КОНСПЕКТ ЛЕКЦІЙ
(опорний)

з дисципліни: **«Основи програмування»**

для студентів напрямку підготовки

12.121 – «Інженерія програмного забезпечення»

Івано-Франківськ
2015

Курс лекцій містить короткі теоретичні відомості для ознайомлення з основами структурної методології програмування мовою C/C++, її обчислювальними можливостями, керуванням порядком обчислень, організацією та обробкою основних типів даних, реалізацією файлового обміну даними тощо, що сприяє подальшому самостійному поглибленню знань та навичок студентів у програмуванні мовами високого рівня.

ЗМІСТ

ВСТУП

Мова програмування (англ. Programming Language) – це штучна мова (система позначень) для запису алгоритмів та структур даних у відповідності до лексичних, синтаксичних та семантичних правил, що задають зовнішній вигляд і дії програми, яку виконує електронно-обчислювальна машина (ЕОМ).

З часу появи перших ЕОМ було створено понад дві з половиною тисячі різноманітних мов програмування, кількість яких щороку зростає. Професійні програмісти зазвичай використовують у своїй практиці декілька мов програмування.

Робочою мовою програмування при засвоєнні дисципліни «Основи програмування» обрано популярну, потужну і затребувану алгоритмічну мову високого рівня (МВР) С, яка забезпечує формування найбільш ефективного машинного коду програми, що досягається прив’язкою мови до структури пам’яті та регістрової архітектури ЕОМ.

Популярність мови С, як універсальної мови для створення сучасного програмного забезпечення (ПЗ), та похідних від неї мов програмування (С++, C#, Objective-C тощо) підтверджується також відомим світовим критерієм – індексом TIOBE (TIOBE Programming Community Index).

Основні теми дисципліни передбачають ознайомлення з основами структурної методології програмування мовою С/С++, її обчислювальними можливостями, керуванням порядком обчислень, організацією та обробкою основних типів даних, реалізацією файлового обміну даними тощо, що сприяє подальшому самостійному поглибленню знань та навичок студентів у програмуванні МВР.

МОДУЛЬ 1. ОРГАНІЗАЦІЯ ПРОГРАМ

Змістовий модуль 1. **Засоби програмування та їх складові**

Тема 1. Мови програмування та засоби створення програм
(*тема для самостійного вивчення*)

Тема 2. Історія та особливості мови С

Мова С – універсальна, процедурна, імперативна мова програмування загального призначення, розроблена у 1972 році Денісом Рітчі у Bell Telephone Laboratories з метою написання операційної системи (ОС) UNIX та програмування в ній. Родовід мови бере початок від мови ALGOL та містить мови Pascal і PL/I. Мову названо С, бо багато її можливостей запозичено від існуючої тоді мови B.

Мінімалізм С полягає в тому, що первісно мова проектувалася із розрахунку використання для системного програмування і тому забезпечує формування ефективного машинного коду програми, завдяки прив'язці до структури пам'яті та регістрової архітектури ЕОМ. Тим не менш, С доволі часто використовуєть-ся для написання прикладного ПЗ.

В 1978 році у видавництві Prentice–Hall вийшла книга Брайана Кернігана та Деніса Рітчі «Мова програмування С» (англ. «The C Programming Language»). Версію мови, що описана у в книзі, називають K&R C (за першими літерами прізвищ авторів).

K&R часто вважають найголовнішою частиною мови, яку повинен підтримувати компілятори С.

Наприкінці 1970-х С стала найпопулярнішою мовою програмування для мікрокомп'ютерів, а протягом 1980-х її прийняли для використання на IBM PC, і її популярність почала зростати досить стрімкими темпами.

У 1983 році Американський національний інститут стандартів (англ. American National Standard Institute, ANSI) сформував комітет X3J11 для створення стандартної специфікації С. У 1989 році стандарт було ратифіковано як ANSI X3.159-1989 «The C Programming Language». Цю версію називають ANSI C, Стандартний С, або C89.

У 1990 році стандарт ANSI C (з декількома незначними модифікаціями) ратифікувала Міжнародна організація із стандартизації (International Organization for Standardization, ISO), як ISO/IEC 9899:1990. Цю версію ще називають C90. По суті, терміни C89 й C90 означають одну мову.

C89 наразі підтримується усіма компіляторами С, і більшість існуючого коду написано на базі даного стандарту. Будь-яка програма, написана стандартною С без використання апаратно-залежних засобів, може бути скомпільована і виконана у будь-якій ОС та апаратній архітектурі без будь-яких обмежень ресурсів.

Специфікація мови ANSI C залишилася відносно статичною аж до 1995 року, коли було внесено незначні поправки до стандарту.

Значному перегляду стандарт піддався наприкінці 1990-х. У 1999 році було опубліковано стандарт ISO 9899:1999, або C99. Цим стандартом введено декілька нових особливостей мови, багато з яких вже було реалізовано у певних компіляторах. C99 є сумісним з C90, однак має деякі відмінності.

У грудні 2011року ISO опублікувала оновлений варіант стандарту для мови C – ISO/IEC 9899:2011. Отже C11 прийшов на зміну C99. У новій специфікації збільшена сумісність з мовою C++ і представлені деякі нові можливості. Оскільки стандарт розвивався доволі тривалий час, більшість сучасних компіляторів вже підтримують описані у специфікації можливості.

Як і більшість імперативних мов, C має можливості для структурного програмування та здійснення рекурсії, є мовою функцій, типів даних, операторів присвоєння та керування порядком обчислень. Весь виконуваний код C міститься у функціях, більшість з яких повертає результат. Ці функції знаходяться або у стандартній бібліотеці, або підтримуються конкретним компілятором, або ж пишуться власне програмістом під конкретну задачу. Відтоді, як C набула великої популярності, для неї було написано чимало різноманітних бібліотек. Дозволяється створення програми у вигляді декількох модулів, що транслюються незалежно. Далі ці модулі компонуються в єдиний виконавчий файл. Ця особливість мови дозволяє компілятору підтримувати об’єктні бібліотеки функцій (навіть з інших мов програмування), створюючи програми з декількох окремих модулів.

Головною рисою мови є її переносимість, тобто, як зазначалось раніше, програма, написана стандартною C без використання апаратно-залежних засобів, може бути скомпільована і виконана у будь-якому програмно-апаратному середовищі.

Одним із наслідків значного поширення та ефективності C є те, що компілятори, бібліотеки та інтерпретатори багатьох інших МВР реалізуються мовою C. Також деякими МВР C використовується як проміжна мова. Завдяки величезній популярності мови, останні стандарти багатьох МВР ввели механізм сумісності, що дозволяє просту і стандартизовану взаємодію між програмами МВР та C.

Мова C, ймовірно, є найпопулярнішою у світі мовою програмування за кількістю написаного нею доступного ПЗ та аудиторією користувачів і програмістів. Версії компіляторів мови існують для багатьох ОС та апаратних архітектур.

Мова C здійснила великий вплив на інші, більш сучасні та розвинуті мови програмування, особливо на C++, Objective-C , а також на Java та C#, які запо-зичили у C синтаксис.

Незважаючи на велику популярність, C часто критикується, мовляв ефективне використання C вимагає від програміста більше навичок, досвіду та зусиль, ніж це потрібно для деяких інших мов програмування.

Контрольні запитання до теми 2:

1. Коли, ким і за яких обставин було створено мову програмування C?
2. Як можна класифікувати C?

3. Яке призначення має С та чим обумовлена її популярність?
4. Які основні специфікації та етапи стандартизації С?
5. Яка основні характеристики С?
6. Як впливає С на розвиток сучасних мов програмування?

Тема 3. Реалізація мови. Структура програми

Для розробки С-програми найбільшу популярність набули інтегроване середовище розробки IDE (англ. Integrated Development Environment) та автономний компілятор фірми Borland Software Corporation.

Windows-орієнтоване середовище Borland C++ (файл **bcw.exe**) має текстовий редактор, компілятор, налагоджувач, компонувальник, менеджер проектів, довідкову систему тощо та може бути застосовано під час розробки різномані-тних DOS- і Windows-додатків, у тому числі й 32-бітових.

Розділи головного меню IDE мають таке ж саме призначення, що й аналогічні розділи практично усіх Windows-орієнтованих середовищ, наприклад, того ж Delphi.

Так, для відкриття нового вікна редагування необхідно після запуску файла bcw.exe обрати команду головного меню IDE File|New|Text Edit. У цьому вікні вводиться текст програми, яка спочатку має ім’я noname00.cpp (рис. 3.1).

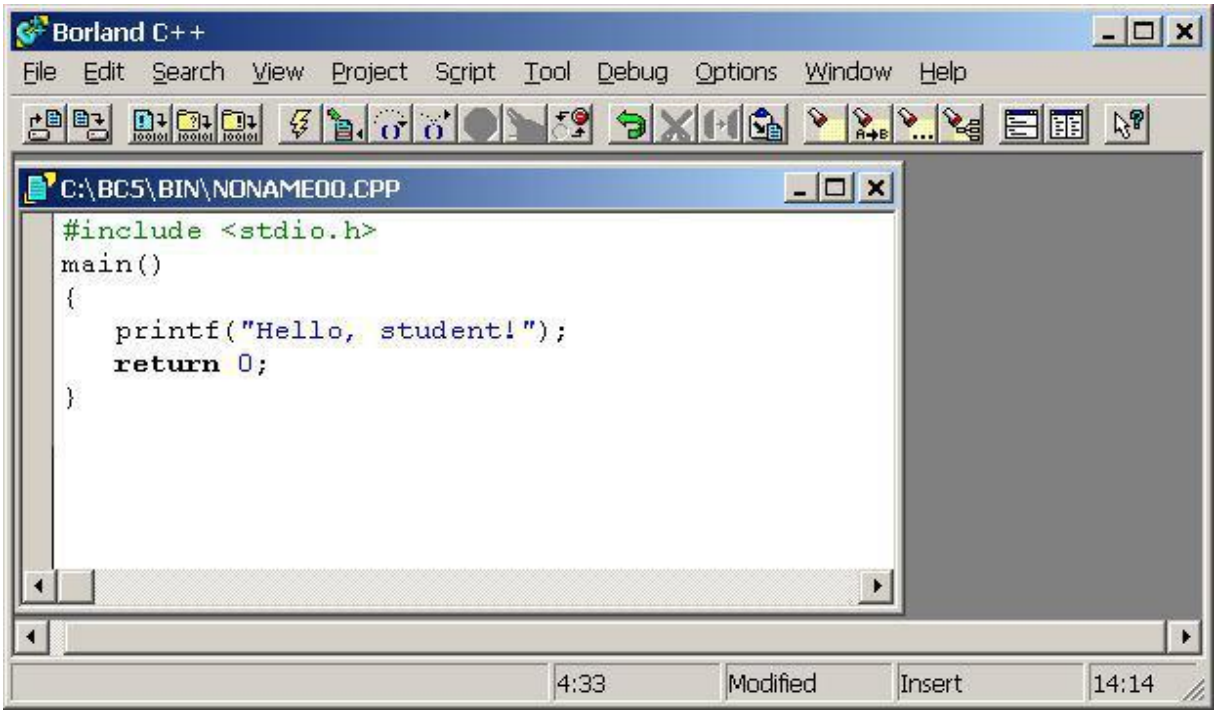


Рисунок 3.1 – Вікно IDE Borland C++

Щоб зберегти введену програму на диску необхідно скористатися командою головного меню IDE File|Save або File|Save as… або відповідною кнопкою на оперативній панелі (SpeedBar).

За допомогою команди головного меню IDE File|Open… або відповідної

кнопки на оперативній панелі (SpeedBar) можна завантажити у вікно редагування будь-який текстовий файл. Зазвичай це файли з розширеннями .c, .cpp або .h.

Якщо у вікні редагування клацнути один раз правою клавішею миші, відкриється оперативне меню (SpeedMenu), в якому необхідно обрати команду, що відкриває діалогове вікно TargetExpert (рис. 3.2).

