В.Х. Власюк, Л. М. Круподьорова

ПРОГРАМУВАННЯ МОВОЮ СІ

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

ВІННИЦЬКИЙ ДЕРЖАВНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ

В.Х. Власюк **Л. М. Круподьорова**

ПРОГРАМУВАННЯ МОВОЮ СІ

Затверджено Ученою радою Вінницького державного технічного університету як навчальний посібник для студентів спеціальності "Програмне забезпечення автоматизованих систем". Протокол № 3 від 25 жовтня 2001 р.

Вінниця ВДТУ 2002

УДК 621.382
В58

Рецензенти

С,В,Юхимчук , доктор технічних наук, професор

В.М.Дубовий , доктор технічних наук, професор

О.В.Прокопчук ,Генеральний директор компанії "LIANA"

Рекомендовано до видання Ученою радою Вінницького державного технічного університету Міністерства освіти і науки України

Власюк В.Х. Круподьорова Л.М.

В58 **Програмування мовою Сі**.Навчальний посібник

-Вінниця: ВДТУ 2002-67с.

У посібнику викладено основи алгоритмічної мови СІ та засоби її застосування до розв'язуваня задач з практичного програмування. Теоретичні відомості супроводжуються прикладами. Акцетована відмінність мовних конструкций СІ у порівнянні з Паскалем, що може бути корисним саме початківцям при переході з однієї мови на іншу. Для кожної теми є тексти завдань для лабораторних робіт наведено зміст контрольних та тестових завдань.

Посібник розроблений у відповідності з планом кафедри до дисципліни "Основи програмування та алгоритмічні мови"

УДК 621.382

ЗМІСТ

ПЕРЕДМОВА

ПРОГРАМУВАННЯ ТА АЛГОРИТМІЧНІ МОВИ

Мова програмування Сі

За декілька десятиліть світової комп'ютерної індустрії було створено безліч різноманітних мов програмування, проте переважна їх кількість або не дуже вдало копіювала, або створювалась для деякого вузького спеціалізованого застосування. Дійсно, це справа програміста, якій мові віддати перевагу - визначати найбільш прийнятний спосіб запису алгоритму задачі або будь-якого процесу обробки інформації за допомогою обчислювальної техніки. Внаслідок своєї еволюції отримали визнання невелика кількість мов програмування, серед яких і добре відома мова Сі. Існують десятки, а може й сотні видань, присвячених детальному опису саме Сі, проте не всі початківці в змозі правильно оцінити та визначити їх користь для себе. Тому на допомогу приходять методичні посібники, які разом з технічною документацією допомагають правильно зорієнтуватися, роблячи перші кроки у мистецтві програм та алгоритмів. Перед Вами посібник, присвячений основам програмування на Сі, хоча передбачається, що читач добре оволодів азами комп'ютерної грамотності, принаймні в об'ємі шкільного курсу (що найчастіше буває, має навики роботи на Паскалі) Тому для розгляду обрано порівняльний розгляд Сі саме з Паскалем - найпопулярнішим на сьогодні у середовищі програмістів-початківців. Тим більше, що вивчення саме цих мов передбачено майже усіма сучасними програмами ВУЗів, що передбачають розгляд інформаційно-комп'ютерних технологій. Матеріал націлений на надання допомоги студентам перших курсів інженерних та економічних спеціальностей вищих навчальних закладів для практичного ознайомлення з основами програмування Зміст цього посібника найкраще відповідає розгляду на другому семестрі першого курсу з програмування, коли у першому семестрі розглядається мова програмування Паскаль, у другому - Сі, хоча звичайно, можливим є надання й відповідних переваг тій чи іншій мові у зв'язку із спеціалізацією та напрямком підготовки студентів. Проте навряд чи можливо професійно навчитися програмуванню, гортаючи навіть найкращу книжку або користуючись найкращою документацією та методичною літературою. Тому вважається, що кожен студент має доступ до більш-менш сучасної комп'ютерної техніки й писатиме власні програми, користуючись практичними прикладами та теоретичними вказівками Посібник корисний у цьому плані, тому що містить відлагоджені програми та фрагменти, які можна використовувати при ознайомлені з керуючими конструкціями мов програмування, та націлює на глибоке розуміння процесу програмування в

цілому Крім цього, охоплено більшість аспектів синтаксису обраної мови програмування, акцентовано увагу на порівняльний аналіз з Паскалем, що може бути корисним при переході від
програмування з однієї мови на іншу. Хоча посібник розрахований на читача-початківця, але він містить чималий перелік бібліотечних функцій із прикладами програмного застосування, який може стати у пригоді і програмісту будь-якого рівня. У посібнику знаходяться тексти лабораторних, контрольних робіт, тестові завдання для перевірки знань, які без особливих ускладнень допоможуть організувати викладачеві нормальний навчальний процес протягом семестру.

Тепер перейдемо до розгляду Сі. Будь-яка мова програмування виникає не стихійно, а в результаті зміни уявлень людей щодо процесу програмування. Згодом такі процеси узагальнюються, формалізуються, досягають відповідного рівня парадигм, для підтримки яких і створюються нові мови. Першочергово мова програмування Сі була задумана у рамках машинно-незалежної операційної системи UNIX, а згодом реалізувалась і в рамках інших популярних ОС, в т.ч. й у MS DOS для задач системного програмування. Це мова відносно низького рівня - вона є посередником між мовами високого рівня та асемблером. Існує декілька причин для вивчення саме Сі, а надалі і його послідовника Сі++. Паскаль був чемпіоном структурного програмування 70-80-х років, тоді як Сі був створений як портативний асемблер високого рівня, що використовувався у системному програмуванні. Згодом виявилося, що Сі успішно робитиме й те, що й Паскаль, і навіть краще. Почнемо з того, що на відміну від Паскаля, компілятор Сі розроблявся для генерації коду, що підгодить до системного програмування. Він керує об'єктами, типовими для обчислювальної техніки: символами, числами, адресами, застосовуючи до них звичайні арифметичні та логічні операції, що існують в ЕОМ. Крім того, ядро мови Сі менше, ніж ядро мови Паскаль. Адже Сі орієнтований на використання стандартних бібліотек, тоді як у Паскаль вбудовані безліч компонентів мови - обробка рядків, математичні дії, введення та виведення у файлах тощо. Фактично Паскаль був створений для підтримки власних бібліотек, що містять програми багатократного використання, з чого випливає, що ця мова пропонує структурне програмування із всякими обмеженнями, більшість з яких мова Сі успішно знімає. На Сі можна писати модульні програми, використовуючи функції, крім того. Сі досить просто дозволяє створювати бібліотеки підпрограм, кожна з яких може бути представленою у вигляді header-файлів(*.h) або бібліотек реалізації(*.c). Згодом ця мова програмування стала однією з найбільш популярних інструментів як системних, так і прикладних програмістів у світі. Звичайно це не найкращий засіб для початкового ознайомлення з програмуванням (адже більш прості, могутні Бейсік та Паскаль непогано підходять для цього), але більшість програмістів прагнуть використовувати саме Сі для своїх серйозних розробок. Привертають увагу такі особливості мови, як вільне висловлювання думок, мобільність та надзвичайна доступність. Багаті бібліотеки функцій забезпечують зв'язок з ОС та апаратурою на різних рівнях. Досить висока швидкість

компіляції при такій же якості об'єктного коду, який отримується за допомогою Сі, не має аналогів серед багатьох відомих мов високого рівня. Викладений нижче матеріал допоможе початківцям досить стисло ознайомитися зі стандартом мови в рамках ANSI, тоді як більш-менш досвідчені програмісти, що мають непоганий досвід роботи на Паскалі та прагнуть працювати на Сі, зможуть оцінити програми та приклади, що демонструють певні мовні відмінності найпопулярніших мов програмування, серед яких протягом десятиліть впевнено почувається і Сі.

Процес ознайомлення та вивчення мови програмування умовно можна представити як послідовне проходження трьох етапів (кожен з етапів може повторюватися протягом навчання). По-перше, це засвоєння синтаксису мови, принаймні настільки, аби компілятор припинив регулярно видавати повідомлення про помилки у застосуванні мовних конструкцій. Другий етап можна присвятити практичному програмуванню - осмисленню правильно побудованих програм. На третьому етапі виробляється стиль програмування, що відповідає суті обраної мови - вмінню писати ясні, короткі, і головне, раціонально продумані програми. Звичайно, ці етапи не є допомогою або чимось обов'язковим, проте саме у такій послідовності та з цих позицій пропонується огляд мови Сі.

1. ОРГАНІЗАЦІЯ ПРОГРАМ ТА БАЗОВІ ТИПИ ДАНИХ

1.1. Функція main (): з цього все починається

Усі програми, написані на мові Сі, повинні містити в собі хоча б одну функцію. Функція **main** () - вхідна точка будь-якої програмної системи, причому ваша справа, де ви її розміщуватимете. Деякі програмісти воліють розміщувати її на початку файлу, дехто в кінці, проте незалежно від цього необхідно пам'ятати наступне: якщо вона буде відсутня, завантажувач не зможе зібрати програму, про що ви отримаєте відповідне попередження. Перший оператор вашої програми має розміщуватися саме в цій функції. Майже мінімальна програма має вигляд:

```
/ * мінімальна програма на Сі * /

main ()
{
return 0;
{
```

Функція починається з імені. У нас вона не має параметрів, тому за її ім'ям розташовуються порожні круглі дужки - (). Далі обидві фігурні дужки -{...} позначають блок або складений оператор, з яким ми працюватимемо, як із одним цілим. У Паскалі аналогічний зміст мають операторні дужки *begin ... end.* Мінімальна програма має лише один оператор - оператор повернення значення *return.* Він завершує виконання програми та повертає в нашому випадку деяке ціле значення (ненульове значення свідчить про помилки в програмі, нульове про успішне її

завершення). Виконання навіть цієї найпростішої програми, як і решти багатьох, проходить у декілька етапів:

Рис. 1. Етапи виконання типової Сі - програми

Перше знайомство із Сі завершимо переліком ідентифікаторів, що фіксуються як службові слова мови та в іншому призначенні використовуватися не можуть:

hit	extern
else	
char	register
for	
float	typedef
do	
double	static
while	
struct	goto
switch	
union	return
case	
long	sizeof
default	
short	break
const	
unsigned	continue
auto	if

1.2. Використання коментарів

Текст на Сі, що міститься у дужках /* та */, ігноруватиметься компілятором, тобто вважатиметься коментарем до програми. Коментарі слідують двом цілям - документуванню програм (наприклад, для супроводу або майбутньої модифікації) та підтримці їх відлагодження. Тобто, в програму бажано вміщувати текст, що хоч якось пояснює її роботу та призначення. Проте не слід надто зловживати коментарями, а використовувати більш розумні форми по найменуванню змінних. функцій тощо. Якщо, наприклад, функція матиме назву add_matrix, очевидно не зовсім раціональним буде включення у програму після її заглавної частини коментар про те, що

/*функція обчислює суму матриць */

У цьому випадку ім'я функції пояснює її призначення. У більш сучасних версіях Сі широко застосовується так званий угорський запис імен, коли ім'я змінної містить в собі інформацію про її призначення і тип.

1.3. Директиви включення

У багатьох програмах ми зустрічаємо використання так званих директив включення файлів. Синтаксис використання їх у програмі такий:

include < file_l>

include < file_2>

... ...

include < file_n>

По-перше, слід звернути увагу на те, що на відміну від більшості операторів, ця директива не завершується крапкою з комою. Використання таких директив призводить до того, що процесор підставляє на місце цих директив тексти файлів у відповідності з тими, що перелічені у дужках <...>. Якщо ім'я файлу міститься у таких дужках, то пошук файлу проводитиметься у спеціальному розділі файлів для включення (усі файли з розширенням .h - header - файли). Якщо даний файл у цьому розділі буде відсутнім, то процесор видасть відповідне повідомлення про помилку, яка є досить типовою для початківців при роботі в інтегрованому мовному середовищі:

< Unable to open include file 'file.h'. >

У цьому випадку достатньо перевірити не тільки наявність header-файлу у відповідній директорії, але й впевнитися у тому, що опція Options \ Directories дійсно відповідає спеціальному розділу, де розташовані файли включення.

