконання програми, а локальні – тільки під час виконання блоку, який складає їх область видимості, причому при кожному звертанні до блоку ці змінні створюються наново і наново визначається їх значення. Це можна змінити, якщо при оголошенні або визначенні задати специфікатор зберігання static. Наприклад, якщо оголосити статичну локальну змінну

усередині функції, то від виклику до виклику дана змінна зберігатиме своє значення. Вся решта змінних створюється із специфікатором за замовчуванням auto, який завжди опускається, що означає, що змінні живуть тільки під час виконання даного блоку. Також можна використовувати специфікатор extern, щоб показати, що оголошувана глобальна змінна також оголошується в іншому модулі програми, і ці змінні за суттю одне і те саме. Також використовується специфікатор register, якщо необхідно для швидкодії зберігати дану змінну в регістрах процесора.

Продемонструємо все сказане щодо змінних на такому прикладі:

```
#include <conio.h>
#include <stdio.h>
  int s=2; //визначаємо глобальну змінну
/* визначаємо функцію f0, поки не замислюємося,
  що це таке – просто окремий блок */
  void f () {
       int z=5; //визначимо локальну змінну
  /* визначимо статичну локальну змінну, ім'я якої збігається
     з ім'ям глобальної */
    static int s=1;
    printf("%i %i %i", s, ::s, z); //виведемо всі три значення
    //змінимо значення локальних змінних на 1.
    s++;
    z++; };
    /* визначимо функцію main() */
   void main ()
 {
    clrscr () ;
    f0;         //викличемо функцію f0
    s=7;        //змінимо значення глобальної змінної
    f0;         //знову викличемо функцію f0
    getch () ;
 }
```

В результаті на екрані з'являться такі числа:
1 (значення локальної змінної s),
2 (значення глобальної змінної s),
5 (значення локальної змінної z),
2 (змінене значення локальної змінної s після операції s++, її значення зберігається між викликами функції),
7 (нове значення глобальної змінної),

5 (значення локальної змінної z, яке оновлюється при кожному виклику функції).

У прикладі використовувалася вже знайома нам функція printf(). Але тут ми виводимо значення змінних, а не рядок тексту. Для цього достатньо в подвійних лапках написати %i – для виведення цілого числа або %f – для виведення дробового числа, і потім через кому записати відповідну змінну. Детальніше з роботою функції введення/виведення ми познайомимося в наступному розділі.

Отже, ми розібралися що таке змінні, тепер подивимося, що з ними можна робити. Наприклад, з них можна складати вирази, використовуючи математичні операції: додавання, віднімання, множення, ділення. Ці операції застосовні до цілих і дійсних типів даних. У них використовуються два значення або операнд, наприклад, два доданки і так далі, і якщо типи операндів не збігаються, наприклад, цілий і дійсний, то результат перетвориться автоматично до довшого і загального типу, наприклад, до дійсного. Якщо типи обох операндів збігаються, то тип результату збігається з ними.

Виняток становить операція взяття залишку від ділення по модулю "%". Вона застосовується тільки до цілих чисел і видає залишок від ділення першого на друге. До арифметичних операцій також відносять інкремент ++ і декремент --, які застосовуються до одного операнда цілого типу. Ці операції збільшують і зменшують, відповідно, значення операнда на 1. Розрізняються префіксна форма (++змінна) і постфіксна (змінна ++). Для більшості випадків це не має значення, але іноді змінюється послідовність виконання. Префіксна завжди видає значення операнда вже змінене, а постфіксна спочатку видає значення операнда, а потім його змінює.

Важливою операцією є присвоєння значення змінній. Наприклад, якщо ми хочемо присвоїти змінній x значення змінної y, то пишемо

```
x = y;
```

а не навпаки. Можна присвоїти змінній результат виразів, математичних дій, наприклад:

```
x = 2 * y + 8;
```

Можна створити множинне присвоєння, наприклад:

```
x = y = 5;
```

тоді спочатку в y запишеться 5, а потім в x – значення y (y даний момент 5).

У мові Сі є побітові операції, які визначені тільки для цілих типів. Нехай є дві змінних типу unsigned char: p=3 (або 00000011 в двійковій) і q=5 (або 00000101 в двійковій). Тоді для них визначені операції побітового І &, побітового АБО |, побітового виключного АБО ^, побітового зрушення вправо >> і вліво <<(бінарні) і доповнення ~ (унарна).

A	B	A&b	a|b	a^b	a<<2	a>>1	~a
3	5	1	7	6	12	1	252
00000011	00000101	00000001	00000111	00000110	00001100	00000001	11111100

За допомогою цих операцій можна досить ефективно обробляти дані. Крім того, досить часто зустрічаються ситуації, коли в змінну потрібно записати її ж значення, але, наприклад, помножене на якесь число. Тоді потрібно написати конструкцію вигляду

```
x = x*3;
```

але для цього можна застосувати скорочений варіант

```
x *= 3;
```

який означає те ж саме. Такі скорочені записи, або присвоєння, пов'язане з операцією, існують для всіх арифметичних і побітових операцій. Всі операції мають різний пріоритет, тобто порядок виконання. Існування пріоритету треба завжди мати на увазі при складанні виразів, в крайньому випадку для підстраховування, – розставляти дужки ().

Як вже згадувалося, результат виразу залежить від операндів, а особливо від лівого операнда в операції присвоєння. Проте, іноді необхідно явним чином перетворити змінну одного типу до іншого типу. При цьому використовується така конструкція:

(необхідний_тип) змінна;

наприклад,

```
(int) x;
```

Перетворення коротших типів до довших завжди відбувається безболісно. Зворотне перетворення є коректним, якщо значення змінної не виходить за діапазон необхідного типу, інакше може сильно змінитися значення, наприклад, змінитися знак числа і навіть його модуль. Продемонструємо все сказане на прикладі такої програми:

```
#include <conio.h>
#include <stdio.h>
  const int s=2; //визначаємо глобальну константу
  //визначаємо глобальні змінні
   int a=3;
   char c=4;
   float z=3.14;
   void main ()
   {
    clrscr () ;
    //виведемо результат додавання цілого і дробового числа
```

```
    // - 6.14 – дробове число
    printf("%f ", a+z);
     //виведемо залишок від ділення 3 на 2 – це 1 – ціле число
     printf("%i ", a%s);
//виведемо результат перетворення дробового числа 3.14 до цілого 3
     printf ("%i ", (int)z);
//присвоїмо цілій змінній дробове число 3.14, одержимо ціле 3
     a=z;
     printf("%i ",a);
//присвоїмо а значення, що виходить за межі діапазону char 300
//перетворимо його до char, потім назад до int, і подивимося, що
//вийшло 44.
     a=300;
     printf("%i ", (int)(char) a);
     getch () ;
}
```

4 Як вводити і виводити інформацію

У будь-яких програмах функції введення і виведення виконують важливу роль. У мовах Сі і Сі++ їх дуже багато. Найпростіший спосіб введення - зчитування по одному символу з клавіатури за допомогою функції getchar(). Вона має такий опис:

```
int getchar();
```

Функція повертає значення цілого типу. Це число можна інтерпретувати як ASCII-код введеного символу. Всі символи (латинські, російські букви, цифри, спеціальні символи) пронумеровані числами від 0 до 255. Таблиця, в якій кожному символу поставлений у відповідності порядковий номер, називається таблицею ASCII-кодів. Для того, щоб взнати який символ ми ввели з клавіатури, нам необхідно ввести змінну цілого типу (або типу char) і записати в неї результат, що повертається функцією:

```
char ch;
ch = getchar ();
```

Щоб вивести символ на екран необхідно використовувати функцію putchar(). Наприклад:

putchar (ch); – виводить символ, записаний в змінну ch.

putchar (68); – виводить символ з ASCII-кодом 68. Це буква 'A'.

putchar (32 + 4) ; – виводить символ з ASCII-кодом 36. ЦЕ '$'.

Для того, щоб ми змогли використовувати функції getchar() і putchar(), необхідно підключити до нашої програми стандартну бібліотеку stdio.h.

Це робиться за допомогою такого рядка, який пишеться на самому початку програми: #include <stdio.h>

Помітимо, що для функції getchar() після вибору символу необхідно натиснути клавішу <Enter>. Іноді це створює певні незручності. Функції getch() і getche() усувають їх. Вони мають такий вигляд:

```
int getch();
int getche();
```

Обидві ці функції вводять символ зразу ж після натиснення відповідної клавіші на клавіатурі (тут не треба додатково натискати клавішу <Enter>). Відмінність між ними полягає у тому, що getche() відображає символ, що вводиться, на екрані дисплея, а getch() – ні. Описи цих функцій знаходяться у файлі conio.h.

4.1 Виведення формату даних

Функція printf забезпечує виведення формату даних. Її можна записати в такому вигляді:

printf("керуючий рядок", аргумент_1, аргумент_2,...аргумент_N

Функція описана в бібліотеці stdio.h. Керуючий рядок містить компоненти трьох типів: звичні символи, які просто копіюються на екран (як в розглянутій раніше програмі "Привіт, Андрію!"), специфікація перетворення, кожна з яких викликає виведення на екран значень чергових аргументів з подальшого списку і керуючі символьні константи.

Кожна специфікація перетворення починається із знаку %. Далі йде буква, що позначає тип значення, що виводиться. Наприклад:

%c – значення аргументу є символ.

%d або %i – значення аргументу є десяткове ціле число.

%e або %E – значення аргументу є дійсне (дробове) число в експоненційній формі вигляду 1.23e+2. Це значить 1.23×10^2 тобто число 123. Або 3.456e-2. Це число рівне $3.456\times10^{-2} = 0.0345$.

%f – значенням аргументу є дійсне десяткове число з плаваючою точкою.

Наприклад: 3.141593. За замовчуванням виводиться 6 знаків після коми (у комп'ютерах для розділення цілої частини від дробової використовується крапка). Якщо ми хочемо, щоб виводилося тільки два знаки після коми, то треба написати так: %.2f. Запис виду %4.2f означає, що для виведення числа буде зарезервовано 4 позиції, дві з яких призначені для дробової частини.

%g або %G – також виводить дійсне число, виключаючи незначущі нулі. Тобто якщо у нас число 5.2300 і ми виводимо його з префіксом %g, то буде надруковано 5.23.

%s – значенням аргументу є рядок символів. Як працювати з рядками ми познайомимося трохи пізніше.

%x або %X – значенням аргументу є шістнадцятіркове ціле число з цифрами 0..9, A, B, C, D, E, F. Наприклад, десяткове число 123 в шістнадцятірковому поданні має вигляд 7B.

%p – значенням аргументу є покажчик. Покажчики детально будуть розглянуті в розділі 7.2.

Якщо після знаку % записаний символ, який не є символом перетворення, то він виводиться на екран. Таким чином, рядок %% приводить до виведення на екран знаку %.

Функція printf використовує керуючий рядок, щоб визначити скільки всього аргументів і які їх типи. Аргументами можуть бути змінні, константи, вирази, виклики функцій. Головне, щоб їх значення відповідали заданій специфікації.

Розглянемо керуючі символи, які також можуть зустрічатися в керуючому рядку. Всі вони починаються з символу \:

\a – подача звукового сигналу;
\b – переведення курсора на одну позицію вліво;
\n – перехід на новий рядок;
\t – горизонтальна табуляція;
\\ – виведення символу \;
\' – виведення символу ' ;
\" – виведення символу " ;
\? – виведення символу ? .

Для повного розуміння роботи функції розглянемо приклад:

```
#include <stdio.h>
 main ()
 {
   int x, y, z;
   char ch;
   float f1, f2;
   double d;
   textcolor(14);
   textbackground(0);
   clrscr () ;
   x = 10;
   y = 20;
   z = x + y;
   ch = '*';
   f1 = 3.14;
   f2 = f1 * 2.5;
   d = 0.5;
```

```
    printf ("Виводимо тільки текст \n" );
    printf("Значення змінної x = %i \n", x );
    printf("Змінні y і z  дорівнюють %3i  і  %3i, відповідно,",  y, z);
    printf("А зараз надрукуємо три зірочки %c %c %c ", ch, ch, ch );
    printf("\t Зробили відступ \n пропустили рядок \n і ще один " );
    printf("%f %f %.4f ", fl, f2, d );
    getch () ;
}
```

В кінці програми написаний оператор getch(), який дозволяє нам зробити затримку перед остаточним завершенням програми. Він чекає, поки користувач натисне яку-небудь клавішу, і лише після цього – вихід з програми. Щоб кожного разу не натискати комбінацію клавіш <Alt> + <F5> для перегляднання результатів, рекомендується в кінці програми завжди використовувати getch(). Як приклад розглянемо таку задачу: для фарбування будинку необхідно купити фарби двох кольорів: червону (для даху) і жовту (для стін). Необхідно скласти програму, яка, знаючи розміри будинку, визначає, скільки знадобиться фарби, щоб пофарбувати лицьову сторону будинку. Вікна фарбувати не потрібно.

```
/* Фарбування будинку */
 #include <stdio.h>
 #include <conio.h>
void main ()
{
   int a, b, c, d, h;
   float S1, S2;        // S1 – площа даху,  S2 – площа стіни
   textcolor(10);
   textbackground(1);
   clrscr();            // Очищення екрану
   a = 10;              // Задаємо розміри будинку
   b = 5;
    c = 2;
   d = 2;
   h = 3;
   S1= 0.5 * a * h;
   S2= a * b – 3 * (c * d) ;
   printf("Потрібно купити %2.1f червоної фарби \n", S1);
   printf("і %2.1f жовтої фарби \n", S2);
   getch () ;
}
```

4.2 Введення формату даних

У написаній вище програмі є один недолік. Програма завжди розраховує площу для одного і того ж будинку. А що робити, якщо нам треба пофарбувати декілька різних будинків? Ми можемо виправляти кожного разу код програми, задаючи нові значення змінних a, b, c, d, h і наново її компілювати. Але це не наш метод! Та й не завжди під рукою є компілятор. Краще, якби програма кожного разу сама запитувала нас розміри будинку.

Для введення інформації в мові Сі передбачена функція введення формату даних. Це функція scanf, визначена в stdio.h. Вона має такий вигляд:

scanf ("керуючий рядок", аргумент_1, аргумент_2, ... аргумент_N) ;

Її опис багато в чому схожий з функцією printf. Керуючий рядок містить тільки специфікатори перетворення. Тобто тип і кількість значень, що вводяться з клавіатури. Ці значення записуються у відповідні аргументи. Як аргументи у функцію scanf подаються не самі змінні, а їх адреси. Докладніше роботу з пам'яттю ми розглянемо далі, а поки що скажемо, що значок '&' – амперсент, поставлений перед змінною, дає її адресу.

Перерахуємо лише деякі специфікатори перетворення для функції scanf:

%c – введення одного символу;

%d або %i – введення цілого десяткового числа типу int;

%D або %I – введення цілого десяткового числа типу long;

%f — введення дійсного числа;

%s – введення рядка символів.

Приклад введення інформації:

```
int x, y, z;
printf("Введіть ціле число: ");
scanf("%i", &x);
printf("'Введіть ще два цілі числа: ");
scanf("%i %i", &y, &z);
printf("Сума трьох чисел = %i", x + y + z);
char ch;
scanf("%c", &ch);
printf("Ви ввели символ %c", ch);
```

Тепер ви можете легко самі переписати програму із попередньої задачі так, щоб вона запитувала вхідні дані з клавіатури.

В іншому прикладі необхідно написати програму для касового апарата, яка обчислює вартість товару. Вхідні дані: ціна за 1 кг продукту і вага покупки.

```
/* Програма - касир */
#include <stdio.h>
#include <conio.h>
```

```
void main 0
{
    float price, mass;      // ціна і вага
    textcolor(14);
    textbackground(0);
    clrscr 0 ;
    window(20, 5, 60, 20);   // створюємо вікно посередині екрану
    printf("Введіть ціну за 1 кг :");
    scanf("%f", &price);
    printf("Введіть вагу:");
    scanf("%f", &mass);
    printf("З вас %.2 крб. \n Дякуємо за покупку!", price * mass);
     getch 0 ;
}
```

У файлі conio.h існують функції cprintf0 і cscanf0, аналогічні printf0 і scanf0. Функції cprintf0 і cscanf0 теж призначені для введення з клавіатури і виведення на екран, але "прив'язані" до конкретних фізичних пристроїв, в даному випадку – консолі, тобто клавіатури і монітора. Функції ж printf0 і scanf0 працюють із стандартним потоком виведення stdout і стандартним потоком введення stdin, які можуть бути і не пов'язані з консоллю, тобто програмно перевизначені для роботи, наприклад, з файлом або іншим пристроєм.

4.3 Введення і виведення інформації за допомогою потоків

У мові Сі++ є інший, можливо, зручніший спосіб введення/виведення інформації. Це робота з потоками. Зручність полягає у тому, що ми не повинні указувати тип значень, що вводяться і виводяться. Програма сама визначає його, виходячи з оголошення змінних. Для роботи з потоками необхідно підключити файл iostream.h. Тоді ми зможемо працювати з двома потоками: cin і cout – стандартні вхідний і вихідний потоки. Вхідним потоком за замовчуванням є клавіатура, а вихідним – дисплей монітора. Запис в потік і прочитування з потоку здійснюється за допомогою операцій << і >>. Наприклад:

```
int x ;
cin >> x;     // прочитуємо значення з клавіатури в змінну x
x++;
cout << x;    // виводимо на екран змінене значення змінної x
```

Якщо треба ввести або вивести декілька змінних, то можна написати так:

```
int  x1, x2, x3;
```

```
cin >> x1 >> x2 >> x3;
x1 += x2 + x3;
cout << "Сума трьох чисел  = " << x1 << "\n";
```

Програма прочитає три цілі числа в змінні x1, x2 і x3, додасть їх і виведе на екран напис " Сума трьох чисел = ", потім результат, що зберігається в x1, і в кінці перейде на новий рядок.

Оскільки потоки – прерогатива мови Сі++, а не Сі, то докладніша робота з потоками, а також їх можливості розглядаються при вивченні курсу – об'єктно-орієнтоване програмування на мові Сі++. Надалі, в посібнику, всі приклади програм написані на мові Сі. Студент може на свій розсуд вибрати, якими операторами введення/виведення він користуватиметься, і повинен прагнути дотримання одного стилю.

5 Умовні оператори

Пам'ятаєте анекдот: "Зустрічаються два шкільні приятелі. А в школу йти нема бажання. Тоді один говорить іншому: "Давай кинемо монету! Якщо орел, то підемо в кіно, якщо решка, то на роликах кататися, ну а якщо монета на ребро впаде, то в школу". У житті часто доводиться виконувати ті або інші дії залежно від певних умов.

У математиці теж зустрічаються умови вигляду:

якщо $x > 0$, то $y = x^2$;

якщо $x < 0$, то $y = 2x + 1$.

Цю умову можна записати коротше:

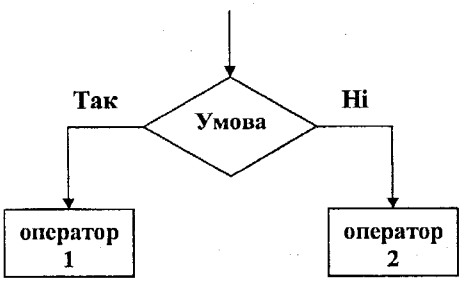
якщо $x > 0$, то $y = x^2$;

інакше $y = 2x + 1$.

Тут, залежно від умови, виконується одна з двох дій:

або $y = x^2$, або $y = 2x + 1$.

У мові Сі для програмування таких ситуацій існує умовний оператор if. Повна форма запису цього оператора виглядає так:




```
if ( умова ) оператор_1; else оператор_2;
```

Тут if (якщо), else (інакше) – ключові слова, оператор_1 і оператор_2 – прості або складені оператори. Якщо виконується умова, записана в круглих дужках, то працює оператор_1, інакше (якщо умова не виконується), то оператор_2. Іноді говорять так: "Якщо умова істинна, то виконується оператор_1, інакше

(умова помилкова) – оператор_2". Як оператори 1 і 2 можуть стояти також умовні оператори.

Для розглянутого прикладу, запис на мові Сі буде таким:

```
if (x > 0)  y = x * x;
   else  y = 2 * x + 1;
```

Значення x повинне бути визначене наперед, до виконання умовного оператора.

Умовний оператор можна зобразити за допомогою блок-схеми. У ромбії записується умова і далі від нього йдуть дві стрілки: одна з написом "Так" – по цій гілці йдемо у випадку, якщо умова істинна, і "Ні" – коли умова помилкова.

Спробуйте самі записати у вигляді блок-схеми розглянутий нами раніше приклад.

А як записати таку умову: "Якщо x > 10 і x < 40, то треба підрахувати x^3, інакше – 2*x+3". Для цього існують логічні операції. Найчастіше використовувані операції – це логічне "і", "або", "ні". Записуються вони за допомогою спеціальних символів: && – операція "і", || – операція "або", ! – операція "ні". Зазвичай, ці операції повинні давати одне з двох значень: істина або хибність. У мові Сі прийняте таке правило: істина – це будь-яке ненульове значення, хибність – це нульове значення. Вирази, що використовують логічні операції, повертають 0 для помилкового значення і 1 – для істинного. Нижче наводиться таблиця істинності для логічних операцій:

Значення x	**Значення y**	x && y	x || y	! x
0	0	0	0	1
0	1	0	1	1
1	0	0	1	0
1	1	1	1	0

Тепер ми можемо написати так:

```
if (x > 10 && x < 40)  y = x * x * x;
   else y = 2 * x + 3;
```

Спробуйте як тренування написати ту ж саму умову, але тільки за допомогою операторів "ні", "або".

В умовному операторі можна порівнювати числа (змінні, вирази) за допомогою операцій,наведених в таблиці 2:

Таблиця 2 – Знаки операцій

Знак операції	Призначення	Знак операції	Призначення
>	Більше	<=	Меньше або дорів.
>=	Більше або дорівнює	==	Дорівнює
<	Менше	!=	Не дорівнює

Умова, що керує розгалуженням обчислень не обов'язково повинна мати форму операції відношення. Вона може приймати вигляд будь-якого виразу, що розраховує результат. Якщо він відмінний від нуля, то умова, як вже було зазначено, буде істинною. Якщо 0, то помилковою.

Наприклад, запис:

```
if (x + 1) оператор1;
   else оператор2;
```

цілком допустимий. Якщо x + 1 не дорівнює нулю, то працює оператор1, інакше – оператор2.

Правила складання програми дозволяють програмісту записувати оператори у довільній формі. Проте, для зручності сприйняття програми рекомендується слово else писати під тим словом if, до якого воно відноситься, наприклад:

```
if  (a == b)
  if  (c < d)  x = 5;
  else x = 12;
else x = 34;
```

Цей приклад відповідає такому запису алгебри:

$$x = \begin{cases} 5, & \text{якщо } a = b \text{ і } c < d \\ 12, & \text{якщо } a = b \text{ і } c \geq d \\ 34, & \text{якщо } a \neq b \end{cases}$$

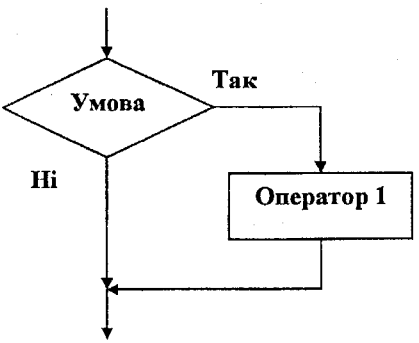
Дія умовного оператора може бути розширена використанням складеного оператора. В цьому випадку використовують групу команд, взятих у фігурні дужки:

```
if (умова)
{
   оператор_1;
   оператор_2;
   ……….
   оператор_N;
}
else
{
   оператор_1;
   оператор_2;
   ……….
   оператор_K;
}
```

Усередині складених операторів можуть бути також умовні оператори, що містять прості або складені оператори.

Ми розглянули повну форму умовного оператора. У мові допускається і коротка форма, яка має вигляд:

if (умова) оператор_1;



Якщо умова істинна, то виконується оператор_1, інакше (якщо умова помилкова) оператор_1 пропускається. У вигляді блок-схеми коротку форму запису оператора if можна подати як на схемі.

У Сі існує інша форма запису умовного оператора. Це операція знак питання – двокрапка. У загальному вигляді умовний оператор виглядає так:

результат = умова ? значення_1 : значення_2;

Якщо умова, що стоїть перед знаком питання істинна, то результату присвоюється значення_1, що стоїть до двокрапки, інакше – значення_ 2, що стоїть після двокрапки. Основна відмінність оператора ? : від оператора if полягає у тому, що if не виробляє значення, а результат операції ? : можна присвоїти змінній.

Наприклад, потрібно написати програму для знаходження максимального значення з двох чисел. За допомогою if ми б записали так:

```
int a, b, max;
……….
if (a > b) max = a;
else max = b;
```

А ось той же код, але тільки вже написаний за допомогою операції ? :

```
int a, b, max;
max =a>b?a:b;
```

Якщо a більше b, то змінній max буде присвоєне значення a, інакше – значення b.

У мові Сі прийняте таке правило. Будь-який вираз з оператором присвоювання, взятий в круглі дужки, має значення, яке дорівнює присвоєному. Наприклад, вираз (p = 5 + 3) має значення 8. Після цього можна написати інший вираз, наприклад: ((p = 5 + 3) < 10), який в даному випадку завжди даватиме істинне значення.

Така конструкція:

```
if ((ch = getch() = 'A') printf("Ви ввели першу букву алфавіту !\n");
```

дозволяє вводити значення змінної ch і виводити істинний результат тільки тоді, коли введеним значенням є буква 'A'. В цьому випадку виконається оператор printf. Жартівлива програма, що пояснює сказане:

"Зустрілися якось три жирафи, і зав'язалася у них суперечка, хто з них вищий. Довго сперечалися. Так і не дійшли згоди. Вирішили звернутися до мудреця – нехай він їх розсудить. Мудрець подумав і сказав: "той з вас найвищий, у кого щонайдовша шия". Так хто ж з них? "

```
/* Найвища жирафа */
#include <stdio.h>
#include <conio.h>
void main ()
{
    float a , b, c, max;
    clrscr () ;
    printf ("Жирафи ! Введіть кожна довжину своєї шиї : \n");
    scanf ("%f  %f %f ", &a, &b, &c);
    if(a > b && a > c)  printf("Перша жирафа найвища ! \n");
      else if (b > c) printf("Друга жирафа найвища ! \n");
         else printf("Третя жирафа найвища ! \ n");
    getch () ;
}
```

І ще один приклад: обчислити значення функції y = 1/x при заданому значенні аргументу x. Якщо введене значення x = 0, то обчислення не виконувати.

```
/* Обчислюємо y = 1/x */
#include <stdio.h>
#include <conio.h>
void main ()
{
   float x, y;
   clrscr () ;
   printf ("Введіть значення змінної x: \n");
   scanf ("%f", &x);
   if (x == 0) printf("Неприпустиме значення змінної x !\n");
   else
   {  y = 1/x;
      printf("При x = %.2f, y = %.2f \n", x, y);
   }
   getch () ; }
```

5.1 Оператор вибору switch

В умовному операторі if ми можемо піти тільки двома шляхами. Але часто бувають ситуації, коли ми повинні вибирати з великої кількості варіантів. Звичайно, це можна реалізувати за допомогою вкладених операцій if. Але в Сі передбачена спеціальна конструкція: оператор вибору switch (перемикач), яка дозволяє піти по одній з декількох гілок алгоритму. Особливо часто використовується при створенні меню. У найзагальнішому вигляді оператор виглядає так:

```
switch( вираз )
  {
    case константа_1 : варіант_1; break;
    case константа_2 : варіант_2; break;
    case константа_N : варіант_N; break;
    default : варіант_N+1;
  }
```

Тут використовуються ключові слова switch, case, default і break. Конструкція працює таким чином. Спочатку обчислюється вираз в круглих дужках. Набуте значення порівнюють зі всіма константами (константними виразами). Всі константи повинні бути різними. Якщо константа дорівнює виразу, то виконується відповідний варіант (одна або декілька інструкцій). Варіант з ключовим словом default реалізується, якщо жоден інший не підійшов (варіант default може бути відсутнім). Якщо default відсутній, а всі результати порівняння негативні, то жоден варіант не виконується. Для припинення подальших перевірок після успішного вибору деякого варіанту використовується оператор break, що забезпечує негайний вихід з перемикача switch. У операторі switch, як і в if, допускаються вкладені інструкції.

Розглянемо таку задачу: треба створити меню з чотирма пунктами. Програма подає запит на 2 числа і залежно від вибраного пункту меню додає, віднімає, перемножує, ділить між собою ці числа або просто виходить з програми.

```
/* Калькулятор */
#include <stdio.h>
#include <conio.h>
int main()
 {
   double a, b, res;
   char ch;
   clrscr () ;
   printf("Введіть два числа : ");
```

```
  scanf("%f %f", &a, &b);
  printf("Виберіть потрібний пункт меню: \n");
  printf (" 1. Додавання \n");
  printf (" 2. Віднімання \n");
  printf (" 3. Множення \n");
  printf (" 4. Ділення \n");
  ch = getch();                    // Вибираємо пункт меню
  switch( ch )    //Залежно від ch виконуємо потрібну операцію
 {
    case '1': res = a + b; break;
    case '2': res = a - b; break;
    case '3' res = a * b; break;
    case '4': res = a / b; break;
    default : printf("Ви помилилися !"); return 0;
 }
 printf("Відповідь : %2.2f", res);
 getch () ;
 return 0;
}
```

Спробуйте видалити один з операторів break в конструкції switch. Що відбудеться?

6 Цикл, цикл, цикл ...

Розв'язування багатьох задач містить дії, що повторюються. Наприклад, потрібно обчислити площу круга при десяти різних значеннях радіусу. Щоб не записувати десять разів формулу для знаходження площі, в мові Сі передбачені оператори повторення, які називаються операторами циклу. Застосування циклів в програмі дозволяє ефективно використовувати машину, приводить до скорочення довжини програми і часу на її складання та налагодження.

Для зручності програмування в мові передбачено три види операторів циклу:

while – оператор циклу з попередньою умовою;

do while – оператор циклу з подальшою умовою;

for – оператор циклу з параметром.

Для всіх трьох видів операторів циклу характерна така особливість. Дії, що повторюються, записуються всього лише один раз. Вхід в цикл можливий тільки через його початок. Необхідно передбачити вихід з циклу: або після нормального його закінчення, або за допомогою оператора break. Якщо цього

не передбачити, то циклічні дії повторюватимуться нескінченно. В такому випадку говорять, що "програма зациклилася".

У цьому розділі ми розглянемо з вами перші два оператори циклу.

Уявіть собі, що ви спринтер і під час тренування багато разів пробігаєте стометрівку. Тренер стоїть на старті і перед кожним забігом запитує вас: "Є ще сили бігти?". Якщо ви говорите "Так", то знову біжите, ну, а якщо "Ні", то біг закінчується. Це типова ситуація, яку відображає оператор циклу з попередньою умовою. Він має таку форму запису:

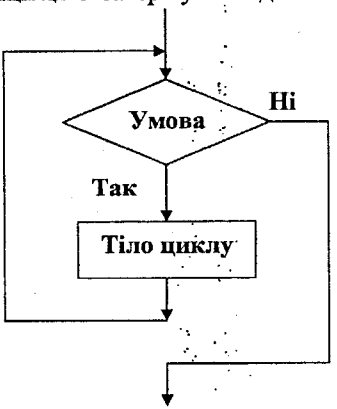
```
while (умова)
{ оператори циклічної частини програми
}
```

Слово while (поки) є службовим. Оператор циклу діє таким чином. Кожного разу заздалегідь перевіряється значення умови, що стоїть в круглих дужках. Поки вона істинна, виконуються оператори циклічної частини, взяті у фігурні дужки. Як тільки умова перестає виконуватися, відбувається вихід за межі циклу. Якщо із самого початку умова помилкова, то оператори циклічної частини не виконуються жодного разу.

Можливий випадок, коли в циклічній частині стоїть оператор break, який передає управління за межі циклу. У такій ситуації цикл завершується до його природного закінчення.

Якщо в циклічній частині стоїть всього один оператор, то фігурні дужки можна не ставити, і оператор циклу набуває вигляду:

```
while (умова) оператор;
```



Приклад. Обчислити $y = x^2$ при x = 2, 4, 6, 8, 10. Фрагмент циклічної частини програми має вигляд:

```
int x, y;
x = 2;
while (x <= 10)
{ y = x * x;
  printf("При x = %i  y = %i \n", x, y);
  x = x + 2;
}
```

За допомогою блок-схем оператор while зображається як на схемі.

Під час виконання програми змінні набувають таких значень:

X	Умова	Значення	Y
2	2≤10	Істина	4
4	4≤10	Істина	16

6	6≤10	Істина	36
8	8≤10	Істина	64
10	10≤10	Істина	100
12	12≤10	Хибність	Вихід із циклу

Нескінченний цикл

Розглянемо приклад програми:

```
#include <stdio.h>
void main ()
{
   float x = 0;
   while(x != 2)
   {
       printf("%4.1f", x);
       x = x + 0.1;
   }
}
```

Правильно було б записати оператор циклу таким чином:

```
while  (x <= 2)
```

При використанні операторів циклу можуть бути випадки, коли цикл виконуватиметься нескінченно. Наприклад, через наближене подання дійсного числа x може ніколи не виконатися точна рівність в операторі

```
while  (x == 2)
```

Розглянемо таку задачу: обчислити об'єм кожної з декількох куль, а потім знайти сумарний об'єм всіх куль. Відомо, що радіус першої кулі r, а радіус кожної подальшої кулі більше попереднього на величину dr. Радіус останньої кулі дорівнює rk. У програмі об'єм кулі позначений через V, а сумарний об'єм через Vm. Программа має вигляд:

```
/* Обчислення об'єму системи куль */
#include <stdio.h>
#include <conio.h>
void main ()
{
    const double pi = 3.14;
    float V, Vm;  // об'єм
    float r, rk;  // радіус
    float dr;     // зміна радіусу
    clrscr () ;
    printf("Введіть значення r, rk і dr: \n");
    scanf("%f%f%f", &r, &rk, &dr);
```

```
    Vm = 0;
    while(r <= rk)
    {
        V=4*pi*r*r*r/3.0;
        Vm += V;
        printf(" r = %.2f     V = %.3f \n", r, V);
        r += dr;
    }
    printf("Загальний об'ем системи :  = %.3f", Vm);
    getch () ;
}
```

А цей приклад демонструє використання циклу при роботі з символами. Дано довільний текст. Ознакою кінця тексту вважати натиснення клавіші <Enter>. Підрахувати загальну кількість введених символів тексту і число букв "N" в тексті. Оскільки наперед невідомо, скільки разів виконуватиметься цикл, для його організації використовуємо оператора циклу while. Умовою закінчення циклу є перевірка введеного символу на код 13 (код клавіші <Enter>).

```
/* Англійська приказка */
#include <stdio.h>
#include <conio.h>
void main ()
{
  int counter =0;  // лічильник букви "N"
  char ch = ' ';    // буква тексту
  clrscr () ;
  while (ch != 13)
  {
    ch = getche ();
    if(ch = = 'N' | | ch = = 'n')
    counter++;
  }
  printf("\nКількість букв 'N' в тексті = %i \n", counter);
  getch () ;
}
```

Тепер розглянемо другий оператор організації циклу do while. Давайте повернемося знову на бігову доріжку. Тільки тепер тренер стоїть на фініші. Після кожного забігу він питає: "Сили є?". Якщо говорите "Так", то знову біжите від старту до фінішу, в іншому випадку біг закінчується.

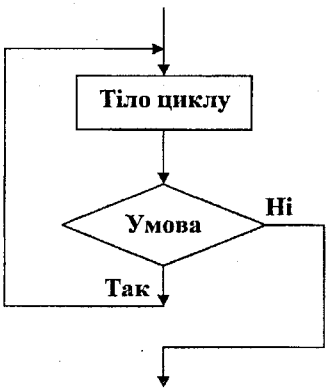
Оператор циклу з подальшою умовою має вигляд:

```
do
{
   оператори циклічної
   частини програми
}
 while (умова);
```

Слова do (робити) і while (поки) є службовими. Оператор циклу з подальшою умовою діє таким чином. Оператори циклічної частини виконуються повторно (принаймні, один раз) до тих пір, поки умова, взята в круглі дужки, залишається істинною. Отже, спочатку виконується циклічна частина, а потім перевіряється умова.

Якщо тіло циклу містить лише один оператор, то фігурні дужки можна не ставити, і оператор циклу набуде вигляду:

do оператор; while (умова);



Блок-схема циклу do-while виглядає таким чином:

При вході в конструкцію зразу ж виконується тіло циклу, а потім перевіряється умова, записана в ромбі. Вона істинна? Так – повертаємося на початок і повторюємо все наново. Ні – виходимо з циклу.

Приклад використання оператора циклу з подальшою умовою: обчислити значення функції $V = x^2$, при x = 2, 4, 6, 8, 10. Фрагмент програми має вигляд:

```
int  x,  y  ;
x  =  2;
 do
 {
     y  =  x  *  x;
     printf("%f,  %f  \n",  x,  y);
     x  +=  2;
 }
 while  (x  <=  10);
```

Тут спочатку задається перше значення аргументу x = 2. Усередині циклічної частини виконуються такі дії:

- обчислюється значення y при поточному значенні x;
- значення x і y виводяться на екран;

- обчислюється нове значення аргументу x надбавкою числа 2 до попереднього значення x.

Циклічна частина програми повторюється до тих пір, поки вираз x <= 10 не стане помилковим. В процесі виконання цієї частини програми змінні приймають такі значення:

X	y	X+=2	Умова	Значення
2	4	4	4≤10	Істина
4	16	6	6≤10	Істина
6	36	8	8≤10	Істина
8	64	10	10≤10	Істина
10	100	12	12≤10	Вихід із циклу

Написати програму, яка запитує числовий пароль. Введене число порівнюється з наперед заданим ключем. Якщо пароль набраний неправильно, необхідно повторити запит на введення числового коду. І так до тих пір, доки не буде введено правильний код. Щоб можна було працювати з великими числами, рекомендується використовувати тип long int.

```
/* Перевірка пароля */
#include <stdio.h>
#include <conio.h>
 void main ()
{
   const long int key = 128500; // заданий числовий код
   long int pass;     // пароль, що вводиться
   clrscr () ;
   do
   {
        printf("Введіть пароль : ");
        scanf("%li", spass);
    }
    while(pass != key);
     printf("Ласкаво просимо в систему !\n");
     getch () ;
}
```

6.1 Цикл for

На минулому тренуванні ми бігали з вами стометрівку до тих пір, поки були сили. Але ж можлива і інша ситуація під час тренування: тренер не питає, втомилися ви чи ні, а дає завдання: "Пробіжи стометрівку 5 разів".

У тих випадках, коли наперед відомо, скільки разів повинен повторитися цикл, використовують оператора циклу з параметром.