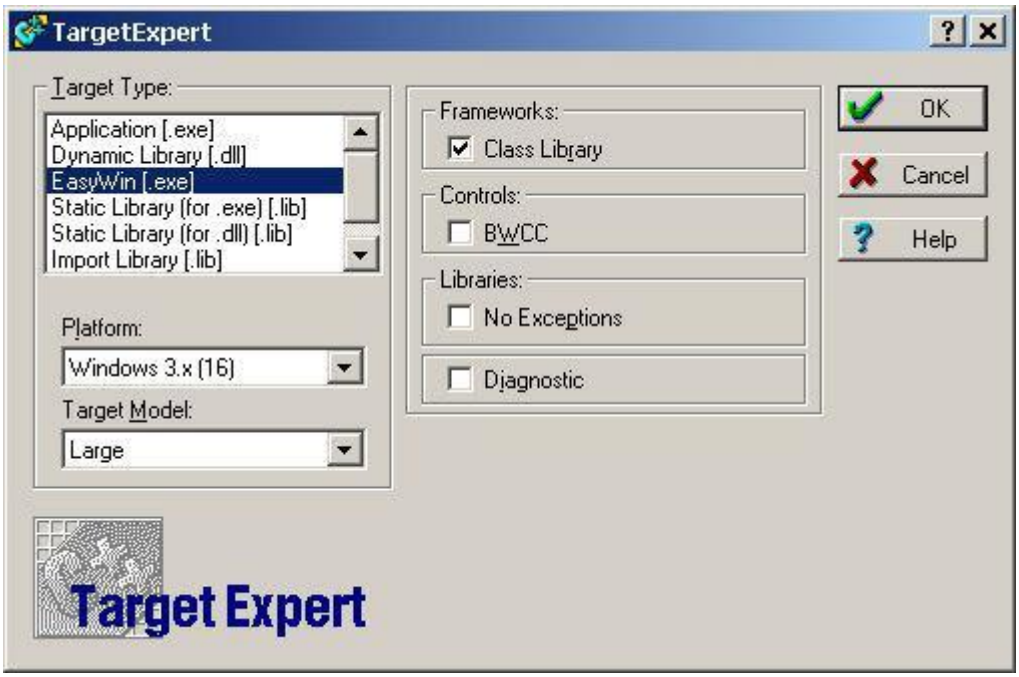


Рисунок 3.2 – Діалогове вікно TargetExpert IDE Borland C++

Наприклад, для компіляції лістингу програми для Windows необхідно у вікні TargetExpert встановити такі ж самі установки, як показано на рис. 3.2 та натиснути кнопку OK або клавішу Enter. Після завершення діалогу з вікном TargetExpert для компіляції та запуску програми необхідно натиснути комбінацію клавіш Ctrl+F9 або скористатися відповідною кнопкою на оперативній панелі (SpeedBar).

Щоб створити консольний додаток для Windows необхідно у вікні TargetExpert встановити Target Type у стан Application [.exe], а Platform – у Win32 та використати модель пам'яті Console (за замовчанням), а далі виконати ті ж самі дії, що й під час компіляції Windows-програми.

Під час компіляції програми IDE відкриває вікно повідомлень (Message) про помилки (Errors), попередження (Warnings), якщо такі є, тощо. Усі помилки та попередження виділяються у вікні редагування, а коментар щодо них міститься у вікні повідомлень. Коментар про помилку містить ім'я файлу (що дуже важливо для багатофайлових програм), номер рядка з помилкою та коротке її пояснення. Текст хибної програми має бути відредагованим та заново скомпі-льованим, без чого неможливо створення виконавчого exe-файлу.

У разі успішної компіляції на екрані з'явиться вікно на кшталт зображеного на рис. 3.3.

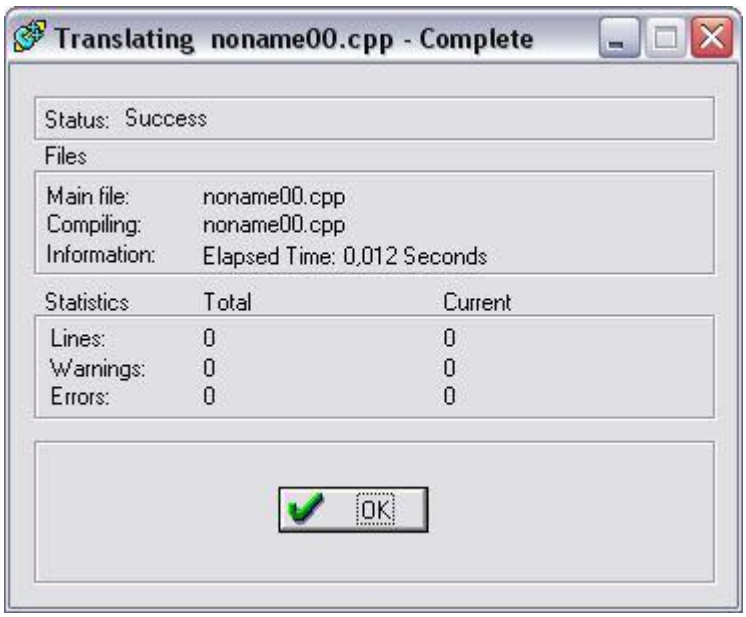


Рисунок 3.3 – Вікно статусу компіляції IDE Borland C++

Для отримання оперативної довідки про будь-які елементи мови програмування достатньо виділити у вікні редагування IDE Borland C++ необхідний елемент (для цього можна клацнути на елементі два рази лівою клавішею миші) та натиснути клавішу F1. Можна також встановити курсор на необхідний елемент програми та натиснути комбінацію клавіш Ctrl+F1.

Автономний компілятор Borland C++ (файл **bcc.exe**) запускається із командного рядка DOS і застосовується для компіляції програм, створених за допомогою будь-якого текстового редактора. Компілятор bcc здатний виконати компіляцію і компонування окремих модулів у виконавчий файл однією командою. Так, наприклад, аби скомпілювати та скомпонувати програму name.c необхідно у командному рядку записати:

```
bcc name.c
```

У разі успішної компіляції та компонування на диску буде створено файл name.exe.

При цьому для С-програм необхідно обов’язково зазначати як ім’я, так і розширення (.c) текстового файла, бо без зазначення розширення текстовий файл буде компілюватися начебто програма C++, тобто як файл з розширенням .cpp.

Окрім можливості компіляції програм C та C++, а також їх автоматичного компонування автономний компілятор Borland C++ здатний компонувати об’єктні модулі (файли з розширенням .obj) та асемблювати програми на асем-блері (файли з розширенням .asm), а також змішувати файли з різними розши-реннями в одній команді. Наприклад, командний рядок:

```
bcc main.c sub1 sub2.obj
```

компілює main.c у main.obj, а sub1.cpp (розширення береться за замовчанням) у sub1.obj і автоматично викликає програму **tlink.exe** аби скомпонувати обидва отримані obj-файли з третім – sub2.obj для створення результуючого файла main.exe.

Комплект поставки Borland C++ містить також 32-бітові автономний компілятор (файл bcc32.exe) і компонувальник (файл tlink32.exe) для створення 32-бітових Windows-додатків.

Найкоротша коректна С-програма має наступний вигляд:

```
main()
{
  return 0;
 }
```

Ця програма містить лише одну головну функцію **main()**.

Відкрита та закрита фігурні дужки обмежують блок функції main(). Зазвичай у цьому блоці розташовуються оператори програми, у тому числі виклики інших функцій. У даному випадку головна функція містить лише один оператор – оператор повернення return, який завершує виконання програми та повертає деяке ціле значення. Ненульові значення оператора return свідчать про помилки у програмі.

Більш складні С-програми можуть містити різні стандартні оголошення або виклики стандартних (бібліотечних) функцій. У такому разі до програми необхідно долучити спеціальні заголовні файли. Наприклад, програма на ANSI C, що містить стандартні оголошення, пов’язані із введенням або виведенням інформації, має починатися директиви включення:

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

Для організації введення-виведення засобами Borland C++ замість наведеного рядка програми записують:

```
#include <iostream.h>
```

Ім’я директиви вказує компілятору на читання тексту з заданого файлу (у даному випадку з файлів stdio.h або iostream.h).

Коментарі ANSI C-програми обмежуються складеними символами: **/*** та ***/**. Такі коментарі можуть використовуватись у будь-якому місці програми, а також займати декілька її рядків. Коментарі Borland C++ починаються з пари символів **//** і все, що знаходиться далі, аж до кінця поточного рядка, є коментарем.

Приклади коментарів:

```
/* Звичайний коментар ANSI C …*/
// Коментар Borland C++ …
```

Ідентифікатори використовуються для присвоєння імен змінним, константам, функціям програми тощо. Ідентифікатори мають починатися з літери і можуть містити виключно літери, цифри та символ підкреслення у будь-якому сполученні. Бажано не використовувати символ підкреслення з самого початку ідентифікатора, бо такі ідентифікатори зарезервовано для системного рівня Borland C++. Довжина ідентифікаторів не обмежена, проте компілятор за замовчанням відрізняє лише перші 32 символи.

Мова C/C++ є чутливою до регістру літер ідентифікаторів, тому, наприклад, наступні ідентифікатори: myVar, myvar або MYVAR – цілком різні ідентифікатори.

Не припустимо у якості ідентифікаторів використання ключових слів мови програмування:

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

Borland C++ розширює наведений список власними зарезервованими словами.

Ключові слова C/C++ записуються виключно малими літерами.

Оголошення змінних у програмі здійснюється за наступними схемами:

ім’я_типу ідентифікатор;

ім’я_типу ідентифікатор_1, …, ідентифікатор_n;

наприклад:

```
int name; /* оголошення цілої змінної на ім’я name */
float a,b,c; /* оголошення декількох дійсних змінних */
```

Змінні, що оголошені у функції main(), є локальними і недоступними з інших функцій. Змінні, що оголошені поза функцією main() та інших функцій, є глобальними і доступними з будь-якого місця програми.

Локальні змінні перед використанням необхідно ініціалізувати за допомогою оператора присвоєння, тобто необхідно задавати їх стартові значення, наприклад, наступним чином:

```
int name;
name=255;
```

або

```
int name=255;
```

Неініціалізовані глобальні змінні автоматично дорівнюють нулю.

Для оголошення у програмі констант використовується ключове слово const, наприклад:

```
const i=0;
const int p=1,s=0;
```

Подальше змінення значень констант у програмі не припустимо.

Контрольні запитання до теми 3:

1. Якими засобами можна розробляти програми C/C++?
2. Яка структура та призначення IDE Borland C++?
3. Які основні етапи розробки програми у IDE Borland C++?
4. Що таке компіляція та компонування програми? Як виконується автономна компіляція програми?
5. Яке розширення мають файли програм C/C++?
6. Як виглядає найкоротша програма? Що таке головна функція?
7. Як завершується виконання програми?
8. Для чого застосовуються заголовні файли (директиви включення)?
9. Як оформлюються коментарі?
10. Які призначення та правила конструювання ідентифікаторів?
11. Що таке ключові слова?
12. Як здійснюється оголошення змінних та констант? Яка між ними різниця?
13. Що означає локальність та глобальність змінних?

Тема 4. Виведення та введення

Стандартні функції **printf()** і **scanf()**, прототипи яких містяться у заголовному файлі stdio.h, здійснюють у ANSI C форматне виведення і введення даних відповідно.

Функція виведення printf() має наступні команди (специфікатори) формату, які починаються з символу **%**:

%c	– символ;
%d, %i	– ціле десяткове число;
%e, %E	– число у форматі x.xx e+xx або x.xx E+xx відповідно;
%f, %g, %G	– десяткове дійсне число;
%o	– вісімкове число;
%s	– рядок символів;
%u	– ціле десяткове число без знаку;
%x, %X	– шістнадцяткове число (наприклад, 7f або 7F відповідно);
%%	– символ %;
%p, %n	– покажчик.

Окрім того, до окремих команд формату можуть бути застосовані моди-

фікатори **l** і **h**, наприклад: %ld – друк long int, %hu – друк unsigned short, %lf – друк long double тощо.

Між символом % і форматом команди може знаходитись ціле число, що зазначає найменше поле, що виділяється для друку. Якщо рядок або число більші за це поле, то вони друкуються цілком, не враховуючи ширину поля. Нуль перед числом формату заповнює нулями усі невикористані місця поля, наприклад:

```
printf(“%05d”,15);
```

Результатом даного оператору буде 00015.

Для зазначення кількості десяткових знаків при виведенні дійсного числа після цілого числа ставлять крапку та число знаків.

Вирівнювання друку відбувається за лівим краєм поля. Для вирівнювання за правим краєм необхідно за символом % поставити символ «мінус».