Існує і другий спосіб - розташування імені файлу у подвійних лапках - "file_n.txt". Тоді пошук файлу ведеться у поточному розділі активного диску, якщо ж пошук буде не вдалим, система закінчує його у спеціальному розділі для header - файлів, як і в першому випадку. Найбільш популярним у початківців є включення файлу stdio.h:

```
#include<stdio.h>

main()

{

printf ("Some informations about main... \ n");
```

```
return 0;
}
```

Слід зауважити, що файли включення іноді можуть вміщувати в собі командні рядки включення інших файлів, причому без ніяких обмежень у глибину вкладання. Цей прийом широко застосовується при розробці великих програмних проектів.

1.4. Тип даних. Базові типи та перетворення типу

Що таке тип даних? Сформулювати це поняття можна й так - множина значень плюс перелік дій або операцій, які можна виконати над кожною змінною цього типу. Вважається, що змінна або вираз належать конкретному типу, якщо його значення міститься в області припустимих значень цього типу.

Арифметичні типи об'єднують цілі та дійсні, цілі типи у свою чергу - декілька різновидів цілих та символьних даних. Скалярні типи включають в себе арифметичні типи, покажчики та перелічувані типи. Агрегатні або структуровані типи містять в собі масиви, структури та файли. Нарешті функції представляють дещо особливий клас, який слід розглядати окремо.

Всі базові типи можна перелічити у такій послідовності:

Char- символ (один байт)
Int - ціле (слово)
Short - коротке ціле (слово)
Long-довге ціле
Float - число з плаваючою комою
Double - число з плаваючою комою подвійної точності

У мові Сі, на відміну від Паскаля, використовується префіксний запис найменування, при якому на початку вказується тип змінної, а потім її ім'я. Наприклад, опис *ini name_x* означає, що ми описали цілу змінну під назвою *name_x*. Точні діапазони та розміри пам'яті для різних типів даних коливаються в залежності від конкретної реалізації. Стандарт Сі гарантує, що приміром, *int* займає не менше пам'яті, ніж *short int*, а *long int* у свою чергу не менше ніж *int*.

Крім того, перші з чотирьох типів можуть використовуватися із модифікаторами *unsigned* (без знаку) та *signed* (із знаком). Іншими словами, за замовчуванням змінні цих типів можуть мати додатні та від'ємні значення (не обов'язковий опис *signed)*, або використовувати лише додатні числа (обов'язковий опис *unsigned)*, при якому кількість змінних збільшується вдвічі порівняно із їх знаковими еквівалентами.

Згадаємо, що компілятор Паскаля виконує автоматичне перетворення типу даних, особливо в математичних виразах, коли найчастіше цілочисельний тип перетворюється у тип з плаваючою комою. Цей стиль підтримує і Сі, при чому значення типу *char* та *int* в арифметичних виразах

змішуються: кожний з таких символів автоматично перетворюється в ціле. Взагалі, якщо операнди різних типів, то перед тим, як виконати операцію, молодший тип "підтягується" до старшого. Результат - старшого типу. Отже,

char ma short перетворюються в int;
float перетворюється в double;

якщо один з операндів long, тоді другий перетворюється відповідно до того ж типу, і результат long;

якщо один з операндів *unsigned,* тоді другий перетворюється відповідно до того ж типу, і результат буде *unsigned.*

Тип результату кожної арифметичної операції виразу є тип того операнду, який має у відповідності за наведеним малюнком більш високий тип приведення. Впорядкованість типів показують вертикальні стрілки (від *int* до *double),* горизонтальні стрілки задають порядок автоматичного приведення у виразі (див. Рис.2).

Отже, процес перетворення відображається такою ієрархією:

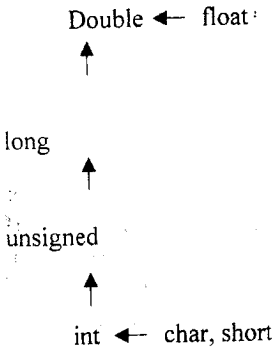

Рис. 2. Впорядкованість типів

Але окрім цього, з'являється можливість і примусового перетворення, щоб дозволити явно конвертувати (перетворити) значення одного типу даних в інший. Загальний синтаксис перетворення типу має два варіанти:

1) (новий тип) вираз;
2) новий тип (вираз);

Наприклад, обидва варіанти перетворення виглядають так:

```
Char letter=a;
Int nasciil=int (letter);
Long iascil=(long)letter;
```

2. КЕРУЮЧІ СТРУКТУРИ МОВИ

2.1. Оператори та операції.

Основу будь-якої мови складають оператори. Оператором – виразом називають вираз, вслід за яким стоїть крапка з комою. У *Ci* крапки з комою використовуються для розділу операторів. Взагалі можливо згрупувати усі оператори у такі класи:

присвоювання;

виклик функції;

розгалуження;

цикл.

Проте оператори найчастіше відносяться до більш ніж одного з чотирьох класів. Наприклад оператор if (a = funct_1(b+c) >d) складається із представників таких класів: присвоювання, виклик функції та розгалужування. У тому і полягає гнучкість *Ci*, що є можливість змішування в одному операторі операторів різних класів. Проте навряд чи слід цим зловживати – програма може вийти правильною, проте надто заплутаною та нечитабельною.

З операторами тісно пов'язані операції. У *Ci* існує такий розподіл на операції:

1) арифметичні операції;
2) операції відношення;
3) логічні операції;
4) операції присвоювання;
5) операції обчислення розміру *sizeof( )* ;
6) операції слідування (кома).

2.2. Арифметичні операції та операції відношення

До цих операцій належать відомі +, -, /, *. Всі вони визначені для змінних типу *int, char, float.* Операція частки за модулем - % не визначена для змінних типу *float,* тому що вона обчислює залишок від частки цілих чисел.

Перевірка та рівність (= =) , перевірка на нерівність (! =) , відомі менше (<) та більше (>) і їх комбінації із знаком (=) складають групу операцій відношення. Ці операції виробляють результат типу *int.* Якщо дане відношення між операндами істинне, то значення цього цілого дорівнює 1, інакше 0.

Поширеною помилкою для новачків та особливо програмістів, що переходять від Паскаля до *Ci*, є використання операції = замість необхідної = = . Не слід плутати операції присвоювання та порівняння, адже їх неправильне використання на відміну від Паскаля не викличе помилок компіляції і ця помилка може дорого коштувати, суттєво впливаючи на вихідні результати програм.

2.3 Логічні операції

&& операція і (and)

|| операція або (or)

! заперечення (not)

Взагалі, результатом логічної операції є всі значення, відмінні від нуля, інтерпретуються як істинні.

Результат логічної операції 1, якщо істина, або нуль у протилежному випадку.

За допомогою оператора заперечення можливо інвертувати результат будь-якого логічного виразу за допомогою ! . Наприклад, вираз ! (a < b) означає вираз (a>= b).

2.4. Операції присвоювання

До цієї групи належать:

= += -= *= /= ++ --

Операції ++ (інкремент) та -- (декремент) можуть бути префіксними та постфіксними. Операція присвоювання = є найбільш типовою що відповідає у Паскалі := . Решта чотири операції є скороченою формою операції присвоювання:

a + = b означає a = a + b

a - = b означає a = a - b

a * =b означає a = a * b

a / = b означає a = a / b

Семантика операції збільшення ++ (інкремент) та зменшення -- (декремент) така:

-+ **a** (**--a**) – збільшує або зменшує значення **a** на одиницю після її використання у виразі.

Модифікатор const попереджує будь-які присвоювання даному об'єкту, а також інші ефекти, що впливають на зміну значення. Покажчик const також не може бути модифікований, хоча сам об'єкт, на який він вказує, може змінюватися. (про покажчик йтиметься окремо). Наприклад якщо є опис:

```
const float  pi = 3.1415926;
const maxint = 32767;
char *const str = "Hello, P…!"; / покажчик – константа
char const *str2  = "Hello!"; / / покажчик на константу
```

Тоді такі оператори є неприпустимі:

```
pi = 3.0;    /* присвоювання значення константі (?)* /
i = maxint++;      /* інкремент  константи (?)* /
str = "The student…?"; /* str вказує на щось інше (?)* /
```

Використання одного лише модифікатора const еквівалентно const int.

2.5 Операції обчислення розміру sizeof( ) та оператор слідування (кома)

Дані операції обчислюють розмір пам'яті, необхідний для розміщення в ній виразів або змінних вказаних типів. Операцію sizeof( ) можна застосовувати до констант, типів або змінних, у результаті чого буде отримано число байт, що відводяться під операнд. Приміром, sizeof(int) обчислює число байт для розміщення змінної типу int. У наступному фрагменті операція sizeof( ) застосовується для змінної – виразу:

```
char buf [100];

while (getline(buf, sizeof(buf-1)!=EOF)

printf (\n"%d",buf);
```

Оператор слідування кома застосовується для зв'язування між собою виразів. Список виразів, розділених між собою комою, обчислюється зліва направо:

```
вираз 1, вираз 2 ... вираз n;
```

Подібні операції виконуються зліва направо. Якщо, приміром, є потреба перед введенням символу з клавіатури перевірити його на тотожність іншому, можна використати наступний оператор розгалужування (див. Розділ II.7):

```
If (c=getchar(),c>'a');
```

Оскільки вираз обчислюється зліва направо, саме друга перевірка є домінуючою при визначенні результату роботи даного оператора.

В контекстах, де кома має спеціальне призначення (скажімо у будь-якому списку фактичних параметрів функції, або у списках ініціації), операція кома може використовуватися лише у дужках.

2.6. Оператори введення та виведення

Якби там не було, реальні програми на Сі не уявляються без інтенсивного використання операцій введення та виведення. Файл включення stdio.h містить в собі деякі макровизначення та змінні, що використовуються у бібліотеці введення – виведення. Найпростіший механізм введення – зчитування по одному символу із стандартного вхідного потоку (за звичаєм, термінал користувача) за допомогою функції getchar(), звертання до якої надає наступний вхідний символ. Ця функція, якщо вона зустрічає ознаку закінчення файлу, повертає значення EOF, що відповідає –1. Замість терміналу можливо підставити будь-який файл: якщо ваша програма використовуватиме progname введення організується таким чином:


```
progname < file1
```

Звертання до putchar(c) видає символ с у стандартний вихідний потік, що за замовчуванням також вважається терміналом. Аналогічно введенню вихідна інформація теж може перенаправлятися в інший файл, для чого використовується >. Якщо використовує putchar, то команда

```
progname > outfile
```

викличе перенаправлення стандартного вихідного потоку у файл outfile, а не на термінал. Розглянемо програму, що забезпечує перетворення символів вхідного потоку у відповідні символи нижнього регістру:

```
#include <stdio.h>

main()

{

int c;

while ((c=getchar())!=EOF)

    putchar(isupper(c) ? tolower(c): c); }
```

Функція isupper(c) перевіряє, чи відноситься її параметр до букв верхнього регістру. Якщо умова виконується, повертається ненульове значення, в протилежному випадку - 0. функція tolower (c) переводить букви з верхнього регістру у нижній.

Дві функції printf ( ) та scanf ( ) дозволяють переводити числові величини в символьне представлення та навпаки. Крім цього, вони дозволяють формувати форматні рядки. Рядок параметрів цих функцій включає звичайні символи, що просто копіюється у вихідний потік, та специфікації перетворень, кожна з яких викликає перетворення та друк чергового параметра (аргументу) функції. Кожна специфікація перетворення починається з символу % та закінчується деяким символом, що задає перетворення:

d - десяткове представлення;

o - вісімкове представлення без знаку;

x - вісімкове представлення без знаку;

h - десяткове представлення без знаку;

c - параметр як один символ;

s - параметр-рядок;

e - Є – формат представлення величини float та double;

f - звичайне представлення величини float та double;

Якщо після символу % не слідує символ перетворення, то цей символ друкується. Приміром, %% виведе сам символ %.