Оператор циклу з параметром має таку форму запису:

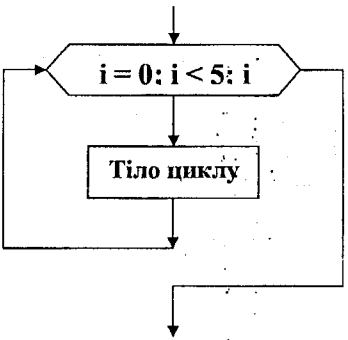
```
for(вираз_1; вираз_2; вираз_3)
{
    оператори циклічної частини програми
}
```

Тут for (для) – службове слово. Далі, в круглих дужках йдуть вирази, розділені крапкою з комою. У них найчастіше фігурує одна змінна, яку називають параметром циклу. За її значенням комп'ютер визначає, потрібно продовжувати виконувати цикл чи необхідно вийти з нього.

При першому вході в цикл працює вираз_1. Найчастіше це ініціалізація змінної циклу. Далі, на місці виразу_2 записується умова. Якщо умова істинна, то виконуються оператори циклічної частини програми. Після цього викликається вираз_3, в якому звичайно змінюється значення керуючої змінної на кожному кроці циклу. Далі знову перевіряється умова у виразі_2 і процес повторюється. Якщо умова помилкова, то відбувається вихід з циклу без виконання операторів циклічної частини.

Звичайно як параметри циклу використовують змінні цілого типу i, j, k, l і т.д.

Приклад циклу for, що виконується 5 разів за допомогою блок-схеми виглядає так:



Продовгуватий шестигранник символізує цикл for. Всередині записуються три вирази. Спочатку змінній i присвоюється значення 0. Далі перевіряється умова i < 5. Якщо це так, то виконується тіло циклу, і ми повертаємося назад, на початок. Виконуємо інкремент змінної i та знову перевіряємо умову i < 5. Якщо вона все ще виконується, то все повторюється, інакше ми виходимо з циклу.

На мові Сі цей цикл виглядає таким чином:

```
int i, a, b;
for(i = 0; i < 5; i++)
{
  /* Тіло циклу */
   a = i * i;
   b = 2 * i + 1;
   printf("%i %i \n", a, b);
```

```
}
```

Циклічна частина програми виконується 5 разів, при цьому параметр циклу і змінюється від 0 до 4. В результаті виконання програми змінні набувають таких значень:

I	A	b
0	0	1
1	1	3
2	4	5
3	9	7
4	16	9

Цей самий приклад, але із зменшенням значень параметра циклу, виглядає так:

```
int i, a, b;
for(i = 4; i >= 0; i- -)
{
   a = i * i;
   b = 2 * i + 1;
   printf( "%i   %i  \n", a, b );
 }
```

Змінні набувають таких значень:

I	A	b
4	16	9
3	9	7
2	4	5
1	1	3
0	0	1

Одночасно із зміною значення змінної циклу можна виконувати і інші операції. Наприклад, нам потрібні дві змінні, одна з яких збільшується на одиницю, а інша зменшується на двійку. Оператори записуються в круглих дужках через кому. Цикл виконується 5 разів:

```
int i, j ;
for (i = 0, j = 10; i < 5; i++, j = j - 2)
{
    printf("%i  %i \n", i, j);
}
```

Тут спочатку відбувається ініціалізація змінних і та j. Далі перевіряється умова виконання циклу i < 5. Тілом циклу є оператор виведення printf. Якщо

тіло циклу складається всього лише з одного оператора, то фігурні дужки можна не ставити.

Кожного разу, після виведення значень i та j на екран, виконується третій вираз, поданий двома операціями: i++ i j = j - 2, розділеними комами.
Змінні i та j під час роботи циклу змінюються таким чином:

i	j
0	10
1	8
2	6
3	4
4	2

Будь-який з трьох виразів в циклі for може бути відсутнім, проте крапка з комою повинна залишатися завжди. Таким чином,

for(;;) printf("Привіт, Андрію ! \n");

є нескінченним циклом. Тобто напис "Привіт, Андрію !" виводитиметься на екран нескінченно. Щоб організувати вихід з такого циклу, необхідно використати ключове слово break (перервати). Воно забезпечує негайний вихід з циклу.

Існує ще один корисний оператор – оператор continue (продовжити). Якщо він зустрічається всередені операції циклу, то вирази, що йдуть за ним, пропускаються, і керування передається на наступний крок циклу (див. приклад 2 далі).

В цьому прикладі необхідно вивести на екран всі букви латинського алфавіту. Оскільки букви латинського алфавіту впорядковані (див. таблицю ASCII-кодів), то можна скласти таку програму:

```
/* Друк букв латинського алфавіту */
#include <stdio.h>
#include <conio.h>
void main ()
{
    char ch;
    clrscr () ;
    for(ch = 'A'; ch <= 'Z'; ch++)
       putchar(ch);
    getch () ;
}
```

Приклад 1. Кожний будній день рибак ходив ловити рибу. Скільки він зловив риб в середньому за день, якщо щоденний улов був таким: 8, 7, 5, 9, 7, 9?

У програмі спочатку вводиться кількість днів, позначених n, а потім організовується цикл по числу днів. Усередині циклу вводиться чергове

значення r – число спійманих риб за день. Це значення додається до накопичуваної суми S.

```
/* Ловися рибка велика і маленька */
 #include <stdio.h>
 #include <conio.h>
void main ()
{
      int i;        // лічильник днів
      int n;        // кількість днів
      int r;        // число риб за день
      float s=0;   // середнє число риб
      clrscr () ;
      printf("Скільки днів рибак ловив рибу ? ");
      scanf("%i", &n);
      for(i = 0; i < n; i++)
      {
        printf("Скільки риб він зловив за день ? ");
        scanf("%i", &r);
        s += r;
      }
       s /= n;
       printf ("-----------------------------------------\n") ;
       printf("Середнє число риб за день = %.lf \n", s);
      getch () ;
}
```

Приклад 2. Написати програму, що обчислює в циклі значення функції y=10/sqrt(x), де sqrt(x) – функція математичної бібліотеки, яка знаходить арифметичний квадратний корінь з числа x. Оскільки значення аргументу може бути тільки додатним, то при введенні від'ємного числа, обчислення y пропускається, і цикл виконується наново. Якщо ввести нуль (що теж не можна – ділення на нуль), то відбувається вихід з циклу.

```
/* Обчислюємо y = 10/sqrt(x)*/
 #include <stdio.h>
 #include <conio.h>
 #include <math.h>
 void main ()
{
      int x;
      float y;
```

```
    clrscr () ;
    for(;;)   // нескінченний цикл
    {
      printf("Введіть число :");
      scanf ( "%i", &x );
      if(x = = 0) break;        // вихід з циклу
      if(x < 0) continue;       // повторне введення числа
      y = 10 / sqrt( x );
      printf("При x = %i   y = %.2f \n", x, y);
    }
     getch () ;
}
```

6.2 Вкладені цикли

Цикли можуть бути вкладені один в інший. При використанні вкладених циклів необхідно складати програму так, щоб внутрішній цикл повністю укладався в циклічну частину зовнішнього циклу. Внутрішній цикл може, в свою чергу, містити інший внутрішній цикл (цикли). Усередині одного циклу можуть бути розташовані відразу декілька циклів, кожний з яких може містити також декілька циклів.

```
for(i = 0; i < 10; i++)
  for(j = 0; j < 7; j++)
    for(j = 0; j < 3; j++)
    {
      тіло циклу;
    }
for(i = a; i < b; i++) {
    for(j = c; j < d; j++)
   {
     тіло циклу 1;
   }
    while (i + j < z)
    {
      тіло циклу 2;
    }
 }
```

Для демонстрації розглянемо такий приклад: необхідно підрахувати кількість щасливих квитків. Квиток вважати щасливим, якщо в його шестизначному номері сума перших трьох цифр дорівнює сумі останніх трьох цифр і дорівнює 13.

Для обчислення в програмі організовуємо потрійний вкладений цикл для знаходження кількості перших трьох чисел, сума яких дорівнює 13. Цю кількість позначимо через n. Тоді загальна кількість щасливих квитків буде рівна n^2.

```
/* Щасливі квитки */
#include <stdio.h>
#include <conio.h>
void main ()
{
    int i, j, k;  // змінні циклів
    int n = 0;    // лічильник щасливих квитків
    clrscr () ;
   for(i = 0; i < 10; i++)
      for(j = 0; j < 10; j++)
        for(k =0; до < 10; k++)
            if(i + j + k == 13) n++;
   n *= n;
   printf ("Число щасливих квитків = %i\n", n);
   getch () ;
}
```

Приклад. Обчислити значення функції y = 3x - t при всіх значеннях x = 1.5, 2, 2.5, 3 і t = 1, 3, 5, 7. У програмі можна організувати два цикли: один зовнішній, по параметру x, а інший внутрішній, по параметру t. Тоді для кожного x змінна t набуватиме значень t = 1, 3, 5, 7. І так всього вийде 16 значень y.

```
/* Вкладені цикли */
#include <stdio.h>
#include <conio.h>
void main ()
{
     double x, t;
     double y;
     clrscr () ;
    printf("  x    t     y    \n");
     printf ("--------------------------------\n") ;
     x = 1.5;
     while(x <= 3)
     {
       t = 1;
```

```
        while (t <= 7)
        {
          y = 3 * x - t;
          printf("%5.1f %5.1f %5.1f \n", x, t, y);
          t += 2;
        }
        x += 0.5;
    }
    getch () ;
}
```

7 Масиви

При написанні різних програм іноді нам доводиться мати справу з великою кількістю даних одного типу. Наприклад, у нас є сім цілих чисел:

5, 13, -3, 72, 0, 27, 34

Ми можемо записати ці числа в змінні, наприклад

as, vd, lop, f, кок, is, ddr.

Але краще б було придумати одне загальне ім'я, а змінні пронумерувати (або, як ще говорять, проіндексувати), як в математиці:

a_0, a_1, a_2, a_3, a_4, a_5, a_6.

У повсякденному житті ми теж зустрічаємося з такою формою запису. У потягу, наприклад, всі місця пронумеровані, а людей, що їдуть у вагоні ми можемо назвати одним словом – пасажир. І звернутись до нього можна так: "Пасажир 13 місця, пред'явіть квиток!".

У програмуванні для зручності роботи з даними одного типу введене поняття масиву. **Масив** – це кінцева сукупність даних одного типу, впорядкованих за значеннями одного типу. Кожен елемент масиву позначається ім'ям масиву з індексом.

Як же правильно записати масив? У математиці, фізиці, хімії і ін. точних науках часто можна зустріти такі форми запису масивів:

- за допомогою нижніх індексів:

  a_0, a_1, a_2, a_3, a_4, a_5, a_6

- за допомогою індексів, взятих в круглі дужки:

  a(0), a(1), a(2), a(3), a(4), a(5), a(6)

- за допомогою фігурних дужок:

  $\{a_i;\}$, 1 = 0, 1,2, ...n

В мові Сі індексація виконується за допомогою квадратних дужок:

a[0], a[1], a[2], a[3], a[4], a[5], a[6]

Тут a – ім'я масиву, а число, вказане в дужках – індекс. Зверніть увагу на той факт, що нумерація починається з нуля.

Якщо ми хочемо використовувати масив в програмі, то ми повинні спершу оголосити його, як і у випадку із змінними. Потрібно вказати тип масиву (тобто тип змінних, що зберігаються в ньому), ім'я масиву і його розмір. Ім'я масиву повинно відповідати правилам створення імен ідентифікаторів. Тобто в імені можна використовувати тільки букви, цифри і знак підкреслення, і ім'я не повинне починатися з цифри. Загальний вид опису масиву такий:

```
тип ім'я[розмір];
```

Наприклад, оголошення масиву для семи цілих чисел:

```
int a[7];
```

Оскільки нумерація масиву при звертанні до його елементів починається з 0, то останній доступний нам елемент a[6]. У мові немає контролю при виході за межі масиву. Тобто якщо ми напишемо a[9], то програма скомпілюється і запуститься, але значення, одержане з масиву, в цьому випадку буде невизначеним. Насправді ми одержимо число, що знаходиться в пам'яті на 10-у місці від початку масиву. А оскільки масив оголошений тільки на 7 елементів, що лежить на 10 місці – ніхто не знає. Тому потрібно дотримуватися обережності при роботі з масивами. Якщо спробувати записати дані за межі масиву, то можна зіпсувати іншу інформацію, що зберігається в пам'яті. Якщо це виявляться системні дані, то робота операційної системи і інших програм може бути порушена. Часто неправильна індексація масиву призводить до зависання ОС або навіть до її перезавантаження.

Можна описувати одночасно декілька масивів одного типу:

```
int a[0], b[17], c[50];
char str[40], ch[5];
```

Розмір масиву (тобто кількість його елементів) повинен бути заданий цілим додатним числом або числовою константою. Запис вигляду:

```
int i;
int mas[i];
```

недопустимий, оскільки i – це змінна, і компілятор не знає, скільки заздалегідь необхідно виділити пам'яті під масив. Правильний опис має такий вигляд:

```
const int n = 10; int mas [ n];
```

або просто

```
int mas[10];
```

При описі масиву його можна зразу ініціалізувати, тобто задати всім елементам початкові значення:

```
double arg[3] = {3.56, 7.32, -0.01};
```

В цьому випадку arg[0]= 3.56, arg[1]= 7.32, arg[2]= -0.01. При ініціалізації розмір масиву можна не указувати (він автоматично визначається шляхом підрахунку кількості перераховуваних значень):

```
char str[] = { ' a', ' b', ' c', ' d', ' e', 'f' };
```

Під масив забуде виділено 6 байтів, оскільки кожен його елемент займає 1 байт (тип char), а їх всього шість. І str[0] = ' a', str[1] = 'b',... str[5] = ' f '.

Елементи масиву можуть стояти як в лівій, так і в правій частині оператора присвоювання, так і у виразах. Тобто їх можна використовувати точно так, як і змінні:

```
b[3] = b[5] + 1;
sum += a[k];
f1 = mas[2 * i + 1];
```

Масиви звичайно використовуються в циклах. Наприклад, цикл

```
int i;
int mas[10];
……….
for( i = 0; i < 10; i++ )
      scanf ( "%i", &mas[i] );
```

організовує введення десяти значень елементів масиву mas. А ось цикл

```
int i;
int mas[10];
……….
for( i = 0; i < 10; i++ )    printf( "%i", mas[i]);
```

виводить на екран 10 значень з масиву mas.

Розглянемо програму, що використовує масиви: вводиться рядок з десяти символів, який потім відображається на екрані дисплея в зворотному порядку. Наприклад, якщо ввести рядок 0123456789, то він зміниться таким чином: 9876543210. Текст програми наведено нижче

```
# include <stdio.h>
main() {
  char s[10]; /* оголошення масиву s з 10 елементів символьного типа.
                 індекси масиву приймають значення від 0 до 9 */
   int i;
   puts( "введіть рядок з десяти символів");
   for(i=0;i<10; i++)
   scanf("%c", &s[i]);  /* введення елементів масиву s */
   for(i=9; i>=0; i--)
   printf("%c",s[i]);/*виведення елементів масиву s в зворотному порядку */
}
```

7.1 Багатовимірні масиви

У мові Сі є можливість роботи з багатовимірними масивами. Наприклад, двовимірний масив можна інтерпретувати як матрицю. Кожна комірка матриці задається двома індексами – номером стовпця і рядка. Тривимірний масив можна зобразити у вигляді куба. Тут вже будуть три індекси – стовпець, рядок і ряд. Перетин відповідних значень – комірка (у вигляді маленького кубика), в якій зберігаються дані (див. рис.2).

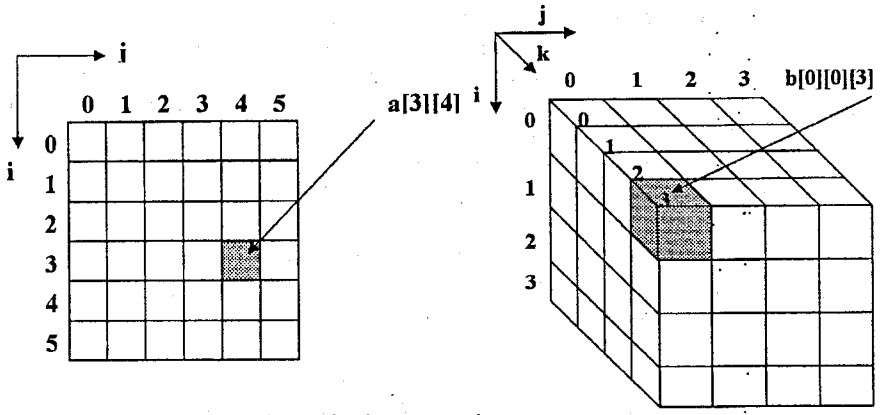


Рисунок 2 – Інтерпретація багатовимірного масиву

Оголошення багатовимірного масиву виконується таким чином:

тип ім'я [розмір1] [розмір 2] ... [розмір N] ;

Наприклад, щоб оголосити масиви, зображені на рисунку, достатньо написати:

```
int a[6][6];
double b[4] [ 4] [ 4];
```

Двовимірний масив можна розглядати як масив масивів. Дійсно, кожний рядок – це одновимірний масив. І у нас декілька (масив) таких рядків. Щоб звернутися до будь-якого елементу масиву, ми повинні вказати номер рядка, а потім номер позиції в рядку (стовпець). Наприклад:

```
a[0][0] = 10;
a[j][i] = a[i][j] + 4;
z = a[3][4]++;
```

З тривимірним масивом ситуація аналогічна. Це є ні що інше, як масив таких матриць, поставлених поряд. Додається ще один вимір –

номер матриці. Звертання до елементів такого масиву задасться трьома індексами:

```
b[3][0][1]= 3.14;
f = b[i] [j] [k]+ b[i] [j] [k];
b[0][0][3] *= 7.34;
```

Аналогічно можна визначити масив з будь-якою розмірністю (тільки б пам'яті вистачило). Дані "кубики" можна роздати студентам, і отримати чотиривимірний масив – новий індекс – номер студента. Студенти сидять в різних класах – номер класу – ще один індекс – у результаті виходить п'ятивимірний масив. На кожному поверсі інституту декілька таких класів. Номер поверху – ще один вимір. Одержали шестивимірний масив. І так можна продовжувати нескінченно довго. Тепер, щоб знайти значення в певному "кубику", треба задати номер поверху, номер класу на цьому поверсі, номер студента, що сидить в цьому класі, а також номер стовпця, рядка і ряду в його "кубику". Але на практиці рідко використовуються масиви з розмірністю більше двох.

При оголошенні багатовимірного масиву, його елементам так само можна задати початкові значення:

```
intmas[2][3] = {{7, 6, 33}, {-4, 56, 0}};
int arr[][] = {{4, 5, 7}, {2, 4, 6}};
```

Для зберігання масиву виділяється суцільна ділянка пам'яті. Якщо ми оголосили двовимірний масив цілих чисел int a[2][3]; то елементи цього масиву лежатимуть в пам'яті послідовно таким чином: a[0][0], a[0][1], a[0][2], a[1][0], a[1][1], a[1][2]. Зверніть увагу, на те, що найшвидше змінюється крайній правий індекс у масиві.

Двовимірні масиви звичайно обробляються у вкладених циклах. Керуюча змінна зовнішнього циклу – перший індекс масиву (змінюється у другу чергу), а керуюча змінна внутрішнього циклу – другий індекс масиву (змінюється в першу чергу). Наприклад, цикл

```
int  i, j;
int bob[10] [20];
……..
for(i = 0;  i < 10;  i++)
{
for(j = 0;  j < 20;  j++)
printf(" %3i ",  bob[i][j]);
printf("  \n  ");
}
```

роздруковує масив bob у вигляді матриці. Для кожного значення змінної і виконується 20 разів тіло внутрішнього циклу, тобто ми виводимо і-ий

рядок масиву. Далі, кожного разу після завершення цього циклу переходимо на новий рядок на екрані, і для наступного значення змінної і повторюємо ті ж дії.

Приклад. Вивести на екран таблицю множення 10×10. Для зберігання таблиці множення використовується масив цілих чисел a.

```
/* Таблиця множення*/
#include <stdio.h>
#include <conio.h>
void main ()
{
   const int n = 10;                          // розмірність масиву
   int a[n][n], i, j;
   clrscr () ;
   for(i = 0; i < n; i++)
       for(j = 0; j < n; j++)
          a[i] [j]= (i+ 1) * (j + 1);  // заповнення таблиці множення
   for(i = 0; i < n; i++)
   {
     for(j = 0; j < n; j++)
        printf("%4i", a[i][j]);       // виведення масиву на екран
     printf("\n");
   }
   getch () ;
 }
```

Приклад. У квадратній матриці 5×5, заповненій випадковими числами, поміняти місцями головну і побічну діагональ.

```
/* Міняємо в матриці місцями діагоналі */
#include <stdio.h>
#include <stdlib.h>
#tinclude <conio.h>
void main ()
{
   const int n = 5;                          // розмір матриці
   int a[n][n], i, j, tmp;
   clrscr () ;
   randomize ();
   for(i = 0; i < n; i++)
  {
      for(j = 0; j < n; j++)
```

```
        {
            a[i][j]= random(100); // заповнюємо матрицю випадковими
                                  // числами
            printf ("%3i", a[i][j]); // і зразу ж виводимо її на екран
        }
        printf("\n");
    }
    for(i = 0; i < n; i++)
    {
        tmp = a[i][i];    // за допомогою тимчасової змінної tmp
        a[i][i]= a[i][n-i-1];    // виконуємо заміну значень на діагоналі
        a[i][n-i-1]= tmp;        // матриці в одному рядку
    }
    printf("\n");
    for(i = 0; i < n; i++)
    {
      for(j = 0; j < n; j++)
          printf("%3i", a[i][j]);  // виводимо змінену матрицю на екран
          printf("\n");
    }
    getch () ;
}
```

Досить цікавим різновидом масивів є рядки. У мові Сі – немає спеціального рядкового типу, в Сі++ – є , але його використання не є обов'язковим, оскільки можна користуватися способом мови Сі. Рядки визначаються як масиви символів, тобто масиви типу char. При зберіганні рядка останнім його символом є спеціальний символ з ASCII кодом 0, передаваний як \0. Він не відображається на екрані, проте зберігається в останньому елементі, тому довжина масиву для зберігання рядка повинна на одиницю перевищувати його довжину. Якщо створити масив типу char певної довжини, наприклад:

```
char  str[80];
```

то в ньому можна буде зберегти рядок будь-якої довжини, що не перевищує 79. Помістити рядок в масив можна або посимвольно, при цьому не забути про нульовий символ в кінці, або за допомогою стандартних функцій. Також можна розмістити рядок при створенні масиву, наприклад, так:

```
char  str[6] = {'H', 'e', 'l', 'l', 'o' };
```

Або так:

```
char  str[6] = "Hello";
```

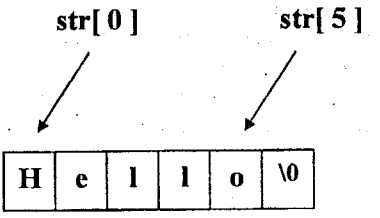
Як і у разі звичних масивів, тут можна написати, так:

```
char str[] = {'H', 'e', 'l', 'l', 'o'};
```

Або так:

```
char str[] = "Hello";
```

У пам'яті елементи рядка зберігатимуться, як показано на рисунку:



Проте і в цьому випадку довжина масиву визначається при його створенні, і розмістити в цьому масиві довший рядок не вдасться. Тому рекомендується створювати рядок максимальної довжини, тобто 255 символів, навіть якщо спочатку в нього записується коротший рядок. В цьому випадку «зайві» комірки будуть виділені, але не використовуватимуться, а надалі це дозволить уникнути багатьох проблем. Отже, рекомендований спосіб створення масиву приблизно такий:

```
char str[255]= "";
```

для спочатку порожнього рядка, і:

```
char str[255]= "Рядок тексту";
```

для не порожнього рядка.

7.2 Покажчики

Всі дані, які використовуються у програмі, зберігаються в пам'яті комп'ютера. Пам'ять, у свою чергу, розбита на комірки по 8 бітів (тобто по 1 байту), і кожна комірка пронумерована, тобто має свою адресу. Адреси задаються цілими числами у порядку зростання. У мові Сі є можливість звертатися до даних в оперативній пам'яті комп'ютера напряму, тобто через адресу комірки. Багато в чому завдяки цій можливості мова і набула такого значного поширення серед програмістів.

Для роботи з адресами пам'яті в мові вводиться особливий тип змінних – покажчики. Отже, **покажчик** — це змінна, яка може зберігати в собі адресу пам'яті іншої змінної (масиву, функції, об'єкту). Як і інші змінні, покажчики повинні бути оголошені до їх використання в програмі. Це робиться так:

```
тип *ім'я;
```

Тут тип визначає тип об'єктів, адреси яких можуть зберігатися в покажчику, ім'я – ім'я покажчика. Ім'я може бути вибране будь-яке, відповідно до правил створення ідентифікатора. Перед ім'ям ставиться зірочка, що і є відмінною ознакою покажчика. Приклади оголошення покажчиків:

```
int *a, *b, *c;
```

```
char  *d;
```
Оголошення char *d говорить про те, що значення, записане за адресою d, має тип char.

Щоб записати адресу деякої змінної в покажчик (або як говорять, примусити покажчик дивитися на цю змінну), потрібно перед ім'ям змінної поставити знак операції & (узяття адреси), а одержаний результат присвоїти покажчику:

```
int  x;
int  *p;
p  =  &x;
```
Тепер покажчик p зберігатиме в собі адресу змінної x. Операцію & не можна застосовувати до констант і виразів; конструкції вигляду &(x + 5) або &27 недопустимі.

А чи можна, знаючи адресу, взнати, яке значення приховано за цією адресою? Так, можна. Для цих цілей визначена операція *. Іноді її називають "прохід за адресою". Якщо перед покажчиком поставити зірочку, то значення, що повертається, буде числом що зберігається за адресою, записаною в покажчику.

Наприклад, якщо p = &x; y = *p; то y = x.

Покажчики можуть зустрічатися і у виразах. Якщо p – покажчик на ціле, тобто мало місце оголошення int *p;, то *y може з'явитися будь-де, де і будь-яка інша змінна, що не є покажчиком. Таким чином, такі вирази цілком допустимі:

```
*p  =  7;
*m  *=  5;
(*g)++;
```
Перше з них заносить число 7 в елемент пам'яті за адресою p, друге збільшує значення за адресою m в п'ять разів, третє додає одиницю до вмісту комірки з адресою g. У останньому випадку круглі дужки необхідні, оскільки операції * і ++ з однаковим пріоритетом виконуються справа наліво. В результаті, якщо наприклад, g = 5, то (*g)++ приведе до того, що *g = 6, а *g++ всього лише змінить саму адресу g (операція ++ виконується над адресою g, а не над значенням *g за цією адресою).

Покажчики можна використовувати як операнди в арифметичних операціях. Якщо p – покажчик, то операція p++; збільшить його значення; тепер воно є адресою наступного елементу. Покажчики і цілі числа можна додавати. Конструкція p + n (p – покажчик, n – ціле число) задає адресу n-го об'єкту, на який указує p. Це справедливо для будь-яких об'єктів (int, char, float і т.п.); транслятор масштабуватиме приріст адреси відповідно до типу, визначеного з відповідного запису.

Тепер самий час розглянути взаємозв'язок між масивами і покажчиками. Як вже було описано вище, адреса конкретного елемента обчислюється компілятором, виходячи з адреси першого елемента, тобто засади масиву, і індексів та розміру базового типу, що дуже схоже на арифметику покажчиків. Насправді, масиви і покажчики – це практично одне й те саме. З одного боку, щоб одержати адресу масиву можна узяти адресу першого його елемента, з другого боку, ім'я масиву без квадратних дужок саме по собі є покажчиком на масив. Одержану тим або іншим способом адресу масиву можна присвоїти покажчику на той самий тип, який є типом даного масиву. Після цього, можна звертатися до будь-якого елементу масиву, збільшуючи або зменшуючи в рамках арифметики покажчиків значення покажчика на необхідну кількість елементів. З другого боку, будь-який покажчик, навіть якщо він зберігає не адресу масиву, можна індексувати за допомогою квадратних дужок, таким чином одержуючи зсув на ту кількість байтів, яку дає добуток індексу на розмір типу покажчика. Отже, нехай є масив:

```
int arr[4] = {1,2,3,4};
```

і покажчик:

```
int *p;
```

Тоді покажчику можна присвоїти адресу масиву або так:

```
p = &arr[0] ;
```

або так:

```
p = arr;
```

Після цього можна вивести на екран значення третього елементу масиву, його індекс дорівнює 2, так:

```
printf("%i ", arr[2]);
```

або так:

```
printf("%i ", *(p+2));
```

або навіть так:

```
printf p%i  ", p[2]);
```

Наведемо невеликий приклад для перевірки:

```
#include <conio.h>
#include <stdio.h>
void main ()
{  clrscr () ;
int arr[4]= {1,2,3,4};
int *p;
p=arr;
printf("%i %i %i \n", arr[2], *(p+2), p[2]);
getch () ; }
```

В результаті на екрані побачимо три трійки, що свідчить про правильність вищенаведених тверджень.

Нарешті, через покажчики можна визначати рядки, тільки в цьому випадку необхідно зразу ж присвоїти їм початкове значення, інакше програма може видавати дивовижні результати, оскільки довжина рядка не буде спочатку визначена. Природно, навіть при ініціалізації не завжди можна поручитися, що рядки більшої довжини, ніж початкові, коректно оброблятимуться. Проте, такий спосіб теж має право на існування. Наведемо приклад:

char *s="hello"; повністю еквівалентний char s[]="hello";

Як і у разі звичних масивів, такий покажчик можна змінювати з використанням арифметики покажчиків, тоді рядок збільшуватиметься або зменшуватиметься, тобто або в рядок не входитимуть перші символи, або як перші видаватимуться символи, відповідні даним, розташованим в пам'яті до рядка, а тільки потім решта рядка. Важливо розуміти, що ознакою кінця рядка є символ '\0', тому його довжина визначається компілятором автоматично, а не прописується в першому елементі, як це робиться в деяких інших мовах програмування. Крім того, такий покажчик теж можна індексувати, використовуючи квадратні дужки. Наведемо приклад:

```
#include <conio.h>
#include <stdio.h>
 void main ()
 {
   clrscr () ;
   char str[255];
   printf("Введіть рядок тексту: ");
   scanf("%s", str);
   printf("Рядок дорівнює = %s \n", str);
   printf("Третя буква = %c \n", str[4]);
   printf("Зсунемо покажчик управо на один символ %s", str + 1);
   getch () ;
 }
```

На екрані з'явиться запрошення ввести рядок. Наприклад, ми надрукували: "Hello". Комп'ютер виведе на екран "Hello", потім п'яту букву цього рядка 'o'. Останній оператор виведення друкуватиме рядок str починаючи з 2 -го символа ('e') і до кінця рядка ('\0'). У нашому випадку на екрані з'явиться: "ello". При введенні рядка в операторі scanf необхідно указувати адресу пам'яті, починаючи з якої записуватиметься значення, що вводиться. Але оскільки ім'я масиву – str є одночасно адресою початку

рядка, то замість цілком законного запису scanf("%s", &str[0]); можна написати простіше:

scanf("%s", str[0]).

Зверніть увагу на те, що специфікатор введення/виведення для рядка – %s, в той час як для символу – %c.

Приклад. Змінити задачу про введення пароля так, щоб як пароль можна було використовувати будь-яку послідовність символів, а не тільки ціле число. Тим самим ми різко зменшуємо шанси підібрати пароль.

```
/* Програма-ревізор */
#include <stdio.h>
#include <conio.h>
#include <string.h>
 void main ()
{
   const char password[] = "diablo2";  // рядок-пароль
   char pass[255];// рядок, що вводиться
   clrscr () ;
   do
  {
      printf("Введіть пароль : ");
      scanf("%s", pass);
   }                                  // поки pass не дорівнює рядку
   while(strcmp(pass, password)!= 0); // password, виконуємо цикл
   printf("Ласкаво просимо в систему !\n");
   getch () ;
}
```

У програмі використана нова функція strcmp(str1, str2), яка вміє порівнювати два рядки посимвольно. Якщо рядки однакові, то вона повертає нуль. Якби ми написали просто while(pass != password), то тоді б вийшов нескінченний цикл, оскільки в цьому випадку порівнюються покажчики, тобто адреси початку рядків pass і password. А вони, звичайно ж, мають різні значення.

Функції для роботи з рядками визначені в заголовному файлі string.h, який і підключається на початку програми. Повний перелік функцій роботи з рядками можна знайти за допомогою help'a.

8 Функції

Ви звертали увагу на той факт, що в текстах пісень, де є приспіви, їх не пишуть кожного разу після чергового куплета, а написавши тільки один раз, далі, в пісні, просто друкують слово "приспів". У програмуванні також часто доводиться мати справу з кодом, що повторюється. І щоб не займатися

безглуздим його розмноженням, достатньо один раз описати дану ділянку коду у вигляді окремої функції, а далі, в місцях, де необхідно виконати даний код, просто викликати цю функцію.

У функцій є дуже багато переваг: по-перше, скорочується код програми. Програма складається з викликів функцій, за назвою яких можна визначити виконувані ними дії. Виходить, щось подібне до короткого плану. Програма стає простішою. Створивши один раз функцію, ви можете її використовувати знов і знов, зменшуючи ризик виникнення помилок. Також функції можна збирати разом для створення цілих бібліотек, які ми підключаємо на початку файлу за допомогою директиви #include.

При створенні власних функцій необхідно дотримуватись таких правил:

- Функція не може бути описана усередині іншої функції. У мовах Сі/Сі++ всі функції глобальні.
- Кожна функція має тип значення, що повертається, ім'я, список вхідних параметрів і тіло функції.
- Ім'я функції повинно бути вибране за правилами створення ідентифікаторів. Доцільно відобразити в імені функції ті дії, які вона виконує. Наприклад, якщо ви написали функцію піднесення до кубу, то нерозумно її називати Vasja(), навіть якщо автор її Вася. Більш відповідне для неї ім'я Cube().
- Функція повинна бути описана перед головною функцією main() або мати прототип.

Наведемо найзагальніший приклад опису написання програми з функцією:

```
/* опис функції */
тип ім'я_функції(параметри)
{
    тіло функції;
};
/* основна функція */
 main ()
{
   тіло функції main;
}
```

Спочатку йде опис функції з вказанням типу вертаного значення, ім'ям функції і переданими їй параметрами. Далі у фігурних дужках записується тіло функції – ті самі дії, які часто зустрічаються у нас в програмі. Виконання програми завжди починається з функції main. В програмі може бути тільки одна функція main(). Усередині неї і відбуваються виклики описаної нами функції. Приклад :

```
#include <stdio.h>
```

```
#include <conio.h>
void Stars()              // опис функції Stars
{
   int i;                 // оголошення локальної змінної
   for (i = 0; i < 10; i++)
   putchar('*');
   printf("\n");
};
 void main() // головна функція – початок виконання програми
{
    clrscr () ;
   Stars();    // виклик функції Stars()
   printf("Привіт, Андрію !");
   Stars();    // виклик функції Stars()
}
```

Тут ми описуємо функцію Stars(), яка друкує 10 зірочок і переходить на новий рядок. Наша функція нічого не повертає, а тип значення, що повертається, повинен бути вказаний перед ім'ям функції. Саме для цих цілей був придуманий тип void – ніщо. Далі, в круглих дужках ми повинні вказати вхідні параметри функції. У нас їх теж немає, але круглі дужки обов'язкові. Усередині функції можуть бути визначені свої внутрішні, або як їх ще називають, локальні змінні. Вони будуть доступні тільки усередині функції і ніяк не перетинаються із змінними у функції main(). Тому, якщо в main() у нас є така ж змінна цілого типу і, то компілятор зрозуміє, що це різні змінні і не сплутає їх.

Давайте зробимо нашу функцію Stars() більш універсальною: хотілося б, щоб вона друкувала кожного разу не по 10 зірочок, а стільки, скільки їй скажуть. Для цього можна ввести глобальну змінну n, яка описується поза всіма функціями – на самому початку, і значення якої доступне з будь-якого місця програми. Отже:

```
#include <stdio.h>
#include <conio.h>
int n;          // опис глобальної змінної
void Stars ()  // опис функції Stars ()
{
    int i;      // оголошення локальної змінної
    for (i = 0; i < n; i++)       // цикл: вивести n зірочок
        putchar ('*');
    printf("\n"); };
```

```
  void main()  // головна функція – початок виконання програми
{
  clrscr () ;
  n = 5;
  Stars(); // виклик функції Stars()
  printf("Привіт, Андрію !");
  n = 15;
  Stars(); // виклик функції Stars()
}
```

Перед кожним викликом функції Stars() ми присвоюємо змінній n певне значення, яке позначає кількість зірочок, що виводяться на екран. Програма стала універсальною, але можна зробити ще краще. Зараз ми пов'язали нашу функцію з глобальною змінною n, що, чесно кажучи, недобре. Уявіть таку ситуацію: ви написали класну функцію і включили її в бібліотеку. Програмісту достатньо підключити бібліотечний файл за допомогою директиви #include і він зможе користуватися вашою функцією. Але він абсолютно нічого не знає, про глобальну змінну n і, відповідно не знає, як задати кількість зірочок. А якщо функції потрібно задати декілька параметрів, то для кожної заводити глобальну змінну?

Для цієї мети передбачений механізм передачі параметрів всередину функції. У описі функції ми указуємо скільки, яких типів і під якими іменами передаються параметри всередину функції. А при виклику функції в круглих дужках указуємо передані значення. Ними можуть бути числа, змінні, константи, результати інших функцій і т.д. Перепишемо наш приклад:

```
#include <stdio.h>
#include <conio.h>
void Stars (int n)              // опис функції Stars () з параметром
{
   int i;                       // оголошення локальної змінної
   for (i = 0; i < n; i++)      // цикл: вивести n зірочок
      putchar('*');
   printf("\n");
};
  void main()       // головна функція – початок виконання програми
{
  int z = 7;                    // опис локальної змінної z
  clrscr () ;
  Stars(5);                     // виклик функції Stars() з параметром 5
  printf("Привіт, Андрію !");
```

```
    Stars(z + 3);              // виклик функції Stars() з параметром 10
}
```

У визначенні функції Star() ми в круглих дужках написали, що функція при виклику вимагатиме як параметр ціле число, яке запишеться в змінну n. Таку змінну ще іноді називають ***формальним параметром***. Цикл усередині функції відпрацює n раз, після чого змінна n буде знищена. При новому звертанні до функції Star( ) ми указуємо нове значення вхідного параметра.

