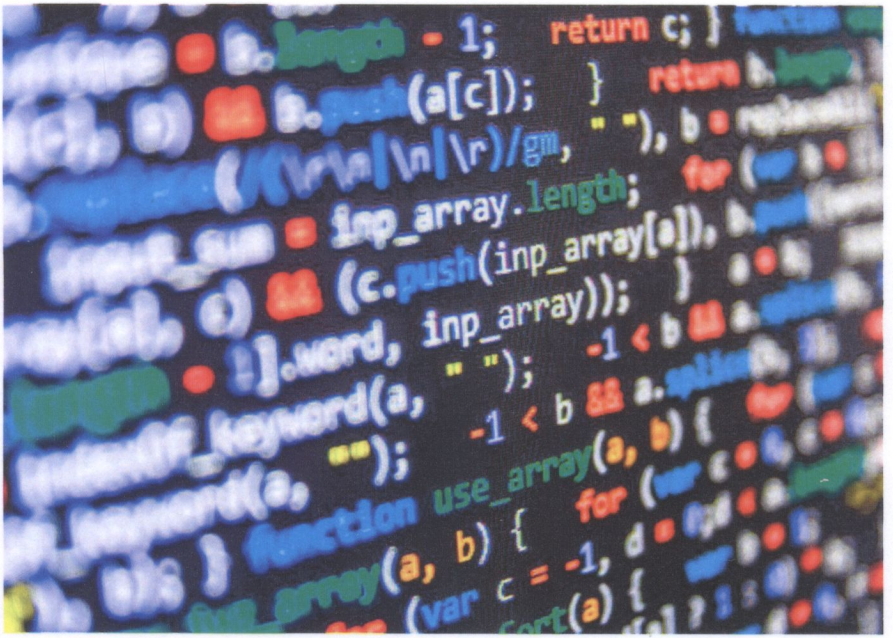
О. М. Бевз, С. М. Довгалець, Р. В. Маслій

ПРОГРАМУВАННЯ

ЧАСТИНА 2

Програмування мовою С



Міністерство освіти і науки України
Вінницький національний технічний університет

О. М. Бевз, С. М. Довгалець, Р. В. Маслій

ПРОГРАМУВАННЯ. ЧАСТИНА 2. ПРОГРАМУВАННЯ МОВОЮ С

Навчальний посібник

Вінниця
ВНТУ
2017

УДК 004.432(075)
ББК 32.973.22я73
Б36

Рекомендовано до друку Вченою радою Вінницького національного технічного університету Міністерства освіти і науки України (протокол №3 від 30.10.2014 р.)

Рецензенти:
А. Я. Кулик, доктор технічних наук, професор
А. М. Пєтух, доктор технічних наук, професор
В. І. Месюра, кандидат технічних наук, професор

Бевз, О. М.

Б36 Програмування. Частина 2. Програмування мовою С : навчальний посібник / Бевз О. М., Довгалець С. М., Маслій Р. В. – Вінниця : ВНТУ, 2017. – 149 с.

Навчальний посібник присвячений основним засадам та принципам формування програмного забезпечення засобами мови С.

В посібнику розглянуто такі питання: основні елементи мови С, прості типи даних, операції та вирази, введення та виведення даних, оператори керування, масиви, функції, рядки мови С, покажчики, робота з файлами, структури, об'єднання, перерахування, динамічне виділення пам'яті.

Навчальний посібник призначений для студентів спеціальностей: 151 – «Автоматизація та комп'ютерно-інтегровані технології», 126 – «Інформаційні системи та технології».

УДК 004.432(075)
ББК 32.973.22я73

ЗМІСТ

ВСТУП

З часу виникнення обчислювальної техніки у 1940-х роках застосування і використання комп'ютерів (*computers*) розвивається шаленими темпами. Програмне забезпечення відіграє важливу роль практично у всіх аспектах повсякденного життя: державному управлінні, банківській справі і фінансах, освіті, транспорті, індустрії розваг, медицині, сільському господарстві і юриспруденції. Кількість, якість і області застосування комп'ютерних програм (*programs*) різко збільшились. В результаті сотні мільярдів доларів витрачаються на розробку програмного забезпечення (*software*), і від ефективності цих програм залежать заробітки і навіть життя більшості людей.

В кінці 1990-х років область знань пов'язана з інформаційними технологіями (information technology) дуже сильно розрослася, в зв'язку з цим було прийнято рішення розділити її на чотири основних дисципліни – інформатика (*computer science*), програмна інженерія (software ingeneering), проектування апаратних платформ (hardware ingeneering) та інформаційні системи (information systems).

Даний навчальний посібник присвячений інформатиці, яка є базовою дисципліною для трьох основних дисциплін. Навчальний посібник є другим з серії посібників призначених для дисципліни «Програмування».

В першому розділі посібника розглядаються основні елементи мови програмування С: алфавіт мови програмування, лексеми (послідовність символів), класифікація типів даних, ключові слова, ідентифікатори, літерали, оператори, коментарі, директиви препроцесора та безпосередньо організація програм.

В другому розділі посібника розглядаються прості типи даних: час існування та область видимості змінних, цілі та дійсні типи даних.

В третьому розділі посібника міститься інформація про операції та вирази: арифметичні операції, операції приведення типів, операції присвоєння, операції інкремента і декремента, операції порівняння, операції зсуву, порозрядні та логічні операції, операція sizeof, операція умови та адресні операції.

В четвертому розділі посібника розглядаються введення та виведення даних: функція виведення printf та функція введення scanf.

В п'ятому розділі посібника розглядаються оператори керування: оператор розгалуження if, оператор розгалуження if-else, оператор множинного розгалуження switch, оператор циклу for, оператор циклу while, оператор циклу do while, оператор break, оператор continue.

В шостому розділі посібника розглядаються масиви: одновимірні та багатовимірні.

В сьомому розділі посібника міститься інформація про функції: область видимості, порожній тип void, передавання аргументів у функцію, рекурсивні функції та прототипи функцій.

У восьмому розділі посібника розглядаються рядки в С: прототипи, функції перетворення буферів, функції перевірки літер, рядок символів та символьні константи.

В дев'ятому розділі посібника розглядаються покажчики: визначення та ініціалізація покажчиків, визначення покажчиків, масиви, операція s++, операції ==, !=, копіювання рядка, посилання та оператор &, посилання як результат функції, покажчики на функцію, покажчики на void та арифметика покажчиків.

В десятому розділі посібника розглядається робота з файлами.

В одинадцятому розділі посібника міститься інформація про структури, об'єднання та перерахування.

В дванадцятому розділі посібника розглядається динамічне виділення пам'яті: функції malloc(), calloc(), realloc() та free().

Матеріал, наведений у посібнику, може використовуватися при викладанні дисциплін: «Програмування», «Обчислювальні методи», «Системне програмування», «Комп'ютерна графіка», «Захист інформації в системах управління», «Компютерні технології та програмування» а також використовуватися студентами спеціальностей: 151 – «Автоматизація та комп'ютерно-інтегровані технології», 126 – «Інформаційні системи та технології», студентами суміжних спеціальностей.

Розділи 5, 7, 8 та 10 написані О. М. Бевзом. Розділи 1, 3, 6 та 12 написані С. М. Довгальцем. Розділи 2, 4, 9 та 11 написані Р. В. Маслієм.

1 ОСНОВНІ ЕЛЕМЕНТИ МОВИ С

1.1 Алфавіт мови програмування

Будь-яка мова, у тому числі й С, має свій алфавіт – набір символів, які дозволені до використання і сприймаються компілятором.

Компілятор – програма, яка призначена для здійснення перевірки коректності програмного коду на відповідність правилам лексики, синтаксису та семантики мови, перетворення цього коду у машинну мову.

Множина символів, які використовуються у мові С, наведена у табл. 1.1.

Таблиця 1.1 – Алфавіт мови С

№ пункту	Символи
1	A B C D E F G H I J K L M N O P Q R S T U V W X Y Z a b c d e f g h i j k l m n o p q r s t u v w x y z
2	_ (символ підкреслювання)
3	0 1 2 3 4 5 6 7 8 9
4	, . ; : ? ‘ ! | / \ ~ * + –) (} { < > [ ] # % & ^ = “
5	Пробіл, табуляція, повернення каретки, нова сторінка, новий рядок

Великі та малі літери вважаються різними символами, тому що мають різні коди в таблиці кодів ASCII. Це важлива відмінність мови С від багатьох інших мов програмування, наприклад від мови Паскаля.

Символи пунктів 1 – 3 в табл. 1.1 використовуються для утворення літералів, ідентифікаторів та ключових слів.

Знаки пунктуації і спеціальні символи (пункт 4, табл. 1.1) використовуються, з одного боку, для організації процесу обчислень, а з іншого боку – для передавання компілятору визначеного набору інструкцій.

Розділові символи (пункт 5, табл. 1.1) відокремлюють один від одного об’єкти, обумовлені користувачем, такі, наприклад, як: літерали та ідентифікатори. Послідовність розділових символів розглядається компілятором як один символ (наприклад, послідовність пробілів як один пробіл).

Символьні та рядкові літерали, а також коментарі можуть містити символи не з алфавіту мови С, а, наприклад, букви російського алфавіту.

1.2 Лексеми

Лексемою називають нероздільну послідовність символів з алфавіту мови С (у найпростішому випадку один символ), які розпізнаються компілятором, тобто компілятор не аналізує компоненти, які входять в цю послідовність.

Лексемами є літерали, ідентифікатори, ключові слова, знаки операцій, а також символи пунктуації, такі як: прямокутні ([]), фігурні ({}) і кутові дужки (<>), двокрапка (:) і кома (,).

Лексеми між собою можуть бути відокремлені розділовими символами (пробіл, символ табуляції тощо) чи такими іншими лексемами, як знаки операцій. Щоб запобігти розбивці елементів лексеми на декілька частин, компілятором забороняється використання розділових символів у ідентифікаторах, багатосимвольних операціях чи ключових словах.

1.3 Ключові слова

Ключові слова – це зарезервовані лексеми, що наділені визначеним змістом. Їх можна використовувати тільки відповідно до значення відомого компілятору мови С. Список ключових слів наведено у табл. 1.2.

Таблиця 1.2 – Список ключових слів

auto	default	extern	long	sizeof	union
break	do	float	register	static	unsigned
case	double	for	return	struct	void
char	else	if	short	switch	volatile
continue	enum	int	signed	typedef	while

1.4 Ідентифікатори

Ідентифікатор – це лексема, що складається з літер, цифр і знаків підкреслення. Ідентифікатори використовуються як імена змінних, функцій, структур тощо.

Першим символом ідентифікатора має бути буква чи знак підкреслення. Для утворення ідентифікаторів можуть бути використані малі або великі букви латинського алфавіту. Мовою С допускається довільна довжина ідентифікатора, однак значення мають тільки перші 31 символ. Два ідентифікатори, для утворення яких використовуються однакові малі та великі букви, вважаються різними.

Ідентифікатори не можуть складатися з декількох слів (не можна використовувати пробіл всередині ідентифікатора) чи містити в собі символи кирилиці (український або російський алфавіт). Створюючи ідентифікатор,

потрібно пам'ятати, що він не повинен збігатися з ключовими словами й іменами функцій стандартної бібліотеки С.

Приклад 1.1. Ідентифікатори.

```
abc, ABC, A128B, a128b.
```

1.5 Класифікація типів даних

Типи даних мови С відіграють важливу роль в обробці даних. Під типом даних розуміють множину допустимих значень цих даних і множину дозволених операцій над ними. Водночас тип даних визначає і розмір пам'яті, що її займають змінні і літерали даного типу. Кожен тип даних має ім'я. Пам'ять не виділяється для типу даних, а виділяється для розміщення змінної або літерала.

У мові С виділяють такі категорії типів:

- прості типи даних;
- похідні типи даних.

Прості типи мають імена, які є ключовими словами мови.

До простих типів належать: скалярні типи і порожній тип – **`void`**.

Тип **`void`** не має значення і введений для опису функцій, які не повертають значень, та для деяких інших цілей.

Скалярні типи поділяються на цілочислові та дійсні типи.

Символьні та цілі типи даних є цілочисловим типом, для якого визначені всі операції з цілими числами.

Похідні типи визначаються на основі простих типів. Похідні типи поділяються на скалярні і структуровані (агрегатні).

До скалярних похідних типів належать:

- перерахування (**`enum`**) – множина поіменованих цілих значень;
- покажчики.

Ім'я типу у покажчиках – це один з простих типів.

Структуровані похідні типи:

- масиви;
- структури (**`struct`**);
- об'єднання (**`union`**).

Ім'ям структурованих похідних типів є ідентифікатор, що визначений користувачем.

На рис. 1.1 зображена класифікація типів даних мови С.

1.6 Літерали

Літерал – це число, символ чи рядок символів, які задаються в програмі у вигляді значення. Літерали називають також константами.

У мові С виділяють чотири типи літералів: цілі, дійсні, символьні та рядкові літерали.

Цілий літерал може подавати число в одній з форм: десятковій, вісімковій чи шістнадцятковій.

Десятковий літерал складається з однієї чи декількох десяткових цифр, причому перша цифра не повинна бути нулем.

Першим символом вісімкового літерала є нуль, наступними символами є вісімкові цифри (0, 1, 2, 3, 4, 5, 6, 7).

Шістнадцятковий літерал починається з обов'язкової послідовності 0x чи 0X, наступними символами є шістнадцяткові цифри (0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A, B, C, D, E, F).

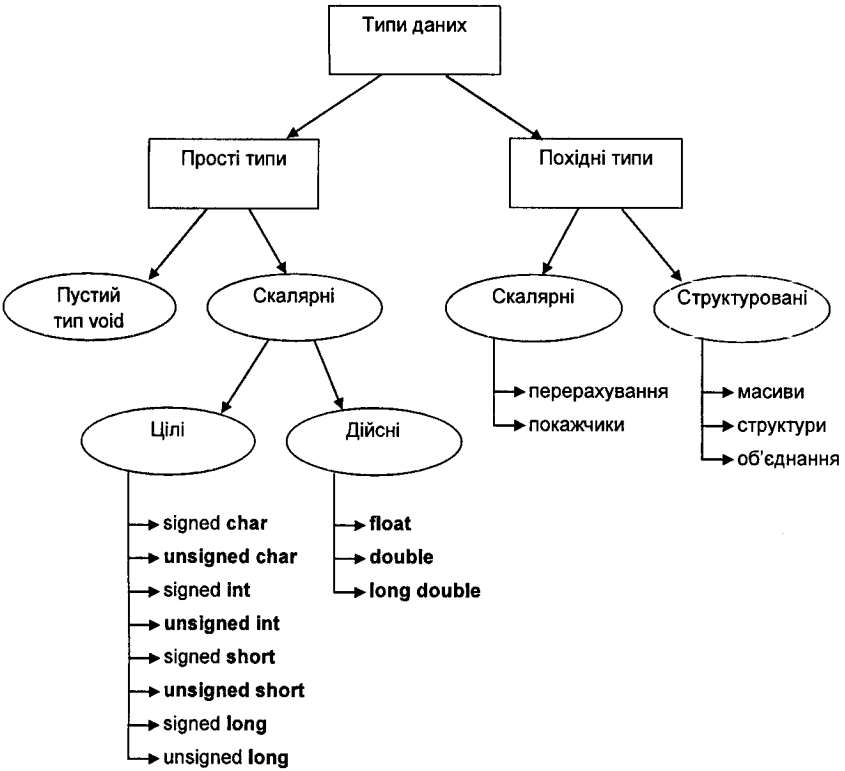


Рисунок 1.1 – Класифікація типів даних

Приклади цілих літералів наведені у табл. 1.3.

Таблиця 1.3 – Цілі літерали

Десяткові літерали	Вісімкові літерали	Шістнадцяткові літерали
16	020	0x10
63	077	0x3F
171	0253	0xAB

Створюючи ідентифікатор, потрібно пам'ятати, що він не повинен збігатися з ключовими словами й іменами функцій стандартної бібліотеки С.

Для того, щоб цілий літерал визначити типом **long**, досить наприкінці літерала поставити букву «l» чи «L».

Приклад 1.2. Цілі літерали типу long.

```
5l, 6l, 128L, 0105L, 0X2A11L.
```

Дійсний літерал – десяткове число, подане у вигляді дійсної величини з десятковою крапкою чи експонентою, має такий формат:

```
[ціла частина].[дробова частина] [E|e [+|-] експонента]
```

Дійсне число складається з цілої і дробової частин і (чи) експоненти. Дійсні літерали є додатними величинами подвоєної точності (мають тип **double**).

Приклад 1.3. Дійсні літерали типу double.

```
115.75, 1.5E-2, -0.025, 0.75, -0.85E2.
```

Для того, щоб дійсний літерал визначити типом **float**, досить наприкінці літерала поставити букву **"f"** чи **"F"**.

Приклад 1.4. Дійсні літерали типу float.

```
115.75f, 1.5E-2f, -0.025F, .075F, -0.85E2f.
```

Символьний літерал є символом, виділеним одинарними лапками (**''**). Керівна послідовність розглядається як поодинокий символ, тому вона є символьним літералом. Значенням символьного літерала є числовий код символа у таблиці кодів ASCII (додаток А).

Приклад 1.5. Символьні літерали.

```
' '    - пробіл,
'Q'    - буква Q,
'\n'   - символ нового рядка,
'\'    - зворотна дробова риска,
'\t'   - горизонтальна табуляція.
```

Символьні літерали мають тип `int` і при перетворенні типів доповнюються знаком.

Рядковий літерал – послідовність символів, яка містить малі та великі літери російського і латинського алфавітів, а також цифри, виділені подвійними лапками (`""`).

Приклад 1.6. Рядкові літерали.

```
"Бамбарабия Кергуду", "місто Вінниця", "Hello world!".
```

У кінець кожного рядкового літерала компілятором додається нульовий символ, що відповідає керівній послідовності `'\0'`.

Рядковий літерал має тип `char[]`. Це означає, що рядок розглядається як масив символів. Відзначимо важливу особливість, кількість елементів масиву дорівнює кількості символів у рядку плюс один, тому що нульовий символ (символ кінця рядка) також є елементом масиву.

1.7 Оператори

Оператор (*statement*) у мові С визначає дію, яка має бути виконана. Оператори можна поділити на такі групи:

- оператори мітки;
- оператор вираз;
- порожній оператор;
- складений оператор;
- ітераційні оператори;
- оператори переходу;
- оператори вибору.

Оператор мітки використовується разом з оператором переходу `goto` для безумовного переходу з точки програми, позначеної оператором `goto` в точку програми, позначену оператором мітки.

Оператор вираз є будь-яким виразом, в кінці якого обов'язково має бути символ крапка з комою (`;`).

Порожній оператор складається з єдиного символа (`;`). Цей оператор використовується в інших операторах, коли по синтаксису потрібно використати оператор, але жодних дій виконувати не потрібно.

Складений оператор являє собою нуль або більше операторів, об'єднаних за допомогою фігурних дужок. Відмінною рисою складеного оператора мови С від інших (наприклад, мови Паскаль) є те, що він визначає нову область дії, тобто змінні, визначені всередині складеного оператора, є локальними. Складений оператор іноді називають *блоком*.

Ітераційні оператори призначені для виконання частини коду програми декілька разів при можливій зміні значень змінних у цій частині коду.

Оператори вибору призначені для умовного переходу до виконання частини коду програми.

Оператори переходу призначені для безумовного переходу. Оператор **return** повертає значення з функції, оператор **break** припиняє виконання ітераційних операторів і оператора **switch**, оператор **continue** переходить на наступну ітерацію у ітераційних операторах.

Класифікація операторів мови С наведена на рис. 1.2.

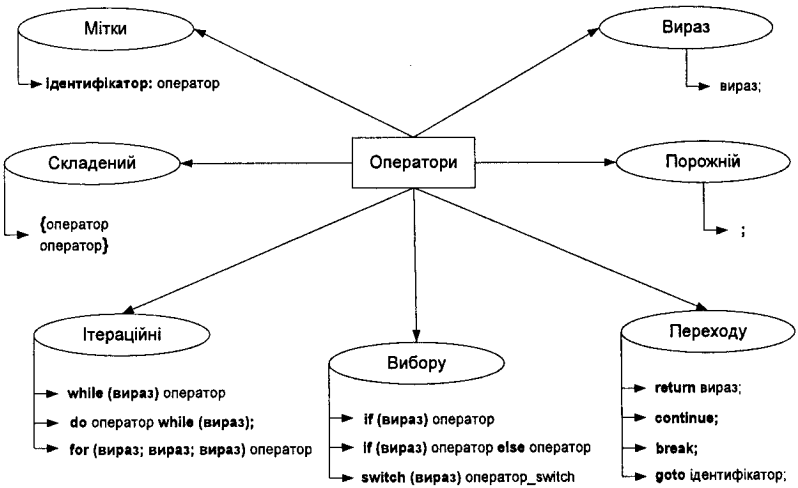


Рисунок 1.2 – Класифікація операторів

Згідно з синтаксисом операторів, зображених на рис. 1.2, на місці слова «оператор» може знаходитися будь-який оператор. Наприклад, оператор **if** може містити оператор **while**, який, в свою чергу, може містити оператор вираз. Винятком з правила є оператори **continue** і **break**, які можуть використовуватися тільки в ітераційних операторах. Це правило не розповсюджується на оператор **switch**, синтаксис якого є особливим, і визначений нижче.

1.8 Коментарі

Коментар – це набір символів, що ігноруються компілятором, і використовуються для написання пояснень до коду програми. Початок та кінець коментарю позначається відповідно лексемами (**/***) та (***/**).

Коментарі можуть бути як однорядковими, так і багаторядковими. Коментарі розглядаються компілятором як розділові символи.

Коментарі можуть використовуватися при зневадженні програми, коли потрібно зробити недоступною для компілятора проблемну (неробочу) частину коду програми.

Наявність коментарів у тексті програми робить її зрозумілішою і дозволяє легко пригадати особливості програми після звертання до неї через тривалий проміжок часу.

Приклад 1.7. Приклади використання коментарів.

```
int a = 2, b = 7;  /* ініціалізація змінних */
                   /* однорядковий коментар */
                   /* при використанні багаторядкових ко-
                      ментарів потрібно бути обережним,
                      щоб усередині послідовності, що
                      ігнорується компілятором, не пот-
                      рапили оператори програми, що та-
                      кож будуть ігноруватися */
```

1.9 Директиви препроцесора

При запуску компіляції програми на С, починає діяти спеціальна програма – препроцесор.

Препроцесор – це попередній обробник тексту програми мовою С, який знаходить директиви препроцесора та виконує їх.

Директиви препроцесора – це команди, що призначені компілятору для виконання дій, які необхідно провести до виконання програмного коду.

Кожна директива препроцесора починається з символа **#**. Тільки після виконання препроцесором всіх своїх директив, починається трансляція тексту програми з мови С на мову машинних команд.

Найчастіше використовуються такі директиви препроцесора:

- **#include** (включає файл у текст програми);
- **#define** (замінює ідентифікатор на літерал).

Заголовні файли підключаються за допомогою директиви препроцесора **#include**, що має дві форми:

- **#include "ім'я файлу"**;
- **#include <ім'я файлу>**.

Якщо ім'я файлу вказане в лапках, то його пошук здійснюється в поточному каталозі користувача. Якщо ім'я файлу задане в кутових дужках, то його пошук проводиться в стандартних директоріях операційної системи, наприклад, для компілятора DevC++ це «C:\Dev-C++\Include\».

Приклад 1.8. Синтаксис підключення заголовних файлів.

```
#include <stdio.h>  /* підключення stdio.h */
#include <stdlib.h> /* підключення stdlib.h */
#include «myfync.h» /* підключення файлу користувача
                      myfync.h */
```

Формат директиви **#define**:

#define <рядок_1> <рядок_2>

де *рядок_1* – ідентифікатор, *рядок_2* – літерал.

Директива *#define* замінить всі появи ідентифікатора *рядок_1* у вихідному файлі на літерал *рядок_2*. Заміна ідентифікатора буде здійснена тільки тоді, коли він формує лексему. Наприклад, ідентифікатор *рядок_1* не буде замінений, якщо він присутній у рядковому літералі чи є частиною довшого ідентифікатора.

Приклад 1.9. Запис директиви **#define**.

```
#define SIZE 12  /* у тексті програми всі лексеми SIZE
                    будуть замінені на цілий літерал 12. */
```

Варто відмітити, що при використанні **#define** немає необхідності у використанні оператора (**;**).

Заголовні файли мають розширення **".h"**. Файли коду містять реалізацію програми користувача і мають розширення **".c"**.

1.10 Виведення інформації на екран

Функція **printf()** призначена для виведення інформації на екран в консольному режимі.

Приклад 1.10. Виведення тексту на екран.

```
printf("Привіт світ!");
printf("Життя прекрасне!");
```

За допомогою функції **printf()** також можна виводити на екран значення змінних.

Приклад 1.11. Виведення значень змінної на екран.

```
int year = 1997;
printf("Перший Гаррі Поттер написаний у %d році", year);
```

У попередньому прикладі спочатку визначається змінна **year** типу **int**, їй присвоюється значення 1997. У функції **printf()** специфікація перетворення **%d** визначає цілий тип змінної, значення якої буде виводитися на екран. В результаті виконання програми на екрані з'явиться рядок:

```
Перший Гаррі Поттер написаний у 1997 році
```

1.11 Структура програми

При написанні програмного коду рекомендується дотримуватися певної структури програми. Програмний код повинен бути написаний таким чином, щоб його умовно можна було б розбити на такі рівні:

- директиви препроцесора;
- оголошення глобальних змінних, функцій та структур;
- визначення глобальних функцій;
- точка входу в програму (головна функція – **main()**);
- тіло головної функції;
- визначення глобальних функцій.

Таким чином, на першому рівні програми у мові С містяться команди, що мають бути виконані компілятором до початку компіляції.

На другому рівні розміщуються команди, які повідомляють процесору комп'ютера про те, які в межах програми будуть існувати дані, структури даних та функції роботи з даними.

Третій рівень містить програмні блоки, виконання яких асоціюється з оголошеними раніше функціями.

Четвертий та п'ятий рівні нерозривно пов'язані та є ім'ям і тілом програмного блока, з якого буде починатися виконання програми. Такий програмний блок в програмі може бути лише в одному екземплярі та повинен відповідати певним правилам, тому він найчастіше називається точкою входу у програму.

Шостий рівень містить програмні блоки для тих функцій, які не містились на третьому рівні.

Дотримуючись вище описаної структури, програміст не лише підвищує зрозумілість своєї програми, але і полегшує завдання компілятора – перевірку та перетворення коду. Також при дотриманні структури зменшується час виправлення програми, оскільки програміст обходить помилки взаємозв'язку програмних блоків.

Програма в С складається з однієї або більше функцій. Одна з функцій, з яких починається виконання програми, повинна мати ім'я **main**. Функція **main()** відрізняється від інших функцій тим, що її не можна викликати зсередини програми, а її параметри задаються операційною системою.

Приклад 1.12. Функція main().

```
int main ()                          /* без параметрів */
int main (int argc, char *argv []) /* з параметрами */
```

Функція **main()** може приймати два параметри (**argc** і **argv[]**) або вони можуть бути взагалі відсутні. Параметр **argc** містить кількість аргументів у командному рядку і є цілим числом, причому він завжди не менше 1, тому що першим аргументом вважається ім'я програми. А параметр

`argv` є покажчиком на масив покажчиків на рядки. У цьому масиві кожен елемент вказує на певний аргумент командного рядка.

Приклад 1.13. Програма Hello world.

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

void main()
{
printf("Hello World \n!");
system("pause");
}
```

Контрольні питання

1. Що таке алфавіт мови програмування? Наведіть категорії символів у алфавіті.
2. Дайте означення компілятора.
3. Дайте означення лексеми. Наведіть приклади лексем.
4. Що називають ключовими словами мови програмування? Наведіть приклади ключових слів.
5. Дайте означення ідентифікатора. Наведіть приклади ідентифікаторів.
6. Що таке тип даних?
7. Які існують категорії типів даних?
8. Дайте означення літерала. Які існують види літералів?
9. Опишіть формат дійсного літерала.
10. Що таке символьний літерал?
11. Поясніть різницю між значеннями `"a"` та `'a'`.
12. Поясніть, у чому різниця між `12.151` та `"12.151"`.
13. Наведіть приклад вісімкових та шістнадцяткових цілих літералів.
14. Що таке оператор? На які групи поділяють оператори мови С?
15. Що таке складений оператор?
16. Для чого призначені оператори вибору та ітераційні оператори?
17. Яке призначення операторів переходу?
18. Що таке коментар? З якою метою використовуються коментарі?
19. Що називають директивами препроцесора?
20. У чому полягає суть застосування кутових дужок та подвійних лапок при підключенні заголовного файлу?
21. Наведіть типову структуру програми, що написана мовою С.

2 ПРОСТІ ТИПИ ДАНИХ

2.1 Оголошення змінних

Типи даних – це спеціальні конструкції мови, що розглядаються компілятором як зразки для створення таких елементів програми, як змінні, літерали і функції.

Будь-який тип визначає:

- обсяг пам'яті, який виділяє компілятор для розміщення елемента;
- форму внутрішнього подання;
- множину допустимих значень;
- множину допустимих операцій, які програміст може виконувати над елементами даного типу.

Змінна – це іменована область пам'яті, де зберігається значення, яке може бути змінено в процесі виконання програми.

В мові С всі змінні, перед їх використанням повинні бути оголошені (*declarate*). При оголошенні задається ім'я змінної та її тип, а також здійснюється виділення пам'яті. Під час оголошення може бути створено декілька змінних. Їх імена в цьому випадку записують при використанні операції послідовного обчислення (,). Крім цього, може бути визначене початкове значення для змінної, таке визначення називають ініціалізацією (*initialization*). Повинно бути одне і тільки одне оголошення змінної в програмі.

Синтаксис оголошення змінної:

<модифікатор> <тип> <ім'я змінної>

де *модифікатор* – кваліфікатор **const** або один з модифікаторів класу пам'яті;
тип – тип даних;
ім'я змінної – ідентифікатор.

Мова С підтримує чотири класи пам'яті, які визначаються такими модифікаторами: **auto**, **static**, **register**, **extern**. Клас пам'яті визначає область видимості і час існування змінної.

При використанні модифікатора **const** змінна, яку ще називають константною змінною або константою, не буде змінювати своє значення протягом виконання програми. Константи повинні бути ініціалізовані при оголошенні.

Приклад 2.1. Оголошення змінних.