Поширеною помилкою використання scanf ( ) у початківців є, наприклад, таке звертання: scanf ("%d", n); замість scanf ("%d", &n).

Параметри цієї функції обов'язково повинні бути покажчиками! Як наприклад, можна розглянути примітивний калькулятор для обробки вхідної інформації:

```
#include <stdio.h>
main ( )
{
double sum, v;
sum = 0;
while (scanf("%f",&v)!=EOF);
printf ("\t%.2f\n", sum += v);
}
```

Програма scanf ("%d", n) зупиняється, коли вичерпується інформація з керуючого рядка або коли виявляється невідповідність вхідної інформації заданим специфікаціям.

2.7. Оператори розгалуження

Усі мови мають можливість умовного розгалуження, що відтворюють відповідні алгоритмічні процеси. Основний оператор цього блоку в C_i if ... else немає ключового слова then, як у Паскалі, тому що реалізація мови віддає перевагу стислості, проте обов'язково вимагає, аби умова, що перевіряється, розміщувалася б у круглих дужках. Оператор, що слідує за логічним виразом, є then-частиною оператора if..else.

Синтаксис команд:

1)

if (<логічний_вираз>)

<оператор 1>;

[else

<оператор 2; >]

Умова істинна, якщо вона не дорівнює нулю, в інших випадках вона невірна. Це означає, що навіть від'ємні значення розглядаються як істині. До того ж, умова, що перевіряється, повинна бути скалярною, тобто

зводиться до простого значення, яке можливо перевірити на рівність нулю. Взагалі не рекомендується використання змінних типу float або double в логічних виразах перевірки умов з причини недостатньої точності подібних виразів. Більш досвідчені програмісти скорочують оператори типу:

```
if(вираз !=0) оператор;
```

до наступного :

```
if(вираз) оператор;
```

Обидва логічні вирази функціонально еквівалентні, тому що будь-яке ненульове значення розцінюється як істинна. Це можна довести у такому фрагменті:

```
//застосування умовного розгалуження
#include <stdio.h>
main ( )
{
int number;
int ok;
printf ("введіть число з інтервалу 1...100");
scanf ("%d",& number);
ok=(1<=number)&&(number <=100);
if(!ok)
printf ("Не коректно !!\n");
return ok;
}
```

Змінній ок присвоюється значення результату виразу: ненульове значення, якщо істина, і в протилежному випадку – нуль. Умовний

оператор if (!ок) перевіряє, якщо ок дорівнюватиме нулю, то !ок дасть позитивний результат й відтоді буде отримано повідомлення про некоректність, виходячи з контексту наведеного прикладу.

2)

<логічний_вираз> ? <вираз_1>:<вираз_2>;

Якщо значення логічного виразу істинне, то обчислюється вираз_1, інакше вираз_2.

3)

```
switch (<вираз цілого типу>)
{
case значення_1:
послідовність_операторів_1;
break;
case значення_2:
послідовність_операторів_2;
break;
.....
case значення_n:
послідовність_операторів_n;
dreak;
[default:
послідовність_операторів_n+1;]
}
```

Вітка default (може не описуватися, про що свідчить наявність [ ]) означає, що якщо ні одна з вище наведених умов не задовольнятиметься, тобто вираз цілого типу не дорівнює жодному із значень, що позначенні у case – фрагментах, керування передається за замовчуванням в це місце програми. Треба також зазначити обов'язкове застосування оператора

break у кожному із case – фрагментів (цей оператор застосовують для негайного припинення виконання операторів while, do, for, switch), що негайно передасть керування у точку програми, що слідує відразу за останнім оператором у switch – блоці.

2.8. Оператори циклу

Оператори циклу з передумовою:

While (<логічний вираз>)

оператор;

цикл закінчується у таких випадках :

1) умовний вираз у заголовку обертається в нуль;
2) у тілі циклу досягнуто місця, де розташований оператор break;
3) у тілі циклу виконаний оператор return;

У перших двох випадках керування передається оператору, розташованому безпосередньо за циклом, у третьому випадку активна на той момент функція завершує свою роботу, повертаючи якесь значення.

Знову ж таки частою помилкою програмістів,які працювали раніше на Паскалі, є використання замість оператора порівняння (= =) оператора присвоювання (=). Наприклад, наступний фрагмент

```
//невірне використання оператора циклу
int main(void)
int j = 5;
while (j = 5)
{
printf ("\n j=5 \ n");
j++
}
```

Компілятор C_i попередить про некоректне присвоювання в даному випадку, виправити які особливих труднощів не викликає. Втім часто такий цикл використовується для перевірки відповіді користувача на питання з програми (так чи ні ?).

```
// фрагмент використання while
printf ("\n підтверджуєте ? так чи ні ?(y\n);");
scanf ("%c, &ch");
while (ch!='y' &&ch!='n')
{
printf ("\n відповідайте так чи ні ...(y/n);");
scanf ("%c",&ch);
}
```

Тіло циклу почне використовуватися якщо користувач введе будь-який символ відмінний від y, або n. Цикл виконується доти, доки користувач не введе y або n.

Цікаво розглянути і наступний приклад, що застосовує оператор while у функції підрахунку факторіала:

```
long factorial (int number)
{
long total;
total=number;
while (number - - )
total *=numder;
return total;
}
```

2) Синтаксис циклу з постумовою

do

operator;

while (<логічний_вираз>);

Ситуації, що приводять до виходу з циклу, аналогічні приведеним для циклу while із передумовою. Характерним є те, що тіло циклу виконується хоча б один раз. На відміну від Паскаля, в якому цикл з постумовою repeat operator until, умова виконується, коли умова не вірна, цикл do, while припиняє використовуватись, коли умовний вираз обертається в нуль (стає невірним). Нехай потрібно лексикографічно порівняти два рядки (два масиви). Головна ідея алгоритму порівняння – перегляд кожного із рядків доти, доки не буде досягнутий кінець найкоротшого з них, або не виявиться їх невідповідність. Цикл do while, що виконує вказані дії, виглядає так:

minlength= (len1<len2)? len1:len2;

do

index++;

while (index < minlength || str1[index] = = str2[index]);

3)

Оператор циклу for:

Цикл for найбільш загальна і могутня форма аніж аналогічний оператор у Паскалі.

Синтаксис:

for ([ініціалізація];[перевірка_умови];[нове_значення])

оператор;

Звернемо увагу на те, що кожен з трьох виразів може бути відсутнім. Перший вираз служить для ініціалізації індексу, другий – для виконання

перевірки кінця циклу, а третій вираз – для зміни значення індексу. Формально роботу циклу можна описати такими кроками:

1 – якщо перший вираз присутній, то він обчислюється;

2 – обчислюється другий вираз (якщо він присутній). Якщо умова виробляє значення 0, тобто вона не вірна, цикл припиняється, у протилежному випадку він буде продовжений;

3 – виконується тіло циклу;

4 – якщо присутній вираз 3, то він обчислюється;

5 – надалі перехід до пункту під номером 2.

Поява у будь-якому місці циклу оператора continue приведе до негайного переходу до пункту 4.

Приклад використання циклу for:

```
//друк парних чисел у проміжку від 500 до 0
#include <stdio.h>
int main (void)
{
long I;
for(I=500; i>=0; i-=2)
printf ("\n%ld",i);
printf("\n");
}
```

Для того, аби продемонструвати гучність цього циклу, розглянемо наступні варіанти цієї ж програми. У першому випадку можливо представити весь перелік обчислень лише в одному операторі for, за яким слідує порожній оператор:

```
#include <stdio.h>
int main (void)
{
```

```
long i;
 for (i=500; i>=0; printf ("\n%ld",i), i-=2 );
}
```

Другий варіант використовує continue:

```
#include <stdio.h>
int main (void)
{
long i;
for (i=500; i>=0; i--)
if(i%2 = =1)
continue;
else
printf ("\n%ld", i);
printf ("\n");
}
```

Справа програміста, який з варіантів обрати, надати перевагу більш стислому викладанню або навіть взагалі скористатися іншим оператором. Цікаво, що цикл, який розглядається :

```
for (вираз1; вираз2;вираз3)
оператор;
```

можна звести до циклу while таким чином:

```
вираз1;
while (вираз 2)
{
оператор;
```

вираз3;

}

Інша справа – чи є в цьому необхідність? Не завжди гучність переважає стисліcть та навпаки. Справа за конкретною ситуацією.

3. СТРУКТУРОВАНІ ТИПИ ДАНИХ

3.1. Покажчики та масиви

Покажчик (вказівник, посилання) - це змінна, що містить в собі адресу іншої змінної. Цей тип в Сі використовується набагато інтенсивніше, аніж, скажімо, у Паскалі, тому що іноді деякі обчислення можна виразити лише за його допомогою, а частково й тому, що з ним утворюються більш компактні та ефективніші програми, аніж ми використовували б інші засоби. Недаремно існує твердження - аби стати знавцем Сі, потрібно бути спеціалістом по використанню покажчиків. З іншого боку цей чудовий засіб потребує великої передбачливості та акуратності, адже можна утворювати посилання на найбільш несподівані точки, проте при деякій дисципліні з допомогою покажчиків можна досягнути розуміння та простоти. Саме про це і йтиметься нижче.

Змінна типу покажчик оголошується подібно звичайним змінним із застосуванням унарного символу "*". Унарна операція "*" трактує свій операнд як адресу деякого об'єкта ("вказівник або покажчик на ...") та використовує її для вибірки вмісту. Далі, унарна операція "&" означає адресу об'єкта. Тоді, наприклад, вираз типу *"px=&x"* означає, що змінній *"px"* присвоюється адреса змінної "x", або, як можна сказати , *"px"* посилається на
"X".

Отже, розглянемо фрагмент:

```
Void func ()
{
i
nt x;

int *px; /* px — покажчик на змінну типу int */
px = & x; /* адреса змінної x заноситься в px */
```

***px=22; /* число зберігається за адресою, на яку вказує px */**

}

Розглянемо вищенаведений фрагмент на конкретному прикладі. Уявимо, що функція займає область пам'яті, починаючи з адреси *0x100.* Крім того, x знаходиться за адресою *O x 102, а px – O x 106.* Тоді перша операція присвоювання, коли значення *& x (0 x 102)* зберігається в *px,* матиме вигляд, зображений на рис. 3.1.

px =&x; ***px = 22;**

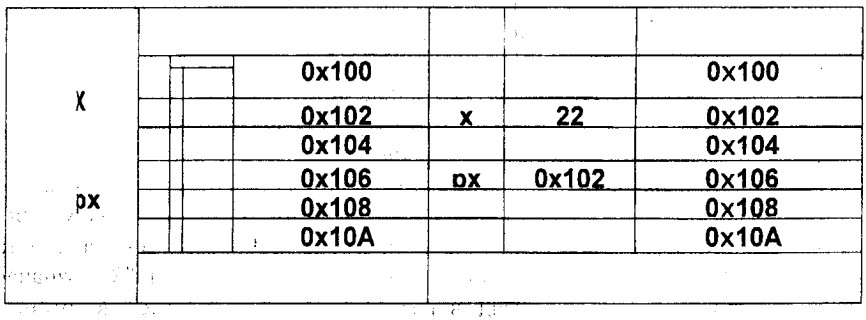

Рис. 3. 1 *Рис. 3. 2*

Наступну операцію, коли число *22* зберігається за адресою, яка знаходиться в *px* та дорівнює *O x 102* (адреса x), відображає *рис. 3.2.* Запис **px* надає доступ до вмісту комірки, на яку вказує px

Наступна програма демонструє найпростіше використання покажчиків, виводячи звичайну послідовність літер алфавіту;

```
# include <stdio.h>
char c; /*змінна символьного типу */
mam()
{
```

```
char *pc; /^покажчик на змінну символьного типу*/
pc = &c;
for (c = 'A '; c <= 'Z'
;c++)
printf (" %c", *pc);
return 0;
}
```

У операторі printf(" % c", * pc) має місце розіменування покажчика (*pc) - передача у функцію значення, що зберігається за адресою, яка міститься у змінній pc. Щоб дійсно довести, що pc є псевдонімом c, спробуємо замінити *pc на c у виклику функції - програма працюватиме аналогічно. Оскільки покажчики обмежені заданим типом даних, типовою серйозною помилкою їх використання буває присвоєння адреси одного типу даних покажчика іншого типу, на що компілятор реагує таким чином:

"Suspicious pointer conversion in function main ()"

На *TC* це лише попередження (підозріле перетворення покажчика у функції main ( )(?!)), і якщо на нього ніяк не відреагувати, то програма працюватиме й надалі (адже помилку зафіксовано не буде) і залишається лише здогадуватися, який буде результат. Зазначимо, що компілятор C++ з приводу такого "підозрілого перетворення" пішов все-таки далі: він просто відмовляється працювати, видаючи повідомлення про помилку.