Часто функція виконує які-небудь розрахунки, і результат цих обчислень потрібно повернути назад в програму. Тип результату, що виконується функцією, вказується при її описі перед ім'ям. Значення повертається усередину функції за допомогою оператора return. Напишемо функцію піднесення числа до квадрату.

```
#include <stdio.h>
#include <conio.h>
float Square(float x)    // опис функції Square() з параметром
{
    float res;
    res = x * x;
    return res;
};
void main ()
{
    float a,b,c;              // опис локальної змінної z
    clrscr () ;
    a = 5.5;
    b = Square(a);            // обчислюємо квадрат числа a
    c = Square(b +3);         // обчислюємо квадрат числа b + 3
    printf("Результат обчислень = %.2f", c);
}
```

Описана функція Square() має один вхідний параметр типу float. Це значення записується усередині функції в змінну x. Далі в локальну змінну res записується квадрат значення x. За допомогою оператора return це значення повертається на те місце, де була викликана функція Square().

Всередину функції може бути передано багато параметрів, але повертає функція тільки одне значення. Значення, передані у функцію, перераховуються через кому. Напишемо функцію, що обчислює середнє арифметичне двох цілих чисел:

```
  // Функція обчислення середнього двох цілих чисел
```

```
  float Average(int x, int y)
  {
   -return (x + y) /2.0;
  };
void main ()
 {
   int a, b;
   printf("Введіть два числа : ");
   scanf("%i %i", &a, &b);
   printf("Середнє арифметичне цих чисел = %.lf \n", Average(a, b));
 }
```

У функції описані два вхідні параметри цілого типу. Не дивлячись на те, що тип у них один, ми не можемо написати так:
float Average(int x, y). Тобто для кожної змінної необхідно указувати її тип.

Усередині однієї функції можуть зустрічатися декілька операторів return. Наприклад, функція для знаходження максимального числа з двох цілих:

```
int max(int a, int b)
{
   if (a > b) return a;
   else return b;
}
```

Якщо у нас в програмі визначається багато функцій, то іноді зручніше визначити їх після функції main(), а перед нею подати їх прототип:

```
#include <stdio.h>
#include <conio.h>
int max(int a, int b);      // прототип функції max
void main()                 // функція main()
{
   float a, b, c;
   clrscr () ;
   a = 5.5;
   b = Square (a);
   c = Square(b + 3);
   printf("Результат обчислень = %.2f", c);
};
int max(int a, int b)    // реалізація функції max()
 {
   if (a > b) return a;
   else return b; }
```

Імена формальних змінних можуть збігатися з іменами інших змінних в програмі, оскільки на них поширюються правила видимості, так що в цьому плані немає приводу для занепокоєння. Якщо необхідно використовувати у функції глобальну змінну, то перед змінною достатньо поставити :: (дві двокрапки), і змінна інтерпретуватиметься як глобальна.

Функції можуть викликати інші функції, якщо вони були оголошені або визначені наперед, у тому числі і самі себе. Це явище називається ***рекурсією***. Рекурсія буває корисна в деяких обчисленнях. Але при цьому слід пам'ятати про те, що часто помилки програміста можуть призвести до нескінченної рекурсії, з другого боку при рекурентному або рекурсивному (кому як подобається) виклику функції одночасно в пам'яті знаходиться декілька її екземплярів, у тому числі і всіх формальних змінних. При великій глибині рекурсії швидкість обчислень зменшується, більш того, дуже велика глибина рекурсії може викликати збій і крах програми. А втім, наведемо програму, яка обчислює *факторіал* невід'ємного цілого числа, тобто добуток всіх натуральних чисел від 1 до даного числа, використовуючи рекурентний спосіб обчислення. Факторіал числа k=1×2×3× (k-1)× k і записується як k! (читається: " k факторіал"). Наприклад, 51 = 1×2×3×4×5 = 120. За правилом, 0! = 1.

```
/* Обчислення факторіалу */
#include <conio.h>
#include <iostream.h>
unsigned long factorial(int);
main ()
{
    int a;
    unsigned long res;
    clrscr () ;
    printf("Введіть ціле невід'ємне число : ");
    scanf("%i", &a);
    res = factorial(a);
    if (res = = 0) printf("Ви ввели неправильне число! \n");
      else printf("Факторіал %i = %li \n", a, res);
    getch () ;
    return 0;
};
unsigned long factorial(int b)
 {
    int c=l;
    if (b < 0) return 0;
```

```
  else if(b < 2) return 1;
       else return b * factorial(b – 1);
}
```

У прототипі можна не указувати імена формальних параметрів, а тільки вказати їх тип, як це зроблено в даному прикладі. У функції main за замовчуванням тип вертаного значення int. І якщо нічого не написати, на місці, де потрібно вказати тип, вертаний функцією, то це буде рівносильно опису вигляду int main(). Тому в кінці функції main ми написали оператор return з довільним цілим значенням. Звичайно 0 означає, що функція main() успішно завершила роботу. Якщо не написати return, то помилки не буде, але компілятор видасть попередження (warning).

Розглянемо ще одне важливе питання: як передавати масиви як параметр всередину функції. Відповідь напрошується сама собою. Потрібно передавати не самі значення масиву (адже їх може бути дуже багато), а покажчик на початок масиву, тобто адресу в пам'яті (а це тільки одне число), де лежить масив.

Наприклад:

```
/* Рядок, що рухається */
#include <stdio.h>
#include <conio.h>
#include <dos.h>
void OutText(char *str)   // функція вимагає покажчик на тип char
{
   int i = 0;
   while (str[i] != '\0' )        // цикл: поки не кінець рядка, виконувати
   {
      printf("%c", str[i]);
      sound(500);
     delay(100);
     nosound();
     i++;
   }
}
void main ()
 {
    char name[20];
    textcolor (14);
    textbackground(0);
    clrscr () ;
```

```
    OutText("Привіт ! \n");
    OutText("Мене звуть Федя. А тебе як? \n");
    scanf("%s", name);
    OutText(name);
    OutText(" давайте жити дружно! \n");
    getch () ;
}
```

У програмі реалізована функція OutText(), вхідним параметром якої є покажчик на тип char (тобто рядок). Рядок виводиться в циклі while() посимвольно, до тих пір, доки не зустрінеться символ кінця рядка '\0'. Після виведення кожної букви відбувається подача звукового сигналу за допомогою функції sound(частота) з бібліотеки dos.h, де частота – ціле число, виражене в герцах. Для ноти "Ля", наприклад, це 440 Гц. Далі, йде затримка на 0.5 секунди, після чого звук вимикається за допомогою функції nosound(), збільшується лічильник букв і, цикл повторюється наново.

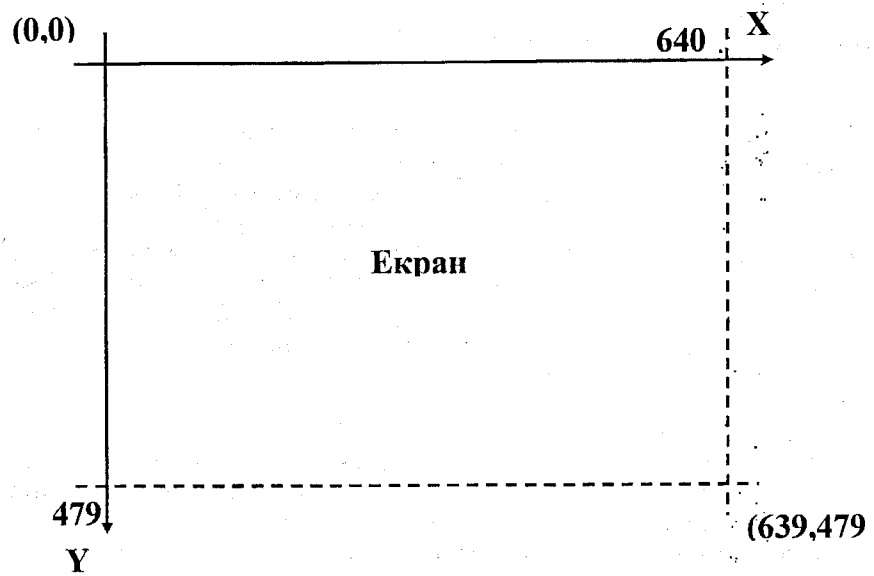
9 Графіка Borland C++

Відеорежимом називається набір параметрів, підтримуваний апаратурою відеокарти (відеоадаптера). На відміну від текстового режиму у відеорежимі можливе виведення не тільки текстової інформації, але і графічної. Зображення формується з точок, так званих *пікселів* (pixel). Найвідомішим атрибутом відеорежиму є *дозвіл екрана.* Наприклад, 640×480. Це означає, що на екрані виводиться 640 пікселів по горизонталі і 480 пікселів по вертикалі. Зустрічаються і інші дозволи: 320×200, 640×480, 800×600, 1024×768 і т.д.

Початок системи координат (тобто точка з координатами (0,0)) знаходиться в лівому верхньому кутку екрана. Вісь X направлена вправо, як і в евклідовій системі координат, а вісь Y, на відміну від звичної системи координат – вниз. Тому всі рисовані об'єкти матимуть додатні координати як по x, так і по y. Важливо відзначити, що оскільки нумерація пікселів починається з нуля, то саме права нижня точка, видима на екрані, матиме координати (639, 479).

Відеорежими також розрізняються за **глибиною пікселів** (pixel depth). Цей параметр визначає кількість різних значень, яких набуває окремий піксель, і, отже, кількість кольорів, що відображаються. Наприклад, у відеорежимі з глибиною пікселів 8 бітів кожен піксель може мати один з 256 різних кольорів (28=256). У режимах з 16-бітовою глибиною пікселів підтримується відображення до 65536 кольорів. Глибина пікселів звичайно дорівнює 4, 8, 16, 24 або 32 бітам. Ми з вами працюватимемо в одному з

найпростіших графічних режимів – 640×480 з глибиною кольору 4 біти, тобто в нашому розпорядженні буде 16 кольорів. Такий режим називається VGA.



Графічні режими з великим дозволом і великою кількістю кольорів називають SVGA (SuperVGA) режимами. Ще подекуди можна зустріти старі графічні режими, такі як EGA, CGA, в яких використовувалося від 2 до 16 кольорів, а дозвіл не перевищував 320×200.

Для організації графіки фірма BORLAND випустила ряд досить універсальний графічних **драйверів** — програм, які забезпечують взаємодію програмної і апаратної частини комп'ютера. Графічні драйвери знаходяться в каталозі BGI і мають таке ж розширення, а саме *.bgi (BGI – Borland Graphics Interface). Ім'я файлу звичайно відображає для роботи з яким саме графічним режимом він призначений. У нашому випадку це файл egavga.bgi. Далі ми побачимо, як його підключити в програмі. Також необхідно в меню Options → Linker → Libraries... поставити хрестик в рядку Graphics library, якщо цього ще не зробили до вас. Слід відмітити, що організація графіки в BORLAND C++ 3.1 зроблена так само, як і в Borland Pascal. Тобто, якщо комусь довелося програмувати на Pascal'і в середовищі BORLAND, то багато речей вам виявляться знайомими. Тепер ми можемо приступити до написання скелета програми.

Для роботи з графікою необхідно підключити графічну бібліотеку graphics.h. Далі йде код програми, необхідний для включення графічного режиму.

```
/* Шаблон програми для роботи з графікою */
#include<graphics.h>
#include<conio.h>
void main ()
{
   int gd, gm;
   gd = DETECT;
   initgraph(&gd, &gm, "");
   // Тут йде послідовність графічних функцій.
   getch () ;
   closegraph ();
}
```

Спочатку ми підключаємо використовувані бібліотеки. Далі нам знадобляться дві змінні цілого типу. Це gd і gm (скорочення від graphics detect і graphics mode). Перша визначає драйвер графічного режиму (CGA, EGA, VGA і т.д.). Якщо змінній присвоїти значення DETECT (як зроблено в нашому випадку), то визначення **(детектування)** найбільш відповідного драйвера і відеорежиму відбувається автоматично. У другу змінну заноситься назва графічного режиму. У нашому випадку цій змінній ми нічого не присвоюємо, але вона нам все одно потрібна. Функція initgraph() переводить екран в графічний режим. Вона має три аргументи. Перші два – це адреси змінних gd і gm. Як третій аргумент передається рядок, що містить шлях до файлу egavga.bgi. Насправді можна скопіювати цей файл в каталог BIN, з якого запускається BORLAND C++ 3.1. Тоді немає необхідності указувати повний шлях, а потрібно всього лише передати порожній рядок. В цьому випадку пошук драйвера виконуватиметься в поточному каталозі. Далі пишуться графічні функції, що формують зображення на екрані. Наприклад, лінії, прямокутники, кола. Після закінчення роботи в графічному режимі необхідно коректно завершити роботу з ним і вийти в текстовий режим. Якщо ми цього не зробимо, система автоматично зробить це за нас, але це вже буде аварійна ситуація і наступне відкриття графічного режиму може пройти невдало. Закриття графічного режиму здійснюється за допомогою функції closegraph(). Ніяких вхідних параметрів ця функція не вимагає. Для того, щоб ми побачили сформоване програмою зображення, треба зробити затримку, як і у випадку з текстовим режимом, наприклад, функцією getch().

Далі йде перелік графічних функцій, які стають доступними при підключенні заголовного файлу graphics.h. Всі змінні, використовувані у

функціях, мають цілий тип і звичайно позначають координати (x, y, xl, yl, x2, y2) або відстань в пікселях (dx, dy, depth, r, rx, ry), колір (color) або значення кута в градусах (start і end).

setbkcolor(color) – встановлює колір фону. Змінна color може набувати значень від 0 до 15.

setcolor(color) – встановлює поточний колір. Саме цим кольором рисуються всі фігури.

Щоб взнати поточні установки кольору фону і поточного кольору, можна скористатися, відповідно, функціями:

int getbkcolor() – повертає колір фону,

int getcolor() – повертає поточний колір,

putpixel(x, y, color) – рисує точку з координатами (x,y) кольором color.

Ми можемо не тільки рисувати точки, але і знати, які точки вже нарисовані на екрані. Це можна зробити за допомогою функції int getpixel(x,y). Наприклад, якщо у нас є змінна цілого типу color, то вираз color = getpixel(x,y); записує значення кольору точки (x,y) в змінну color.

line(xl, yl, x2, y2) – рисує лінію поточним кольором від точки (xl,yl) до точки (x2, y2).

rectangle(xl, yl, x2, y2) – рисує прямокутник із сторонами, паралельними краям екрану, одна з діагоналей якого є відрізком з координатами (xl,yl) і (x2,y2). Ці точки можна сприймати, як верхній лівий і нижній правий кутки прямокутника.

bar(xl, yl, x2, y2) – рисує аналогічно rectangle() прямокутник і зафарбовує за поточним шаблоном і поточним кольором заповнення.

bar3d(xl, yl, x2, y2, depth, flag) – рисує аналогічно bar() зафарбований прямокутник і на його основі створює «тривимірну» панель завглибшки depth. Якщо flag не рівний 0, то верхня частина панелі зафарбована.

circle(x, y, r) – рисує коло радіусом r з центром в точці (x,y).

arc(x, y, start, end, r) – рисує дугу кола радіусом r з центром в точці (x,y) від кута start до кута end.

pieslice(x, y, start, end, r) – рисує сектор кругової діаграми – круговий сектор радіусом r з центром в точці (x,y) від кута start до кута end. Зафарбовується за поточним шаблоном кольором заповнення.

ellipse(x, y, start, end, rx, ry) – рисує дугу еліпса від кута start до кута end, вимірюваних в градусах, і з радіусами по x – rx і по y – ry. При заданні start = 0; і end = 360; рисується еліпс (не плутати з овалом!). Якщо значення rx і ry рівні між собою, то вийде коло.

fillellipse(x, y, rx, ry) – рисує зафарбований еліпс з центром в (x,y) з радіусами rx і ry.

sector(x, y, start, end, rx, ry) – рисує еліптичний сектор з центром в (x,y) від кута start до end і радіусами rx, ry.

outtext("рядок тексту") – виводить рядок тексту, взятий в подвійні лапки в поточну позицію на екрані.

outtextxy(x, y, "рядок тексту") – виводить рядок тексту, взятий в подвійні лапки, за координатами (x, y).

Є функції установки різних стилів. Наприклад,

setlinestyle(style, pattern, width) – завдання стилю лінії. Параметр style визначає стиль лінії і може набувати значень від 0 до 4 (неперервна, точкова, штрих-пунктирна, пунктирна). Якщо style=4, то у змінну pattern записується десяткове подання 16-ти бітового шаблону, в якому нулі позначають погашені точки, а одиниці – точки, що світяться. Параметр width набуває значення 1 або 3 і визначає товщину лінії.

setfillstyle(p, color) – встановлює тип заливки. Параметр p задає узор заливки і може набувати значень від 0 до 12. Який номер відповідає якому узору, студентам пропонується перевірити самостійно. Наприклад, в циклі нарисувати 13 зафарбованих прямокутників в один рядок, змінюючи стиль заливки для кожного. Другий параметр – color задає колір заливки.

Крім того, за допомогою функції **floodfill(x, y, borde)** можна зафарбувати будь-яку замкнуту фігуру, межа якої має колір border, а точка (x,y) знаходиться усередині області, яку потрібно зафарбувати.

Більш детальні приклади написання програм з використанням графічних функцій можна знайти в посібнику авторів „Лабораторний практикум для вивчення мови Сі". Повний перелік функцій графічного режиму можна одержати за допомогою вбудованої системи help'a.

10 Анімація

Це, напевно, найцікавіша тема. Під анімацією ми з вами розумітимемо швидко змінне зображення екрану, нарисовані об'єкти якого змінюють свої координати і тим самим рухаються по екрану. Ми можемо задати їм певну траєкторію руху або примусити їх підкорятися певним законам, які, можливо, динамічно мінятимуться в ході виконання програми.

Давайте подумаємо, як можна організувати анімацію. Нам необхідний цикл, в тілі якого послідовно формуватимуться кадри зображення і швидко виводимуться на екран. Перед прорисовуванням кожного кадру, потрібно очищати всю область екрану і потім рисувати вже змінену картинку. Вихід з циклу – натиснення будь-якої клавіші. Усередині циклу поставимо невелику затримку, щоб людське око встигало сприймати кожен рисований кадр.

Розглянемо приклад: написати програму, в якій кулька рухається зліва направо. Досягши правої межі вона перестрибує у своє початкове положення, і все повторюється спочатку. Для рисування кульки (замість кульки може бути будь-яка інша фігура), ми визначимо функцію Draw(x, y), при виклику якої відбувається прорисовування зображення.

```
/* Рухома кулька */
#include <conio.h>
#include <graphics.h>
#include <dos.h>
void Draw(int xl, int yl)
{
    circle(xl, yl, 20);
}
void main ()
{
    int x, y;
    int gd = DETECT, gm;
    x = 20;                    // ініціалізація початкових
    y=200;                     // координат кульки
    initgraph(&gd, &gm, "");
    while(!kbhit()) // Виконувати цикл: поки не натиснута будь-яка
                    // клавіша
   {
       cleardevice();          // очищення екрану
       Draw(x, y);             // рисуємо кульку за координатами (x, y)
       delay(10);              // затримка
       x++;                    // зміщуємо кульку вправо на один піксель
       if(x > 620) x =20;      // якщо дійшли до правої межі, то x = 20.
   }
    getch () ;
   closegraph ();
}
```

Основний цикл програми – while. Функція kbhit() повертає не нуль, якщо була натиснута будь-яка клавіша і нуль, якщо буфер клавіатури порожній. Функція описана у файлі conio.h. За допомогою функції delay(мілсек) з бібліотеки dos.h, ми організовуємо затримку на 10 мілісекунд для кожного кадру.

У написаної нами програми є серйозний недолік – зображення мерехтить. Чому так відбувається? Річ у тому, що функція cleardevice() працює не дуже

швидко, оскільки треба зафарбувати весь екран, і наше око встигає уловити момент, коли на екрані нічого немає. Що ж робити? Перше, що приходить в голову – це спробувати очищати не весь екран, а тільки ту область, де нарисований об'єкт. Давайте перепишемо програму так, щоб спочатку рисувалася кулька кольором фону за старими координатами, а потім – іншим кольором на новому місці.

```
/* Рухома кулька - 2 */
 #include <conio.h>
 #include <graphics.h>
 #include <dos.h>
void Draw(int xl, int yl)
{  circle (xl, yl, 20);
  }
void main ()
{   int x, y;
    int gd = DETECT, gm;
    x = 20;
    y = 200;
    initgraph(&gd, &gm, "");
    setbkcolor(1);          // встановлюємо колір фону 1 – синій
    while(!kbhit())         // виконувати цикл: поки не натиснута клавіша
   {
       setcolor(1);         // рисуємо кольором фону кульку за
       Draw(x, y);          // старими координатами (x, y)
       x++;
       if (x > 620) x = 20;
       setcolor(10);        // рисуємо  зеленим кольором
       Draw(x, y);          // кулька за оновленими координатами (x, y)
       delay(10);           // затримка
   }
   getch () ;
   closegraph (); }
```

Якщо у вас комп'ютер досить швидкий, то ви помітите невелике поліпшення, але мерехтіння повністю не зникне. І чим складніша фігура у функції Draw, тим довше працює прорисовування (і стирання!) зображення, і тим більше помітне мерехтіння. Тобто такий спосіб годиться для анімації простих картинок. Для того, щоб можна було позбавитися мерехтіння, необхідно зрозуміти принцип роботи монітора і механізм формування зображення. Усередині кінескопа є так звана електронно-променева пушка, яка "стріляє" електронами (зарядженими частинками) по внутрішній поверхні

екрану. Ця поверхня покрита спеціальною люмінісцентним речовиною, яка світиться при зіткненні з електронами. Пучок електронів можна відхилювати за допомогою магнітного поля і при необхідності вимикати. У кожен момент часу на екрані світиться тільки одна точка – те місце, куди летять електрони. Ця пляма, що світиться, починає рухатися з лівого верхнього кутка по горизонталі. Дійшовши до кінця екрану, промінь вимикають, і переводять на початок наступного рядка. Цей процес називається горизонтальним ретрасуванням (див. рис.3). Так продовжується, поки промінь не пройде всі рядки екрану, і не опиниться в правому нижньому куті. Далі промінь знову вимикають, і переводять в початок екрану. Це – вертикальне ретрасування. Промінь "бігає" настільки швидко, що людське око сприймає всі точки, як єдине ціле. За одну секунду екран монітора обновляється 60 разів! В цьому випадку говорять, що вертикальна розгортка екрану 60 Гц. Чим вище це число, тим менше втомлюються очі. Безпечним вважається значення, яке дорівнює 75 Гц і вище.

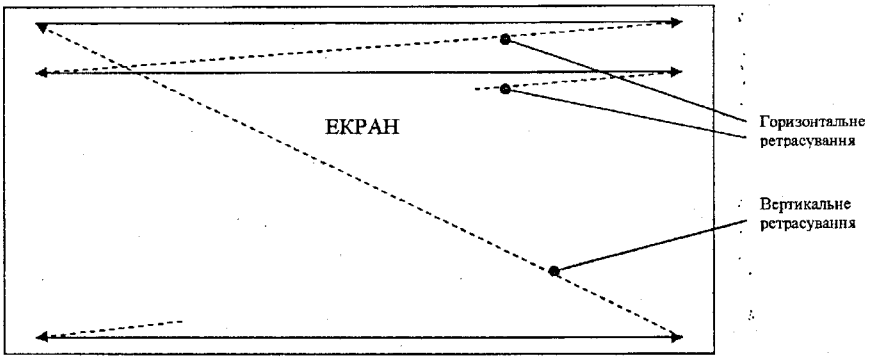


Рисунок 3 – Зображення екрана при ретрасуванні

Зараз існують монітори з частотою вертикальної розгортки до 120 Гц.

Дані про зображення беруться з відеопам'яті. Це є не що інше, як масив пікселів. І коли ми "рисуємо" що-небудь на екрані, ми насправді записуємо нові значення в цей масив. При проходженні променя по екрану прочитується значення відповідного пікселя з відеопам'яті, і піксель виводиться на екран потрібним кольором (інтенсивністю). Що ж буде, якщо ми запишемо у відеопам'ять нові значення пікселів в той момент, коли у нас вже частина попереднього зображення вже сформована? Пам'ять оновиться (швидкість читання/запису в пам'ять набагато більша, ніж швидкість переміщення променя по екрану), і далі промінь рисуватиме вже новий кадр.

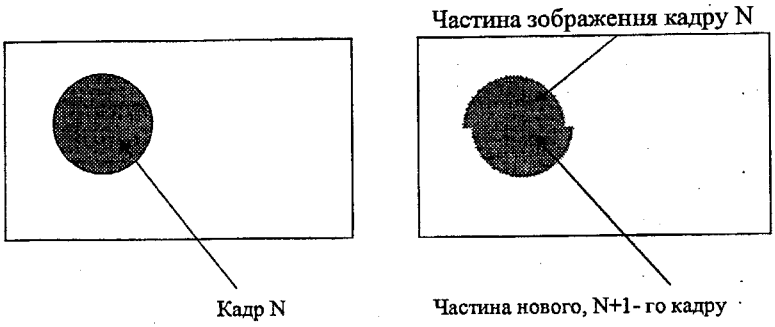


Рисунок 4 – Оновлення відеопам'яті

Вийде ситуація, показана на рисунку 4. Ми бачитимемо частину зображення від попереднього кадру і частину від наступного. Як же розв'язати цю проблему? Потрібно просто оновлювати відеопам'ять під час вертикального ретрасування, коли кадр N вже нарисований, а кадр N+1 ще навіть не чіпали.

Нижче наводиться функція, яка чекає початку найближчого вертикального ретрасування. Вона перевіряє четвертий біт порту 0x3DA. Якщо він дорівнює 1, то йде вертикальне ретрасування, якщо 0, то зображення кадру ще до кінця не сформовано. Як тільки промінь досяг правого нижнього кутка, виставляється 1 і функція завершує роботу, надаючи нам можливість швидко відновити відеопам'ять.

Функцію доцільно ставити безпосередньо перед перерисуванням (перемиканням відеосторінок).

```
void WaitWerticalRetrace()
{
  while (inportb(0xSDA) & 0x08);
  while (! (inportb(0xSDA) & 0x08));
}
```

Приклад. Написати програму, в якій кулька, що має початкову швидкість, рухається рівномірно по екрану, відскакуючи від його стінок.

```
/* Міні - більярд */
#include <graphics.h>
#include <stdio.h>
#include <conio.h>
#include <dos.h>
void WaitWerticalRetrace()
```

```
{
  while (inportb(OxSDA)& 0x08);
  while (! (inportb(OxSDA)& 0x08));
}
void main(void)
{
   int gd = DETECT, gm;
   int x, y, dx, dy;
   initgraph(&gd, &gm, "");
   setbkcolor(1);       // колір фону 1 – синій
   x = 100;
   y=100;               // задаємо початкові координати
   dx = 2;              // і швидкість по горизонталі
   dy =2;               // і по вертикалі
   while (!kbhit())     // цикл: поки не натиснута будь-яка клавіша
  {
      setcolor(1);
     WaitWerticalRetrace(); // чекаємо початку вертикального
                            // ретрасування
     circle (x, y, 15);     // рисуємо кольором фону коло
     x += dx;               // збільшуємо x на dx
     y += dy;               // і y на dy
     if(x > 625 || x < 15)
        dx = -dx;           // якщо край екрану, то змінюємо знак у dx
      if(y > 465 || y < 15)
         dy = -dy;          // якщо край екрану, то змінюємо знак у dy
     setcolor (14);         // рисуємо коло жовтим кольором
     circle (x, y, 15);     // за новими координатами
  }
 closegraph();
}
```

Для запам'ятовування і переміщення зображень застосовуються також динамічні змінні. Для цього використовуються фушкції getimage, imagesize, putimage.

Для запам'ятовування бітового зображення заздалегідь повинен бути виділений необхідний об'єм ОП за допомогою функції malloc. Докладніше робота з динамічним зображенням розглянута в посібнику авторів частина 2 "Лабораторний практикум для вивчення мови Сі".

Таблиця 3 – Таблиця кольорів, що використовуються в графічному модулі

0	BLACK	Чорний
1	BLUE	Синій
2	GREEN	Зелений
3	CYAN	Голубий
4	RED	Червоний
5	MAGENTA	Фіолетовий
6	BROWN	Коричневий
7	LIGHGRAY	Світло-сірий
8	DARKGRAY	Темно-сірий
9	LIGHTBLUE	Світло-синій
10	LIGHTGREEN	Яскраво-зелений
11	LIGHTCYAN	Яскраво-блакитний
12	LIGHTRED	Світло-червоний
13	LIGHTMAGENTA	Яскраво-фіолетовий
14	YELLOW	Жовтий
15	WHITE	Білий

11 Робота з файлами

У файлах розміщуються дані, призначені для тривалого зберігання. Кожному файлу призначається використовуване при звертанні до нього унікальне ім'я. Для роботи з файлами в Сі необхідно підключити стандартну бібліотеку stdio.h. Всі файли можна умовно розділити на дві категорії: текстові і двійкові (або ще їх називають бінарні).

Текстові – це файли, в яких записаний тільки текст. Такі файли звичайно мають розширення *.txt, *.ini, *.bat, *.log і т.д. Наприклад, файли, в яких містяться початкові тексти програм, також є текстовими. Тобто файли з розширеннями *.c, *.cpp, *.pas, *.asm і т.д. можна переглянути будь-яким текстовим редактором, наприклад WordPad.

Вся решта файлів – файли, що містять інформацію, коди програм, рисунки, звукові дані і т.д., є бінарними. Наприклад, *.exe, *.com, *.avi, *.dat, *.bmp – двійкові файли. Файли з розширенням *.doc теж відносяться до двійкових, оскільки окрім тексту вони можуть містити і іншу інформацію, наприклад рисунки, графіки, таблиці.

Перш ніж читати або записувати інформацію у файл, він повинен бути відкритий. Це можна зробити за допомогою бібліотечної функції fopen. Вона повертає покажчик на файл. Далі в програмі ми працюватимемо саме з покажчиком на файл, а не писатимемо кожного разу ім'я того або іншого файлу. Покажчик на файл необхідно оголосити. Це робиться так:

FILE *ist;

Тут FILE – ім'я типу, описане в стандартному визначенні stdio.h, ist – покажчик на файл. Звернення до функції fopen в програмі виконується виразом:

ist = fopen("ім'я файлу", "вид використання файлу");

Ім'я файлу може бути, наприклад:

"c:\\my_file.txt", "a:\\doc\ \data.txt", "temp. txt " і т.і.

Вид використання файлу може бути:

"rt" – відкрити існуючий текстовий файл для читання.

"wt" – створити новий текстовий файл для запису (якщо файл з вказаним ім'ям вже існує, то він буде переписаний),

"at" – доповнити текстовою файл.

"rt+" – відкрити існуючий текстовий файл для запису і читання.

"wt+" – створити новий файл для запису і читання.

Тут використовуються скорочення від англійських слів r (read) – читати, w(write) – писати, a (add) – додавати. Буква t (text) – означає, що ми будемо виконувати всі операції з текстовим файлом. Якщо ми хочемо працювати з бінарним файлом, то замість t треба написати b (binary). Наприклад, ist = fopen("max.dat", "rb") – відкриває двійковий файл max.dat з поточного каталога для читання. Якщо в результаті звертання до функції fopen виникає помилка (наприклад, файлу max.dat не існує), то вона повертає покажчик на константу NULL. Тому після кожного виклику функції fopen бажано перевіряти значення посилання покажчика.

Після закінчення роботи з файлом він повинен бути закритий. Це робиться за допомогою бібліотечної функції fclose. Як вхідний параметр їй необхідно передати покажчик на той файл, який ми хочемо закрити.

Розглянемо інші бібліотечні функції, які використовуються для роботи з файлами (всі вони описані у файлі stdio.h):

1. Функція putc записує символ у файл і має такий опис:

   putc(ch, ist);

   ch – символ для запису (змінна типу char), ist – покажчик на файл, повернений функцією fopen.

2. Функція getc читає символ з файлу і має такий вигляд:

   ch = getc(ist);

   Тут так само ch типу char, ist – покажчик на файл, з якого читається черговий символ і записується в ch. Якщо викликати цю функцію в циклі, то будуть послідовно прочитані всі символи з файлу. Досягши кінця файлу функція поверне значення константи EOF (End Of File) – кінець файлу.

3. Функція feof визначає кінець файлу при читанні двійкових даних і має такий опис:

   status = feof (ist);

   Тут ist – покажчик на файл, status – змінна типу int, в яку досягши кінця файлу записується ненульове значення. Інакше записується 0.

4. Функція fputs записує рядок символів у файл. Вона відрізняється від функції puts тільки тим, що як другий параметр повинен бути записаний покажчик на файл. Наприклад:

```
FILE *ist;
ist = fopen("my.txt", "wt");
fputs("Привіт, Андрію!", ist);
fclose (ist);
```

   записується у файл, пов'язаний з покажчиком ist, рядок тексту "Привіт, Андрію!".

5. Функція fgets читає рядок символів з файлу. Вона відрізняється від функції gets тільки тим, що як другий параметр повинен бути записаний покажчик на файл. Наприклад:

```
FILE *p;
char str[50];
p = fopen("c:\\story.txt", "rt");
fgets(str, p);
fclose(p);
```

6. Функція fprintf виконує ті ж дії, що і функція printf, але працює з файлом. Її відмінність у тому, що як перший параметр задається покажчик на файл. Розглянемо приклад:

```
FILE *p;
int x = 10;
p = fopen("a:\\data.txt", "wt");
fprintf (p, "Значення змінної x = %i", x) ;
fclose(p);
```

   Функція виводить рядок тексту і значення змінної x, але тільки не на екран, як це робила функція printf, а у файл a:\data.txt.

7. Так само є функція, аналогічна функції scanf. Це fscanf. Знову ж таки, єдина відмінність у тому, що у останньої перший параметр – покажчик на файл, і читає інформацію вона не з клавіатури, а з файлу.

```
FILE *p;
char str [100] ;
int x, y;
p = fopen("noname.cpp", "rt");
```

```
fscanf (p, "%s %i %i", str, &x, &y);
fclose(p);
```

Тут ми читаємо з файлу рядок тексту (до першого пропуску) і далі два цілі числа x і y.

8. При відкритті файлу ми автоматично опиняємося в його початку, і далі переміщаємося по файлу, читаючи послідовно дані. А що коли нам необхідно прочитати тільки одне число, що знаходиться на 100-ій позиції. Або потрібна інформація знаходиться в кінці файлу. Не перечитувати ж весь файл! Якраз для цього існує функція fseek, яка дозволяє виконувати читання і запис з довільної позиції файлу. Вона має вигляд:

```
fseek(ist, count, access);
```

Тут ist – покажчик на файл, count – номер байта щодо заданої початкової позиції, access – спосіб задання початкової позиції. Змінна access може набувати таких значень:

0 – початкова позиція задана на початку файлу.

1 – початкова позиція вважається поточною.

2 – початкова позиція вважається з кінця файлу.

Приклади з функцією fseek:

```
/* Читає 10-ий символ з файлу з покажчиком ist */
  fseek(ist, 10, 0);
  ch = getc(1st);
  /* Читає 5-ий символ з кінця файлу */
  fseek(ist, -5, 2);
  ch = getc (ist);
  /* Читає з файлу 10 символів через один і виводить їх на екран */
  for(i = 0; i < 10; i++)
  {
      fseek(ist, 2, 1);
      fscanf (ist, "%c", &ch);
      printf("%c", ch);
  }
```

Наведемо приклад програм, одна з яких створює текстовий файл і зберігає в ньому дані різних типів, а друга прочитує ці дані з файлу і виводить їх на екран.

```
#include <stdio.h>
#include <conio.h>
int main()
{
   FILE *f;
```

```
    char name[15];                  // ім'я
    int age;                        // вік
    float weight;                   // вага
    f = fopen("c:\\database.txt", "wt");  // створюємо на диску C: файл
    if (f == NULL)
    {
          printf("Помилка створення файлу ! \n");
          return 1;       // вихід з програми
    }
    /* Прочитуємо дані з клавіатури */
    printf("Введіть ім'я студента : ");
    scanf("%s", name);
    printf("Введіть його вік : ");
    scanf ("%i", sage);
    printf("Введіть вагу : ");
    scanf ("%f", &weight);
    /* Тепер збережемо все це у файл */
    fprintf(f, "%s \n %i \n %f", name, age, weight);
    fclose(f);      // закриємо файл
    return 0;
}
```

І програма, яка прочитує дані з файлу:

```
#include <stdio.h>
#include <conio.h>
int main()
{
    FILE *f;
    char name[15];   // ім'я
    int age;         // вік
    float weight;    // вага
    f = fopen("c:\\database.txt", "rt");   // відкриємо файл
    if (f == NULL)
    {
          printf("Не можу відкрити файл ! \n");
          return 1;   // вихід з програми
    }
    /* Прочитуємо дані з файлу */
```

```
    fscanf(f, "%s %i %f", name, sage, sweight);
    /* Виводимо дані на екран */
    printf("Ім'я студента : %s \n", name);
    printf("Вік : %i \n", age);
    printf("Вее : %f \n", weight );
    fclose(f);      // закриємо файл return 0;
}
```

І ще один приклад програми, яка виводить на екран вміст текстового файлу:

```
/* Програма-переглядач текстових файлів */
#include <stdio.h>
#include <conio.h>
void main ()
{
    FILE *f;  // Покажчик на файл
    char ch;  // Читаний символ
    f = fopen("c:\\my.txt", "rt");
    while (feof (f) == 0)  // цикл: поки немає кінця файлу f, виконуємо
цикл
    {
      ch = getc(f);
      printf("%c", ch);
    }
    fclose (f);
    getch () ;
}
```

12 Технологія програмування в середовищі BORLAND C++

Процес розробки програм з використанням інтегрованого середовища Borland C++ включає створення програмних файлів з розширенням .cpp і обробку програми, зокрема:

– компіляцію програми: формування obj-файлів з cpp-файлів;
– компоновку (редагування зв'язків) програми: формування файлів з obj- файлів;
– виконання програми: обробку даних відповідно до заданого алгоритму.

Спрощена схема процесу розробки програм в середовищі Borland зображена на рис. 5.

В процесі виконання різних етапів обробки і виконання програми можуть бути знайдені і видані повідомлення про помилки.

12.1 Основні види роботи з текстовими файлами

Розробка текстових файлів за допомогою текстового редактора системи Borland C++ включає такі види робіт: створення нового файлу, його збереження на МД і виклик файлу з МД у вікно редактора.

Створення нового файлу може включати створення тексту на екрані і доповнення його текстом з раніше створеного файлу, розміщеного на МД. Це можуть бути, наприклад, файли з розширеннями cpp, dat, з текстами програм, початкових даних для обробки і ін. Для створення нового файлу треба мати очищене вікно редактора. Його можна одержати зразу після входу в редактор або після створення попереднього файлу. Щоб викликати додаткове вікно редактора після створення і запам'ятовування файлу, треба:

- викликати меню опції File мишею або за допомогою команди Alt+F;
- виконати в ньому команду New введенням символу n або N.