Приклади:

```
printf(″Result=%d\n″,x);
printf(″%–5.2f″,y);
```

де ″\n″ – керуючий символ переведення рядка, тобто після виведення на екран значення змінної (у даному разі x) положення курсору переміститься на наступний рядок (керуючі символи змістовніше розглянуто у темі 10).

Введення інформації з клавіатури супроводжується відображенням її на екрані. У разі, коли функція виведення scanf() має декілька параметрів, дані потрібно вводити послідовно через пробіл, символ табуляції або нового рядка. Після введення необхідно натиснути клавішу Enter.

Функція введення scanf() автоматично перетворює введені дані у зазначений формат:

%c	– читання символу;
%d, %i	– читання десяткового цілого числа;
%e, %f, %g, %E, %G	– читання числа типу float;
%h	– читання числа типу short int;
%o	– читання вісімкового числа;
%s	– читання рядка символів;
%x	– читання шістнадцяткового числа;
%p, %n	– читання покажчика.

У функції scanf() також припустимо застосування модифікатору формату l, наприклад %lE – читання дійсного числа типу long double.

Усі змінні функції scanf() мають бути зазначені за допомогою адреси, тобто з префіксом **&**. Цей префікс не є обов’язковим лише для масивів.

Приклади:

```
scanf(″%d″,&x);
```

```
scanf(″%i%f%i″,&a,&b,&c);
```

Символ «*» після % перед кодом формату вказує прочитати дані, але не присвоювати введене значення. Так, наприклад, оператор

```
scanf(“%d%*c%d”,&i,&j);
```

під час введення: 50+20 присвоює змінній і значення 50, змінній j значення 20, а символ «+» буде проігноровано.

В команді формату може бути зазначена найбільша ширина поля, що має бути зчитаною. Наприклад:

```
scanf(“%5s”,str);
```

вказує прочитати з потоку введення лише перші п’ять символів. Таким чином, при введенні: 123456789 символьний масив буде містити лише: 12345, а інші символи будуть проігноровані.

Якщо у керуючому рядку зустрічаються якість інші символи, то це означає, що їх треба визначити та пропустити. Так, наприклад, якщо ввести потік символів 5plus10 за допомогою оператора:

```
scanf(“%dplus%d”,&i,&j);
```

змінній і буде присвоєне значення 5, змінній j значення 10, а символи plus будуть проігноровані.

За допомогою функції scanf() можна задавати множину пошуку – набір символів, з якими будуть порівнюватися зчитані с клавіатури символи. Функція зчитує символи доти, доки вони зустрічаються у множині пошуку. Як тільки введений символ не співпав з символами з цієї множини, функція scanf() одразу переходить до наступного специфікатору формату.

Множина пошуку символів задається у квадратних дужках одразу після символу «%», наприклад:

```
#include
<stdio.h> main()
{
  char s[10],t[10];
  scanf(“%[0123456789]%s”,s,t);
  printf(“\n%s..%s”,s,t);
  return 0;
}
```

Після введення з клавіатури: 123abc456 на екран буде видано: 123..abc456, бо символ а не входить до множини пошуку і введення за першим специфікатором формату припиняється для початку введення за другим специ-

фікатором.

Можна також використовувати символ «-» (дефіс) для завдання проміжків множини пошуку, наприклад:

```
scanf(“%10[A-Z1-5]%s”,s,t);
```

де 10 – загальне поле введення.

Можна також визначити символи, що не входять до множини пошуку. Для цього перед першим з них треба поставити символ «^».

У Borland C++ для виведення даних використовується ідентифікатор потоку виведення **cout** і оператор <<, а для введення даних – ідентифікатор потоку введення **cin** і оператор >>, наприклад:

```
cin>>x;
cout <<″Result=″<<x;
```

Контрольні запитання до теми 4:

1. Яким чином здійснюється введення та виведення у С?
2. Що таке стандартна бібліотека введення-виведення?
3. Як функціонує функція printf()?
4. Як функціонує функція scanf()?
5. Що таке поле введення та виведення?
6. Як задається множина пошуку під час введення?
7. Як працюють ідентифікатори потоків введення та виведення С++?

Тема 5. Консольний обмін даними (*тема для самостійного вивчення*)

Тема 6. Типи даних

Числові типи С/С++ поділяються на цілочислові та дійсні.

Точні діапазони значень і розміри пам’яті для різних числових типів даних визначаються певним компілятором та операційною системою ЕОМ.

Діапазони значень цілих чисел Borland C++:

Тип даних	Розмір, байт	Діапазон значень
signed char	1	–128…127
unsigned char	1	0…255
signed short int	2	–32768…32767
unsigned short int	2	0…65535
signed int	2	–32768…32767
unsigned int	2	0…65535
signed long int	4	–2147483648…2147483647
unsigned long int	4	0…4294967295

Слово signed не є обов’язковим. Також не є обов’язковим слово int в ти-

пах short int, long int і unsigned int.

Мова С використовує цілі типи також і для зображення логічних тверджень, тобто нуль означає хибність (false), а ненульове значення – істину (true).

У C/C++ припустимо використання суфіксів **U** (або **u** – unsigned), **L** (або **l** – long) і **H** (або **h** – short) для змінення типу цілочислового літералу, наприклад, 1234L має тип long, 1234U – unsigned int, а 1234UL – unsigned long.

Застосування префіксів **0** (нуль) та **0X** (або 0x) дозволяє записувати цілі числа у вісімковому та шістнадцятковому форматах відповідно, наприклад: 0724, 0x9FAC тощо.

Числа з плаваючою комою (дійсні числа) у C/C++ записуються у формі з десятковою крапкою або за допомогою «наукової нотації», наприклад: 375.5, або 3.755E2, або 37.55e+01, або 0.3755e3, або 37550E–2 тощо.

Діапазони значень дійсних чисел Borland C++:

Тип даних	Розмір, байт	Діапазон значень
float	4	$\pm3.4\cdot10^{-38}\ldots\pm3.4\cdot10^{38}$
double	8	$\pm1.7\cdot10^{-308}\ldots\pm1.7\cdot10^{308}$
long double	10	$\pm3.4\cdot10^{-4932}\ldots\pm1.1\cdot10^{4932}$

Використання суфіксів **F** (або **f** – float) і **L** (або **l** – long double) дозволяє змінювати тип дійсного літералу, наприклад, число 8.75f має тип float, а 8.75L – long double.

Контрольні запитання до теми 6:

1. Які основні числові типи даних?
2. Які основні характеристики цілочислових типів?
3. Які основні характеристики дійсних типів?
4. Яким чином отримуються діапазони значень для числових типів даних?
5. Яким чином можна змінювати тип числового літерального значення?
6. Як правильно вибрати необхідний тип даних?

Тема 7. Вирази та операції

У програмах C/C++ використовуються наступні оператори: арифметичні, відношень, логічні, оператор заперечення, інкременту і декременту, порозрядні, присвоєння.

Арифметичні оператори C/C++:

Оператор	Дія	Приклад
+	додавання	c=a+b;
–	віднімання	c=a–b;
*****	множення	c=a*b;
/	ділення	c=a/b;
%	остача від ділення	c=a%b;

Оператори відношень C/C++:

Оператор	Опис	Приклад
<	менше	(a<b)
<=	менше або дорівнює	(a<=b)
>	більше	(a>b)
>=	більше або дорівнює	(a>=b)
==	дорівнює	(a==b)
!=	не дорівнює	(a!=b)

Логічні оператори **&&** і || об'єднують вирази відношень у відповідності до правил для логічного «І» (AND) і «АБО» (OR) відповідно та використовуються у складених логічних виразах, наприклад:

((a<b)&&(b<c))

або

((a<b)||(c<d))

Унарний оператор заперечення **!** використовується для інвертування результату будь-якого логічного виразу. Наприклад, наступні вирази є еквівалентними:

(!(a<b)) і (a>=b)

та

(!(a==b)) і (a!=b)

Оператор інкременту (++) додає до операнду одиницю, а оператор декременту (--) віднімає її, наприклад:

```
k++; /* k=k+1 */
k--; /* k=k–1 */
x=k++; /* x=k; k=k+1; */
x=--k; /* k=k–1; x=k; */
```

Порозрядні оператори C/C++:

Оператор	Опис	Приклад
&	порозрядне «І»	c=a&b;
|	порозрядне «АБО»	c=a|b;
^	порозрядне виключне «АБО»	c=a^b;
<<	зсув бітів уліво	c=a<<b;
>>	зсув бітів управо	c=a>>b;
~	порозрядне заперечення	c=~a;

Традиційний оператор присвоєння (=) застосовується для присвоєння результату будь-якого виразу змінній, а також для одночасної ініціалізації декількох змінних, наприклад:

```
z=1;
x=y+n;
c=a=b=d+50;
```

У C/C++ застосовуються також скорочені оператори присвоєння:

+=, -=, *=, /=, %=, &=, |=, ^=, <<=, >>=,

наприклад:

```
count+=x; /* count=count+x; */
count*=x; /* count=count*x; */
```

У складних виразах операції виконуються у відповідності до пріоритету та асоціативності (порядку обчислень):

Рівень пріоритету	Оператори	Асоціатив-ність
1.	**(), ., [], –>**	зліва направо
2.	***, &, !, ~, ++, --, +, –, (тип), sizeof**	справа наліво
3.	***, /, %**	зліва направо
4.	**+, –**	зліва направо
5.	**<<, >>**	зліва направо
6.	**<, <=, >, >=**	зліва направо
7.	**==, !=**	зліва направо
8.	**&**	зліва направо
9.	**^**	зліва направо
10.	**|**	зліва направо
11.	**&&**	зліва направо
12.	**||**	зліва направо
13.	**?:**	справа наліво
14.	**=, *=, /=, –=, +=, %=, <<=, >>=, &=, ^=, |=**	справа наліво
15.	**,**	зліва направо

Borland C++ містить достатньо обширну бібліотеку стандартних функцій. Найбільш використовувані наступні математичні функції:

Виклик функції	Призначення	#include
abs(*x*), **labs(*x*)**	– абсолютне значення цілого *x*	math.h
acos(*x*), **acosl(*x*)**	– арккосинус *x*, радіан	math.h
asin(*x*), **asinl(*x*)**	– арксинус *x*, радіан	math.h
atan(*x*), **atanl(*x*)**	– арктангенс *x*, радіан	math.h
atan2(y,*x*),**atan2l(y,*x*)**	– арктангенс y/*x*, радіан	math.h
ceil(*x*), **ceill(*x*)**	– округлення *x* до більшого цілого	math.h
cos(*x*), **cosl(*x*)**	– косинус *x*	math.h
div(*x*,*y*), **ldiv(*x*,*y*)**	– частка і остача від ділення цілих *x* і *y*	stdlib.h

exp(*x*), **expl(*x*)**	– експонента e^x	math.h
fabs(*x*), **fabsl(*x*)**	– абсолютне значення дійсного *x*	math.h
floor(*x*), **floorl(*x*)**	– округлення *x* до меншого цілого	math.h
fmod(*x*,*y*), **fmodl(*x*,*y*)**	– остача від ділення дійсних *x* і *y*	math.h
log(*x*)	– натуральний логарифм *x*	math.h
log10(*x*)	– десятковий логарифм *x*	math.h
modf(*x*,&*y*), **modfl(*x*,&*y*)**	– розбивка дійсного *x* на цілу і дробову частини	math.h
poly(…)	– значення поліному	math.h
pow(*x*,*y*),**powl(*x*,*y*)**	– значення *x* у степені *y*	math.h
pow10(*x*),**pow10l(*x*)**	– значення 10 у степені *x*	math.h
rand()	– генероване випадкове ціле від 0	stdlib.h
random(*n*)	– генероване випадкове ціле від 0 до *n*-1	stdlib.h
randomize()	– ініціалізація генератора випадкових чисел	stdlib.h, time.h
sin(*x*), sinl(*x*)	– синус *x*	math.h
sinh(*x*), **sinhl(*x*)**	– гіперболічний синус *x*	math.h
sqrt(*x*), sqrtl(*x*)	– квадратний корінь *x*	math.h
srand(*n*)	– ініціалізація генератора передбачуваних випадкових чисел	stdlib.h
tan(*x*), tanl(*x*)	– тангенс *x*	math.h
tanh(*x*), tanhl(*x*)	– гіперболічний тангенс *x*	math.h

Деякі функції, із-за їх відсутності серед бібліотечних, необхідно конструювати самостійно, за допомогою інших наявних функцій. Так, напри-клад, логарифм за основою $\log_a x$ можна замінити виразом C/C++: log(*x*)/log(*a*), а котангенс ctg(*x*) – виразом: 1/tan(*x*) тощо.