```
int apples;       /* оголошення змінної без модифікаторів */
auto int apples;  /* використання модифікатора auto */
```

```
int cherry, bananas;  /* оголошення декількох змінних */
const auto int apples = 7; /* ініціалізація константи */
```

2.2 Цілі типи даних

Тип даних **int** призначений для відображення цілих чисел. Розмір пам'яті, який виділяється для змінної типу **int** визначений, як правило, розміром машинного слова, для більшості сучасних комп'ютерів це 32 біта (4 байти). Приклади літералів цілих чисел: **1**, **2**, **10**, **101**. При використанні 4-х байтів для зберігання цілих чисел діапазон їх подання від мінус 2147483648 до плюс 2147483647. Таким чином тип **int** може подавати 4294967294 цілих чисел, тобто 2^{32}. Якщо є потреба оголосити змінну цілого типу, потрібно використати ключове слово **int**.

Приклад 2.2. Оголошення однієї змінної типу **int**.
```
int numberOfStudent;
```

Приклад 2.3. Оголошення декількох змінних типу **int**.
```
int numberOfStudent, i, j;
```

Приклад 2.4. Оголошення та ініціалізація змінних типу **int**.
```
int numberOfStudent = 25, i = 0, j = 2;
```

Тип даних **char** призначений для зберігання одного символа з таблиці ASCII (додаток А). Змінна типу **char** займає в пам'яті 1 байт. Приклади символьних літералів: **'t'**, **'a'**, **'4'**, **'R'**, а також спеціальні символи, такі як **'\0'**, **'\n'**. Під час компіляції символьні літерали переводяться у відповідне, згідно з таблицею ASCII, ціле значення. Наприклад, **'A'** буде переведено у ціле число 65.

Ініціалізація змінної **char** здійснюється двома способами, що наведені нижче, рекомендується застосовувати перший спосіб, другий вважається поганою програмістською практикою.

Приклад 2.5. Перший спосіб.
```
char letter1 = 'a';
```

Приклад 2.6. Другий спосіб.
```
char letter2 = 97;    /* в таблиці ANSII число 97 відповідає
                         символу 'a'*/
```

Один з видів літералів – це рядкові літерали (*string literals*).

Рядок – це набір символів, і для його подання використовуються подвійні лапки (**""**), на відміну від символів, які подаються у одинарних лап-

ках (''). Прикладом рядкового літерала є "Hello World \n", в прикладі першої програми мовою С.

Типи даних short int та long int, на відміну від базового типу int, дозволяють використовувати, відповідно, менше чи більше пам'яті для подання даних. Якщо для подання цілого числа достатній діапазон значень від мінус 32768 до плюс 32767, то доцільно використовувати тип short int. Відповідно тип long int з діапазоном значень від мінус 2147483648 до плюс 2147483647 використовується у випадку, коли потрібний більший діапазон значень для подання числа порівняно з типом short int. Ключове слово int необов'язкове при оголошенні змінних типів short int чи long int.

Як правило, змінні типу short int займають в пам'яті 2 байти, int – 4 байти, long int – 4 чи 8 байтів.

Для базових типів даних один біт використовується для подання знака числа. Якщо відомо, що змінна ніколи не буде приймати негативних значень, для її оголошення можна використати ключове слово unsigned. Воно визначає тип, у якому біт, що у базових типах використовується для знака, буде використаний для збереження даних, збільшивши таким чином удвічі діапазон значень для подання числа.

При оголошенні змінних типу unsigned int, unsigned short int та unsigned long int ключове слово int не є обов'язковим. На відміну від unsigned, ключове слово signed вказує, що тип даних знаковий, але використання signed не є обов'язковим, так як усі базові типи даних за замовчуванням знакові.

Тип даних unsigned char використовується для подання символів російського алфавіту, тому що коди російських букв в таблиці ANSII знаходяться в діапазоні значень від 127 до 255.

2.3 Дійсні типи даних

Тип даних float, скорочення від «floating point», призначений для відображення дійсних чисел. Розмір пам'яті, який виділяється для змінної типу float визначений розміром машинного слова. Як правило, це 4 байти, з яких 1 біт відводиться для знака, 8 біт для надлишкової експоненти і 23 біта для мантиси, діапазон значень змінної типу float приблизно дорівнює від 3.14E-38 до 3.14E+38.

Тип float використовується у випадках, коли не потрібно подвійної точності. Літерали типу float мають використовувати суфікси, в іншому випадку вони будуть інтерпретуватися як тип double. Приклади літералів типу float: 3.1415926f, 4.0f, 6.022e+23f. Змінні типу float визначаються, використовуючи ключове слово float.

Приклад 2.7. Оголошення змінних типу **float**.

```
float pi, chislo;
```

Приклад 2.8. Оголошення та ініціалізація змінних типу **float**.

```
float pi = 3.1415926f, chislo = 4.0f;
```

Тип даних **double** схожий на тип **float**. Відмінність полягає у тому, що **double** дозволяє зберігати дійсні числа подвійної точності. Розмір типу, як правило, два машинних слова, тобто для більшості комп'ютерів 8 байтів.

Біти пам'яті розподіляються в такий спосіб: 1 біт для знака, 11 біт для експоненти і 52 біта для мантиси. Без врахування старшого біта мантиси діапазон значень змінної типу **double** дорівнює від 1.7E-308 до 1.7E+308. Приклади літералів типу **double**: **3.1415926535897932**, **4.0**, **6.022e+23**. Використання літерала **4** замість **4.0** буде інтерпретовано як **int**. Змінні типу **double** визначаються, використовуючи ключове слово **double**.

Приклад 2.9. Оголошення змінних типу **double**.

```
double pi, chislo;
```

Приклад 2.10. Оголошення та ініціалізація змінних типу **double**.

```
double pi = 3.1415926535897932, chislo = 4.0;
```

У деяких реалізаціях мови С існує тип **long double**, розмір якого, як правило, три машинних слова, тобто для більшості комп'ютерів 12 байтів.

Загальна інформація про описані вище прості типи даних подана в табл. 2.1 (з врахуванням того, що розмір машинного слова 4 байти).

Таблиця 2.1 – Прості типи даних

Типи даних	Розмір пам'яті, байт	Діапазон значень
[signed] int	4	від - 2147483648 до 2147483647
[signed] char	1	від -128 до 127
float	4	від 3.14E-38 до 3.14E+38
double	8	від 1.7E-308 до 1.7E+308
[signed] long [int]	4	від - 2147483648 до 2147483647
[signed] short [int]	2	від - 32768 до 32767
long double	12	від 3.4E-4932 до 3.4E+4932
unsigned int	4	від 0 до 4294967295
unsigned short [int]	2	від 0 до 65535
unsigned long [int]	4	від 0 до 4294967295
unsigned char	1	від 0 до 255

2.4 Час існування та область видимості змінних

Кожна змінна, оголошена в програмі, має дві найважливіші характеристики:

- час існування;
- область видимості.

Ці характеристики взаємозалежні та істотно впливають на можливість використання змінної у програмі. Взаємозв'язок характеристик визначається способом виділення пам'яті для змінної.

Час існування або «час життя» змінної вимірюється у таких двох відносних одиницях:

- локальний «час життя» – це час існування змінної при виконанні блока, у якому вона оголошена;
- глобальний «час життя» – це час існування змінної при виконанні всієї програми.

Область видимості або область дії змінної також вимірюється в двох відносних одиницях:

- до кінця блока, у якому оголошена змінна;
- до кінця файлу, у якому оголошена змінна.

Керувати цими характеристиками змінних програміст може двома шляхами:

- зміною місця оголошення змінної у програмі;
- використанням модифікаторів **auto**, **register**, **static**, **extern**.

Автоматична (auto) змінна має локальну область дії і відома тільки всередині блока, у якому вона визначена. Для автоматичної змінної виділяється тимчасова пам'ять. Пам'ять виділяється при вході в блок, а при виході з нього пам'ять, виділена для змінної, вважається вільною, тобто змінна знищується. Якщо модифікатор класу пам'яті невизначений, то змінна вважається автоматичною.

Регістрова (register) змінна відрізняється від автоматичної лише пам'яттю, що виділяється для її збереження. Регістрова змінна зберігається в регістрі процесора, і, відповідно, доступ до цієї змінної набагато швидший, ніж до змінної, яка зберігається в оперативній пам'яті (**auto**). У випадку відсутності вільних регістрів регістрова змінна стає автоматичною.

Зовнішня (extern) змінна дозволяє здійснити її оголошення без визначення, тобто без виділення пам'яті. Використовуючи модифікатор **extern**, можна шляхом оголошення звернутися до змінної, визначеної в іншому місці. Наприклад, можна визначити всі глобальні змінні в одному файлі, а в інших файлах отримувати до них доступ через оголошення із модифікатором **extern**.

Статичній (static) змінній виділяється пам'ять після її оголошення і зберігається до кінця виконання програми. Статичні змінні при оголошенні ініціалізуються нульовими порожніми значеннями, які можуть бути цілими чи дійсними.

Контрольні питання

1. Що таке тип даних? Що визначає тип даних?
2. Що таке змінна? Опишіть синтаксис оголошення змінної.
3. Які класи пам'яті підтримує мова С?
4. Охарактеризуйте модифікатор **const**.
5. Наведіть приклади оголошення змінних типу **int**.
6. Наведіть приклади ініціалізації змінних типу **int**.
7. Порівняйте типи **short int** та **long int**.
8. Для чого використовується ключове слово **unsigned**?
9. Охарактеризуйте тип **char**.
10. Наведіть приклади оголошення змінних типу **char**.
11. Наведіть приклади ініціалізації змінних типу **char**.
12. Охарактеризуйте тип **float**. В чому різниця між типами **float** та **double**?
13. Наведіть приклади оголошення змінних типу **float**.
14. Наведіть приклади ініціалізації змінних типу **float**.
15. Наведіть приклади оголошення змінних типу **double**.
16. Наведіть приклади ініціалізації змінних типу **double**.
17. Що таке час існування змінної?
18. Що таке область видимості змінної?
19. Охарактеризуйте автоматичну змінну.
20. Охарактеризуйте регістрову змінну.
21. Охарактеризуйте зовнішню змінну.
22. Охарактеризуйте статичну змінну.

Контрольні завдання

1. Яка інформація буде виведена програмою (табл. 2.2)?

Таблиця 2.2 – Варіанти завдань

Варіант	Код програми	Варіант	Код програми
1	`float g= 1.3e2, r=2.1e2; printf("%f",g=g+1.1e2 +r);`	4	`float e=1.3e16, f=1.e15; float q=f+e; printf ("%f", m = f+q);`
2	`float a, b=2.1, c, d=12; a=c=b+d; printf ("%f %f",a,c);`	5	`float x, p, t=2.1e1; p=t+x=2.8e1; printf ("% c",x,t);`
3	`float q, a=q=3.1e2, p; printf ("%f",p=q+a);`	6	`float x, t=2.1e1; t=t+x=2.8e1; printf ("%f",x,t);`

2. Згідно з варіантом визначіть тип змінної **a**, залежно від призначення даних, що в ній мають зберігатися (табл. 2.3).

Таблиця 2.3 – Варіанти завдань

Варіант	Змінна **a** призначена для зберігання	Варіант	Змінна **a** призначена для зберігання
1	розрядності шини даних ЕОМ	16	літери з тексту
2	`99.44`	17	розміру пікселя на екрані
3	довжини слова	18	`1066`
4	довжини екватора земної кулі	19	відстані від Землі до Місяця в кілометрах
5	`'/n'`	20	`0XAA`
6	`063`	21	`'/t'`
7	кількості літер в даному завданні	22	імені диска ЕОМ
8	кількості операторів в програмі	23	часу виконання програми на ЕОМ в секундах
9	`1234`	24	`'\b'`
10	`3.14159`	25	`2.1736`
11	`'M'`	26	стипендії студента
12	`2.0 e30`	27	`'\0'`
13	`0X2B`	28	`01234`
14	`'\\'`	29	вартості ЕОМ
15	імені системного диска	30	розміру файлу в байтах

3 ОПЕРАЦІЇ ТА ВИРАЗИ

3.1 Загальні відомості

Вираз – це комбінація знаків операцій і операндів, результатом якої є певне значення.

Операнд – це те, над чим виконуються операції. У мові С як операнди можуть виступати літерали, змінні, елементи масиву, результати виклику функції, вирази. Будь-який операнд, що має константне значення, називається константним виразом. Кожен операнд має тип.

Знаки операцій визначають дії, що повинні бути виконані над операндами. Кожен операнд у виразі може бути виразом. Значення виразу залежить від розташування знаків операцій і круглих дужок у виразі, а також від пріоритету виконання операцій. Пріоритет виконання операцій поданий у табл. 3.1.

Таблиця 3.1 – Пріоритет виконання операцій

Пріоритет	Операція	Типи операцій	Порядок виконання
1	() [] .	Вираз	Зліва направо
2	->- ~ ! * & ++ -- sizeof	Унарні	Справа наліво
3	приведення типу ()	Унарний	Справа наліво
4	* / %	Мультиплікативні	Зліва направо
5	+ -	Адитивні	
6	<< >>	Зсув	
7	< > <= >=	Відношення	
8	== !=	Відношення (рівність)	
9	&	Порозрядне «І»	
10	^	Порозрядне виключне «АБО»	
11	|	Порозрядне «АБО»	
12	&&	Логічне «І»	
13	||	Логічне «АБО»	
14	?:	Умовна	
15	= *= /= %= += -= &= |= >>= <<= ^=	Просте і складене присвоєння	Справа наліво
16	,	Послідовне обчислення	Зліва направо

Приклад 3.1. Пріоритет операцій.

```
int a = 2, b = 3, c = 4, d = 7, e, f;
e = a + b / c + d & a * b - d % a;
printf("e = %d", e);/*Виведе на екран: e = 1*/
```

Порядок виконання операцій у виразі, що наведений в прикладі 3.1., зображений на рис. 3.1.

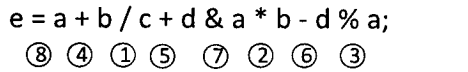


Рисунок 3.1 – Порядок виконання операцій у виразі

Спочатку будуть виконані операції з найвищим пріоритетом, тобто `b/c` (3/4 = 0), `a*b` (2*3 = 6) та `d%a` (7%2 = 1). Вираз можна переписати як: `e=a+0+d&6-1`.

Наступними виконаються адитивні операції `a+0+d` (2+0+7=9) та `6-1` (=5) і вираз можна переписати як: `e=9&5`.

Наступною виконається операція порозрядного «І» `9&5` (=1) і останньою операція присвоювання `e=1`, в результаті якої змінна `e` буде містити число `1`.

Так як операція `()` має найвищий пріоритет, для зміни пріоритету частини виразу його можна взяти у дужки, таким чином ця частина виразу виконається спочатку. У випадку вкладених дужок, в першу чергу, виконається вираз у внутрішніх дужках.

Приклад 3.2. Пріоритет операцій.

```
int a = 2, b = 3, c = 4, r;
r = a + b * c;
printf("r = %d", r); /* Виведе на екран: r = 14 */
r = (a + b) * c;
printf("r = %d", r); /* Виведе на екран: r = 24 */
```

3.2 Арифметичні операції

До арифметичних операцій належать операції додавання (`+`), віднімання (`-`), множення (`*`), ділення (`/`) і визначення залишку від ділення (`%`).

Типи операндів арифметичних операцій, крім операції (`%`), можуть відрізнятися і для них справедливі правила перетворення типів.

Операндами операції (`%`) повинні бути цілі числа. Типом результату є тип операндів після перетворення.

Арифметичні вирази мовою С записуються в рядок, так як в такому вигляді легше ввести програму в комп'ютер.

Операція множення (`*`) виконує множення операндів.

Відомості про арифметичні операції наведені в табл. 3.2.

Таблиця 3.2 – Арифметичні операції

Дія	Арифметична операція	Алгебраїчний вираз	Вираз мовою С
Додавання	+	t + 9	t + 9
Віднімання	-	r – f	r – f
Множення	*	ab	a * b
Ділення	/	x/y чи x:y	x / y
Залишок від ділення	%	z mod w	z % w

Приклад 3.3. Множення операндів.

```
int k = 5;
float f = 0.2;
double g;
g = f * k;
printf("g = %f", g);/*Виведе на екран: g = 1.0*/
```

Тип добутку змінних **f** та **k** перетвориться до типу **double**, потім результат присвоюється змінній **g**.

Операція ділення (**/**) виконує ділення першого операнда на другий. Якщо ділене та дільник цілі, то за наявності залишку він відкидається. При спробі ділення на нуль видається повідомлення про помилку під час виконання.

Приклад 3.4. Операція ділення.

```
int a = 7, b = 5, c = 0, r;
r = a / b;
printf("a/b = %d", r); /* Виведе на екран: a/b = 1 */
r = b / a;
printf("b/a = %d", r); /* Виведе на екран: b/a = 0 */
```

Операція (**%**) визначає залишок від ділення першого операнда на другий. Знак результату залежить від конкретної реалізації. Якщо другий операнд дорівнює нулю, то видається повідомлення про помилку.

Приклад 3.5. Визначення залишку від ділення операндів.

```
int a = 7, b = 5, c = 0, d;
d = a % b;
printf("a %% b = %d", d);/* Виведе на екран: a % b = 2 */
d = b % a;
printf("b %% a = %d", d);/* Виведе на екран: b % a = 5 */
d = b % c;                /* Помилка */
```

3.3 Операції приведення типів

У виразах мовою С допускається використання операндів різних типів. При обчисленні виразу тип кожного операнда може бути перетворений до

іншого типу. Перетворення типів можуть бути неявними, при виконанні операцій і викликів функцій, чи явними, при виконанні операцій приведення типів.

Неявне перетворення здійснюється автоматично для приведення операндів виразів до загального типу чи для розширення коротких величин до розміру цілих величин, використовуваних у машинних командах. Виконання перетворення залежить від специфіки операцій і від типу операнда чи операндів. Існує загальне правило: при обчисленні виразів операнди перетворяться до типу того операнда, що має найбільший розмір.

У процесі обчислення виразу операнди різних типів приводяться до єдиного за таким правилом:

- будь-яке **char**, **short** або **enum** перетворяться в **int** або **unsigned**;
- якщо після першого кроку протиріччя в розходженні типів не усунені, то відбувається додаткове перетворення типів операндів відповідно до ієрархії типів, наведеної на рис. 3.1.

int < unsigned < long < unsigned long < float < double < long double

Рисунок 3.1 – Ієрархія типів

Синтаксис операції явного перетворення типів:

(<тип>) <ідентифікатор>

де *тип* – визначає тип даних до якого здійснюється перетворення.

Приклад 3.6. Операція приведення типів.

```
double f;
int a = 2, b = 3;
f = a / b;
printf("a/b = %d", f); /* Виведе на екран: a/b = 0 */
f = (double)a / b;
printf("a/b = %f", f)  /* Виведе на екран: a/b = 0.666 */
```

В останньому виразі перед операцією ділення виконується операція перетворення типу змінної **a**. Спочатку **a=2**, а при явному перетворенні **a=2.0**. Тепер перший операнд **a** став типу **double**, а другий **b** – типу **int**. Раніше говорилося про те, що перед виконанням операції обидва операнда приводяться до одного типу. При цьому тип, який має більш низький пріоритет, перетворюється до типу з більш високим пріоритетом. Тобто **int** перетворюється у **double**. Потім виконується операція ділення, в результаті якої f=0,66666.

3.4 Операції присвоєння

Операція простого присвоєння використовується для заміни значення лівого операнда значенням правого операнда. При присвоюванні відбувається перетворення типу правого операнда до типу лівого операнда.

Приклад 3.7. Операція простого присвоєння.

```
int t;
char f = 'a';
long z = 10;
t = f + z;
printf("t = %l", t) /* Виведе на екран: t = 107 */
```

Значення змінної **f** перетворюється до типу **long**, обчислюється вираз **f+z**, результат перетворюється до типу **int** і потім присвоюється змінній **t**.

Синтаксис простих операцій присвоювання:

<операнд_1>= <операнд_1> <бінарна операція> <операнд_2>

Існує ціла група операцій присвоювання, що поєднують просте присвоєння з однією з бінарних операцій. Такі операції називаються складеними операціями присвоювання.

Синтаксис складених операцій присвоювання:

<операнд_1> <бінарна операція> = <операнд_2>

Кожна операція складеного присвоювання виконує перетворення, що здійснюються відповідною бінарною операцією.

Складені операції присвоювання подані у табл. 3.3.

Таблиця 3.3 – Складені операції присвоювання

Лексема	Пояснення складеної операції присвоювання
*=	помножити та присвоїти
/=	поділити та присвоїти
%=	залишок від ділення та присвоїти
+=	додати та присвоїти
-=	відняти та присвоїти
&=	порозрядне логічне «І» та присвоїти
|=	порозрядне логічне «АБО» та присвоїти
>>=	зсув вправо та присвоїти
<<=	зсув вліво та присвоїти
^=	порозрядне логічне «виключне АБО» та присвоїти

Приклад 3.8. Види арифметичних операцій.

```
a + = 2;    /*Еквівалентно виразу: a = a + 2;*/
b - = c;    /*Еквівалентно виразу: b = b - c;*/
c / = 3;    /*Еквівалентно виразу: c = c / 3;*/
d >> = 4;   /*Еквівалентно виразу: d = d >> 4;*/
e % = a;    /*Еквівалентно виразу: e = e % a;*/
```

3.5 Операції інкремента і декремента

Операції інкремента (**++**) і декремента (**--**) є унарними операціями присвоювання. Ці операції, відповідно, збільшують чи зменшують значення операнда на одиницю. Операнд може бути цілого або дійсного типу, чи бути покажчиком.

Операції інкремента і декремента мають дві форми запису: префіксна, коли операнд розташовується після знака операції (наприклад, **++a**, **--a**); постфіксна, коли операнд розташовується перед знаком операції (наприклад, **a++**, **a--**).

У випадку префіксної форми запису зміна операнда відбувається до його використання у виразі і результатом операції є збільшене чи зменшене значення операнда.

У випадку постфіксної форми запису операнд спочатку використовується для обчислення виразу, а потім відбувається зміна операнда.

Приклад 3.9. Операції інкремента (постфіксна і префіксна).

```
int f = 1, d, t = 1, c;
d = (f++) *5;
printf("d = %i\n", d);/* Виведе на екран d = 5*/
printf("f = %i\n", f);/* Виведе на екран f = 2*/
c = (++t)*5;
printf("c = %i\n", c);/* Виведе на екран c = 10*/
printf("t = %i\n", t);/* Виведе на екран t = 2*/
```

У прикладі 3.9. спочатку відбувається множення **f*5**, а потім збільшення змінної **f** на одиницю. У результаті вийде **d=5**, **f=2**, а при використанні префіксної операції змінна **t** збільшується на одиницю і відбувається множення **t*5**. У результаті отримаємо **c=10** та **t=2**.

У випадку, якщо операції збільшення чи зменшення використовуються не у виразі, а як самостійні оператори, префіксна і постфіксна форми запису стають еквівалентними.

3.6 Операції порівняння

Мова С підтримує дві групи операцій порівняння: операції рівності та операції відношення. Всі операції порівняння наведені в табл. 3.4.

Вирази, в яких присутні операції порівняння, як правило, використовуються в операторах керування як умова. Для операторів вибору (**if**, **if-else**) – це умова, за якої вибирається напрямок подальшого ходу програми. Для операторів циклів (**while**, **do-while**, **for**) – це умова, за виконання якої буде виконана чергова ітерація циклу (у випадку, якщо значення виразу не дорівнює нулю), а при невиконанні якої буде здійснено вихід з циклу (у випадку, якщо значення виразу дорівнює нулю).

Таблиця 3.4 – Операції порівняння

Алгебраїчний запис	Лексеми операцій	Приклад виразу у С	Значення виразу
Операції рівності			
=	=	x == y	x дорівнює y
≠	!=	x != y	x не дорівнює y
Операції відношення			
>	>	x > y	x більше y
<	<	x < y	x менше y
≥	>=	x>=y	x більше чи дорівнює y
≤	<=	x<=y	x менше чи дорівнює y

Приклад 3.10. Операції порівняння.

```
int a = 1, b = 4;
printf("%i \n", a <= b);   /* Виведе на екран: 1*/
printf("%i \n", a == b);   /* Виведе на екран: 0*/
printf("%i \n", a != b);   /* Виведе на екран: 1*/
printf("%i \n", a > b);    /* Виведе на екран: 0*/
```

3.7 Операції зсуву

Порозрядні логічні операції зсуву вліво (<<) та вправо (>>) застосовуються до даних цілих типів. Результат операцій зсуву теж буде цілим і відповідає двійковому поданню числа, зміщеному вліво чи вправо на задану кількість бітів. При цьому висунуті за межі подання типу біти втрачаються.

Операції зсуву здійснюють зсув лівого операнда вліво (<<) чи вправо (>>) на кількість бітів, що задається правим операндом.

Синтаксис операцій зсуву:

```
<Вираз_1> << <Вираз_2>
<Вираз_1> >> <Вираз_2>
```

де *Вираз_1* – змінна цілого типу, *Вираз_2* – змінна чи літерал цілого типу, який визначає кількість зсунутих бітів.

Приклад 3.11. Операції зсуву.

```
int a = 9;              /* змінна a в двійковому поданні:
                           a = 1001 */
int b = a >> 2;         /* b в двійковому поданні:
                           b = 10 */
printf("b = %d", b);   /* Виведе на екран: 10*/
int c = a << 2;         /* c в двійковому поданні:
                           c = 100100 */
prir.tf("c = %d", c);   /* Виведе на екран: 36 */
```

3.8 Порозрядні операції

До порозрядних операцій належать:

- операція порозрядного логічного "І" (&);
- операція порозрядного логічного "АБО" (|);
- операція порозрядного "виключного АБО" (^).

Таблиці істинності порозрядних операцій зображенні у табл. 3.5.

Таблиця 3.5 – Таблиці істинності порозрядних операцій

a	b	a & b	a | b	a ^ b
0	0	0	0	0
0	1	0	1	1
1	0	0	1	1
1	1	1	1	0

Операнди порозрядних операцій можуть бути будь-якого цілого типу. За необхідності над операндами виконуються перетворення за замовчуванням, тип результату – це тип операндів після перетворення.

Операція порозрядного логічного І (&) порівнює кожен біт першого операнда з відповідним бітом другого операнда. Якщо обидва порівнюваних біти дорівнюють одиниці, то відповідний біт результату дорівнюватиме 1, у протилежному випадку – 0.

Операція порозрядного логічного АБО (|) порівнює кожен біт першого операнда з відповідним бітом другого операнда. Якщо кожен з порівнюваних бітів дорівнює 1, то відповідний біт результату дорівнюватиме 1, у протилежному випадку – 0.

Операція порозрядного виключного АБО (^) порівнює кожен біт першого операнда з відповідними бітами другого операнда. Якщо один з порівнюваних бітів дорівнює 0, а другий біт дорівнює 1, то відповідний біт результату дорівнює 1, у протилежному випадку, тобто коли обидва біти дорівнюють 1 чи 0, біт результату – 0.

Приклад 3.12. Операція порозрядного порівняння.

```
int a = 5, b = 11, c;
c = a | b;
```

```
printf("c = %d", c); /*Виведе на екран: c = 15*/
c = a & b;
printf("c = %d", c); /*Виведе на екран: c = 1*/
c = a ^ b;
printf("c = %d", c); /*Виведе на екран: c = 14*/
```

При обчисленні порозрядних операцій кожен операнд подається у двійковій формі. Так значення змінної **a** у двійковій формі буде 0101_2, значення змінної **b** – 1011_2. Використовуючи таблицю 3.5 для відповідної порозрядної операції, отримується результат у двійковій формі, який потім переводиться у десяткову форму. Приклад обчислення порозрядних операцій наведений на рис. 3.2.

$$\begin{array}{r} 0101_2 \\ \mid\ 1011_2 \\ \hline 1111_2 = 15_{10} \end{array} \qquad \begin{array}{r} 0101_2 \\ \&\ 1011_2 \\ \hline 0001_2 = 1_{10} \end{array} \qquad \begin{array}{r} 0101_2 \\ \wedge\ 1011_2 \\ \hline 1110_2 = 14_{10} \end{array}$$

Рисунок 3.2 – Приклад обчислення порозрядних операцій

3.9 Логічні операції

До логічних операцій належать операція логічного «І» (**&&**) та операція логічного «АБО» (**||**). В кожній операції можуть брати участь операнди різних типів. Результатом логічної операції є 0 чи 1, тип результату **int**.

Результатом застосування операції логічного «І» (**&&**) до операндів, що мають нульові значення, є 0. Якщо один з операндів дорівнює 0, то результатом операції є 0. Якщо обидва операнда мають ненульове значення, результат операції дорівнює 1.

Результатом застосування операції логічного «АБО» (**||**) до операндів, що мають нульові значення, є 0. У випадку, якщо один з операндів має ненульове значення, результат операції дорівнює 1.

Таблиці істинності логічних операцій зображені у табл. 3.6.

Таблиця 3.6 – Таблиці істинності логічних операцій «І» та «АБО»

A	B	A && B	A || B
0	0	0	0
0	не «0»	0	1
не «0»	0	0	1
не «0»	не «0»	1	1

Приклад 3.13. Логічні операції.

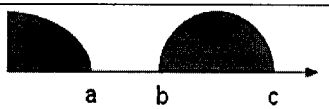
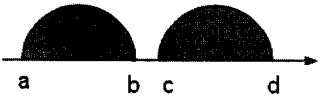
```
int A = 10, B = 0, C = 4, D;
D = A && B;
printf("D = %d", D); /*Виведе на екран: D = 0*/
D = A || B;
```

```
printf("D = %d", D); /*Виведе на екран: D = 1*/
D = A && C;
printf("D = %d", D); /*Виведе на екран: D = 1*/
D = A || C;
printf("D = %d", D); /*Виведе на екран: D = 1*/
```

За допомогою логічних операцій «І» та «АБО», і операторів порівняння можна конструювати вирази для перевірки входження в діапазон (діапазони) значень змінних. Такі вирази, як правило, використовують в операторах керування для конструювання складних умов. Приклад таких виразів поданий у табл. 3.7.

У першому пункті табл. 3.7 поданий вираз, який повертає 1 у випадку, якщо значення змінної **x** входить у діапазон (a,b). У другому пункті поданий вираз, який повертає 1, якщо значення змінної **x** входить у один з діапазонів $(-\infty,a)$ чи $(b,+\infty)$. У третьому пункті поданий вираз, який повертає 1, якщо значення змінної **x** входить у один з діапазонів $(-\infty,a]$ або (b,c). У четвертому пункті поданий вираз, який повертає 1, якщо значення змінної **x** входить у один з діапазонів (a,b) або $[c,d]$.

Таблиця 3.7 – Приклади виразів подання діапазону значень

№ пункту	Графічне подання	Математичне подання	Подання в мові С
1		$x \in (a,b)$	`x>a&&x<b`
2		$x \in (-\infty,a) \cup (b,+\infty)$	`x<a||x>b`
3		$x \in (-\infty,a] \cup (b,c)$	`x<=a||x>b &&x<c`
4		$x \in (a,b) \cup [c,d]$	`x>a&&x<b || x>=c&&x<=d`

Задача 3.1. Програма, яка зчитує з клавіатури символ і виводить на екран 1, якщо цей символ є малою буквою англійського алфавіту, і 0 – в протилежному випадку.

Для вирішення задачі потрібно скористатися таблицею кодів ASCII, в якій малі букви англійського алфавіту кодуються числами, а саме в діапазоні чисел від 97 до 122 закодовані літери від `'a'` до `'z'`. Для задання ви-

разу, який буде приймати значення 1, у випадку попадання значення введеного символа в потрібний діапазон, доцільно використати вираз з пункту 1 табл. 3.7.

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

int main(){
  char x;
  scanf("%c", &x);
  printf("%i \n", x>= 97 && x<= 122);
  system("PAUSE"); return 0;
}
```

Результат роботи програми при введенні символа «r»:

```
r
1
```

Результат роботи програми при введенні символа «R»:

```
R
0
```

Задача 3.2. Програма, яка зчитує з клавіатури символ і виводить на екран 1, якщо цей символ є малою чи великою буквою англійського алфавіту, і 0 – в протилежному випадку.

Для задання виразу, який буде приймати значення 1, у випадку попадання значення введеного символа в потрібні діапазони, доцільно використати вираз з пункту 4 табл. 3.7.

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

int main(){
  char x;
  scanf( "%c", &x );
  printf( "%i \n", x>=65 && x<=90 || x>=97 && x<=122 );
  system("PAUSE"); return 0;
}
```

Результат роботи програми при введенні символа «a»:

```
a
1
```

Результат роботи програми при введенні символа «A»:

```
A
1
```

3.10 Операція послідовного обчислення

Операція (,) – операція послідовного обчислення для двох чи більше виразів використовується там, де за синтаксисом допустимий тільки один вираз. Обчислення виконуються зліва направо. При виконанні операції послідовного обчислення перетворення типів не відбувається.

Приклад 3.14. Операція послідовного обчислення.