Відповідальність за ініціалізацію покажчиків повністю покладається на програміста. Взагалі існує декілька методів присвоювання змінній - покажчику осмисленого значення. Крім вже згаданих вище (присвоєння покажчику адреси змінної або значення іншого покажчика, до цього часу правильно проініціалізованого) можливим також є використання функції розподілу пам'яті (alloc, malloc).

x = (int *) malloc (sizeof(int));

У якості аргументу функції динамічного розподілу пам'яті *malloc* задається кількість байтів, яку слід зарезервувати. Функція повертає покажчик на тип **void,** тому подібна змінна сумісна з будь-яким покажчиком *(void *x).* В разі. коли в пам'яті не вистачить місця для розміщення цілого, функція поверне *0;*

тому для коректного опрацювання ситуації доцільно користуватися такою перевіркою:

```
If ((int*) malloc (sized (int))) !=0)
{
*x =../* ініціалізація */
}
```

У мові *Сі* покажчики та масиви є еквівалентами, отже їх слід розглядати одночасно. Масив (або вектор) - неперервний блок пам'яті, що містить в собі множину елементів одного й того ж типу. Оголошується шляхом вказування розмірності (кількості елементів), що є додатним, цілим константним виразом, розташованим у квадратних дужках:

Тип імя_масиву [розмір масиву];

Компілятор відведе під масив пам'ять розміром *(sizeof(mun)* розмір масиву)* байтів. Приміром, опис float *vector_f[25]* оголошує вектор з 25 елементів типу *float*. Ініціалізація масивів, тобто присвоєння початкових значень є можливим лише для змінних класу пам'яті *extern* або *static* (масиви, віднесені до *auto register*, ініціюватися не можуть):

```
Static int group [5] = [136,2299,348,22,117];
```

На відміну від Паскаля, усі масиви індексуються, починаючи з 0. Так масив 5 цілих чисел містить в собі 5 елементів, що нумеруються таким чином: group [O], group [I], group [2], ..., group [4]. Це означає, що значення індексу повинні належати діапазону від 0 до величини на одиницю меншу, ніж розмір масиву, вказаний при його описі. Найбільш серйозною помилкою використання масивів є вихід за границю масиву, скажімо спроба звернутися до неіснуючого елемента group [5] і т.д. Помилки виникають, коли робиться спроба записати або прочитати значення за адресою пам'яті, яка не була вказана в програмі. Як наслідок, можливі як невірні результати обчислень, так і відмова роботи системи в цілому. На відміну від Паскаля, який попередить вас у такій ситуації, при використанні Сі лише програміст несе відповідальність за перевірку виходу за границю масиву!

Слід пам'ятати, що ім'я масиву фактично є константою - покажчиком, що посилається на перший з елементів, послідовно розташованих в пам'яті. Важливо відмітити, що мова йде саме про константу! Початкова адреса масиву визначається компілятором під час опису масиву і ніколи не може бути перевизначеною.

Далі - про тісний зв'язок масивів та покажчиків. Якщо розглянути опис char *strl, на перший погляд він являє собою опис звичайної змінної - покажчика на тип *char*. Однак, крім цього, такий опис визначає масив з початковою адресою str і поки що невизначеною кількістю елементів. Фактично будь-яке згадування імені масиву означає покажчик на початок цього масиву, тобто ім'я масиву є вираз посилання. Це призводить до

ряду корисних наслідків. Нехай є опис масиву *int a[f0]* та *int *pa* - покажчик на ціле. Присвоювання *pa= &a[0]* встановлює *pa* покажчик на нульовий елемент *a*, тобто *pa* містить адресу *a[0]*. Отже, якщо ім'я масиву є синонім для місцезнаходження нульового елемента, це рівняння можна записати і у такому вигляді: pa = a. Не дивно, що значення *a[i]* можна записати як **(a+i)*. Ці форми абсолютно еквівалентні. Якщо застосувати операцію *&* до обох частин цього рівняння, отримуємо, що *&a[i]* та a+i також ідентичні: *a+i* - адреса *i-того* елемента відносно a. З іншого боку, якщо *pa* - покажчик, то його можна використовувати з індексом: *pa [i]* ідентично **(pa+i)*. Отож, будь-який масив та індексний вираз можна записати як покажчик та зміщення або навпаки, причому це можна робити навіть в одному операторі. Як змістовний приклад використання масивів та покажчиків розглянемо дві реалізації задачі підрахунку появи кожної літери у вхідному потоці. У першій реалізації описується масив з 256 цілих та використовуються звичайні операції без застосування покажчиків.

/* програма підрахунку частоти появи символів у текстовому файлі з використанням масивів. Version 1 */

```
# include <stdio.h>
long fr- [256];
main()
{
int
i, ch;
while(ch=getchar(),ch(=EOF)
fr-[ch]++;
for (i=0; <256; i++)
printf("/n Частота символу %d+%d/n",i,fr[i]);
}
```

У циклі з вхідного потоку вводяться символи, перевіряється символ ch на збіг з EOF, та якщо такого збігу немає, то змінюється відповідне значення масиву fr, причому у якості індексу виступає ch.

У другій реалізації той самий цикл, проте тут використано адресну арифметику:

/*програма підрахунку частоти появи символів у текстовому файлі з використанням масивів Version 2*/

```
# include<stdio.h>
long fr[256];
main ()
{
int
i,ch;
while (ch=getohar(),ch!=EOF)
```

```
(*(fr+ch))++;
for (i=0; i <256; i ++)
printf( "/n Частота символу %d=%d
/n", i, fr [i]);
}
```

Важливим підкласом одновимірних масивів є масиви символів, або рядки. У стандартній бібліотеці *(string.h)* містяться функції роботи з рядками. (Опис деяких з них ви знайдете у другій частині посібника). Вважається, що кожний рядок завершується нульовим символом - '/0'. Якщо рядки задаються у вигляді літералів, то компілятор автоматично приєднує нульовий символ у кінець рядка. В результаті формування рядка символ за символом, відповідність за це покладається на програміста.

3.2. Структури

Сукупність елементів даних, можливо й різних типів (стандартних чи визначених програмістом), згрупованих під одним ім'ям з метою спрощення маніпулювання ними представляють собою структури. Оголошення структур виконується за допомогою ключового слова *struct* . Наприклад, розглянемо опис

```
struct mystruct{
        };
struct mystruct s, *ps;
/* s - змінна типу структури mystruct; ps покажчик на тип mystruct */
```

Крім того, можливий і такий опис з використанням typedef:

```
typedef struct mystruct {...} MYSTRUCT;
MYSTRUCT s; // те саме, що й struct mystruct s;
```

Доступ до окремої компоненти структури забезпечується операторами вибору: . (прямий селектор компоненти) та -> (непрямий селектор компоненти), наприклад,

```
struct mystruct{
int i;
char str [21];
double d;
} s, *sptr=&s;
s.i=3;
sptr->d=1,23;
```

Пам'ять розподіляється у структурі покомпонентно, зліва-направо, від молодших до старших адрес пам'яті. У наступному прикладі

```
struct
mystruct {
    inf i;
    char sir [2];
    double d;
    }s;
```

об'єкт s займає достатню кількість пам'яті для розміщення цілочисельного значення, символьного рядка та значення змінної типу подвійної точності.

З метою ознайомлення з цим типом даних розглянемо найпростіший приклад представлення поняття «дата», що складається з декількох частин: число (день, місяць, рік), назва тижня та місяця:

```
    struct date
    {
    int
day;
    int month;
    int yaer;
    char day name [5];
    char mon _ name [4];
    } data_ l,data 2;
```

Слід відзначити, що на відміну від описів інших типів даних, опис структури не виділяє місця у пам'яті під елементи структури. Її опис визначає лише так званий шаблон, що описує характеристики змінних, що будуть розміщуватися у конкретній структурі. Щоб ввести змінні та зарезервувати для них пам'ять, необхідно або після фігурної дужки, що завершує опис структури, вказати список ідентифікаторів, як це зроблено у вищенаведеному прикладі, або окремо оголосити змінні типу, як ми це робимо у звичайних випадках. Ініціалізація структури подібна до тієї, що у масивах, але з урахуванням розміщення даних різного типу. Звертання до окремих елементів структури теж не викликає труднощів:

```
Data_l. year =1997;
```

Доцільним та корисним є зв'язок структур та покажчиків, який дозволяє обійти деякі складні моменти. Так опис *date *pdate* утворить покажчик на структуру типу date. Використовуючи цей покажчик, можна звернутися до будь-якого елемента структури шляхом застосування операції -> , тобто *date -> year* , або що еквівалентно операції *(*pdate). year.* Однак слід зауважити» що спільне використання цих типів потребує від програміста достатньо високої кваліфікації, аби використовувати можливості найбільш ефективно та безпомилково.

3.3. Файлові потоки

Термін файловий поток являє собою послідовну структуру інформації, що записується або зчитується з диску. З цим поняттям тісно пов'язується поточна позиція, що являє собою внутрішній покажчик файлу. Склад потоку задається структурою FILE, що визначена безпосередньо у *stdio. h.* Вона містить різні подробиці, що стосуються активного файлу:

```
typedef struct
{
short level; /*рівень буферу */
unsigned flags; /*статус файлу */
char fd; /*дескриптор файлу */
char hold; /*попередній символ, якщо немає буферу */
short bsize; /*розмір буферу */
unsigned char *buffer; /*буфер передавання даних */
unsigned char *curp; /*поточний активний покажчик */
short token; /*перевірка коректності */
}FILE
```

Функція fopen використовується для відкриття файлового потоку. Першим параметром виступає ім'я файлу, другим визначається його тип:

FILE *fopen(char *filename,char *type);

Другий параметр, що визначає тип файлу відкриття, може приймати такі значення:

"r" - відкриття файлу без дозволу на модифікацію. Файл відкривається лише для читання.

"w" - створення нового файлу тільки для запису. Перезаписує будь-який існуючий файл з таким ім'ям.

"a" - відкриття файлу тільки для додавання інформації, починаючи з кінця файлу. Якщо файл не існує, він створюється.

"r+" - відкриття існуючого файлу для читання та запису.

"rw" - читання, запис.

"w+" - створення нового файлу для читання та запису. Перезаписує будь-який існуючий файл з таким ім'ям.

"a+" - відкриває файл у режимі читання та запису для додавання нової інформації в кінець файлу. Якщо файл не існує, він створюється.