Також у заголовному файлі math.h визначено багато корисних символічних констант математичного призначення, наприклад:

M_E=2.71828182845904523536 /* *e* */
M_LN10=2.30258509299404568402 /* ln(10) */
M_PI=3.14159265358979323846 /* */
M_SQRT2=1.41421356237309504880 /* $\sqrt{2}$ */

та інші, які використовуються у математичних виразах замість їх літеральних значень.

Контрольні запитання до теми 7:

1. Які основні оператори та як вони використовуються?
2. Який порядок виконання операцій у складних виразах?
3. Які основні функції математичного призначення і як ними користуватися?
4. Як працює скорочений оператор присвоєння?
5. Чим відрізняються префікса й суфіксна форми операторів інкременту і дек-

ременту?

6. Коли і яким чином застосовуються символічні константи?

Тема 8. Оператори. Умовні оператори

Алгоритмічна конструкція, що дозволяє обирати ту чи іншу послідовність дій залежно від певних умов, називається розгалуженням. Існують такі різновиди конструкції розгалуження: альтернатива і поліваріантний вибір.

В алгоритмічній мові C/C++ алгоритми розгалуження реалізуються умовними операторами, що функціонують за наступними схемами (рис. 8.1).

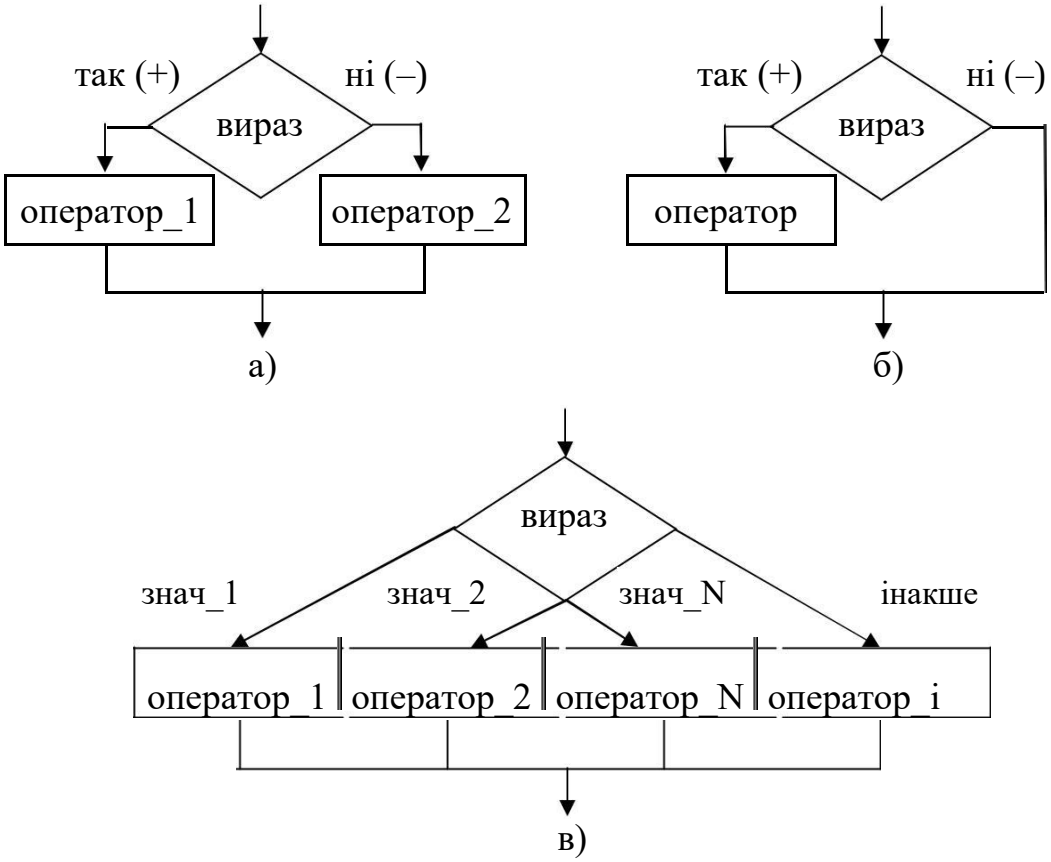


Рисунок 8.1 – Схеми алгоритмів розгалуження

Альтернативне розгалуження (рис. 8.1, а) реалізується умовним оператором за таким синтаксисом:

if (вираз) оператор_1; else оператор_2;

де if (якщо) та else (інакше) – зарезервовані слова; оператор_1 і оператор_2 – довільні оператори, що виконуються в залежності від результату обчислення виразу. Якщо цей вираз є істинним (має ненульове значення), то виконується оператор_1 і керування передається наступному за умовним оператору, а оператор_2 пропускається; якщо вираз є хибним (має нульове значення), то оператор_1 пропускається, проте виконується оператор_2.

В той же час і оператор_1 і оператор_2 можуть бути як простими так і складеними. Складені оператори обмежуються фігурними дужками, наприклад:

```
if(a>b)
b++; else
{
 printf(″%d\n″,a);
 a++;
}
```

Скорочена форма умовного оператора (рис. 8.1, б) реалізує безальтернативне розгалуження (корекцію) за таким синтаксисом:

if (вираз) оператор;

тобто якщо вираз є істинним (має ненульове значення), то оператор виконується, якщо хибним (має нульове значення) – не виконується, і на цьому дія умовного оператора припиняється, наприклад:

```
if(x<=y) x+=y;
```

Логічні оператори **&&** і || об'єднують окремі вирази умовних операторів у складені вирази відношень у відповідності до правил для логічних «І» (AND) та «АБО» (OR) відповідно, наприклад:

```
if((x>=a) && (x<=b)) …; /* якщо a<=x<=b … */
if((x<=a) || (x>=b)) …; /* якщо x<=a або x>=b … */
```

У C/C++ існує також скорочений оператор if–else, який ще має назву умовний вираз або операція умови:

вираз_1 ? вираз_2 : вираз_3;

Програма обчислює вираз_1, і якщо він є істинним (має ненульове значення), то результат усього виразу дорівнює виразу_2, у противному разі результат дорівнює виразу_3.

Умовний вираз цілком відповідає наступному оператору альтернативного розгалуження:

if (вираз_1) вираз_2; else вираз_3;

Отже наступні оператори, що знаходять найбільше з двох чисел, є цілком еквівалентними:

```
if(x>y) max=x; else max=y;
```

та

```
max=(x>y)?x:y;
```

За допомогою умовного виразу можна, наприклад, наступним чином розв’язати задачу заміни нулем найменшого серед двох цілих чисел x і y:

```
(x>y)?x:y=0;
```

Гілки деякого розгалуження можуть містити інші розгалуження:

```
if (вираз_1) оператор_1;
    else if вираз_2 оператор_2;
        else оператор_3;
```

Поліваріантний вибір (рис. 8.1, в), що дозволяє виконувати одну з декількох алгоритмічних гілок залежно від значення деякого виразу, у C/C++ реалізується оператором вибору за таким синтаксисом:

```
switch (вираз)
{
 case значення_1 : оператор_1; break;
 case значення_2 : оператор_2; break;
 …
 default : оператор_замовчання;
}
```

Оператор вибору виконується за таким алгоритмом. Спочатку обчислюється значення виразу switch. Потім це значення порівнюється по черзі з запропонованими значеннями селекторів case (значення_1, значення_2, …, значення_N тощо). Щойно результат порівняння дасть значення «істина», одра-зу ж виконується відповідний оператор або набір операторів, аж до першого наявного оператора break. Якщо значення виразу switch не збігається з жод-ним значенням селекторів case, то виконується необов’язкова гілка default, або, якщо її нема, виконання оператора вибору завершується. Оператор break у кожному блоці case негайно призводить до завершення оператора switch.

Приклад застосування оператора switch:

```
switch(i)
{
 case 1:printf(″i=1\n″); break;
 case 2:printf(″i=2\n″); break;
 default:printf(″i=All\n″);
}
```

Контрольні запитання до теми 8:

1. Яка алгоритмічна конструкція називається розгалуженням?
2. Які різновиди конструкції розгалуження і як вони працюють?
3. Який синтаксис умовного оператора?
4. Який синтаксис та призначення умовного виразу або операції умови?
5. Що таке складений оператор?
6. Як поєднуються в умовних операторах прості вирази в складений?
7. Яке призначення та як функціонує оператор поліваріантного вибору?
8. Як підвищити ефективність алгоритмів розгалуження?

Змістовий модуль 2. **Керування порядком обчислень**

Тема 9. Оператори циклу

Під час розробки програми доволі часто постає потреба виконати один і той самий оператор декілька разів. Для цього застосовують оператори циклів, що реалізують алгоритмічну конструкцію повторення. Кожне повторне виконання операторів, що являють собою тіло циклу, називається ітерацією.

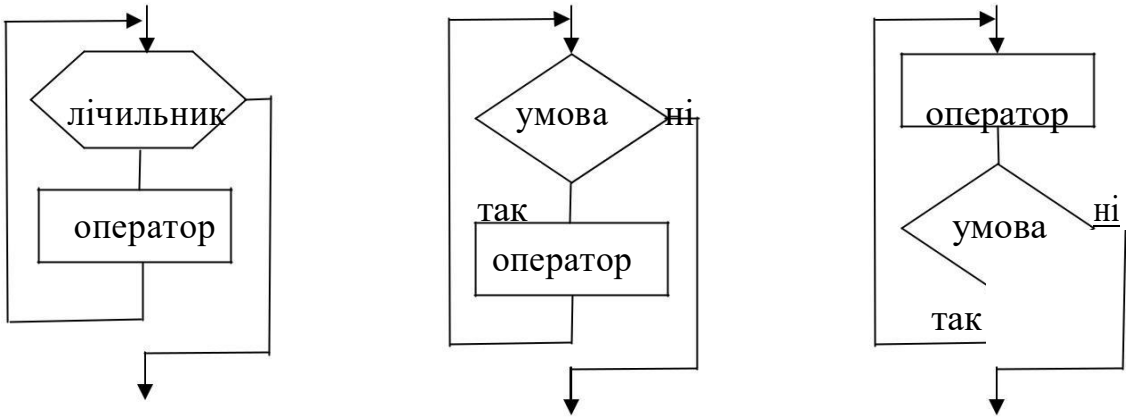
В програмах C/C++ застосовуються цикли трьох видів (рис. 9.1):

– із параметром-лічильником (**for**);

– із передумовою (**while**);

– із постумовою або післяумовою (**do–while**).

Вибір того чи іншого оператора циклу визначається застосованим алгоритмом розв’язку певної задачі.



а) б) в)

Рисунок 9.1 – Схеми операторів циклу

Основна форма циклу for має наступний синтаксис (рис. 9.1, а):

for (вираз_1; вираз_2; вираз_3) оператор;

де for – зарезервоване слово; вираз_1 (ініціалізуючий) виконується лише один раз, перед початком роботи циклу; вираз_2 зазвичай є виразом відношення, що

визначає умову виконання оператору; вираз_3, як правило, застосовується для змінення параметра циклу (збільшення або зменшення)з заданим кроком.

Наприклад, наступні рядки програми виводять на екран спочатку числа від 0 до 9, а потім від 9 до 0 за допомогою оператора циклу for (кожне з нового рядка):

```
for(i=0;i<=9;i++) printf(″%i\n″,i);
for(i=9;i>=0;i--) printf(″%i\n″,i);
```

Кожен із виразів заголовку оператора for може бути відсутнім, проте символ «;» вказується обов'язково. Наприклад, якщо вирази оператора for зовсім поминути, то отримаємо безкінечний цикл:

```
for(; ;) …;
```

Для об'єднання декількох послідовно записаних підвиразів оператора for в один загальний вираз застосовується операція «кома». Так, наприклад, наступний оператор обчислює значення суми s чисел від 1 до 10:

```
for(n=1,s=0;n<=10;n++) s+=n;
```

При використанні оператора for дозволяються внутрішні оголошення змінних циклу, але ці змінні можна використовувати лише у межах даного циклу, наприклад:

```
for(int i=1;i<=n;i++) …;
```

Цикл із передумовою (while) працює за наступним синтаксисом (рис. 9.1, б) і застосовується для повторного виконання оператора упродовж усього часу, доки значення виразу є істинним (ненульовим):

while (вираз) оператор;

У циклі з передумовою перевірка умови виконання циклу (значення виразу) відбувається ще до першого виконання його тіла. Якщо умова істинна (ненульове значення виразу), то виконується оператор (тіло циклу), інакше – цикл не виконується, тобто цикл виконується, доки умова не стане хибною. Це означає, що тіло циклу з передумовою може не виконатися жодного разу.