В результаті буде сформовано вікно редактора і курсор знаходитиметься в першій позиції першого рядка екрану. Ім'я нового файлу, розташоване в розриві верхньої рамки вікна редактора, буде NONAME00.CPP. При записуванні файлу на МД система попросить перейменувати файл.

Для створення файлу треба ввести його текст на екран за допомогою клавіатури.

Текст, створений на екрані, можна зберегти з тим самим або з новим ім'ям. Для запам'ятовування нового файлу використовується команда F2(Save). На екрані з'явиться додаткове вікно, в якому потрібно вказати куди і з яким ім'ям записати файл. Наприклад: C:\LABRAB1.CPP. Потім потрібно натиснути клавішу Введення. Вміст екрану буде записаний на МД, і у верхній рамці вікна екрану з'явиться ім'я файлу, в який поміщений текст з редактора.

В процесі створення і корегування тексту потрібно періодично запам'ятовувати текст з ОП редактора у файл з тим самим ім'ям за допомогою клавіші F2. Якщо в процесі роботи з текстом відбувся збій ПК, текст в ОП редактора може бути пошкоджений. Але збережеться текст, записаний у файл на МД, звідки його можна викликати у вікно редактора.

Для створення копії розробленого файлу з іншим ім'ям використовується команда File/Save as.

Для виклику у вікно редактора тексту з файлу на МД можна використовувати команду File/Open(F3). Вона дозволяє завантажити з МД файл із списку в додатковому вікні Open a File. Якщо це текст програми, її можна компілювати і виконувати.

Спосіб завантаження файлу за допомогою F3 доцільно використовувати, якщо необхідний файл ще не викликався під час даного сеансу, тобто з ним ще не проводилася робота. Якщо файл вже відкривався, але вікно з цим файлом було закрите, можна використовувати команду Alt+0.

Вона викликає вікно Window List зі всіма відкритими і закритими вікнами. Серед закритих вікон треба вибрати вікно з необхідним файлом і відкрити його кнопкою ОК.

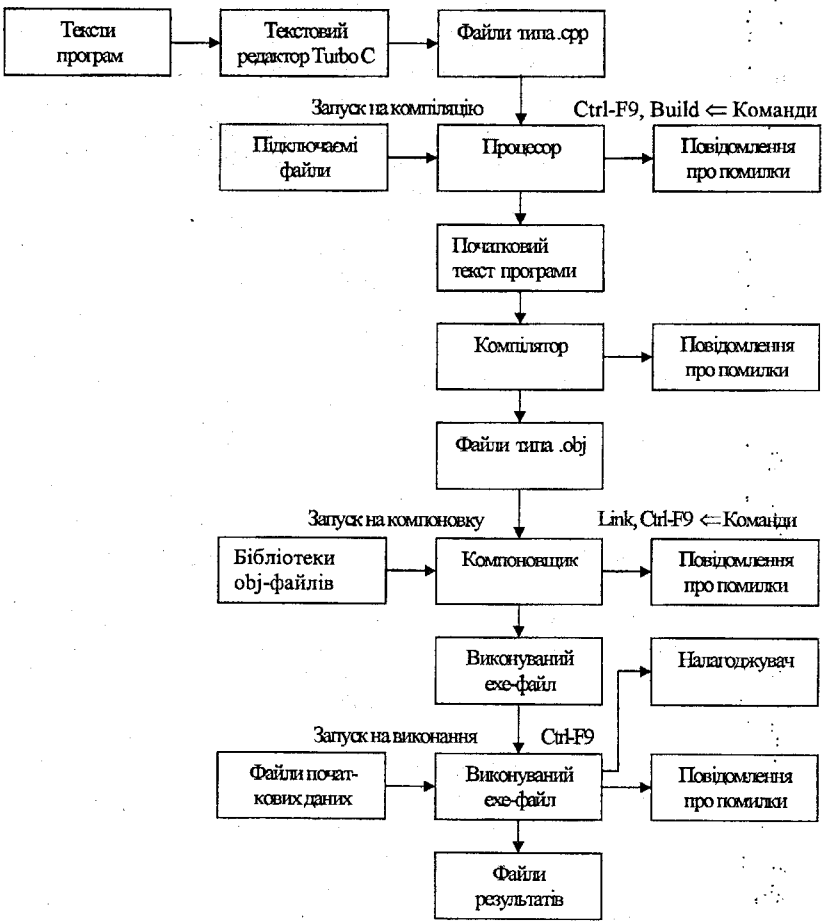


Рисунок 5 – Схема процесу розробки програм в середовищі Borland C++

Для первинного виклику текстів в редактор доцільно використовувати також команду Change dir для установки поточного каталога. Під час розробки програм можна вводити з нього файли командою F3 і запам'ятовувати командою F2. У цей каталог виводитимуться сформовані

файли типу .obj і типу .exe.

У вікнах редактора одночасно може бути відкрито до **дев'яти** файлів.

12.2 Препроцесор

Могутнім засобом, що дозволяє полегшити розробку програм, є препроцесор Borland C++. Він використовується для обробки тексту початкового файлу (початкової програми) на першому етапі компіляції. Препроцесор виконує директиви про виконання визначених дії до початку компіляції початкового тексту програми, зокрема підставляє всі зовнішні файли і розширює всі макровиклики. Директиви препроцесора – це команди керування трансляцією. Текст, оброблений препроцесором, який поступає на компіляцію, не містить директив препроцесора.

Директиви препроцесора звичайно використовуються для того, щоб полегшити модифікацію початкової програми і зробити її такою, що легко адаптується для застосування в різних реалізаціях компілятора Сі. Директиви дозволяють замінити лексеми (слова, вирази) в тексті програми іншими значеннями, вставити в початковий файл вміст іншого файлу і ін.

Склад і призначення директив препроцесора наведено в табл. 4.

Таблиця 4 – Директиви препроцесора Borland C++

Директиви	Призначення директиви
#define	Макроозначення
#undef	Відміна макроозначення
#include	Включення файлів
#if #ifdef #ifndef	Умовна підстановка фрагмента тексту
#elif #else	Альтернатива для #ifdef і #ifndef
#endif	Кінець тексту, що умовно підставляється
#line	Задає константу – номер наступного рядка файлу
#pragma	Задає вказівки компілятору
#error	Задає повідомлення про помилку на етапі компіляції

У одному рядку може бути одна директива. Деякі директиви можуть мати аргументи. Директиви можуть бути в будь-якому місці початкового файлу. Їх дія розповсюджується від місця їх розташування до кінця файлу. До і після символу # можуть бути пропуски. Наприклад є ідентичними директиви

#include <stdio.h>

include <stdio.h>

Найчастіше використовуються директиви #include і #define.

12.2.1 Включення файлів. Директива #include

Директива #include призначена для включення початкового файлу, ім'я якого задане в директиві, в поточний файл, в якому задана директива #include, в місце, визначене директивою.

Формат директиви:

#include <ім'я файлу>

#include "ім'я файлу"

де ім'я файлу – це ім'я фізичного файлу, якому може передувати ім'я пристрою і директорії, як це прийнято в DOS. Якщо в кутових дужках або в лапках задане повне ім'я файлу, то препроцесор шукає файл за заданим ім'ям маршруту, ігноруючи інші каталоги. Якщо ім'я файлу в директиві задане неповно і взято в кутові дужки (< і >), то пошук файлу відбувається в системних бібліотеках і в каталогах, визначених в рядку Options/Directories/Include directories. Якщо ім'я файлу взято в лапки ("), то препроцесор починає пошук файлу в каталозі, що містить файл з даною директивою #include, і потім у файлах, визначених в рядку Options/Directories/Include directories.

Включаємий файл також може містити директиви препроцесора і обробляється препроцесором так само, як якби він входив до складу початкового файлу. Таким чином, директива #include може бути вкладеною.

Наприклад, у файлі labrab.cpp є директива #include:

```
// Це початок файлу labrab.cpp
# include <stdio.h>
# include "defs.h"
```

При цьому в самий початок тексту поточного файлу labrab.cpp включається текст файлу <stdio.h>, ім'я якого задане в директиві #include. Нехай файл програми labrab.cpp розташований в підкаталозі C:\LABC. Пошук файлу <stdio.h> ведеться в системних бібліотеках і в списку директорій, визначеному в параметрах Options/Directories/Include directories. Пошук файлу defs.h ведеться серед початкових файлів програми, в каталозі і якщо його там немає, то використовується список директорій в рядку Options/Directories/Include directories.

Засіб #include – хороший спосіб зібрати в одному файлі глобальні оголошення змінних великої програми. Він гарантує, що всі початкові файли використовуватимуть одні і ті ж оголошення. Це спрощує корегування при змінах і адаптації програми.

12.2.2 Іменовані константи і макроозначення. Директива #define

Для визначення іменованих констант і макроозначень використовується директива #define, для їх відміни – директива #undef.

Директиви #define звичайно застосовують для заміни осмисленими

ідентифікаторами часто використовуваних в програмі констант, ключових слів, виразів, операторів і груп операторів. Іменовані константи і макроозначення директив можуть бути зібрані в одному файлі, що включається за допомогою директиви #include.

Форма директив:

- #define <ідентифікатор> <текст> – для іменованої константи і макроозначення без параметрів;
- #define <ідентифікатор> (<список-параметров>) <текст> — для макроозначення з параметрами;
- #undef <ідентифікатор> — для відміни визначення,

де ідентифікатор – ім'я іменованої константи або макроозначення; текст – значення іменованої константи або макроозначення.

Ідентифікатори іменованих констант в мові Сі прийнято записувати символами верхнього регістра (великими буквами), щоб відрізнити їх від імен змінних і функцій. Текст іменованої константи відділяється від його ідентифікатора одним або більше пропусками.

Директива #define іменованої константи замінює в тексті початкового файлу, розташованому після неї, ідентифікатор на текст, який іде за ідентифікатором. Цю заміну називають **макропідстановкою**.

Приклади іменованих констант:

```
#define  PI 3.1415926
#define  M 4
#define  N 6
#define  DL 80
#define DLL ( DL + 10 )
#define STUD struct data
```

У наведеному прикладі кожне входження в початковому тексті ідентифікатора PI буде замінене на 3.1415926, M – на 4, N – на 6, DL – на 80. Ідентифікатор DLL буде замінений на текст (DL+10), який після підстановки тексту для DL набуде вигляду: 80+10. Дужки у виразі потрібні, щоб не викликати помилок при підстановці. Наприклад, в програмі може бути: var=DLL*20;

Після обробки препроцесором цей оператор набуде вигляду:

```
var=(80+10) * 20;
```

Слово STUD в тексті програми буде замінене struct data.

Якщо текст не поміщається на одному рядку, він може бути продовжений на наступному. Для цього в кінці рядка повинен бути символ \ – продовження рядка. Наприклад:

```
#define MESS  "Спроба створити файл не вдалася через  брак місця\n"
```

У програмі може бути оператор printf(MESS); з його допомогою буде

надруковане повідомлення, визначене іменованою константою MESS.

Текст іменованої константи після ідентифікатора може бути опущений. В цьому випадку всі ідентифікатори будуть видалені з початкового тексту програми. Наприклад:

```
#define REG1 register
#define REG2
```

У тексті початкового файлу REG1 буде замінене на register, а всі слова REG2 будуть видалені.

Директива #undef відміняє поточне значення іменованої константи або макроозначення. Після відміни їм може бути поставлене у відповідність нове значення. Директиви #define і #undef можуть бути в будь-якому місці програми, зокрема в тексті функції.

Ідентифікатори, які відображають деяку послідовність дій, задану операторами або виразами мови Сі, називають **макроозначеннями**. Звертання до макроозначення з програми називають **макровикликом**. Макроозначення схоже на визначення функції. Але заміна виклику функцій макровикликом може підвищити швидкість виконання програми порівняно з викликом функцій, оскільки для макровиклику не вимагається виклику функції і заміни формальних параметрів фактичними. Наприклад:

```
#define A 2      // призначення A = 2;
#define PR printf ("A=%d\n",A)  // макроозначення без параметрів;
 main ()    PR;                 // макровиклик PR
 #undef A          // відміна призначення A;
 #define A  3     // нове призначення A = 3;
 PR;               // макровиклик  PR
```

Після обробки препроцесором текст програми набуде вигляду:

```
{ printf ("A = %d\n", A );    – виведення: A = 2
                              – для  #undef  A – відміна призначення  A;
   A = 3;              - для  #define A 3 – призначення A = 3;
   printf ("A = %d\n", A );        – виведення: A = 3
}
```

Макроозначення можуть бути з параметрами і без параметрів. У макроозначенні з параметрами за ідентифікатором йде в круглих дужках список формальних параметрів. Між ідентифікатором і лівою (відкриваючої) дужкою пропуски не допускаються. У списку параметрів може бути один або більше пропусків. Між закриваючою дужкою списку параметрів і початком тексту повинен бути пропуск. Список формальних параметрів містить один або більше ідентифікаторів, розділених комами. При виклику макроозначення замість його формальних параметрів будуть підставлені фактичні параметри. Але це чисто *текстова* підстановка, а не виклик функції із заміною

параметрів. Причому при виклику допускаються пропуски скрізь, у тому числі і після імені макроозначення перед відкриваючою круглою дужкою.

Приклад:

```
#define TWO 2                    // TWO – іменована константа
#define FOUR TWO * TWO  // – макроозначення без параметрів
#define PX printf  (" x рівний %d\n", x) //  "  "  "
#define FMT  " x, %d\n"          // іменована константа
#define MSG  "Це приклади директиви define "  //  "      "
main ( )
  { int x = TWO;
   PX;                  // виклик макроозначення PX
   x = FOUR;            //виклик макроозначення  FOUR
   printf(FMT, x );
   printf("%s\n", MSG);
  }
```

Після препроцесора оператори програми набудуть вигляду:

```
  { int x= 2;
  printf(" x дорівнює %d\n", x);
  x =  2*2;
  printf ("x = %d\n", x);
  printf ("%s\ n ", "Це приклади директиви define");
   }
```

Приклад макроозначень з параметрами:

```
#define SQUARE( x )   x * x
#define PR(x) printf("x дорівнює %d.\n",  x).
```

// Після препроцесора програма набуде вигляду:

```
    main( )
  { int x = 4, z;                          { int   x = 4, z ;
   z= SQUARE ( x );                          z = x * x;
   PR( z );                                  printf("x дорівнює %d.\n", z);
   z = SQUARE(z);                            z=z*z;
   PR(z);                                    printf("x дорівнює %d.\n", z);
    }                                      }
```

У прикладі:

– SQUARE(x) – макроозначення з параметром x;
– x * x – замінюючий його текст;
– PR(x) – макроозначення з параметром x;
– printf ("x дорівнює %d.\n", x) – замінюючий його текст.

Підстановка імен – фактичних параметрів функції в рядкові константи не

виконується. Тому в списку фактичних параметрів оператора printf в рядковій константі залишається ім'я x.

Вибір макроозначень або функцій можна виконувати, виходячи з того, що макроозначення призводять до збільшення необхідного об'єму пам'яті (для макропідстановок), а функції – до збільшення часу роботи програми (для виклику функцій замість макропідстановок).

Приклади макроозначень з параметрами:

```
#define MAX( X, Y)  ((X) > (Y) )? (X) : (Y)
#define MULT ( A, B )  ( (A) * (B) )
 main ( )
 { int r, a=1, b = 2,  c = 4, d = 0,  e = 1;
   r = MAX( 1, 2);
   r = MAX( a+ b, c - d );
   r = MULT(d + 2, e +3); }
```

В прикладі

макроозначення:	текст, що їх заміняє:
MAX(X, Y)	((X) > (Y))? (X) : (Y)
MULT(A, B)	((A) * (B))

Після обробки макророзширень препроцесором оператори набудуть вигляду:

```
r = (1) > (2) ) ? (1) : (2);
r = ( (a + b > (c - d) ) ? (a + b ) : (c – d);
r = ( (d+2 ) * (e+3) );
```

12.3 Компіляція програм

12.3.1 Встановлення параметрів середовища і режимів роботи компілятора

Розроблення програм в системі Borland C++ звичайно включає такі етапи:

- установлення параметрів середовища;
- завантаження програми або створення тексту програми за допомогою редактора Borland C++;
- створення проекту багатофайлової програми, якщо програма складається більше ніж з одного файлу;
- установка режимів роботи компілятора;
- компіляція програми і створення obj-файлу;
- компонування і створення exe-файлу програми, що розробляється;
- виконання програми.

Для установки і збереження параметрів середовища треба ввійти в меню Options, вибрати в ньому команду Environment (Середовище) і натиснути Введення. На екрані розвернеться додаткове меню, за допомогою якого

можна налагодити середовище. За допомогою команди Options/Directories можна визначити імена довідників зовнішнього оточення Borland C++: директорії включаємих файлів (Include Directories) і директорію бібліотечних файлів (Library Directories). Можна задати директорію вихідних файлів (Output Directory), якщо треба, щоб файли .pbj, .exe і .obj записувалися не в поточний каталог, а в інший, заданий користувачем. Зберегти параметри налагодження середовища можна за допомогою команди Options/Save. При виході з середовища його конфігурація буде записана у файл tcconfig.tc. Склад зберігаємих параметрів можна встановити командою Options/Save у вікні Save Options.

Встановити режими роботи компілятора можна за допомогою команди Options/Compiler. За цією командою розвертається меню встановлення параметрів компілятора. Команди цього меню встановлюють параметри компілятора, можна встановити параметри генерації об'єктного коду (команда Advanced code generation); зокрема:

- вибрати одну з можливих в Borland C++ моделей пам'яті (Model); за замовчуванням приймається модель Small (Мала);
- оптимізувати програму за розміром або швидкістю її виконання, використання регістрових змінних і реорганізацією циклів і варіантних операторів (команда Optimizations);
- визначити інтерпретацію початкового модуля компілятором; встановити довжину ідентифікатора Borland C++(команда Source, параметр Identifier Length); за замовчуванням довжина ідентифікатора дорівнює 32 символам;
- визначити, як компілятор повинен реагувати на виявлення в програмі помилок різних типів.

Команда Options/Compiler/Messages дозволяє встановити режими, що визначають умови видачі повідомлень про помилки, попередження і припинення компіляції.

На час відлагодження програми доцільно включити майже всі підпараметри таблиць Compiler/Messages, які визначають видачу попереджень і повідомлень про помилки.

12.3.2 Види компіляції програм

Система Borland C++ має 3 різні команди компіляції: Compile, Make і Build all. Вони відрізняються відношенням компілятора до об'єктних модулів, що входять до складу програми.

Команда Compile to OBJ (Alt+F9) компілює програму: з файлу .cpp або .c формується об'єктний модуль, файл типу .obj. Для компіляції береться файл, завантажений в редактор. Розширення файлів для компіляції

встановлюється на сторінці Options\Compiler\C++ Options; треба встановити в блоці Use C++ Compiler значення On для опції C++ always.

Команда Make (Створити exe-файл) формує виконувану програму, файл типу .exe. Для цього вона викликає компіляцію, потім компоновку програми за допомогою редактора зв'язків (Linker). Якщо в процесі компіляції зустрілося звертання до інших функцій, середовище перевіряє, чи були зроблені у файлі з початковим текстом цієї функції які-небудь зміни з моменту його останньої компіляції; якщо зміни були, виконується компіляція цієї функції і формування її obj-файлу. Проте, якщо cpp-файл функції не знайдений, система використовує існуючий obj-файл без контролю його змін.

Команда Build all (Створити) подібна опції Make. За таким виключенням: з її допомогою виконується пошук і перекомпіляція всіх початкових файлів незалежно від того, чи були в них зміни.

При відлагодженні однофайлових програм доцільно використовувати команду Compile(Alt+F9). Але можна використовувати і Make (F9). Команду Build all треба використовувати тільки при розробці багатофайлових програм.

Умовна компіляція дозволяє компілювати окремі частини програми тільки при виконанні умов, заданих директивами препроцесора. Цей засіб дозволяє створювати різні версії програм.

Переривання процесу компіляції програми може бути при виявленні заданої межі кількості помилок або попереджень на етапі компіляції або програмістом за допомогою команди Ctrl+Break. Кількість знайдених помилок і попереджень, при яких відбувається переривання компіляції, встановлюється і діалоговому вікні Compiler Messages.

12.3.3 Налагоджування програм на етапі компіляції

Налагоджування програм включає пошук і виправлення помилок програми. Borland C++ дозволяє одночасно проглядати повідомлення про помилки і редагувати текст програми, тобто виправляти помилки. Це полегшує процес пошуку і усунення помилок. Помилки в програмі можуть бути:

- синтаксичні (формальні), які знаходить компілятор;
- компоновки, пов'язані з взаємозв'язком програми і підпрограм;
- виконання: логічні, семантичні, введення-виведення.

На етапі компіляції можуть бути знайдені помилки при написанні операторів мови, наприклад:

- лишні символи в ключових словах і іменах змінних;
- пропущені символи;
- спроба використовувати неоголошені змінні;

- відсутність прототипу функції;
- невідповідність типів в списках формальних і фактичних параметрів.

В процесі компіляції на екрані з'являється додаткове вікно, в якому подане ім'я компільованої програми і інформація про файли, оброблювані компілятором, а в розриві верхньої рамки додаткового вікна — найменування процесу обробки: Compiling .

Під час компіляції у вікні повідомлень (Message) можуть накопичуватися повідомлення. Вони можуть бути трьох типів: інформаційні, попередження і про помилки. До складу інформаційного повідомлення входить ім'я процесу обробки програми і ім'я компільованого файлу.

Після закінчення компіляції в нижньому рядку додаткового вікна Compiling з'являється повідомлення про успішність компіляції:

Success : Press any key

В випадку виявлення попереджень або помилок в нижньому рядку цього додаткового вікна з'являється відповідне повідомлення:

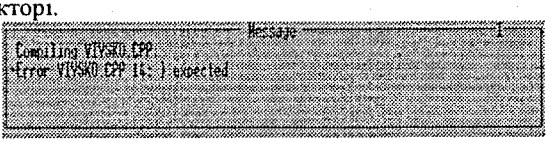
Warnings : Press any key

або: Errors : Press any key ;

Якщо після успішної компіляції натиснути будь-яку клавішу, наприклад Esc або Введення, здійснюється перехід у вікно редактора. Вікно повідомлень при цьому не формується.

Приклад вікна Message з повідомленням про результати компіляції поданий на рис. 6, де 14 – номер рядка файлу, в якому розташовані оператори, що викликали попередження і помилку.

Якщо при компіляції знайдені помилки і(або) попередження, то після закінчення компіляції натиснення клавіші Esc викликає перехід у вікно редактора. Якщо натиснути Введення, активним стає вікно Message, розташоване в нижній частині екрану, підсвічується перше попередження у вікні повідомлень і рядок у вікні редактора, оператор якого викликав це повідомлення. Повідомлення відноситься до файлу, що знаходиться в даний момент в редакторі.



F1Help Space Vien sourse Edit sourse F10 Menu

Рисунок 6 – Приклад вікна повідомлень про результати компіляції

Вікно Message містить список повідомлень компілятора про помилки і попередження, знайдені в програмі, в послідовності їх виявлення. Список повідомлень вікна Message використовується для пошуку помилок в

програмі. При видачі повідомлень кожне повідомлення у вікні Message займає один рядок вікна повідомлень і, якщо повідомлення не вміщається у видиму частину вікна повідомлень, його можна вводити у вікно за допомогою клавіш управління курсором ← і →. При активному вікні повідомлень за допомогою F5 можна розкрити вікно повідомлення на весь екран і побачити більшу кількість повідомлень. Але для роботи з помилками краще працювати з двома вікнами: редактора і повідомлень.

Якщо на екрані розташовані вікно редактора і вікно повідомлень й вікно повідомлень активне, то за допомогою клавіш керування положенням курсора ↑ і ↓ можна проглядати (прокручувати, відстежувати) послідовно всю інформацію вікна повідомлень. При переміщенні підсвічування повідомлень вікна Message у вікні редактора синхронно підсвічуються рядки програми, що викликали помилку. У разі потреби в редактор може бути завантажений і інший файл (наприклад, включений в програму директивою #include), в якому знайдено помилку.

Якщо у вікні повідомлень при підсвіченому попередженні або повідомленні про помилку натиснути Введення:

- активним стає вікно редактора;
- у нижньому рядку вікна редактора з'являється повідомлення про помилку або попередження;
- курсор встановлюється під оператором програми, який викликав повідомлення;
- у вікні повідомлень рядок, який підсвічувався перед натисненням Введення, залишається поміченим яскравою крапкою зліва.

При підсвічуванні повідомлення про помилку або попередження у вікні повідомлень натиснення F1 викликає підказку (пояснення) про помилку. Погасити її можна клавішею Esc, командою Alt+F3 або закриттям вікна.

Натиснення F1 з вікна повідомлень за відсутності повідомлень про помилки або при підсвічуванні інформаційного повідомлення, наприклад про ім'я скомпільованого файлу, викликає на екран інформацію про призначення вікна повідомлень. Повторне натиснення F1 викликає на екран Index – зміст системи допомоги.

Основні повідомлення про помилки на етапі компіляції наведені в додатках посібника "Лабораторний практикум для вивчення мови Сі, част. 2" цих же авторів.

Корегування синтаксичних помилок

Для пошуку і виправлення помилок треба курсор у вікні повідомлень помістити на перше повідомлення і натиснути Введення. Курсор переміститься у вікно редактора під оператор, що викликав повідомлення про помилку. Можна виправити помилку.

Якщо повідомлень про помилки декілька і треба продовжити їх виправлення, можна повернутися у вікно повідомлень, вибрати наступне повідомлення і т.д.

Але можна не повертатися у вікно повідомлень для вибору наступного повідомлення, а ввести команду Alt + F8 (Next error – наступна помилка), і редактор помістить курсор в рядок з наступною помилкою, а у вікні повідомлень рядок з повідомленням про наступну помилку буде зліва помічений точкою, що світиться.

До попередньої помилки можна повернутися командою Alt+F7 (Previous error – попередня помилка).

Одна помилка може викликати безліч повідомлень. В цьому випадку виправлення даної помилки не вимагає переглядання інших повідомлень про неї: їх треба пропустити і вибрати повідомлення про наступну помилку.

В процесі коректування помилок можна видаляти і додавати рядки в початковий текст програми. При висвіченні помилкового оператора редактор це враховує.

Не завжди оператор, в якому знайдена помилка, є помилковим. Наприклад; неправильно оголошена змінна викличе повідомлення про помилку у всіх операторах, що використовують цю змінну.

Одна із переваг Borland C++ – можливість роздільної компіляції програмних файлів однієї багатофайлової програми. Їх можна об'єднати в одну програму за допомогою директив #include або за допомогою засобів проекту.

12.4 Компонування програм

12.4.1 Команди компонування програм

Компонування програм виконує компонувальник (Linker) після успішної компіляції або після компіляції, в процесі якої знайдені тільки попередження.

Компонування програм включає пошук необхідних об'єктних модулів і формування їх взаємозв'язків (дозвіл посилань). Результатом компонування є exe-програма, тобто програма, готова до виконання. Для запуску такої програми треба вибрати її за допомогою клавіш управління курсором і натиснути клавішу Введення. Всі налагоджені програми багаторазового використання доцільно зберігати у вигляді exe-файлів.

Компонування програми може бути викликана командами меню Compile і Run: Link, Make(F9), Bulid All. А також гарячими клавішами:

- F4 – перше виконання програми до курсора;
- F7 – перше порядкове виконання програми;
- F8 – те ж з виконанням функцій за одне натиснення клавіші

Кожна з названих команд за допомогою компонувальника створює виконувану програму, файл типу .exe. Роботою компонувальника керують опції, які можна задати за допомогою команд меню Options/Linker.

Команда Compile/Link (компонування і створення exe-файлу) створює з об'єктних модулів (файлів типу .obj) виконувану програму (файл типу .exe) і записує її у файл. За командою Link завжди використовуються поточні версії obj - файлів – об'єктних модулів, без відстежування їх відповідності файлам початкового тексту.

Команди Make і Build All викликають компонувальник після виконання компіляції, тобто формування об'єктних модулів.

Команди F4, F7 або F8, введені після компіляції програми, викликають її компонування і виконання програми до курсора (F4) або до першого рядка з виконуваними операторами (F7, F8). Команди F4, F7 або F8, введені до компіляції програми, викликають її компіляцію, компонування і виконання до курсора або до першого рядка програми.

12.4.2 Налагодження програми на етапі компонування

В процесі компонування на екрані з'являється додаткове вікно, в якому знаходиться ім'я exe-програми, формованої компонувальником, і підсумки компонування, а в розриві верхньої рамки додаткового вікна – найменування процесу обробки: Linking.

Під час компонування у вікні повідомлень можуть накопичуватися повідомлення про помилки, знайдені під час компонування. Після закінчення компонування в нижньому рядку додаткового вікна з'являється повідомлення про успішність компонування.

Якщо після успішного компонування натиснути клавішу Esc або Введення, здійснюється перехід у вікно редактора. Вікно повідомлень при цьому не формується.

Вікно повідомлень (Message) формується, якщо після компіляції було хоча б одне попередження. Після успішного компонування за командою Link у вікні повідомлень з'являється рядок з повідомленням, наприклад, у вигляді Linking LR1.EXE:

Приклад вікна з повідомленням про результати компіляцій і компонування поданий на рис.7

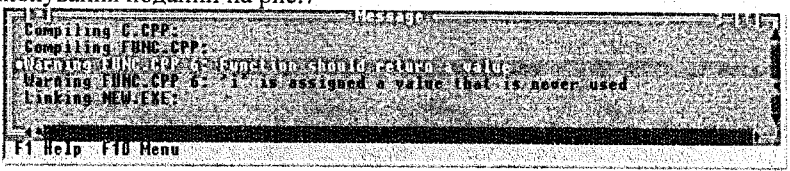


Рисунок 7 – Вигляд вікна з повідомленнями про результати компіляції і компонування

Після виконання компіляції і компонування за допомогою команд Make (F9) або Build All у вікні повідомлень видаються повідомлення, аналогічні повідомленням про результати компіляції і компонування.

Якщо і в процесі компонування знайдені помилки або попередження, в нижньому рядку додаткового вікна видається повідомлення:

Warnings : Press any key

Або Errors : Press any key

Якщо після цього натиснути клавішу Esc, здійснюється перехід у вікно редактора. Нижню частину екрану займає вікно повідомлень (Message). Воно містить список повідомлень компонувальника про помилки, знайдені в програмі, в послідовності їх виявлення. Список повідомлень вікна Message використовується для аналізування і пошуку помилок в програмі. Якщо після появи повідомлення про результати компонування натиснути Введення, активним стає вікно повідомлень, підсвічується перше повідомлення про помилку. За кожною помилкою, знайденою компонувальником, в вікні повідомленнь формується повідомлення у вигляді:Linker Error:текст-повідомлення-про-помилку.

Приклад вікна повідомлень з повідомленням про помилку, знайдену компонувальником, поданий на рис. 8.

Рисунок 8 – Приклад вікна повідомлень з помилкою, знайденою компонувальником

За допомогою клавіш керування положенням курсора можна послідовно переглядати повідомлення у вікні повідомлень. Підсвічування у вікні редактора операторів, в яких компонувальником знайдені помилки, не виконується. Після закінчення переглядання повідомлень про помилки компонувальника можна перейти у вікно редактора. За допомогою F5, знаходячись у вікні Message, можна розкрити вікно повідомлень на весь екран. Основні повідомлення про помилки на етапі компонування подані в посібнику авторів Круподьорової Л.М, Пєтуха А.М. "Лабораторний практикум для вивчення мови Сі".

12.5 Налагодження програм на етапі виконання програми

12.5.1 Основні поняття

Запуск на виконання успішно скомпільованої програми з середовища Borland C++ виконується командою Ctrl+F9. Проте виконання програми може не дати правильних результатів. Це означає, що в програмі за

відсутності формальних, синтаксичних помилок є, наприклад, семантичні, смислові помилки. Вони можуть бути через неправильно поставлену або неправильно зрозумілу задачу, помилки в алгоритмі або при невідповідності тексту програми її алгоритму.

Прояв помилок програми на етапі її виконання може бути у вигляді:

- немає результатів;
- передчасна зупинка програми; може бути частина результатів;
- результати неправильні;
- програма некоректна при звертанні до файлів;
- зациклення;
- "зависання" системи: ПК не реагує ні на які клавіші і команди.

У ряді випадків може бути повідомлення системи про знайдені помилки програми, локалізовані системою. Виявлення і коректування помилок етапу виконання програми – один з найскладніших і трудомістких процесів налагодження програм. Це пов'язано і з тим, що поява помилки на етапі виконання програми припиняє її виконання, але оператор, що викликав помилку, не локалізується системою. Місце виникнення помилки може бути визначене програмістом в процесі трасування програми.

Перелік помилок Run-time errors – етапу виконання поданий в посібнику авторів Круподьорової Л.М, Пєтуха А.М. "Лабораторний практикум для вивчення мови Сі".

Для успішного налагодження програм потрібно:

- чітко уявляти собі задачу від її постановки до реалізації за допомогою розроблених алгоритму і програми;
- за можливості локалізувати частину програми, в якій виявляється помилка; для цього:

  а) при появі помилки визначити оператор програми, починаючи з якого програма виконується неправильно (немає результатів або вони неправильні);

  б) починаючи з цього місця програми візуально контролювати правильність результатів (значень виразів і змінних) виконання програми в процесі її трасування.

Для контролю правильності виконання програми за допомогою засобів налагодження можна використовувати таку послідовність дій:

- підготувати систему до налагодження;
- встановити у вікно перегляду (Watch) вирази і змінні, які треба контролювати в процесі трасування;
- виконати трасування програми;
- після кожного етапу трасування перевіряти правильність поточних значень змінних програми.

12.5.2 Підготовка системи до налагодження програми

У складі Borland C++ є інтегрований налагоджувач. Він є складовою частиною середовища Borland C++ і має в основному меню підменю Options, за допомогою якого можна побудувати систему налагодження, і підменю Debug та Run, за допомогою яких можна виконати налагодження програми. За допомогою налагоджувача можна використати такі засоби:

- трасування програми;
- переглядання зміни значень контрольованих змінних і виразів в процесі трасування;
- модифікацію значень змінних для тестування програми;
- переглядання значень фактичних параметрів викликаних функцій;
- роботу з точками переривання;
- переглядання вихідної інформації програми.

Перш ніж почати налагодження програми за допомогою налагоджувача, треба налагодити середовище відповідним чином, а саме: встановити в стан On: параметр Options/Debugger/Source Debugging, тобто включити засоби налагодження на рівні початкового тексту програми. Зазвичай вищезгадуваний параметр за замовчуванням встановлений в стан On.

Після налагодження середовища потрібно задати вирази і імена змінних (за допомогою (Ctrl+F7), значення яких контролюються у вікні перегляду (Watch). Потім – трасувати, тобто виконувати програми перевіряти поточні значення заданих виразів, змінних і(або) результати роботи програми у вікнах перегляду (Watch) та користувача (у файлі stdout – на екрані). У вікні перегляду можуть відображатися значення змінних і виразів всіх типів: від цілих до структур і файлів.

Майже всі способи трасування програми і переглядання поточних значень контрольованих змінних можна виконати за допомогою функціональних клавіш. Початок трасування програми виконується натисненням клавіші, відповідної вибраному типу її виконання (F4, F7, F8). Завершити налагоджувальне трасування програми можна за допомогою команди Ctrl+F2.

12.5.3 Установлення, видалення і переглядання поточних значень

Для того, щоб контролювати поточні значень виразів і змінних програми, треба визначити їх склад, виходячи із значення і алгоритму обробки даних. А після зупинки програми переглядати ці значення в одному з вікон.

Щоб задати (додати або видалити) імена змінних і виразів, поточні значення яких треба спостерігати у вікні перегляду, можна використовувати команди:

- Debug/Watches/Add watch або Ctrl + F7 – для додання імен;
- Debug/Watches/Delete watch або клавішу Del – для видалення

імен і виразів з вікна Watch;

- Debug/Watches/Edit watch – для редагування імен змінних і виразів у вікні перегляду;
- Debug/Watches/Remove all watches – для видалення всіх виразів, зокрема імен змінних, з вікна перегляду.

Найпростіше і швидко можна встановити ім'я контрольованої змінної у вікно перегляду (Watch) одним з таких способів:

- підвести курсор під ім'я змінної або вираз і ввести команду Ctrl+F7; копіювати вираз у вікно перегляду можна за допомогою клавіші →;
- встановити курсор під який-небудь пропуск; ввести команду Ctrl+F7; на екрані з'явиться додаткове вікно; у ньому треба написати необхідне ім'я або вираз і натиснути Введення: вміст додаткового вікна буде введений у вікно перегляду.

За одне виконання команди Ctrl+F7 можна ввести тільки одне ім'я або вираз. Якщо треба ввести ряд імен, команду Ctrl+F7 треба виконати необхідну кількість раз. Якщо під час виконання команди Ctrl+F7 змінна мала певне значення, воно буде підсвічене у вікні перегляду у вигляді: ім'я: значення; якщо вона не мала ніякого значення, то у вікні перегляду поряд з ім'ям буде підсвічене Undefined symbol 'X'.

Наприклад:

```
a: Undefined symbol 'a'
b: 2.0
c: ' D'
d: "ABCDE"
```

Після зупинки програми у вікно перегляду можуть бути виведені будь-які типи значень. При цьому символи (char) виводяться в апострофах ('), рядки в лапках ("). Значення елементів масивів і структур – у фігурних дужках у вигляді списку значень, розділених комами. Наприклад, елементи масиву a:

```
a: { 1, 2, -3, 4}
```

Для текстових файлів виводиться список значень його елементів у фігурних дужках. Наприклад:

```
fid: DS:056A  – покажчик на тип FILE
*fid: = {службова інформація "63001970ТУ-134 800.\n6302 ...
```

Для файлів, що складаються із записів, для контролю у вікно перегляду виводиться у фігурних дужках список значень записів; значення кожного запису в лапках ("). Наприклад:

```
fr: DS:055A  – покажчик на тип FILE
* fr:  {службова інформація "1-й запис", "2-й запис",
```

Для виключення змінної з вікна перегляду треба:

- перейти у вікно перегляду;
- вибрати (підсвітити) за допомогою клавіш керування переміщенням курсора рядок з необхідним ім'ям і натиснути клавішу Del.

Для видалення з вікна переглядання всіх виразів потрібно скористатися командою Debug/Watches/Remove all watches.

Установку і видалення імен з вікна перегляду можна виконувати як до початку, так і під час трасування програми, тобто під час чергової зупинки її виконання для контролю поточних значень.

При кожній зупинці програми в процесі її трасування можна:

- контролювати поточні значення виразів у вікні перегляду і у вікні результатів (за допомогою команди Alt+F5);
- додавати вираз у вікно перегляду (Ctrl+F7,Ins) і видаляти їх з вікна (клавіша Del);
- при необхідності змінювати поточні значення змінних за допомогою Debug/Evaluate/modify (Ctrl+F4).