Як видно з цього опису, функція повертає покажчик на структуру FILE. Процес посимвольної обробки текстового файлу можна забезпечити, використовуючи функцію fgetc(), що аналогічна за своїми діями функції getchar(), проте, на відміну від останньої, перша потребує аргументу типу **FILE*** та, вочевидь, читатиме інформацію з будь-якого файлу, відкритого за допомогою fopen(). Як приклад, розглянемо пошук заданого рядка у файлі, що вже створений, та виведемо повідомлення про результати роботи програми.

```
/* програма здійснює пошук у файлі */
#include <stdio.h>
#include <string.h>
#include <process.h>
void mam(){
FILE* filename;
if((filename=fopen("c:\\test.txt","rt"))
==NULL)
{puts("Error Open File!!!");
exit(-l);
}
char str[256], char
len[256];
puts("Введіть рядок для пошуку.");
scanf("%s", str);
int equal=0;
while((len[0]=fgetc(filename))!=char(EOF))
{
if(len[0]=
=str[0])
{for(int i=l; i<strlen(str); i++)
if((len[i]=fgetc(fclename))= =char(EOF))
break;
len[i]='\0';
if(strcmp(str, len)= =0) { puts("Рядок знайдений.");
eq
ual=l;}
```

```
}
    }
    if(equal= =0) puts ("Рядок не
знайдений.");
    fclose(filename);
    }
```

Функції, що реалізують роботу з файлами, зібрані у бібліотечному файлі stdio.h. На жаль, об'єм посібника не дозволяє розглянути роботу з файлами більш детально. Проте опис найбільш вживаних функцій міститься у другій частині даного посібника. Надалі розглянемо більш складне завдання: потрібно створити файл з клавіатури та вивести до друку лише останні n рядків. Фрагмент утворення файлу можна представити у вигляді:

```
/* утворення файлу з клавіатури */
char ch;
FILE *FILENAME;
FILENAME=fopen("tab.txt","w");
printf( "Введення тексту завершіть натисненням tab\n");
while((ch=getchar())!='\t')
{
putc(ch,
FILENAME);
if(ch=='\n')numstr++;
}
fclose(FILENAME);
```

Окрім звичайного введення, проводився підрахунок введених рядків у змінній numstr. Це надасть змогу не витрачати час при виведенні кількості рядків, що задовольняють умови програми. Отже, заключна частина виглядатиме приблизно так:

```
/* відкриття файлу для читання та здійснення пошуку */
int last=0, i=0;
FILENAME=fopen("tec.txt","r");
if(!FILENAME).
printf ("Помилка!");
else{
prmtf ("Введіть кількість останніх рядків для перегляду \n");
```

```
        scanf("%d", &last);
        while ((ch=fgetc (FILENAME)) !=EOF)
{
if ((ch= ='\n' &&(i>=numstr-last-1))
        { while ((ch==fgetc (FILENAME)) !=EOF)
        putchar (ch);}
        else
        if(ch=='\n') i++
            }
          }

fclose(FILENAME);
        }
```

4.ФУНКЦІОНАЛЬНО-МОДУЛЬНЕ ПРОГРАМУВАННЯ

4.1. Функціональний підхід

Що означає термін програма взагалі? Просто послідовність операцій над структурами даних, що реалізують алгоритм: розв'язання конкретної задачі. Спочатку ми розмірковуємо відносно того, що повинна робити дана програма, які конкретні задачі вона повинна розв'язувати та які саме шляхи цього процесу. Буває, і це характерно для більшості задач, вихідна задача достатньо довга та складна, у зв'язку з чим програму складно проектувати та писати, а тим більше, супроводжувати, якщо не використовувати методів керування її розмірами та складністю. Для цього потрібно використати відомі прийоми функціонально-модульного програмування для структурування програм, що полегшує їх створення, розуміння суті та супровід.

Нижче піде мова про процедурне (функціональне) програмування на *Ci*. Існують досить добре розвинуті методи процедурного програмування, що базуються на моделі побудови програми як деякої сукупності функцій. Прийоми програмування пояснюють, як розробляти, організовувати та реалізовувати функції, що складають програму.

Основу процедурного програмування на будь-якій мові програмування складає процедура (походить від назви) або функція (як різновид, що саме відповідає мові програмування Сі). Функція-модуль що містить деяку послідовність операцій, її розробка та реалізація у програмі може розглядатися як побудова операцій, що вирішують конкретну задачу

(підзадачу). Однак взагалі функція може розглядатися окремо як єдина абстрактна операція. І, щоб її використовувати, користувачеві необхідно зрозуміти, інтерфейс функції - її вхідні дані та результати виконання. Легко буде зрозуміти ту функцію, що відповідає абстрактним операціям, - необхідним, для рішення задачі. Функцію та її використання у програмі можна у такому разі представляти у термінах задачі, а не в деталях реалізації. Припустимо, необхідно розробити функціональний модуль, що розв'язує таке питання: існує вхідний список певних даних, який необхідно відсортувати, переставляючи його елементи у певному порядку. Ця функція може бути описана., як абстрактна операція сортування даних, що може бути частиною вирішення деякої підмножини задач. Функція, що реалізує цю операцію, може бути використаною у багатьох програмах, якщо вона створена як абстракція, що не залежить від реалізації (контексту програми).

Функції мають параметри, тому їх операції узагальнені для використання будь-якими фактичними аргументами відповідного типу. Що є вхідними даними для функції? Вхідними даними для неї є аргументи та глобальні структури даних, що використовуються функцією. Вихідними даними є ті значення, які функція повертає, а також зміни глобальних даних, модифікації. Функціональний модуль, що не використовує глобальні дані, параметризується вхідними параметрами. Функція є операція над будь-якими аргументами відповідного типу, адже вона не оперує конкретними об'єктами у програмі. Тому її можна використовувати безліч разів з різними параметрами не тільки в одній програмі, а в інших зі структурами даних того ж типу. Інтерфейс буде зрозумілий з опису прототипу функції, а об'єкти даних, описані в його реалізації, зрозумілі з локальних оголошень функції. Тому при параметризації входу та локалізацій описів функція представляє собою тип самодокументованого модуля, який легко використовувати. Крім цього, функціям притаманна модульність, її широко використовують для надання функціям більшої ясності, можливості повторного використання, що, таким чином, допомагає скоротити витрати, пов'язані з її реалізацією та супроводом.

4.2. Функції

Їх представлення включає в собі такі моменти:

1) тип значення функції, який вона повертає (в разі, коли значення не повертатиметься, то тип функції вказується void);

2) число і тип формальних параметрів код (тіло) функції, який буде виконаний безпосередньо у місці її виклику;

3) вказання відносно видимості функції поза файлом, в якому вона визначається;

4) локальні зміни, що можуть приховувати глобальні зміни.

Слід чітко розрізняти поняття опису та представлення функцій. Перший дає можливість організувати доступ до функції (розташовує її в область видимості), про яку відомо, що вона external (зовнішня). Представлення визначає, задає дії, що виконується функцією при її активізації (виклику).

4.3. Функції, що не повертають значення

Функції типу *void* (ті, що не повертають значення), подібні до процедур Паскаля. Вони можуть розглядатися як деякий різновид команд, реалізований особливими програмними операторами. Оператор *func ( );* виконує функцію *void func ( ),* тобто передасть керування функції, доки не виконаються всі її оператори. Коли функція поверне керування в основну програму, тобто завершить свою роботу, програма продовжить своє виконання від того місця, де розташовується наступний оператор за оператором ***func().***

```
/*демонстраційна програма*/
# include <stdio.h>
void func 1(void);
void func 2(void);
main ( ){
func 1 ( );
funs 2 ( );
return 0;}
void func 1 (void)
{
//тіло
}
void func 2 (void)
{
//тіло
}
```

Звернемо увагу на те, що текст програми починається з оголошення прототипів функції – схематичних записів, що повідомляють компілятору ім'я та форму кожної функції у програмі. Для чого використовується прототипи? У великих програмах це правило примушує Вас планувати проекти функцій та реалізовувати їх таким чином , як вони були сплановані. Будь-яка невідповідність між прототипом (оголошенням) функції та її визначенням (заголовком) призведе до помилки компіляції, на яку потрібно відреагувати. Кожна з оголошених функцій має бути визначена у програмі, тобто заповнена операторами, що її виконують. Спочатку йтиме заголовок функції, який повністю збігається з оголошеним раніше прототипом функції, але без заключної крапки з комою. Фігурні дужки обмежують тіло функції. В середині функції можливий виклик будь-яких інших функцій, але неможливо оголосити функцію в середині тіла іншої функції. Нагадаємо, що Паскаль дозволяє працювати із вкладеними процедурами та функціями.

Надалі розглянемо приклад програми, що вирішує відоме тривіальне завдання – обчислює корені звичайного квадратного рівняння, проте із застосуванням функціонального підходу:

```
# include <stdio.h>
# include <stdlib.h>
# include <conio.h>
# include <math.h>
float A, B, C;
/*функція прийому
даних */
void GetData ()
{
      clrscr ();
```

```
        printf("Input A, B, C:");
        scanf("%f%f%f, &A,&B,&C);
}
/*функція запуску основних обчислень */
void Run ()
{
float D;
float XI,X2;
if((A==0)&&(B!-0))
{
        X1 = (-C)/B;
        Printf("\nRoot: %f",X1);
        Exit (0);
}
D=B*B-4*A*C;
If (D<0) printf ("\nNo roots...");
If(D==0) {
X1=(-B)/(2*A)
printf("\nTwo equal roots: Xl=X2=%f", X1);
}
if(D>0)
{
        Xl=(-B+sqrt(D))/(2*A);
        X2=(-B+sqrt(D))/(2*A);
        printfT\nRoot XI: %f\nRoot X2: %Г, XI, X2);
        }
}
/*головна функція програми
*/
void main()
{

GetDat
a();
Run();
}
```

Якщо в описі функції не вказується її тип, то за замовчуванням він приймається як тип *int*. У даному випадку обидві функції описані як *void*, що не повертають значення. Якщо ж вказано, що функція повертає значення типу void, то її виклик слід організувати таким чином, аби

значення, що повертається, не використовувалося б. Просто кажучи таку функцію не можливо використовувати у правій частині виразу. У якості результату функції остання не може повертати масив, але може повертати покажчик на масив. У тілі будь-якої функції може бути присутнім вираз *return*, який не повертає значення. І, насамкінець, усі програмні системи, написані за допомогою мови *Ci*, повинні містити функцію *main()*, що є вхідною точкою будь-якої системи. Якщо вона буде відсутня, завантажувач не зможе зібрати програму, про що буде отримано відповідне повідомлення.