```
int a, b, c;                    /* при оголошенні змінних */
for(a = 0, b = 0; i < 10; i++) /* при ініціалізації змінних
                                  в операторі for */
```

3.11 Операція умови (? :)

У мові С присутня одна тернарна операція – умовна операція.
Синтаксис тернарної опрації:

<Операнд_1> ? <Операнд_2> : <Операнд_3>

Операнд_1 повинний бути цілого або дійсного типу, чи бути покажчиком. Він оцінюється з погляду його рівності 0. Якщо *Операнд_1* не дорівнює 0, то обчислюється *Операнд_2* і його значення є результатом операції. Якщо *Операнд_1* дорівнює 0, то обчислюється *Операнд_3* і його значення є результатом операції. Потрібно зазначити, що обчислюється або *Операнд_2*, або *Операнд_3*, але не обидва. Тип результату залежить від типів *Операнда_2* та *Операнда_3*.

Приклад 3.15. Операція умови (? :).

```
int a = 2, b =4 , max;
max = (b <= a) ? a : b;
```

Результат виразу **b<=a** дорівнює нулю, тому змінній **max** присвоюється значення змінної **b**.

Контрольні питання

1. Що таке вираз?
2. Що таке операнд?
3. Що може виступати як операнд?
4. Від чого залежить значення виразу?
5. Що таке пріоритет виконання операцій?
6. Наведіть приклади арифметичних операцій.
7. Для чого призначена операція (%)?
8. Що таке перетворення типів і в яких випадках воно відбувається?
9. Наведіть правило неявного перетворення типів.
10. Наведіть синтаксис операції явного перетворення типів.

11. Охарактеризуйте операції простого присвоювання.
12. Охарактеризуйте операції складеного присвоювання.
13. Наведіть синтаксис операції складеного присвоювання.
14. Поясніть операцію складеного присвоювання (`%=`).
15. Поясніть операцію складеного присвоювання (`<<=`).
16. Для чого призначені операції інкремента і декремента?
17. В чому різниця між постфіксною і префіксною операціями інкремента та декремента?
18. На які групи поділяються операції порівняння? Охарактеризуйте кожну з цих груп.
19. Для чого використовуються вирази, в яких застосовуються операції порівняння?
20. Поясніть операцію порівняння (`!=`).
21. Що таке порозрядні логічні операції зсуву?
22. Наведіть синтаксис операцій зсуву.
23. Що таке порозрядні логічні операції? Які існують порозрядні логічні операції?
24. Охарактеризуйте порозрядну логічну операцію (`|`).
25. Охарактеризуйте порозрядну логічну операцію (`^`).
26. Охарактеризуйте порозрядну логічну операцію (`&`).
27. Наведіть таблиці істинності порозрядних логічних операцій.
28. Що таке логічні операції?
29. Охарактеризуйте логічну операцію (`&&`).
30. Охарактеризуйте логічну операцію (`||`).
31. Наведіть таблиці істинності порозрядних логічних операцій «І» та «АБО».
32. Охарактеризуйте операцію послідовного обчислення.
33. Наведіть синтаксис операції (`?:`).
34. Від чого залежить тип результату тернарної операції (`?:`)?

Контрольні завдання

1. Обчисліть значення виразів:
   а) `1.5<7`;
   б) `'3'<'a'`;
   в) `'Z'<'a'`;
   г) `false<true`;
   д) `false<-1`.
   Які неявні перетворення типів виконуються під час їх обчислення?
2. Нехай `n` – тризначне число. Напишіть вирази, які присвоюють змінним `a`, `b`, `c`, відповідно, кількість сотень, десятків і одиниць числа `n`.
3. Обчисліть значення виразу згідно з варіантом (табл. 3.8).

Таблиця 3.8 – Варіанти завдань

Варіанг	Вираз
1	`g = a^b + c%d;`
2	`g = a|b + c/d;`
3	`g = a%=b + c|d;`
4	`g = a<b?c:d>>1;`
5	`g = a>=b ? b&c : d<<(3 + c&d);`
6	`g = a>=b ? b^c : d<<1 + c/d;`
7	`g = a^=b%c>>1 - c&&a<<d;`
8	`g = a|=++b+ --c<<a-2;`
9	`g = --a^=--b- --d<<c-c%d-2;`
10	`g = c++>b ? d<<(3 + c&d) : d<<1&c ;`
11	`g = d<<1>a ? b%a-c<<1 : d&&a^c;`
12	`g = d<<1>-a ? b/a - c << 4:d&&a^b%d- --a&2;`

4. Перепишіть програму без помилок. Яка інформація буде виведена програмою? Вкажіть положення курсору після виводу.

```
# include <studio. h>
main()
<
float a=b=01.5e2;
 float z;x;
x=0x2e02;
printf ("\ n%c=% f ",x,x);
return 0;
>
```

5. Напишіть умову того, що відрізки [a, b] та [c, d] осі OX:
   а) не мають спільних точок;
   б) мають хоча б одну спільну точку.
6. Напишіть умову того, що точки з координатами (x1, y1) та (x2, y2):
   в) збігаються;
   г) не збігаються.

4 ВВЕДЕННЯ ТА ВИВЕДЕННЯ ДАНИХ

Введення та виведення даних в мові С виконується з використанням *потоків* – послідовності символів з порядковою організацією. Кожен рядок складається з нуля чи більшої кількості символів і закінчується символом нового рядка.

При запуску програми до неї автоматично підключаються три потоки. Стандартний *потік введення*, як правило, під'єднується до клавіатури, а стандартний *потік виведення* – до пристрою виведення інформації на екран монітора (екран). Операційні системи нерідко дозволяють переадресовувати ці потоки на інші пристрої. Третій потік – стандартний *потік помилок* – також під'єднується до екрана.

В даному розділі розглядаються функції **printf()** та **scanf()**, які виводять дані в стандартний потік виведення, і вводять дані зі стандартного потоку введення.

4.1 Функція виведення printf

Функція **printf()** призначена для форматованого виведення у консольне вікно повідомлень і значень змінних. Для використання функції **printf()** потрібно підключити заголовний файл **<stdio.h>**, який містить визначення макросів, констант, оголошення функцій для операцій стандартного введення та виведення.

Приклад 4.1. Підключення заголовного файлу **<stdio.h>**.

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

Функція **printf()** може складатися з одного чи декількох параметрів. Першим параметром є рядок форматування, іншими параметрами можуть бути літерали, змінні чи вирази, значення яких необхідно вивести на екран.

Синтаксис виклику функції **printf()** виглядає таким чином:

```
printf("<рядок форматування>", <список аргументів> );
```

Текст, який необхідно вивести на екран, потрібно записувати у *рядок форматування*, який виділяється подвійними лапками. У *списку аргументів* як аргументи можуть виступати літерали, змінні чи вирази.

Приклад 4.2. Виведення тексту на екран.

```
printf("Hello World!");
printf("Привіт Світ!");
```

Рядок форматування може містити такі елементи як:

- літеральні символи;
- специфікації перетворення;
- керівні послідовності.

Специфікації перетворення (табл. 4.1) є символьними комбінаціями, кожна з яких починається з символа відсоток (`'%'`) і закінчується специфікатором перетворення. За допомогою специфікацій перетворення можна здійснювати форматування значень змінних.

Специфікатор перетворення записують в те місце у рядку форматування, в яке потрібно вивести значення змінної.

Для виведення текстів кирилицею можна використати функцію **`setlocale()`**, для цього потрібно підключити заголовний файл **`<locale.h>`**.

Приклад 4.3. Встановлення підтримки української мови у консолі.

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

int main()
{
  setlocale (LC_ALL, "Ukrainian");
  printf("Вітаю, Світе!\n ");
  system ("pause"); return 0;
}
```

Результат роботи програми:

```
Вітаю, Світе!
```

Таблиця 4.1 – Специфікації перетворення цілих чисел у функції **`printf()`**

Специфікація перетворення	Призначена для виведення	Тип даних
`% i`	цілих чисел зі знаком	**`int, short, long`**
`% d`	цілих чисел зі знаком	**`int`**
`% hd`	цілих чисел зі знаком	**`short`**
`% ld`	цілих чисел зі знаком	**`long`**
`% u`	беззнакових цілих чисел	**`unsigned int`**
`%o`	цілих чисел у вісімковій системі числення	**`int, short, long`**
`%x` чи **`%X`**	цілих чисел у шістнадцятковій системі числення	**`int, short, long`**

Приклад 4.4. Виведення на екран літералів.

```
printf(" %d ", 2013); /* виведе на екран: 2013 */
```

Приклад 4.5. Виведення на екран різної кількості значень змінних.

```
int a = 2000, b = 10, c = 3;
printf(" %d ", a);          /* виведе на екран: 2000 */
printf(" %d %d ", a,b);     /* виведе на екран: 2000 10 */
printf(" %d,%d,%d", a,b,c);/* виведе на екран: 2000,10,3 */
```

Приклад 4.6. Виведення на екран значення виразу.

```
int a = 2000, b = 10, c = 3;
printf(" %d ", a + b + c);/* виведе на екран: 2013 */
```

У попередньому прикладі спочатку буде обчислено значення виразу, а потім це значення буде виведено на місце специфікації перетворення (**%d**) у рядку форматування.

Приклад 4.7. Використання специфікаторів перетворення цілих чисел.

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

int main(void)
{
  printf("%d %d %d %i\n", 123, +123, -123, 123);
  printf("%hd\n", 32000);
  printf("%ld\n", 1000000000);
  printf("%u\n", 123);
  printf("%o %o\n", 8, 16);
  printf("%x %X\n", 15, 26);

  system("pause"); return 0;
}
```

Результат роботи програми:

```
123 123 -123 123
32000
1000000000
123
10 20
f 1A
```

Для виведення на екран значення однієї змінної, її потрібно записати другим параметром функції **printf()**. При цьому у рядку форматування специфікацію перетворення потрібно поставити у позицію, в яку необхідно вивести значення змінної (рис. 4.1).

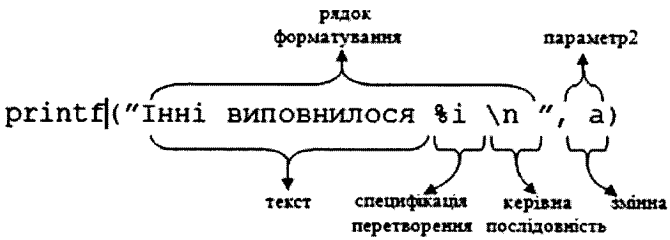


Рисунок 4.1 – Виведення на екран значення однієї змінної

Для виведення на екран значень декількох змінних, їх потрібно записувати у параметри, починаючи з другого. При цьому у рядку форматування специфікації перетворення потрібно поставити у позиції, в які необхідно вивести значення змінних (рис. 4.2).

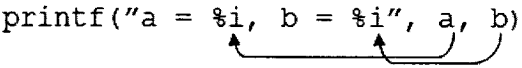


Рисунок 4.2 – Виведення на екран значень двох змінних

На рис. 4.2 стрілками відмічено, що значення змінної **a** буде виведено на місце першої специфікації перетворення (**%i**) у рядку форматування, а значення змінної **b**, відповідно, на місце другої специфікації перетворення.

Таблиця 4.2 – Специфікації перетворення дійсних чисел у функції **printf()**

Специфікація перетворення	Призначена для виведення	Тип даних
%f	дійсних чисел, в форматі числа з плаваючою точкою	**float, double**
%n.mf	дійсних чисел в форматі з фіксованою точкою, де *n* – кількість цифр цілої частини, *m* – дробової частини	**float, double**
%e чи **%E**	дійсних чисел в експоненційному поданні	**float, double**
%g чи **%G**	дійсних чисел у форматі того специфікатора (%f чи %e), який дозволить подати число у більш короткому вигляді	**float, double**

Приклад 4.8. Використання специфікацій перетворення дійсних чисел.

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

int main(void)
{
  printf("%f\n", 123.456);
  printf("%8.2f %.1f\n", 123.456, 123.456);
  printf("%.2e %.3E\n", 123.45, 123.45);
  printf("%g %G\n", 12345.6789, 1234567.89);
  system("pause"); return 0;
}
```

Результат роботи програми:

```
123.456000
  123.46 123.5
1.23e+002 1.235E+002
12345.7 123457E+006
```

Таблиця 4.3 – Специфікації перетворення для символів та рядків у функції **printf()**

Специфікація перетворення	Призначена для виведення	Тип даних
%c	символів	**char**
%s	рядків	**char []**
%%	символа %	

Приклад 4.9. Використання специфікаторів перетворення для виведення символів та рядків.

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

int main(void)
{
  setlocale( LC_ALL, "Ukrainian" );
  char ch = 'У';
  char str [] = "Це рядок!";
  char *strPtr = "Це також рядок!";

  printf( "Виведення символа \'%c\' \n", ch );
  printf( "%s \n", str );
  printf( "%s \n", strPtr );

  system("pause"); return 0;
}
```

Результат роботи програми:

```
Виведення символа 'У'
Це рядок!
Це також рядок!
```

У рядку форматування для виведення деяких літералів використовуються *керівні послідовності*, які є спеціальними символами комбінації, кожна з яких починається з символа (\).

Таблиця 4.4 – Керівні послідовності

Керівна послідовність	Опис
`\t`	Перемішує курсор на наступну позицію горизонтальної табуляції
`\b`	Переміщує курсор на одну позицію назад в поточному рядку
`\n`	Перехід на новий рядок
`\a`	Подає звуковий сигнал або візуальне попередження
`\r`	Переміщує курсор на початок поточного рядка
`\"`	Виводить символ подвійних лапок
`\'`	Виводить символ одинарних лапок
`\0`	Виводить нульовий символ
`\\`	Виводить зворотній слеш
`\ddd`	Виводить вісімкове значення символа ASCII
`\xddd`	Виводить шістнадцяткове значення символа ASCII

Керівні послідовності вигляду **\ddd** та **\xddd** дозволяють подати будь-який символ з таблиці кодів ASCII (додаток А) у вигляді вісімкових чи шістнадцяткових чисел відповідно.

4.2 Функція введення даних scanf

Функція **scanf()** призначена для введення даних з клавіатури.

Для використання функції **scanf()** потрібно підключити заголовний файл **<stdio.h>**.

Синтаксис виклику функції **scanf**:

```
scanf ("<рядок форматування>", &змінна1, &змінна2, …);
```

Першим аргументом функції **scanf()** є *рядок форматування*, який записується в подвійних лапках і містить специфікатори перетворення, що визначають типи змінних, в які будуть записані введені дані.

Другий параметр, та кожен наступний, є адресою змінної, в яку зчитуються дані (*&змінна1, &змінна2, ...*).

Специфікатори перетворення функції **scanf()** наведені у табл. 4.5.

Таблиця 4.5 – Специфікації перетворення для функції **scanf()**

Специфікатор перетворення	Призначенний для введення	Тип даних
%i	цілих чисел зі знаком	**int, short, long**
%d	цілих чисел зі знаком	**int, short, long**
%f	дійсних чисел, в форматі числа з плаваючою точкою	**float, double**
%c	символів	**char**
%s	рядків	**char []**
%u	беззнакових десяткових цілих	**unsigned int**
%x	шістнадцяткових цілих	**int**
%o	вісімкових цілих	**int**

Приклад 4.10. Зчитування цілих чисел за допомогою функції **scanf()**.

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

int main(void)
{
  setlocale( LC_ALL, "Ukrainian" );
  int a, b, c, d, e, f, g;
  printf ("Введіть шість цілих чисел: ");
  scanf( "%d %i %i %u %o %x", &a, &b, &c, &d, &e, &f );
  printf( "%d %d %d %d %d %d\n", a, b, c, d, e, f );
  system("pause"); return 0;
}
```

Результат роботи програми:

```
Введіть шість цілих чисел: -20 -20 0x20 20 020 20
-20 -20 32 20 16 32
```

Приклад 4.11. Зчитування дійсних чисел за допомогою функції **scanf()**.

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

```
int main(void)
{
  setlocale( LC_ALL, "Ukrainian" );
  float a, b, c;
  printf ("Введіть три дійсних числа: ");
  scanf ("%f %f %f", &a, &b, &c );
  printf( "%.5e %.5f %.5g \n", a, b, c );
  system("pause"); return 0;
}
```

Результат роботи програми:

```
Введіть три дійсних числа: 123,456 123,456 123,456
1,23456e+002 123,45600 123,46
```

Приклад 4.12. Зчитування символів та рядків за допомогою функції **scanf()**.

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

int main(void)
{
  setlocale( LC_ALL, "Ukrainian" );
  char a, b[10];
  printf ("Введіть рядок: ");
  scanf("%c%s", &a, b );
  printf("Символ: \'%c\'\nРядок: \"%s\" \n ", a, b );
  system("pause"); return 0;
}
```

Результат роботи програми:

```
Введіть рядок: Ucraine
Символ: 'U'
Рядок: "craine"
```

Задача 4.1. Програма, яка зчитує два цілих числа та виводить на екран їх добуток та суму.

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

int main()
{
  setlocale( LC_ALL, "Ukrainian" );
  int a, b;
  printf( " Введіть два цілих числа: " );
  scanf( " %i %i ", &a, &b );
  printf( " Добуток: %i \n", a * b );
```

```
  printf( " Сума: %i \n ", a + b );
  system("pause"); return 0;
}
```

Результат роботи програми:

```
Введіть два цілих числа: 12 2
Добуток: 24
Сума: 14
```

Задача 4.2. Програма, яка запитує у користувача його ім'я та вітається з користувачем, згадуючи його ім'я.

```
#include <stdlib.h>
#include <stdio.h>
#include <locale.h>
int main()
{
  setlocale (LC_ALL, "Ukrainian");
  char name[10];
  printf ("Введіть ваше ім'я ");
  scanf ("%s", name);
  printf ("Привіт %s!\n", name);
  system("pause"); return 0;
}
```

Результат роботи програми:

```
Введіть ваше ім'я: Анжеліка
Привіт Анжеліка!
```

Контрольні питання

1. Для чого використовується функція **printf**?
2. Опишіть синтаксис функції **printf**.
3. Що таке рядок форматування?
4. Що таке специфікатори перетворення?
5. Наведіть приклади специфікаторів перетворення.
6. Що повертає функція **printf**?
7. Що таке керівна послідовність?
8. Наведіть приклади керівних послідовностей.
9. Для чого використовується функція **scanf**?
10. Опишіть синтаксис функції **scanf**.
11. Для чого потрібно ставити знак амперсенд перед кожним ідентифікатором змінної в функції **scanf**?

5 ОПЕРАТОРИ КЕРУВАННЯ

Оператори керування обчислювальним процесом використовуються для організації: розгалуження, циклічного повторення операторів програми залежно від виконання заданих умов, а також передавання керування в певне місце програми. Умову в операторах керування задають за допомогою умовного виразу, що може бути будь-яким виразом мови С. У випадку операторів циклу умовні вирази задають умову виходу з циклу.

Умовні вирази, як правило, задаються при використанні операцій порівняння:

- операцій відношень (<, >, <=, >=);
- операцій рівності (==, !=).

Для конструювання більш складніших умов в умовних виразах можуть застосовуватися логічні операції:

- логічне «АБО» (||);
- логічне «І» (&&).

Результат обчислення умовного виразу можна інтерпретувати як «істина» (true), у випадку, якщо він не дорівнює нулю або «хибність» (false), у випадку, якщо він дорівнює нулю.

5.1 Оператори вибору if та if-else

Для реалізації розгалуження, у випадку необхідності вибору одного з двох можливих шляхів виконання програми, застосовуються оператори вибору **`if`** та **`if-else`**.

Синтаксис оператора вибору **`if`**:

```
if ( <умовний вираз> ) <true-оператор>;
```

Схема алгоритму оператора вибору **`if`** зображена на рис. 5.1.

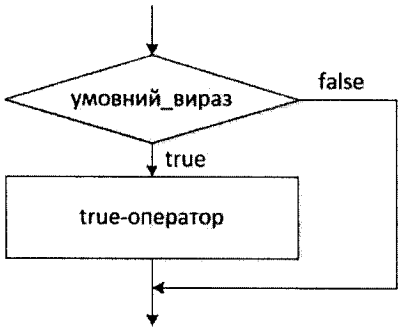


Рисунок 5.1 – Схема алгоритму оператора вибору **`if`**

Виконання оператора **if** починається з обчислення *умовного виразу*. У випадку, коли значення *умовного виразу* відмінне від нуля – виконується – *true-оператор* (будь-який оператор С, див. рис. 1.2), а потім наступний за **if** оператор. У іншому випадку, тобто коли значення виразу дорівнює нулю – *true-оператор* пропускається і керування передається наступному за **if** оператору.

Задача 5.1. Програма обчислення вартості телефонної розмови з врахуванням 20% знижки, яка надається по суботах та неділях.

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

int main()
{
  setlocale (LC_ALL, "Ukrainian");
  int time;  /* тривалість розмови у хвилинах */
  int day;   /* день тиждня*/
  float sum; /* вартість розмови (за хвилину) */

  printf("Введіть тривалість розмови:\n");
  scanf("%i", &time);
  printf("Введіть день тижня (1-7):\n");
  scanf("%i", &day);
  sum = 2.4 * time;

  if (day == 6 || day == 7)
  {
    printf("Знижка наявна у розмірі 20%\n");
    sum = sum * 0.8;
  }
  printf("Вартість розмови становить: %3.2f\n", sum);
  system("PAUSE"); return 0;
}
```

Результат роботи програми:

```
Введіть тривалість розмови: 10
Введіть день тижня (1-7): 2
Вартість розмови становить: 24,00
```

Розглянутий вище оператор **if** можна розглядати як окремий випадок оператора **if-else**.

Синтаксис оператора **if-else**:

```
if ( <умовний вираз> ) <true-оператор>;
else <false-оператор>;
```

Виконання оператора **if-else** починається з обчислення *умовного виразу*, якщо його значення відмінне від нуля, то виконується *true-оператор*, а потім наступний за **if-else** оператор. У випадку, якщо значення *умовного виразу* дорівнює нулю, то виконується *false-оператор*, а потім наступний за **if-else** оператор. Як *true-оператор*, так і *false-оператор* можуть бути будь-якими операторами С (див. рис. 1.2).

Схема алгоритму оператора вибору **if-else** зображена на рис. 5.2.

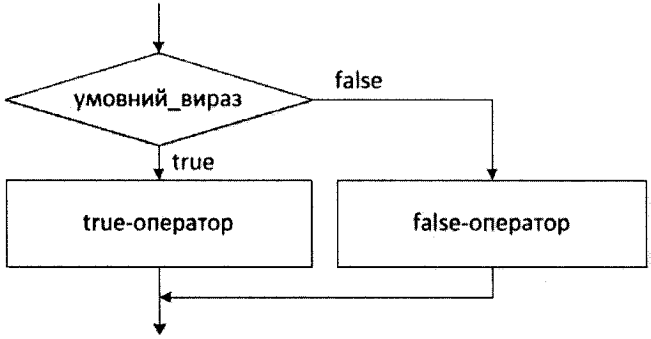


Рисунок 5.2 – Схема алгоритму оператора вибору **if-else**

Приклад 5.1. Істинність та хибність умовного виразу.

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

int main()
{
  int i, j;
  scanf ("%d %d", i, j);
  if (i < j)
    printf("%d < %d - true", i, j);
  else
    printf("%d < %d - false", i, j);
  system("PAUSE"); return 0;
}
```

Результат роботи програми при введенні i = 19, j = 80:

```
19 < 80 - true
```

Результат роботи програми при введенні i = 5, j = 4:

```
5 < 4 - false
```

Задача 5.2. Програма обчислення площі кільця. Радіуси отвору та кільця задати з клавіатури, у випадку, якщо введений радіус отвору більший за радіус кільця, видати повідомлення про помилку.

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

int main()
{
  setlocale (LC_ALL, "Ukrainian");
  float S;     /* S - площа кільця */
  float R, r; /* R - радіус кільця, r - радіус отвору */

  printf("Введіть R, r: \n");
  scanf("%f %f",&R,&r);    /* Введення значень радіусів */

  if (r > R)
    printf("Помилка!");    /* Повідомлення про помилку */
  else
  {
    S = 3.14*( R*R - r*r ); /* Обчислення площі кільця */
    printf("Площа кільця S = %2.2f\n", S);
  }
  system("PAUSE"); return 0;
}
```

Результат роботи програми:

```
Введіть R, r: 7 6
Площа кільця S = 40,82
```

Оператори **if** та **if-else** можуть бути вкладеними. У цьому випадку *true-оператор* чи *false-оператор* містять оператори **if** або **if-else**.

Задача 5.3. Програма розв'язку рівняння ax = b.

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

int main()
{
  setlocale (LC_ALL, "Ukrainian");
  float a, b, x;
  printf("Введіть a, b: \n");
  scanf("%f %f", &a, &b);

  if(a != 0)
    printf("x = %2.2f\n", b/a);
  else
```

```
    if(b == 0)
      printf("x - будь-яке число\n");
    else
      printf("Розв'язку немає\n");
  system("PAUSE"); return 0;}
```

Результат роботи програми:

```
Введіть a, b: 4 5
x = 1,25
```

Для усунення неоднозначності компілятор С інтерпретує вкладені *if* таким чином, що гілка *else* належить до найближчого попереднього *if*.

5.2 Оператор вибору switch

Для реалізації множинного розгалуження, у випадку необхідності вибору більше, ніж двох можливих шляхів виконання програми, застосовується оператор **switch**.

Синтаксис оператора **switch**:

```
switch ( <Вираз> )
{
case <конст. вираз 1>: <набір операторів 1>; break;
case <конст. вираз 2>: <набір операторів 2>; break;
.....
case <конст. вираз n>: <набір операторів n>; break;
default : <набір операторів за замовчуванням>;
}
```

Вираз, що іде за ключовим словом *switch* у круглих дужках, може бути будь-яким виразом, допустимим в мові С, значення якого повинне бути цілим.

Значення цього виразу є ключовим для вибору з декількох варіантів. Тіло оператора **switch** складається з наборів операторів, кожен з яких позначений ключовим словом *case* з наступним константним виразом. Потрібно зазначити, що використання цілого константного виразу є істотним недоліком, який властивий оператору **switch**.

Так як константний вираз обчислюється під час трансляції, він не може містити змінні чи виклики функцій. Зазвичай, як константний вираз, використовують цілі або символьні літерали.

Усі константні вирази в операторі **switch** повинні бути унікальні. Крім наборів операторів, позначених ключовим словом *case*, може бути, але не обов'язково, один набір, позначений ключовим словом *default*.

Набір операторів може бути порожнім або містити один чи більше операторів. Причому в операторі **switch** не потрібно набір операторів об'єднувати за допомогою фігурних дужок, як у складеному операторі.

Схема алгоритму оператора **switch** зображена на рис. 5.3.

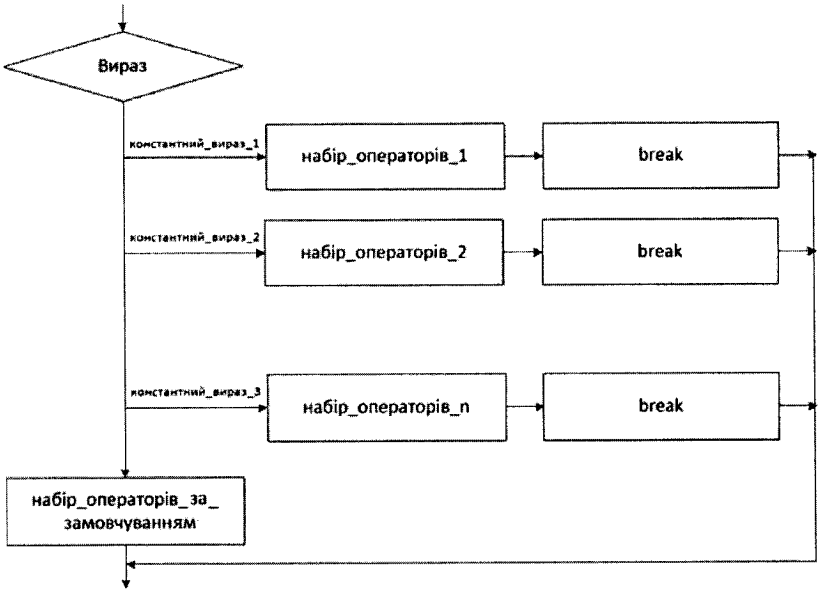


Рисунок 5.3 – Схема алгоритму оператора вибору **switch**

Схема виконання оператора **switch** така:

- обчислюється вираз в круглих дужках;
- обчислене значення виразу послідовно порівнюються з константними виразами, що ідуть за ключовими словами *case*;
- якщо один з константних виразів збігається зі значенням виразу, то управління передається набору операторів, що позначений відповідним ключовим словом *case*;
- якщо жоден з константних виразів не дорівнює виразу, то управління передається набору операторів, позначеному ключовим словом *default*, а у випадку його відсутності управління передається наступному після **switch** оператору.

Використання оператора *break* дозволяє в необхідний момент перервати послідовність виконуваних операторів у тілі оператора **switch**, шляхом передавання управління оператору, що іде за **switch**.

Варто знати, що за відсутності операторів *break* в тілі оператора **switch** і збігу значення виразу з яким-небудь константним виразом, вико-

нується наскрізний прохід через усе тіло оператора, що залишилося у **switch**, охоплюючи й оператор гілки *default*.

Задача 5.4. Програма, яка виводить назву дня тижня за умови введення його порядкового номера (1 – понеділок, 2 – вівторок, … і т. д.). У випадку виходу за діапазон (від 1 до 7) виводиться повідомлення про помилку.

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

int main()
{
  setlocale (LC_ALL, "Ukrainian");
  int day; /* день тижня */
  printf( "Введіть день тижня (1-7): " );
  scanf( "%i", &day );
  switch( day )
  {
    case 1: printf("Понеділок \n"); break;
    case 2: printf("Вівторок \n"); break;
    case 3: printf("Середа \n"); break;
    case 4: printf("Четвер \n"); break;
    case 5: printf("П'ятниця \n"); break;
    case 6: printf("Субота \n"); break;
    case 7: printf("Неділя \n"); break;
    default : printf ("Помилка! \n");
  }
  system("PAUSE"); return 0;
}
```

Результат роботи програми:

```
Введіть день тижня (1-7): 3
Середа
```

5.3 Оператор циклу for

Оператор **for** – найбільш загальний спосіб організації циклів в мові С. Синтаксис оператора **for**:

```
for ( <Вираз_1> ; <Вираз_2> ; <Вираз_3> )
<тіло циклу>
```

Вираз_1, як правило, використовується для ініціалізації змінних, що керують циклом. *Вираз_2* є умовним виразом, тобто визначає умову, за якої *тіло циклу* буде виконуватися. *Вираз_3* визначає зміну змінних, що керують циклом після кожного виконання *тіла циклу*.

Схема виконання оператора **for** така:

- обчислюється *Вираз_1*;
- обчислюється *Вираз_2*;
- у випадку, якщо *Вираз_2* має значення відмінне від нуля (тобто умова істинна, true), виконується *тіло циклу*, потім обчислюється *Вираз_3* і здійснюється перехід на наступну ітерацію циклу, тобто виконується перехід до обчислення *Виразу_2*. Якщо значення *Виразу_2* дорівнює нулю (тобто умова хибна, false), то управління передається оператору, що іде за оператором **for**.

Оператор **for** є циклом з передумовою, тобто перевірка умови завжди виконується на початку циклу. Це означає, що *тіло циклу* може жодного разу не виконатися, якщо умова виконання відразу буде хибною. Тіло циклу може бути будь-яким оператором С (див. рис. 1.2).

Схема алгоритму оператора циклу **for** зображена на рис. 5.4.

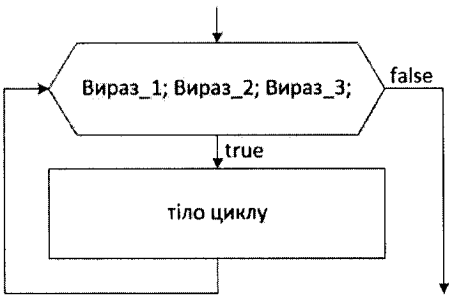


Рисунок 5.4 – Схема алгоритму оператора циклу **for**

Задача 5.5. Програма, яка виводить на екран таблицю квадратів перших п'яти непарних доданих чисел.