Наступний фрагмент програми виводить на екран латинський алфавіт у верхньому регістрі:

```
int c=65;
while(c<=′Z′) printf(″%c″,c++);
```

Тіло циклу має містити хоча б один оператор, що впливає на умову закін-

чення ітераційного процесу, інакше відбудеться «зациклення»:

```
while(x<=n) y+=i; /* приклад зациклення */
```

Оператор циклу з постумовою (do–while) є «перевернутим» циклом while, тобто виконується оператор, а вже потім перевіряється значення виразу (рис. 9.1, в). Цикл працює за наступним синтаксисом і повторюється доти, доки значення виразу буде істинним (ненульовим):

do оператор; while (вираз);

Отже цикл з постумовою за будь-яких обставин буде виконано принаймні один раз – в цьому і полягає його головна відмінність від циклу з передумовою.

Наступний фрагмент програми виводить на екран латинський алфавіт у верхньому регістрі, тобто виконує ті ж самі дії, що і попередній фрагмент, але за допомогою циклу do–while:

```
int c=65;
do print(″%c″,c++); while(c<=′Z′);
```

Зауважимо, що оператори тіла циклу з постумовою мають впливати на умову закінчення ітераційного процесу, інакше відбудеться «зациклення».

Для передчасного завершення будь-якого циклу застосовуються оператори переходу **break** та **continue**. Для примусового переривання циклу слід викликати оператор break. Оператор continue здійснює пропуск усіх інструкцій, записаних за її викликом у тілі циклу, та керування передається на перевірку виразу, що обумовлює завершення (або продовження) циклу.

У разі повторного виконання у тілі циклу декількох операторів маємо складений оператор циклу. Його тіло обмежується фігурними дужками і може містити будь-які оператори, у тому числі і циклу.

Якщо оператор циклу містить інші циклічні оператори, то йдеться про вкладені цикли (цикли у циклі). Ступінь вкладеності при цьому може бути достатньо глибоким, в залежності від складності ітераційного процесу. Окрім того, цикли одного типу можуть бути вкладеними у цикл іншого типу.

Приклади вкладених циклів:

```
for(i=first;i<=last;i++)
  for(j=v;j>=w;j--) …;

while(x<=n)
  for(i=last;i>=first;i-=x) …;
```

Вкладені цикли працюють наступним чином. Виконується перша ітерація зовнішнього циклу і якщо у циклі знаходиться оператор вкладеного циклу, то програма передає керування на його виконання. Після завершення усіх ітерацій

вкладеного циклу починається наступна ітерація зовнішнього циклу, під час якої знову передається керування на оператор вкладеного циклу і т.д. Алгоритм закінчується після завершення усіх зовнішніх ітерацій.

Неприпустимо перетинання окремих циклів.

Якщо для примусового переривання циклу використовується оператор переходу break, то переривається той цикл, у якому він викликається. Після переривання внутрішнього циклу керування передається заголовку зовнішнього циклу для перевірки умови його продовження.

Аналогічно працює і оператор continue, але його використання забезпечує виконання нової ітерації того циклу, в якому виклик continue здійснено.

Як і раніше, під час застосування вкладених циклічних алгоритмів необхідно забезпечувати скінченність ітераційного процесу аби уникнути можливого «зациклення».

Взагалі рекомендується уникати застосування операторів переходу для передчасного завершення як простих, так і вкладених циклічних процесів.

Для безумовного завершення поточної функції (у тому числі функції main()) використовуються оператори **exit()** і **abort()**, прототипи яких знаходяться у заголовному файлі stdlib.h, наприклад:

```
exit(0);
abort();
```

Не бажано завершувати поточну функцію (у тому числі функцію main()) шляхом застосування операторів на кшталт exit() і abort(), розташо-ваних у тілі циклу.

Контрольні запитання до теми 9:

1. Коли застосовуються ітераційні процеси та що таке цикли?
2. Які існують різновиди циклів?
3. Який синтаксис операторів циклу?
4. Яке призначення мають вирази оператора for?
5. У чому полягає відмінність різних умовних циклів?
6. Що може спричинити «зациклення» програми?
7. Як працюють оператори переривання циклу break та continue?
8. Що означає термін «вкладені цикли»? Як вони працюють?
9. Яка може бути ступінь вкладеності ітераційного процесу?
10. Як працюють оператори переходу у вкладених циклах?
11. Коли вкладений циклічний процес може «зациклитись»?
12. Як забезпечити безумовне завершення головної функції програми?

Тема 10. Особливості роботи з символьним типом даних

Доволі часто у практиці програмування виникають ситуації, коли для розв’язку певної задачі доводиться обробляти не окремі дані (наприклад, числових або символьного типів), а деякі їх послідовності.

Річ у тім, що для зберігання й обробки послідовностей однотипних даних у мовах програмування зазвичай застосовуються масиви. Насправді ж багато задач може бути розв’язано без використання будь-яких структур даних.

Серед задач такого типу частіше за все виступають алгоритми накопичення, тобто коли замість зберігання усієї послідовності даних (чисел, символів тощо) у програмі оголошується спеціальна змінна, в якій саме й накопичується результат кожної попередньої ітерації.

Так, наприклад, для обрахування суми або добутку значень деякої числової послідовності зовсім нема потреби в оголошенні масиву, бо такі дії цілком можуть бути виконані за допомогою лише одного циклу, під час введення даних.

Значення символьного (літерного) типу char – непустий символ кодової таблиці ASCII (англ. American Standard Code Information Interchange – американський стандартний код обміну інформацією) алфавіту ПК.

Оголошення змінної символьного типу у С-програмі здійснюється наступним чином:

```
char c;
```

або

```
unsigned char c;
```

Можна разом із оголошенням символьної змінної присвоїти їй початкове значення (в одинарних лапках), наприклад:

```
char c='A' ;
```

або

```
char c='\x41'; // 41 – шістнадцятковий ASCII-код символу 'A'
```

або

```
char c='\101'; // 101 – вісімковий ASCII-код символу 'A'
```

Зображення символів у вигляді їх шістнадцяткового або вісімкового ASCII-кодів зручно лише для керуючих символів (коди ASCII '\x01'–'\x1f'). Крім того, деякі керуючі символи можна записувати за допомогою так званих керуючих послідовностей, наприклад:

'\a' – звуковий сигнал; **'\n'** – переведення рядка; **'\r'** – повернення каретки;
'\t' – горизонтальна табуляція;
'\v' – вертикальна табуляція.

Використання керуючих послідовностей символів можливо всюди, де можуть вживатись друковані символи. Наприклад, використання керуючої послідовності '\n' у операторі printf() після виведення переводить курсор на наступний рядок:

printf(″Курсор буде переведено на наступний рядок\n″);

Символьній змінній можна присвоювати значення іншої символьної змінної, а також можна порівнювати символьні змінні між собою. Символи вважаються рівними, якщо рівні їх ASCII-коди. Більшим є символ з більшим ASCII-кодом.

До значень символьного типу корисно застосовувати макроси класифікації символів **is…**, визначені у заголовному файлі ctype.h:

isalnum – алфавітно-цифровий символ;
isalpha – алфавітний символ;
isascii – символ ASCII (0<=c<=0x7E);
iscntrl – керуючий код ASCII;
isdigit – цифра від ′0′ до ′9′;
isgraph – друкований символ, крім пробілу;
islower – мала літера від ′a′ до ′z′;
isprint – друкований символ або пробіл;
ispunct – символ пунктуації;
isspace – пробіл, табуляція, повернення каретки, переведення рядка, вертикальна табуляція;
isupper – велика літера від ′A′ до ′Z′;
isxdigit – шістнадцяткова цифра від ′0′ до ′9′ або літера від ′A′ до ′F′ (від ′a′ до ′f′).

Ці макроси повертають ненульове значення (істину), якщо символ належить до заданої множини символів, або нуль (хибність) – у противному разі.

Дія цих макросів не розповсюджується на символи з кодами ASCII більше, ніж 0x7E (наприклад, кирилицю).

До даних символьного типу можуть бути застосовані наступні функції:

Функція	Призначення	Приклад	#include
getc()	– читання символу з відображенням	c=getc(stdin);	stdio.h
getch()	– читання символу без відображення	c=getch();	conio.h
getchar()	– читання символу з відображенням	c=getchar();	stdio.h
getche()	– читання символу з відображенням	c=getche();	conio.h
putc()	– відображення символу	putc(c,stdout);	stdio.h
putch()	– відображення символу	putch(c);	conio.h
putchar()	– відображення символу	putchar(c);	stdio.h

Приклади:

```
char c; // оголошення символьної змінної
c=getche(); // читання символу
if(isdigit(c)) s++; // застосування макросу класифікації символів
```

Контрольні запитання до теми 10:

1. Що таке послідовність даних?
2. Яким чином обробляються послідовності?
3. Як за допомогою послідовностей уникають використання масивів?
4. Що таке кодова таблиця ASCII?
5. Яке призначення мають керуючі символи? Які їхні коди?
6. Як оголошуються та використовуються змінні символьного типу?
7. Що означають та як застосовуються керуючі послідовності?
8. Коли і як застосовуються макроси класифікації символів?
9. Які стандартні функції C/C++ застосовуються до даних символьного типу?

Тема 11. Функції користувача

Функції є самостійними, логічно завершеними одиницями програми, що виконують певні дії відповідно до алгоритму розв'язку конкретної задачі. Завдяки застосування функцій спрощується процес програмування, а самі програми стають більш структурованими, зрозумілішими та лаконічними.

Кожна С-програма містить як мінімум одну функцію, а саме функцію main(), та може використовувати інші, наприклад, бібліотечні функції. Для розв'язку у програмі окремої самостійної задачі бажано розробити для неї й окрему функцію (так звану функцію користувача).

Для використання в С-програмі функції користувача її необхідно:
– оголосити;
– визначити (описати);
– викликати.

Оголошення функції зазвичай виконується перед початком функції main() і містить тип результату функції, її ім'я та список параметрів (зазначених у круглих дужках через кому):

тип Ім'я_функції (список_параметрів);

Таке оголошення називається прототипом функції, наприклад:

```
float Cube(float x);
int Fun(float y,int z);
void F1(int, double);
void F2(void);
```

Тип функції вказує тип значення, що повертається функцією в програму. Зазвичай, це один з числових типів (int, float тощо), рядок або покажчик на більш складний тип мови. Іноді функція взагалі не повертаю результату. Така функція має тип void.

Ім'я функції призначене для її ідентифікації у програмі і має бути записане за правилами побудови ідентифікаторів мови. Бажано аби ім'я функції відображувало її призначення.

Списком формальних параметрів позначаються вхідні дані, значення яких передаються функції під час виклику у вигляді фактичних параметрів (аргументів). Кількість, черговість та типи параметрів функції має бути узгоджено з оператором виклику функції у програмі. Для кожного з параметрів необхідно вказати тип, навіть коли типи послідовних параметрів збігаються. Типи параметрів функції можуть бути довільними типами мови. Параметром функції, що не опрацьовує жодних вхідних значень, зазвичай виступає ключове слово void.

Як правило прототип функції цілком збігається з її заголовком у визначенні, проте імена параметрів у прототипі не мають ніякого значення. Їх взагалі можна не вказувати, зазначивши лише тип.

Визначення функцій (їх описи) виконуються після оголошення прототипів. Рекомендується такі визначення розташовувати після функції main().

Опис функції починається з заголовка, який містить тип результату функції, її ім’я та список параметрів (зазначених у круглих дужках через кому) і зазвичай співпадає з прототипом (але без прикінцевої крапки з комою).

Вміст самої функції (її тіло) обмежується фігурними дужками, на кшталт функції main(), між якими можна оголошувати змінні, викликати інші функції, виконувати оператори тощо. Не дозволяється у функції створювати внутрішні функції.