4.4.Передача параметрів

Усі параметри, за винятком параметрів типу покажчиків та масивів, передаються за значенням. Це означає, що при виклику функції їй передаються тільки значення змінних. Сама функція не в змозі змінити цих значень у функції, яку викликали. Розглянемо такий приклад

```
void fn(int i)
{
i-10;
}
int
main(void)
{
int
i=0;
fn(i);
return 0;
}
```

При передачі параметрів за значенням у функції утворюється локальна копія, що приводить до збільшення об'єму необхідної пам'яті. При виклику функції стек відводить пам'ять для локальних копій параметрів, а при виході з функції ця пам'ять звільняється. Цей спосіб використання пам'яті не тільки потребує додаткового її об'єму, але й віднімає додатковий час для зчитування. Наступний приклад демонструє, що при активізації (виклику) функції копії створюються для параметрів, що передаються за значенням, а для параметрів, що передаються за допомогою покажчиків цього не відбувається. У функції два параметри - *one, two* - передаються за значенням, *three* - передається за допомогою покажчика. Оскільки третім параметром є покажчик на тип *int*, то він, як і всі параметри подібного типу, передаватиметься за вказівником:

```
#include<stdio .h>
void test(int one, int two.int *
three)
{
printf( "\n Адреса one дорівнює %p",
&one );
printf( "\n Адреса two дорівнює %p", &two );
printf( "\n Адреса three дорівнює %p", &three );
*three+=l;
}
main()
{
int al, bl;
int cl=42;
printf( "\n Адреса al дорівнює %p", &al);
printf( "\n Адреса bl дорівнює %p", &bl);
printf( "\n Адреса cl дорівнює %p", &cl);
test(al,bl,&cl);
printf("\n Значення cl = %d\n",cl);
}
```

На виході ми отримуємо наступне:

```
Адреса al дорівнює FEC6
Адреса bl дорівнює FEC8
Адреса cl дорівнює FECA
Адреса one дорівнює FEC6
Адреса two дорівнює FEC8
Адреса three дорівнює
FECA Значення cl = 43
```

Після того, як змінна **tree* в тілі функції *test* збільшується на одиницю, нове значення буде присвоєно змінній *cl*, пам'ять під яку відводиться у функції *main 0.*

У наступному прикладі напишемо програму, що відшукує та видаляє коментарі з програми на *Ci.* При цьому слід не забувати коректно опрацьовувати рядки у лапках та символьні константи. Вважається, що на вході - звичайна програма на *Ci.* Перш за все напишемо функцію, що відшукує початок коментаря ***(/*):***

```
/*функція відшукує початок коментаря
*/ void  void rcomment (int c)
{
int
d;
if(c=='/')
        if    ((d=getchar(
))=='*') in_comment ();
        else if(d=='/')
        {
        putchar
(c);
        rcomment (d);
        }

else
        {
        putchar
(c);
        putchar (d);
        }
        elseif(c=='\''||c==
""    )
        echo_quote (c);
        else
        putchar (c);
          }
```

Функція *recomment (int c)* відшукує початок коментаря, а коли знаходить, викликає функцію in comment (), що відшукує кінець коментаря. Таким чином, гарантується, що перша процедура дійсно ігноруватиме коментар:

```
/*функція      відшукує      кінець
коментаря */

        void in_comment (void)
        {
        int c, d;
```

```
        c=getchar ();
        d=getchar ();
        while (c!='*'||d!-'/')
        {
        c
  =d;
d=getchar ();
        }

        }
```

Крім того, функція *rcomment (int c)* шукає також одинарні та подвійні дужки, та якщо знаходить, викликає *echo quote (int c).* Аргумент цієї функції показує, зустрілась одинарна або подвійна дужка. Функція гарантує, що інформація всередині дужок відображається точно та не приймається помилково за коментар:

```
        /*функція відображає інформацію без
коментаря*/     void echo_quote (int c)
        {
        int
d;
        putchar (c);
        while ((d=getchar()) !=c)
        {
        putchar
(d);
        if(d=='\\')
putchar(getchar());
        }
        putchar(
d);
```

До речі, функція *echo quote (int c)* не вважає лапки, що слідують за зворотною косою рискою, заключними. Будь-який інший символ друкується так, як він є насправді. А на останок текст функції main() даної програми, що відкривається переліком прототипів визначених нами функцій;

```
        /*      головна
програма      *      /
#include<stdio .h>
        void
rcomment(int c);
        void in_commeny(void);
        void echo_quote(int c);
        main()
        {
        i
nt c, d;
        while
((c=getchar())!=EOF)
rcomment(c);
        return 0,
```

Програма завершується, коли *getchatO* повертає символ кінця файлу. Це був типовий випадок проектування програми із застосуванням функціонального підходу.

4.5. Класи пам'яті

Будь-яка змінна та функція, описана у програмі на Сі, належить до конкретного класу пам'яті, що визначає час її існування та область видимості. Клас пам'яті для функції завжди *external,* якщо перед її описом не стоїть специфікатор *static.* Клас пам'яті для змінної задається або за розташуванням її опису, або явно при допомозі спеціального специфікатора класу пам'яті, що розташовується перед звичайним описом. Усі змінні *Ci* можна віднести до одного з таких класів пам'яті:

1) **automatic** (автоматична, локальна)

Явний спосіб зустрічається рідко. Кожна змінна, описана в тілі функції (середині блоку), обмеженого фігурними дужками, відноситься до класу пам'яті для автоматичних (локальних) змінних:

```
        int anyfunc(void)
        {
        ch
ar item;
        ...
.....
```

}

Область дії локальної змінної *item* поширюється лише на функцію, в якій вона оголошена. Пам'ять відводиться динамічно під час виконання *програми* при вході у блок, в якому описана відповідна змінна. Локальна змінна тимчасово зберігається в стеку, коли функція починає свою роботу. Після закінчення своєї роботи функція знищує виділену стекову пам'ять, відкидаючи за необхідністю всі збережені змінні, тобто при виході з блоку пам'ять, відведена під усі його автоматичні змінні, автоматично звільняється (звідси й термін - *automatic).* З цієї причини декілька функцій безконфліктно можуть оголошувати локальні змінні з ідентичними іменами (це найчастіше буває з іменами лічильників циклів, індексів масивів тощо).

Отже, область видимості такої змінної розпочинається з точки її опису і закінчується в кінці блоку, в якому змінна описана. Доступ до таких змінних із зовнішнього блоку неможливий.

Застосування автоматичних змінних в локальних блоках дозволяє наближати опис таких змінних до місця їх розташування. Наступний приклад демонструє опис автоматичних змінних в середині блоку:

```
#incluide
<stdio.h>

main () {
 printf("\n Знаходимося в main().'");
 int i;
 for(i=10;i>0;i-)
 pnntf("\n%d", i);
 printf("\n");}
```

2) **register** (регістрова)

Цей специфікатор може використовуватися лише для автоматичних змінних або для формальних параметрів функції. Він вказує компілятору на те, що користувач бажає розмістити змінну не в оперативній пам'яті, а на одному з швидкодіючих регістрів комп'ютера, від чого програма виконуватиметься більш ефективніше. Звісно, це стосується перш за все саме тих змінних, звертання до яких у функції виконуватиметься найчастіше. На практиці на цей тип змінних накладаються деякі обмеження, що відображають реальні можливості конкретної машини. У

випадку надлишкових та недопустимих описів подібний специфікатор просто ігнорується.

3) **extern** (зовнішня, глобальна)

Будь-яка змінна, описана не в тілі функції (без специфікатора пам'яті), за замовчуванням відноситься до *extern* - змінних (або глобальних змінних). Глобальні змінні продовжують існувати протягом усього життєвого циклу програми. Якщо користувач не вкаже ініційоване значення таким змінним, їм буде присвоєно початкове нульове значення. Найчастіше оголошення таких змінних розташовується безпосередньо перед main();

```
/*files l. C*/
#mclude
<stdio.h>
int globalvar;
main()
{
<operators>
}
```

Будь-які оператори у будь-якій функції файлу filef.c можуть виконувати читання та запис змінної globalvar. Але це ще не все! Виявляється, що глобальні змінні завжди залишаються під контролем завантажувача програми, що здійснює збір програми із множини obj-файлів. Саме завдяки цьому до зовнішніх змінних можливий доступ з інших файлів. Для того, аби таку змінну можна було б використовувати в іншому файлі, слід задати специфікатор extern:

```
/*file 2.c*/
# include <stdio.h>
main ( )
{
extern globalvar;
printf("globalvar: %d", globalvar);
```

}

Опис extern globalvar; вказує компілятору на те, що ця змінна визначена як зовнішня та її опис знаходиться за межами данного файлу. У даному випадку опис extern розташований в середині функції, тому його дія впливає лише на одну функцію. Якщо розмістити його зовні будь-якої функції, то його дія поширеться на весь файл від точки опису.

Цікаво, якщо в середині блоку описана автоматична змінна, ім'я якої збігається з іменем глобальної змінної, то в середині блоку глобальна змінна маскується локальною. Це означає, що в такому блоці видною буде саме автоматична, тобто локальна змінна.

4) **static**(статична)

шоб обмежити доступ до змінних, дозволяючи зберегти їх значення між викликами функцій, слід оголошувати їх статичними. Статична змінна може бути *внутрішньою* або *зовнішньою.* Внутрішні статичні змінні локальні по відношенню до окремої функції, подібно автоматичним, проте на відміну від останніх продовжують існувати, а не виникають та знищуються при кожній активації функції. Це означає, що внутрішні статичні змінні є власною, постійною пам'яттю для функції:

```
int funct(void)
{
static int value=20;
...
}
```

Компілятор відведе постійну область пам'яті для змінної value та проініціалізує її значення. Ця ініціалізація не повторюватиметься щоразу при активізації функції. В подальшому змінна матиме те значення, яке вона отримала по завершенні її останньої роботи. Слід відзначити, що така змінна буде назавжди прихованою для завантажувача даної, обмежена

функцією, в якій вона була оголошена, а функції не мають доступу до статичних змінних, оголошених в інших функціях.

Зовнішні статичні об'єкти відомі в тому файлі, в якому описані, проте в інших файлах вони невідомі. Таким чином, забезпечується спосіб об'єднання даних та підпрограм,які маніпулюють ними, таким чином, що інші підпрограми та дані у будь-якому випадку не зможуть конфліктувати з ними.

4.6. Додаткові можливості функції main()

Слід зауважити, що крім цього, програма на *Ci*, що виконується, посилає функції *main()* три параметри (аргументи): argc, *argv* і *env*. Кожний з них являє собою таке;

argc - ціле число аргументів командного рядка, що посилається функції *main;*

argv - масив покажчиків на рядки. Для версії ДОС *argv[0]* визначається як повний шлях програми, що виконується, *argv[l]* та *argv [2]* відповідно вказує на перший та другий після імені програми рядок командного рядка, *argv[argc-1]* вказує на останній аргумент, *argv [argc]* містить NULL.

- *env* - також є масив покажчиків на рядки, причому кожний елемент *env[]* містить рядок типу *ENVVAR= значення. ENVVAR* - це ім'я змінної середовища.

Можливо для першого ознайомлення з *Ci* ця інформація не є обов'язковою, проте не може не зацікавити приклад програми, що демонструє найпростіший шлях використання аргументів, що посилаються функції main:

```
/*Використання аргументів функції main()*/
#include <stdio.h>
#include <stdlib.h>
void main(int argc, char *argv[], char *env[]) {
int i;
printf("Значення argc = %d\n\n", argc);
printf("В командному рядку міститься %d параметрів \n\n", argc);
for (i=0;i<argc;i++)
printf("argv[%d]: %s\n", i, argv[i]);
printf("Середовище містить такі рядки: \n");
```

```
    for (i=0;env[i]!=NULL; i++)
    printf("env[%d]: %s\n", i, env[i]);
    return 0;
}
```

Організуємо виконання *програми* з командним рядком таким чином:
C:>c:\tc\testargs.exe 1_st_arg "2_arg "34 "dummy" stop!
В результаті роботи програми ви отримаєте приблизно таке:
Значення argc = 7
В командному рядку міститься 7 параметрів
argv[0]: c:\tc\testargs.exe
argv[1]: 1_st_arg
argv[2]: 2_arg
argv[3]: 3
argv[4]: 4
argv[5]: dummy
argv[6]: stop!

Середовище містить такі рядки:
env[0]:
COMSPEC=C:\COMMAND.COM
env[1]: PROMPT==pg
env[2]:PATH=C:\SPRINT;C:\DOS;C:\TC;

Максимальна загальна довжина командного рядка, включаючи пропуски та ім'я самої програми, не може перевищувати 128 символів, що є DOS-обмеженням.

4.7 Стиль написання програм

Про засоби та стилі програмування чимало сказано й написано, проте хоч і існують цікаві рекомендації відносно цього, слід все-таки визнати, що гарний стиль був і залишається справою смаку. Компілятор з мови програмування *Сі* не є винятком - він не розрізняє стилі програм і йому абсолютно нецікаво, чи дотримуєтесь ви при написанні програм хоч трохи здорового глузду. Втім, спосіб оформлення програми перш за все орієнтує її на розуміння з боку програміста або користувача, а ніж на компілятор. Вона відображає спосіб вирішення даної обчислювальної задачі людиною. Отже, скільки існує програмістів, стільки, можливо, існує й стилів, але неможливо переоцінити важливість використання логічного та ясного стилю написання програм. Якщо над великим програмним проектом працює декілька програмістів, їм зручно використовувати саме єдиний стиль. Це допоможе уникнути великої кількості помилок та надасть можливість усім учасникам однієї справи краще розумітися у всьому проекті як цілому. Отже, програміст повинен відчувати задоволення від

вигляду гарно оформленої програми та намагатися створювати саме такі програмні продукти. Зрештою, досвід у цьому переконує.