```
#include <stdlib.h>
#include <stdio.h>
#include <locale.h>
int main()
{
  setlocale (LC_ALL, "Ukrainian");
  int x = 1;
  int y, i;
  for( i = 0; i <= 5; i++ )
  {
    y = x * x; /* обчислення квадрата числа */
    printf("%3i\t%4i\n", x, y);
    x += 2;
  }
  system("PAUSE"); return 0;}
```

Результат роботи програми:

```
1       1
3       9
5       25
7       49
9       81
```

5.4 Оператор циклу while

Оператор **while** є циклом із передумовою.
Синтаксис оператора **while**:

```
while ( <умовний вираз> )
<тіло циклу>
```

Як *тіло циклу* може використовуватися будь-який оператор С (див. рис. 1.2). Схема виконання однієї ітерації циклу оператора **while**:

- обчислюється *умовний вираз*;
- якщо значення *умовного виразу* дорівнює нулю (тобто умова хибна, false), то виконання оператора **while** закінчується і виконується наступний за **while** оператор. Якщо ж значення *умовного виразу* відмінне від нуля (тобто умова істинна, true), то виконується *тіло циклу* і відбувається перехід на наступну ітерацію циклу.

Схема алгоритму оператора циклу **while** зображена на рис. 5.5.

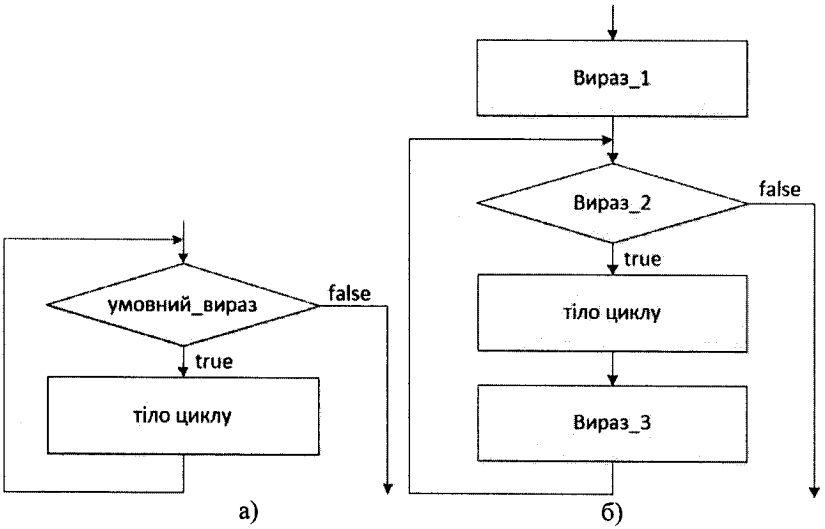


Рисунок 5.5 – Схема алгоритму оператора циклу **while**

На рис. 5.5, а зображена загальна схема алгоритму оператора циклу **while**, схему алгоритму на рис. 5.5, б можна інтерпретувати як розгорнуту схему алгоритму оператора **for**.

Задача 5.6. Програма, яка виводить на екран таблицю значень функції $y = 2x^2 - 5x - 8$ у діапазоні від -2 до 2. Крок зміни аргументу – 0,5.

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

int main()
{
  setlocale (LC_ALL, "Ukrainian");
  float xmin=-2, xmax=2; /* xmin, xmax-границі діапазону */
  float dx = 0.5; /* dx - крок зміни аргументу */
  float y, x;
  printf ("  x    |  y \n" );
  x = xmin;

  while ( x <= xmax )
  {
    y = 2*x*x - 5*x - 8;
    printf ("%2.2f | %.2f \n", x, y );
    x = x + dx;
  }

  system("pause"); return 0;
}
```

Результат роботи програми:

```
   x    |    y
-2,00   |   10,00
-1,50   |    4,00
-1,00   |   -1,00
-0,50   |   -5,00
 0,00   |   -8,00
 0,50   |  -10,00
 1,00   |  -11,00
 1,50   |  -11,00
 2,00   |  -10,00
```

5.5 Оператор циклу do-while

Оператор **do-while** є циклом із післяумовою і використовується в тих випадках, коли необхідно виконати тіло циклу хоча б один раз.

Синтаксис оператора **do-while**:

```
do
<тіло циклу>
while ( <умовний вираз> ) ;
```

Схема виконання однієї ітерації циклу оператора **do-while**:

- виконується *тіло циклу*, яке може бути будь-яким оператором С (див. рис. 1.2);
- обчислюється *умовний вираз*;
- якщо значення *умовного виразу* дорівнює нулю (тобто умова хибна, false), то виконання оператора **do-while** закінчується і виконується наступний за **do-while** оператор. Якщо значення *умовного виразу* відмінне від нуля (умова істинна, true), то відбувається перехід на наступну ітерацію циклу.

Схема алгоритму оператора циклу **do-while** зображена на рис. 5.6.

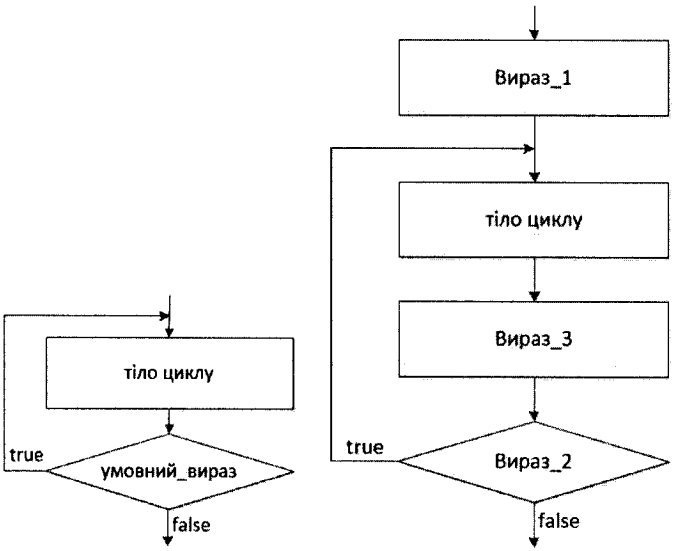


Рисунок 5.6 – Схема алгоритму оператора циклу **do-while**

На рис. 5.6, а зображена загальна схема алгоритму оператора циклу **do-while**, на рис. 5.5, б зображена розгорнута схема алгоритму оператора циклу **do-while**, в якій *Вираз_1*, *Вираз_2* та *Вираз_3* відповідає виразам, визначеним для оператора **for**.

Задача 5.7. Програма обчислення суми чисел введених з клавіатури. У випадку введення числа «0» припинити обчислення.

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

int main()
{
  setlocale (LC_ALL, "Ukrainian");
  int sum = 0, ch;
  printf("Введіть числа (0 - кінець введення): ");
  do{
    scanf("%i", &ch);
    sum = sum + ch;
  }while(ch != 0);

  printf("Сума = %i \n", sum);

  system("PAUSE"); return 0;
}
```

Результат роботи програми:

```
Введіть числа (0 - кінець введення): 3 2 1 0
Сума = 6
```

5.6 Оператор break

Безумовний оператор **break** використовується для швидкого виходу з циклів **for**, **while** чи оператора розгалуження **switch**. При вкладенні циклів оператор **break** дозволяє вийти з тіла самої внутрішньої інструкції циклу. Після виконання інструкції **break** керування передається першій інструкції, що розташована за останньою інструкцією циклу, що перервався.

Приклад 5.2. Використання операторів **break** та **for** замість оператора циклу **do-while** у задачі 5.7.

```
  for( i = 0; i < 10; i++ )
  {
    scanf("%i", &ch);
    sum = sum + ch;
    if( ch == 0 )
      break;
}
```

5.7 Оператор continue

Безумовний оператор **continue** призначений для переривання ітерації циклу. В тілі циклу виконуються лише інструкції, що знаходяться до опе-

ратора **continue**, ті інструкції, що знаходяться після нього – ігноруються і відбувається перехід до умовного виразу. На відміну від оператора **break** управління передається не на наступну після циклу інструкцію, а на наступний крок ітерації.

Задача 5.8. Програма обчислення суми непарних елементів в послідовності чисел від 0 до 9.

```
#include <stdlib.h>
#include <stdio.h>
int main()
{
  int sum = 0, i;

  for(i = 0; i < 10; i++)
  {
    if (i%2 == 0)
      continue;
    sum = sum + i;
  }
  printf("sum = %i \n", sum);
  system("PAUSE"); return 0;
}
```

Результат роботи програми:

```
sum = 25
```

Контрольні питання

1. Для чого призначені оператори керування?
2. Які операції, зазвичай, застосовуються при конструюванні умовних виразів?
3. Для чого призначені оператори вибору **if** та **if-else**?
4. Наведіть синтаксис оператора вибору **if**.
5. Наведіть синтаксис оператора вибору **if-else**.
6. Зобразіть схему алгоритму оператора вибору **if**.
7. Зобразіть схему алгоритму оператора вибору **if-else**.
8. Для чого призначений оператор вибору **switch**?
9. Наведіть синтаксис оператора вибору **switch**.
10. У якому випадку тіло оператора циклу можна не брати в фігурні дужки?
11. Наведіть синтаксис оператора циклу **for**.
12. Зобразіть схему алгоритму оператора циклу **for**.
13. Наведіть синтаксис оператора циклу **while**.
14. Зобразіть схему алгоритму оператора циклу **while**.
15. Наведіть синтаксис оператора циклу **do-while**.
16. Зобразіть схему алгоритму оператора циклу **do-while**.

17. Як реалізувати неперервний цикл за допомогою операторів циклу з передумовою?
18. Порівняйте оператори циклу з перед- та післяумовами.
19. Порівняйте оператори циклу **for** та **while**.
20. Для чого призначений безумовний оператор **break**?
21. Для чого призначений безумовний оператор **continue**?

Контрольні завдання

1. Напишіть умовний вираз, який перевіряє чи:
   а) значення змінної **a** типу **int** є **100**;
   б) значення змінної **b** типу **int** без остачі ділиться на **3**;
   в) мають однакову парність значення змінних **a** та **b**.
2. Напишіть умовний вираз, який перевіряє чи:
   а) змінна **c** символьного типу є великою латинською буквою (від **'A'** до **'Z'**);
   б) змінна **c** символьного типу є шістнадцятковою цифрою;
   в) змінна **c** символьного типу є двадцятковою цифрою.
3. Зобразити фрагмент схеми програми, який відповідає такому фрагменту програми:
   а) `if ( c<3 ) if (c == 2)  a++;  else b++; a += 1;`
   б) `if ( c<3 ) if (c == 2)  a++;  b++; a += 1;`
   в) `if ( c<3 ) if (c == 2)  a++;  else b++; if ( c<2 ) c++; else  a += 1; { c++; b++;}`
4. За допомогою операторів вибору і присвоювання записати фрагмент програми, який обчислює значення змінної x за таким правилом:

   а) $x = \begin{cases} x + 1, \text{при } a > 0 \text{ та } b = 0; \\ a + b, \text{при } a \le 0 \text{ та } b = 0; \\ a - b, \text{ в інших випадках.} \end{cases}$

   б) $x = \begin{cases} 1, \text{при } i \in [20,100); \\ -1, \text{при } i \ge 100; \\ 0, \text{в інших випадках.} \end{cases}$

   в) $x = \begin{cases} 255, \text{при } i = 1 \text{ або } i = 2 \text{ або } i = 5; \\ 100, \text{при } i \in (6,12); \\ 0, \text{в інших випадках.} \end{cases}$
5. Напишіть програму, яка визначає чи належить введене з клавіатури число відрізку [2, 4] або [-1, 1].
6. Напишіть програму, яка визначає чи лежить введене з клавіатури число за межами відрізка [0, 1].
7. Напишіть програму, яка визначає чи лежить введене з клавіатури число за межами відрізків [3, 6] та [-1, 1].

8. Напишіть програму, яка по введеному номеру дня тижня (1 – понеділок, ..., 7 – неділя) виводить на екран розклад занять цього дня. Використати оператор вибору `switch`.
9. Напишіть програму, яка по введеному номеру елемента в періодичній таблиці хімічних елементів виводить на екран його назву. Використати оператор вибору `switch`.
10. Напишіть програму, яка по введеному році виводить на екран місце проведення олімпійських ігор. Використати оператор вибору `switch`.
11. Напишіть програму, яка виводитиме на екран числа від 1 до 10, використовуючи:
    а) оператор `for`;
    б) оператор `while`;
    в) оператор `do-while`.
12. Напишіть програму, яка виводить на екран таблицю римських чисел, діапазон у десятковому еквіваленті від 1 до 100.
13. Напишіть програму, яка обчислює та виводить на екран суму перших N членів ряду: 1 + 1/2 + 1/3 + ¼ + ... Кількість N членів ряду задати з клавіатури.
14. Напишіть програму, яка обчислює число «Пі» із заданою користувачем точністю. Для обчислення числа «Пі» потрібно скористатися частковою сумою ряду: 4 - 4/3 + 4/5 - 4/7 + 4/9 - 4/11 + ... Точність обчислення визначається кількістю членів ряду.
15. Напишіть програму, яка знаходить суму ряду з точністю $\varepsilon=10^{-4}$, загальний член якого:
    а) $a_n = \frac{n!}{n^n}$; б) $a_n = \frac{3^n n!}{(3n)!}$;
    в) $a_n = \lg(n!)\, e^{-n\sqrt{n}}$; г) $a_n = \frac{1}{2^n} + \frac{1}{3^n}$.
16. Напишіть програму, яка обчислює та виводить на екран суму парних цілих чисел від 2 до 10.
17. Напишіть програму, яка обчислює суму введених з клавіатури чисел до тих пір, поки не буде введено число 0.
18. Напишіть програму, яка виводить таблицю значень функції $y= -2{,}4x^2+5x-3$ у діапазоні від -2 до 2, із кроком 0,5.
19. Напишіть програму, яка виводить таблицю значень функції y=sin(x) у діапазоні від 1 до 5, із кроком 0,5.

6 МАСИВИ

6.1 Загальні поняття

Масив – набір однотипних змінних, які об'єднані під одним ім'ям.

Елементи масиву розміщуються у пам'яті послідовно, один за одним. Елемент масиву може мати декілька індексів. Кількість індексів елемента визначає кількість вимірів масиву. У мові С найбільш використовуваними є одновимірні та двовимірні масиви.

При оголошенні масиву задається:

- тип даних елементів масиву;
- ім'я масиву;
- кількість вимірів;
- кількість елементів у кожному вимірі.

Синтаксис оголошення n-вимірного масиву:

```
<тип> <ім'я масиву>[<вимір_1>][<вимір_2>]…[<вимір_n>];
```

де *тип* – тип даних елемента масиву;

ім'я масиву – ідентифікатор;

вимір_1, вимір_2, …вимір_n – визначають кількість елементів у відповідному вимірі (*вимір_1* – кількість елементів у першому вимірі, *вимір_2* – кількість елементів у другому вимірі і т. д.).

Загальна кількість елементів n-вимірного масиву визначається як добуток елементів всіх його вимірів: *вимір_1*вимір_2*…*вимір_n*.

Приклад 6.1. Оголошення масивів.

```
int array1[100];      /* оголошення одновимірного масиву */
int array2[10][10];   /* оголошення двовимірного масиву */
int array3[10][5][2];/* оголошення трьохвимірного масиву */
```

Масиви належать до структурованих типів даних (див. рис. 1.1).

До будь-якого елемента масиву можна звернутись, вказавши ім'я масиву та значення індексу (індексів) масиву.

Найчастіше масиви використовуються для вирішення таких задач:

- пошук деякого значення у масиві;
- сортування елементів масиву в порядку зростання або спадання;
- підрахунок кількості елементів у масиві, що задовольняють задану умову.

Масиви є ефективними при використанні у нескладних задачах, поки не з'являється потреба ефективного додавання та видалення елементів масиву.

6.2 Одновимірні масиви

Кожен елемент одновимірного масиву має один індекс. Нумерація елементів одновимірного масиву у мові С починається з нуля, тобто значенням індексу першого елемента є нуль, другого – один, третього – два і т. д.

Одновимірний масив можна уявити як таблицю, в якій присутній лише один рядок. При зверненні до елемента масиву необхідно вказати ім'я масиву, а також, в квадратних дужках, значення індексу, яке вказує на місцезнаходження необхідного значення в масиві.

Синтаксис оголошення одновимірного масиву:

```
<тип> <ім'я масиву>[n];
```

де n – кількість елементів масиву.

Розглянемо одновимірний масив цілих чисел з п'яти елементів:

```
int Array [5];
```

Для звернення до елементів масиву з метою їх заповнення послідовністю чисел 7, -5, 9, 1, 0 необхідно записати такий код:

```
Array[0] = 7; Array[1] = -5; Array[2] = 9; Array[3] = 1;
Array[4] = 0;
```

На рис. 6.1 подано графічне зображення одновимірного масиву **Array**.

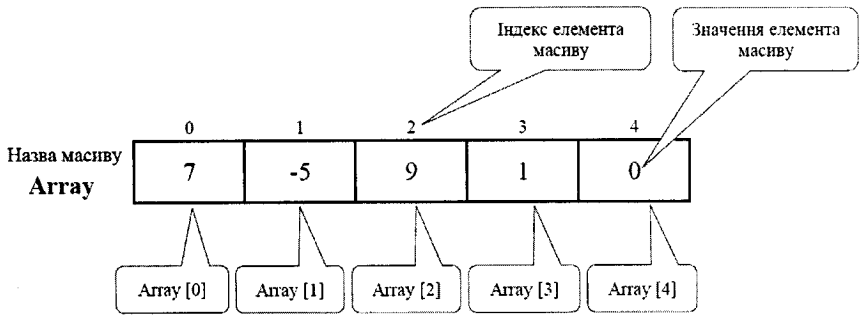


Рисунок 6.1 – Зображення одновимірного масиву

Розглянемо рис. 6.1, для отримання значення, яке зберігається у масиві **Array** під індексом 2, потрібно звернутися до масиву таким чином: **Array[2]**. Використавши функцію **printf("%i", Array[2])**, на екрані побачимо значення цього елемента: 9.

Існує декілька можливостей ініціалізації масиву. Розглянемо їх на прикладах.

Можна присвоїти деякі значення елементам масиву при його оголошенні.

Приклад 6.1. Ініціалізація одновимірного масиву при оголошенні.

```
int array[8] = {2, 2, 1, 1, 1, 9, 8, 0};
```

Ефективним способом заповнення масиву є заповнення його у циклі.

Приклад 6.2. Ініціалізація одновимірного масиву в циклі.

```
for (i = 0; i < 8; i++)
{
  array[i] = i;
  printf("%i ", array[i]);
}
```

Результат роботи програми:

```
0 1 2 3 4 5 6 7
```

Можна присвоїти значення окремим елементам масиву за допомогою оператора присвоювання.

Приклад 6.3. Ініціалізація окремих комірок одновимірного масиву.

```
int array[8] = {5, 5, 5, 5, 5, 5, 5, 5};
array[5] = array[6] = array[7] = 7;
for (i = 0; i < 10; i++)
  printf("%i ", array[i]);
```

Результат роботи програми:

```
5 5 5 5 5 7 7 7
```

Також можна заповнювати масив числами, введеними з клавіатури. Для цього використовується функція **scanf()**.

Приклад 6.4. Ініціалізація одновимірного масиву з клавіатури.

```
for (int i = 0; i < 10; i++)
{
  scanf("%i", &array[i]);
  printf("%i ", array[i]);
}
```

Задача 6.1. Програма знаходження максимального елемента одновимірного масиву.

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

int main()
{
  int max = - 999999999;
  int array[5] = {1, 3, 5, 3, 0};
  for(i = 0; i < 5; i++)
  {
    if(array[i] > max)
    {
      max = array[i];
    }
  }
  printf("max = %i\n", max);
  system("PAUSE"); return 0;
}
```

Результат роботи програми:

```
max = 5
```

6.3 Багатовимірні масиви

Багатовимірним називається масив, у якому є два або більше вимірів, причому доступ до кожного елемента такого масиву здійснюється за допомогою певної комбінації двох або більше індексів.

Найпростішою формою багатовимірного масиву є двовимірний масив. Розташування будь-якого елемента в двовимірному масиві позначається двома індексами **n** і **m**. Такий масив можна подати у вигляді таблиці, в якій **n** рядків та **m** стовпців.

Синтаксис оголошення двовимірного масиву:

```
<тип> <ім'я масиву>[n][m];
```

де n – кількість рядків;
m – кількість стовпців.

Приклад 6.5. Ініціалізація двовимірного масиву при оголошенні.

```
int a[3][4] = { { 5, 6, 7, 10 },
                { 4, 9, 2, 1  },
                { 7, 8, 5, 12 } };

int a[2][5] = { { 5, 6, 7, 7, 3 },
                { 4, 9, 7, 1, 12 } };
```

Усі властивості одновимірних масивів зберігаються і для багатовимірних масивів. Тривимірний масив можна уявити як об'ємну фігуру – паралелепіпед. Більше трьох вимірів можна вважати математичною абстракцією.

Приклад 6.6. Ініціалізація тривимірного масиву при оголошенні.

```
int a[2][2][3] = { { { 1, 2, 3 }, { 4, 5, 6 } },
                   { { 7, 8, 9 }, { 10, 11, 12 } } };

int a[3][2][4] = { { { 1, 2, 3, 8 }, { 4, 5, 6, 10 } },
                 { { 1, 2, 3, 4 }, { 4, 5, 6, 5 } },
                   { { 7, 8, 9, 11 }, { 10, 11, 12, 3 } } };
```

Індексація двовимірного масиву зображена на рис. 6.2.

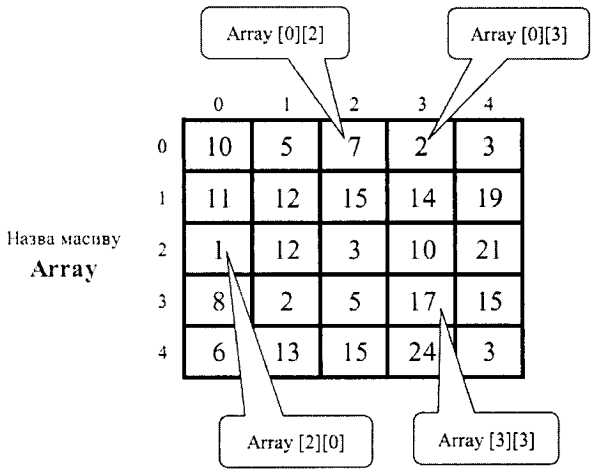


Рисунок 6.2 – Зображення двовимірного масиву

Для зчитування, виведення на екран і оброблення двовимірних масивів, зазвичай, використовують вкладені цикли (зовнішній та внутрішній). Зовнішній цикл відповідає за індексацію рядків, а внутрішній цикл – за індексацію стовпчиків. Таким чином внутрішній цикл призначений для оброблення всіх елементів поточного рядка.

Приклад 6.7. Заповнення двовимірного масиву **A[n][m]** з клавіатури.

```
for( i = 0; i < n ; ++i )
  for( j = 0 ; j < m ; ++j )
    scanf("%i ", A[i][j]);
```

Приклад 6.8. Виведення на екран двовимірного масиву `A[n][m]` у вигляді таблиці, елементи масиву в рядку розділені пробілом.

```
for( i = 0; i < n; ++i ){ /* Виводимо на екран i-й рядок */
  for( j = 0; j < m; ++j )
    printf("%i ", A[i][j]);
  printf("\n"); } /* перехід на новий рядок */
```

Задача 6.2. Програма, в якій елементам двовимірного масиву, що знаходяться вище головної діагоналі присвоюється значення `'1'`, елементам, що знаходяться нижче головної діагоналі – `'-1'`, а тим, що на головній діагоналі – `'0'`.

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

int main()
{
  int array[4][4];
  int n = 5, i, j;
  for (i = 0; i < n; i++)
  {
    for (j = 0; j < n; j++)
    {
      if (i == j) array[i][j] = 0;
      if (i > j) array[i][j] = -1;
      if (i < j) array[i][j] = 1;
      printf("%3i ", array[i][j]);
    }
    printf("\n");
  }
  system("pause"); return 0;
}
```

Результат роботи програми:

```
  0  1  1  1  1
 -1  0  1  1  1
 -1 -1  0  1  1
 -1 -1 -1  0  1
 -1 -1 -1 -1  0
```

Контрольні питання

1. Дайте означення терміну «масив».
2. Що задається при оголошенні масиву?
3. Що визначає розмірність масиву?
4. Наведіть синтаксис n-вимірного масиву.
5. Для вирішення яких задач найчастіше використовуються масиви?
6. Що таке одновимірний масив?
7. Наведіть синтаксис одновимірного масиву.

8. Які існують способи ініціалізації масивів?
9. Що таке двовимірний масив?
10. Наведіть синтаксис двовимірного масиву.
11. Особливості використання циклів при ініціалізації багатовимірних масивів.

Контрольні завдання

1. Напишіть програмний код, який виконує такі дії над одновимірним масивом:
   а) відображення на екрані значення третього елемента масиву символів `str`;
   б) обчислення суми `50` елементів масиву дійсних чисел `mDouble`;
   в) запис у перші шість елементів масиву дійсних чисел `mDouble` значення `999`.
2. Напишіть програмний код, який виконує такі дії над одновимірним масивом:
   а) ініціалізація елементів масиву цілих чисел `mas[10]` нулями;
   б) інкрементація кожного з `20` елементів масиву `best`;
   в) заповнення з клавіатури дійсними числами масиву `total`, що містить `9` елементів;
   г) виведення на екран у вигляді стовбця `12` елементів масиву цілих чисел `year`.
3. Напишіть програму, яка обчислює середнє арифметичне значення масиву цілих чисел `mas[10]`, що вводяться з клавіатури.
4. Напишіть програму, яка обчислює значення медіани масиву цілих чисел `mas[10]`, що вводяться з клавіатури.
5. Напишіть програму, яка обчислює значення моди масиву цілих чисел `mas[10]`, що вводяться з клавіатури.
6. Напишіть програму, яка знаходить та виводить на екран мінімальний за модулем елемент масиву цілих чисел `mas[10]`.
7. Напишіть програму, яка знаходить мінімальний елемент масиву цілих чисел `mas[10]` та виводить на екран значення його індексу.
8. Напишіть програму, яка обчислює добуток значень елементів масиву цілих чисел `mas[10]` з парними значеннями індексу.
9. Напишіть програму, яка обчислює суму значень елементів масиву цілих чисел `mas[10]` з непарними значеннями індексу.
10. Напишіть програму, яка обчислює суму значень елементів масиву цілих чисел `mas[10]`, розташованих між першим і останнім від'ємним елементом.
11. Напишіть програму, яка обчислює суму значень елементів масиву цілих чисел `mas[10]`, розташованих між першим і другим нульовими елементами.

12. Напишіть програму, яка знаходить в масиві цілих чисел `mas[20]` найбільшу серію одиниць, що розміщуються підряд. Вивести на екран значення індексів цієї серії.
13. Напишіть програму, яка в масиві цілих чисел `masInt` з `12` елементів підраховує кількість пар елементів, що відповідають умові `masInt[i]< masInt[i+1]`.
14. Напишіть програму, яка в масиві дійсних чисел `masFloat` з `10` елементів знаходить максимальний елемент та відхилення кожного елемента від максимального.
15. Напишіть програму, яка в масиві `masTemp` з `24` елементів, що містить температури повітря протягом дня, знаходить мінімальну, максимальну та середню температуру повітря.
16. Напишіть програму, яка обчислює та виводить на екран середнє арифметичне значення елементів двовимірного масиву цілих чисел `mas2D` розміром 5×5.
17. Напишіть програму, яка обчислює та виводить на екран суму додатніх елементів двовимірного масиву цілих чисел `mas2D` розміром 5×5. Значення масиву ініціалізувати в програмі.
18. Напишіть програму, яка зчитуватиме з клавіатури двовимірний масив цілих чисел `mas2D` розміром 3×3 та виводитиме на екран максимальне та мінімальне значення масиву. Значення масиву ініціалізувати в програмі.
19. Напишіть програму, яка зчитуватиме з клавіатури двовимірний масив цілих чисел `mas2D` розміром 4×5 та виводитиме на екран суму чисел, що знаходяться на головній та побічній діагоналях масиву.
20. Напишіть програму, яка обчислює та виводить на екран кількість ненульових непарних за значенням індексу елементів тривимірного масиву цілих чисел `mas3D` розміром 2×3×4. Значення масиву ініціалізувати в програмі.
21. Напишіть програму, яка обчислює та виводить на екран суму елементів чотиривимірного масиву цілих чисел `mas4D` розміром 2×3×4×2, сума значень індексів яких менша 5. Значення масиву ініціалізувати в програмі.
22. Із заданої на площині множини точок вибрати три різні точки так, щоб різниця між площею кола, обмеженого окружністю, що проходить через ці три точки, і площею трикутника, з вершинами в цих точках, була мінімальною.

7 ФУНКЦІЇ

7.1 Основні поняття

Функція – це програмний блок, що має унікальне ім'я, і містить групу виразів та операторів, що виконують одну або декілька логічно завершених дій.

Кожна програма написана мовою С повинна містити функцію **main()**, яка при запуску програми викликається автоматично.

Функція описується спеціальними синтаксичними засобами і може бути викликана з різних частин функції **main()** або з інших функцій.

Синтаксис виклику функції такий:

```
<ім'я функції> (<аргумент1>, <аргумент2>,…)
```

де *ім'я функції* – ідентифікатор, за допомогою якого здійснюється виклик функції;

аргумент1, аргумент2,… – список аргументів функції. Аргументом називають значення, яке передається у функцію.

Виклик функції може бути частиною виразу чи аргументом іншої функції.

Стандартна бібліотека С містить широкий набір функцій, які можна використати у програмі, підключивши відповідні заголовні файли, наприклад:

- **<math.h>** – виконання загальних математичних обчислень;
- **<string.h>** – обробка рядків та символів;
- **<stdio.h>** – введення/виведення інформації;
- **<stdlib.h>** – виділення пам'яті, контроль процесу виконання програми.

Підключення заголовних файлів до програми здійснюється за допомогою директиви **#include**.

В табл. 7.1 наведені деякі функції заголовного файлу **<math.h>**.

Окрім функцій стандартної бібліотеки С програміст може визначати власні функції.

Синтаксис визначення функції:

```
<тип> <ім'я функції> (<параметр1>, <параметр2>,…)
{
  <тіло функції>
}
```

де *тип* – тип даних значення, яке повертається функцією у місце виклику;

ім'я функції – ідентифікатор, що визначає ім'я функції;

параметр1, параметр2,… – список параметрів функції. Кожен параметр складається з типу даних та ідентифікатора, як і будь-яке оголошення змінної (наприклад: **int x**). Змінні, які оголошені як параметр, є локальними для даної функції. Вони приймають значення аргументів при виклику функції;

тіло функції – це блок операторів у фігурних дужках {}.

Тип кожного аргументу функції повинен відповідати типу відповідного параметра, оголошеного при визначенні функції.

Таблиця 7.1 – Функції заголовного файлу **<math.h>**

Функція	Призначення	Приклад
abs(x)	повертає модуль числа x	abs(-14.0) дорівнює 14
acos(x)	повертає арккосинус x	acos(0.0) дорівнює 1.570796
asin(x)	повертає арксинус x	asin(0.0) дорівнює 0.0
atan(x)	повертає арктангенс x	atan(1.0) дорівнює 0.785398
ceil(x)	повертає x округлений до більшого	ceil(4.2) дорівнює 5.0
cos(x)	повертає косинус x	cos(3.14) дорівнює-0.999999
cosh(x)	повертає гіперболічний косинус x	cosh(0.0) дорівнює 1.0
exp(x)	повертає експоненту x(e в степені x)	exp(1.0) дорівнює 2.718282
floor(x)	повертає x округлений до меншого	floor(4.9) дорівнює 4.0
log(x)	повертає натуральний логарифм x	log(2.718282) дорівнює 1.0
log10(x)	повертає логарифм x за основою 10	log10(1000) дорівнює 3.0
pow(x,y)	повертає x піднесене до степеня y	pow(2.0,5.0) дорівнює 32.0
sin(x)	повертає синус x	sin(0.0) дорівнює 0.0
sinh(x)	повертає гіперболічний синус x	sinh(0.0) дорівнює 0.0
sqrt(x)	повертає квадратний корінь з x	sqrt(4.0) дорівнює 2.0
tan(x)	повертає тангенс x	tan(0.785398) дорівнює 1.0
tanh(x)	повертає гіперболічний тангенс x	tanh(10.0) дорівнює 1.0

Приклад 7.1. Використання стандартних функцій та функцій, визначених у програмі.

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

int add( int a, int b )          /* визначення функції add */
{
  int q;
  q = a + b;
  return (q);
}

int main ()
{
  setlocale( LC_ALL, "Ukrainian" );
  int t;
  t = add( 7, 4 );               /* виклик функції add */
  printf("Сума = %i \n", t); /* виклик функції printf з
                                заголовного файлу stdio.h */
  system( "PAUSE" );            /* виклик функції system з
                                заголовного файлу stdlib.h */
  return 0;
}
```

Результат роботи програми:

```
Сума = 11
```

На початку програми здійснюється визначення функції **add()**, далі здійснюється визначення функції **main()**. Функція **main()** починається з оголошення змінної **t** типу **int**, після цього, здійснюється виклик функції **add()**. Подібність виклику та визначення функції зображено на рис. 7.1.