Якщо встановлене вікно редагування з вікном перегляду, можна переводити курсор з вікна редагування у вікно перегляду і назад за допомогою миші або клавіші F6. Для редагування виразу, що "підсвічується" у вікні перегляду, треба натиснути клавішу Введення.

12.5.4 Трасування програми

Трасування програм – це реєстрація програмних подій в послідовності їх виконання. Вона використовується для поетапного (почастинного) виконання програми з метою аналізування результатів роботи програми після закінчення кожного етапу.

Трасування програми може бути у вигляді:

- порядкового виконання програми:
  а) з порядковим виконанням функцій (F7);
  б) з виконанням функцій від одного натиснення клавіші F8;
- виконання програми до рядка, в якому встановлений курсор (F4);
- виконання програми по частинах, за кожною командою Ctrl + F9 – до рядка, в якому встановлена чергова контрольна точка.

Для того, щоб почати трасування програми, необхідно:

- заздалегідь виконати її компіляцію і компоновку, наприклад, за допомогою команди F9;
- виконати компіляцію і компоновку за допомогою команд трасування, наприклад F4 або F7, після чого продовжити трасування,

Команди F4, F7, F8 і Ctrl+F9 можна використовувати як окремо (тільки F7 або тільки F8), так і разом – в міру необхідності: то F7, то F4 і т.д.

Від одного натиснення клавіш F7 або F8 виконуються:

- декілька операторів, розташованих в одному рядку;
- один оператор, якщо він розташований в декількох рядках.

Для виконання програми до курсора треба заздалегідь встановити курсор в тому рядку, до якого треба виконати програму, і натиснути клавішу F4.

Контрольними точками зупинки названі точки, в яких повинна відбутися зупинка програми. Задати точки зупинки в програмі найпростіше за допомогою команди Ctrl+F8. Для цього треба встановити курсор в рядок, перед виконанням якого повинна відбутися зупинка і виконати команду Ctrl+F8. Як тільки на рядку встановлена точка зупинки, рядок підсвічується, наприклад, червоним кольором. У рядку, визначеному як точка зупинки, повинен бути хоча б один виконуваний оператор. Це не може бути порожній рядок, коментарі, опис змінної або оператор заголовка функції.

Після установки контрольних точок програму треба відкомпілювати і зкомпонувати (наприклад, за допомогою F9), а потім почати її виконання. Після досягнення контрольної точки виконання програми припиниться. Оператори рядка з точкою зупинки не будуть виконані. Рядок, на якому відбулася зупинка, буде підсвічений. Після зупинки програми можна працювати з вікнами для візуального контролю значень виразів. Після контролю можна продовжити виконання програми за допомогою клавіш F4, F7, F8 або за допомогою команди Ctrl+F9 – для її виконання до наступної точки зупинки або до кінця програми.

Щоб відмінити точку зупинки, треба підвести курсор до рядка з підсвіченою точкою зупинки і виконати команду Ctrl+F8. Команда Ctrl+F8 включає і відключає точки зупинки при повторному виконанні.

За допомогою Ctrl+Break можна перервати виконання програми у будь-який момент її виконання (наприклад, при зацикленні).

Після кожної зупинки програми можна аналізувати значення контрольованих змінних, переглядаючи їх:

- у вікні перегляду (Watch);
- у вікні обчислення виразів (команда Debug/Evaluate/modify);
- на екрані виведення результатів програмою (файл stdout).

Такий перегляд може полегшити розуміння того, що робить програма. Під час зупинки виконання програми можна не тільки контролювати значення змінних, але і змінювати їх значення для перевірки різних варіантів ходу виконання програми.

Якщо в результаті переглядання значень знайдено помилку, можна завершити налагоджувальне виконання програми за допомогою команди Ctrl+F2 і виправити помилку. Після внесення змін в текст програми її потрібно перекомпілювати (Alt+F9 або F9) і знову виконати трасування

програми або виконати її повністю за допомогою команди Ctrl+F9.

12.5.5 Робота з функціями

Всі функції, які викликаються з програми, повинні мати прототип оголошення і текст функції. Оголошення може бути у файлі основної програми або в будь-якому підключуваному файлі, наприклад за допомогою директиви препроцесора #include. Наприклад, якщо не дати директиву #include <math.h> і включити в програму оператор з викликом функції sqrt з системної бібліотеки: b = sqrt(a);
після компіляції програми з'явиться повідомлення

Compiling LR3.C:
• Warning LR3.C 41: Call to function 'sqrt' with no prototype

Для того, щоб взнати ім'я головного файлу, в якому знаходиться прототип названої функції (наприклад, sqrt), треба одержати по ній підказку. Для цього треба в тексті програми підвести курсор під ім'я цієї функції і ввести команду Ctrl+F1. На екрані з'явиться підказка з форматом виклику функцій і її призначенням та *з ім'ям головного файлу*, в якому знаходиться прототип названої функції. Залишається тільки помістити у файл з програмою, що містить виклик функції (наприклад, sqrt), яка не має прототипу, директиву з підключенням головного файлу, в якому міститься її прототип. Наприклад:

#include <math.h>

Після повторної трансляції попередження про відсутність прототипу функції sqrt не з'явиться.

При кожному виклику функції Borland C++ запам'ятовує виклик і передавані їй фактичні параметри в стеку виклику. Кожного разу під час чергової зупинки програми при її трасуванні можна дати запити стека виклику за допомогою команди Ctrl+F3–опцій меню Debug/Call stack (Налагодження/Виклик стека). При цьому в додаткове вікно виводиться список функцій, викликаних для виконання до даного моменту із списками значень фактичних параметрів. Список функцій дається в послідовності, зворотній їх виклику, тобто функція, викликана першою, буде в нижньому рядку, а викликана останньою – у верхньому рядку списку.

При виході з функції інформація про неї видаляється із стека. Отже, для того, щоб бачити фактичні параметри функції, треба встановити точку зупинки або зупинитися іншим способом після входу у функцію.

Якщо в списку стека вибрати один з його рядків і натиснути клавішу Введення, на екран буде викликаний початковий файл з текстом викликаної функції; відбудеться автоматичний перехід у вікно редактора, причому курсор буде встановлений на рядок тексту цієї функції, оператори якої виконувалися в ній останніми.

Переглядати значення фактичних параметрів можна також при використанні рекурсивних функцій.

Під час припинення виконання програми можна викликати на екран текст визначення будь-якої активізованої функції користувача, за допомогою пункту меню Search/Locate function (Пошук/Місцезнаходження функції). Після виклику цього пункту меню на екрані з'являється додаткове вікно для введення імені функції (рис. 9).

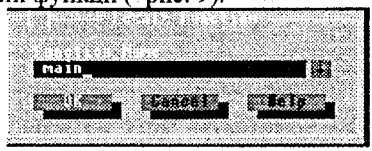


Рисунок 9 – Приклад вікна для пошуку місцезнаходження функції

Після набору імені необхідної функції і натиснення клавіші Введення в редактор завантажується файл з текстом визначення заданої функції і(або) активізується вікно редактора з цією функцією, причому курсор встановлюється на перший рядок визначення функції (на заголовок). У такий спосіб можна шукати визначення функцій тільки після початку виконання програми, наприклад, після першого виконання команди F7.

12.6 Розробка багатофайлових програм

12.6.1 Багатофайлові програми. Основні поняття

Одне з достоїнств Borland C++ полягає в можливості розробки багатомодульних і багатофайлових програм. Під модулем розумітимемо одну з функцій програми. Програма, що складається з ряду модулів, називається **багатомодульною**. Якщо програма складається з великої кількості функцій, їх можна згрупувати за призначенням в ряд файлів типу .cpp з початковими текстами цих функцій. У кожному файлі може бути ряд функцій. Причому компіляцію функцій цих файлів (команда Alt+F9) можна виконувати окремо. А об'єднати ряд програмних файлів в одну програму можна за допомогою директив #include або за допомогою засобів проекту програми. При цьому розробляється багатофайлова програма, тобто виконувана програма, до складу якої входять модулі (функції) з ряду програмних файлів.

До складу багатофайлової програми можна включати файли різних типів, зокрема з розширеннями:

- .h – хедер-файли, з глобальними змінними і макроозначеннями;
- .cpp – програмні файли; серед них повинен бути файл з головною функцією (main) і можуть бути файли з іншими функціями програми або файли з оголошеннями типів, глобальних змінних і констант;

- .obj – об'єктні модулі;
- .lib – бібліотеки об'єктних модулів, стандартних і призначених для користувача.

Хедер-файл може включати:

- прототипи (оголошення) функцій;
- описи глобальних змінних;
- описи типів і макроозначень;
- інші файли (директиви #include), що містять описи, необхідні для трансляції даної програми.

12.6.2 Розробка багатофайлових програм за допомогою директиви # include

Ряд файлів з текстами програм можна підключити до файлу з основною функцією (main) за допомогою #include – директив препроцесора.

Приклад визначення багатофайлової програми за допомогою директив:

```
// Файл MYMAIN.C – з текстом головної функції
#include“myglob.h” – підключення файлу з глобальними змінними;
#include“myfunc.cpp” – підключення файлу з функціями програми.
 main ( ) { . . . текст головної функції . . . )
```

Тексти програмних файлів можна компілювати окремо. Компонування багатофайлової програми (команда Link) виконується тільки з активного вікна з файлом, що містить функцію main. З цього ж вікна можна виконати команду Make – компіляції і компонування програми. При виконанні команди Make без попередньої компіляції файлів, що підключаються, виконується їх компіляція, а потім компонування програми. Після чого програму можна запустити на виконання.

Приклад програми з двох файлів:

```
// Багатофайлова програма, розроблена за допомогою #include.
// Файл extfun.cpp – з функціями, що викликаються
#inclule <stdio.h>
// ----------Функція виведення меню на екран ------------
void menu ( )
{ const char *menu =
     { "МЕНЮ ДЛЯ ВИБОРУ ВИГЛЯДУ ОБРОБКИ:\n"
          "1 - читання файлу \n"
          "2 - доповнення файлу \n"
          "3 - видалення записів \n"
          "4 - корегування записів \n"};
puts(menu); }
```

```
// Функції, що викликаються за допомогою масиву імен функцій:
    void cht( ) { printf("\n читання файлу"); }
    void dop( ) { printf("\n доповнення файлу"); }
    void ud( ) { printf ("\n видалення записів "); }
    void kor( ) { printf("\n корегування записів"); }
/* Файл mainincl.cpp – з функцією main.
   Основна програма для:
  – підключення файлу з викликаними функціями за допомогою #include;
  – використання меню і масиву покажчиків на функції.*/
# include <conio.h>
# include <stdio.h>
# include "extfun.cpp" // – підключення файлу з функціями
# include <stdlib.h>
     // оголошення функцій:
 void cht( ), dop( ), ud( ), kor( ), menu( );
 // масив імен функцій:
 void ( * obr[ ]) ( ) = { cht, dop, ud, kor };
     void main( )
   { int i;                    clrscr( );
   menu( );//–виклик функції для виведення меню
   while ( 1 ) // – "вічний" цикл
   { printf("\n введіть номер запиту: ");
       scanf ("%d",&i);
       if (i >= 1 && i<= 4 )
           // виклик функції за допомогою масиву функцій:
       obr [i-1] ( );
       else exit(0);
 } }
```

Попередня програма складається з двох файлів: extfun.cpp і mainincl.cpp.

У файлі extfun.cpp розташовані функції:

– menu – для виведення меню на екран;

– cht, dop, ud, kor – викликаємо функції без параметрів.

У файлі mainincl.cpp розташовані:

а) #include – директиви препроцесора для підключення:

– <conio.h>, <stdio.h>, <stdlib.h> – головних файлів Сі++;

– extfun.cpp – файлу з функціями розробника програми;

б) main – основна функція, з якої викликаються функції menu, cht, dop, ud, kor, розташовані у файлі extfun.cpp.

12.6.3 Розробка проектів багатофайлових програм

Багатофайлова програма може бути розроблена за допомогою команд з меню Project і Window.

Формування проекту багатофайлової програми може включати:

- визначення імені проекту;
- підключення до проекту програмних файлів;
- роздільну компіляцію підключених файлів; в результаті компіляцій формуються їх файли типу .obj;
- компоновку багатофайлової програми за допомогою команди Link; в результаті компонування формується виконуваний exe-файл багатофайлової програми, наприклад, з ім'ям prl.exe;
- групову компіляцію і компоновку програми з допомогою, команди Make; формується виконуваний exe-файл;
- виконання багатофайлової програми за допомогою команди Ctrl+F9; після виконання багатофайлової програми формується файл типу .dsk;
- формування файлу проекту; наприклад з ім'ям prl.prj, за допомогою команди Options/Save.

Замість роздільної компіляції і компонування програми так само, як і при формуванні однофайлової програми, можна використовувати команду Make (F9) або просто команду F9. В результаті також будуть сформовані файли типу .obj, .exe і .dsk. Файл типу .exe, наприклад prl.exe, можна запустити на виконання поза середовищем Borland C++, з середовища DOS.

Для подальшого запуску багатофайлової програми з середовища достатньо зберегти файл проекту і відкрити його в середовищі Borland C++.

Розглянемо процес формування проекту багатофайлової програми за допомогою команд середовища Borland C++.

Для формування проекту багатофайлової програми в середовищі треба виконати команду Project/Open project. При цьому на екрані з'являється діалоговий блок для вибору або введення нового імені файлу проекту. За допомогою цього блоку можна:

- почати формування нового проекту;
- відкрити раніше сформований проект.

Для початку формування нового проекту треба визначити ім'я нового проекту. Воно повинно мати розширення prj. Це розширення заготовлене у вікні введення імені. Його треба доповнити основною частиною імені проекту, наприклад, у вигляді prl.prj. Основна частина імені (до розширення) використовується для формування імені файлу виконуваної програми з розширенням .exe. Наприклад, якщо ввести ім'я prl.prj, то після формування виконуваної програми (.exe-файлу) вона матиме ім'я prl.exe.

Після визначення імені нового проекту в нижній частині екрану формується вікно проекту із заголовком Project:PR1 (рис. 10).

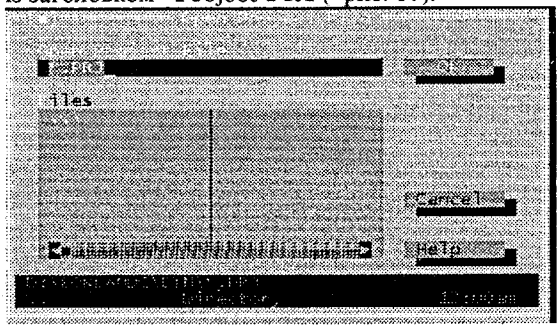


Рисунок 10 – Вид діалогового блоку для формування імені

В процесі формування проекту багатофайлової програми до складу проекту можна включити ряд програмних файлів. Це реалізується за допомогою команди Project/Add item. За цією командою відкривається діалоговий блок для вибору файлів проекту із списку файлів. Файл треба вибрати і натиснути кнопку Add. Цим способом до складу проекту можна додати один і більше файлів. Після включення всіх необхідних файлів до складу проекту треба натиснути кнопку Done (Виконано).

В результаті визначення складу проекту вікно проекту в нижній частині екрану містить список файлів проекту (рис. 11) В кожному рядку вікна представлена інформація про один файл. Для компіляції файлів у вікні проекту треба вибрати рядок з ім'ям цього файлу і виконати команду Alt+F9 – компілювати. У рядку файлу з'являється інформація про результат компіляції. Після компіляції можна виконати команду Link, внаслідок чого формується виконуваний exe-файл. Можна виконати команду Ctrl+F9 - виконати програму. В результаті формується dsk-файл.

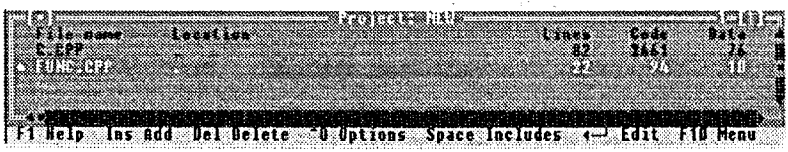


Рисунок 11 – Приклад вікна проекту PR1

Замість компіляції і компоновки можна виконати команду Make, яка компілює файли проекту і формує виконуваний exe-файл. Для збереження файлу проекту (наприклад, з ім'ям prl.prj) використовується команда

Options/Save зі встановленим в стан On параметром Project. Аналогічно встановлюється параметр збереження робочого столу параметром Desktop.

Зберегти файл проекту можна також за допомогою команди Window/Project notes (Примітки проекту). За цією командою формується і відкривається вікно Project notes для редагування тексту, в яке можна ввести довільний текст, наприклад, з рекомендаціями про використання проекту. Після відкриття цього вікна можна зберегти сформований проект за допомогою команди Save (F2), навіть якщо у вікно Project notes не було введено текст, тобто воно залишилося порожнім. Файл проекту матиме ім'я, введене при формуванні нового проекту, наприклад prl.prj.

Надалі, після завершення сеансу роботи в середовищі, на початку наступного сеансу роботи для запуску сформованого проекту достатньо відкрити файл проекту, щоб продовжити роботу над ним. Таким чином, файл проекту містить інформацію про склад проекту і може містити примітки з рекомендаціями по його використовуванню.

Щоб запустити раніше сформований проект, треба виконати команду Project/Open project. У діалоговому вікні треба вибрати ім'я сформованого файлу проекту. Для цього у вікні введення задана маска імені, наприклад C:\TC*.prj. Треба натиснути Введення, з'явиться вікно із списком файлів з розширенням .prj. У ньому вибрати ім'я файлу проекту, наприклад prl.prj, і натиснути Введення. Проект буде завантажений в середовище. Його можна запустити на компіляцію і виконання, хоча в середовищі при цьому не буде жодного вікна з файлами проекту.

Викликати вікно проекту можна за допомогою команди Window/Project. У нижній частині екрану буде сформоване вікно, наприклад Project:PR1. У ньому розташований список файлів проекту. Для виклику у вікно редактора файлу проекту достатньо вибрати у вікні проекту рядок з необхідним файлом і двічі клацнути мишею. Можна працювати з файлами відкритого проекту.

Прикладом файлів для формування багатофайлової програми за допомогою проекту можуть бути ті ж файли, що і для формування багатофайлової програми за допомогою директиви #include, за винятком того, що у файлі з основною програмою немає рядка

#include "extfun.cpp" // – підключення файлу з функціями.

Відкрити файли з проектами можна командою Window/Project.

12.7 Робота з мишею

Головним і найважливішим способом підключення курсора миші є діставання значення регістрів пам'яті, що безпосередньо відповідають за її роботу, оскільки спеціальної функції, яка підключала б мишу в Turbo C++ 3.1

не передбачено. Звертаються при цьому до переривання 0x33 за допомогою функції intr.

Розглянемо приклад написання програми, що реалізує підключення миші. Для зручності, описання усіх функції, що відповідають за роботу миші помістимо у файл MOUSE.H, а головну програму – у файл PUTMOUSE.CPP

```
// Лістинг файлу MOUSE.H
#include<dos.h>
#define LEFT 0x01
    //0x01 – значення натиснутої лівої клавіші
#define RIGHT 0x02
    //0x02 – значення натиснутої правої клавіші
#define NOPRESSED 0x00
    //клавіша не натиснута
REGPACK reg;
// REGPACK – структура значень, що передаються і повертаються
                        функцією intr.
//=======================EXTENSION================
struct point   //структура точка
{
int x;
int y;
};
struct rect    //структура прямокутник (кнопка)
{
int x1;
int y1;
int x2;
int y2;
};
//=================================================
void initmouse()   //ініціалізація миші
{
reg.r_ax=0x00;
intr(0x33,&reg);
/* функція intr, змінює інтерфейс для виконання програмних
переривань, за допомогою структури REGPACK */
}
void showmouse(int show) //показати курсор, якщо 1; прибрати,
якщо 0
```

```
{
int f;
if(show==1)f=0x01;else f=0x02;
reg.r_ax=f;
intr(0x33,&reg);
}
point getmousepos()   /*визначення позиції курсора, за допомогою
                              структури point*/
{
point p;
reg.r_ax=0x03;
intr(0x33,&reg);
p.x=reg.r_cx;
p.y=reg.r_dx;
return p;
}
void setmousepos(point p)  //встановити позицію курсора
{
reg.r_cx=p.x;
reg.r_dx=p.y;
reg.r_ax=0x04;
intr(0x33,&reg);
}
int getbutton()  //повертає значення натиснутої клавіші, або
                        ненатискання
{
reg.r_ax=0x03;
intr(0x33,&reg);
return reg.r_bx;
}

int in(point p,rect r) /*визначає, чи належить курсор області
                              вказаної структури rect */
{
int ret=0;
if((p.x>r.x1)&&(p.x<r.x2)&&(p.y>r.y1)&&(p.y<r.y2))ret=1;
else ret=0;
return (int)(ret==1);
}
```

У файлі PUTMOUSE.CPP для прикладу моделюється меню з чотирьох кнопок: три перших ідентичні, четверта – для виходу.

Зауваження. Для роботи із програмою необхідний файл KEYRUS.COM (якщо необхідно використовувати український чи російський шрифти).

```
// Лістинг файлу PUTMOUSE.CPP
#include <iostream.h>
#include <stdlib.h>
#include <conio.h>
#include <stdio.h>
#include <graphics.h>
#include"mouse.h"       // підключення файлу MOUSE.H
void click1()      // функція клавіші 1
{
 showmouse(0);
 cleardevice();
 setbkcolor(BLUE);
 setfillstyle(3,RED);
 bar(5,5,635,475);
 setfillstyle(1,DARKGRAY);
 bar(10,10,630,470);
 settextstyle(0, HORIZ_DIR, 2);
 setcolor(LIGHTGREEN);
 outtextxy(150,100," Натиснута 1 клавіша: ");
 settextstyle(0, HORIZ_DIR, 1);
 outtextxy(130,300," Для повернення в меню натисніть праву
                    клавішу миші ");
 while(getbutton()!=RIGHT)
{
  if(getbutton()==RIGHT) // якщо натиснута права кнопка
  goto m2;
 }
 m2:cleardevice();
 closegraph();
}
void click2()        // функція клавіші 2
{
 showmouse(0);
 cleardevice();
 setbkcolor(BLUE);
```

```
 setfillstyle(3,YELLOW);
 bar(5,5,635,475);
 setfillstyle(1,WHITE);
 bar(10,10,630,470);
 settextstyle(0, HORIZ_DIR, 2);
 setcolor(BLACK);
outtextxy(150,100," Натиснута  клавіша 2: ");
 settextstyle(0, HORIZ_DIR, 1);
 outtextxy(130,300," Для повернення в меню натисніть праву
                          клавішу миші ");
while(getbutton()!=RIGHT)
{
  if(getbutton()==RIGHT) // якщо натиснута права кнопка
  goto m3;
}
 m3:cleardevice();
 closegraph();
}
void click3()        // функція клавіші 3
{
 showmouse(0);
 cleardevice();
 setbkcolor(BLUE);
 setfillstyle(3,YELLOW);
 bar(5,5,635,475);
 setfillstyle(1,RED);
 bar(10,10,630,470);
 settextstyle(0, HORIZ_DIR, 2);
 setcolor(BLACK);
outtextxy(150,100," Натиснута  клавіша 3: ");
 settextstyle(0, HORIZ_DIR, 1);
 outtextxy(130,300," Для повернення в меню натисніть праву
                          клавішу миші ");
while(getbutton()!=RIGHT)
{
  if(getbutton()==RIGHT) // якщо натиснута права кнопка
  goto m3;
}
  m3:cleardevice();
```

```
 closegraph();
}

REGPACK r;

void main()       // головна програма
{
 m1:
 int d=DETECT,q;
 point a;
 rect t={200,150,400,200};
 rect v={200,220,400,270};
 rect w={200,290,400,340};
 rect m={200,360,400,410};   // ініціалізація кнопок в структурі rect
initgraph(&d,&q,"c:\\tc\\bgi\\");
 r.r_ax=0x00;
 intr(0x33,&r);
 r.r_ax=0x01;
 intr(0x33,&r);
 setfillstyle(5,1);
 bar(t.x1,t.y1,t.x2,t.y2);
 bar(w.x1,w.y1,w.x2,w.y2);
 bar(m.x1,m.y1,m.x2,m.y2);
 bar(v.x1,v.y1,v.x2,v.y2);   // побудова кнопок меню
 setbkcolor(BLACK);
 setfillstyle(3,RED);
 bar(5,5,635,475);
 setfillstyle(1,BLACK);
 bar(10,10,630,470);
 settextstyle(0, HORIZ_DIR, 2);
 setcolor(LIGHTBLUE);
 outtextxy(110,50," Виберіть будь ласка пункт: ");
 while(1)
 {
  if (in(getmousepos(0,t)) // якщо курсор знаходиться на першій
                                кнопці
  {
   showmouse(0);
   setcolor(4);
```

```
setfillstyle(9,5);
bar(t.x1,t.y1,t.x2,t.y2);
setcolor(15);
setfillstyle(8,5);
bar(w.x1,w.y1,w.x2,w.y2);
bar(m.x1,m.y1,m.x2,m.y2);
bar(v.x1,v.y1,v.x2,v.y2);
settextstyle(0, HORIZ_DIR, 2);
setcolor(LIGHTGREEN);
outtextxy(t.x1+20,t.y1+15,"клавіша 1");
outtextxy(w.x1+20,w.y1+15,"клавіша 3");
outtextxy(m.x1+60,m.y1+15,"Вихід");
outtextxy(v.x1+20,v.y1+15,"клавіша 2");
showmouse(1);
while(in(getmousepos(),t)) /*поки курсор знаходиться на першій
                    кнопці, кнопки повторно не перерисовуються */
{
 if(getbutton()==LEFT)   // якщо натиснута ліва кнопка миші
 {
  click1();
  goto m1;
 }
}
}
if (in(getmousepos(),v))  // якщо курсор знаходиться на другій
                                кнопці
{
 showmouse(0);
 setcolor(4);
 setfillstyle(9,5);
 bar(v.x1,v.y1,v.x2,v.y2);
 setcolor(15);
 setfillstyle(8,5);
 bar(t.x1,t.y1,t.x2,t.y2);
 bar(w.x1,w.y1,w.x2,w.y2);
 bar(m.x1,m.y1,m.x2,m.y2);
 settextstyle(0, HORIZ_DIR, 2);
 setcolor(LIGHTGREEN);
 outtextxy(t.x1+20,t.y1+15,"клавіша 1");
```

```
outtextxy(w.x1+20,w.y1+15,"клавіша 3");
outtextxy(m.x1+60,m.y1+15,"Вихід");
outtextxy(v.x1+20,v.y1+15,"клавіша 2");
showmouse(1);
while(in(getmousepos(),v)) /* поки курсор знаходиться на другій
                    кнопці, кнопки повторно не перерисовуються */
{
 if(getbutton()==LEFT) // якщо натиснута ліва кнопка миші
 {
  click2();
  goto m1;
 }
}
}
if(in(getmousepos(),w))  // якщо курсор знаходиться на третій
                              кнопці
{
 showmouse(0);
 setcolor(4);
 setfillstyle(9,5);
 bar(w.x1,w.y1,w.x2,w.y2);
 setcolor(15);
 setfillstyle(8,5);
 bar(t.x1,t.y1,t.x2,t.y2);
 bar(m.x1,m.y1,m.x2,m.y2);
 bar(v.x1,v.y1,v.x2,v.y2);
 settextstyle(0, HORIZ_DIR, 2);
 setcolor(LIGHTGREEN);
 outtextxy(t.x1+20,t.y1+15,"клавіша 1");
 outtextxy(w.x1+20,w.y1+15,"клавіша 3");
 outtextxy(m.x1+60,m.y1+15,"Вихід");
 outtextxy(v.x1+20,v.y1+15,"клавіша 2");
 showmouse(1);
 while(in(getmousepos(),w)) /* поки курсор знаходиться на третій
                    кнопці,  кнопки повторно не перерисовуються */
 {
  if(getbutton()==LEFT)  // якщо натиснута ліва кнопка миші
  {
   click3();
```

```
    goto m1;
   }   }
  }
  if(in(getmousepos(0,m)) // якщо курсор знаходиться на четвертій
                                кнопці
  {
   showmouse(0);
   setcolor(4);
   setfillstyle(9,5);
   bar(m.x1, m.y1, m.x2, m.y2);
   setcolor(15);
   setfillstyle(8,5);
   bar(t.x1, t.y1, t.x2, t.y2);
   bar(w.x1, w.y1, w.x2, w.y2);
   bar(v.x1, v.y1, v.x2, v.y2);
   settextstyle(0, HORIZ_DIR, 2);
   setcolor(LIGHTGREEN);
   outtextxy(t.x1+20, t.y1+15,"клавіша 1");
   outtextxy(w.x1+20, w.y1+15,"клавіша 3");
   outtextxy(m.x1+60, m.y1+15,"Вихід");
   outtextxy(v.x1+20, v.y1+15,"клавіша 2");
   showmouse(1);
   while(in(getmousepos(0,m)) /* поки курсор знаходиться на
         четвертій  кнопці, кнопки повторно не перерисовуються */
   {
    if(getbutton()==LEFT) // якщо натиснута ліва кнопка миші
    {
     closegraph();
     exit(0);
     }   }
  }
  if((!in(getmousepos(0,t))&&(!in(getmousepos(0,w))&&(!in(getmousepos
(0,m))&& (!in(getmousepos(0,v)))
   {
    showmouse(0);
    setcolor(15);
    setfillstyle(8,5);
    bar(t.x1, t.y1, t.x2, t.y2);
    bar(w.x1, w.y1, w.x2, w.y2);
```

```
    bar(m.x1, m.y1, m.x2, m.y2);
    bar(v.x1, v.y1, v.x2, v.y2);
    settextstyle(0, HORIZ_DIR, 2);
    setcolor(LIGHTGREEN);
    outtextxy(t.x1+20, t.y1+15,"клавіша 1");
    outtextxy(w.x1+20, w.y1+15,"клавіша 3");
    outtextxy(m.x1+60, m.y1+15,"Вихід");
    outtextxy(v.x1+20, v.y1+15,"клавіша 2");
    showmouse(1);
  while(((!in(getmousepos(0,t))&&(!in(getmousepos(0,w))&&(!in(getmous
epos(0,m))&&(!in(getmousepos(0,v))))
   /* поки курсор не знаходиться на жодній кнопці, кнопки повторно
                                                  не перерисовуються */
    if(kbhit());
   }
  }
  closegraph();
 }
```

ПРИМІТКА. Для запуску програми з російським або українським шрифтом необхідно створити BAT - файл, який одночасно запускає файл KEYRUS.COM і файл програми. Наприклад, для запуску файлів KEYRUS.COM і PUTMOUSE.EXE необхідно прописати:

```
Keyrus.com
Putmouse.exe
```

і зберегти файл із розширенням *.BAT. Для запуску BAT - файлу усі три файли повинні бути в одній папці.

13 Бібліотечні функції

Створення програм на Сі не уявляється без широкого інтенсивного застосування стандартних бібліотечних функцій мови, адже її синтаксис не містить операцій, що працюють безпосередньо із складними об'єктами – рядками символів, множинами, масивами, списками. У мові навіть не передбачено ні таких звичних для багатьох операцій введення-виведення, ні вбудованих методів доступу до файлів, як це, приміром, ми бачили у Паскалі. Що це – явний недолік або якась особливість мови?

Звернемо увагу на те, що Сі орієнтований на використання стандартних бібліотек та бібліотек користувача, тоді як у Паскаль вбудовано безліч всіляких компонентів мови, у тому числі й введення-виведення. Тому ця особливість робить ядро мови Сі значно меншим та компактним за об'ємом,

що надає можливість великій кількості бібліотечних функцій працювати більш ефективно, при цьому забезпечувати тісний зв'язок з операційною системою на різних рівнях. Розуміння саме цієї переваги поглиблюється в міру регулярної роботи із Сі.

Для полегшення виконання лабораторних, контрольних та практичних робіт, викладених у другій частині посібника, тут пропонується перелік близько ста вісімдесяти функцій, що групуються за алфавітом. Після назви функції йде її прототип, опис роботи (призначення функції) та, у більшості випадків, конкретний програмний приклад допомоги, вибраний із вбудованого Help'у мови або написаний спеціально для демонстрації можливого застосування розглядуваної функції на практиці.

За мету не ставилося намагання охопити максимально весь перелік функцій – та й навряд чи у кількості міг би бути сенс. У запропонований список не увійшли функції, що не мають широкого застосування саме у курсі початкового програмування на Сі. Тут у переважній більшості містяться насамперед функції, що мають свої прототипи у файлах stdio.h, stdlib.h, string.h, з якими неодмінно стикатимуться програмісти, роблячи свої перші кроки у мистецтві проектування програм на Сі.

abort

Функція: ненормальне завершення виконання програми.

Синтаксис: #include<stdlib.h>

```
void abort(void);
```

Опис: дана функція виводить повідомлення про припинення роботи ("Abnormal program termination") в поток stderr та припиняє роботу програми шляхом виклику функції exit з кодом завершення 3.

Приклад:

```
#include <stdio.h>
#include<stdlib.h>
int main(void)
{   printf(" Викликаємо функцію abort()\n");
    abort();
    return 0;   /* ця точка не досягатиметься */
}
```

abs

Функція: повертає абсолютне значення цілого числа.

Синтаксис: версія для дійсних чисел та комплексних чисел.

Прототипи у файлах math.h та complex.h (відповідно int abs(int x); та double abs (complex x)).

Опис: функція abs повертає абсолютне значення цілого аргументу x. Якщо функція abs викликається при підключеному файлі stdlib.h, abs буде сприйматися як макрокоманда, що розширюється до вбудованого коду. Функція abs повертає ціле значення .

Приклад:

```
#include<stdio.h>
#include<math.h>
 int main(void)
{ int number=-1234;
  printf("число:%d абсолютне значення:%d\n",number,
                        abs (number));
  return 0;
}
```

acos

Функція: обчислює арккосинус.

Синтаксис: версія для дійсних та для комплексних чисел.

```
#include <math.h>                    #include<complex.h>
 double acos(double x)               complex acos(complex x)
```

Опис: acos повертає арккосинус введеного значення. Аргументи acos повинні бути у діапазоні від -1 до 1.

arc

Функція: рисує дугу кола.

Синтаксис: #include <graphics.h>

```
    void far arc(int x, int y, int stangle, int endangle, int radius);
```

Опис: arc рисує поточним кольором дугу з центром в точці з координатами (x,y) та радіусом radius. Дуга рисується від кута stangle до кута endangle. Якщо stangle дорівнює 0 та endangle дорівнює 360, функція arc нарисує повне коло. Кут для arc відраховується проти годинникової стрілки, де 0 градусів відповідають 3 годинам на циферблаті, 90 градусів – 12 годинам тощо.

Приклад:

```
#include <graphics.h>
#include <stdlib.h>
# include <stdio.h>
#include <conio.h>
 int main(void)
{
    int graphdriver = DETECT, gmode, errorcode;
```

```
    int midx,midy;
    int stangle = 45, endangle = 135;
    int radius = 100;
    initgraph(&graphdriver, &gmode, "");
    errorcode = graphresult();
/*результат ініціалізації */
    if(errorcode != grOk)
/* якщо помилка */
  { printf("Помилка :%s\n", grapherrormessage (errorcode));
    printf("Для закінчення натисніть будь-яку клавішу\n");
    getch();
    exit(1);    /* завершення з кодом помилки */
  }
    midx = getmaxx() / 2;
    midy = getmaxy() / 2;
    setcolor(getmaxcolor( 1));
    arc(midx,midy, stangle, endangle, radius);  /* рисування дуги*/
    getch();
    closegraph();
    return 0;
}
```

asin

Функція: обчислює арксинус.

Синтаксис: версія для дійсних та комплексних чисел.

```
#include <math.h>          double asin(double x);
#include<complex.h>        complex asin(complex x);
```

Опис: asin повертає арксинус введеного значення. Аргументи повинні бути у діапазоні від -1 до .1.

Приклад:

```
#include <stdio.h>
#include <math.h>
int main(void)
{
  double result;
  double x = 0.5;
  result = asin(x);
  printf("Арксинус від %f дорівнює %f\n",x, result);
  return 0;
}
```

atan

Функція: обчислює арктангенс.

Синтаксис: версія для дійсних та комплексних чисел.

```
#include <math.h>          double atan(double x);
#inciude<complex.h>        complex atan(complex x);
```

Опис: atan повертає арктангенс введеного значення. Комплексний арктангенс визначається так: atan(z)=-0.5*i*log((1+i*z))

atan2

Функція: обчислює арктангенс від y/x.

Синтаксис: #include <math.h>

```
double atan2 (double y, double x);
```

Опис: функція повертає арктангенс від y/x та проводить корегування результату, навіть якщо результуючий кут ближче до pi/2 або -pi/2 (x ближчий до 0).

atof

Функція: перетворює рядок у число типу double.

Синтаксис: #include <math.h>

```
double atof(const char * s);
```

Опис: atof перетворює символьний рядок, адресний покажчик якого знаходиться у аргументі s, в число подвійної точності (тип double).

Символи повинні відповідати формату:

[пропуски] [знак] [ddd] [.] [ddd] [e | E[знак]ddd]

Функція припиняє перетворення на першому нерозпізнаному символі.

Приклад:

```
#include<stdlib.h>
#lnclude<stdio.h>
int main(void)
{
  double n1;
  char *str = "123.45";
  n1 = atof(str);
  printf(" рядок = %s число = %f\n", str, n1);
  return 0:
}
```

atoi

Функція: перетворює рядок в ціле число.

Синтаксис: #include<stdlib.h>

```
int atoi(const char *s);
```

Опис: функція atoi перетворює символьний рядок const char *s , адресний покажчик якого знаходиться в аргументі s, в число типу int; дана функція розпізнає (у такому порядку):

- необов'язковий рядок табуляції та пропусків;
- необов'язковий знак;
- рядок цифр.

Символи повинні відповідати формату: [пропуски] [знак] [ddd]. Функція припиняє перетворення на першому нерозпізнаному символі. Функція atoi не відслідковує переповнення.

Приклад:

```
#include<stdlib.h>
#include<stdio.h>
 int main(void)
{
  int n;
  char *str = "12345":
  a = atoi(str);
  printf(" рядок = %s ціле = %d\n",str,n);
  return 0;
}
```

atol

Функція: перетворює рядок у число типу long.

Синтаксис: #include <stdlib.h>

long int atol(const char * s);

Опис: функція atol перетворює символьний рядок, адресний покажчик якого знаходиться в аргументі s, в число типу long; дана функція розпізнає у такому порядку:

- необов'язковий рядок табуляції та пропусків;
- необов'язковий знак;
- рядок цифр.

Символи повинні відповідати формату: [пропуски] [знак] [ddd]. Функція припиняє перетворення на першому нерозпізнаному символі. Функція atoi не відслідковує переповнення. Якщо функція atol() викликається з аргументом "123.23", буде повернено довге ціле значення "123L" , а підрядок ".23" буде проігнорований.

bar

Функція: рисує прямокутник.