Додаткові змінні, що можуть бути оголошені у функції, називаються локальними або автоматичними і зберігаються у стеку, тобто є доступними лише у даній функції під час її роботи. Такі змінні рекомендується розташовувати перед першим оператором функції.

Функція завершує своє виконання або коли досягнуто її кінця (виконано усі оператори), або завдяки оператору **return**. Типізовані функції обов’язково мають містити хоча б один оператор return, одразу за яким зазначається відповідне значення, що повертається цією функцією у програму. Лише void-функції можуть використовувати оператор return без завершального значення, або зовсім його не містити.

Приклад опису функції, що обчислює третю степінь переданого їй аргументу:

```
float Cube(float x)
{
  return x*x*x;
}
```

Звертання до функції у необхідному місці програми відбувається за допомогою оператора виклику – вказується ім’я функції та перелічуються у круглих дужках її фактичні параметри (аргументи). Якщо функція повертає результат, то її виклик можна використовувати у виразах як звичайний операнд, тип якого збігається з типом значення функції. Окрім того, виклик такої функції можна здійснити автономно, тобто через окремий оператор. У такому разі значення, що повертає функція, ігнорується. Виклик void-функції здійснюється виключно автономно.

Під час звертання до функції всі її формальні параметри отримують копії значень відповідних фактичних параметрів (аргументів), заданих оператором виклику. Тому кількість, порядок розташування та тип фактичних і формальних параметрів функції, як зазначалося раніше, повинно бути узгоджено (за винятком спеціальних функцій зі змінною кількістю параметрів). Гарантується недоторканість фактичних параметрів (аргументів) під час роботи функції. Останнє правило не розповсюджується на параметри-масиви, параметри-рядки та на деякі ще особливі параметри.

Приклади виклику функцій:

```
d=a*a+Cube(b)/3;
F1(x1,x2);
printf(″Result=%f″,Cube(a+b));
```

У першому прикладі обчислюється значення змінної d за формулою: $d=a^2+b^3/3$. Для визначення кубу змінної b застосовано виклик функції Cube(b), опис якої показано раніше за текстом. Виклик даної функції передає їй аргумент b через фактичний параметр x. Функція обчислює третю степінь параметра x і повертає її за допомогою оператора return у вираз програми у якості результату.

У другому прикладі здійснюється виклик функції, яка отримує два параметра x1 і x2 та, скоріш за все, не повертає жодного значення, тобто має тип void.

Третій приклад демонструє виклик раніше розглянутої функції для обчислення третьої степені суми двох дійсних чисел у якості аргументу іншої функції, у даному разі – printf().

Оператори виклику функцій можуть розташовуватись як у межах функції main(), так і у межах інших функцій.

Локальні змінні та формальні параметри функцій запам’ятовуються у стеку. Область дії таких змінних розповсюджується лише на дану функцію і після завершення функцій їх значення стають невідомими. У стеку запам’ятовуються також адреси повернення з функцій

На відміну від локальних, глобальні змінні запам’ятовуються у сегменті даних програми і доступні з будь-якої її функції. Використання глобальних змінних може бути корисним для оголошення великих структур даних з метою економії стекової пам’яті, але звертання у функціях до глобальних змінних програми порушує засади структурного програмування і може призвести до помилок, які дуже складно викрити та виправити.

Якщо оператор у функції містить виклик цієї ж функції, така функція називається рекурсивною.

Головною вимогою до рекурсивних функцій є необхідність забезпечення коректного виходу з них, аби не призвести до переповнення стеку і «зависанню» системи.

Наприклад, обчислення факторіала числа n ($n!=1\cdot2\cdot3\cdot\ldots\cdot n$) дуже зручно виконувати за допомогою рекурсивної функції.

```
double Factorial(int n)
{
 if(n>1) return n*Factorial(n–
 1); return 1;
}
```

Передача у функції масивів, рядків та більш складних структур даних має певні особливості та потребує застосування спеціальних прийомів програмування та частково розглянуто у відповідних наступних темах.

Контрольні запитання до теми 11:

1. Що таке функція та яке її призначення в програмі?
2. Які види функцій може містити програма?
3. Що необхідно для використання в програмі функції користувача?
4. Що таке прототип, формальні та фактичні параметри (аргументи) функції?
5. Яку структуру має функція користувача, та як здійснюється її виклик?
6. Який тип можуть мати функції та їх параметри? Що таке void-функції та void-параметри?
7. Яка функція називається рекурсивною та які вимоги до її написання?

Тема 12. Класи пам’яті даних (*тема для самостійного вивчення*)

Тема 13. Покажчики (*тема для самостійного вивчення*)

Модуль 2. Організація даних

Змістовий модуль 3. **Прості структури даних**

Тема 14. Масиви

У масивах об’єднуються елементи одного типу. Елементи масиву розташовуються у пам’яті послідовно, починаючи з першого елементу (перший елемент має індекс 0) .

Оскільки перший індекс масиву завжди дорівнює нулю, то для будь-якого оголошення на зразок:

t name[n];

де t – тип даних, припустимі значення індексів знаходяться у діапазоні від 0 до n–1. Використання індексу за межами цього діапазону призведе до звернень до областей пам’яті, що не належать масиву.

Отже для оголошення масиву необхідно оголосити змінну (ім’я масиву) та додати слідом у квадратних дужках ціле число, котре буде визначати кіль-

кість елементів у масиві. Наприклад, оголошення:

```
int m[100];
```

визначає масив на ім’я m, здатний зберігати до 100 значень типу int. Типові приклади оголошення вищезгаданого масиву:

```
#define MAX 100
…
int m[MAX];
```

або

```
const max=100;
int m[max];
```

Аби сформувати вираз, що має відношення до конкретного елементу масиву, необхідно після імені масиву зазначити індекс цього елементу (у квадратних дужках). У якості індексу має бути цілий вираз – константа, змінна, результат функції тощо. Наприклад, оператор:

```
m[5]=55;
```

присвоює значення 55 шостому елементу масиву m.

Послідовний доступ до декількох елементів масиву здійснюється за допомогою циклічних алгоритмів. Наприклад, наступний фрагмент програми присвоює нуль значенням перших n елементів масиву:

```
for (int i=0;i<n;i++) m[i]=0;
```

Оголошення масивів можуть бути глобальними і локальними. Якщо змінна-масив – глобальна, то елементи масиву автоматично ініціалізуються нулями. Якщо ж масив локальний у функції, то його значення є непередбачуваними і потребують ініціалізації.

Для ініціалізації необхідно оголосити масив (як зазвичай), після чого у фігурних дужках записати (через кому) список значень констант. Наприклад, оголошення:

```
int m[10]={0,1,2,3,4,5,6,7,8,9};
```

створює масив із десяти цілих значень на ім’я m і послідовно присвоює значення від 0 до 9 кожному елементу масиву від m[0] до m[9].

Можна переписати розглянуте оголошення наступним чином:

```
int m[]={0,1,2,3,4,5,6,7,8,9};
```

У такому разі порожні квадратні дужки дозволяють компілятору автома-

тично визначати точний розмір пам’яті, необхідний для зберігання перерахованих елементів.

При явному завданні розміру масиву не є обов’язковим завдання значень для кожної позиції масиву. Наприклад, оголошення:

```
int m[10]={0,1,2,3,4};
```

створює масив із десяти цілих значень, але ініціалізує лише перші п’ять.

Завдання зайвої кількості елементів у фігурних дужках призводить до помилки компіляції.

Якщо m – це ім’я певного масиву, то вираз sizeof(m) визначає число байт, що займає масив m у пам’яті, а sizeof(m[0]) – число байт, що займає один елемент масиву.

Під час попередньої ініціалізації масиву можна використовувати sizeof() для коректного визначення числа елементів масиву, наприклад:

```
int m[]={0,1,2,3,4};
#define MAX sizeof(m)/sizeof(m[0])
```

Аби не допустити подальших корегувань масиву, необхідно перед його оголошенням вказати ключове слово const, наприклад:

```
const float mas[]={0,1,2,3,4,5,6,7,8,9};
```

Зазвичай початковими значеннями масив заповнюється з клавіатури або файлу за допомогою циклу. Так, наприклад, наступний фрагмент програми демонструє заповнення масиву цілих чисел з клавіатури. Накопичення значень завершується коли користувачем введено число нуль, або коли вичерпано число комірок (MAX), попередньо виділене для зберігання масиву директивою #define:

```
…
n=0; /* n – лічильник елементів масиву
*/ do
{
  printf("m[%i]=",n);
  scanf("%i",&m[n]);
}while((m[n++])&&(n!=MAX));
n=(m[n–1])?n:--n;
…
```

Масиви можна передавати у функції користувача. Так, наступна функція обраховує і повертає суму значень деякого масиву дійсних чисел:

```
double Sum(double data[], int k)
```

```
{
 double s=0;
 while(k>0) s+=data[--k];
 return s;
}
```

Викликати цю функцію можна, наприклад, наступним чином:

```
printf(″Result=%lf\n″,Sum(m,n));
```

де n – кількість елементів у масиві m.

При передачі масиву у функцію передається не вміст масиву, а лише його адреса (ім'я масиву). Це означає, що функція оперує безпосередньо з параметром-масивом, тобто може змінювати його значення. Для уникнення таких змін використовуються параметри-константи, наприклад:

```
void Func(const double data[], int k)
```

Можна також формальний параметр, через який задається масив, оголосити як покажчик, а для звертання у функції до елементів масиву використовувати операцію розадресації.

Масив може мати два або й більше вимірів. Такі масиви називаються багатовимірними.

Найбільш розповсюдженим на практиці є використання двовимірних масивів або так званих матриць. Елементи таких масивів запам'ятовуються рядок за рядком. Можна вважати, що двовимірний масив є масивом одновимірних масивів (підмасивів).

Так, наприклад, після оголошення у програмі матриці дійсних чисел:

```
int x[3][4];
```

розташування її елементів x[i][j] буде відповідати наступній таблиці:

x[0][0]	x[0][1]	x[0][2]	x[0][3]
x[1][0]	x[1][1]	x[1][2]	x[1][3]
x[2][0]	x[2][1]	x[2][2]	x[2][3]

де перший індекс i є номером рядка, а другий j – номером стовпця (індексація кожного з вимірів починається з нуля).

У пам'яті комп'ютера елементи такого масиву розташується наступним чином: x[0][0], x[0][1], x[0][2], x[0][3], x[1][0], x[1][1], x[1][2], x[1][3], x[2][0], x[2][1], x[2][2], x[2][3] і займають 24 (12×2) байт пам'яті.

Ініціалізувати такий масив можна в один із наступних способів:

```
float x[3][4]={0.0,0.1,0.2,0.3,1.0,1.1,1.2,1.3,2.0,2.1,2.2,2.3};
```

або

```
float x[3][4]=
{
  {0.0,0.1,0.2,0.3},
  {1.0,1.1,1.2,1.3},
  {2.0,2.1,2.2,2.3}
};
```

де, наприклад, вираз x[1][2] відповідає значенню 1.2.

До речі, в оголошенні багатовимірного масиву дозволяється залишати порожніми перші квадратні дужки, тобто опускати розмірність найстаршого виміру, наприклад:

```
float x[][4]={0.0,0.1,0.2,0.3,1.0,1.1,1.2,1.3,2.0,2.1,2.2,2.3};
```

У цьому випадку кількість рядків матриці обчислюється як частка від ділення кількості даних (у даному прикладі їх дванадцять) на відому кількість елементів у кожному рядку матриці (у даному прикладі – чотири).

Матриця може складатися з одного рядка, або з одного стовпця, або навіть мати лише один елемент. В останньому випадку оголошення та ініціалізація матриці може виглядати, наприклад, наступним чином:

```
int x[1][1]={100};
```

Для доступу до елементів багатовимірного масиву застосовується класична індексна форма звертання, тобто вказавши в окремих квадратних дужках значення індексу для кожного з вимірів, наприклад:

```
x[1][2]=(a+b)/c;
```

або

```
int i,j,n=3,m=4;
for(i=0;i<n;i++)
  for(j=0;j<m;j++) printf("%f ",x[i][j]);
```

В останньому прикладі розглянуто код виведення на екран дванадцяти дійсних чисел – елементів двовимірного масиву.