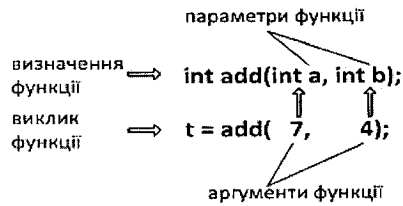


Рисунок 7.1 – Визначення та виклик функції

При виклику функції **add()** як аргументи їй передаються два значення: **7** і **4**, які відповідають параметрам **int a** та **int b**.

Значення аргументів при виклику (**7** і **4**) будуть скопійовані в локальні змінні **int a** та **int b**.

У функції **add()** оголошується інша локальна змінна **int q**, якій присвоюється результат виразу **a+b**.

Наступний рядок коду **return (q)** завершує роботу функції **add()** і повертає значення **q** у функцію **main()**. Програма продовжує роботу з того ж місця, в якому вона була перервана викликом функції **add()**. Значення змінної **q** в цей момент дорівнює **11** і воно стає значенням функції, що викликається. Потім це значення присвоюється змінній **t** (рис. 7.2).

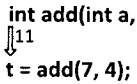


Рисунок 7.2 – Повернення значення функцією

7.2 Види виклику функцій

Існує декілька видів виклику функцій. Розглянемо їх на такому прикладі.

Приклад 7.2. Види виклику функцій.

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

int sub( int a, int b )
{
  int q;
  q = a - b;
  return (q);
}
int main()
{
  setlocale( LC_ALL, "Ukrainian" );
  int x=2, y=1, t;
  t = sub( 7, 4 );
  printf( "Перший %i \n", t );
  printf( "Другий %i \n", sub(7, 4) );
  printf( "Третій %i \n", sub (x, y) );
  t = 4 + sub(x, y);
  printf( "Четвертий %i \n", t );
  system("PAUSE"); return 0;
}
```

Результат роботи програми:

```
Перший 3
Другий 3
Третій 1
Четвертий 5
```

У прикладі 7.2. створена функція **sub()**, яка повертає різницю значень аргументів. В програмі продемонстровано чотири варіанти виклику функції **sub()**.

Перший варіант виклику:

```
t = sub( 7, 4 );
printf( "First %i \n" , t );
```

Другий варіант виклику:

```
printf( "Second %i \n", sub(7, 4) );
```

У цьому варіанті функція **sub()** викликається як аргумент функції **printf()**.

Третій варіант виклику:

```
printf( "Third %i \n", sub(x, y) );
```

У цьому варіанті замість числових літералів (7, 4) як аргументи функції **sub()** виступають змінні **x** та **y**.

Четвертий варіант виклику:

```
t = 4 + sub( x, y );
printf( "Fourth %i \n", t );
```

У цьому варіанті показано можливість роботи з функцією, як зі звичайною змінною такого ж типу, який повертає функція.

7.3 Область видимості

Видимість змінних, оголошених у функції, обмежується тілом функції, тобто вони є локальними змінними функції і не можуть бути використані поза нею. У прикладі 7.2. було б неможливо використовувати змінні **a**, **b** або **q** безпосередньо у функції **main()**, так як вони є локальними змінними функції **add()**. Крім того, було б неможливо використовувати змінну **t** у функції **add()**, так як вона є локальною змінною функції **main()**.

У випадку необхідності видимості змінної, як у межах функції так і поза нею, потрібно використати глобальну змінну. Для створення глобальної змінної потрібно оголосити її поза межами усіх функцій, в тому числі і функції **main()**.

7.4 Порожній тип void

Визначення функції починається з типу даних значення, яке буде повертатися функцією. Але, якщо не потрібно повертати жодного значення, то використовується порожній тип **void**.

Наприклад, потрібно створити функцію, яка буде виводити повідомлення на екран. Такій функції не потрібно повертати жодного значення.

Приклад 7.3. Використання порожнього типу **void**.

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

void message ()       /* функція яка не повертає значення */
{
  printf( "ЦЕ СПАРТА!!!\n" );
}

int main ()
{
  setlocale( LC_ALL, "Ukrainian" );
  message();
  system( "PAUSE" ); return 0;
}
```

Результат роботи програми:

```
ЦЕ СПАРТА!!!
```

В списку параметрів функції також може використовуватися ключове слово **void**, для того, щоб показати, що функція не має параметрів при виклику. Наприклад, функція **message()** могла бути визначена як:

```
void message (void)
{
  printf( "ЦЕ СПАРТА!!!\n" );
}
```

Хоча необов'язково вказувати **void** в списку параметрів. У мові С список параметрів може бути порожнім, якщо він не потребує жодних параметрів. Дужки необхідно використовувати навіть за відсутності параметрів. Їх наявність вказує на те, що це виклик функції, а не ім'я змінної або інший оператор мови С.

7.5 Передавання аргументів у функцію

В мові С змінні у функцію як аргументи передаються за значенням (*by value*).

Приклад 7.4. Передавання аргументів у функцію за значенням.

```
int x = 7, y = 4, t;
t = add( x , y);
```

У функцію **add()** передаються значення змінних **x** і **y**, тобто 7 і 4 відповідно (рис. 7.3).

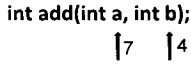


Рисунок 7.3 – Передавання аргументів у функцію за значенням

При виклику функції **add()** у локальні змінні **a** і **b** копіюються значення змінних **x** і **y**. Тому будь-яка зміна значень **a** або **b** в межах функції **add()** не вплине на значення змінних **x** і **y**.

7.6 Рекурсивні функції

Рекурсивна функція – це функція, в тілі якої здійснюється виклик цієї ж функції, тобто вона викликає саму себе. За допомогою рекурсії легко вирішуються такі задачі, як сортування або обчислення факторіала числа.

Факторіал цілого невід'ємного числа *n* записується як *n!* і обчислюється за формулою:

$$n! = n \times (n-1) \times (n-2) \times (n-3) \times \ldots \times 1.$$

Наприклад, факторіал п'яти є добутком $5 \times 4 \times 3 \times 2 \times 1$, який дорівнює 120.

Розглянемо рекурсивну функцію для обчислення факторіала.

Приклад 7.5. Обчислення факторіала.

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

int fact( int a )
{
  if ( a > 1 )
    return ( a * fact(a - 1) );
  else
    return (1);
}

int main ()
{
  setlocale( LC_ALL, "Ukrainian" );
  int n;
  printf( "Введіть число: " );
  scanf( "%i", &n );
  printf( "%i! = %i \n", n, fact(n) );
```

```
system( "PAUSE" ); return 0;}
```

Результат роботи програми:

```
Введіть число: 5
!5 = 120
```

Функція **fact()** викликатиме сама себе, постійно зменшуючи аргумент наступного виклику на одиницю до тих пір, поки аргумент не дорівнюватиме 1. В цьому випадку функція поверне число 1 (рис. 7.4).

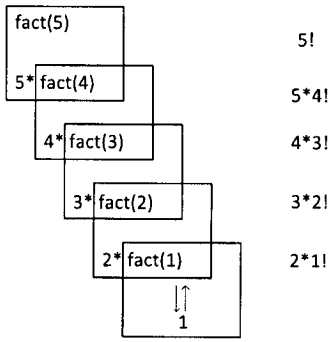


Рисунок 7.4 – Приклад обчислення 5! за допомогою рекурсивної функції

7.7 Прототипи функції

Прототип функції – це оголошення функції, яке не містить тіло функції і повідомляє компілятору:

- тип даних значення, яке повертає функція;
- число, тип та порядок параметрів, які функція приймає.

Прототип дає можливість викликати функцію до моменту її визначення.

Прототипи функцій використовуються для перевірки коректності виклику функцій. У випадку неузгодження прототипу функції та її визначення виникає помилка на етапі компіляції.

Синтаксис оголошення прототипу функції такий:

```
<тип> <ім'я функції> (<параметр1>, <параметр2>,…);
```

Цей синтаксис ідентичний визначенню функції, за винятком того, що він не містить тіло самої функції, а в кінці ставиться крапка з комою (;).

Компілятор ігнорує імена змінних, вказаних в прототипі функції, тобто при оголошенні прототипу достатньо вказати лише типи параметрів.

Приклад 7.6. Використання прототипу функції без застосування імен змінних у прототипі.

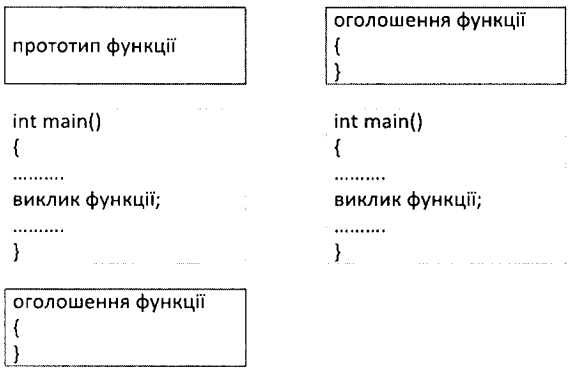
```
#include <stdlib.h>
#include <stdio.h>
#include <locale.h>

int function( int, int, int ); /* оголошення прототипу
                                  функції*/

int main()                     /* головна програма*/
{
  setlocale( LC_ALL, "Ukrainian" );
  printf( "Сума = %d \n", function (4, 5, 6) );
  system( "PAUSE" ); return 0;
}

int function(int a, int b, int c)    /* визначення функції*/
{
  return a + b + c;
}
```

На рис. 7.5 зображено структури програм з використанням прототипу функції та без нього.



а) з прототипом б) без прототипу

Рисунок 7.5 – Структура програми з використанням функції

Контрольні питання

1. Дайте означення терміну «функція».
2. Що Ви знаєте про функцію **main()**? Коли вона викликається?
3. Наведіть синтаксис виклику функції.
4. Які Ви знаєте заголовні файли стандартної бібліотеки С?
5. Наведіть синтаксис визначення функції.

6. Наведіть приклади та призначення функцій з заголовного файлу `math.h`.
7. В чому полягає різниця між визначенням і викликом функції?
8. В чому полягає різниця між аргументом і параметром функції?
9. Поясніть, що таке «повернення функцією значення».
10. Яке ключове слово потрібно використати, щоб зазначити, що функція не буде повертати жодного значення?
11. Наведіть види виклику функції.
12. Чи може вираз бути аргументом функції?
13. Чи може літерал бути аргументом функції?
14. Яка область видимості змінних оголошених у функції?
15. Що таке передавання аргументів у функцію за значенням?
16. Що таке рекурсивна функція?
17. Що таке прототип функції?
18. Наведіть синтаксис оголошення прототипу функції.

Контрольні завдання

1. Напишіть програму, яка, використовуючи функції з заголовного файлу `math.h`, буде виконувати такі дії:
   а) обчислювати модуль дійсного числа;
   б) округлювати до меншого дійсне число;
   в) округлювати до більшого дійсне число;
   г) обчислювати квадратний корінь дійсного числа;
   д) обчислювати логарифм числа за основою 10;
   е) підносити до степеня дійсне число;
   ж) обчислювати тангенс дійсного числа.
2. Напишіть функцію, яка обчислює та повертає суму значень двох змінних типу `int`, що передаються у функцію як аргументи.
3. Напишіть функцію, яка обчислює та виводить на екран значення периметра прямокутника, значення сторін якого передаються у функцію як аргументи.
4. Напишіть функцію, яка виконує арифметичні дії над першими двома своїми параметрами – змінними типу `float`. Третій параметр (змінна типу `int`) задає саму арифметичну дію, наприклад, можливі її значення: 1 –додавання, 2 – віднімання, 3 – множення, 4 – ділення. Результат виконаної арифметичної дії вивести на екран.
5. Напишіть функцію, яка перевіряє чи можна утворити трапецію з чотирьох відрізків, значення довжин яких задаються як аргументи.
6. Напишіть функцію обчислення об'єму циліндра. Аргументами функції мають бути змінні типу `float`, що задають радіус та висоту циліндра (формула для обчислення об'єму циліндра: $V = 2\pi rh$).
7. Напишіть функцію, що повертає найбільше значення з трьох заданих як аргументи змінних типу `int`.

8. Напишіть функцію, яка визначає кількість цифр цілого числа, що передається в неї як аргумент.
9. Напишіть функцію, яка обчислює та повертає відстань між двома точками з координатами (`x1`, `x2`) та (`y1`, `y2`), що отримані як аргументи (змінні типу `double`).
10. Напишіть нерекурсивну функцію, що обчислює *n*-е число ряду Фібоначі (0, 1, 1, 2, 3, 5, 8, 13, 21, ...).
11. Напишіть функцію, яка перевіряє чи є простим число, передане в неї як аргумент.
12. Напишіть функцію, яка обчислює кількість ненульових значень масиву цілих чисел, переданого у неї як аргумент.
13. Напишіть функцію, що повертає найменше значення серед елементів переданого їй масиву дійсних чисел.
14. Напишіть функцію, що повертає середнє арифметичне елементів переданого їй цілочисельного масиву. Використати функцію для знаходження середньої оцінки у групі з 10 студентів.
15. Напишіть функцію, яка отримує натуральне число і повертає результат «істина», якщо це число є простим. За допомогою функції знайти всі числа «близнюки», що менші 100 («близнюки» – прості числа, різниця між якими становить 2. Наприклад, 11 і 13).
16. Напишіть функцію, яка отримує ціле число і визначає чи є воно паліндромом (число є паліндромом, якщо його запис читається однаково зліва направо та навпаки. Наприклад, 1331. 45154). Скориставшись функцією, напишіть програму, що знаходить всі менші 100 натуральні числа, квадрат яких є паліндромом.
17. Напишіть функцію, яка отримує натуральне число і повертає результат «істина», якщо це число є простим. За допомогою функції перевірити гіпотезу Гольдбаха для деякого парного числа **n** (n > 2). Гіпотеза полягає у такому: кожне парне число **n** можна подати у вигляді суми двох простих чисел.
18. Напишіть функцію, яка запитує відстань у фарлонгах і перетворює її у ярди (1 фарлонг = 220 ярдів). Використати функцію для переведення 10 значень відстані із фарлонгів у ярди.

8 РЯДКИ В МОВІ С

В даному розділі розглядаються функції стандартної бібліотеки С, які спрощують обробку рядків та символів. Розглядаються функції, що описані в заголовних файлах:

- `stdio.h`;
- `string.h`;
- `ctype.h`;
- `stdlib.h`.

За допомогою цих функцій можна здійснювати операції з текстом, подібні до тих, що виконуються функціями форматованого введення / виведення `printf` та `scanf`.

8.1 Рядки і символи

У мові С немає окремого типу даних «рядок символів», подібного до типу `string` у алгоритмічній мові PASCAL. Тому робота з рядками реалізована шляхом використання одновимірних масивів типу `char`.

Рядок символів в мові С є одновимірним масивом типу `char`, останнім елементом якого є нульовий байт (`NULL`). Нульовий байт є ознакою закінчення рядка і визначається символьним літералом `'\0'`. Тому, якщо рядок містить `k` символів, в масиві потрібно передбачити місце для нульового байта, вказавши розмірність `k+1`.

Так, для збереження у масиві рядкового літерала `"underdog"`, необхідно описати масив `char s[9]`. В кінці рядкового літерала `'\0'` вказувати не потрібно, оскільки це зробить компілятор мови С (рис. 8.1).

Приклад 8.1. Збереження рядкового літерала у масиві `char`.

```
char s[9] = "underdog";
```

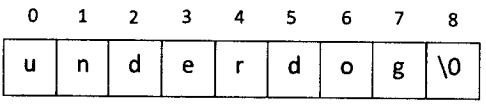


Рисунок 8.1 – Рядок як масив символів типу `char`

Приклад 8.2. Варіанти ініціалізації рядка символів.

```
char color [] = {'g', 'r', 'e', 'e', 'n', '\0' };
char color [6] = "green";
char color [] = "green";
```

Масиву символів можна присвоїти рядок, використовуючи функцію **scanf()**.

Приклад 8.3. Зчитування рядка у масив символів **string[20]**.

```
scanf("%s", string);
```

Приклад 8.4. Виведення на екран масиву символів.

```
#include <string.h>
#include <stdlib.h>

int main ()
{
  char myString[] = "It's a beautiful life!";
  printf("%s\n", myString);
  system("pause"); return 0;
}
```

Результат роботи програми:

```
It's a beautiful life!
```

8.2 Функції операцій над рядками

Прототипи функцій, що працюють з рядками символів, містяться у заголовному файлі **string.h**. Функції операцій над рядками описані в табл. 8.1, функції порівняння рядків описані в табл. 8.2.

Таблиця 8.1 – Функції операцій над рядками

Прототип функції	Опис функцій
`int strlen(const char *s)`	Визначає довжину рядка (кількість символів без символа **NULL**).
`char * strcpy(char * s1, const char *s2)`	Копіює рядок **s2** у рядок **s1**. Повертає значення **s1**.
`char * strncpy(char * s1, const char *s2, size_t n)`	Копіює не більше, ніж **n** символів рядка **s2** у рядок **s1**. Повертає значення **s1**.
`char * strcat(char *s1, const char *s2)`	Об'єднує рядок **s2** з рядком **s1**. Перший символ рядка **s2** переписує символ **NULL** рядка **s1**. Повертає значення **s1**.
`char * strncat(char *s1, const char *s2)`	Об'єднує не менше, ніж **n** символів рядка **s2** з рядком **s1**. Перший символ рядка **s2** переписує символ **NULL** рядка **s1**. Повертає значення **s1**.

Приклад 8.5. Використання функції **strlen()**.

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

int main ()
{
  char str[] = "Hello World!";

  printf("%i\n", strlen( str ) );

  system("pause"); return 0;
}
```

Результат роботи програми:

```
12
```

Приклад 8.6. Використання функцій **strcpy()** та **strncpy()**.

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

int main ()
{
  char xStr[] = "Hello World!";
  char yStr[25], zStr[25];
  int i;

  printf("%s\n", xStr, strcpy(yStr, xStr) );

  for(i = 0; i < 5; i++)
  {
    strncpy(zStr, xStr, i+1);
    zStr[i+1] = '\0';
    printf("%s\n", zStr);
  }
  system("pause"); return 0;
}
```

Результат роботи програми:

```
Hello World!
H
He
Hel
Hell
Hello
```

Приклад 8.7. Використання функцій **strcat()** та **strncat()**.

```
#include <stdio.h>
#include <string.h>
#include <stdlib.h>
#include <locale.h>

int main ()
{
  setlocale( LC_ALL, "Ukrainian" );
  char s1[20] = "не можна ";
  char s2[] = "помилувати";
  char s3[20] = "стратити ";

  printf("%s\n", strcat(s1, s2) );
  printf("%s\n", strncat(s3, s1, 6) );

  system("pause"); return 0;
}
```

Результат роботи програми:

```
не можна помилувати
стратити не можна
```

Таблиця 8.2 – Функції порівняння рядків

Прототип функції	Опис функцій
`int strcmp(const char *s1, const char *s2);`	Порівнює рядок **s1** з рядком **s2**. Функція повертає 0, значення менше 0 чи більше 0, якщо **s1** відповідно дорівнює, менший чи більший, ніж **s2**.
`int strncmp(const char *s1, const char *s2, size_t n);`	Порівнює до **n** символів рядка **s1** з рядком **s2**. Функція повертає 0, значення менше 0 чи більше 0, якщо **s1** відповідно дорівнює, менший чи більший, ніж **s2**.

Приклад 8.8. Використання функцій порівняння рядків.

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

int main ()
{
  char s1[] = "ab";
  char s2[] = "abc";
  char s3[] = "abcd";
  char s4[] = "abc";
  char s5[] = "ac";
```

```
  printf("%i ", strcmp(s1, s2) );
  printf("%i ", strcmp(s3, s2) );
  printf("%i ", strcmp(s2, s4) );
  printf("%i ", strncmp(s2, s3, 3) );
  printf("%i\n", strcmp(s1, s5) );

  system("pause"); return 0;
}
```

Результат роботи програми:

```
-1 1 0 0 -1
```

8.3 Функції перетворення рядків

У заголовному файлі **stdlib.h** описані деякі функції перетворення рядків. Ці функції перетворюють рядки в цілі значення і значення з плаваючою точкою. У табл. 8.3 наведений перелік та опис функцій перетворення рядків.

Таблиця 8.3 – Функції перетворення рядків

Прототип функції	Опис функції
`double atof (const char *nPtr)`	Перетворення рядка **nPtr** в тип **double**
`int atoi (const char *nPtr)`	Перетворення рядка **nPtr** в тип **int**
`long atol (const char *nPtr)`	Перетворення рядка **nPtr** в тип **long int**

Приклад 8.9. Використання функцій перетворення рядків.

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

int main()
{
  double dValue;
  int iValue;
  long lValue;

  dValue = atof("3.14");
  iValue = atoi("2001");
  lValue = atol("1000000");

  printf("%.2f %i %ld\n", dValue, iValue, lValue);

  system("PAUSE"); return 0;
}
```

Результат роботи програми:

```
3.14 2001 1000000
```

8.4 Функції введення/виведення символів та рядків

Окрім розглянутих функцій форматованого введення/виведення **printf** та **scanf** в заголовному файлі **stdio.h** описані також функції, що призначені для операцій з символьними та рядковими даними (табл. 8.4).

Таблиця 8.4 – Функції введення/виведення стандартної бібліотеки С

Прототип функції	Опис функції
`int getchar (void)`	Вводить наступний символ зі стандартного пристрою введення і повертає його в форматі цілого
`char *gets (char *s)`	Вводить символи зі стандартного пристрою введення в масив до тих пір, поки не зустріне символ нового рядка чи індикатор кінця файлу
`int putchar (int c)`	Друк символа, що зберігається в **c**
`int puts (const char *s)`	Друк рядка **s** з наступним символом нового рядка
`int sprintf (char *s, const char *format, … )`	Еквівалент функції **printf** за винятком того, що результат виведення запам'ятовується в масиві **s**, а не відображається на екрані
`int sscanf (char *s, const char *format, … )`	Еквівалент функції **scanf** за винятком того, що введення здійснюється з масиву **s**, а не з клавіатури

Приклад 8.10. Використання функцій **gets()** та **putchar()**.

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

void palindrom (char *);

int main()
{
  setlocale( LC_ALL, "Ukrainian" );
  char string[80];

  printf("Введіть рядок:\n");
  gets(string);

  palindrom(string);
```

```
  system("PAUSE"); return 0;
}

void palindrom (char *s)
{
  if(s[0] == '\0')
    return;
  else
  {
    palindrom(&s[1]);
    putchar(s[0]);
  }
}
```

Результат роботи програми:

```
Введіть рядок:
play your card right and I will sort you out
tuo uoy tros liww I dna thgir drac ruoy yalp
```

Функція **gets()** зчитує символи зі стандартного пристрою введення і передає їх своєму аргументу – масиву типу **char** до тих пір, поки не зустрінеться символ нового рядка чи індикатор кінця файлу. Символьний літерал **'\0'** додається в масив після закінчення зчитування. Для друку рядка в зворотньому напрямку програма викликає функцію **palindrom()**. Якщо перший елемент масиву, отриманий функцією **palindrom()** є нульовим байтом (**'\0'**), то відбувається повернення з функції. В іншому випадку відбувається виклик функції **palindrom()** і її аргумент вказує на адресу нового масиву, який починається з елемента **s[0]**. Коли рекурсивний виклик закінчується, функція **putchar()** виводить елемент **s[0]**.

Приклад 8.11. Використання функцій **puts()** та **getchar()**.

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

int main()
{
  setlocale( LC_ALL, "Ukrainian" );
  char c, string[80];
  int i = 0;
  puts ("Введіть рядок:");

  while( ( c = getchar() ) != '\n' )
    string [i++] = c;

  string[i] = '\0';
```

```
  puts (string);

  system("PAUSE"); return 0;
}
```

Результат роботи програми:

```
Введіть рядок:
To be or not to be
To be or not to be
```

Програма закінчує введення символів, коли **getchar()** зчитує введений користувачем символ нового рядка (**'\n'**). В масив **string** додається символьний літерал **'\0'**, щоб масив зміг сприйматися як рядок. Функція **puts()** виводить на екран рядок, що міститься в **string**.

Приклад 8.12. Використання функції **sscanf()**.

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

int main()
{
  float y;
  int x;
  char str [] = "1914 3.1415";

  sscanf(str, "%d%f", &x, &y);
  printf("%s %6d \n%s %5.3f\n", "int:", x, "float:", y);
  system("PAUSE"); return 0;
}
```

Результат роботи програми:

```
int:   1914
float: 3.141
```

Програма зчитує значення типу **int** та **float** з масиву **str** і запам'ятовує величини, відповідно, в змінних **x** та **y**. Величини **x** та **y** виводяться на друк. Масив **str** є першим аргументом функції **sscanf()**.

8.5 Функції перевірки літер

У заголовному файлі **ctype.h** описані прототипи функцій, що призначені для перевірки літер. Кожна функція отримує як аргумент символ – поданий типом **int** або індикатор кінця файлу (**EOF**). Ці функції оперують з символами як з цілими числами (табл. 8.5).

Таблиця 8.5 – Функції перевірки літер

Прототип функції	Опис функції
`int islower(int c)`	Повертає ненульове значення, якщо `c` є малою літерою і 0 в інших випадках
`int isupper(int c)`	Повертає ненульове значення, якщо `c` є великою літерою і 0 в інших випадках
`int isalnum(int c)`	Повертає ненульове значення, якщо `c` є цифрою чи літерою і 0 в інших випадках
`int isalpha(int c)`	Повертає ненульове значення, якщо `c` є буквою і 0 в інших випадках
`int tolower(int c)`	Якщо `c` є великою літерою, то повертає `c` як малу літеру. В іншому випадку повертає аргумент без змін

Приклад 8.13. Використання функцій перевірки літер.

```
#include <ctype.h>
#include <stdio.h>
#include <locale.h>
#include <stdlib.h>

int main(void)
{
  setlocale( LC_ALL, "Ukrainian" );
  char ch;
  while( ch != '1' )
  {
    ch = getchar();
    if ( isalnum(ch) )
      printf( "%c - це літера чи цифра\n", ch );
    if ( isalpha(ch) )
      printf( "%c - це літера\n", ch );
    if ( islower(ch) )
      printf( "%c - це мала літера\n", ch );
    if ( isupper(ch) )
      printf( "%c - це ВЕЛИКА літера\n", ch );
  }
  system("pause"); return 0;
}
```

Результат роботи програми:

```
Q - це літера чи цифра
Q - це літера
Q - це ВЕЛИКА літера
1 - це літера чи цифра
```

Програма буде працювати, поки з клавіатури не буде введений символ `'1'`. На кожній ітерації циклу **while** за допомогою функції **getchar()** в змінну **ch** зчитується символ.

Контрольні питання

1. Для чого потрібні рядки?
2. Що таке нульовий символ?
3. Наведіть приклад функцій, які працюють з рядками.
4. Наведіть приклад функцій перевірки літер.
5. Як звертатись до елементів рядка?
6. Які операції можна виконувати з рядками?
7. Наведіть приклади ініціалізації рядків.
8. Що таке символьні константи?
9. Для чого потрібні символьні константи?
10. Чим відрізняється символ від рядка, який містить 1 символ?

Контрольні завдання

1. Напишіть програму, яка знаходить та виводить на екран скільки разів у масиві символів **s[10]** зустрічається введений з клавіатури символ. Масив ініціалізувати у програмі.
2. Напишіть програму, яка виводить на екран тільки голосні букви, що зустрічаються у масиві символів **s[10]**. Масив ініціалізувати у програмі.
3. Напишіть програму, яка обчислює та виводить на екран кількість цифр, що зустрічаються у масиві символів **s[10]**. Масив ініціалізувати у програмі.
4. Напишіть програму, яка зчитує з клавіатури слово та виводить його на екран в зворотньому напрямку (з останньої літери до першої).
5. Напишіть програму, яка зчитує з клавіатури слово та виводить на екран всі його літери через пробіл.
6. Напишіть програму, яка перевіряє чи є введений з клавіатури рядок паліндромом.
7. Напишіть програму, яка зчитує з клавіатури слово та виводить на екран «true», якщо слово є паліндромом, і «false» – в іншому випадку.
8. Напишіть програму, яка зчитує з клавіатури 5 слів та виводить на екран найдовше з них.
9. Напишіть програму, яка вводить з клавітури 3 рядки, що є цілими значеннями, перетворює ці рядки в цілі числа та виводить на екран суму значень цих чисел.
10. Напишіть програму, яка використовує функцію **strcmp** для порівняння двох рядків, що вводяться користувачем. Програма повинна визначити: перший рядок більший, менший чи рівний другому та вивести відповідне повідомлення.