Щодо універсальних рекомендацій по стилю програмування (якщо про таке взагалі можна говорити), то їх можна викласти у двох таких кроках відносно побудови програм:

чітко сформулюйте основну ідею програми;

створіть програму, що відповідає структурі цієї ідеї.

5. ЗАВДАННЯ ДЛЯ ПОТОЧНОГО КОНТРОЛЮ

5.1. Перелік лабораторних робіт

Коментар щодо виконання лабораторних робіт. Виконання лабораторної роботи зараховується студентові лише при наявності належним чином оформленого звіту. Звіт оформлюється на окремих аркушах паперу і здається викладачеві у встановлені строки після проведення практичної роботи на комп'ютері.

На титульному листі у заголовку вказується номер та назва лабораторної роботи. Наприклад:

ВДТУ

Кафедра ПМ та ОС

Звіт про виконання

лабораторної роботи №2

«Структуровані типи даних у мові Сі»

Наприкінці титульного листа потрібно написати хто і коли виконав лабораторну роботу. Наприклад:

Виконавець студент групи 2-ПЗ-01

Якубовський Ю.О.

(власноручний підпис)

Спеціальність прикладна математика

Керівник доцент кафедри ПМ та ОС

Власюк В. Х. *(власноручний підпис)*

Завдання отримано 26.04.2002 р.

Завдання здано 27.05. 2002 р.

Зміст роботи описує весь хід виконання роботи, тобто, які дії було виконано студентом, які теоретичні та практичні відомості було отримано

в ході вивчення даної теми, та, насамкінець, роздруківка текстів програм та використаної літератури.

Лабораторна робота N 1-2

«Програмування лінійних та розгалужуваних обчислювальних процесів. Прості типи даних та їх застосування».

1. Напишіть програму, що підраховує пропуски, символи табуляції та нового рядка у вхідній послідовності символів, що вводяться з клавіатури.
2. Напишіть програму, що друкує введену інформацію по одному слову в рядку. (Слова - послідовності символів, що містять в собі букви та цифри,та розділяються між собою пропусками).
3. Розрахуйте факторіал числа, що вводиться з клавіатури, коректно передбачивши введення від'ємних чисел.
4. Напишіть програму, що знаходить суму чисел, які передують першому від'ємному числу у введеній послідовності.
5. Напишіть програму, що знаходить корені звичайного квадратного рівняння .
6. Знайдіть найбільший загальний дільник одночасно не рівних нулю цілих чисел *a* та *b*, таких що *a>=b>==0.* (Використайте алгоритм *Євкліда*).
7. Обчисліть середнє арифметичне та середнє геометричне трьох чисел, що вводяться з клавіатури.
8. Перевірте, чи є введене число з клавіатури простим числом.
9. Обчисліть висоту трикутника, якщо відомі його площа та різниця між основою та висотою.
10. Дано три сторони трикутника A, B, C. Визначити, чи буде він прямокутним.
11. Обчисліть значення кускової функції з використанням комбінованих умов : $y=x^2+15.5x$ при $x>=-f$ та $y=2x^{3-}7.x$ при $x<-f$).
12. Напишіть програму повного дослідження сукупності коренів біквадратного рівняння. (Якщо коренів не існує, повинно бути виведене відповідне повідомлення, інакше - два або чотири корені.)
13. Відшукайте мінімальне та максимальне з десяти чисел, що вводяться з клавіатури.
14. З n чисел, що вводяться з клавіатури, визначте парні та непарні.
15. Напишіть програму, що визначає належність числа *p*, яке вводиться з клавіатури, до діапазону між min та max, що можуть коригуватися у процесі виконання програми.
16. Напишіть функцію, що перетворює літери, які вводяться з клавіатури, із великих на малі.

17. Функція y = f(x) прямує до нуля при x, що прямує до нескінченості. Протабулюйте функцію з кроком n. від нуля до x ;

a) y=2x-3/x*x+2; a=i;h=l;

b)y=x/x*x+3;a=0;h=l;

c)y=9/x*x+2;a=2;h=l;

Лабораторна робота N 3-4

" ***Керуючі структури мови. Типи, операції та вирази.*** "

1. З рядка символів, що вводяться з клавіатури, відокремити та вивести лише цифри.
2. Написати програму, що підраховує пропуски, символи: табуляції, нового рядка та видимі символи.
3. Напишіть програму, що видаляє символ, який визначається користувачем, із вхідного потоку.
4. Написати програму копіювання входу на вихід із заміною рядка з одним та більше пропусків на один пропуск. (Тут і далі, якщо не оговорюється інше, рядком вважати послідовність символів, що закінчується символом переходу на новий рядок).
5. Напишіть програму, що друкує гістограму появи різноманітних введених символів у вхідному рядку.
6. Напишіть програму друку усіх рядків довжиною більше 80-ти символів.
7. Написати програму визначення символів табуляції(Tab) та повернення (Backspace) замінивши при цьому ці символи видимими символами: символ табуляції - \t; крок назад - \b ; зворотна коса риска - \ \;

При цьому невидимі символи стають видимими .

8. Підрахуйте кількість рядків слів, символів у вхідному потоці, вважаючи за слово будь-яку послідовність символів, що не містить пропусків, символів табуляції або переходів на інший рядок.
9. Напишіть програму, що видаляє:
   a) хвостові пропуски та табуляції у кожного введеного рядка.;
   b) рядки, що складаються лише з одних пропусків ;
   c) рядки, що містять в собі лише однакові символи.
10. Напишіть функцію, що "перевертає" символьний рядок st і (розташовує символи у зворотному порядку). Утворіть код для її перевірки.
11. Напишіть програму, що "перевертає" введені символьні рядки, використовуючи: функцію з вправи №9.
12. Напишіть функцію , що приймає два аргументи argl та arg2, яка видаляє кожний символ у рядку argl, що збігається з деяким символом у рядку arg2.

13. Напишіть програму, що друкує у напрямку спаду усі дільники введеного числа.
14. Напишіть програму, що визначає діапазон значень змінних типу int, char, long, short як signed, так і unsigned шляхом друку відповідних значень із стандартних header-файлів.
15. Відшукати екстремум функції на відрізку [a,b] , запам'ятати результати та вивести на друк отриману таблицю (значення аргументів та функції), після чого надрукувати екстремальні значення, як додаткове завдання, побудуйте графік на екрані, використовуючи відомі вам графічні засоби :

a) *cos* x+sin x+e*x+l; a=0;b=l;

b)(2-lnx)/3;a=l;b=2;

c) cos x+sin x+x*x-x; a=0;b=l;

d) sinx/x; a=l; b=5;

e) (sinx*x-cosx*x)/2; a=0;b=l;

Лабораторна робота N 5-6

"Функції та їх застосування, Структуровані типи даних: покажчики, масиви та структури"

1. Побудуйте, основну структуру, що працюватиме як найпростіший калькулятор. Серед звичних арифметичних операцій визначте оператор частки за модулем %.

2. Напишіть програму, що перевіряє програму Сі на елементарні синтаксичні помилки , такі як непарні круглі, квадратні або фігурні дужки.
3. Напишіть функцію, що циклічно зсуває набір значень у масиві на одну позицію вправо.
4. Напишіть програму, що обчислює середнє значення із набору значень з плаваючою комою типу double, що зберігаються у масиві.
5. Побудуйте масив, що імітує колоду гральних карт. Напишіть функцію, яка тасує карти.

6. Напишіть програму транспонування матриці (10x10) (рядки та стовпці міняються ролями).
7. Знайдіть найбільший елемент цілочислової матриці розміром 20x20.
8. Перевірте, чи в даній цілочисловій матриці 17x17 суми елементів у всіх рядках та стовпцях рівні між собою.
9. Напишіть програму, що ініціалізує двовимірну одиничну матрицю m[s][s] (s-розмір масиву);

m[n][m]==l n==m

m[n][m]==0 n==m

10. Напишіть функцію, що оголошує параметр типу char* та повертає довжину рядка, який передається у функцію в якості аргументу.
11. Напишіть програму тестування пам'яті, що використовує покажчики для заповнення блоків пам'яті контрольними значеннями із наступною їх перевіркою.
12. Напишіть програму, що використовує покажчики для обміну двох змінних будь-якого типу. (Мається на увазі, що змінні мають однаковий розмір).
13. Напишіть програму, що видаляє усі коментарі з програми на Сі. Не забувайте коректно опрацьовувати рядки у дужках та символьні константи. (Коментарі в *Сі* не можуть вкладатися один в одного.)
14. Напишіть програму, що підраховує частоту використання слів -у тексті та виводить інформацію про підрахунок. Оформлення результатів зробіть у вигляді таблиці.
15. Оголосіть структуру, що описує телефонний номер абонента та виконайте її у тестовій програмі. Крім того, оголосіть вкладену структуру з двома телефонними номерами: один - звичайний ; другий - факсимальний.
16. Напишіть функцію, що порівнює дві структури телефонних номерів із вправи 14, та виконайте її у тестовій програмі. Функція повинна повертати "істина", якщо структури рівні, або ні, у протилежному випадку.
17. Реалізуйте пошук найдовшого слова у тексті та виведіть його на екран із вказанням про його довжину.
18. Напишіть програму роеехрг, що обчислює польський інверсний вираз із командного рядка, де кожний операнд або знак операції є окремим аргументом. Наприклад, виклик програми роеехрг 5 2 3 + * обчислюватиме 5*(2+3).
19. Напишіть програму, що друкує останні n рядків, що вводяться.
20. Розмістити рядки по довжині, надрукувавши на екрані їх у відсортованому вигляді.
21. Написати програму, що читає програму на *Сі* та друкує в алфавітному порядку кожну групу імен змінних, що збігаються у перших трьох . символах, але відрізняються будь-де у наступних літерах. Не слід розглядати слова із символьних рядків та коментарів.
22. Напишіть програму, що друкує список усіх слів та для кожного з них - список номерів рядків, в яких це слово зустрічається. Як додаткове завдання, передбачте ігнорування тих слів, що найчастіше зустрічаються, за вибором користувача.
23. Напишіть програму друкування окремих слів, що вводяться, відсортованими у оберненій залежності від частоти їх появи. Разом із словом виведіть його лічильник.

5.2. Контрольні роботи

Контрольні роботи виявляють поточний рівень підготовленості студентів. Вони містять 10 різнопланових питань у декількох варіантах. Роботи виконуються письмово та здаються викладачеві для перевірки.

1 варіант

1. Оголосити змінну var2, що означає масив з 10 покажчиків на символьний тип, та змінну Var2 - масив чисел подвійної точності. Чи є можливим таке оголошення в одній програмі, тобто чи будуть вищенаведені ідентифікатори (var2 та Var2) вважатися різними ?

2. Чи є помилки у цьому фрагменті? Які будуть результати?

```
{int n;
float nP=1.0;
printf("\n вивести номер");
scanf("%u",&n);
nP/=n;
if nP>2 then nP =2;
printf("\n R=%2,2d",nP)
}
```

3. Написати прототип та тіло функції, яка приймає два цілих значення та повертає їх середнє арифметичне.

4. Опишіть та проініціалізуйте масив 6 символьних змінних.

5. Вкажіть помилки у такому фрагменті:

```
float d= 1.5;
float *pi;
int b;
pi=&d;
b=(int) 2*pi;
b=(ini)(*pi)*3;
```

6. Чи є помилки у цій програмі? Які будуть результати?

```
#include <stdio.h>
main {
int num;
double fnum;
num = 32767;
fnum = 3.2767;
printf(" \nnum=%d", &um.);
printf("\nfun=%f,&num);}
```

7. Написати функцію, яка б переводила цілі значення кілометрів у милі.