Як вже зазначалося, при передачі масиву у функцію передається не вміст масиву, а лише його адреса (ім’я масиву).

Для передачі багатовимірного масиву у якості аргументу функції необхідно задати мінімум інформації, достатньої для перетворення індексних виразів в адреси пам’яті. Зокрема для двовимірного масиву достатньо передати кількість стовпців аби компілятор зміг самостійно обчислити адреси елементів на початку кожного рядка.

Визначимо, наприклад, символічні константи ROWS і COLS, що задають кількість рядків та стовбців відповідно у двовимірному масиві:

```
#define ROWS 5
#define COLS 8
```

Тепер можна оголосити, наприклад, наступну функцію AnyFn() з масивом у якості параметра:

```
void AnyFn(int data[ROWS][COLS]);
```

Можна навіть не зазначати кількість рядків:

```
void AnyFn(int data[][COLS]);
```

В обох випадках компілятор перетворить вираз data[n][m] в адресу.

Аби організувати передачу в одну й ту ж саму функцію масивів з різною кількістю рядків необхідно в оголошенні функції зазначити додатковий параметр, наприклад:

```
long Sum(int data[][COLS],int numRows);
```

Тут параметр numRows повідомляє функції AnyFn(), скільки рядків містить масив data. Глобальна константа COLS, як і раніше, інформує компілятор і функцію про кількість стовпців в одному рядку.

Так, наступна функція обраховує і повертає суму значень деякого двовимірного масиву цілих чисел:

```
long Sum(int data[][COLS],int numRows)
{
 long s=0;
 for(int r=0;r<numRows;r++)
   for(int c=0;c< COLS;c++) s+=data[r][c];
 return s;
}
```

Викликати цю функцію можна тепер для двовимірних масивів з різною кількістю рядків, наприклад, наступним чином:

```
printf(″Result1=%li\n″,Sum(matrix1,n1));
printf(″Result2=%li″,Sum(matrix2,n2));
```

де n1 і n2 – відповідна кількість рядків у двовимірних масивах matrix1 і matrix2, попередньо оголошених, наприклад, наступним чином:

```
…
#define COLS 8
…
```

```
const int n1=5,n2=7;
int matrix1[n1][COLS], matrix2[n2][COLS];
```

Матриці доволі часто застосовуються у математиці, особливо під час розв'язку задач, що містять системи алгебраїчних та диференційних рівнянь.

Контрольні запитання до теми 14:

1. Коли у програмах застосовуються масиви?
2. Який синтаксис оголошення масиву?
3. Що таке розмір та що таке розмірність масиву? Який масив вважається багатовимірним? Що таке числова матриця?
4. Як можна ініціалізувати масив?
5. Як здійснюється індексація елементів масиву?
6. Як здійснюється звертання до окремих елементів масиву?
7. Яким чином можна опрацьовувати одразу декілька елементів масиву?
8. Як зберігаються елементи багатовимірного масиву і який обсяг пам'яті вони займають?
9. Як здійснюється передача і опрацювання масивів у функціях?
10. Наведіть приклади практичного застосування одновимірних та багатовимірних масивів.

Тема 15. Алгоритми обробки масивів

При роботі з масивами найбільш розповсюдженими є задачі сортування (упорядкування).

Під час розв'язку задачі сортування зазвичай висувається вимога мінімального використовування додаткової пам'яті, що не припускає застосування додаткових (допоміжних) масивів. Упорядкування масиву – це зміна порядку розташування його елементів за певним критерієм. Наприклад, в числовому масиві це може бути упорядкування елементів за зростанням значень або за спаданням, для символьного масиву – це розташування елементів за абеткою тощо.

Для оцінки швидкодії різних алгоритмів сортування, використовують, як правило, два показники – кількість присвоєнь та кількість порівнянь.

Всі методи сортування можна поділити на дві великі групи – прямі (елементарні) та поліпшені (удосконалені) методи.

Прямі методи сортування по принципу, який лежить в основі методу, поділяються на три підгрупи:

– сортування вставкою (включенням);
– сортування вибором (виділенням);
– сортування обміном («бульбашкове» сортування).

Поліпшені методи сортування ґрунтуються на тих самих принципах, що і прямі, але використовують деякі оригінальні математичні ідеї для прискорення процесу сортування. Серед удосконалених методів сортування найчастіше використовуються методи Хоара, Шелла, пірамідальне сортування (за допомогою дерева), метод злиття тощо.

Прямі методи на практиці використовуються дуже рідко, бо мають відносно низьку швидкодію. Однак вони вдало ілюструють суть заснованих на них поліпшених методів. Окрім того, у деяких випадках (зазвичай при невеликій довжині масиву і (або) особливому початковому розташуванні його елементів) деякі із прямих методів можуть навіть перевершувати поліпшені методи.

Під час сортування вставкою масив умовно поділяється на дві частини – відсортовану і не відсортовану. Елементи із не відсортованої частини по черзі вибираються та вставляються у відсортовану частину так, щоб не порушити в ній наявну упорядкованість елементів. На початку роботи алгоритму у якості відсортованої частини масиву приймають тільки один перший елемент (елемент із нульовим індексом), а у якості не відсортованої частини – усі інші елементи.

Для реалізації наведеного методу можна запропонувати декілька алгоритмів, що будуть відрізнятися способом пошуку позиції вставки. Розглянемо один з них (упорядкування елементів за зростанням значень):

```
…
for(i=1;i<n;i++) /* n – кількість елементів у масиві */
  for(j=0;j<=i–1;j++)
    if (m[i]<m[j]) /* > – для упорядкування за спаданням */
    {
      t=m[i];
      for(k=i;k>=j+1;k--) m[k]=m[k–1];
      m[j]=t;
    }
…
```

На першому етапі алгоритму сортування вибором перший елемент масиву (елемент із нульовим індексом) по черзі порівнюється з наступними і при невиконанні поставленої умови упорядкування міняється з ними місцями. В результаті першого проходу на першому місці масиву буде знаходитися найбільший або найменший елемент (в залежності від умови упорядкування). На наступних етапах до другого, третього і т.д. аж до останнього елементів масиву буде застосовуватися аналогічний алгоритм обробки. При цьому відсортована частина масиву на кожному наступному етапі збільшується на один елемент та із обробки виключається.

Розглянутий алгоритм може мати наступний вигляд (упорядкування елементів за зростанням значень):

```
…
for(i=0;i<n–1;i++) /* n – кількість елементів у масиві */
  for(j=i+1;j<n;j++)
    if(m[j]<m[i]) /* > – для упорядкування за спаданням */
    {
      t=m[i];
      m[i]=m[j];
```

```
      m[j]=t;
    }
…
```

Під час сортування обміном зліва направо по черзі порівнюємо два сусідніх елемента, і якщо їх розташування не відповідає заданій умові упорядкування, то вони обмінюються місцями. Після першого проходу на останній позиції масиву буде стояти найбільший або найменший елемент (в залежності від умови упорядкування), тобто «спливла» перша «бульбашка». Кожний наступний прохід обмінів виконується аналогічно, але тільки до вже упорядкованої частини масиву.

Розглянемо алгоритм сортування обміном (упорядкування елементів за зростанням значень):

```
…
for(i=0;i<n–1;i++) /* n – кількість елементів у масиві */
  for(j=0;j<n–i–1;j++)
    if (m[j+1]<m[j]) /* > – для упорядкування за спаданням */
    {
      t=m[j];
      m[j]=m[j+1];
      m[j+1]=t;
    }
…
```

У кожному конкретному випадку, в залежності від умов поставленої задачі, програміст сам вирішує питання щодо застосування того чи іншого алгоритму сортування масивів.

Задача пошуку у масиві полягає у визначенні номерів певних елементів масиву або їх значень. Розрізняють задачі пошуку в упорядкованому та неупорядкованому масивах. В неупорядкованому масиві пошук можна здійснити лише за допомогою перегляду всього масиву (лінійний пошук). Якщо масив упорядкований (відсортований), то до нього може бути застосовано спеціальні методи пошуку, наприклад, бінарний пошук тощо.

Контрольні запитання до теми 15:

1. Що таке сортування масиву?
2. Які відомі методи сортування масивів і як оцінюється їхня швидкодія?
3. Навести алгоритм одного з прямих методів сортування масиву.
4. Як поліпшити швидкодію методу сортування масиву?
5. На яких засадах ґрунтуються поліпшені методи сортування масивів?
6. Як може виглядати функція, яка здійснює сортування масиву?
7. Яке формулювання задачі пошуку у масиві?
8. Що таке лінійний пошук?
9. Наведіть приклади відомих алгоритмів пошуку у масиві.

Тема 16. Символьні рядки

Символьний рядок – особливий вид масиву, елементами якого є значення ASCII-кодів символів. Останнім символом рядка повинен бути так званий нуль-символом (ASCII 0) – ознака кінця рядка.

Оголосити символьний рядок можна двома способами – як масив типу char або як рядковий покажчик на тип char, наприклад:

```
char s[128];
char *sPtr;
```

У першому випадку оголошується символьний масив s, що може зберігати від 1 до 128 елементів типу char з урахуванням останнього нуль-символу. Розмір пам'яті, що займає рядок s, суворо зафіксовано (у даному разі це 128 байт), у той час, як поточна довжина такого рядка може змінюватися в зазначених межах під час виконання програми.

Рядковий покажчик (див. друге оголошення) вказує на масив значень типу char, тобто є адресою місцезнаходження першого символу рядка.

Глобальні рядкові змінні оголошуються за межами усіх функцій програми, доступні із усіх її функцій та не потребують ініціалізації. Неініціалізовані локальні рядки доступні лише у своїх функціях та зберігають непередбачувані значення.

Символьні рядки можна ініціалізувати безпосередньо під час оголошення, наприклад:

```
char str1[80]= ″Borland C++″;
char str2[10]= ″″; /* літеральний нульовий рядок
*/ char str3[]=″My string″;
char str4[]={′s′,′t′,′r′,′i′,′n′,′g′,0};
char *sPtr=″string pointer″;
const char *str=″Літеральний рядок, довжиною
у\ декілька рядків екрану″;
```

Якщо граничну кількість символів рядка не вказано або рядок оголошено як рядковий покажчик (наприклад, str3, str4 або sPtr, str), то розмір такого рядка встановлюється компілятором автоматично за кількістю елементів-ініціалізаторів з урахуванням нуль-символу.

Нульовий рядок містить лише нуль-символ, а довжина такого рядка дорівнює нулю.

Доступ до окремих символів рядка здійснюється через індексні вирази, тобто так само, як і до елементів масиву, наприклад:

```
str1[9]=′\0′; /* str1=″Borland C″ */
str3[3]=′S′; /* str3=″My String″ */
sPtr[7]=′p′; /* sPtr=″string pointer″ */
```

З рядковими покажчиками можна працювати так само, як і зі звичайними покажчиками. Виходячи з цього, наступний оператор буде виконувати ті ж самі дії, що й останній оператор попереднього прикладу:

```
*(sPtr+7)=′p′; /* покажчик на сьомий символ рядка sPtr */
```

Необхідно зауважити, що у C/C++ оператор присвоєння можна застосовувати лише до рядкових покажчиків, наприклад:

```
sPtr=″new string pointer″;
```

Для присвоєння значень рядкам, оголошеним як символьні масиви, необхідно застосовувати спеціальні бібліотечні функції копіювання рядків (табл. 16.1).

Для введення символьного рядка застосовуються функції **scanf()** та **gets()**, прототипи яких знаходяться у заголовному файлі stdio.h, наприклад:

```
scanf(″%s″,s1);
gets(s2);
```

За допомогою функції scanf() можна ввести рядок до першого введеного пробілу, символу табуляції або натискання клавіші Enter. Функція gets() дозволяє ввести рядок, що містить будь-які розділові символи, проте не захищає користувача від введення зайвої кількості символів, ніж обумовлено під час оголошення рядка.

Для виведення символьного рядка застосовуються функції **printf()** та **puts()**, прототипи яких знаходяться у заголовному файлі stdio.h, наприклад:

```
printf(″%s″,s1);
puts(s2);
```

У разі використання динамічної пам’яті (купи) для роботи з рядковими покажчиками необхідно скористатися спеціальними функціями **malloc()** або **calloc() –** для виділення та **free() –** для вивільнення пам’яті, з прототипами у заголовному файлі stdlib.h, наприклад:

```
#define MAXSTR 128
…
char *p;
p=(char *)malloc(MAXSTR);
…
free(p);
```

В програмах C++ можна застосовувати також альтернативні оператори керування пам’яттю, а саме **new** – для виділення та **delete** – для вивільнення

пам’яті у купі.