11. Напишіть програму, яка визначає чи є введена з клавіатури лексема ідентифікатором.
12. Напишіть програму, яка визначає чи є введена з клавіатури лексема ключовим словом мови С.
13. Напишіть програму, яка знаходить та виводить на екран найкоротше та найдовше слово у введеному з клавіатури рядку.
14. Напишіть програму, яка перетворює введений з клавіатури рядок таким чином, щоб спочатку у ньому розташовувалися цифри, а потім літери. Перетворений рядок вивести на екран.
15. Напишіть програму, яка виводить на екран код ASCII та відповідний йому символ.
16. Напишіть програму, яка вводить рядки тексту в масив символів `s[50]`, використовуючи функцію `gets`. Виведіть рядки в верхньому та нижньому регістрі.
17. Напишіть програму, яка перетворює введений з клавіатури рядок таким чином, щоб усі букви в ньому були відсортовані за зростанням.
18. Напишіть програму, яка перетворює введений з клавіатури рядок таким чином, щоб регістр кожної букви рядка був змінений на протилежний.
19. Напишіть програму, яка перетворює введений з клавіатури рядок таким чином, щоб у ньому залишилися лише слова, які є ідентифікаторами, усі інші слова видалити.
20. Напишіть програму, яка обчислює скільки разів зустрічається задане у програмі слово у введеному з клавіатури рядку.
21. Напишіть програму, яка виводить ті слова у введеному з клавіатури рядку, розмір яких більше заданого у програмі.
22. Напишіть програму, яка визначає яке слово у введеному з клавіатури рядку зустрічається найчастіше.
23. Напишіть програму, яка виводить на екран ті слова у введеному з клавіатури рядку, які містять лише одну цифру.
24. Напишіть програму, яка виводить на екран найдовше та найкоротше слово у введеному з клавіатури рядку.
25. Напишіть програму, яка виводить на екран ті слова у введеному з клавіатури рядку, які не містять голосних букв.
26. Напишіть програму, яка виводить на екран кожне слово, з введеного з клавіатури рядка у зворотньому напрямку.
27. Напишіть програму, яка виводить на екран всі цифри, що зустрічаються у введеному з клавіатури рядку.
28. Напишіть програму, яка виводить на екран слова з введеного з клавіатури рядка, що складаються виключно з цифр.
29. Напишіть програму, яка виводить на екран всі слова-паліндроми з введеного з клавіатури рядка.

9 ПОКАЖЧИКИ

Типова ЕОМ має масив послідовно нумерованих або адресованих комірок (секцій) пам'яті, якими можна маніпулювати окремо або прилеглими групами. Кожна з комірок має мінімальний розмір, один байт. Комірки нумеруються послідовно (рис. 9.1).

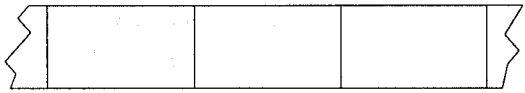


Рисунок 9.1 – Послідовна нумерація комірок пам'яті

На рис. 9.1 адреси комірок зображуються у вигляді двокрапки та номера комірки. Наприклад, «:1979».

Покажчик – це змінна, значенням якої є адреса пам'яті.

У «звичайних» змінних безпосередньо міститься деяке значення. Покажчик же містить адресу змінної, в якій міститься конкретне значення. Кажуть, що змінна *безпосередньо* посилається на значення, а покажчик посилається на значення *опосередковано*. Посилання на значення через покажчик називається *опосередкованою адресацією*.

Покажчики є незамінними для:

- створення масивів довільної величини (динамічних масивів);
- передавання масивів у функцію та доступ до їх елементів;
- отримання з функції декілька значень через її аргументи;
- створення зв'язних списків.

9.1 Оголошення та ініціалізація покажчиків

Синтаксис оголошення покажчика такий:

```
<тип> * <ім'я покажчика>
```

де *тип* – це тип даних, на який вказує покажчик;
ім'я покажчика – ідентифікатор.

Тип даних покажчика явно невизначений. Покажчики містять адреси пам'яті та при цьому не можуть мати окремий тип даних, оскільки важливою є не тільки адреса першої комірки, але і їх кількість. Для зберігання даних може бути залучена різна кількість комірок. Таким чином, при оголошенні покажчика основну роль відіграє його ім'я. Символ (*) належить

до імені і застосовується для того, щоб визначити змінну, як покажчик на тип даних, що характеризує значення за адресою.

Приклад 9.1. Оголошення покажчика.

```
int * aPtr, a;
```

У прикладі оголошується змінна **aPtr** типу **int *** (покажчик на ціле значення), що читається як «**aPtr** – покажчик на ціле», чи «**aPtr** – вказує на об'єкт типу **int**». Крім цього визначається змінна **a** цілого типу. Символ (*) належить тільки до **aPtr**, він означає, що змінна, яка оголошується, є покажчиком.

Покажчики повинні бути ініціалізовані одразу при визначенні або при використанні окремого оператора присвоювання. Покажчик може бути ініціалізований нулем, макросом **NULL** або значенням адреси.

Синтаксис ініціалізації покажчика при визначенні:

*<тип> * <ім'я покажчика> = <ініціалізатор>;*

Під *ініціалізатором* розуміється адреса змінної або вже існуючий покажчик.

Приклад 9.2. Ініціалізація покажчиків.

```
int a = 21;
int *aPtr1 = &a;    /* ініціалізація при визначенні
                       значенням адреси змінної a */
int *aPtr2;
aPtr2 = &a;         /* ініціалізація з використанням
                       окремого оператора присвоювання*/
int *aPtr3 (NULL); /* ініціалізація макросом NULL */
int *aPtr4 = 0;     /* ініціалізація нулем */
int *aPtr5 = aPtr1;/* ініціалізація при визначенні
                       вже існуючим покажчиком */
```

9.2 Операції взяття адреси та розіменування

Операція взяття адреси (**&**) є унарною операцією, яка повертає адресу свого операнда.

Приклад 9.3. Операція взяття адреси (**&**).

```
int a = 5; /* визначення та ініціалізація змінної a */
int *aPtr; /* визначення покажчика aPtr */
aPtr = &a; /* ініціалізація покажчика aPtr */
```

В останньому рядку прикладу 9.3. здійснюється присвоєння покажчику **aPtr** адреси змінної **a**. Після цього можна сказати, що змінна **aPtr** «вказує на» **a**. На рис. 9.2 наведене графічне подання покажчика на змінну цілого типу в результаті дій з прикладу 9.3.

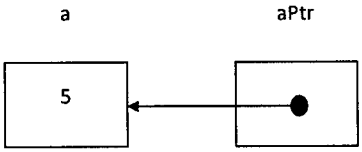


Рисунок 9.2 – Графічне подання покажчика на змінну цілого типу

Приклад 9.4. Використання операції взяття адреси (**&**) (рис. 9.3).

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

int main()
{
  int a = 8;
  int * p = &a;
  printf("p=%p \n&p=%p \n a=%i\n &a=%p\n", p, &p, a, &a);
  system("PAUSE"); return 0;
}
```

Результат роботи програми:

```
p=0022FF44
&p=0022FF40
a=8
&a=0022FF44
```

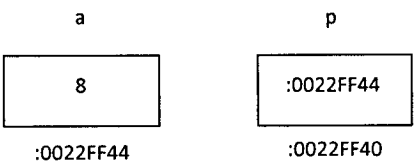


Рисунок 9.3 – Розміщення в пам'яті і вміст покажчика **p** та змінної **a**

В змінній **a** зберігається значення **8**, в покажчику **p** зберігається значення адреси змінної **a**, тобто **p** вказує на комірку з ідентифікатором **a**.

Унарна операція (*), як правило, називається операцією *опосередкованої адресації* або *розіменування*, повертає значення об'єкта, на який операнд (тобто покажчик) посилається.

Приклад 9.5. Розіменування покажчика (рис. 9.4).

```
int a = 1492;
int * aPtr = &a;   /* ініціалізація покажчика aPtr значенням
                      адреси змінної a */
int b = * aPtr;    /* через покажчик aPtr змінній b
                      присвоюється значення 1 */
```

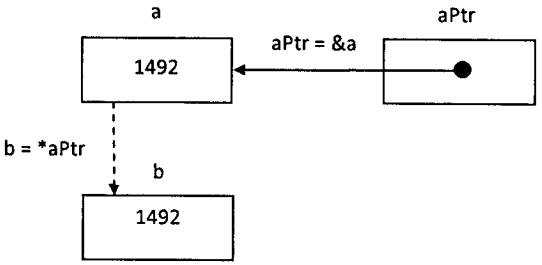


Рисунок 9.4 – Розіменування покажчика

Розглянемо приклад 9.5. Спочатку оголошується та ініціалізується змінна **a** . Потім визначається та ініціалізується покажчик **aPtr** значенням адреси змінної **a**. Нехай необхідно змінній **b** присвоїти значення змінної, на яку вказує покажчик. Для цього потрібно застосувати операцію розіменування покажчика (*). Для розіменування покажчика **aPtr** потрібно записати: ***aPtr**. Цей вираз словами можна описати як «те, що міститься за адресою, на яку вказує **aPtr**». Отримане при розіменуванні значення присвоюється змінній **b**.

9.3 Використання кваліфікатора const з покажчиками

Кваліфікатор **const** дає можливість програмісту повідомити компілятору про те, що значення вказаної змінної не повинно змінюватися.

Існує чотири способи передати функції покажчика:

1) неконстантний покажчик на неконстантні дані;
2) неконстантний покажчик на константні дані;
3) константний покажчик на неконстантні дані;
4) константний покажчик на константні дані.

Приклад 9.6. Способи визначення покажчика з const.

```
int *a1; /* спосіб №1 */
int const *a2; /* спосіб №2 */
const int *a3; /* спосіб №2 */
int *const a4; /* спосіб №3 */
int const *const a5; /* спосіб №4 */
const int *const a6; /* спосіб №4 */
```

Найвищий рівень доступу надається неконстантним покажчикам на неконстантні дані. В цьому випадку дані можуть змінюватися через розіменування покажчика, а сам покажчик може змінюватися і посилатися на інші елементи даних.

Неконстантний покажчик на константні дані – це покажчик, який може змінюватися і посилатися на будь-який об'єкт даних відповідного типу, але елемент даних, на які він вказує, не може змінюватися. Такий покажчик міг би передавати масив як аргумент функції, яка обробляє кожен елемент масиву і при цьому не змінює дані.

Приклад 9.7. Неконстантний покажчик на константні дані.

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

void printChar ( const char *sPtr );

int main()
{
  char string [] = " Hello World !";
  printChar( string );

  system ("pause"); return 0;
}

void printChar ( const char *sPtr )
{
  for (; *sPtr != '\0'; sPtr++)
    printf ( "%c" , *sPtr);
}
```

Результат роботи програми:

```
Hello World !
```

Константний покажчик на неконстантні дані – покажчик, який завжди посилається на одне й те ж місце в пам'яті, а розташовані там дані можуть змінюватися.Такий покажчик призначається за замовчуванням для імені масиву. Ім'я масиву є константним покажчиком на початок масиву. До всіх елементів масиву можна звертатися і змінювати їх, використовуючи ім'я масиву та його індекс. Константний покажчик на неконстантні дані може передавати масив як параметр функції, яка звертається до його елементів, використовуючи тільки індекс масиву. Константі покажчики повинні бути ініціалізовані при оголошенні.

Приклад 9.8. Константний покажчик на неконстантні дані.

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

```
int main()
{
  int a, b;
  int *const ptr = &a;

  *ptr = 7;
  ptr = &b; /* помилка, спроба присвоїти покажчику ptr
               нову адресу  */
  system ("pause"); return 0;
}
```

Мінімальні права доступу дає константний покажчик на константні дані. Такий покажчик завжди вказує на одне й те ж місце в пам'яті, а розташовані за цією адресою дані не можуть модифікуватися. Цьому варіанту відповідає передавання масиву функції, яка тільки переглядає масив за допомогою індексу і не змінює його елементи.

Приклад 9.9. Константний покажчик на константні дані.

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

int main()
{
  int a = 5;
  int b;
  const int *const ptr = &a;

  printf("%d \n", *ptr);

  *ptr = 7; /* помилка, спроба через покажчик ptr
               змінити вміст змінної a */
  ptr = &b; /* помилка, спроба присвоїти покажчику ptr
               нову адресу  */

  system ("pause"); return 0;
}
```

9.4 Зв'язок між покажчиками і масивами

Масиви та покажчики в мові С тісно пов'язані один з одним. Ім'я масиву можна розглядати як константний покажчик на перший елемент масиву. Над покажчиками можна виконувати різні операції, в тому числі використовувати з покажчиком індексні вирази.

Приклад 9.10. Ім'я масиву як покажчик на перший елемент.

```
int mas [5] = { 1, 2, 3, 4, 5 };
```

```
printf("*mas = %i \n ", *mas); /* Виведе на екран: 1 */
printf("*mas = %i \n ", *(mas+0)); /* Виведе на екран: 1 */
```

Приклад 9.11. Ім'я масиву як константий покажчик.

```
int mas [5] = { 5, 4, 3, 2, 1 };
mas++;     /* помилка, константний покажчик mas не може бути
              змінений */
```

Приклад 9.12. Доступ до елемента масиву через покажчик.

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

int main()
{
  float mas [4] = { 0.67, 2.56, 45.0, 100.7 };
  float * ptr;
  ptr = mas;

  printf(" mas[1] = %.2f \n ", mas[1] );
  printf("*(mas+1) = %.2f \n ", *(mas+1) );
  printf("*(ptr+1) = %.2f \n ", *(ptr+1) );
  printf(" ptr[1] = %.2f \n ", ptr[1] );

  system("pause"); return 0;
}
```

Результат роботи програми:

```
 mas[1] = 2.56
*(mas+1) = 2.56
*(ptr+1) = 2.56
 ptr[1] = 2.56
```

Літерал `1` в попередньому прикладі, що використовується у виразах `*(ptr+1)` та `*(mas+1)` називається *зміщенням*. Таки чином, розглянуто чотири варіанти звернення до елемента масиву:

- індексація масиву (`mas [1]`);
- зміщення з іменем масиву як покажчиком (`*(mas+1)`);
- покажчик/зміщення (`*(ptr+1)`);
- індексація покажчика (`ptr [1]`).

9.5 Вирази та арифметичні операції з покажчиками

Покажчики є допустимими операндами в арифметичних виразах, виразах присвоювання та порівняння.

До покажчиків може застосовуватися обмежений набір арифметичних операцій:

- покажчик може бути інкрементований (++) чи декрементований (--);
- до покажчика може бути додано ціле число (+ чи +=);
- від покажчика можна відняти ціле число (- чи -=);
- можна обчислити різницю покажчиків.

Приклад 9.13. Арифметичні операції з покажчиками.

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

int main()
{
  int m[10] = {0, 1, 2, 3, 4, 5, 6, 7, 8, 9};
  int *mPtr, *mPtr2;
  mPtr = &m[0];
  mPtr2 = &m[7];

  printf("%p %d \n", mPtr, *mPtr);
  printf("%p %d \n", mPtr + 2, *(mPtr + 2));
  mPtr++;
  printf("%p %d \n", mPtr, *mPtr);
  mPtr+=3;
  printf("%p %d \n", mPtr, *mPtr);
  mPtr-=1;
  printf("%p %d \n", mPtr, *mPtr);

  printf("%p %d \n", mPtr2 - mPtr, mPtr2 - mPtr);

  system ("pause"); return 0;
}
```

Результат роботи програми:

```
0022FF10 0
0022FF18 2
0022FF14 1
0022FF20 4
0022FF1C 3
00000004 4
```

У прикладі визначається та ініціалізується масив **m[10]** з десяти цілих чисел. Визначаються два покажчики, які ініціалізуються значеннями адрес нульового та сьомого елементів масиву **m**. Потім виводяться на екран адреси та значення елементів масиву **m**, на які вказує покажчик **mPtr** після виконання арифметичних дій. На рис. 9.5 зображене графічне подання зміщення покажчика **mPtr** по масиву **m** в результаті виконання арифметичних дій.

Кількість байтів, на які зміщується покажчик в результаті виконання арифметичних дій, визначається типом даних, на який покажчик вказує. Розглянемо деякі типи та зміщення покажчика для них: **int** (4 байти), **long** (4 байти), **unsigned int** (4 байти), **float** (4 байти), **double** (8 байтів), **char** (1 байт).

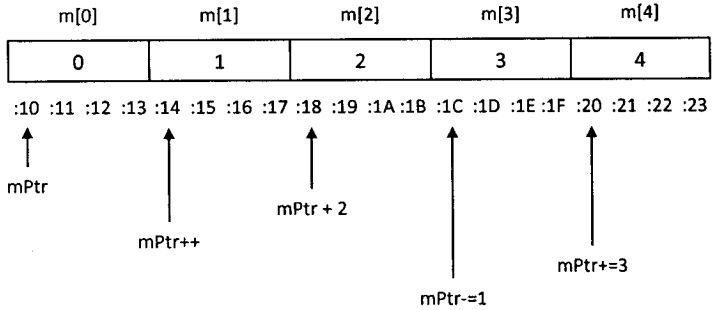


Рисунок 9.5 – Зміщення покажчика **mPtr** по масиву **m** в результаті виконання арифметичних дій

Приклад 9.14. Зміщення покажчика для різних типів даних (рис. 9.6).

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

int main()
{
  int mInt[4] = {0, 1, 2, 3};
  char mChar[4] = {'a', 'b', 'c', 'd'};
  float mFloat[4] = {0.5, 1.5, 2.5, 3.5};
  double mDouble[4] = {0.8, 1.8, 2.8, 3.8};

  int *mPtrInt = &mInt[0];
  char *mPtrChar = &mChar[0];
  float *mPtrFloat = &mFloat[0];
  double *mPtrDouble = &mDouble[0];

  printf( "%p %p ", mPtrInt, mPtrInt + 1 );
  printf( "%d %d \n", *mPtrInt, *(mPtrInt + 1));

  printf( "%p %p ", mPtrChar, mPtrChar + 1 );
  printf( "%c %c \n", *mPtrChar, *(mPtrChar + 1) );

  printf( "%p %p ", mPtrFloat, mPtrFloat + 1 );
  printf( "%.1f %.1f \n", *mPtrFloat, *(mPtrFloat + 1));

  printf( "%p %p ", mPtrDouble, mPtrDouble + 1);
  printf( "%.1f %.1f \n", *mPtrDouble, *(mPtrDouble + 1));
```

```
  system ("pause"); return 0;
}
```

Результат роботи програми:

```
0022FF30 0022FF34 0 1
0022FF2C 0022FF2D a b
0022FF10 0022FF14 0.5 1.5
0022FEF0 0022FEF8 0.8 1.8
```

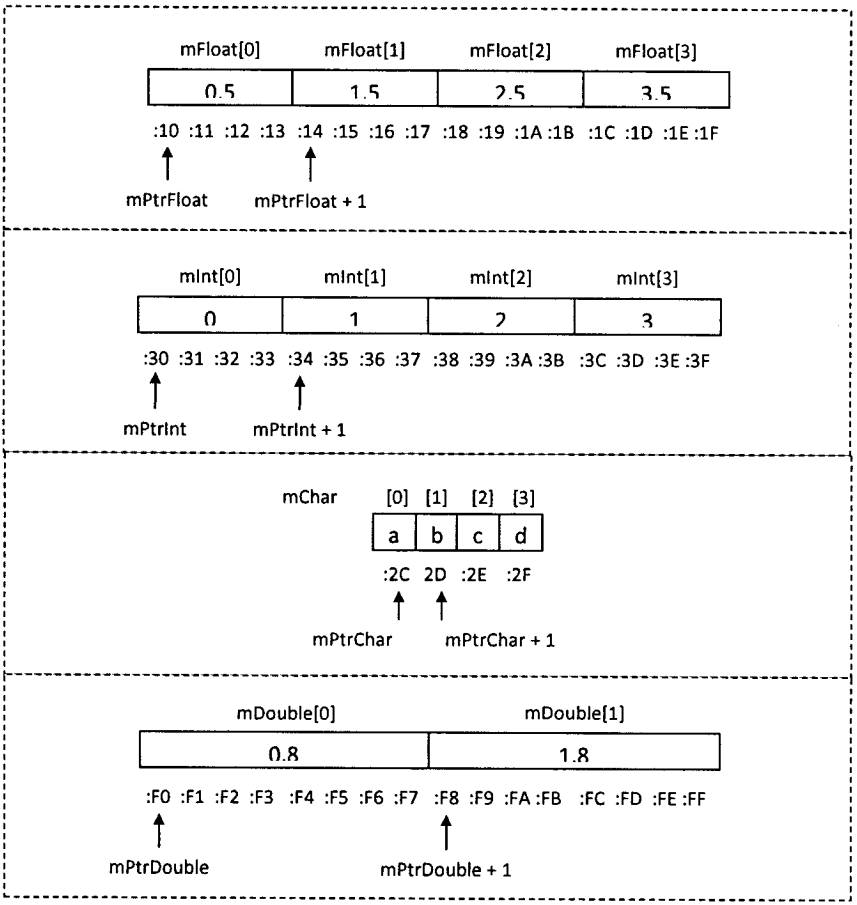


Рисунок 9.6 – Зміщення покажчика для різних типів даних

9.6 Покажчики як аргументи функцій

Крім передавання аргументів функції за значенням у мові С передбачене передавання аргументів функції через покажчик. Якщо функція призначена для зміни змінної у програмі, в якій функція викликається, то потрібно використовувати передавання аргументів через покажчик. Так як при передаванні змінної за значенням функція отримує тільки копію змінної.

Розглянемо передавання аргументів функції через покажчик на прикладі програми, що здійснює заміну місцями значень двох змінних.

Приклад 9.15. Заміна місцями значень двох змінних.

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

void zamina (int *x, int *y)
{
  int t;
  t = *x;
  *x = *y;
  *y = t;
}

int main ()
{
  int a, b;
  a = 3; b = 7;
  printf ("a =% d b =% d", a, b);
  zamina (&a,  &b);
  printf ("a =% d b =% d", a, b);
  system ("pause"); return 0;
}
```

Результат роботи програми:

```
a = 3 b = 7
a = 7 b = 3
```

Так як покажчики **x** та **y** містять адреси змінних **a** та **b**, то ***x** і ***y** забезпечують опосередкований доступ до значень змінних **a** та **b**.

Приклад 9.16. Переведення масиву значень з сантиметрів у дюйми.

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

const int SIZE = 5;

void toCentimeters(double *);
```

```
int main()
{
  setlocale (LC_ALL, "Ukrainian");
  double inchesArr[SIZE] = {5.0, 11.3, 33.1, 98.1, 15.5};

  toCentimeters( inchesArr );
  int j;
  for (j = 0; j < SIZE; j++)
    printf ("%.3f сантиметрів\n", inchesArr[j]);
  system ("pause"); return 0;
}

void toCentimeters(double *ptrArr)
{
  int j;
  for (j = 0; j < SIZE; j++)
  {
    *ptrArr *= 2.54;
    ptrArr++;
  }
}
```

Розглянемо передавання аргументів функції через покажчик при використанні рядкового масиву на прикладі функції копіювання рядків та функції визначення довжини рядка.

Приклад 9.17. Копіювання рядків.

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

void copyStr(char *dest, const char* src)
{
  while (* src)
    *dest++ = *src++;
  *dest = '\0';
 }
int main()
{
  char *str1 = "Hello World!";
  char *str2;
  copyStr( str2, str1 );
  printf ("%s", str2);
  system ("pause"); return 0;
}
```

Результат роботи програми:

```
Hello World!
```

В даному прикладі функція **main()** викликає функцію **copyStr()**, яка копіює **str1** в **str2**.

Значення **src** розміщується за адресою, на яку вказує **dest**. Потім обидва покажчики збільшуються і на наступній ітерації передається наступний символ. Цикл закінчується, коли в **src** буде знайдений символ кінця рядка, в цьому місці **dest** присвоюється значення **'\0'**.

Приклад 9.18. Визначення довжини рядка.

```
#include <stdio.h>
#include <stdlib.h>
int length (char * str)
{
  char * pot = str;
  while (* pot != '\0')
    pot++;
  return (pot - str);
}

int main()
{
  printf("%d",length("Revolution"));

  system ("pause"); return 0;
}
```

Результат роботи програми:

```
10
```

Розглянемо приклад 9.8., спочатку через параметр у функцію передається покажчик **str** на перший елемент масиву типу **char**. Визначається допоміжний покажчик **pot** та ініціалізується значенням покажчика **str**. В циклі здійснюється інкрементація покажчика **pot**, поки не буде виявлено символ кінця рядка **'\0'**. Функція повертає різницю покажчиків **pot** та **str**, яка і буде визначати довжину рядка.

9.7 Масиви покажчиків на рядки

Масиви рядків є двовимірними масивами типу **char**. На відміну від масивів рядків масиви покажчиків на рядки використовують стільки пам'яті, скільки потрібно для зберігання рядків, що задані при ініціалізації. Як приклад розглянемо масив **partWorld**, який може знадобитися для опису частин світу.

```
char *partWorld [6] = {"Asia", "Africa", "America",
                       "Europe", "Australia", "Antarctic"};
```

Вираз **partWorld [6]** визначає масив з шести елементів. Специфікація **char *** визначає, що елементами масиву є покажчики на тип **char**. На рис. 9.7 наведене наглядне подання масиву **partWorld**.

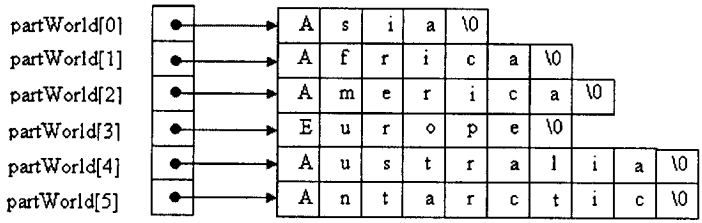


Рисунок 9.7 – Наглядне подання масиву **partWorld**

Таким чином хоча масив **partWorld** має фіксований розмір, він може зберігати символьні рядки довільної довжини.

9.8 Покажчики на функцію

Покажчик на функцію – це змінна, що містить адресу пам'яті, у якій розташована функція. Аналогічно до масиву, в якому ім'я масиву є адресою першого елемента масиву, ім'я функції – це адреса початку програмного коду функції.

Покажчики на функцію можуть:

- бути передані функціям як аргументи;
- повертатися функціями;
- зберігатися у масивах;
- присвоюватися іншим покажчикам на функцію.

У програмі в мові С адресою функції слугує її ім'я без дужок і аргументів. Розглянемо наступну програму, в якій порівнюються два рядки, введені користувачем. Зверніть увагу на оголошення функції **check()** і покажчик **p** всередині **main()**. Покажчик **p** є покажчиком на функцію.

Приклад 9.19. Порівняння рядків з використанням покажчиків на функцію.

```
#include <stdio.h>
#include <string.h>
#include <stdlib.h>
#include <locale.h>

void check (char *a, char * b, int (*cmp) (const char *,
const char *));

int main (void)
{
  setlocale (LC_ALL, "Ukrainian");
  char s1 [20], s2 [20];
  int (*p) (const char *, const char *); /* Покажчик на
                                            функцію */
  p = strcmp;     /* Присвоювання адреси функції
```

```
                        strcmp покажчику p */
  printf ("Введіть два рядка:\n");
  scanf("%s",s1);
  scanf("%s",s2);
  check (s1, s2, p);  /* Передача адреси функції strcmp за
                         допомогою покажчика p */
  system("pause"); return 0;
}

void check (char * a, char * b, int (* cmp) (const char *,
            const char *))
{
  printf ("Перевірка рядків... \n");
  if (!(*cmp) (a, b))
    printf ("Рядки однакові. \n");
  else
    printf ("Рядки не однакові. \n");
}
```

Проаналізуємо цю програму докладно. У першу чергу розглянемо оголошення покажчика **p** в **main()**:

```
int (* p) (const char *, const char *);
```

Це оголошення повідомляє компілятору, що **p** – це покажчик на функцію, що має два параметри типу **const char *** і повертає значення типу **int**. Дужки навколо **p** необхідні для правильної інтерпретації оголошення компілятором. Подібна форма оголошення використовується також для покажчиків на будь-які інші функції, потрібно лише внести зміни залежно від типу, що повертається, і параметрів функції.

Тепер розглянемо функцію **check()**. У ній оголошені три параметри: два покажчика на символьний тип (**a** і **b**) та покажчик на функцію **cmp**. Зверніть увагу на те, що покажчик на функцію **cmp** оголошено в тому ж форматі, що і **p**. Тому в **cmp** можна зберігати значення покажчика на функцію, що має два параметри типу **const char*** і повертає значення **int**. Як і в оголошенні **p**, круглі дужки навколо ***cmp** необхідні для правильної інтерпретації цього оголошення компілятором.

Спочатку в програмі покажчику **p** присвоюється адреса стандартної бібліотечної функції **strcmp()**, яка порівнює рядки. Потім програма просить користувача ввести два рядки і передає покажчики на них функції **check()**, яка їх порівнює. Всередині функції **check()** вираз **(*cmp) (a, b)** викликає функцію **strcmp()**, на яку вказує **cmp**, з аргументами **a** і **b**. Дужки навколо ***cmp** обов'язкові. Існує й інший, більш простий, спосіб виклику функції за допомогою покажчика:

```
cmp (a, b);
```

Однак перший спосіб використовується частіше (рекомендується використовувати саме його), тому що при другому способі виклику покажчик **cmp** дуже схожий на ім'я функції, що може збити з пантелику тих, хто читає програму. В той же час у першого способу запису є свої переваги, наприклад, добре видно, що функція викликається за допомогою покажчика на функцію, а не імені функції.

Виклик функції **check()** можна записати, використовуючи безпосередньо ім'я **strcmp()**:

```
check (s1, s2, strcmp);
```

В цьому випадку вводити в програму додатковий покажчик **p** немає необхідності.

9.9 Покажчики на void

Покажчик на void – це покажчик, що може вказувати на будь-який тип даних. Такі покажчики призначені для використання в певних випадках, наприклад при передаванні покажчика у функцію, яка працює незалежно від типу даних, на який вказує покажчик. Водночас покажчик на **void** має великі обмеження: дані, на які він вказує, не можуть безпосередньо розіменовуватись (за відсутності типу), з цієї причини потрібно вказувати тип перед тим як розіменувати дані, на які вказує цей покажчик.

Приклад 9.20. Покажчик на **void**.

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

void increase( void* data, int psize )
{
  setlocale( LC_ALL, "Ukrainian" );
  if ( psize == sizeof(char) )
  {
    char* pchar;
    pchar = (char *) data;
    ++(* pchar);
  }
  else
    if ( psize == sizeof(int) )
    {
      int* pint;
      pint = (int *) data;
      ++(* pint);
    }
}
```

```
int main()
{
  char a = 'x';
  int b = 1602;
  increase ( &a, sizeof(a) );
  increase ( &b, sizeof(b) );
  printf( "a = %c \nb = %i \n", a, b );
  system ( "pause" ); return 0;
}
```

Результати роботи програми:

```
a = y
b = 1603
```

В функцію **increase()** передаються по черзі адреси змінної **a** (типу **char**) та змінної **b** (типу **int**). Функція приймає покажчик на **void**, тому неважливо, що було передано з основної програми. Але для роботи з переданими даними, необхідно розіменувати покажчик **data**. Не знаючи тип, що передається, виникає проблема. Це вирішується за допомогою конструкції **if-else** та функції **sizeof()**, яка дозволяє визначити розмір пам'яті, що виділяється під змінну певного типу.

9.10 Покажчики на покажчики

В мові С дозволяється створювати покажчики на покажчики, що вказують на дані чи інші покажчики. Для цього потрібно додавати при оголошенні покажчика на покажчик ще один оператор (*).

Приклад 9.21. Створення покажчика на покажчик (рис. 9.8).