Синтаксис: #include <graphics.h>

void far bar(int left,int top,int right,int bottom);

Опис: прямокутник зафарбовується, використовуючи колір та шаблон заповнення. bar не рисує контур прямокутника; для того, щоб нарисувати контур двовимірного прямокутника, використовуйте bar3d, з параметром depth, рівним 0. Верхній лівий та нижній правий кути прямокутника задані параметрами (left,top) та (right, bottom), відповідно. Координати надаються у пікселях.

Приклад:

```
#include <graphics.h>
#include <stdlib.h>
#include <stdio.h>
#include <conio.h>
int main(void)
{
int graphdriver = DETECT, gmode, errorcode;
int midx,midy,i;
initgraph(&graphdriver, &gmode, "");
errorcode =graphresult();          /* результат ініціалізації */
 if(errorcode != grOk)             /* якщо похибка */
{ printf( "Помилка :%s\n", grapherrormessage (errorcode));
  getch();
  exit(1);
}
midx = getmaxx() / 2;
midy = getmaxy() / 2;
/* цикл за шаблонами заповнення */
 for(i=SOLID_FILL; i<USER_FILL; i++)
setfillstyle(i,getmaxcolor());
bar(midx - 50, midy 50, midx + 50, midy+50);
getch(); } getch();
closegraph();
return 0;
}
```

bar3d

Функція рисує тривимірний стовпчик.

Синтаксис: #include <graphics.h>

```
void far bar3d(int left, int top, int right int bottom,
                int depth, int topflag);
```

Опис: bar3d рисує тривимірний прямокутний стовпчик, потім зафарбовує його, використовуючи шаблон та колір заповнення. Тривимірній контур

стовпчика рисується поточним кольором та типом лінії. Глибина стовпчика в точках екрану задається параметром depth. Параметр topflag визначає, чи буде рисуватися вершина тривимірного стовпчика. Якщо topflag не 0, вершина зображається, в протилежному випадку вершина не рисується, (створюється можливість ставити декілька стовпчиків один на одного). Верхній лівий та нижній правий кут прямокутника задаються (left, top) та (right, bottom), відповідно. Для обчислення типової глибини для bar3d візьміть 35 відсотків ширини стовпчика, наприклад:
bar3d(left, top, right, bottom, (right-left) /4,1).

bioskey

Функція: інтерфейс з клавіатурою за допомогою функцій BIOS.

Синтаксис: #include<bios.h>

int bioskey(int cmd);

Опис: bioskey шляхом переривання BIOS 0x16 виконує різні операції з клавіатурою. Параметр cmd визначає операцію. Значення, що повертається функцією bioskey, залежить від значення операції, яка визначається аргументом cmd:

0 – якщо молодші 8 бітів ненульові, bioskey повертає код ASCII наступної натиснутої клавіші, що очікує в черзі. Якщо молодші 8 бітів дорівнюють 0, то старші 8 бітів містять у собі розширені коди клавіатури, опис яких ви можете знайти у документах про технічне забезпечення IBM PC.

1 – відбувається перевірка, чи мало місце натиснення клавіші. Якщо при перевірці повертається значення 0, це означає, що клавіша не була натиснута. У протилежному випадку повертається значення чергової натиснутої клавіші. Сам код чергової натиснутої клавіші зберігається та буде повернений наступним викликом функції bioskey при значенні параметру cmd, який дорівнює нулю.

2 – запитує поточний статус клавіш типу SHIFT. Значення отримується при логічному додаванні наступних кодів (операції "АБО")

Біт 7 0x80 INSERT натиснуто
Біт 6 0x40 CAPS натиснуто
Біт 5 0x20 NUM LOCK натиснуто
Біт 4 0x10 SCROLL LOCK натиснуто
Біт 3 0x08 ALT натиснуто
Біт 2 0x04 CTRL натиснуто
Біт 1 0x02 LEFT SHIFT натиснуто
Біт 0 0x01 RIGHT SHIFT натиснуто

Приклад:

```
#include <stdio.h>
```

```
#include <bios.h>
#include <ctype.h>
#define RIGHT 0x0001
#define LEFT 0x0002
#define CTRL 0x0004
#define ALT 0x0008 int main(void)
{
int key, modifiers;
while (bioskey(1) == 0);
/* Очікування введення.*/;
/* Тепер отримаємо значення клавіші */
 key = bioskey(0);
/* чи використовувались клавіші типу SHIFT */
 modifiers = bioskey(2);
if (modifiers)
{ printf("[");
if (modifiers & RIGHT) printf("RIGHT ");
if (modifiers & LEFT ) printf("LEFT ");
if (modifiers & CTRL ) printf("CTRL ");
if (modifiers & ALT ) printf("ALT ");
printf("] ");
}
/* виведення символу натиснутої клавіші */
if (isalnum(key & OxFF)) printf("'%c'\n",key);
else printf("%#02x\n",key);
}
```

cabs

Функція: абсолютне значення комплексного числа.

Синтаксис: #include <math.h>

double cabs(struct complex z);

Опис: cabs – макрокоманда, що обчислює абсолютне значення комплексного числа z. z є структурою типу complex: структура визначена у math.h таким чином:

```
struct complex
{ double x,y;
};
```

де x є дійсною частиною, а y – уявною. Виклик cabs еквівалентний виклику sqrt. При використанні C++ можливо користуватися типом complex (файл complex.h) та функцією abs.

calloc

Функція: виділяє пам'ять із heap-а для динамічного розподілу.

Синтаксис: #include <stdlib.h>

void * calloc(size_t nitems, size_t size);

Опис: calloc забезпечує доступ до пам'яті heap. heap доступна для динамічного розподілу блоків пам'яті змінної довжини. Багато структур даних, наприклад, дерева та списки, використовують розподіл пам'яті heap. calloc виділяє блок пам'яті розміром nitems x size. Блок обнульовується. Якщо розмір блоку перевищує 64 Кб , використовуйте функцію farcalloc. calloc повертає покажчик на виділений блок. calloc повертає NULL, якщо недостатньо пам'яті для виділення нового блоку, або nitems або size нульові.

Приклад:

```
#include<stdio.h>
#include<alloc.h>
#include<string.h>
  int main(void)
  {
    char *str = NULL;
    /* виділити пам'ять для рядка. */
    str =calloc(10,sizeof(char));
    if (str)
    {
    strcopy(str,"Hello");   /* скопіювати у рядок "Hello" */
    printf("Рядок :%s\n",str);
    free(str);                    /* звільнити пам'ять */}
    else
    {
      printf("Недостатньо пам'яті \n");
       return (0);
 }
```

ceil

Функція: округлення.

Синтаксис: #include <math.h>

double ceil(double x);

Опис: ceil знаходить найближче справа ціле число, що є не меншим x; повертає знайдене число (типу double). Див. також floor, fmod.

Приклад:

```
#include<math.h>
#include<stdio.h>
```

```
int main(void)
{
  double number = 112.54;
  double down, up;
  down = floor(number);
  up = ceil(number);
  printf("число:%5.21f\n",number);
  printf("Округлення зліва :%5.2f\n",down);
  printf("Округлення справа :%5.2f\n",up);
  return 0;
}
```

cgets

Функція: зчитує рядок з консолі.

Синтаксис: #include<conio.h>
char *cgets(char *str);

Опис: cgets зчитує символьний рядок з консолі та зберігає його (або його довжину) у буфері-параметрі str. Зчитування символів продовжується доти, доки не зустрінеться комбінація CR/LF (повернення каретки /переведення рядка) або поки не буде зчитаною вказана кількість символів. Якщо cgets зчитує комбінацію CR/LF, вона замінює їх на \0 (нульове обмеження рядка).

Див. також cputs, fgets, getch, getche, gets.

Приклад:

```
#include <stdio.h>
#include <conio.h>
  main()
{
  char buffer[83];
  char *p;
  buffer[0] = 81;              /* місце під 81 символ */
  p = cgets(buffer);
  printf("\ncgets підрахувала %d символів:\"%s\"\n",buffer[1], p);
  printf("Повертається покажчик %p, buffer[2] на %p\n",p,&buffer);
  buffer[0] = 6;               /* місце під 5 символів та заключний 0 */
  printf("Введіть декілька символів\n");
  p = cgets(buffer);
  printf("\ncgets зчитала %d символів: \"%s\"\n,buffer[1], p);
  printf("Повертається покажчик %p, buffer[2] на %p\n",p,&buffcr);
  return 0;
}
```

chdir

Функція: змінює поточну директорію.

Синтаксис: #include<dir.h>

int chdir(const char * path);

Опис: специфікація також може задаватися в аргументі path, наприклад, chdir("a:\\tc"), проте змінюється тільки поточна директорія; сам активний пристрій не змінюється. При успішному завершенні функція chdir повертає 0, інакше – 1. Функція chdir підтримується на UNIX-системах.

Приклад:

```
#include<stdio,h>
#include<stdlib.h>
#include<dir.h>
char old_dir[MAX_DIR];
char new_dir[MAX_DIR];
int main(void)
{
  if(getcurdir(0,old_dir))
  {
   perror( "getcurdir()") ;
   exit(l);
  }
  printf(" Поточна директорія : \\%s\n",old_dir);
  if(chdir("\\")
  {
   perror("chdir()");
   exit(1);
  }
  if(getcurdir(0, new_dir) )
   {
   perror("getcurdir()");
   exit(1);
   }
   printf( "Тепер поточна директорія: \\%s\a",new_dir);
   if(chdir(old_dir)
   {
   perror("chdir()");
   exit(l);
   }     return 0;
}
```

chsize

Функція: змінює розмір файлу.

Синтаксис: #include<io.h>

int chsize(int handle, long size);

Опис: chsize змінює розмір файлу, пов'язаного з дескриптором handle. Можливе зменшення або збільшення файлу відносно його початкового розміру залежно від значення аргументу size. Режим, в якому відкривається файл, повинен мати дозвіл на запис. Якщо chsize збільшує файл, до нього приєднуються нульові символи (\0).

Приклад:

```
#include<string.b>
#include<fontl.h>
#include<io.h>
iot main(vold)
{
  int handle;
  char buf[ll] = "0123456789";
  /* створити текстовий файл з 10 байтів */
  handle = open("TTT.TXT", O_CREAT);
  write(handle, buf, strlen(buf));
  chsize( handle, 5);              /* обрізати файл до 5 байтів */
  close( handle);                  /*закрити файл */
  return 0;
}
```

circle

Функція: рисує коло заданого радіусу з центром у точці з координатами (x,y).

Синтаксис: #include <graphics.h>

void far circle(int x, int y, int radius);

Опис: circle рисує коло поточним кольором з центром у точці (x,y) та радіусом radius.

cleardevice

Функція: очищає графічний екран.

Синтаксис: #include <graphics.h>

void far cleardevice(void);

Опис: cleardevice очищає (відповідно зафарбовує поточним кольором фону) весь графічний екран та переміщує поточну позицію в (0,0).

_close, close

Функція: закриває файл.

Синтаксис: #include<io.h>
```
int _close (int handle);
int close (int handle);
```

Опис: _close(close) закриває файл, пов'язаний з дескриптором handle. Аргумент handle означає дескриптор файлу, що присвоюється йому з викликом функцій _creat, creat, creatnew, creattemp, dup, dup2, _open або open. Функція не записує ознаку кінця файлу (Ctrl-Z) у кінець файлу. При успішному завершенні повертає значення 0. (інакше -1).

Див. також close, _creat, open, read, write.

Приклад:
```
#include<string.h>
#include<fontl.h>
#include<io.h>
 int main(void)
{
ink handle;
char buf[ 11] = "0123456789";
handle = open("N.DOC",0_CREAT);
if(handle > -1)
{  write(handle, buf, strlen(buf));
   close( handle); }
   else
       printf( "Помилка при відкритті файлу\n");
       return 0;
}
```

closegraph

Функція: завершує роботу графічної системи.

Синтаксис: #include <graphics.h>
```
void far closegraph(void);
```

Опис: closegraph звільняє всю пам'ять, що виділена під роботу графічної системи. Зазвичай, завершує роботу графічної системи.

Див. також initgraph, setgraphbufsize.

clreol

Функція: вилучає символи до кінця рядка у текстовому вікні.

Синтаксис: #include<conio.h>
```
void clreol(void);
```

Опис: clreol вилучає усі символи від позиції курсора до кінця рядка у поточному текстовому вікні без переміщення курсора.
Див. також clrscr, delline, window.
Приклад:

```
#include<conio.h>
int main(void)
{
  clrscr();
  gotoxy(14, 4);
  getch();
  clreol();
  return 0;
}
```

clrscr

Функція: очищає вікно у текстовому режимі.
Синтаксис: #include<conio.h>
void clrscr(void);
Опис: clrscr очищає текстове вікно та переміщує курсор в позицію (1,1).
Див. також clreol, delline, window.
Приклад:

```
#include<conio.h>
int main(void)
{
 int i;
 clrscr();
 for (i=0; i<20; i++)
 cprintf("%d\r\n",i);
 cprintf("\r\n Для очистки екрану натисніть будь-що, окрім Reset");
 getch();
 clrscr();
 cprintf("Чисто!");
 return 0;
}
```

complex

Функція: створює комплексні числа.
Синтаксис: #include<complex.h>
complex complex(double real, double imag);

Опис: створює комплексне число з дійсної та уявної частини. Якщо параметр imag відсутній, то уявна частина вважається рівною 0. В complex.h, крім того, перевизначаються операції + , - , * , / , += , -= , *= , /= , = , = = ,
Приклад:

```
#include<complex.h>
int main(void)
{
 double x = 3.1, y=4.2;
 complex z = complex(x,y);
 ……
return 0;
}
```

cos

Функція: обчислює косинус.
Синтаксис: Дійсна та комплексна версія:

```
#include <math.h>                    #include<complex.h>
 double cos(double x);               complex cos(complex x);
```

Опис: cos повертає косинус введеного значення. Кут задається у радіанах.
Див. також acos, asin, atan, complex, sin, tan.

cprintf

Функція: здійснює форматоване виведення на екран.
Синтаксис: #include<conio.h>

```
int cprintf(const char *format[,argument,...]);
```

Опис: cprintf отримує набір аргументів, застосовує для кожного аргументу специфікацію формату, яка міститься у рядку формату з покажчиком format, та виводить відформатовані дані на екран у текстове вікно. Число аргументів та специфікацій повинно бути однаковим. Опис: специфікації формату наведено у функції printf(). На відміну від функцій fprintf() та printf(), cprintf() не перетворює символи переводу рядка (\n) у послідовність переводу каретки та переводу рядка(\r\n). cprintf() повертає кількість виведених символів.
Приклад:

```
#include<conio.h>
 int main(void)
 {  clrscr();
    window(10,10,80,25);
/* виведення у вікно деякого тексту */
   cprintf( "Hello, dummy !!!\r\n");
```

```
    getch();
    return 0;
}
```

cputs

Функція: виводить рядок на екран.

Синтаксис: #include<conio.h>

int cputs(const char * str);

Опис: cputs виводить рядок str, що закінчується нулем, у деяке текстове вікно. До рядка не приєднується символ нового рядка. Функція cputs повертає останній виведений символ. Див. також cgets, fputs, putch, puts.

Приклад:

```
#include<conio.h>
int main(void)
{
clrscr();
window(15,15,70,25);
cputs("Hello, Marry..! \r\n");
return 0;
}
```

creat

Функція: створює новий файл або перезаписує існуючий.

Синтаксис: #include <sys\stat.h>

int creat(const char * path, int amode);

Опис: creat створює новий файл або підготовлює для перезапису існуючий файл, ім'я котрого знаходиться в аргументі path. amode застосовується тільки для нових файлів. Файл створюється у режимі, що вказується у глобальній змінній _fmode (O_TEXT або O_BINARY). Якщо файл вже існує та встановлений атрибут запису, creat обрізає файл до довжини 0 байтів, залишаючи незмінними атрибути файлів. Якщо існуючий файл має атрибут "тільки читання", виклик creat буде невдалим та файл залишиться незмінним. amode може набувати одного з таких значень (визначених у файлі stat.h, значення amode та режим доступу):

S_IWRITE – дозвіл на запис

S_IREAD – дозвіл на читання

S_IREAD|S_IWRITE – дозвіл на "читання та запис"

До речі, DOS-дозвіл на запис несе за собою дозвіл на читання. Функція creat підтримується в системах UNIX.

Приклад:

```
#include<sys\stat.h>
```

```
#include<string.h>
#include<fcntl.h>
#include<io.h>
 int main(void)
{
int handle;
char buf[11] = "0123456789";
/* змінити режим з текстового у двійковий */
_fmode = O_BINARY;        /* змінити режим у двійковий */
/* створити двійковий  файл для читання та запису */
 handle = creat("MY.EXT",S_IREAD | S_IWRITE);
/* записати у файл */
write(handle, buf, strlen( buf) );
close(handle);                    /* закрити файл */
 return 0;
}
```

creatnew

Функція: створює новий файл.

Синтаксис: #include <dos.h>

int creatnew(const char * path, int mode);

Опис: creatnew ідентична функції _creat, тільки якщо файл існує, то creatnew видає помилку та не змінює файл. Аргумент mode в creatnew може бути однією з таких констант(в dos.h):

FA_DRONLY – тільки читання
FA_HIDDEN – прихований файл
FA_SYSTEM – системний файл

Приклад:

```
#include<stdlib.h>
#include <errno. h>
#include<dos.h>
#include<io.h>
 int main(void)
{
 int handle;
char buf[11] = "0123456789";
/* спроба створити неіснуючий файл */
handle = creatnew("MY.FIL",0);
if (handle == -1)
    printf("MY.FIL вже існує\n");
```

```
        else {
        printf("MY.FIL успішно створений \n");
        write( handle, buf,strlen(buf));
        close(handle);
             }
    return 0;
   }
```

creattemp

Функція: створює унікальний файл у директорії, що вказана у маршруті.

Синтаксис: #include <dos.h>

int creattemp(char * path, int attrib);

Опис: path є ім'ям маршруту, що завершується символом (\). creattemp бере аргумент attrib – слово атрибутів DOS. Файл завжди відкривається у двійковому режимі. При успішному створенні файлу покажчик файлу встановлюється на початок файлу. Аргумент attrib визначається як константа в dos.h:

FA_DRONLY – тільки читання
FA_HIDDEN – прихований файл
FA_SYSTEM – системний файл

Приклад:

```
#include<string.h>
#include<stdio.h>
#include<io.h>
 int main(void)
{
int handle;
char pathname[128];
strcpy(pathname, "\ \");
handle = creattemp(pathname,0);
printf("Був створений файл %s\n",pathname);
close(handle);
return 0;
 }
```

cscanf

Функція: виконує форматоване посимвольне введення з консолі.

Синтаксис: #include<conio.h>

int cscanf(char * format[,address,...]);

Опис: cscanf зчитує з консолі та переглядає набір полів, що вводяться, по одному символу. Кожне поле форматується відповідно до специфікації формату. Специфікація формату наведена в описі функції scanf. З багатьох причин cscanf може припинити сканування відповідного поля до його нормального завершення (символу пропуску) або взагалі завершити введення. Ці причини аналогічні для функції scanf. cscanf повертає число успішно введених полів (інакше повертається значення 0). Якщо cscanf прагне зчитати кінець файлу, то повертається значення EOF.
Див. також fscanf, getche, scanf, sscanf.
Приклад:

```
#include<como.h >
 int main(void)
{
char string(80);
clrscr();
cprintf("Введіть рядок:");
cscanf("%s",string);          /* зчитати рядок */
/* вивести введений рядок */
cprintf( "Введено наступне : %s", string);
return 0;
}
```

delay

Функція: затримує виконання програми на інтервал у мілісекундах.
Синтаксис: #include<dos.h>

```
void delay (unsigned milliseconds);
```

Опис: при виклику функції delay виконання програми зупиняється на час, що визначається параметром milliseconds. Див. також nosound, sleep, sound.
Приклад:

```
/*видає звук з частотою 440 Гц протягом 500 мілісекунд */
 #include <dos.h>
 int main(void)
{
sound(440);
delay(500);        // затримка
nosound();         // відміна звуку
return 0;
}
```

delline

Функція: вилучає рядок у текстовому вікні.

Синтаксис: #include<conio.h>

void delline(void);

Опис: delline вилучає рядок, у якому знаходиться курсор і переміщає усі рядки, нижче данного, на один вище. Функція delline працює у поточному активному вікні. Див. також clreol, clrscr, insline, window.

detectgraph

Функція: визначає графічний драйвер та графічний режим при перевірці апаратного забезпечення.

Синтаксис: #include <graphics.h>

void far detectgraph (int far*graphdriver, int far *graphmede);

Опис: detectgraph визначає ваш системний графічний адаптер та обирає режим, що забезпечує найвищий дозвіл для цього адаптера. Якщо графічний адаптер не знайдено, *graphdriver встановлюється в мінус 2 та graphresult буде повертати мінус 2 (grNotDetected). Див. також graphresult, initgraph.

Приклад:

```
#include <graphics.h>
#include <stdlib.h>
#include <stdio.h>
#include <conio.h>
/* Найменування різних графічних адаптерів*/
char *gname[]={"потрібно визначення", "CGA", "EGA", "EGA з 64К",
"монохромний EGA", "IBM 8514", "Hercules монохромний",
"AT&T 6300 PC". "VGA", "IBM 3270" };
int main(void)
{ int graphdriver, gmode, errorcode;
 etectgraph(&graphdriver,&gmode,""); /*визначення адаптера*/
 errorcode = graphresult();
 if(errorcode != grOk)
 {   printf ("Помилка :%s\n", grapherrormessage(errorcode));
     getch();   exit(1);          /* завершення з кодом помилки */
 }
 clrscr();
printf( "Встановлена така плата: %s",gname[graphdriver]);
getch();
return 0; }
```

div

Функція: проводить ділення двох цілих одне на одне та повертає частку та залишок.

Синтаксис: #include <stdlib.h>

div_t div(int numer, int denom);

Опис: div ділить два цілих та повертає частку та залишок в структурі типу div_t - numer\denom. Структура div_t визначена в stdlib.h

```
typedef struct {
int quot;       /* частка */
int rem;        /* залишок */}
div_t;
```

Приклад:

```
#include<stdlib.h>
#include<stdio.h>
div_t x;
int main(void)
{
x = div(10,3);
printf("10 div 3 = %d, ост = %d", x.quot, x.rem);
return 0;
}
```

drawpoly

Функція: рисує контур багатокутника.

Синтаксис: #include <graphics.h>

void far drawpoly (int numpoints, int far *polypoints);

Опис: drawpoly рисує багатокутник, що має numpoints точок, використовуючи поточні тип лінії та колір, *polypoints вказує на послідовність з (numpoints x 2) цілих чисел. Кожна пара чисел x та y є координатами вершини. Див. також fillpoly, floodfill, graphresult, setwritemode.

ellipse

Функція: рисує еліптичні дуги.

Синтаксис: #include <graphics.h>

void far ellipse (int x,int y,int stangte, int endangle,
int xradius, int yradius).

Опис: ellpise рисує еліптичну дугу з центром в точці з координатами (x,y), горизонтальною та вертикальною осями радіусом xradius та yradius, відповідно, поточним кольором. Дуга рисується від кута stangle до кута

endangle. Якщо stangle дорівнює 0 та endangle дорівнює 360, функція arc нарисує повний еліпс. Див. також arc, circle, fillellipse, sector.

eof

Функція: визначає, чи досягнутий кінець файлу.

Синтаксис: #include<io.h>

int eof(int handle);

Опис: функція eof визначає, чи досягнутий кінець файлу, пов'язаного з дескриптором handle. Якщо поточна позиція є кінцем файлу, функція eof повертає значення 1; в протилежному випадку – значення 0. При виникненні помилки повертається значення мінус 1, а глобальна змінна errno отримує значення EBADF – неправильний номер файлу. Див. також clearerr, feof, ferror, perror.

Приклад:

```
#include<process.h>
#include<string.h>
#include<stdio.h>
#include<io.h>
int main(void)
{
FILE *temp_file;
int handle;
char msg[] = "Тест- приклад";
char ch;
/* створити унікальний тимчасовий файл */
 if ((temp_file=tmpfile()) == NULL)
{
  perror("Відкриття файлу не відбулося !!!!");
  exit(1);
}
/* отримати дескриптор даного файлу * /
handle = fileno(temp_file);
/* записати у файл дані */
write(handle, msg,strlen(msg));
/* перемістити покажчик на початок файлу */
lseek(handle,01, SEEK_SET);
/*зчитування даних до появи кінця файлу */
do
{ read(handle, &ch, 1);  printf("%c",ch); }
```

```
while( feof (handle);
/* закрити тимчасовий файл fclose(temp_file);
return 0;
}
```

execl, execle, execv, execve, execvp, execvpe

Функції: завантажують та запускають інші програми.

Синтаксис:

```
int execl(char * pathname, char * arg0, arg1, ..., argn, NULL);
int execle(char * pathname, char * arg0, arg1, ..., argn, NULL,
        char ** envp);
int execlp(char * pathname, char * arg0, arg1, ..., argn, NULL);
int execlpe(char *pathname, char * arg0, arg1, .... argn, NULL,
        char ** envp);
int execv(char * pathname, char * argv[]);
int execve(char * pathname, char * argv[], char ** envp);
int execvp(char * pathname, char * argv[]);
int execvpe(char * pathname, char * argv[],
char ** envp);
```

Файл, що містить прототип – process. h

Опис: сімейство функцій exec... завантажує та запускає інші програми, відомі як "дочірні" процеси. Якщо виклик функції exec... завершується успішно, "дочірній" процес накладається на "батьківський" процес; причому повинно бути достатньо пам'яті для завантаження та виконання "дочірнього" процесу. pathname – це ім'я файлу активізованого "дочірнього" процесу. Функції exec... провадять пошук pathname, використовуючи стандартний алгоритм системи DOS. При успішному завершенні функції exec... не повертають ніякого значення. При виникненні помилки функції exec... повертають значення мінус 1, а глобальна змінна errno отримує конкретне значення. exec... унікальна для DOS. Див. також: abort, exit, searchpath, spawn.

exit

Функція: припиняє виконання програми.

Синтаксис: #include<stdlib.h>

```
void exit(int status);
```

Опис: функція exit припиняє активізований процес. Перед виходом з процесу усі файли закриваються, записується буферне виведення та викликаються зареєстровані "функції виходу"(оголошення у функції atexit). У будь-якому випадку для активізованого процесу забезпечується аргумент status, що є статусом виходу для даного процесу. Значення, що дорівнює

нулю, використовується для позначення нормального виходу з процесу, а ненульове значення означає наявність помилки.
Див. також функції abort, atexit, exec..., keep .
Приклад:

```
#include<stdlib.h>
#include<conio.h>
#include<stdio.h>
 int main(void)
{
  int status;
  printf("Натисніть клавішу 1 чи 2\n");
  status = getch();
  exit(status-'0');
  return 0; /* ця точка не досягається * /
}
```

exp

Функція: функція експоненти, що повертає значення e в степені x.
Синтаксис: у файлах `#include<math.h>` `#include<complex.h>`

```
double exp(double x);   complex exp(complex x);
```

Приклад:

```
#include<stdio.h>
 #include<math.h>
 int main(void)
 {
 double result;
 double x = 4,0;
 result = exp(x);
 printf("'e' в степені %f (e#%f) = %1f\n",x,x,result);
 return 0;
 }
```

fabs

Функція: повертає абсолютне значення числа з плаваючою точкою.
Синтаксис:

```
#include<math.h>
        double fabs(double x);
```

Опис: fabs обчислює абсолютне значення x та повертає його як double.
Приклад:

```
#include<stdio.h>
#include<math.h>
int main(void)
```

```
{ float number = -232.5;
  printf("число: %f, абсолютне значення %f\n", number,
        fabs( number));
return 0;
}
```

fclose

Функція: закриває потік.

Синтаксис: #include <stdio.h>
int fclose (FILE * stream);

Опис: функція fclose закриває вказаний потік stream; усі буфери, пов'язані з потоком stream, перед закриттям звільняються. Буфери, розміщені системою звільняються під час процесу закриття. Функція fclose при успішному завершенні повертає 0. Див. також close, fdopen, fflush, fopen, freopen.

fcloseall

Функція: закриває відкриті потоки.

Синтаксис: #include<stdio.h>
int fcloseall(void);

Опис: функція fcloseall закриває усі відкриті потоки, за винятком stdin та stdout, stdprn, stdaux та strerr. fcloseall повертає кількість закритих потоків. За наявності помилки повертається EOF.

Див. також fclose, fdopen, fopen, freopen.

Приклад:

```
#include<stdio.h>
 int main(void)
{ FILE *fpl,*fp2;
  int streams_closed;
/* відкрити два потока * /
fp1 = fopen("my.txt","w");
fp2 = fopen("your.txt","w");
streams_closed = fdoseall();
/* закрити */
if(streams_closed == EOF)
/* вивести повідомлення про помилку */
perror(" Помилка ");
else
/* вивести результат роботи функції */
printf("Було закрито %d потока\n", streams _closed);
return 0; }
```

fdopen

Функція: пов'язує потік з логічним номером файлу.

Синтаксис: #include<stdio.h>

FILE * fdopen(int handle, char * type);

Опис: fdopen пов'язує потік з дескриптором, отриманим функціями creat, dup, dup2 або open. Тип потоку повинен збігатися з режимом, в якому був відкритий handle. При успішному завершенні fdopen повертає новий відкритий поток stream. У випадку помилки, функція повертає нуль (NULL). fdopen підтримується системами UNIX.

Див. також функції fclose, freopen, fopen, opene.

Приклад:

```
#include<sys\stat.h>
#include <stdio.h>
#include <fcntl.h>
#include<io.h>
int main(void)
{
int handle;
FILE *stream;
/* відкрити файл */
handle = open("MY.TXT", 0_CREAT, S_IREAD | S_IWRITE);
stream = fdopen(handle, "w");        //потік
if (stream == NULL)
printf( "ПОМИЛКА fdopen\n");
else { fprintf(stream, "Hello, world\n");
       fclose(stream); }
 return 0;
}
```

feof

Функція: знаходить кінець файлу (EOF) у потоці.

Синтаксис: #include<stdio.h>

int feof(FILE * stream);

Опис: функція feof є макрокомандою, яка проводить перевірку даного потоку stream на ознаку кінця файлу (EOF). Ознака кінця файлу відміняється при кожній операції введення. Див. також clearerr, eof, ferror, perror.

Приклад:

```
#include <stdio.h>
int main(void)
{ FILE *stream; char ch;
```

```
stream = fopen("my.txt", "r");
ch = fgetc(stream);
if ( feof (stream))  /* кінець файлу? */
printf("Кінець файлу \n");
fclose(stream);   /* закрити файл */
return 0; }
```

fflush

Функція: записування складу потоку у файл.

Синтаксис: #include<stdio.h>
int fflush(FILE * stream);

Опис: функція fflush записує у файл склад буферу, пов'язаного з потоком stream, якщо він був відкритий для виведення. fflush не впливає на небуферизовані потоки. Див. також fclose, flushall, setbuf, setvbuf.

Приклад:

```
#include<string.h>
#include<stdio.h>
#include<conio.h>
#include<io.h>
void flush(FILE *stream);
int main(void)
{  FILE *stream:
   char msg[] = "Тест";
/* створити файл */
 stream = fopen("UM.FIL","'w");
/* записати у файл дані */
 fwrite(msg,strlen(msg), 1 ,stream);
clrscr(); printf("Для скидання потоку тисніть любу клавішу ...\n");
getch();
/* скинути дані у файл, без його закриття */
flush(stream); printf ("Вміст буферів скинуто у файл \n");
return 0; }
void flas(FILE *stream)
{ int duphandle;
/* скинути внутрішні буфери файлу */
fflash(stream);
/* створити другий дескриптор */
duphandle = dup(fileno( stream);
/* закрити другий дескриптор для скидання буферів DOS */
close(duphandle); }
```

fgetc

Функція: отримує символ з потоку.

Синтаксис: #include<stdio.h>

int fgetc(FILE * stream);

Опис: fgetc повертає наступний символ із вказаного вхідного потоку stream. При успішному завершенні повертається символ, перетворений до типу int без розширення знаку. При досяганні кінця файлу повертається EOF. Див. також fgetchar, fpytc, getc, ungetc, ungetch.

Приклад:

```
#include<string.h>
#include<stdio.h>
#include<conio.h>
 int main(void)
{
FILE *stream;
char msg[] = "Тестовий приклад";
char ch;
stream = fopen("MY.FIL","w+");      /* файл для модифікації */
fwrite(msg,strlen(msg),1, stream);  /* записати у файл дані */
fseek(stream,0,SEEK_SET);           /* перейти на початок файлу */
do
{ ch = fgetc(stream);               /* ввести символ з файлу */
  putch(ch);                        /* вивести символ на екран */
}
while(ch!=EOF);
fclose(stream);
return 0;  }
```

fgetchar

Функція: отримує символ з потоку stdin.

Синтаксис: #include<stdio.h>

int fgetchar(void);

Опис: fgetchar повертає наступний символ з потоку stdin. Визначається як fgetc(stdin). При успішному завершенні повертає символ, перетворений до типу int без розширення знаку. При досяганні кінця файлу або помилки повертає EOF. Див. також fgetc, fputchar, getchar.

Приклад:

```
#include<stdio.h>
int main(void)
{ char ch;
```

```
printf ("Введіть символ, натисніть <Enter>.\n");  /* запит */
ch = fgetchar();                  /* ввести символ з потоку stdin */
printf ("Зчитаний символ: '%c'\n", ch);  /* надрукувати */
return 0;
}
```

fgetpos

Функція: повертає місцезнаходження покажчика поточної позиції у файлі.

Синтаксис: #include<stdio.h>

int fgetpos(FILE * stream, fpos_t *pos);

Опис: fgetpos зберігає позицію покажчика файлу, пов'язаного з потоком stream, у місці, на яке вказує pos. Конкретне числове значення може бути корисним тільки для повторного звертання до функції fgetpos(). При успішному завершенні fgetpos повертає 0. Див. також fseek, fsetpos, ftell, tell.

Приклад:

```
#include<string.h>
#include<stdio.h>
int main(void)
{ FILE *stream;
  char string[] = "Тест-приклад";
  fpos_t filepos;
/* створити файл для його модифікації */
  stream = fopen("2MY.FIL","w+");
  fwrite(string, strlen(string), 1, stream);
/* записати дані*/
/* повідомити положення покажчика */
fgetpos(stream, &filepos);
printf ("Покажчик знаходиться у %ld позиції \n");
fclose(stream);
return 0;
}
```

fgets

Функція: отримує рядок символів з потоку.

Синтаксис: #include<stdio.h>

char * fgets(char s, int n, FILE *stream);

Опис: fgets зчитує з потоку stream рядок символів та розміщує його в s. Введення завершується після вводу n-1 символу або при введенні символу

переходу на наступний рядок, з огляду на те, що виникне раніше. На відміну від gets, fgets припиняє введення рядка при отриманні символу переходу на наступний рядок. Нульовий байт приєднується в кінець рядка для індикації його кінця. При успішному завершенні повертається покажчик на s, при помилці або кінці файлу – NULL. Див. також cgets, fputs, gets.

filelength

Функція: отримує розмір файлу у байтах.

Синтаксис: #include<io.h>

long filelength(int handle);

Опис: функція filelength повертає довжину у байтах файлу, що відповідає дескриптору handle. Див. також fopen, lseek, open.

fileno

Функція: отримує дескриптор файлу.

Синтаксис: #include<stdio.h>

int fileno(FILE * stream);

Опис: fileno є макрокомандою, що повертає логічний номер файлу для заданого потоку stream. Якщо поток stream має більше одного номеру, функція fileno повертає номер, призначений даному потоку при першому відкритті. Див. також fdopen, fopen, freopen.

Приклад:

```
#include<stdio.h>
 int main(void)
{  FILE *stream;
   int handle;
   stream = fopen("ccc.c", "w");
/* отримати дескриптор файлу */
 handle = fileno(stream);
/* надрукувати його */
printf("Дескриптор файлу = %d\n", handle);
fclose( stream);
return 0;
}
```

fillpoly

Функція: рисує та зафарбовує багатокутник.

Синтаксис: #include <graphics.h>

void far fillpoly (int numpoints, int far *polypoints);

Опис: fillpoly рисує контур багатокутника, що має numpoints точок, використовуючи поточні вид лінії та колір (як це робить drawpoly). Polypoints вказує на послідовність з (numpointsx 2) цілих чисел. Кожна пара чисел x та y є координатами вершини. Див, також drawpoly, graphresult, setfillstyle.

Приклад:

```
#include <graphics.h>
#include <stdlib.h>
#include <stdio.h>
#include <conio.h>
int main(void)
{
int graphdriver = DETECT, gmode, errorcode;
int i, maxx, maxy;
int poly[10];
initgraph(&graphdriver &gmode,"");
errorcode = graphresult();
maxx = getmaxx();
maxy=getmaxy();
poly[0] = 20;          /* 1 точка */
poly[1] = maxy/2;
poly[2]= maxx-20;  /* 2 точка */
poly[3] = 20;
poly[4] = maxx-50;  /* 3 точка */
poly[5] = maxy-20;
poly[6] = maxx/2;    /* 4 точка */
poly[7] = maxy/2;
poly[8] = poly[0]; /* повернення у початкову точку */
poly[9] = poly[1];
/* цикл по шаблонам */
for (i=EMPTY_FILL;  i<USER_FILL;  i++)
{
 setfillpattern( i, getmaxcolor( 1));
/* визначили шаблон */
fillpoly(5,poly);
/* нарисували заповнений багатокутник */
getch(); }
 closegraph();
return 0; }
```

floor

Функція: округлення у меншу сторону.

Синтаксис: #include<math.h>

```
double floor(double x);
```

Опис: floor знаходить найбільше ціле число, що не перевищує значення x. Повертає ціле число (double).

flushall

Функція: звільняє буфери усіх потоків.

Синтаксис: #include<stdio.h>

```
int flushall(void);
```

Опис: функція flushall звільняє усі буфери, пов'язані з відкритими вхідними потоками, та записує у відповідні файли усі буфери, що мають відношення до відкритих вихідних потоків. Усі потоки залишаються відкритими, flushall повертає число відкритих вхідних та вихідних потоків. Див. також fclose, fcloseall, fflush.

Приклад:

```
#include<stdio.h>
int main(void)
{
FILE *stream;
stream = fopen("U.C", "w");
printf("Звільнено %d потоків \n", flushall());
fclose(stream);
return 0;
}
```

fmod

Функція: обчислює X за модулем Y, тобто залишок від ділення X/Y.

Синтаксис: #include <math.h>

```
double fmod(double x, double y);
```

Опис: fmod обчислює X за модулем Y (тобто залишок F, що задовольняє рівності X=IY+F для цілого I та 0<=F<Y). Функція fmod повертає залишок F, де X=IY+F (як описано вище). При Y=0 fmod також повертає 0.