(1.7km =1.0m).

8. Оголосіть перераховуваний тип даних для днів тижня. Утворіть три змінні такого типу, одну з яких проініціалізуйте.

9. Вкажіть помилки у такому фрагменті:

```
int i= 1;
int *pi=&i;
int b;
b=2*(&pi);
b=2*pi;
b=(*pi)*3;
```

10. Замінити цикл for() на еквівалентний цикл while у фрагменті:

```
main() {
unsigned char c;
for(c=32;(c<128);c+
+)    {    if((c%32)==0)
printf("\n");
printf("%c",c);
}
printf(
"\n);
return 0;}
```

2 варіант

1. Оголосити змінну yar1, що означає:

а) масив беззнакових цілих ; б) масив з 10 покажчиків на літерний тип; в) покажчик на змінну символьного типу.

2. Чому коректно не відпрацює така програма? Які будуть результати?

```
int value;
void main() {
int k =100 /
value;
printf(""k=%d\n",k);}
```

3. Вкажіть помилки у застосуванні функцій у такій програмі . Як їх виправити ?

```
void main() {
int x:
x=mult(l,10):}
int mult(double
i,int z)
```

```
}
printf("i*z=%f",
i*z);
```

4. Оголосити структуру, що являє собою зображення точки у тривимірному просторі: X,У,2-координати, колір зображення. Утворіть дві змінні такого типу, одну з яких проініціалізуйте.

5. Що надрукує така програма?

```
    Int a=10;
    int
main(void)
    {int    a
=30;
    print("a=%d",a); }
```

6. Написати програму, що рахує факторіал n (невід'ємного цілого).

7. Вкажіть помилки у таких операціях:

```
float d= 1.5;
float *pi;
int b;
pi==&d;
b=(int)2*pi;
b=(int)(*pi)*3;
```

8. Написати функцію, яка б приймала два масиви типу char (рядки символів) та повертала повідомлення як результат збігу рядків. Розмір рядків однаковий і передається у функцію.

5.3. Тестові завдання

Тестові завдання передбачають відповіді на 12 питань, до кожного з яких пропонується по три варіанти альтернативних відповідей. Заздалегідь відомо, що лише одна відповідь з описаних a)-b)-c) є правильною. З переліку варіантів на кожне з питань слід обрати єдиний, на ваш погляд, правильний варіант.

1 варіант

1. Наступна директива #define Clear 1 визначає, що;

a) символічна константа Clear дорівнює 1;

b) змінна Clear проініацілізована та дорівнює 1;

c) ф-ція Clear визначається як 1;

2. Опис node *WORD означає, що:

a) покажчик node вказує на змінну типу node;

b) покажчик node вказує на змінну типу word;

c) описаний масив node елементів типу word;

3. Ім'я масиву є синонімом місце розташування:

a) нульового елемента масиву;

b) першого елемента масиву;

c)п-того елемента, де п-розмірність масиву.

4. Нехай у тексті є макропідстановка:

#define max(a,b) ((a>(b)?(a): b)) Тоді рядок x=max(p+g,r+s) буде замінено на :

a)x=((p+g)>(r+s)?(p+g):(r+s)

b) x=((p+g)<(r+s)?<p+g):(r+s)

c) x=((p+g)o(r+s)?(p+g):(r+s)

5. Команда continue; викликає:

a) негайний вихід з циклів та умовних розгалужень (switch);

b) негайний перехід до наступної ітерації (перевірки умови виконання);

c) завершення виконання програми,

6. Чи є помилка в операторі
for (i=OJ=i+2;; i++J--)?

a) так, -відсутній вираз умови виконання;

b) так, використано декілька параметрів

c) ні· помилки немає.

7. Вираз типу a+=2 означає;

a) a=a+2;

b) a+2=a;

c) a=<a>a+2) ?a:a+2;

8. Опис int *a[3] означає

a) масив з трьох покажчиків на ціле;

b) покажчик на масив із трьох цілих елементів;

c) функцію, що повертає покажчик на іпі;

9. Наступний опис float *nmk.() означає

a) функцію, що повертає покажчик на float;

b) покажчик на функцію, що повертає float;

c) масив з покажчиків на float;

10. Нехай є опис float: *x. Що означає операція &x?

a) Значення за адресою, яка зберігається в x, приводиться до базового типу float;

b) адресу змінної-покажчика x;

c) операцію розіменування покажчика.

11. Операція *вираз && вираз* повертає результат

a) 1, якщо хоча б один з операторів не дорівнює 0;

b) 0, якщо обидва вирази дорівнюють нулю;

c) 1, якщо обидва вирази дорівнюють нулю.

12. Який результат виробляє операція x==y, якщо χ та y еквівалентні ?

d) 1;
e) 0;
f) NULL.

2 варіант

1. Вираз типу a-=2 означає:

a)a==a-2;

b)a-2=a;

c)a=(a>a-2) ?a:a-2;

2. Опис code *sir означає, що:

a) покажчик соае вказує на змінну типу str;

b) покажчик 5Iг вказує на змінну типу code;

c) описаний масив сосіе елементів типу str;

3. Що означає такий запис int(*a)[3]:

a) покажчик на масив з трьох цілих;

b) масив з трьох покажчиків на ціле;

c) покажчик на третій елемент масиву a;

4. Що означає такий запис int (*funk)()

a) покажчик на функцію, що повертає ціле:

b) функція, що повертає покажчик на іпі;

c) масив покажчиків на цілий аргумент;

5. Оператор while ((c=getchar()==' ') && c==-'\n');

a) здійснює пропуск пропусків та символів (\n) у вхідному потоці;

b) не має смислу, тому що параметр циклу не проініціалізоваиий та не змінюється;

c) відпрацює вхолосту у зв'язку із неможливістю виконання умови хоча б один раз.

6. Нехай є опис float *x. Що означає операція *x?

a) Значення за адресою, яка зберігається в x, приводиться до базового типу Поаі;

b) адресу змінної-покажчика x;

c) нульовий елемент масиву x;

7. Що означає операція double(x), де x-вираз?

a) x перетворюється до типу double;

b) expression не дорівнює нулю;

c) вираз expression більший або рівний нулю.

3. Операція *вираз && вираз* повертає результат

a) 1, якщо хоча б один з операторів не дорівнює 0;

b) 0, якщо обидва вирази дорівнюють нулю;

c) 1, якщо обидва вирази дорівнюють нулю.

4. Що надрукує оператор puts("n\n"),, якщо у програмі існує макропідстановка #definenl?

a)n\n;

b)n;

c) 1 .

5. Скільки разів відпрацює тіло циклу:

```
hit i=3;
while(i=3)
{       printf("\nHello,
Marry");
i=i+3;
}
```

a) Ні разу;

b) Один раз;

c) Три рази;

d) Нескінченне число ;

6. Нижче наведений фрагмент обробки лише додатних чисел масиву:

for(i=0; i<n; i++) {if(a[i]<0) operator;

/*далі обробка додатних чисел */};

Який саме operator слід використати, аби дійсно оброблялися лише додатні числа ?

a) continue;

b)halt;

c)break.;

d) exit;

7. Що надрукує такий фрагмент, якщо k==2?

Switch(k

) {

case 1 :

printf("a");

case 2 :

prmtfTb");

case 3 :

b) активізується функція double з вхідним параметром x;

c) стандартна ф-ція, що повертає *розмір* змінної x у байтах, що має тип double;

8. Що надрукує оператор printf("YES\n"),, якщо у програмі існує макропідстановка #define YES I ?

a) YES;

b) l;

c) нічого, оскільки наявна помилка ;

9. Розшифруйте польський інверсний запис (кожна операція йде вслід за своїм операндом): 12-34+*==

a) (l-2)$^{!ls}$(3+4)$^{=}$

b)(1+2)*(3-4)

c) 1*(2-3)+4;

10. Команда break; викликає :

a) негайний вихід з циклів та умовних розгалужень(5луи:сп) ;

b) негайний перехід до наступної ітерації (перевірки умови виконання конструкції);

c) завершення виконання програми.

11. У циклі while (expression) operator;
оператор виконується, доти, доки

a) expression дорівнює нулю;

b) expression не дорівнює нулю;
c) вираз expression більший або рівний нулю.

12. Операція вираз | вираз повертає результат

a) 1, якщо хоча б один з операторів відмінний від 0;

b) 0, якщо обидва вирази дорівнюють нулю;
c) 1, якщо обидва вирази дорівнюють нулю.

3 варіант

1. Який результат виробляє операція x?y:z?

a) y, якщо x не дорівнює 0;

b) x, якщо y<2;

c) **X,** ЯКЩО y>2;

2. У циклі do operator while (expression); оператор виконується доти, доки :
a) expression дорівнює нулю;

```
printf("c");
default:
printf ("d");}
```

a)ab;

b)abc,

c)abcd;

a) bed,

8. Фрагмент *'wbi\e(strl[iJ=str2[ij)* i++; означає:

a) перезапис складу одного рядка в інший
b) порівняння рядків strl [i] та str2 i]),
c) приєднання рядка strl [i] до str2 [i])
9. Вираз a[2][3] еквівалентний виразу з покажчиком:
a)T(a+2)+3);
b) T(a[2])[3]),

c) (*(*a)+2))+3),

10. Операція ++a збільшує :

a) значення змінної a до використання у виразі;
b) значення змінної a після використання у виразі;
c) є скороченим записом a+1=a,
11. Коментарем є рядок, обмежений :
a) символами /* ...*/;
b) символами (* ...*);
c) символами * ...*\;
12. Якщо локальна змінна має ім'я адаптоване глобальній змінній у програмі, то

a) локальна змінна маскує глобальну змінну;
b) глобальна змінна закриває локальну;
c) це викликає помилку використання змінних.

ЛІТЕРАТУРА

1. З.С.Проценко, П.Й.Чаленко, А.Б.Ставровський Техніка програмування мовою Сі: навч. посібник.-К.: Либідь, 1993.-224с.
2. Бабе Б. Просто и ясно о Borland C++: Пер. с англ.- М.:"Бином", 1995,-400с.
3. Беррі Р., Микинз Брайн Язык Си. Введение для програмистов: Пер. с англ. – М.:"Финансы и статистика",1998.-191с.
4. Башкин А.В., Дубнер П.Н. Работа в Турбо Си. –М. "СП Ланит", 1997.-183с.
5. Болски М.И. Язык програмирования Си. Справочник. – М.: Радио и связь, 1988.-96с.
6. Джехани Наройян програмирование на языке Си: Пер. с англ.- Мн.: Радио и связь, 1988.-270с.
7. Касаткин А.И. Профессиональное програмированик на языке Си. Пер. с англ.- Мн.: "Высш. школа", 1998.-301с.
8. Хошаба О.М. Методичні вказівки до виконання лабораторних робіт з дисципліни "Мікропроцесори та програмне забезпечення інтелектуальних систем" Частина І. Операційна система Unix. Для студентів бакалаврських напрямків 6.0915 та 6.0804.-Вінниця: ВДТУ, 1998. – 42с.
9. Стариков Ю.А. Мобильность програм и особености реализации языка Си. – М.: МП "Память", 1997. – 188с.

Навчальний посібник

Василь Харитонович Власюк
Людмила Михайлівна Круподьорова

Програмування мовою CI

Навчальний посібник

Оригінал-макет підготовлено авторами
Редактор В.О. Дружиніна
Коректор З.В. Поліщук

Підписано до друку 2.07.02р.
Формат 29.7х42 ¼ Гарнітура Times New Roman
Ум.друк.арк. 2,69
Тир. 100 прим.
Зам. № 2002-164

Віддруковано в комп'ютерному інформаційно-видавничому центрі ВДТУ
м, Вінниця, Хмельницьке шосе, 95, ВДТУ, ГНК, 9-й поверх
Тел. **(0432) 44-01-59**