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

int main()
{
  int x;
  int *y;
  int **z;
  x = 2014;
  y = &x;
  z = &y;

  printf("x =   %i &x = %p\n", x, &x);
  printf("*y =  %i &y = %p y = %p\n", *y, &y, y);
  printf("**z = %i &z = %p z = %p *z = %p", **z, &z, z,*z);

  system ( "pause" ); return 0;
}
```

Результати роботи програми:

```
x = 2014 &x = 0022FF44
*y = 2014 &y = 0022FF40 y = 0022FF44
**z = 2014 &z = 0022FF3C z = 0022FF40 *z = 0022FF44
```

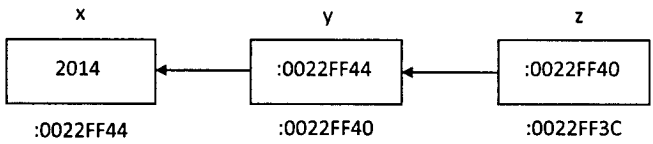


Рисунок 9.8 – Покажчик на покажчик

Контрольні питання

1. Що таке «покажчик»?
2. У чому відмінність між безпосередньою та опосередкованою адресацією?
3. Наведіть ряд задач для яких покажчики є незамінними.
4. Наведіть синтаксис оголошення покажчика.
5. Наведіть синтаксис ініціалізації покажчика.
6. Чим можуть бути ініціалізовані покажчики?
7. Що є результатом застосування операції взяття адреси (`&`)?
8. Що таке розіменування з використанням унарної операції (`*`)?
9. Для чого використовується кваліфікатор const?
10. Наведіть чотири способи передавання покажчика у функцію.
11. Що таке неконстантний покажчик на неконстантні дані?
12. Що таке неконстантний покажчик на константні дані?
13. Що таке константний покажчик на константні дані?
14. Що таке константний покажчик на неконстантні дані?
15. У чому полягає зв'язок між покажчиками і масивами?
16. Наведіть чотири способи звернення до елемента масиву.
17. Наведіть обмежений набір арифметичних операцій, які можуть бути застосовані до покажчиків.
18. Особливості зміщення покажчика для різних типів даних.
19. У чому різниця при передаванні аргументів у функцію за значенням і через покажчик?
20. Особливості застосування масивів покажчиків на рядки.
21. Що таке покажчик на функцію?
22. Наведіть ряд дій дозволених з покажчиками на функцію.
23. Що таке покажчик на `void`?
24. Які існують обмеження для покажчика на `void`?
25. Особливості застосування покажчиків на покажчик.

Контрольні завдання

1. Виконайте такі дії:

   а) здійсніть оголошення масиву **mas** типу **int** з **10** елементів і присвойте їм значення від 0 до 9;

   б) здійсніть оголошення покажчика **masPtr** на тип **int**;

   в) виведіть на екран елементи масиву **mas**, використовуючи звернення за індексом;

   г) присвойте покажчику **masPtr** адресу початку масиву **mas** двома способами;

   д) виведіть на екран елементи масиву **mas**, використовуючи звернення покажчик/зміщення, при цьому як покажчик використайте **masPtr**;

   е) виведіть на екран елементи масиву **mas**, використовуючи звернення покажчик/зміщення, при цьому як покажчик використайте ім'я масиву;

   ж) виведіть на екран елементи масиву **mas**, використовуючи індексацію покажчика **masPtr**;

   и) зверніться до п'ятого елемента масиву, використавши всі чотири способи доступу до елементів масиву: ім'я масиву/індекс, ім'я масиву/зміщення, покажчик/зміщення, покажчик/індекс.

2. Напишіть заголовок та прототип:

   а) функції **swap**, яка має два параметри – покажчики на змінні цілого типу та не повертає значень;

   б) функції **kvadr**, яка має два параметри – покажчики дійсного типу та не повертає значень;

   в) функції **func**, яка має один параметр – покажчик на **double** та повертає покажчик на **double**;

   г) функції **func1**, яка має два параметри – ціле число та покажчик на функцію **func2** та повертає цілий тип.

3. Напишіть заголовок та прототип:

   а) функції **func1**, що має параметр – масив цілих чисел **masLong** типу **long** і не повертає значень;

   б) функції **func2**, що має параметр – масив цілих чисел **masInt** і повертає ціле число.

10 РОБОТА З ФАЙЛАМИ

Зберігання даних в змінних і масивах є тимчасовими, всі ці дані втрачаються при закінченні роботи програми. Для постійного зберігання великих об'ємів даних використовуються файли.

В даному розділі розглядаються такі функції стандартної бібліотеки С, які пов'язані з роботою з файлами і визначені у заголовному файлі **<stdio.h>**:

- **fopen()** – відкриття файлу;
- **fclose()** – закриття файлу;
- **fprintf()** – запис даних у файл;
- **fscanf()** – зчитування даних з файлу;
- **feof()** – визначення маркера кінця файлу.

10.1 Відкриття та закриття файлу

Мова С розглядає будь-який файл як послідовний потік байтів (рис. 10.1). Кожен файл закінчується маркером кінця файлу. Відкритий файл повертає покажчик на структуру **FILE** (визначену в **<stdio.h>**), яка містить інформацію, необхідну при роботі з файлом.

В мові С непередбачено можливості задання структури файлу. Саме тому програміст повинен сам передбачити структуру файлу, яка буде відповідати вимогам конкретного додатка.

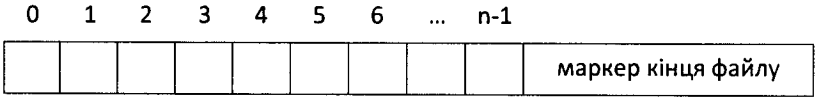


Рисунок 10.1 – Вигляд файлу з n-байтами з точки зору мови С

Для того, щоб відкрити файл потрібно визначити файлову змінну, яка є покажчиком на структуру **FILE**. Синтаксис визначення файлової змінної такий:

*FILE * <файлова змінна>;*

де *FILE* – структура визначена в заголовному файлі **<stdio.h>**;
файлова змінна – ідентифікатор.

Приклад 10.1. Оголошення файлової змінної.

```
FILE * file;
```

Відкриття файлу здійснюється шляхом виклику функції **fopen()**.
Синтаксис виклику функції відкриття файлу **fopen()**:

```
fopen("файл", "режим відкриття");
```

де *файл* – ім'я файлу або шлях до файлу;
режим відкриття – визначає один з можливих режимів відкриття файлу (табл. 10.1).

Таблиця 10.1 – Режими відкриття файлу

Режим	Призначення
r	Відкриття файлу для зчитування
w	Створення файлу для запису. Якщо файл уже існує, його вміст видаляється
a	Дозапис: відкриття чи створення файлу для запису в кінець файлу
r+	Відкриття файлу для оновлення (зчитування, запис)
w+	Створення файлу для оновлення. Якщо файл уже існує, його вміст видаляється
a+	Дозапис: відкриття чи створення файлу для оновлення: запис здійснюється в кінець файлу

Приклад 10.2. Відкриття файлу функцією **fopen()**.

```
FILE * f1, *f2, *f3;
f1 = fopen("in.txt", "r");  /* відкриття для зчитування */
f2 = fopen("out.txt", "w"); /* відкриття для запису */
f3 = fopen("d:\out2.txt", "a");/* відкриття для дозапису */
```

Якщо при відкритті файлу відбувається помилка, функція **fopen()** повертає **NULL**. Для перевірки коректності відкриття файлу, як правило, використовують оператор **if**.

Приклад 10.3. Перевірка коректності відкриття файлу.

```
FILE *f;
if( (f = fopen ("in.txt", "r"))  ==  NULL )
{
  printf("Помилка! Не вдалось відкрити файл!\n");
}
```

Функція **fclose()** призначена для закриття файлу. Якщо функція **fclose()** не викликається явно, операційна система, як правило, сама закриває файли при закінченні роботи програми.

Синтаксис виклику функції закриття файлу:

```
fclose(<файлова змінна>);
```

де *файлова змінна* – ідентифікатор, що визначає файлову змінну.

Приклад 10.4. Закриття відкритого файлу.

```
FILE *f = fopen ("in.txt", "r");
fclose(f); /* закриття файлу in.txt */
```

10.2 Запис та зчитування даних з файлу

Функція **fprintf()** призначена для запису даних у файл.

Синтаксис виклику функції **fprintf()** такий:

```
fprintf(<файлова змінна>,"рядок форматування", змінні);
```

де *файлова змінна* – ідентифікатор, що визначає файлову змінну;

рядок форматування – визначає форматування при виведенні у файл;

змінні – одна чи декілька змінних, значення яких буде виводитися у файл.

Приклад 10.5. Запис даних у файл при використанні **fprintf()**.

```
fprintf(f, "A little less conversation");
fprintf(f, "a = %i", value);
```

Приклад 10.6. Запис у файл суми чисел масиву.

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

int main()
{
  int mas[5] = {0, 1, 2, 3, 4};
  int sum = 0, i;

  FILE *file;
  file = fopen("out.txt", "w");

  for(i = 0; i < 5; i++)
    sum += mas[i];

  fprintf(file, "%i", sum );
```

```
    system("pause");return 0;
}
```

Функція **fscanf()** призначена для зчитування даних з файлу.

Синтаксис виклику функції **fscanf()** такий:

fscanf(<файлова змінна>,"рядок форматування",&змінна1,…);

де *файлова_змінна* – ідентифікатор, що визначає файлову змінну;

рядок форматування – визначає форматування при зчитуванні у файл;

змінна1, … – одна чи декілька змінних, в які будуть зчитуватися дані з файлу.

Приклад 10.7. Зчитування даних з файлу при використанні **fscanf().**

```
fscanf(f, "%d", &iValue); /* зчитування цілого значення */
fscanf(f, "%s", sValue);  /* зчитування рядка */
```

Приклад 10.8. Обчислення суми чисел (1, 2, 3, 4, 5), що зберігаються у файлі.

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

int main()
{
  int mas[5];
  int sum = 0, i;

  FILE *file;
  file = fopen("in.txt", "r");

  for(i = 0; i < 5; i++)
  {
    fscanf(file, "%i", &mas[i]);
    sum += mas[i];
  }

  printf("sum = %i", sum );

  system("pause");return 0;
}
```

Результати роботи програми:

```
sum = 15
```

Функція **feof()** призначена для того, щоб перевірити чи встановлений індикатор кінця файлу, який повідомляє програму про закінчення зчитування даних. Функція повертає ненульове значення, якщо індикатор кін-

ця файлу встановлено, в протилежному випадку повертається нуль. Аргумент, що передається у функцію **feof()** – це покажчик на файл, для якого потрібно перевірити індикатор кінця файлу (наприклад, файлова змінна). Як правило, функцію **feof()** використовують в умовному виразі оператора циклу **while**.

Приклад 10.9. Використання функції **feof()** в умовному виразі оператора **while**.

```
while( !feof( file ) )
```

В цьому прикладі **while**, що містить в собі виклик **feof()**, буде виконуватися доти, поки не буде встановлений індикатор кінця файлу.

Приклад 10.10. Програма, яка зчитує з файлу **input.txt** дані, і перевіряє, скільки разів повторюється перший символ. У вихідний файл **output.txt** записує цей символ. Потім знову відкриває файл **output.txt** і дописує кількість повторів першого символа.

```
#include <stdlib.h>
#include <stdio.h>
#include <string.h>
#include <locale.h>

int main()
{
  setlocale (LC_ALL, "Ukrainian");
  char s, s1;
  int i = 0;
  FILE *fin = fopen("input.txt","rt");
  if(fin == NULL)
  {
    printf("Файл не знайдено!");
    system("pause"); return 0;
  }

  fscanf(fin, "%c", &s1); /* зчитування першого символа */

  while ( !feof(fin) )
  {
    fscanf( fin,"%c", &s );
    if ( s==s1 )
      i++;
  }
  fclose(fin);
  FILE *fout = fopen( "output.txt","w" );
  fprintf(fout, "%i\n", i);
  fclose(fout);
  FILE * fout = fopen( "output.txt","a" );
```

```
    fprintf(fout, "%c", s1);
    fclose(fout);

    system("PAUSE"); return 0;
}
```

Результати роботи програми наведені на рис. 10.2.

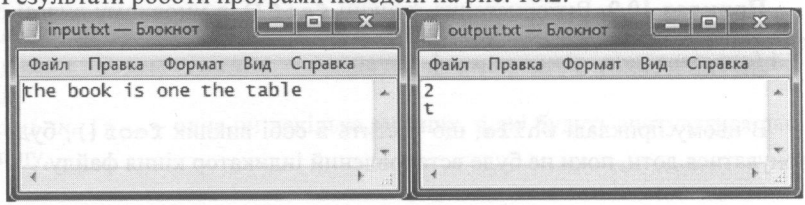


Рисунок 10.2 – Результати роботи програми з прикладу 10.10.

Контрольні питання

1. Як виглядає файл з n-байтами з точки зору мови С?
2. Наведіть синтаксис визначення файлової змінної.
3. Наведіть синтаксис виклику функції `fopen()`.
4. Які існують режими відкриття файлу?
5. Яким чином здійснюється перевірка коректності відкриття файлу?
6. Для чого призначена функція закриття файлу?
7. Наведіть синтаксис виклику функції закриття файлу `fclose()`.
8. Наведіть синтаксис виклику функції `fprintf()`.
9. Наведіть синтаксис виклику функції `fscanf()`.
10. Для чого призначена функція `feof()`?

Контрольні завдання

1. Напишіть програму, яка зчитуватиме двовимірний масив 5×5 цілих чисел з файлу `"in.txt"` та виводитиме їх на екран.
2. Напишіть програму, яка виводить у файл `"out.txt"` таблицю квадратів цілих чисел від 1 до 10.
3. Напишіть програму, яка виводить у файл `"out.txt"` десяткові цілі числа в діапазоні від 1 до 100 та відповідні їм числа у двійковому поданні у вигляді таблиці.
4. Напишіть програму, яка виводить у файл `"out.txt"` десяткові цілі числа в діапазоні від 1 до 100 та відповідні їм числа у вісімковому поданні у вигляді таблиці.
5. Напишіть програму, яка виводить у файл `"out.txt"` десяткові цілі числа в діапазоні від 1 до 100 та відповідні їм числа у шістнадцятковому поданні у вигляді таблиці.
6. Напишіть програму, яка виводитиме 10 довільних слів у файл `"out.txt"`, кожне слово з нового рядка.

7. Напишіть програму, яка зчитуватиме з файлу "in.txt" слово, перевірятиме, чи є воно у файлі "out.txt", і, у разі відсутності, додаватиме його у кінець файлу "out.txt".
8. Напишіть програму, яка зчитуватиме з файлу "in.txt" *N* цілих чисел, сортуватиме їх за методом «бульбашки» та виводитиме відсортовані числа в файл "out.txt". Кількість *N* задається першим числом у файлі.
9. Напишіть програму, яка зчитує з текстового файлу "in.txt" текстові дані і виводить на екран кількість повторів кожного з 256 символів коду ASCII, що зустрічаються в цьому файлі.
10. Напишіть програму, яка зчитує з файлу "in.txt" слова і створює рейтинг слів, які найчастіше зустрічаються. Результат вивести в файл "out.txt". Як текстові дані у файлі "in.txt" використати один з творів Тараса Шевченка.
11. Напишіть програму, яка зчитує з текстового файлу "in.txt" дані і виводить на екран всі рядки файлу в оберненому порядку.
12. Напишіть програму, яка дає змогу додавати, редагувати, видаляти, шукати слова у файлі. Для програми написати консольне меню користувача.
13. Напишіть програму, яка вирівнює текст файлу по правому краю, додаючи в текст кожного непустого рядка потрібну кількість пробілів (шириною тексту вважати 50 символів).
14. Напишіть програму, яка генерує зображення прямокутника, сторони якого зображаються символом, що введений з клавіатури. Розмір квадрата задати рендомно в діапазоні від 2 до 25. Зображення прямокутника вивести у файл "out.txt".
15. Напишіть програму, яка зсуває всі символи текстового файлу "in.txt" на декілька знаків по таблиці ASCII кодів і записує результат у файл "out.txt". Кількість знаків, на які потрібно зсунути, задати з клавіатури.
16. Напишіть програму, яка виводить в файл "out.txt" таблицю всіх ASCII кодів та відповідних їм символів.
17. Напишіть програму, яка всі символи текстового файлу "in.txt" виводить в зворотному порядку в файл "out.txt".
18. Напишіть програму, яка всі слова текстового файлу "in.txt" виводить в зворотному порядку в файл "out.txt".
19. Напишіть програму, яка всі парні слова з текстового файлу "in.txt" виводить в текстовий файл "out1.txt", а непарні – в файл "out2.txt".

11.1 Структури

Структура – це похідний структурований тип даних, який є сукупністю логічно пов'язаних змінних, об'єднанних під одним ім'ям. На відміну від масивів, які можуть містити лише елементи одного типу, структури можуть складатися зі змінних різних типів даних.

Структури часто використовуються для того, щоб визначити записи, які мають зберігатися у файлах. Покажчики і структури можуть слугувати основою для створення більш складніших структур даних, таких як: зв'язні списки, черги, стеки та дерева.

Синтаксис визначення структури:

```
struct <ім'я структури>
{
<опис елементів>
}<список змінних>;
```

Ключове слово `struct` визначає структуру. *Ім'я структури* – це ідентифікатор, що визначає ім'я створюваного структурованого типу. В блоці *опис елементів* здійснюється оголошення елементів структури. Елементи структури називають *полями.* Поля можуть бути будь-якого базового чи похідного типу, наприклад, масивом, покажчиком, об'єднанням або іншою структурою. У *списку змінних* здійснюється визначення змінних типу структури.

Список змінних може бути порожнім. У цьому випадку при визначенні структури не резервується місце під елементи структури в пам'яті комп'ютера. Здійснюється лише створення нового типу даних, який можна використовувати для визначення змінних типу структури. Змінні типу структури називають *об'єктами* структури.

Приклад 11.1. Визначення структури без списку змінних.

```
struct DATE
{
  int day, month, year;
};
```

Приклад 11.2. Визначення структури зі списком змінних.

```
struct DATE
{
  int day, month, year;
}date1, date2;
```

Визначення структури може не містити імені. Такі структури називають *анонімними*. При такому визначенні обов'язково потрібно заповнити *список змінних*.

Приклад 11.3. Визначення анонімної структури.

```
struct
{
  int day, month, year;
}date1;
```

Синтаксис оголошення змінної типу структури такий:

struct *<ім'я структури> <ім'я змінної>;*

де *ім'я змінної* – це ідентифікатор, який визначає ім'я змінної типу структури (об'єкт).

Приклад 11.4. Оголошення змінних типу структури.

```
struct DATE date1, date2, date3;
struct DATE date[10];
struct DATE *datePtr;
```

Для звернення до полів структури використовується дві операції:

- операція *«крапка»* (.);
- операція *«стрілка»* (->).

Операція «крапка» використовується при зверненні до поля структури через об'єкт структури:

Об'єкт_структури.поле_структури

Операція «стрілка», що складається зі знака мінус (-) та знака більше (>) без пробілу між ними, використовується при зверненні до поля структури через покажчик на структуру:

Покажчик_на_структуру->поле_структури

Приклад 11.5. Звернення до елементів структури.

```
struct DATE date;
date.day = 10;              /* ініціалізація поля day через
                               об'єкт date */
struct DATE *datePtr = &date;
datePtr->year = 2014;       /* ініціалізація поля year через
                               покажчик datePtr */
```

Введення/виведення даних у структуру виконується поелементно.

Приклад 11.6. Введення даних в поле **day** об'єкта **date** структури **DATE**.

```
scanf( "%i", &date.day );
```

Ініціалізацію полів структури можна здійснити при визначенні об'єкта структури.

Приклад 11.7. Ініціалізація полів об'єкта структури **DATE**.

```
struct DATE date = {1, 2, 3};
```

Приклад 11.8. Ініціалізація масиву об'єктів структури **DATE**.

```
struct DATE date[3] = {{1,3,1980},{5,1,1990},{1,1,2002}};
```

Задача 11.1. Програма, яка зчитує дані про успішність студентів (оцінки з трьох предметів) з файлу, і виводить на екран інформацію про середній бал кожного студента. Приклад даних, поданих у файлі, наведений у таблиці 11.1.

Таблиця 11.1 – Дані про успішність студентів

Прізвище	Філіпчук			Олєйнікова			Білоус			Подольський		
Група	1CI-10			1CI-10			1CI-10			1CI-10		
Оцінки	5	5	4	5	5	5	4	4	4	5	4	4

```
#include <stdio.h>
#include <string.h>
#include <stdlib.h>
#include <locale.h>

const int numMarks = 3;
struct STUDENT
{
  char studentName[20];      /* прізвище студента */
  char groupName[10];        /* назва групи */
  int masMarks[ numMarks ]; /* масив оцінок */
  float averageMark;         /* середній бал */
  };

int initMasObj(int *numStudents, STUDENT *studentPtr)
{
  FILE *file;
  if( (file = fopen("in2.txt", "r"))  ==  NULL )
  {
    printf("Помилка! Не вдалося відкрити файл!\n");
    return 0;
  }

  fscanf( file, "%i", &(*numStudents) );
  for(int i = 0; i < *numStudents; i++)
```

```
  {
    fscanf(file, "%s", (studentPtr + i)->studentName);
    fscanf(file, "%s", (studentPtr + i)->groupName);

    float sumMarks = 0;

    for(int j = 0; j < numMarks; j++)
    {
      fscanf(file, "%i", &(studentPtr + i)->masMarks[j]);
      sumMarks += (studentPtr + i)->masMarks[j];
    }
    (studentPtr + i)->averageMark = sumMarks / 3;
  }
}

int main(void)
{
  setlocale (LC_ALL, "Ukrainian");
  struct STUDENT masObj[50];
  int numStudents;

  initMasObj(&numStudents, &masObj[0]);

  printf("Інформація про студентів: \n");

  for( int i = 0 ; i < numStudents ; i++ )
    printf("%s %s %.1f \n", masObj[i].studentName,
           masObj[i].groupName, masObj[i].averageMark);

  system("PAUSE"); return 0;
}
```

Результат роботи програми:

```
Інформація про студентів:
Філіпчук 2АВ-10 4,7
Олейнікова 2АВ-10 5,0
Білоус 2АВ-10 4,0
Подольський 2АВ-10 4,3
```

11.2 Ключове слово typedef

Ключове слово **typedef** дає програмісту механізм для створення синонімів (чи псевдонімів) для раніше визначених типів. Використання **typedef** дозволяє замінювати громіздкі послідовності імен типів у визначеннях новими іменами.

Синтаксис оголошення **typedef**:

typedef *<ім'я існуючого типу> <ім'я синоніму>;*

Часто використовують **typedef**, щоб дати скорочене ім'я типу структури.

Приклад 11.9. Створення синоніма структурного типу.

```
typedef struct STUDENT stud;
```

В попередньому прикладі **stud** визначений як синонім **struct STUDENT**, тому створюючи об'єкт структури, потрібно замість **struct STUDENT obj** написати **stud obj**.

Іноді **typedef** використовують для визначення типу структури, тоді зникає необхідність у використанні імені структури.

Приклад 11.10. Створення синоніма структурного типу без імені структури.

```
typedef struct
{
  char studentName[20];
  char groupName[10];
} stud;
```

В попередньому прикладі створюється тип **stud**, для створення об'єкта структури потрібно написати **stud obj**.

Приклад 11.11. Оголошення **typedef**.

```
typedef unsigned long int ULINT;
ULINT b;              /* еквівалентно unsigned long int b; */
typedef char STRING[55];
STRING str[10];       /* еквівалентно char str[10][55]) */
```

Синонімічні типи, введені за допомогою **typedef**, подібно до змінних можуть мати обмежену область видимості або бути глобальними.

Приклад 11.12. Області видимості типів, введених за допомогою **typedef**.

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

typedef int INT;     /* глобальний тип INT */
INT function1(void)
{
  static INT count = 0;
  return ( ++count );
}

int function2(void)
{
```

```
  typedef int INTEGER; /* локальний тип INTEGER */
  INTEGER c;
  while(( c = function1() ) < 250 )
  ;
  return c;
}
int main(void)
{
  setlocale (LC_ALL, "Ukrainian");
  printf("Число викликів функції function1(): %d\n",
                                          function2());
  system("PAUSE"); return 0;
}
```

Результат роботи програми:

```
Число викликів функції function1(): 250
```

Програма, наведена у прикладі 11.12. ілюструє обмеженість області видимості типу **INTEGER**, який визначено локально в межах функції **function2()**, його використання поза межами цієї функції призведе до помилки на етапі компіляції.

11.3 Об'єднання

Об'єднання (**union**) – похідний структурований тип даних, елементи якого розділяють одну й ту ж область пам'яті. Елементами об'єднання можуть бути змінні будь-якого типу даних.

Синтаксис визначення об'єднання:

```
union <ім'я об'єднання>
{
<опис елементів>
}<список змінних>;
```

Об'єднання використовуються, коли необхідно отримати доступ до одних і тих же даних різними способами. Оскільки для зберігання усіх елементів об'єднання використовується спільна область пам'яті, розмір змінної типу об'єднання визначається розміром найбільшого елемента, об'явленого в даному об'єднанні.

Над об'єднаннями можна виконувати такі операції:

- присвоювання об'єднання іншому об'єднанню того ж типу;
- взяття адреси;
- доступ до елементів об'єднання за допомогою операцій «крапка» чи «стрілка».

Синтаксис оголошення змінної типу об'єднання такий:

union *<ім'я об'єднання> <ім'я змінної>;*

Приклад 11.13. Порівняння розміру пам'яті, необхідної для збереження змінних типу об'єднання та типу структури.

```
#include <stdio.h>
#include <stdlib.h>
#include <locale.h>
union {
  char charValue;      /* 1 байт */
  int intValue;        /* 4 байти */
  double doubleValue; /* 8 байт */
}UNION;

struct {
  char charValue;      /* 1 байт */
  int intValue;        /* 4 байти */
  double doubleValue; /* 8 байт */
}STRUCT;

int main(void)
{
  setlocale (LC_ALL, "Ukrainian");
  printf("Розмір пам'яті в байтах:\n");
  printf("Змінна UNION: %d\n", sizeof(UNION));
  printf("Змінна STRUCT: %d\n", sizeof(STRUCT));
  system("pause"); return 0;
}
```

Результат роботи програми:

```
Розмір пам'яті в байтах:
Змінна UNION: 8
Змінна STRUCT: 16
```

Як видно з попереднього прикладу, найбільший розмір як у структурі так і у об'єднанні займає елемент **doubleValue** (8 байтів). Розмір цього елемента визначає розмір змінної типу об'єднання. Розмір змінної типу структури визначається як сума розмірів його елементів.

Приклад 11.14. Приклад організації доступу до однієї області пам'яті різними способами за допомогою об'єднань.

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

union dig
{
  struct
  {
```

```
    int digit0, digit1, digit2, digit3, digit4, digit5;
  } digits;
  int digit[6];
};

int main(void)
{
  setlocale (LC_ALL, "Ukrainian");
  union dig data;  /*оголошення змінної типу dig*/
  printf("Введіть 6 чисел: ");
  scanf("%d", &data.digits.digit0);
  scanf("%d", &data.digits.digit1);
  scanf("%d", &data.digits.digit2);
  scanf("%d", &data.digits.digit3);
  scanf("%d", &data.digits.digit4);
  scanf("%d", &data.digits.digit5);

  printf("Ви ввели такі числа:");
  for( int i = 0; i < 6; i++ )
    printf( " %d", data.digit[i] );

  system("PAUSE"); return 0;
}
```

Результат роботи програми:

```
Введіть 6 чисел: 4 8 15 16 23 42
Ви ввели такі числа: 4 8 15 16 23 42
```

В попередньому прикладі елементи об'єднання, а саме об'єкт структури **digits** та масив цілих чисел **digit[]**, займають одну й ту ж область пам'яті. Щоб показати цю властивість об'єднання спочатку з клавіатури ініціалізуються елементи структури (змінні цілого типу), а потім на екран виводиться вміст масиву **digit**.

11.4 Бітові поля

В мові С існує можливість задання кількості бітів для збереження елементів структури чи об'єднання типу **unsigned** чи **int** шляхом визначення бітових полів. Бітові поля дозволяють краще використовувати пам'ять, зберігаючи дані з використанням мінімально необхідної кількості бітів.

Бітові поля корисні по таких причинах:

- якщо обмежено місце для зберігання інформації, можна зберегти декілька логічних змінних в одному байті;
- деякі інтерфейси пристроїв передають інформацію, закодувавши біти в один байт;

- деяким процедурам кодування необхідно отримати доступ до окремих бітів в байті.

Синтаксис визначення бітових полів для структури такий:

```
struct <ім'я структури>
{
  <тип> <ім'я1> : <довжина поля>;
  <тип> <ім'я2> : <довжина поля>;
  <тип> <ім'я3> : <довжина поля>;
}<список_змінних>;
```

де *довжина поля* – це цілий літерал, який визначає кількість бітів, що відводяться для збереження елемента структури.

Приклад 11.15. Визначення бітових полів.

```
struct Date{
  unsigned short nMonthDay : 6 ; /* 1..31  (6 біт) */
  unsigned short nMonth : 5 ;    /* 1..12  (5 біт) */
  unsigned short nYear : 8 ;     /* 1..100 (8 біт) */
}date;
```

Для змінної **nMonthDay** відводиться у пам'яті 6 бітів для збереження даних про число місяця (діапазон – 1...31), для змінної **nMonth** – 5 бітів для збереження даних про номер місяця (діапазон – 1...12), для змінної **nYear** – 8 бітів для збереження даних про рік (діапазон – 1...100).

11.5 Перерахування

Перерахування (**enum**) – скалярний похідний тип даних, що є множиною поіменованих цілих значень.

Синтаксис визначення перерахування:

```
enum <ім'я перерахування>
{
<список перерахування>
} <список змінних>;
```

де *ім'я перерахування* – ідентифікатор, який визначає ім'я типу перерахування;

список перерахування – рядкові літерали розділені комами, які відповідають цілим значенням, що може приймати створюваний тип перерахування;

список змінних – змінні типу перерахування.

За замовчуванням цілим значенням першого рядкового літерала у списку перерахування є 0, значення кожного наступного літерала збільшуєть-

ся з кроком 1. При визначенні перерахування можна задавати значення окремим елементам списку перерахування. Для зберігання значень елементів перерахування використовується тип **int**.

Перерахування зручно використовувати для запобігання появи у коді програми числових літералів, логічне значення яких може бути незрозумілим читачеві. Також перерахування – це більш безпечна заміна макровизначенням **#define**.

Приклад 11.16. Визначення перерахування зі значеннями за замовчуванням.

```
enum DOW
{
  MON, TUE, WED, THU, FRI, SAT, SUN
};
```

В попередньому прикладі літералу **MON** присвоюється значення 0, літералу **THU** – 1, відповідно літералу **SUN** присвоюється значення 6.

Приклад 11.17. Визначення перерахування зі встановленням значень певним елементам списку перерахування.

```
enum type
{
  CHAR = 1, SHORT, FLOAT = 4, INT = 4, DOUBLE = 8
};
```

В попередньому прикладі літералу **SHORT** присвоюється значення 2, інші літерали визначені явно.

Синтаксис оголошення змінної типу перерахування такий:

enum *<ім'я перерахування> <ім'я змінної>;*

де *ім'я змінної* – це ідентифікатор, який визначає ім'я змінної типу перерахування.

Приклад 11.18. Оголошення змінної типу перерахування.