Див. також ceil, floor, modf.

fopen

Функція: відкриває потік.

Синтаксис: #include <stdio.h>

```
FILE * fopen(char * filename, char * type);
```

Опис: функція fopen відкриває файл, іменований параметром filename та пов'язує його з відповідним потоком stream. Функція fopen як результат повертає адресний покажчик, який буде ідентифікувати поток stream в подальших операціях. При успішному завершенні fopen повертає покажчик на відкритий потік stream. У випадку помилки функція повертає нуль (NULL). Див. також creat, fclose, fwrite, setmode і т.д.

Приклад:

```
/* Програма створює дубль AUTOEXEC.BAT */
#include <stdio.h>
int main(void)
 {
FILE *in, *out;
if((in = fopen("\\AUTOEXEC.BAT","rt"))==NULL)
{
 fprintf(stderr, "Не можу відкрити файл \n");
return(1);
}
 if((out = fopen("\\AUTOEXEC.BAK","wt"))==NULL)
{
fprintf(stderr, "Не можу відкрити вихідний файл \n");
return(1);
}
while( ! feof(in) ) fputc(fgetc(in), out);
fclose(in);
fclose(out); return 0;
}
```

fprintf

Функція: здійснює форматоване виведення у потік.

Синтаксис:

```
#include <stdio.h>
int fprintf(FILE *stream,
            const char *format[, argument,...]);
```

Опис: fprintf отримує набір аргументів, по одному для кожної специфікації формату, та виводить дані у потік. Кількість аргументів повинна збігатися з кількістю специфікацій. Опис специфікацій формату наведено в описі функції printf(). fprint() повертає кількість виведених байтів. При появі помилки повертається EOF. Див. також cprintf, fscanf. printf, putc, sprintf.

Приклад:

```
#include<stdio.h>
 int main(void)
```

```
{
 FILE *stream;
 int i=100;
 char c = 'C';
 float f= 1.234;
 stream = fopen("X.Y", "w+");  /* відкрити файл для зміни */
 fprintf(stream,"%d %c %f ",i,c,f);  /* вивести у файл дані */
 fclose(stream);                     /* закрити файл */
 return 0;
}
```

fputc

Функція: виводить символ у потік.

Синтаксис: #include <stdio.h>
int fputc(int c, FILE * stream);

Опис: функція fputc виводить символ с у потік stream. У випадку успіху повертається символ с , помилки – EOF. Див. також fgetc, putc.

Приклад:

```
#include<stdio.h>
 int main(void)
 {
 char msg[] = "абвгдежзиклм.. ";
 int i=0;
 while(msg[i])
 {
  fputc(msg[i ], stdout);
 i++;
  } return 0;
 }
```

fputchar

Функція: виводить символ у потік stdout.

Синтаксис: #include <stdio.h>
int fputchar(int c);

Опис: функція fputchar виводить символ с у потік stdout. Функція fputchar(c) визначається як fputc(c, stdout). Див. також fgetchar,putchar.

Приклад:

```
#include<stdio.h>
 int main(void)
 {  char msg[] = "Message";
```

```
int i=0;
while(msg[i])
{
fputchar(msg[i]):
i++;
}
 return 0;
 }
```

fputs

Функція: виводить рядок символів у потік.

Синтаксис: #include <stdio.h>

int fputs(char * string, FILE * stream);

Опис: функція fputs копіює рядок, обмежений нульовим байтом, у потік stream. Вона не приєднує в кінець рядка символ переходу на новий рядок та не виводить нульовий символ. При успішному завершенні fputs повертає останній виведений символ. Див. також fgets, gets, puts.

Приклад:

```
#include<stdio.h>
int main(void)
{
/* вивести рядок у поток */
 puts( "вивести рядок у поток ", stdout);
return 0;
}
```

fread

Функція: зчитує дані з потоку.

Синтаксис: #include <stdio.h>

size_t fread(void *ptr, size_t size, size_t n, FILE * stream);

Опис: функція fread зчитує n елементів даних, кожний довжиною size байтів, з потоку stream у блок з адресним покажчиком ptr. Загальна кількість байтів введення дорівнює n × size. Див. також fopen, fwrite, printf, read.

Приклад:

```
#include<string.h>
#include<stdio.h>
int main(void)
FILE *stream;
char msg[]="Тест";
char buf[20];
```

```
if (stream=fopen("ccc.eee,"w+"))==NULL)
{
fprintf(stderr,"Не можу відкрити файл \n");
return 1;
}
fwrite(msg, strlen(msg)+1,1, stream);  /*вивести у файл дані */
fseek(stream, SEEK_SET,0);          /*перейти на початок файлу*/
fread(buf, strlen(msg)+1,1, stream);     /*прочитати дані*/
printf("%s\n", buf);
fclose(stream);
return 0;
}
```

free

Функція: звільняє пам'ять, виділену під блок.

Синтаксис: `#include<alloc.h>`

`void free(void *block);`

Опис: free звільняє блок пам'яті, виділений функціями calloc, malloc або realloc. Див. також calloc, malloc, realloc, strdup.

Приклад:

```
#include<string.h>
#include<stdio.h>
#include<alloc.h>
int main(void)
{
char *str;
str = malloc(10);            /* виділити пам'ять під рядок */
strcpy(str, "Hello");         /* скопіювати в рядок "Hello" */
printf("Рядок: %s\n",str);  /* вивести рядок */
free(str);                   /* звільнити пам'ять */
return 0;
}
```

freopen

Функція: пов'язує з потоком новий файл.

Синтаксис: `#include <stdio.h>`

`FILE * freopen(char * filename, char * mode, FILE "stream);`

Опис: функція freopen заміняє вказаним файлом відкритий поток stream. Функція freopen закриває файл, пов'язаний зі stream, незалежно від файлу відкриття. Її можна використовувати для зміни потоків, пов'язаних з stdin,

stdout або stderr. При успішному завершенні freopen повертає покажчик на відкритий поток stream. У випадку помилки функція повертає нуль (NULL). Див. також fclose, fdopen, fopen, open, setmode.

Приклад:

```
#include<stdio.h>
 int main(void)
{
/*перенаправити станд. виведення у файл */
if (freopen("OUTPUT.TTT","w", stdout) == NULL);
fprintf(stderr, "Помилка перенаправлення потоку \n");
/*це виведення буде здійснюватися у файл */
printf("Це виведення буде здійснюватися у файл \n");
/* закрити стандартне виведення */
close(stdout);
return 0;
}
```

fscanf

Функція: виконує форматоване введення з потоку.

Синтаксис: #include <stdio.h>

int fscanf(FILE * stream, char * format[, adress,...]);

Опис: fscanf() приймає посимвольно набір полів, що вводяться, скануючи їх із потоку. Потім кожне поле з потоку форматується відповідно до специфікації формату, яка передається в fscanf у вигляді покажчика на рядок format. Отримане в результаті цього поле fscanf запам'ятовується в аргументах, що передаються функції fscanf після параметру format. Кількість аргументів повинна збігатися з кількістю специфікацій формату. Опис специфікацій формату наведено в описі функції scanf().

Див. також atof, cscanf, fprintf, printf, scanf, sscanf.

Приклад:

```
#include<stdlib.h>
#include<stdio.h>
int main(void)
{ int i;
 printf("Введіть ціле число:");
/*ввести із стандартного потоку stdout ціле число*/
 if(fscanf(stdin,"%d",&i))  printf("Ціле дорівнює: %d\n",i);
 else  {    fprintf (stderr, "Помилка зчитування цілого\n");
   exit(1);  }
 return 0; }
```

fseek

Функція: встановлює покажчик файлу в потоці.

Синтаксис: #include <stdio.h>

int fseek(FILE * stream, long offset, int fromwhere);

Опис: fseek встановлює адресний покажчик файлу, що відповідає потоку stream, в нову позицію, яка має зміщення offset відносно місця у файлі, що визначається параметром fromwhere. Параметр fromwhere може мати одне з трьох значень 0, 1 або 2, що подані трьома символічними константами (файл stdio.h):

SEEK_SET (0) початок файлу;
SEEK_CUR (1) позиція поточного покажчика файлу;
SEEK_END (2) кінець файлу (EOF);

Функція fseek звільняє будь-який символ, записаний за допомогою функції ungetc. Див. також fgetpos, fopen, fsetpos, ftell, lseek.

Приклад:

```
#include <stdio.h>
 int main(void)
{
FILE *stream;
stream = fopen("M.EXT", "r");
print("filesize of M.EXT is %ld bytes\n", filesize(stream));
}
 long filesize(FILE *stream)
{
 long curpos, length;
curpos = ftell(stream);   /*збереження положення покажчика */
fseek(stream, OL, SEEK_END); /* перейти в кінець файлу */
length = ftell(stream);      / *отримати положення покажчика */
fseek(stream, curpos, SEEK SET); /* минуле положення */
return(length);
}
```

fwrite

Функція: записує дані у потік.

Синтаксис: #include <stdio.h>

size_t fwrite(void * ptr, size_t size, size_t. n, FILE * stream);

Опис: fwrite додає n елементів даних, кожне величиною size байтів у даний вихідний потік. Дані записуються з ptr. Загальна кількість виведених байтів дорівнює n*size. ptr повинен бути оголошеним як покажчик на деякий

об'єкт. При успішному завершенні fwrite повертає число виведених елементів (не байтів). Див. також fopen, fread.

Приклад:

```
#include<stdio.h>
struct mystruct {
    int i;
    char ch;
}
int main(void)
{
FILE *stream;
struct mystruct s;
if((stream = fopen("test.cpp", "wb"))==NULL)
{
fprintf(stderr, "не можу відкрити файл \n");
return 0:
}
s.i = 0;
s.ch = 'A';
fwrite(&s,sizeof(s), 1,stream);  /* виведення у файл */
fclose( stream);
return 0; }
```

gcvt

Функція: перетворення числа з плаваючою точкою у символьний рядок.

Синтаксис: #include <stdlib.h>

char * gcvt(double value, int ndig, char *buf);

Опис: gcvt перетворює value в рядок символів в коді ASCII, обмежений нулем, та запам'ятовує рядок в buf. gcvt , повертає адресу рядка.

Див. також ecvt, fcvt.

getarccoords

Функція: видає координати останнього звертання до функції arc.

Синтаксис: #include <graphics.h>

void far getarccoords(struct arccoordstype far *arccoords);

Опис: getarccoords заповнює структуру arccoordstype, на яку вказує arccoords, інформацією про останній виклик arc.

```
struct arccoordstype
{
int X, Y;
```

```
    int xstart, ystart, xend, yend;
    };
```
Приклад:
```
  #include <graphics.h>
  #include <stdlib.h>
  #include <stdio.h>
  #include <conio.h>
   int main(void)
  {
  int graphdriver = DETECT, gmode, errorcode;
  struct srccoordstype arcinfo;
  int midx,midy;
  int stangle = 45,endangle = 270;
  char sstr[80],estr[80];
  initgraph(&graphdriver; &gmode, "");
  errorcode = graphresult();
  midx = getmaxx() / 2;
  midy = getmaxy() / 2;
  setcolor(getmaxcolor( 1));
  arc(midx, midy, stangle, endangle , 100);
  getarccoords((&arcinfo);
  sprintf(sstr, "%d, %d", arcinfo. xstart, arciofo.ystart);
  sprintf(sstr, "%d, %d", arcinfo .xend, arcinfo.yend);
  outtextxy(arcinfo.xstart, arcinfo.ystart, estr);
  outtextxy(arciafo.xend, arcinfo.xend, estr);
  getch();
  closegraph();
  return 0 ;
  }
```

getc

Функція: вводить символ з потоку.

Синтаксис: #include <stdio.h>

int getc(FILE *stream);

Опис: getc–це макрокоманда, що отримує наступний за порядком символ з вхідного потоку stream та збільшує покажчик у потоці на 1. При успішному завершенні функція getc повертає зчитаний символ після перетворення його в int.

Приклад:
```
  #include<stdio.h>
```

```
int main(void)
{
 char ch;
printf( "Введіть символ :");
ch = getc(stdin);
printf("Був введений символ '%c'\n",ch);
return 0;
}
```

getch

Функція: вводить символ з консолі без ехо-друку (відображення).

Синтаксис: #include<conio.h>

int getch(void);

Опис: getch зчитує один символ, безпосередньо з консолі, без виводу його на екран.

Приклад:

```
#include<conio.h>
#include<stdio.h>
int main(void)
{
int c;
c = getch();
if(!c) extended = getch();
if(extended) printf("Розширений символ\n");
else printf("Нерозширений символ\n");
return 0;
}
```

getchar

Функція: вводить символ з потоку stdin.

Синтаксис: #include <stdio.h>

int getchar(void);

Опис: getchar – макрокоманда, що вводить символ з потоку stdin. Вона визначена як getc(stdin).

Приклад:

```
#include<stdio.h>
int main(void)
{
char c;
while((c=getchar())!='\n')
printf("%c",c); return 0; }
```

getche

Функція: вводить символ з консолі та відображає його на екрані.

Синтаксис: #include<conio.h>

int getche(void);

Опис: getche зчитує один символ з консолі та одночасно відображає його у текстовому вікні на екрані.

Приклад:

```
#include<conio.h>
int main(void)
{
 char ch;
 printf( "Введіть символ:");
 ch = getche();
 printf("\n введений символ - '%c'\n",ch);
 return 0;
}
```

getcolor

Функція: повертає поточний колір рисунка.

Синтаксис: #include <graphics.h>

int far getcolor(void);

Опис: колір рисунка – це значення, в яке встановлюються пікселі, коли рисуються графічні зображення – лінії, кола тощо.

getdate

Функція: отримує системну дату.

Синтаксис: #include <dos.h>

void getdate(struct date * datep);

Опис: getdate заповнює структуру date (з покажчиком datep) системною інформацією про поточну дату. Структура date визначається так:

```
struct date
{
int da_year; /* рік */
char da_day; /* день місяця */
char da_mon; /* місяць (1= січень ... ) */ };
```

Приклад:

```
#include<dos.h>
#include<stdio.h> int main(void)
{
struct date d;
```

```
getdate(&d);
printf("рік :%d\n", d.da_year);
printf( "місяць: %d\n", d.da_mon);
printf("день: %d\n", d.da_day);
return 0;
}
```

getmaxx

Функція: повертає максимальну координату X екрана.

Синтаксис: #include <graphics.h>

```
int far getmaxx(void);
```

Опис: getmaxx повертає максимальне (відносно екрана) значення X для поточного драйвера та режиму.

getmaxy

Функція: повертає максимальну координату Y екрана.

Синтаксис: #include <graphics.h>

```
int far getmaxy(void);
```

Опис: getmaxy повертає максимальне (відносно екрана) значення Y для поточного драйвера та режиму.

getpixel

Функція: повертає колір заданої точки.

Синтаксис: #include <graphics.h>

```
unsigned far getpixel(int x, int y);
```

Опис: getpixel повертає колір точки (x,y).

gets

Функція: отримує рядок символів з потоку.

Синтаксис: #include<stdio.h>

```
char *gets(char *s);
```

Опис: gets зчитує рядок символів, що закінчується символом переводу рядка у змінну *s зі стандартного вхідного потоку stdin. Див. також cgets, fgets, fputs, getc, puts, scanf.

Приклад:

```
#include <stdio.h>
 int main(void)
{
char string[128];
gets(string);
printf("Рядок = '%s'\n.string);
}
```

getw

Функція: вводить з потоку ціле число.

Синтаксис: #include <stdio.h>

int gefw(FILE *stream);

Опис: getw повертає наступне ціле із потоку вказаного покажчиком. getw неможливо використовувати, коли потік відкрито у текстовому режимі.

getx

Функція: повертає координату x поточної графічної позиції.

Синтаксис: #include <graphics.h>

int far getx(void);

Опис: getx знаходить координату x поточної графічної позиції. Значення береться відносно поточної області.

gety

Функція: повертає координату y поточної графічної позиції.

Синтаксис: #include <graphics.h>

int far gety (void);

Опис: getx знаходить координату y поточної графічної позиції. Значення береться відносно поточної області.

gotoxy

Функція: позиціонує курсор у текстовому вікні.

Синтаксис: #include<conio.h>

void gotoxy(int x, int y);

Опис: gotoxy переміщує курсор у текстовому вікні у вказану позицію.

Приклад:

```
#include<conio.h>
int main(void)
{
clrscr();
gotoxy(35,12);
cprintf("H E L L 0 ");
getch();
return 0; }
```

hypot

Функція: обчислює гіпотенузу прямокутного трикутника.

Синтаксис: #include<math.h>

double hypot(double x, double y);

Опис: функція hypot обчислює значення z, де $z^2=x^2+y^2$ та $z>=0$

Приклад:

```
#include <stdio.h>
#include <math.h>
int main(void)
{
double result;
double x = 3.0; double y = 4.0;
result = hypot(x,y);
printf("Гіпотенуза = %f\n", result);
return 0;
}
```

imag

Функція: повертає уявну частину комплексного числа.

Синтаксис: #include<complex.h>

double imag(complex x);

Опис: комплексне число складається з двох чисел, функція imag повертає одне з них – уявну частину.

Приклад:

```
#include<stream.h>
#include<complex.h>
int main(void)
{
double x=3.1,y=4.2;
complex z = complex(x,y);
cout << "z = " << z << "\n";
cout << "Дійсна частина = " <<; real(z) << "\n";
cout << "Уявна частина = " << imag(z) << "\n";
cout << "Комплексне спряження = " << conj(z) << "\n";
return 0;
}
```

initgraph

Функція: ініціалізує графічну систему.

Синтаксис: #include <graphics.h>

void far initgraph(int far *graphdriver, int far *graphmode,
char far *pathtodriver)

Опис: initgraph ініціалізує графічну систему шляхом завантаження графічного драйвера з диска та переводить систему у графічний режим. Після

звертання до initgraph *graphdriver встановлюється у поточний графічний драйвер, а *graphmode – у поточний графічний режим.

Приклад:

```
#include <graphics.h>
#include<stdlib.h>
#include<stdio.h>
 #include <conio.h>
 int main(void)
{  int graphdriver = DETECT, gmode, errorcode;
   initgraph(&graphdriver, &gmode, "");
   errorcode = graphresult();
   if( errorcode != grOk)
   {
    printf("Помилка: %s\n",grapherrormessage( errorcode));
    getch(); exit(1);  /* завершення з кодом помилки */
   }
line(0,0,getmaxx(),getmaxy());
closegraph();
return 0; }
```

insline

Функція: вставляє порожній рядок у текстовому вікні.

Синтаксис: #include<conio.h>

```
void insline(void);
```

Опис: insline вставляє у поточному текстовому вікні порожній рядок. Усі нижчі рядки прокручуються вниз. Найнижчий рядок зникає. insline використовується у текстовому режимі.

isalpha

Функція: макрокласифікації символів (alphabetic).

Синтаксис: #include <ctype.h>

```
int isalpha(int ch);
```

Опис: isalpha – макрокоманда, що класифікує цілі значення в коді ASCII шляхом перегляду таблиці. При true вона повертає ненульове значення та 0 при false. Вона визначена тільки у випадку, якщо isascii(ch) дорівнює true або ch = EOF.

Приклад:

```
#include<ctype.h>
#include<stdio.h>
 int main(void)
```

```
{
char c = 'C';
if(isalpha(c))
printf("%c - alphabetic\n",c);
else
printf("%c - isn't alphabetic\n",c);
return 0;
}
```

isascii

Функція: макрокласифікація символів (ASCII).

Синтаксис: #include <ctype.h>
int isascii(int ch);

Опис: isascii – це макрокоманда, яка класифікує цілі значення в коді ASCII переглядом таблиці. isascii визначена для всіх цілих значень. isascii повертає ненульове значення, якщо молодший байт ch лежить у діапазоні від 0 до 127 (0x00-0x7F).

Приклад:

```
#include<ctype.h>
#include<stdio.h>
 int main(void)
{
 char c = 'C';
if(isascii(c))
printf("%c - is ascii\n",c);
else
printf("%c - isn't ascii\n",c);
return 0;
}
```

islower

Функція: макрокласифікація символів (нижній регістр).

Синтаксис: #include <ctype.h>
int islower(int ch);

Опис: islower – це макрокоманда, яка класифікує цілі значення в коді ASCII переглядом таблиці. Вона визначена тільки у тому випадку, якщо isascii(ch) дорівнює true чи ch = EOF.

Приклад:

```
#include<ctype.h>
#include<stdio.h>
int main(void)
```

```
{
char c = 'C';
if (islower(c))
printf("%c – буква у нижньому регістрі \n",c);
else printf("%c – Не є буквою у нижньому регістрі \n",c);
return 0;
}
```

isupper

Функція: макрокласифікації символів (верхній регістр).

Синтаксис: #include <ctype.h>

int isupper(int ch);

Опис: isupper – це макрокоманда, яка класифікує цілі значення в коді ASCII переглядом таблиці. При true вона повертає ненульове значення та 0 при false. Вона визначена лише у тому випадку, якщо isascii(ch) дорівнює true або ch = EOF.

Приклад:

```
#include<ctype.h>
#include<stdio.h>
 int main(void)
{
 char c = 'C';
if(isupper(c))
printf("%c – буква у верхньому регістрі \n",c);
else printf("%c – не є буквою у верхньому регістрі \n",c);
return 0;
}
```

itoa

Функція: перетворює ціле значення у рядок символів.

Синтаксис: #include<stdlib.h>

char *itoa(int value, char * string, int radix);

Опис: функція перетворює значення value у рядок символів, що закінчується нулем, та записує результат у параметр string. Для функції itoa значення value – ціле. Параметр radix визначає базис, який буде використаний при перетворенні значення value; він лежить між 2 та 36 (включно). Якщо значення value від'ємне, і значення radix – 10, перший символ рядка string – знак мінус (-).

Приклад:

```
#include<stdlib.h>
#include<stdio.h>
```

```
int main(void)
{
 int number = 12345;
char string[25];
itoa(number, string, 10);
printf("Ціле: %d, рядок: %s\n", number, string);
return 0;
}
```

line

Функція: рисує лінію між двома вказаними точками.

Синтаксис: #include <graphics.h>

void far line(int x1, int y1, int x2, int y2);

Опис: line рисує лінію, використовуючи поточні колір, тип та товщину ліній, між двома точками, визначеними як (x1, y1) та (x2, y2), не змінюючи поточну позицію (CP).

Приклад:

```
#include <graphics.h>
#include <stdlib.h>
 #include <stdio.h>
#include <conio.h>
 int main(void)
{
int graphdriver = DETECT, gmode, errorcode;
int xmax,ymax;
initgraph(&graphdriver, &gmode, "");
errorcode = graphresult();
if(errorcode != grOk)
{
printf("Помилка: %s\n", grapherrormessage (errorcode));
exit(1);
}
setcolor(getmaxcolor( 1));
xmax = getmaxx();
ymax = getmaxy();
line(0, 0, xmax, ymax);  /* нарисувати діагональну лінію */
closegraph();
return 0;
}
```

linerel

Функція: рисує лінію, з визначенням довжини, від поточної позиції (CP).
Синтаксис: #include <graphics.h>
void far linerel(int dx, int dy);
Опис: linerel рисує лінію від CP до точки, що знаходиться на (dx, dy) від CP. CP переміститься на CP + (dx,dy).

lineto

Функція: рисує лінію від поточної позиції (CP) у точку з координатами (x,y).
Синтаксис: #include <graphics.h>
void far lineto(int x, int y);
Опис: lineto рисує лінію від CP до точки з координатами (x,y), після чого переносить (CP) в (x,y).

log

Функція: обчислює натуральний логарифм ln(x).
Синтаксис: версія для дійсних та комплексних чисел:

```
#include<math.h>              #include<complex.h>
double log(double x);         complex log(complex x);
```

Опис: log обчислює натуральний логарифм від x. Натуральний логарифм для комплексних чисел визначається як : log(z) = log(abs(z)) + i arg(z)
Див. також complex, exp, log10, sqrt.
Приклад:

```
#include<math.h>
#include<stdio.h>
int main(void)
{
double result;
double x = 8.6872;
result = log(x);
printf("Натуральний логарифм від %f дорівнює %f \n", x, result);
return 0;
}
```

log10

Функція: обчислює десятковий логарифм lg(x).
Синтаксис: версія для дійсних та комплексних чисел:

```
#include<math.h>              #include<complex.h>
double log10(double x);       complex log10( complex x);
```

Опис: log10 обчислює десятковий логарифм від x. Десятковий логарифм комплексного числа визначається як: log10(z) = log(z)/log(10)
Див. також complex, exp, log.

lowvideo

Функція: встановлює низьку інтенсивність символів виведення.
Синтаксис: #include<conio.h>
void lowvideo(void);
Опис: lowvideo обнуляє біт інтенсивності у полі атрибутів поточного кольору символів. Див також highvideo, normvideo, textattr, textcolor.
Приклад:

```
#include<conio.h>
 int main(void)
{ clrscr();
  lowvideo();
  cprintf( "Текст низької інтенсивності");
  highvideo();
  gotoxy(1,2);
  cprintf("Текст високої інтенсивності");
  return 0;
}
```

lseek

Функція: переміщує покажчик читання-записування у файлі.
Синтаксис: #include <io.h>
long lseek(int handle, long offset, int fromwhere).
Опис: lseek встановлює покажчик файлу, пов'язаний з параметром handle, у нову позицію, яка знаходиться на offset байтів відносно параметра fromwhere. Параметр fromwhere може мати одне з трьох значень 0, 1 або 2, що подані трьома символічними константами (в stdio.h):
SEEK_SET (0) – початок файлу
SEEK_CUR (1) – поточна позиція
SEEK_END (2) – кінець файлу
Функція lseek повертає значення зміщення нової позиції покажчика, вимірюваної у байтах від початку файлу.

ltoa

Функція: перетворює ціле long значення в рядок символів.
Синтаксис: #include<stdlib.h>
char *ltoa(long value, char *string, int radix);

Опис: перетворює значення value в рядок символів, що закінчується нулем, та записує результат у параметр string. Для функції ltoa значення value – довге ціле (тип long). Параметр radix визначає базис, який буде використаний при перетворенні значення value; він лежить між 2 та 36 (включно). Якщо значення value від'ємне, і значення radix 10, перший символ рядка string – знак мінус (-). Функція повертає покажчик на рядок string.

Див. також itoa, ultoa.

malloc

Функція: виділяє область динамічної пам'яті.

Синтаксис: #include<stdlib.h>

#include<alloc.h>

void *malloc(size_t size);

Опис: забезпечує виділення блоку пам'яті розміром size байт. Вона дозволяє програмі виділяти пам'ять за необхідності та в такому об'ємі, як того потребує ситуація. Динамічна область пам'яті є доступною для динамічного розташування блоків пам'яті змінної довжини ("дерева", "списки" тощо). malloc повертає покажчик на блок виділеної пам'яті. Функція є доступною у UNIX- системах. Див також calloc, farmalloc, free, realloc.

Приклад:

```
#include<stdio.h>
#include<string.h>
#include<alloc.h>
#include<process.h>
 int main(void)
{
char *str;
if((str = malloc(10)) == NULL)  /*виділити пам'ять під рядок*/
{
 printf( "Недостатньо пам'яті \n");
 exit(1);    /* завершення */
}
strcpy(str, "Hello");
printf("Рядок: %s\n",str);
free(str);    /* звільнити пам'ять */
return 0;
}
```

max

Функція: повертає найбільше з двох значень.

Синтаксис: #include<stdlib.h>

(type) max(a,b);

Опис: макрокоманда проводить порівняння двох значень одного типу та повертає найбільше з них.

min

Функція: повертає найменше з двох значень.

Синтаксис: #include<stdlib.h>

(type)min(a,b);

Опис: макрокоманда проводить порівняння двох значень одного типу та повертає найменше з них.

modf

Функція: розділяє число типу double на цілу та дробову частину.

Синтаксис: #include<math.h>

double modf(double x, double *iptr)

Опис: modf розділяє число типу double на цілу та дробову частину. Ціла частина розміщується у iptr, а дробова частина повертається як значення. Див. також fmod, ldexp.

Приклад:

```
#include<math.h>
#include<stdio.h>
int main(void)
{
 double fraction, integer;
double number = 154321.679;
fraction = modf(number,&integer);
printf("Ціла та дробова частини числа %lf = %f й %f",
        number,   integer, fraction);
return 0;
}
```

moverel

Функція: переміщує поточну граф-позицію (CP) на заданий проміжок довжини.

Синтаксис: #include <graphics.h>

void far moverel (int dx,int dy);

Опис: moverel переміщує позицію (CP) на dx точок по осі X та на dy точок по осі Y. Див. також moveto.

movetext

Функція: копіює текст на екрані з одного прямокутника в інший.

Синтаксис: #include<conio.h>

```
int movetext(int left, int top, int right, int bottom,
                    int destleft, int desttop);
```

Опис: movetext копіює вміст прямокутника на екрані (аргументи left, top, right та bottom), в новий прямокутник, що має такі самі розміри. Лівий верхній кут цього прямокутника визначатиметься параметрами destleft, desttop. movetext працює у текстовому режимі. Див. також gettext, puttext.

moveto

Функція: переміщує поточну позицію (CP) у точку з координатами (x,y).

Синтаксис: #include <graphics.h>

```
void far moveto(int x,int y);
```

Опис: moveto переміщує поточну позицію (CP) у позицію з координатами (x,y) у поточній області. Див також: moverel.

Приклад:

```
#include <graphics.h>
#include <stdlib.h>
#include <stdio.h>
 #include <conio.h>
 int main(void)
{ int graphdriver = DETECT, gmode, errorcode;
  char msg[80];
  initgraph(&graphdriver, &gmode, "");
  errorcode = graphresult();
  moveto(25,35);
/*перемістити позицію(CP)в точку (25,35) */
putpixel(getx(),gety(),getmaxcolor());
sprintf(msg," (%d,%d)",getx(),gety());
outtextxy(20,30,msg);
moveto(100,100);   /*нарисувати відносно поточної позиції*/
putpixel(getx(),gety(),getmaxcolor());
sprintf(msg," (%d,%d)",getx(),gety());/* повідомлення */
outtext(msg);
getch();
closegraph();
return 0; }
```

normvideo

Функція: встановлює нормальну інтенсивність символів виведення.

Синтаксис: #include<conio.h>
void normvideo(void);

Опис:normvideo встановлює початкове значення біту інтенсивності у полі атрибутів кольору символів. Див. також highvideo, lowvideo, textattr, textcolor.

nosound

Функція: відключає вбудований динамік.

Синтаксис #include<dos.h>
void nosound(void);

Опис: відключає вбудований динамік, що був включений функцією sound.

Приклад: /*Звук частотою 7 Гц протягом 5 сек. Можливо динамік вашого комп'ютера не зможе генерувати такий звук. Тоді змініть параметр або динамік чи, може, комп'ютер?*/

```
int main(void)
{
sound(7);
delay(5000);
nosound();
return 0;
}
```

_open, open

Функції: відкривають файл для читання або записування.

Синтаксис: #include <fcntl.h>
int _open(const char *filename, int of lags);
#include <sys\stat.h>
int open(const char *filename, int access [, unsigned mode]);

Опис: _open відкриває файл, (ім'я у параметрі filename), та підготовлює його для операцій читання та/або записування, що визначаються параметром oflags. Файл відкривається у режимі, що визначається fmode. Для функції _open параметр oflags, залежно від DOS, може набувати значень з переліку: O_RDONLY, O_WRONLY, O_RDWR, O_NOINHERIT, O_DENYALL, 0_DENYWRITE, O_DENYREAD, 0_DENYNONE(визначені у файлі fcntl.h).

Опис: open відкриває файл, (ім'я – параметр filename), та підготовлює його до операцій читання / записування, залежно від параметру access. Для створення файлу у звичному режимі, ви можете присвоїти відповідне значення _fmode. Для функції open параметр access конструюється шляхом логічного побітового додавання прапорців-міток, що перераховуються у двох

списках. З першого списку може використовуватися лише один прапорець; решта з них можуть застосовуватися у будь-яких логічних комбінаціях. Список прапорців 1:

O_RDONLY – тільки для читання.

O_WRONLY – тільки для записування.

O_RDWR – для читання та записування.

Список 2: прапорці доступу: O_APPEND , O_CREAT, O_TRUNC, O_EXCL , O_BINARY, O_TEXT. Константи (O_...) визначені у файлі fcntl.h. При успішному завершенні open повертає ціле невід'ємне число handle – логічний номер відкритого файлу. Покажчик файлу встановлюється на його початок. Див. також close, fopen, read, write.

outtext

Функція: відображає рядок у вікні екрану.

Синтаксис: #include <graphics.h>

void far outtext(char far *textstring);

Опис: outtext відображає рядок тексту у вікні, використовуючи поточні встановлені параметри вирівнювання тексту, а також поточні шрифт, напрямок та розмір. outtext виводить текст у поточну позицію. Для забезпечення сумісності кодів підтримки при використанні різних шрифтів використовуйте textwidth та textheight для визначення розмірів рядка.

outtextxy

Функція: відображає рядок у вказаній області.

Синтаксис: #include <graphics.h>

void far outtextxy(int x,int y, char far *textstring);

Опис: outtextxy відображає рядок тексту у вікні екрану в заданій позиції (x,y), використовуючи поточні встановлені параметри вирівнювання тексту, а також поточні шрифт, напрямок та розмір. Для забезпечення сумісності кодів підтримки при використанні різних шрифтів використовуйте textwidth та textheight для визначення розмірів рядка. outtext працює у графічному режимі. Див. також outtext, textheight, textwidth.

pieslice

Функція: рисує та зафарбовує сектор кола.

Синтаксис: #include <graphics.h>

void far pieslice(int x,int y,int stangle, int endangle,int radius);

Опис: pieslice рисує та зафарбовує сектор кола з центром в точці (x,y) та радіусом radius. Сектор рисується від кута stangle до кута endangle. Кут для pieslice вимірюється у градусах та відраховується проти годинникової стрілки, де 0 градусів – 3 години на циферблаті, 90 градусів – 12 годин тощо. Див. також fillellipse, sector, setfillstyle.

poly

Функція: конструює поліном із заданих аргументів.

Синтаксис: #include<math.h>

double poly(double x, int degree, double coeff[]);

Опис: poly конструює поліном від x степені degree із коефіцієнтами coeff[0], coeff[1], coeff[n]. Наприклад, якщо n=4, поліном буде таким: coeff[4]x^4 + coeff[3]x^3 + coeff[2]x^2 + coeff[1]x + coeff[0]. Функція poly повертає обчислене для даного x значення поліному.

Приклад:

```
#include<stdio.h>
#include<math.h>
/* поліном: x^3 - 2*x^2 + 5*x - 1 */
int main(void)
{
double array[] = {-1.0, 5.0,-2.0, 1.0};
double result;
result = poly (2.0,3, array);
printf("Поліном x^3-2*x^2+5*x-1  при x=2.0 дорівнює:
%lf\n", result);
return 0;
}
```

pow

Функція: обчислює x в степені y.

Синтаксис: дійсна версія #include<math.h>

double pow(double x, double y);

комплексна версія #include<complex.h>

complex pow(complex x,complex y); complex pow(complex x,double y); complex pow(double x,complex y);

Опис: pow обчислює вираз x в степені y. Див. також complex, exp, sqrt.

printf

Функція: проводить форматоване виведення в поток stdout.

Синтаксис: #include<stdio.h>

int printf(const char *format [,argument, ...]);

Опис: printf отримує набір аргументів, застосовує до кожного відповідну специфікацію формату з рядка format та виводить відформатовані дані в stdout. Кількість аргументів повинна відповідати кількості специфікацій формату у format. Рядок формату – символьний рядок, що складається з двох типів об'єктів – просто символи та специфікації перетворення. Прості

символи переносяться на виведення без будь-яких змін. Специфікації формату для функцій сімейства printf () мають таку форму:

%[прапорці] [ширина] [точність] [F | N | h | l | L] type

Кожна специфікація починається з символу % та закінчується певним символом, що задає подання:

d – десяткове подання ;

o – вісімкове подання без знаку;

x – шістнадцятіркове подання без знаку;

h – десяткове подання без знаку;

c – параметр як один символ ;

s – параметр-рядок;

є – E - формат подання величин float та double;

f – звичайне подання величин float та double;

Якщо після символу % не йде символ подання, то цей символ друкується. Після нього йдуть ознаки в переліченому вище порядку, а квадратні дужки означають необов'язковість застосування у кожному з організованих викликів. Символи формату єдині для сімейства функцій printf(). Кожна функція повертає кількість байтів виведення. Робота функції зупиняється, коли вичерпується інформація з керуючого рядка або коли виявляється невідповідність вхідної інформації заданим специфікаціям. У випадку помилки printf() повертає EOF.

Приклад:

```
#include<stdio.h>            /*використання printf()*/
#define square (x) ((x)* (x))
#define pi 3. 1415926
int main(void)
{
float x=5. 9;
printf ("\n This program demonstrates printf ("...");
printf(" Значення квадрату змінної x =%10. 4f", square (x));
printf("Значення числа пі =%10. 4f", pi);
printf( " Значення i* пі =%10. 4f", i* pi);
return 0;
}
```

putc

Функція: виводить символ у потік.

Синтаксис: #include<stdio. h>

int putc (int c, FILE* stream);

Опис: putc виводить символ c у вказаний вихідний потік stream. При успішному завершенні функція putc повертає символ c.

putch

Функція: виводить символ на екран.

Синтаксис: #include<conio. h>

int putch (int c);

Опис: putch виводить символ с у поточне текстове вікно. Ця функція виводить символ у текстовому режимі безпосередньо на консоль, putch не перетворює символ переводу рядка (\n) у пару: переведення рядка, повертання каретки. Див. також cprintf, cputs, getch, pute, putchar.

putchar

Функція: виводить символ у потік stdout.

Синтаксис: #include <stdio. h>

int putchar (int c);

Опис: putchar можна цілком замінити на putc(ch,stdout) (працює аналогічно). При успішному завершенні putchar повертає виведений символ с. Див. також fputchar, printf, putc, putch, puts.

Приклад:

```
#include<stdio. h>
/* визначення деяких символів для рисування рамок*/
#define LEFT_TOP 0xDA
#define RIGHT_TOP 0xBF
#define HORIZ 0xC4
#define VERT 0xB3
#define LEFT_BOT 0xC0
#define RIGHT_BOT 0xD9
int main (void)
 {
char i, j;
putch (LEFT_TOP);   /* верхня частина рамки*/
for (i=0; i<10; i+ 4-)
putch ( HORIZ);
putch (RIGHT_TOP);
for (l=0; i<4; i++) {   /* центральна частина*/
putch (VERT);
for (j=0; j<10; j++)
putch(' ');
putch (VERT);
putch ('\n');}
putch (LEFT_BOT);   /* нижня частина рамки*/
```

```
far (i=0; i<10; i++)
putch (HORIZ);
putch (RIGHT_BOT);
putch ('\n');
return 0;
}
```

putimage

Функція: виводить на екран бітове зображення.