```
enum DOW dayOfWeek;
```

Допустимі операції над перерахуваннями такі:

- можна присвоїти символьний літерал типу **enum** змінній того ж типу;
- можна використовувати операції порівняння (==, !=) над змінними типу **enum**;
- можна застосовувати арифметичні операції щодо символьних літералів типу **enum**.

Недопустимі операції над перерахуваннями такі:

- не можна використовувати арифметичні операції щодо змінних типу **enum**;
- не можна використовувати символьні літерали типу **enum** для індексу масиву;
- не можна використовувати операції інкремента та декремента щодо змінних типу **enum**;
- не можна використовувати операції відношення (<, >, <=, >=) над змінними типу **enum**;
- не можна використовувати складені операції присвоювання (+=, -= тощо) щодо змінних типу **enum**.

Приклад 11.19. Використання перерахування у операторі **switch**.

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

enum
{
  MONDAY = 1, TUESDAY, WEDNESDAY, THURSDAY, FRIDAY,
  SATURDAY, SUNDAY
}dayOfWeek;

int main(void)
{
  setlocale( LC_ALL, "Ukrainian" );
  printf( "Введіть номер дня тижня (1-7): \n" );
  scanf("%i", &dayOfWeek);

  switch (dayOfWeek)
  {
    case MONDAY:    printf( "Понеділок \n" ); break;
    case TUESDAY:   printf( "Вівторок \n" ); break;
    case WEDNESDAY: printf( "Середа \n" ); break;
    case THURSDAY:  printf( "Четвер \n" ); break;
    case FRIDAY:    printf( "П'ятниця \n" ); break;
    case SATURDAY:  printf( "Субота \n" ); break;
    case SUNDAY:    printf( "Неділя \n" ); break;
  }
  system("PAUSE"); return 0;
}
```

Результат роботи програми:

```
Введіть номер дня тижня (1-7): 6
Субота
```

В програмі з попереднього прикладу в змінну типу перерахування (**dayOfWeek**) зчитується значення дня тижня. Обирається та гілка опера-

тора **switch**, значення символьного літерала якої відповідає введеному значенню змінної **dayOfWeek**.

Приклад 11.20. Використання перерахування у операторі **for**.

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

enum DOW
{
  MON = 1, TUE, WED, THU, FRI, SAT, SUN
};

int main(void)
{
  static char *dayOfWeek[] = {"", "Monday", "Tuesday",
                              "Wednesday", "Thursday","Friday",
                              "Saturday", "Sunday"};
  int i;
  for( i = MON; i <= SUN ; i++ )
    printf( "%s %i \n", dayOfWeek[i], i );

  system("PAUSE"); return 0;
}
```

Результат роботи програми:

```
Monday 1
Tuesday 2
Wednesday 3
Thursday 4
Friday 5
Saturday 6
Sunday 7
```

В програмі з попереднього прикладу символьні літерали, визначені в перерахуванні, використовуються в циклі **for** для задання діапазону значень індексів масиву рядків (**dayOfWeek**).

Контрольні питання

1. Що таке «структура»?
2. Наведіть ряд задач, в яких структури найчастіше використовуються.
3. Наведіть синтаксис визначення структури.
4. Що таке поле структури?
5. Що таке об'єкт структури?
6. Що таке анонімна структура?
7. Наведіть синтаксис оголошення змінної типу структури.
8. Для чого використовується операція вибору «крапка» («.»)?
9. Для чого використовується операція непрямого доступу «->»?

10. Як здійснюється введення/виведення даних у елементи структури?
11. Для чого використовується ключове слово **typedef**?
12. Наведіть синтаксис оголошення **typedef**.
13. Що таке об'єднання?
14. Наведіть синтаксис визначення об'єднання.
15. Які операції дозволені над об'єднаннями?
16. Наведіть синтаксис оголошення змінної типу об'єднання.
17. Що таке бітові поля?
18. Наведіть синтаксис визначення бітових полів.
19. Наведіть причини корисності бітових полів.
20. Що таке перерахування?
21. Наведіть синтаксис визначення перерахування.
22. Наведіть синтаксис оголошення змінної типу перерахування.
23. Наведіть допустимі операції над перерахуваннями.
24. Наведіть недопустимі операції над перерахуваннями.

Контрольні завдання

1. Що буде виведено програмою:

```
union Data
{
  struct
  {
    unsigned int param1 :1;
    unsigned int param2 :1;
    unsigned int param3 :1;
    unsigned int param4 :1;
    unsigned int param5 :1;
    unsigned int param6 :1;
    unsigned int param7 :1;
    unsigned int param8 :1;
  };
  unsigned char Byte;
} info;

info.Byte = 0xAB;
printf("%d\n",info.param6);
```

2. Описати структуру з ім'ям STUDENT, що містить такі поля:
   - прізвище та ініціали;
   - номер групи;
   - успішність (масив з п'яти елементів).

   Напишіть програму, що виконує такі дії:
   - введення з клавіатури даних у масив, який складається з десяти структур типу STUDENT;
   - записи повинні бути упорядковані за зростанням номера групи;

- виведення на дисплей прізвищ і номерів груп для всіх студентів, включених у масив, якщо середній бал студента більше 4,0;
- якщо таких студентів немає, вивести відповідне повідомлення.

3. Описати структуру з ім'ям AEROFLOT, яка містить такі поля:
   - назва пункту призначення рейсу;
   - номер рейсу;
   - тип літака.

   Напишіть програму, що виконує такі дії:
   -введення з клавіатури даних у масив, який складається із семи елементів типу AEROFLOT;
   -записи повинні бути розміщені за алфавітом за назвами пунктів призначення;
   -виведеннння на екран пунктів призначення і номерів рейсів, які обслуговуються літаком, тип якого введений із клавіатури;
   -якщо таких рейсів немає, видати на дисплей відповідне повідомлення.
4. Описати структуру з ім'ям TRAIN, яка містить такі поля:
   -назва пункту призначення;
   -номер потяга;
   -час відправлення.

   Напишіть програму, яка виконує такі дії:
   -введення з клавіатури даних у масив, який складається з восьми елементів типу TRAIN;
   -записи повинні бути упорядковані за номерами потягів;
   -виведення на екран інформації про потяг, номер якого введений із клавіатури;
   -якщо таких потягів немає, видати на дисплей відповідне повідомлення.
5. Описати структуру з ім'ям PRICE, яка містить такі поля:
   -назва товару;
   -назва магазину, у якому продається товар;
   -вартість товару в гривнях.

   Напишіть програму, яка виконує такі дії:
   -введення з клавіатури даних у масив, який складається з восьми елементів типу PRICE;
   -записи повинні бути розміщені за алфавітом за назвами магазинів;
   -виведення на екран інформації про товари, які продаються в магазині, назва якого введена з клавіатури;
   -якщо такого магазину немає, видати на дисплей відповідне повідомлення.

12 ДИНАМІЧНЕ ВИДІЛЕННЯ ПАМ'ЯТІ

Динамічне виділення пам'яті – спосіб виділення оперативної пам'яті комп'ютера для об'єктів у програмі (змінна, масив), при якому виділення пам'яті під об'єкт здійснюється під час виконання програми.

В даному розділі розглядаються такі функції стандартної бібліотеки С, які пов'язані з динамічним виділенням пам'яті та визначені у заголовному файлі `<stdlib.h>`:

- `malloc()` та `calloc()` – для виділення пам'яті;
- `realloc()` – для змінення розміру виділеної області пам'яті;
- `free()` – для звільнення виділеної пам'яті.

12.1 Загальні відомості

При динамічному розподілі пам'яті об'єкти розміщуються в так званій «купі» (англ. *heap* – набір «вільних» блоків пам'яті). При створенні об'єкта програми вказується розмір запитуваної під об'єкт пам'яті, і, в разі успіху, виділена область пам'яті, умовно кажучи, «вилучається» з «купи», стаючи недоступною при подальших операціях виділення пам'яті. Протилежна за змістом операція – звільнення зайнятої раніше під який-небудь об'єкт пам'яті. Пам'ять, що звільняється, повертається в «купу» і стає доступною для подальших операцій виділення пам'яті. На рис. 12.1 подано схематичне зображення «купи».

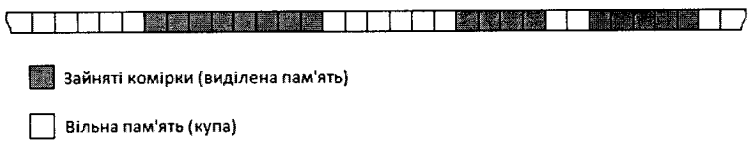


Рисунок 12.1 – Схематичне зображення «купи»

Зі збільшенням кількості створених у програмі об'єктів розмір доступної пам'яті зменшується. В результаті цього, виникає необхідність постійно звільняти раніше виділену пам'ять. Для коректної роботи програма повинна забезпечити звільнення всієї пам'яті, виділеної під час виконання. Некоректний розподіл пам'яті призводить до так званого «витоку» ресурсів, коли виділена пам'ять не звільняється. Численні витоки пам'яті можуть призвести до вичерпання всієї оперативної пам'яті і порушити роботу операційної системи.

Ще однією проблемою, пов'язаною з динамічним виділенням пам'яті – є проблема фрагментації пам'яті. Виділення пам'яті відбувається блоками – безперервними фрагментами оперативної пам'яті (кожен блок – це кілька байтів, що йдуть підряд). У якийсь момент, в купі просто може не виявити-

ся блоку відповідного розміру і, навіть, якщо вільної пам'яті достатньо для розміщення об'єкта, операція виділення пам'яті закінчиться невдачею.

Наприклад, потрібно розмістити у пам'яті масив з 6 блоків за наявності 8 вільних блоків (рис. 12.2).

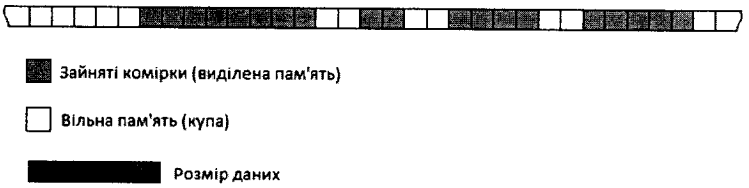


Рисунок 12.2 – Демонстрація виділення пам'яті

Як відомо, масив – це послідовний набір однотипних даних, тому він потребує послідовно розміщених у пам'яті 6 блоків.

Як видно на рис. 12.2, неможливо виділити цю пам'ять, хоча кількість блоків для цього є достатньою, тому операція закінчиться невдачею.

12.2 Функції динамічного виділення пам'яті

Для динамічного виділення пам'яті використовуються функції `malloc()` та `calloc()`, що описані в заголовному файлі `<stdlib.h>`.

Прототип функції `malloc()` такий:

```
void * malloc (size_t <size>);
```

Параметром функції `malloc()` є розмір (у байтах) області пам'яті *size*, яка виділяється. Функція повертає покажчик на початок цієї області пам'яті. Цей покажчик завжди має тип *void*, і може бути приведений до необхідного типу даних при розіменуванні. Якщо функції не вдалося виділити необхідну область пам'яті, повертається покажчик на `NULL`. Виділена область пам'яті неініціалізована, тобто залишається з невизначеними значеннями.

Приклад 12.1. Застосування функції `malloc()`.

```
#include <stdlib.h>
#include <stdio.h>
#include <string.h>
#include <locale.h>

int main ()
{
  setlocale (LC_ALL, "Ukrainian");
  int masSize, i;
```

```
  char *string;
  printf ("Введіть довжину рядка: ");
  scanf ("%i", &masSize);

  string = ( char* ) malloc (masSize + 1);

  if ( string == NULL )
    exit (1);

  for ( i = 0; i < masSize; i++ )
    string[ i ] = rand() % masSize + 'a';

  string[ masSize ]='\0';
  printf ("Рядок: %s\n", string);
  free (string);   /* звільнення пам'яті */

  system("pause"); return 0;
}
```

Результат роботи програми:

```
Введіть довжину рядка: 12
Рядок: flkefeggkiff
```

Програма з попереднього прикладу генерує рядок довжиною, яка зазначена користувачем (змінна **masSize**), і заповнює його випадковими символами. Можлива довжина цього рядка обмежена лише кількістю вільної пам'яті в системі, яку **malloc()** може виділити.

Прототип функції **calloc()** такий:

```
void * calloc (size_t <num>, size_t <size>);
```

Функція **calloc()** виділяє область пам'яті для масиву з *num* елементів, кожен з яких має розмір *size* байт, та ініціалізує всі комірки виділеної пам'яті нулями. В результаті буде виділена пам'ять розміром *size*num* байт. У разі успіху повертається покажчик на область виділеної пам'яті. Цей покажчик завжди має тип *void*, і може бути приведений до необхідного типу даних при розіменуванні. Якщо функції не вдалося виділити необхідну область пам'яті, повертається покажчик на **NULL**.

Приклад 12.2. Застосування функції **calloc()**.

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

int main (void)
{
```

```
    setlocale (LC_ALL, "Ukrainian");
    int masSize, i;
    int * p;

    printf ("Введіть кількість елементів масиву: ");
    scanf ("%d", &masSize);

    p = (int*) calloc (masSize, sizeof(int));

    if (p == NULL)
      exit (1);

    printf ("Введіть %d чисел: ", masSize);

    for ( i = 0; i < masSize ; ++i )
      scanf ("%d", &p[ i ]);

    printf ("Ви ввели: ");
    for ( i = 0; i < masSize ; ++i )
      printf ( "%d ", p[ i ] );
    printf ("\n");

    free (p);      /* Звільнення пам'яті */
    system ("pause"); return 0;
}
```

Результат роботи програми:

```
Введіть кількість елементів масиву: 5
Введіть 5 чисел: 3 14 159 2653 58979
Ви ввели: 3 14 159 2653 58979
```

Потрібно зазначити, що функція **calloc()** потребує для свого виконання більше часу, оскільки проводить ініціалізацію виділеної пам'яті. У випадках, коли є критичною швидкодія програми, доцільно користуватися функцією **malloc()**.

Здійснити заміну функції **calloc()** на **malloc()** можна, переписавши рядок: **p = (int*) calloc (masSize, sizeof(int))** на **p = (int*) malloc (masSize * sizeof(int))**.

Функція **realloc()** дозволяє змінювати розмір області пам'яті раніше виділеної за допомогою функцій **malloc()** або **calloc()**.

Прототип функції **realloc()** виглядає таким чином:

*void * realloc (void * <ptr>, size_t <size>);*

Функція **realloc()** змінює розмір області пам'яті, на яку вказує покажчик *ptr*. Аргумент *size* задає новий розмір області пам'яті у байтах.

Вміст області пам'яті при цьому не змінюється. У разі успіху функція повертає покажчик на область пам'яті, розмір якої був змінений аргументом *size*. Цей покажчик завжди має тип *void*, і може бути приведений до необхідного типу даних при розіменуванні. Якщо не вдалося змінити розмір області пам'яті, функція повертає покажчик на **NULL**.

Приклад 12.3. Застосування функції **realloc()**.

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

int main()
{
  setlocale (LC_ALL, "Ukrainian");
  int *A, i;

  if(( A = (int*) malloc ( 10 * sizeof(int))) == NULL )
    exit(0);

  for( i = 0 ; i < 10 ; i++ )
  {
    A[i] = i;
    printf("%d ", A[i]);
  }
  printf ("\n");

  if(( A = (int*) realloc( A, 29 * sizeof(int))) == NULL )
    exit(0);

  for( ; i < 29 ; i++ )
    A[ i ] = i % 5;

  for( i = 0 ; i < 29 ; i++ )
    printf("%d ", A[i]);

  free (A);        /* Звільнення пам'яті */
  printf ("\n");
  system ("pause"); return 0;
}
```

Результат роботи програми:

```
0 1 2 3 4 5 6 7 8 9
0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 0 1 2 3 4 0 1 2 3 4 0 1 2 3
```

За допомогою функції **malloc()** у програмі виділяється пам'ять для десяти елементів типу **int** та заповнюється створений масив **A** числами від **0** до **9**. Після цього за допомогою функції **realloc()** проводиться розширення масиву **A** до розміру 29-и елементів типу **int**. Доповнення маси-

ву заповнюється остачами від ділення на 5 порядкового номера комірки масиву A, при чому вміст перших десяти елементів залишається незмінним.

При виклику функції **free()** область пам'яті, раніше виділена функціями **malloc()** чи **calloc()** буде звільнена, що робить її доступною для подальшого використання.

Прототип функції **free()** такий:

```
void free (void * <ptr>);
```

Як параметр у функцію **free()** передається покажчик *ptr* на динамічно виділену область пам'яті.

Використання функції **free()** показане в прикладах 12.1. – 12.3.

12.3 Зв'язні списки

Зв'язні списки є однією з форм роботи з великими об'ємами даних у мові С. Для їх реалізації необхідно використовувати структури та динамічне виділення пам'яті. Кожним елементом зв'язного списку є об'єкт структури. Об'єкти структури у зв'язному списку пов'язуються за допомогою поля, що містить покажчик на структуру. Список створюється шляхом динамічного виділення пам'яті під об'єкт структури та подальшого зв'язування з вже існуючими елементами. Особливістю такої організації даних є те, що при виконанні програми завжди задіяно стільки елементів списку, скільки необхідно для вирішення певної задачі.

Задача 12.1. Реалізувати однонаправлений зв'язний список, елементи якого зберігають прізвище. Передбачити функції додавання, видалення та перегляд елементів списку.

```
#include <stdlib.h>
#include <stdio.h>
#include <conio.h>
#include <locale.h>

struct stud
{
  char name[20];
  struct stud *next;
};
struct stud *first = 0;

void add();
void print();
void delete();
```

```
void printAll();
void menu();

int main()
{
  setlocale (LC_ALL, "Ukrainian");
  menu();
  system("pause");
  return 0;
}

void Menu()
{
  char c;
  do
  {
    system("cls");
    printf("1. Додати елемент списку \n");
    printf("2. Переглянути елементи списку по одному \n");
    printf("3. Вивести весь список \n");
    printf("4. Видалити елемент списку \n");
    printf("Для виходу з програми натисніть (esc) \n");
    c = getch();
    if(c == '1')
      add();
    if(c == '2')
      print();
    if(c == '3')
      printAll();
    if(c == '4')
      delete();
  }while(c != 27);
}

void add()
{
  system("cls");
  struct stud *ob = (struct stud *) malloc(sizeof (struct
                                                    stud));
  struct stud *temp = 0;
  printf("Введіть ім'я: ");
  scanf("%s", ob->name);
  ob->next = NULL;
  if(first == 0)
  {
    first = ob;
  }
  else
  {
    temp = first;
    while(temp->next != first)
    {
```

```
      temp = temp->next;
    }
    temp->next = ob;
  }
  ob->next = first;
}

void print()
{
  system("cls");
  int c;
  struct stud *temp = first;
  if(temp != 0)
  {
    printf("%s \n", (*temp).name);
    printf("\n");
    printf("Для переходу до наступного елемента списку
            натисніть клавішу вправо \n");
    printf("Для виходу в попереднє меню натисніть
            клавішу(esc)\n");
    do
    {
      c = getch();
      if(c == 77)
      {
        temp = (*temp).next;
        system("cls");
        printf("%s \n", (*temp).name);
        printf("\n");
        printf("Для переходу до наступного елемента списку
                натисніть клавішу вправо \n");
        printf("Для виходу в попереднє меню натисніть
                клавішу(esc)\n");
      }
    }while(c != 27);
  }
  else
  {
    printf("Список пустий! \nДля виходу в попереднє меню
            натисніть будь-яку клавішу \n");
  getch();
  }
}

void printAll()
{
  system("cls");
  if(first != 0)
  {
    struct stud *temp = first;
    int i = 1;
    printf("%i. %s \n", i, temp->name);
```

```
    while(temp->next != first)
    {
      temp = temp->next;
      i++;
      printf("%i. %s \n", i, temp->name);
    }
  }
  else
    printf("Список пустий! \n");
  printf("Для виходу в попереднє меню натисніть будь-яку
          клавішу \n");
  getch();
}

void delete()
{
  system("cls");
  if(first != 0)
  {
    int k, i = 1, n;
    struct stud *temp = first;
    struct stud *nex;
    struct stud *ant;
    do
    {
      system("cls");
      printf("%i. %s \n", i, temp->name);
      while(temp->next != first)
      {
        temp = temp->next;
        i++;
        printf("%i. %s \n", i, temp->name);
      }
      temp = first;
      n = i;
      i = 1;
      printf("Введіть номер елемента: ");
      scanf("%i", &k);
    }while(k<=0 || k>n);
    if (k != 1)
    {
      while(temp->next != first)
      {
        ant = temp;
        temp = temp->next;
        i++;
        if(i==k)
          break;
      }
      nex = temp->next;
      ant->next = nex;
      free(temp);
```

```
    }
    else
    {
      if(first->next != first)
      {
        while(temp->next != first)
        {
          temp = temp->next;
        }
        ant = first;
        first = first->next;
        temp->next = first;
        free(ant);
      }
      else
        first = 0;
    }
  }
  else
  {
    printf("Список пустий! \nДля виходу в попереднє меню
           натисніть будь-яку клавішу \n");
    getch();
  }
}
```

Результати роботи програми:

```
1. Додати елемент списку.
2. Переглянути елементи списку.
3. Вивести весь список.
4. Видалити елемент списку.
Для виходу з програми натисніть <esc>…
```

Після вибору пункту «1»:

```
Введіть ім'я:
```

Після введення імен «Ira» та «Anya» і вибору пункту 3:

```
1. Ira
2. Anya
Для виходу в попереднє меню натисніть будь-яку клавішу…
```

Після вибору пункту «4»:

```
1. Ira
2. Anya
Введіть номер елемента:
```

Контрольні питання

1. Що таке динамічне виділення пам'яті?

2. Що таке «купа»?
3. В чому полягає проблема фрагментації пам'яті?
4. Які функції слугують для динамічного виділення пам'яті?
5. Наведіть прототип функції `malloc()`.
6. Наведіть прототип функції `calloc()`.
7. Для чого призначена функція `realloc()`?
8. Наведіть прототип функції `realloc()`.
9. Для чого призначена функція `free()`?
10. Наведіть прототип функції `free()`.
11. Що таке зв'язні списки?

Контрольні завдання

1. Напишіть програму, яка впорядковує за зростанням елементи динамічного масиву методом «пухирця».
2. Напишіть програму, яка створює динамічний масив з 20 випадкових елементів в діапазоні від 1 до 10 та виводить його на екран.
3. Напишіть програму, яка у введеному з клавіатури рядку відбирає непарні цифри і заповнює ними один динамічний масив, відбирає парні цифри і заповнює ними інший динамічний масив. Вміст масивів вивести на екран.
4. Напишіть програму, яка вставляє 10 випадкових цілих чисел у діапазоні від 0 до 50 у впорядкований зв'язний список.
5. Напишіть програму, яка об'єднує два впорядкованих зв'язних списки в один впорядкований зв'язний список. Функція `merge()` має отримати покажчики на перший вузол кожного списку і повернути покажчик на перший вузол об'єднаного списку.
6. Модернізуйте програму з задачі 12.1. таким чином, щоб у ній був реалізований:
   а) однонаправлений зв'язний циклічний список;
   б) двонаправлений зв'язний нециклічний список;
   в) двонаправлений зв'язний циклічний список.
7. Модернізуйте програму з задачі 12.1. таким чином, щоб кожен вузол списку зберігав ціле значення, а також був реалізований:
   а) однонаправлений зв'язний нециклічний список;
   б) однонаправлений зв'язний циклічний список;
   в) двонаправлений зв'язний нециклічний список;
   г) двонаправлений зв'язний циклічний список.
8. Напишіть програму, яка містить поточну інформацію про заявки на авіаквитки.
   Кожна заявка містить таку інформацію:
   - пункт призначення;
   - номер рейсу;
   - прізвище та ініціали пасажира;

- бажану дату вильоту.

Програма повинна забезпечувати:

- збереження всіх заявок у вигляді однонаправленого зв'язного списку;
- додавання заявок у список;
- видалення заявок;
- виведення заявок за заданим номером рейсу і датою вильоту;
- виведення усіх заявок.

9. Напишіть програму, яка містить поточну інформацію про книги в бібліотеці.

Потрібно зберігати таку інформацію про книги:

- номер УДК;
- прізвище та ініціали автора;
- назву книги;
- рік видання;
- кількість екземплярів даної книги в бібліотеці.

Програма повинна забезпечувати:

- початкове формування даних про всі книги в бібліотеці у вигляді однонаправленого зв'язного списку;
- додавання даних про книги, які надходять у бібліотеку;
- видалення даних про книги, які списуються;
- по запиту видаються відомості про наявність книг у бібліотеці, упорядковані за роками видання.

10. Автоматизована інформаційна система на залізничному вокзалі містить інформацію про відправлення потягів далекого прямування. Для кожного потяга вказується:

- номер потяга;
- станція призначення;
- час відправлення.

Дані в інформаційній системі організовані у вигляді однонаправленого зв'язного списку.

Напишіть програму, яка:

- забезпечує початкове введення даних в інформаційну систему і формування списку;
- здійснює виведення вмісту всіх елементів списку;
- за введеним номером потяга виводить усі дані про цей потяг;
- за введеною назвою станції призначення виводить дані про всі потяги, що прибувають до цієї станції.

Програма повинна забезпечувати діалог за допомогою меню і контроль помилок при введенні даних.

РЕКОМЕНДОВАНА ЛІТЕРАТУРА

1. Керниган Б. Язык программирования Си = The C programming language / Б. Керниган, Д. Ритчи. — [2-е изд.] — М. : Вильямс, 2007. — 304 с.

2. Шилдт Г. С: Полное руководство, классическое издание = C: The Complete Reference, 4th Edition. / Герберт Шилдт. — М. : Вильямс, 2010. — 704 с.

3. Прата С. Язык программирования С: Лекции и упражнения = C Primer Plus. / Прата С. — М. : Вильямс, 2006. — 960 с.

4. Гукин Д. Язык программирования Си для «чайников» = C For Dummies. / Гукин Д. — М. : Диалектика, 2006. — 352 с.

5. Axel-Tobias Schreiner. Object oriented programming with ANSI-C. / Axel-Tobias Schreiner. — Hanser, 2011. — 223 p.

6. Шилдт Г. Полный справочник по С [Текст]: пер. с англ. / Герберт Шилдт. — [4-е изд.] — М. : Издательский дом «Вильямс», 2004. — 704 с.

7. Седжвик Р. Фундаментальные алгоритмы на С. Анализ/Структуры данных/Сортировка/Поиск [Текст]: пер. с англ. / Седжвик Р. — СПб: ООО «ДиаСофт», 2003. — 672 с.

8. Ткачов В. В. Комп'ютерні технології та програмування [Текст]. Т.1. Теоретичні відомості: навч. посібник / Ткачов В. В., Огєєнко П. Ю., Макітренко Р. В. – Д. : Національний гірничий університет, —2011. — 173 с.

9. Керниган Б. Язык программирования Си. Задачи по языку Си. / Керниган Б., Ритчи Д., Фьюэр А. — Москва : Финансы и статистика, 1985. — 279 с.

10. Дейтел Х. Как программировать на С. / Х. Дейтел , П. Дейтел. — К. : Бином-Пресс, 2006. — 912 с.

11. Уэйт М. Название: Язык Си руководство для начинающих / Уэйт М., Прата С., Мартин Д. — М. : Мир, 1988. — 512 с.

12. Хэзфилд Р. Искусство программирования на С: Фундаментальные алгоритмы, структуры данных и примеры приложений / Р. Хэзфилд, Л. Кирби. — К. : ДиаСофт, 2001. — 736 с.

СЛОВНИК НАЙУЖИВАНІШИХ ТЕРМІНІВ

Аргумент функції – function argument
Арифметичні операції – arithmetic operations
Блок – block
Виклик за значенням – call by value
Виклик функції – function call
Вираз – expression
Ідентифікатор – identifier
Ініціалізація – initialization
Кваліфікатор – qualifier
Керівні послідовності – escape sequence
Ключові слова – keywords
Коментар – comment
Константа – constant
Лексема – lexem
Літерал – literal
Логічні операції – logical operations
Масив – array
Об'єднання – union
Операнд – operand
Оператор – statement
Оператор вибору – selection statement
Оператор вираз – expression statement
Оператор ітерації – iteration statement
Оператор кома – coma operator
Оператор мітки – labeled statement
Оператор переходу – jump statement
Оператори присвоювання – assignment operators
Операції порівняння – relational operations
Операції рівності – equality operations
Операція – operation
Параметр функції – function parameter
Перерахування – enumeration
Побітові логічні операції – bitwise operations
Прототип – prototype
Рядковий літерал – string literal
Символ – symbol
Складене присвоювання – compound assignment
Складений оператор – compound statement
Структура – structure
Тип даних – data type
Умовний вираз ?: – conditional expression
Функція – function

Додаток А
Таблиця кодів ASCII

Dec	Char	Dec	Char	Dec	Char	Dec	Char	Dec	Char	Dec	Char	Dec	Char	Dec	Char
0	NUL	32	space	64	@	96	`	128	А	160	а	192	└	224	р
1	☺	33	!	65	A	97	a	129	Б	161	б	193	┴	225	с
2	☻	34	"	66	B	98	b	130	В	162	в	194	┬	226	т
3	♥	35	#	67	C	99	c	131	Г	163	г	195	├	227	у
4	♦	36	$	68	D	100	d	132	Д	164	д	196	─	228	ф
5	♣	37	%	69	E	101	e	133	Е	165	е	197	┼	229	х
6	♠	38	&	70	F	102	f	134	Ж	166	ж	198	╞	230	ц
7	•	39	'	71	G	103	g	135	З	167	з	199	╟	231	ч
8	◘	40	(	72	H	104	h	136	И	168	и	200	╚	232	ш
9	○	41	)	73	I	105	i	137	Й	169	й	201	╔	233	щ
10	◙	42	*	74	J	106	j	138	К	170	к	202	╩	234	ъ
11	♂	43	+	75	K	107	k	139	Л	171	л	203	╦	235	ы
12	♀	44	,	76	L	108	l	140	М	172	м	204	╠	236	ь
13	♪	45	-	77	M	109	m	141	Н	173	н	205	═	237	э
14	♫	46	.	78	N	110	n	142	О	174	о	206	╬	238	ю
15	☼	47	/	79	O	111	o	143	П	175	п	207	╧	239	я
16	►	48	0	80	P	112	p	144	Р	176	▒	208	╨	240	Ё
17	◄	49	1	81	Q	113	q	145	С	177	▓	209	╤	241	ё
18	↕	50	2	82	R	114	r	146	Е	178	▓	210	╥	242	Є
19	‼	51	3	83	S	115	s	147	У	179	▓	211	╙	243	є
20	¶	52	4	84	T	116	t	148	Ф	180	▓	212	╘	244	Ї
21	§	53	5	85	U	117	u	149	Х	181	▓	213	╒	245	ї
22	▬	54	6	86	V	118	v	150	Ц	182	▓	214	╓	246	Ў
23	↨	55	7	87	W	119	w	151	Ч	183	▓	215	╫	247	ў
24	↑	56	8	88	X	120	x	152	Ш	184	▓	216	╪	248	°
25	↓	57	9	89	Y	121	y	153	Щ	185	▓	217	┘	249	·
26	→	58	:	90	Z	122	z	154	Ъ	186	▓	218	┌	250	·
27	←	59	;	91	[	123	{	155	Ы	187	▓	219	▌	251	√
28	∟	60	<	92	\	124	|	156	Ь	188	▓	220	▄	252	№
29	↔	61	=	93	]	125	}	157	Э	189	▓	221	▐	253	¤
30	▲	62	>	94	^	126	~	158	Ю	190	▓	222	▀	254	■
31	▼	63	?	95	_	127	DEL	159	Я	191	┐	223	■	255	

Навчальне видання

Бевз Олександр Миколайович
Довгалець Сергій Михайлович
Маслій Роман Васильович

Програмування
Частина 2
Програмування мовою С

Навчальний посібник

Редактор Є. Плетньова

Оригінал-макет підготовлено Р. Маслієм

Підписано до друку 14.06.2017 р.
Формат 29,7×42¼. Папір офсетний.
Гарнітура Times New Roman.
Ум. друк. арк. 8,66.
Наклад 50 (1-й запуск 1–20) пр. Зам. № 2017-205.

Видавець та виготовлювач –
Вінницький національний технічний університет,
інформаційний редакційно-видавничий центр.
ВНТУ, ГНК, к. 114.
Хмельницьке шосе, 95, м. Вінниця, 21021.
Тел. (0432)59-87-38,
publish.vntu.edu.ua; *email*: kivc.vntu@gmail.com.
Свідоцтво суб'єкта видавничої справи
Серія ДК № 3516 від 01.07.2009 р.