Синтаксис: #include <graphics. h>

void far putimage (int left, int top, void far * bitmap, int op);

Опис: putimage розміщує бітове зображення, збережене за допомогою getimage, знову на екран, де лівий верхній кут зображення знаходиться в точці (left, top). bitmap вказує область пам'яті, де зберігається потрібне зображення. Параметр int op є складеним оператором, що визначає обчислення кольору для кожної точки (pixel) екрану.

Див. також getimage, imagesize, putpixel.

Приклад:

```
#include <graphics. h>
#include <stdlib. h>
#include <stdio. h>
#include <conio. h>
#define ARROW_SIZE 10
void draw_arrow (int x, int y);
int main (void)
{
int graphdriver = DETECT, gmode, errorcode;
void* arrow;
int x, y, maxx; unsigned int size;
initgraph(&graphdriver, &gmode, "");
errorcode = graphresult();
maxx = getmaxx();
x =0;
y = getmaxy()/2;
draw_arrow (x, y);
/* нарисувати об'єкт, що переміщується*/
/* визначити розмір пам'яті для зберігання зображення*/
size=imagesize(x, y-ARROW_SIZE, x+ (4*ARROW_SIZE),
                    y+ ARROW_SIZE);
arrow = malloc (size);  /* виділити пам'ять*/
```

```
getimage(x, y-ARROW_SIZE, x+ (4* ARROW_SIZE),
                    y+ ARROW_SIZE, arrow);
while (! kbhit ())  /* продовжити до натиснення клавіші*/
{
/* стерти старе зображення*/
putimage (x, y-ARROW_SIZE, arrow, XOR_PUT);
x+ = ARROW_SIZE;
if (x >= maxx) x - 0;
putimage (x, y-ARROW_SIZE, arrow, XOR_PUT); }   /* зображ.* /
free ( arrow);
closegraph();
return 0;
}
void draw_arrow (int x, int y) {  /* стрілка на екрані*/
moveto (x, y);
linerel (4* ARROW_SIZE, 0):
linerel (-2 *ARROW_SIZE, -1* ARROW_SIZE);
linerel (0, 2* ARROW_SIZE);
linerel (2* ARROW_SIZE, -1* ARROW_SIZE);
}
```

putpixel

Функція: виводить піксель у задану точку екрану.

Синтаксис: #include <graphics. h>

```
void far putpixel(int x, int y, int color);
```

Опис: putpixel відображає точку color та з координатами (x, y).

Див. також getpixel, putimage.

Приклад:

```
#include <graphics. h>
#include <stdlib. h>
#include <stdio. h>
#include <conio. h>
#include<dos. h>
#define PIXEL_COUNT 1000
#define DELAY_TIME 100
int main (void)
{
int graphdriver = DETECT, gmode, errorcode;
int x, y, l, color, maxx, maxy, maxcolor, seed;
initgraph (&graphdriver, &gmode, "");
```

```
errorcode = graphresult();
maxx = getmaxx();
maxy = getmaxy();
maxcolor = getmaxcolor()+ 1;
while (! kbhit() {
seed = random (32767);  /* генератор випадкових чисел*/
srand (seed);
for ( i=0; i<PIXEL_COUNT; i++)
{
x = random (inaxx);
y = random (maxy);
color = random (maxcolor);
putpixel (x, y, color);
}
 delay (DELAY_TIME);
srand (seed);
for (i=0; i<PIXEL_COUNT; i++)
{
x = random (maxx);
y = random (maxy):
color = random (maxcolor):
if (color = = getpixel (x, y);
putpixel (x, y, 0);
}
closegraph();
return 0;
}
```

puts

Функція: виводить рядок в потік stdout.

Синтаксис: #include<stdio. h>

int puts (const char* s);

Опис: puts копіює рядок символів з нульовим кінцем у стандартний вихідний потік stdout, причому додає в кінець символ переходу на новий рядок.

Див. також cputs, fputs, gets, printf, putchar.

Приклад:

```
#include <stdlib. h>
#include <string. h>
#include <conio. h>
```

```
/*програма друкує останні n рядків, що вводяться*/
main (void)
{
struct Str
  {
   char string [256]:
  };
struct Str strings [100];
int i, j, n;
clrscr;
printf("Введіть рядки (натисніть TAB чи
                ENTER для закінчення) \n");
i=0;
while (1)
{
gets (strings [i]. string);
if (strings[i]. string[0] ==9) break;
i++; }
printf ("Введіть останні n рядків, що виводитимуться\n");
scanf ("%i", &j);
for (n=i-j; n<i; n++)
puts (strings[n]. string);
getch();
}
```

puttext

Функція: копіює текст з пам'яті на екран, у текстовому режимі.

Синтаксис: #include<conio. h>

int puttext (int left, int top, int right, int bottom, void* source);

Опис: puttext розміщує вміст області пам'яті, адресованої параметром source, у прямокутник на екрані (параметри left, top, right, bottom). Усі координати є абсолютними координатами екрану, а не текстового вікна. Лівий верхній кут екрану має координати (1, 1). Див. також gettext, movetext, window.

putw

Функція: виводить у потік ціле значення.

Синтаксис: #include <stdio. h>

int putw (int w, FILE* stream);

Опис: putw виводить ціле у вказаний потік. Дана функція не викликає та не очікує ніякого спеціального вирівнювання у файлі. При успішному завершенні putw повертає виведене ціле w. Див. також getw, printf.

rand

Функція: генератор випадкових чисел.

Синтаксис: #include<stdlib. h>
int rand (void);

Опис: rand використовує мультиплікативний конгруентний генератор випадкових чисел з періодом 2 в степені 32 для отримання псевдовипадкових чисел у діапазоні від 0 до RAND_MAX. Символічна константа RAND_MAX визначена в stdlib.h. Див. також random, randomise, srand.

Приклад:

```
#include <stdio. h>
#include <stdlib. h>
int main (void)
{
int i;
printf ("10 випадкових чисел від 0 до 99 \n\n");
for (i=0: i<10; i++) printf ("%d\n", rand()%100);
return 0;
}
```

random

Функція: генератор випадкових чисел.

Синтаксис: #include<stdlib. h>
int random (int num);

Опис: random повертає випадкове число у діапазоні від 0 до num-1. random (num) – це макро, визначене у вигляді (rand () %num).

Див. також rand, randomise, srand.

randomize

Функція: ініціалізація генератора випадкових чисел.

Синтаксис: #include<stdlib. h>
#include<time. h>
void randomize (void);

Опис: randomize ініціалізує генератор випадкових чисел деяким випадковим числом. Див. також rand, random.

_read, read

Функції: зчитування даних з файлу.

Синтаксис: #include<io. h>
int _read (int handle, void* buf, unsigned len);
int read (int handle, void* buf, unsigned len);

Опис: _read робить спробу прочитати len байтів з файлу, пов'язаного з handle, у буфер, адресований параметром buf. Функція _read є безпосереднім викликом операції читання DOS. У файлі, відкритому у текстовому режимі, функція read не вилучає символи "повертання каретки".

Функція read робить спробу прочитати len байтів з файлу, пов'язаного з handle, у буфер, адресований параметром buf. У файлі, відкритому у текстовому режимі, функція read вилучає символи "повертання каретки" та видає "кінець файлу" (EOF) при отриманні символу Ctrl-Z. Параметр handle у функціях той самий, який отримують на виході функцій creat, open, dup, dup2. Обидві функції починають читання дискових файлів з поточного положення покажчика файлу. Після закінчення читання функція збільшує покажчик файлу на кількість зчитаних байтів.

Приклад:

```
#include<stdio. h>
#include<io. h>
#include<alloc. h>
#include<fcntl. h>
#include<process. h>
#include<sys\stat. h>
 int main (buffer)
{
 void* buf;
int handle, bytes;
buf = malloc (10);
/* Пошук файлу BIB. TTT спроба зчитати з нього 10 байтів, перед
запуском програми необхідно створити файл BIB. TTT*/
 if((handle - open ( "BIB. TTT", O_RDONLY | O_BINARY)) == -1)
{
printf("Помилка при відкритті файлу. \n");
exit (1); }
if ((bytes = read (handle, buf, 10)) ==-1)
 {
printf ("Помилка читання. \n");
exit (1);
} printf("Зчитано, %d байтів. \n", bytes);
return 0;
}
```

real

Функція: повертає дійсну частину комплексного числа або перетворює число з двійково-десяткового кодування назад у float, double або long, double.

Синтаксис:

```
#include<complex. h>                    #include<bcd. h>
double real (complex x)                 double real (bcd x);
```

Опис: real повертає дійсну частину комплексного числа.

Див. також bed, complex, imag.

realloc

Функція: перерозподіляє пам'ять.

Синтаксис: #include<stdlib. h>

```
void* realloc (void *block, size_t size);
```

Опис: realloc намагається стиснути або збільшити попередньо виділений блок до розміру у size байтів. Аргумент block вказує на блок пам'яті, отриманий при активізації функції malloc, calloc або realloc. Якщо block є нульовим покажчиком, realloc працює так само, як і malloc, realloc змінює розмір виділеного блоку пам'яті і за необхідності копіює його вміст у новий блок. realloc повертає адресу блока, яка може відрізнятися від початкової.

Див. також calloc, farrealloc, free, malloc.

Приклад:

```
#include<stdio. h>
#include <string. h>
#include<alloc. h>
int main (void)
{ char* str;
  str = malloc (10);
  strcpy (str,"Hello");
  printf("рядок: %s\n, Адреса: %p\n", str, str);
  str = realloc (str, 20);
  printf ("'рядок: %s\n, Нова адреса: %p\n", str, str);
  free (str);
  return 0;
}
```

rectangle

Функція: рисує прямокутник.

Синтаксис: #include <graphics. h>

```
void far rectangle ( int left, int top, int right, int bottom);
```

Опис: rectangle рисує прямокутник з використанням поточних атрибутів: (left, top) – координати лівого верхнього кута прямокутника, а (right, bottom) – його правий нижній кут.
Див. також bar, bar3d, setcolor, setlinestyle.

remove

Функція: видаляє файл.
Синтаксис: #include<stdio. h>
int remove (const char* filename);
Опис: remove вилучає файл, ім'я якого визначається параметром filename. Це макрокоманда, яка просто транслюється у виклик unlink.
Див. також unlink.
Приклад:

```
#include<stdio. h>
 int main (void)
{ char file [80];
printf("Введіть ім'я файлу, що видалятиметься "/
gets (file);
if (remove (file) = = 0) /* видалити файл*/
printf("Файл %s видалений \n".);
return 0;
}
```

rename

Функція: перейменовує файл.
Синтаксис: #include<stdio. h>
int rename (const char * oldname, const char * newname);
Опис: rename змінює ім'я файлу з oldname на newname.
Приклад:

```
#include<stdio. h>
int main (void)
{ char oldname [80], newname [80];
  printf("Ім'я файлу:");
  gets (oldname);
  printf("Нове ім'я:");
  gets (newname);
  if (rename ( oldname, newname) = =0)
  printf ("файл перейменовано з %s у %s\n", oldname, newname);
  else perror ( "rename");
  return 0; }
```

rewind

Функція: встановлює покажчик на початок потоку.

Синтаксис: #include <stdio. h>

int rewind (FILE* stream);

Опис: rewind (stream) еквівалентно fseek (stream, 0L, SEEK_SET), за виключенням того, що rewind обнуляє ознаки кінця файлу та помилки, тоді як fseek обнуляє лише ознаку кінця файлу. Див також fopen, fseek, ftell.

scanf

Функція: виконує форматизоване введення з потоку stdin.

Синтаксис: #include<stdio. h>

int scanf (const char* format [. adress,...]);

Опис: scanf проглядає вхідні поля, символ за символом, зчитуючи їх з потоку stdin. Потім кожне поле форматується у відповідності із специфікацією формату, яка передається scanf аргументом format. Кожна специфікація починається з символу % та закінчується деяким символом, що задає тип представлення:

d – десяткове подання;

o – вісімкове подання без знаку;

x – шістнадцятіркове подання без знаку;

h – десяткове подання без знаку;

c – параметр як один символ:

s – параметр-рядок;

e – E -формат подання величин float та double;

f – звичайне подання величин float та double;

Після цього вона записує відформатоване введення за адресами, що задаються аргументами, що йдуть за форматним рядком, scanf повертає кількість успішно прочитаних, перетворених та записаних вхідних полів. Число специфікацій формату повинно збігатися з числом адрес. Специфікації формату функцій scanf мають форму:

% [*] [ширина] [F/N] [h/1/L] символ типу

Див. також cscanf, fscanf, getc, vfscanf, vsscanf.

Приклад:

```
#include <stdio. h> /* примітивний калькулятор*/
main()
{ double sum, v;
 sum=0;
 while (scanf (" %f", &v)! =EOF)
printf ("\ t%. 2f\n", sum+ =v)
}
```

setbkcolor

Функція: встановлює поточний колір фону, використовуючи палітру.

Синтаксис: #include <graphics. h>

void far setbcolor (int color);

Опис: setbcolor встановлює фон кольором color. Аргумент color може бути іменем або номером:

0 - BLACK 1- BLUE 2- GREEN 3- CYAN 4- RED 5- MAGENTA
6 - BROWN 7- LIGHTGRAY 8- DARKGRAY 9- LIGHTBLUE
10 - LIGHTGREEN 11 – LIGHTCYAN 12 - LIGHTRED 13 - LIGHTMAGENT
14 - YELLOW 15 - WHITE.

Ці символічні імена визначені в graphics.h.

Приклад:

```
#include <graphics. h>
#include <stdlib. h>
#include <stdio. h>
#include <conio. h>
 int main (void)
{
 int graphdriver = EGA, gmode == EGAHI, errorcode;
int bkcolor, x, y, maxcolor;
char msg [80];
initgraph( &graphdriver, &gmode, "");
errorcode = graphresult();
if ( errorcode != grOk) {
printf("Помилка: %s\n", grapherrormessage (errorcode));
getch();
exit (1);
}
maxcolor = getmaxcolor();
/* максимальний індекс кольору*/
settextjustify (CENTER_TEXT, CENTER_TEXT;
x = getmaxx() / 2;
y = getmaxy() / 2;
for (bkcol=0; bkcol<=maxcolor (); bkcol++)          /* по кольорах*/
{
  cleardevice();
  setbkcolor (bkcol);
  if (bkkol == WHITE)
  setcolor (EGA_BLUE);
```

```
    sprintf (msg, "Background color: %d", bkcol);
    outtext (x, y, msg);
    getch();
  } closegraph();
    return 0;
}
```

setmode

Функція: встановлює режим відкриття файлу.

Синтаксис: #include<fcntl. h>

int setmode (mt handle, unsigned amode);

Опис: setmode встановлює режим відкриття файлу (бінарний або текстовий), що відповідає параметру handle. Аргумент amode набуває при цьому значення або O_BINARY, або O_TEXT (fcntl. h).

sin

Функція: обчислює синус.

Синтаксис: Дійсна версія та комплексна версія:

#include <math. h> #include<complex. h>

double sin (double x); complex sin (complex x);

Опис: sin обчислює синус x. Кут задається у радіанах. Синус комплексного числа визначається як sin (z) = (exp (i* z) -exp (-i* z))/(2i).

sinh

Функція: обчислює гіперболічний синус.

Синтаксис: Дійсна версія та комплексна версія

#include <math. h> #include<complex. h>

double sinh (double x); complex sinh (complex x);

Опис: обчислює гіперболічний синус: $(e^{x}-e^{-x})/2$. Для комплексного числа визначається як sinh (z) = (exp (z) -exp (-z))/2.

sleep

Функція: затримує виконання програми на заданий інтервал часу.

Синтаксис: #include<dos. h>

void sleep (unsigened seconds);

Опис: при виклику функції sleep виконання програми призупиняється на час (у секундах), заданий параметром seconds. Див. також delay.

Приклад:

```
#include<dos. h>
#include<stdio. h>
```

```
int main (void)
{
 int i;
for (i=1; i<3; 1++)
   {
    printf ("Зупинка на %d секунд\n", i):
    sleep (i);
    }
return 0:
}
```

sound

Функція: запускає вбудований динамік вашого ПК на генерацію звуку вказаної частоти.

Синтаксис: #include<dos. h>

void sound (unsigned frequency);

Опис: sound запускає вбудований динамік на генерацію звуку вказаної частоти. Частота визначається параметром frequency (в герцах, тобто циклах на секунду). Див. також delay, nosound.

Приклад:
```
/* Звук частоти 7 Гц протягом 15 секунд          */
/*Чи спроможний ваш ПК генерувати такий звук?*/
int main (void)
{ sound (7);
  delay ( 15000);
  nosound();
}
```

sprintf

Функція: провадить форматоване виведення в рядок.

Синтаксис: #include<stdio. h>

int sprintf (char* buffer, const char* format [, argument,...]);

Опис: sprintf отримує набір аргументів, застосовує до кожного специфікацію формату із рядка format та виводить відформатовані дані в рядок.

Див. також printf, fprintf.

sqrt

Функція: для дійсного аргументу обчислює квадратний корінь.

Синтаксис: Дійсна версія та комплексна версія

```
#include <math. h>                  #include<complex. h>
double sqrt (double x);             complex sqrt (complex x);
```

Опис: sqrt обчислює додатний квадратний корінь від параметра x. Для комплексного числа x sqrt(x) дає комплексний корінь, з аргументом arg, це arg (x)/2. Комплексний квадратний корінь визначається:
sqrt (z) =sqrt (abs (z)) (cos (arg (z)/2)+ isin (arg (z)/2))

sscanf

Функція: виконує форматоване введення з рядка.
Синтаксис: #include<stdio. h>

int sscanf(const char *buffer, const char* format [, adress,...]);

Опис: sscanf переглядає набір вхідних полів, по одному символу, зчитуючи їх з рядка. Потім кожне поле форматується відповідно до специфікації формату, що передається sscanf через аргумент format. В кінці sscanf зберігає введені відформатовані поля за адресами, які передаються як аргументи після format. Число аргументів відповідає числу специфікацій формату. Див. також fscanf, scanf.
Приклад:

```
#include<stdio. h>
char buffer [] = "a 3. 14159 12 a-string\n";
int main (void)
{
char ch; float f; int i;
char string [20];
sscanf (buffer,"%c %f %d %s",  &ch, &f, &i, string);
printf ("%c %f %i %s", ch, f, i, string);
return 0;
}
```

stpcpy

Функція: stpcpy копіює один символьний рядок в інший.
Синтаксис: #include <string. h>

char* stpcpy (char *dest, const char* src);

Опис: stpcpy копіює рядок src у рядок dest до нульового символу закінчення рядка, stpcpy повертає dest+ strlen(src).
Приклад:

```
#include<stdio. h>
#include <string. h>
int main (void)
{
char string [10];
char* strl = "abcdefghi";
```

```
stpcpy ( string, strl);
printf ("%s\n", string);
return 0;
}
```

strcat

Функція: додає один рядок до іншого.

Синтаксис: #include<string. h>

char* strcat (char* dest, char* src);

Опис: strcat додає копію одного рядка до іншого. strcat повертає покажчик на об'єднаний рядок.

Приклад:

```
#include <string. h>
#include<stdio. h>
 int main (void)
{
char destination [25];
char ""blank = " ",* c = "C++", *turbo = "Turbo";
strcpy (destination, turbo);
strcpy(destination, blank);
strcpy (destination, c);
printf ("%s\n", destination);
return 0;
}
```

strchr

Функція: відшуковує у рядку першу появу даного символу.

Синтаксис: #include <string. h>

char* strchr (const char* s, int c);

Опис: strchr переглядає рядок, проводячи пошук заданого символу. Функція strchr реагує на першу появу символу c у рядку s. Див. також strcspn, strrchr.

strcmp

Функція: порівнює два рядки.

Синтаксис: #include<string.h>

int strcmp (char* s1, const. char* s2);

Опис: strncmp виконує беззнакове порівняння рядків s1 та s2, починаючи з першого символу у кожному рядку доти, доки не зустрінеться перша невідповідність або рядки не закінчаться, strcmp повертає такі значення:

< 0, якщо s1 менше s2;

= = 0, якщо s1 дорівнює s2;
> 0, якщо s1 більше s2.

Див. також strcrnpl, stricmp, strncmp, strncmpl.

Приклад:

```
# include <string. h>
#include<stdio. h>
 int main (void)
{
 char* buf1 = "yyy".* buf2 = "ttt",* buf3 = "fff;
int ptr;
ptr = strcmp (buf2, buf1);
if (ptr>0)
      printf ("buf2 більше, ніж buf1 \n");
      else printf ("buf2 менше ніж buf1\n");
ptr = strcmp (buf2, buf2);
if (ptr>0)
       printf ("buf2 більше ніж buf3\n");
       else   printf ("buf2 менше ніж buf3\n");
return 0;
}
```

strcmpi

Функція: порівнює рядки str1 та str2 без розгляду їх регістрів.

Синтаксис: #include <string. h>

int strcmpi (const char * s1, const char* s2);

Опис: strcmpi виконує беззнакове порівняння рядків s1 та s2 без розгляду регістрів (аналогічно stricmp). Вона повертає значення (<0, 0, >0) залежно від результату порівняння s1 (або частини його) та s2 (або частини його), strcmpi повертає значення: (int)

< 0, якщо s1 менше s2;
= =0, якщо s1 дорівнює s2;
> 0, якщо s1 більше s2.

Див. також strcmp, strcoll, stricmp, strncmp.

strcoll

Функція: порівнює два рядки.

Синтаксис: #include <string. h>

int strcoll (char* s1, char* s2);

Опис: strcoll виконує порівняння рядків s1 й s2 відповідно до списку, що визначається за допомогою setlocale. strcoll повертає значення:

< 0, якщо s1 менше s2;
= =0, якщо s1 дорівнює s2;
> 0, якщо s1 більше s2.

Див. також strcmp, strcmpi, stricmp, strncmp, strncmpi, strnicmp, strxfrm.

Приклад:

```
#include <string. h>
#include<stdio. h>
int main (void)
{
char* two = "Microsoft";
char* one = "Borland";
int check;
check = strcoll(one, two);
if (check)
printf("Рядки рівні \n");
if(check<0)
printf ("%s йде попереду %s\n", one, two);
if (check>0)
printf ("%s йде попереду %s\n", two, one);
return 0;
}
```

strcpy

Функція: копіює один рядок в інший.

Синтаксис: #include<string. h>

char* strcpy (char* dest, const char* src);

Опис: stpcpy копіює байти з рядка src у рядок dest. strcpy повертає dest.

strlen

Функція: обчислює довжину рядка.

Синтаксис: #include <string. h>;

size_t strlen (const char* s);

Опис: strlen обчислює довжину рядка s. strlen повертає число символів у рядку str, не рахуючи нульове закінчення.

Приклад:

```
#include<stdio. h>
#include<string. h>
 int main (void)
{ char* string = "Том & Jerry";
```

```
printf ("%d\n", strlen (string));
return 0;
}
```

strlwr

Функція: перетворює букви верхнього регістру у букви нижнього регістру.

Синтаксис: #include <string. h>

char* strlwr (char* s);

Опис: strlwr перетворює букви верхнього регістру (A-Z) рядка s у букви нижнього регістру (a-z). strlwr повертає покажчик на рядок s.

Приклад:

```
#include<stdio. h>
#include<string. h>
int main (void)
{
char* string = "AbbA ";
printf("Рядок до виклику: %s\n", string);
strlwr (string);
printf ("Рядок після виклику strlwr: %s\n", string);
return 0;
}
```

strncat

Функція: додає частину одного рядка до іншого.

Синтаксис: #include <string. h>

char* strncat (char* dest, const char* src, size_t maxlen);

Опис: strncat копіює maxlen символів рядка src у кінець dest і потім додає нульовий символ. Максимальна довжина результуючого рядка strlen (dest)+ maxlen. strncat повертає dest.

Приклад:

```
#include<string. h>
#include<stdio.h>
int main (void)
{ char destination [25];
  char* source = "States";
  strcpy ( destination, "United");
  strncat (destination, source, 7);
  printf ("%s\n", destination);
  return 0;
}
```

strncmp

Функція: порівнює частину одного рядка з іншим рядком.

Синтаксис: #include<string. h>

int strncmp (const char* s1, const. char* s2, size_t maxlen);

Опис: strncmp робить те саме порівняння, що й функція strcmp, але переглядає лише maxlen символів. strncmp повертає такі значення:

< 0, якщо s1 менше s2;

==0, якщо s1 дорівнює s2;

> 0, якщо s1 більше s2.

Див. також strcmp, strcoll, stricmp, strncmpi strnicmp.

strncmpi

Функція: порівнює одну частину одного рядка з іншою частиною без розгляду регістру.

Синтаксис: #include <string. h>

int strcmpi (const char *s1, const char* s2, size_t n);

Опис: strncmpi виконує беззнакове порівняння рядків s1 та s2, максимальною довжиною n байтів, починаючи з першого символу у кожному рядку. Регістри при цьому не враховуються, strncmpi повертає такі значення:

< 0, якщо s1 менше s2;

==0, якщо s1 дорівнює s2;

> 0, якщо s1 більше s2.

strncpy

Функція: копіює дану кількість байтів з одного рядка в інший з відкиданням або додаванням, якщо це необхідно.

Синтаксис: #include<string. h>

char* strncpy (char* dest, const char* src, int maxlen);

Опис: strncpy копіює точно maxlen символів з рядка src в рядок dest, якщо потрібно, відкидаючи чи додаючи нулі у dest. Функція strncat повертає dest.

Приклад:

```
#include<stdio. h>
#include<string. h>
int main (void)
{
char string [10]:
char* strl = "abcdefghi";
stpncpy (string, strl, 3);
```

```
printf (" %s\n", string);
return 0; }
```

strnicpm

Функція: порівнює частину одного рядка з іншим без розгляду регістрів.

Синтаксис: #include <string. h>

int strnicmp (const char* s1, const char* s2, size_t maxlen);

Опис: strnicmp виконує знакове порівнювання s1 та s2, максимально maxlen байтів, починаючи з першого символу у кожному рядку. Регістри при цьому не враховуються; strncmpi повертає такі значення;

< 0, якщо s1 менше s2;

= =0, якщо s1 дорівнює s2;

> 0, якщо s1 більше s2.

strnset

Функція: змінює задану кількість символів у рядку на даний символ.

Синтаксис: #include <string. h>

char *strnset(char* s, int ch, size_t n);

Опис: strnset копіює у перші n байтів рядка s символ ch. Якщо n>strlen(s), то strlen(str) отримує значення n. Функція strnset повертає s.

strpbrk

Функція: відшуковує в рядку першу появу будь-якого символу з заданого набору.

Синтаксис: #include<string. h>

char* strpbrk (const char* s1, const char* s2);

Опис: strpbrk проглядає рядок s1 в пошуках першого символу, що є у рядку s2. strpbrk повертає покажчик на першу появу в рядку s1 будь-якого символу з рядка s2.

strrev

Функція: "перевертає" рядок.

Синтаксис: #include<string. h>

char* strrev (char* s);

Опис: strrev "перевертає" (тобто записує з кінця у зворотному порядку) усі символи у рядку, (за виключенням нуля-закінчення). (Наприклад, зміна рядка string\0 виконується на рядок gnirts\0). strrev повертає покажчик на "перевернутий" рядок.

Приклад:

```
#include<string. h>
#include<stdio. h>
```

```
int main (void)
{ char  *forward = "string";
  printf("Перед strrev: %s\n", forward);
  strrev (forward);
  printf("Після strrev: %s\n", forward);
  return 0;
}
```

strtod

Функція: перетворює рядок у число подвійної точності.

Синтаксис: #include<stdlib. h>

double strtod (const char* s, char** endptr);

Опис: strtod перетворює символьний рядок s у число подвійної точності. Рядок s – послідовність символів. Символи повинні відповідати формату: [ws] [sn] [ddd] [.] [ddd] [fmt. [sn] ddd], де

[ws] – символ пропуску;
[sn] – знак (+ або -);
[ddd] –цифри;
[fmt] – символи e або E;
[.] – десяткова точка.

Див. також atof.

Приклад:

```
#include<stdio. h>
#include<stdlib. h>
int main (void)
{
char input [80],* endptr;
double value;
printf ("Введіть число з плаваючою точкою: "); gets (input);
value = strtod ( input, &endptr);
printf("Рядок: %s, число: %lf\n", input, value);
return 0;
}
```

strupr

Функція: перетворює букви нижнього регістру у букви верхнього регістру.

Синтаксис: #include<string. h>

char* strupr (char* s);

Опис: strupr перетворює букви нижнього регістру у рядку s в букви верхнього регістру, strupr повертає s. Див. також strlwr.

tan

Функція: обчислює тангенс кута.

Синтаксис: Дійсна версія та комплексна версія

```
#include <math. h>                  #include<complex. h>
double tan (double x);              complex tan (complex x);
```

Опис: tan обчислює тангенс. Кути задаються у радіанах. Комплексний тангенс визначений як tan(z) =sin(z)/cos(z). Функція tan повертає тангенс x, sin (x)/cos (x). Див. також atan, atan2, complex, cos, sin.

tanh

Функція: обчислює гіперболічний тангенс.

Синтаксис: Дійсна версія та комплексна версія:

```
#include <math. h>          #include<complex. h>
double tanh (double x);     complex tanh (complex x);
```

Опис: tanh обчислює гіперболічний тангенс sinh(x)/cosh(x). Функція tanh повертає гіперболічний тангенс x. Див. також complex, cos, cosh, sinh, tan.

textbackground

Функція: обирає новий колір фону для тексту.

Синтаксис: #include<conio. h>

```
void textbackground (int newcolor);
```

Опис: textbackground обирає колір фону у текстовому режимі. Функція працює тільки для функцій, пов'язаних з виведенням на екран у текстовому режимі. newcolor визначає новий колір фону. Ви можете визначити його як ціле (від 0 до 7) або як одну з символічних констант з файлу conio.h.: BLACK-0, BLUE-1, GREEN-2, CYAN-3, RED-4, MAGENTA-5, BROWN-6, LIGHTGRAY-7, DARKGRAY-8, LIGHTBLUE-9, LIGHTGREEN-10, LIGHTCYAN-11, LIGHTRED-12, LIGHTMAGENTA-13,YELLOW-14, WHITE-15.

Приклад:

```
#include<conio. h>
int main (void)
{
int i,j;
clrscr();
for (i=0; i<9; i++ )
  {
```

```
    for (j=0; j<80;  j++);
    cprintf("C");    cprintf ("\r\n");
    textcolor(i+ 1);
    textbackground (i);
    }
  return 0;
  }
```

textcolor

Функція: встановлює колір символів у текстовому режимі.

Синтаксис: #include <conio. h>

void textcolor (int newcolor);

Опис: textcolor визначає колір символів, які виводяться на екран у текстовому режимі. Таблиця № 3 відображає можливі кольори (як символьні константи) та їх числові значення. Константи та числові значення вказані у описі void textbackground (int newcolor). Ви можете зробити символи, що блимають, додаючи BLINK до кольору символів. Наприклад: textcolor (CYAN+ BLINK). Див. також textbackground.

Приклад:

```
#include<conio. h>
 int main (void)
{
int i;
for (i=0; i<15; i++)
  {
   textcolor (i);
   cprintf ("Обраний колір символів. \r\n");
   }
 return 0;
}
```

textheight

Функція: повертає висоту рядка у пікселях.

Синтаксис: #include <graphics. h>

int far textheight (char far* textstring);

Опис: графічна функція textheight бере поточний розмір шрифту та фактор збільшення та визначає висоту textstring. Функція використовується для обчислення відстані між рядками, висоти вікна тощо.

Див. також gettextsettings, textwidth.

textmode

Функція: переводить екран у текстовий режим.

Синтаксис: #include<conio. h>

void textmode (int newmode)

Опис: textmode обирає вказаний текстовий режим. Можна визначити текстовий режим (аргумент newmode), використовуючи символьну константу перелічуваного типу text_modes (з файлу conio. h).

Див. також gettextinfo, window.

textwidth

Функція: повертає ширину рядка у пікселях.

Синтаксис: #include <graphics. h>

int far textwidth (char far* textstring);

Опис: графічна функція textwidth бере довжину рядка, розмір шрифту та фактор збільшення й визначає ширину textstring в пікселях. Див. також gettextsettings, outtext, outtextxy, setteststyle, textheight.

toascii

Функція: переводить символи у формат ASCII.

Синтаксис: #include<ctype. h>

int toascii (int c);

Опис: toascii – макрокоманда, яка конвертує ціле с в код ASCII, очищаючи усі, крім молодших семи бітів; при цьому отримані значення знаходяться в проміжку від 0 до 127. toascii повертає конвертоване значення с.

Приклад:

```
#include<stdio. h>
#include<ctype. h>
 int main (void)
{ int number, result;
  number = 547;
  result = toascii (number);
  printf ("%d %d\n", numder, result);
  return 0; }
```

tolower

Функція: переводить символи у символи нижнього регістру.

Синтаксис: #include<ctype. h>

int tolower (int ch);

Опис: tolower – функція, що переводить ціле ch (в проміжку EOF до 255) в його значення для нижнього регістру (a-z) (якщо були символи верхнього регістру (A-Z)).

toupper

Функція: транслює символи у верхній регістр.

Синтаксис: #include<ctype. h>

int toupper (int ch);

Опис: toupper – функція, що перетворює ціле ch (в межах від EOF до 255) в значення верхнього регістру (A-Z) (якщо до цього був нижній регістр (a-z)).

Приклад:

```
#include<string. h>
#include<stdio. h>
#include<ctype. h>
 int main (void)
{
 int length, i;
 char* string = "this is a string.";
 length = strlen ( string);
 for (i=0; i<length; i++)
   {
     string [i] = toupper (string [i]);
    }
 printf (" %s\n", string);
return 0;
}
```

ultoa

Функція: перетворює число типу довге ціле без знаку у рядок.

Синтаксис: #include<stdlib. h>

char* ultoa (unsigned long value, char* string, int radix);

Опис: utoa перетворює value у рядок, який закінчується нульовим символом та розміщує результат в string, value має тип unsigned long.

Див. також itoa, ltoa.

Приклад:

```
#include<stdlib. h>
#include<stdio. h>
int main (void)
{
unsigned long lnumber = 3123456789L;
char string [25];
ultoa(lnumber, string, 10);
printf("рядок = %s, unsigned long = %lu\n", string, lnumber);
return 0; }
```

vfprintf

Функція: направляє відформатоване виведення у поток.

Синтаксис: #include <stdio. h>

int vfprintf (FILE *stream, char* format, va_list arglist);

Опис: vfprintf відома як додаткова точка входу для функцій printf. Вона веде себе так само, як і printf() – двійники, проте має доступ до покажчика на список аргументів, а не до самого аргументного списку. Див. також printf.

wherex

Функція: повертає горизонтальну позицію курсора у вікні.

Синтаксис: #include<conio. h>

int wherex (void);

Опис: wherex повертає координату x позиції курсора (у поточному текстовому вікні), wherex повертає ціле число від 1 до 80.

Див. також gotoxy, wherey.

wherey

Функція: повертає вертикальну позицію курсора у вікні.

Синтаксис: #include<conio. h>

int wherey (void);

Опис: wherey повертає координату Y позиції курсора (у поточному текстовому вікні), wherey повертає ціле число від 1 до 25.

Див. також gotoxy, wherex.

window

Функція: визначає у текстовому режимі активне вікно.

Синтаксис: #include<conio. h>

void window (int left, int top, int right, int bottom);

Опис: window визначає текстове вікно на екрані. left і top – екранні координати лівого верхнього кутка вікна, right і bottom – екранні координати правого нижнього кутка. 80-колонний режим: 1, 1, 80, 25 та 40-колонний режим: 1, 1, 40, 25. Див. також clrscr, textmode.

Приклад:

```
#include<conio. h>
 int main (void)
{ window(10, 10, 40, 11);
  textcolor(BLACK);
  textbackground (WHITE);
  cprintf ("Це тест \r\n");
  return 0; }
```

write

Функція: записує дані у файл.

Синтаксис: #include<io. h>

int write (int handle, void* buf, unsigned len);

Опис: write записує буфер, що містить дані, у файл або на пристрій, що відповідає номеру handle. handle – логічний номер, write – повертає число записаних байтів. Див. також creat, lseek, open, read.

Література

1. М. Эллис, Б. Страуструп Справочное руководство по языку C++ с комментариями: Пер. с англ. - Москва: Мир, 1992.
2. Стенли Б. Липпман C++ для начинающих: Пер. с англ. 2т. - Москва: Унитех; Рязань: Гэлион, 1992.
3. Бруно Бабэ Просто и ясно о Borland C++: Пер. с англ. - Москва: БИНОМ, 1994.
4. Подбельский В.В. Язык C++: Учебное пособие. - Москва: Финансы и статистика, 1995.
5. Ирэ Пол Объектно-ориентированное программирование с использованием C++: Пер. с англ. - Киев: НИИПФ ДиаСофт Лтд, 1995.
6. Т. Фейсон Объектно-ориентированное программирование на Borland C++ 4.5: Пер. с англ. - Киев: Диалектика, 1996.
7. Т. Сван Освоение Borland C++ 4.5: Пер. с англ. - Киев: Диалектика, 1996.
8. Г. Шилдт Самоучитель C++: Пер. с англ. - Санкт-Петербург: BHV-Санкт-Петербург, 1998.
9. У. Сэвитч C++ в примерах: Пер. с англ. - Москва: ЭКОМ, 1997.
10. К. Джамса Учимся программировать на языке C++: Пер. с англ. - Москва: Мир, 1997.
11. Скляров В.А Язык C++ и объектно-ориентированное программирование: Справочное издание. - Минск: Вышэйшая школа, 1997.
12. Х. Дейтел, П. Дейтел Как программировать на C++: Пер. с англ. - Москва: ЗАО "Издательство БИНОМ", 1998.- 640 с.
13. Климова Л.М. СИ++. Практическое программирование. – Москва: "Кудиц-образ", 2001.- 586 с.

Навчальне видання

Людмила Михайлівна Круподьорова
Анатолій Михайлович Пєтух

Технологія програмування мовою Сі

Частина 1

Навчальний посібник

Оригінал-макет підготовлено Круподьоровою Л.М., Пєтухом А.М.

Редактор Т.О.Старічек

Науково-методичний відділ ВНТУ
Свідоцтво Держкомінформу України
серія ДК № 746 від 25.12.2001
21021, м. Вінниця, Хмельницьке шосе, 95, ВНТУ

Підписано до друку 24.01.2007 р.
Формат 29,7×42 1/4
Друк різографічний
Тираж 75 прим.
Зам. № 2007-014

Гарнітура Times New Roman
Папір офсетний
Ум.друк.арк. 11.7

Віддруковано в комп'ютерному інформаційно-видавничому центрі
Вінницького національного технічного університету
Свідоцтво Держкомінформу України
серія ДК № 746 від 25.12.2001
21021, м.Вінниця, Хмельницьке шосе, 95, ВНТУ