Таблиця 16.1 – Стандартні функції опрацювання символьних рядків

Функція	Призначення	#include
atof()	– перетворення рядка у дійсне число типу double	stdlib.h
atoi()	– перетворення рядка у ціле число типу int	stdlib.h
atol()	– перетворення рядка у довге ціле число типу long	stdlib.h
itoa()	– перетворення цілого числа у рядок	stdlib.h
ltoa()	– перетворення довгого цілого числа у рядок	stdlib.h
ultoa()	– перетворення беззнакового довгого цілого числа у рядок	stdlib.h
stpcpy()	– копіювання рядка в рядок	string.h
strcat()	– додавання рядка до рядка (конкатенація)	string.h
strchr()	– пошук заданого символу, починаючи з початку рядка	string.h
strcmp()	– порівняння двох рядків	string.h
strcmpi()	– порівняння двох рядків без урахування регістру	string.h
strcpy()	– копіювання рядка в рядок	string.h
strcspn()	– пошук підрядка, що не містить задані символи	string.h
strdup()	– копіювання рядка у динамічну пам’ять	string.h
stricmp()	– порівняння двох рядків без урахування регістру	string.h
strlen()	– визначення довжини рядка	string.h
strlwr()	– перетворення у рядку великих літер у малі	string.h
strncat()	– додавання частини рядка до рядка (конкатенація)	string.h
strncmp()	– порівняння частини двох рядків	string.h
strncmpi()	– порівняння частини двох рядків без урахування регістру	string.h
strncpy()	– копіювання частини рядка в рядок	string.h
strnicmp()	– порівняння частини двох рядків без урахування регістру	string.h
strnset()	– заміна частини символів рядка заданим символом	string.h
strpbrk()	– пошук у рядку першого будь-якого символу іншого рядка	string.h
strrchr()	– пошук заданого символу, починаючи з кінця рядка	string.h
strrev()	– змінення порядку розташування символів на зворотній	string.h
strset()	– заміна усіх символів рядка заданим символом	string.h
strspn()	– пошук підрядка, що містить задані символи	string.h
strstr()	– пошук заданого підрядка	string.h
strtod()	– перетворення рядка у дійсне число типу double	stdlib.h
strtok()	– розкладання рядка на окремі підрядки	string.h
strtol()	– перетворення рядка у довге ціле число типу long	stdlib.h
strtoul()	– перетворення рядка у беззнакове довге ціле число	stdlib.h
strupr()	– перетворення у рядку малих літер у великі	string.h

Рядки, як й масиви, можна передавати у функції (див. тему 14).

Контрольні запитання до теми 16:

1. Що таке символьний рядок?
2. Які відомі типи рядкових змінних?
3. Що таке глобальні та локальні рядкові змінні?
4. Як оголошуються та ініціалізуються рядкові змінні різних типів?
5. Що таке загальна та поточна довжина рядка?
6. Як здійснюється доступ до окремих символів рядка?
7. Як присвоюються значення рядковим змінним?
8. За допомогою яких функцій вводяться та виводяться рядки?
9. Яке призначення має заголовний файл string.h?
10. Які основні стандартні функції застосовуються для обробки рядків?
11. Яким чином покажчики використовуються для обробки рядків?
12. Як використовується динамічна пам'ять для роботи з символьними рядками?

Змістовий модуль 4. **Складні структури даних**

Тема 17. Структури та об’єднання

Структури формують оболонку навколо одного чи декількох логічно пов’язаних між собою компонентів одного чи різних типів (у тому числі масивів, інших структур тощо). Ці компоненти називають полями або елементами.

Існує декілька різних способів оголошення структур та змінних структурного типу.

Для оголошення шаблону структури застосовується ключове слово **struct**, наприклад:

```
struct dateStruct
{
  char day; /* день */ char
  month; /* місяць */
  unsigned int year; /* рік
*/ };
```

де dateStruct – тег (ім’я) структури; day, month і year – список її полів.

Поля структури розташовуються у пам’яті саме в тій послідовності, що й в оголошені шаблону. Тип поля може бути простим або складеним. Імена полів в одному шаблоні не можуть збігатися, проте вони можуть збігатися з іменами інших змінних програми або полів інших структур.

Оголошення структурної змінної date здійснюється в один з наступних способів:

```
struct dateStruct date; /* через посилання на оголошений шаблон */
```

або

```
struct dateStruct /* одночасно з оголошенням шаблону */
{
```

```
  char day;
  char month;
  int year;
} date;
```

або

```
typedef struct dateStruct /* за допомогою декларації typedef */
{
  char day;
  char month;
  int year;
} DateStruct;
DateStruct date;
```

Структурні змінні можна ініціалізувати під час оголошення, не порушуючи послідовність та сумісність типів полів в шаблоні, наприклад:

```
DateStruct date={22,3,1992};
```

Для звертання до елементів структури застосовується крапкова нотація (операція «крапка») або операція «стрілка», наприклад:

```
date.day=22;
date.month=3;
(&date)–>year=1992;
```

У разі, коли поле структури є вкладеною структурою, то для звертання до її полів операція «крапка» застосовується кількаразово, відповідно до глибини вкладення.

Структура не може оголошувати в якості поля структури іншу структуру свого ж власного типу. Водночас така вкладена структура може бути покажчиком з довільним типом, у тому числі й покажчиком на структуру (навіть свого типу).

Після оголошення масиву структур, наприклад, наступним чином:

```
typedef DateStruct Dim[n];
Dim m;
```

де n – наперед задана кількість елементів масиву, звертання до полів такої структури даних здійснюється за допомогою рівнозначних операцій «крапка» або «стрілка», наприклад:

```
x=m[1].day;
y=(m+1)–>month;
```

Можна присвоїти значення однієї структурної змінної іншій за умови,

якщо обидві змінні мають спільний шаблон. Так, наприклад, після оголошення:

```
DateStruct date1,date2;
```

наступний оператор виконує копіювання інформації із структури date2 в структуру date1:

```
date1=date2;
```

Порівняння структур між собою безпосередньо не є припустимим, проте дозволяється порівняння їх окремих членів.

Об'єднання дають змогу записувати в одну і ту ж ділянку пам'яті дані різних типів і розмірів. Об'єднання практично ідентичні структурам, але на відміну від останніх, елементи в об'єднанні перекривають один одного, розташовуючись за однією адресою, а розмір об'єднання визначається розміром найдовшого його поля.

Оголошення об'єднання подібне до оголошення структури за винятком ключового слова **union** замість struct. Так само, як і для структур, шаблон об'єднання можна оголошувати окремо або одночасно з оголошенням змінних, наприклад:

```
typedef unsigned char
Byte; union word
{
  unsigned int w; /* перекриває lobyte і hibyte
  */ struct
  {
    Byte loByte; /* перекриває перший байт w */
    Byte hiByte; /* перекриває другий байт w */
  } bytes;
} uw;
```

Приклад демонструє об'єднання word, що забезпечує корисну можливість для доступу до окремих байтів та слів у довгих цілих значеннях. Оскільки компоненти об'єднання записуються в спільній пам'яті, то поле w перекривається байтами структури з власними полями loByte і hiByte. Якщо, наприклад, далі у програмі виконати оператор присвоєння uw.w=0x1234, то uw.bytes.loByte і uw.bytes.hiByte приймуть відповідно значення 0x34 і 0x12.

Для звертання до елементів об'єднання також застосовується крапкова нотація (операція «крапка») або операція «стрілка».

На зразок структур, об'єднання можна ініціалізувати під час оголошення, задаючи сумісне значення лише для першого елементу, наприклад:

```
union word uw={0x1234};
```

Таке присвоєння є одночасним присвоєнням значень іншим елементам об’єднання. Це твердження є слушним лише для об’єднань з елементами однакового розміру. В інших випадках такі ініціалізації є безкорисними.

Отже, в конкретний момент часу можна визначити значення лише одного з полів об’єднання, фактично знищивши значення інших.

До змінних з типом об’єднання можуть застосовуватись такі ж самі операції, що й до структурних змінних, наприклад, присвоєння (копіювання об’єднання).

Оскільки мова С інтерпретує структури та об’єднання як звичайні змінні, то й дозволяється їх передавати у функції або повертати у якості значення результату виконання функції. Також можна передавати у функції значення окремих полів структури (об’єднання).

Оголосимо, для прикладу, та одразу ініціалізуємо структурну змінну, яка визначає певні координати x і y на площині:

```
typedef struct coordinate
{
  int x;
  int y;
} Coordinate;
Coordinate r={5,10};
```

Але ж зручніше написати функцію, яка приймає значення координат:

```
Coordinate Fun(int a,int b)
{
  Coordinate rTemp;
  rTemp.x=a;
  rTemp.y=b; return
  rTemp;
}
```

після чого можливі наступні оператори:

```
Coordinate r;
r=Fun(5,10);
```

Наступний приклад демонструє передачу структур у якості аргументів параметрам функції:

```
int CmpCoordinate(Coordinate r1,Coordinate r2)
{
  return ((r1.x==r2.x)&&(r1.y==r2.y));
}
```

Функція CmpCoordinate() викликається для порівняння певних координат у тому місті програми, де передбачається логічний вираз.

Контрольні запитання до теми 17:

1. Що таке структура та об’єднання? Яка між ними різниця?
2. Як оголошуються та ініціалізуються структури та об’єднання?
3. Що таке тег та поле структури (об’єднання)?
4. Як здійснюється доступ до елементів структури (об’єднання)?
5. Що таке бітові поля?
6. Як зручніше оголосити масив структур?
7. Які операції можуть застосовуватися до структур (об’єднань)?
8. Як передати структуру (об’єднання) або окреме їх поле у функцію?
9. Чи може бути структура (об’єднання) результатом функції?

Тема 18. Розвинуті прийоми застосування функцій користувача [3–6] (*тема для самостійного вивчення*)

Тема 19. Обмін даними з файлами

Файл у мові C/C++ – абстрактне поняття, яким позначається сукупність даних на зовнішньому носії інформації або певний термінальний пристрій (наприклад, клавіатура, дисплей, принтер тощо). У першому випадку здійснюється прямий (блочний), а у другому – послідовний (побайтовий) файловий обмін.

Високорівневий обмін даними у С-програмах є уніфікованим завдяки системі буферизації. Послідовність байтів, що надходить від певного файлу (логічного пристрою) або передається до файлу, називається потоком. Проміжна область оперативної пам’яті, через яку відбувається обмін потоками між даними програми і файлами, називається буфером.

Для здійснення обміну даними між програмою та файлом необхідно насамперед створити відповідний потік у програмі, пов'язаний безпосередньо з цим файлом, тобто відкрити файл для доступу (читання або запису):

```
FILE *f;
f=fopen(fileName,fileMode);
```

де fopen() – стандартна (stdio.h) функція відкриття існуючого (або створення нового) фізичного файлу на ім’я fileName з заданим режимом обміну fileMode та його зв’язування з файловою змінною f.

Параметр fileName задає ім’я файлу, записане за правилами відповідної операційної системи, наприклад:

```
char fName1[]="student.txt";
char *fName2="D:\\MYFILES\\pr1.cpp";
```

Параметр fileMode задає один з наступних режимів відкриття файлу:

Режим	Двійковий	Текстовий	Опис
″r″	″rb″	″rt″	– відкриття файлу для читання
″w″	″wb″	″wt″	– відкриття або створення файлу для запису
″a″	″ab″	″at″	– відкриття файлу для доповнення даними

Прим. Режим без додаткового символу b або t є текстовим за замовчанням.

Для режиму ″r″ передбачається, що файл, призначений для відкриття, вже існує. Режим ″w″ дозволяє створити новий або перезаписати старий файл с таким самим ім’ям.

До ключової літери кожного з режимів можна додати знак «+», а саме: ″r+″, ″w+″, ″a+″ або ″rb+″, ″wb+″, ″ab+″ тощо, після чого відкриті файли можна буде використовувати як для читання, так і для запису.

Функція fopen() у разі успішного відкриття потоку повертає адресу (покажчик) створеної структури FILE, або повертає порожній покажчик NULL, якщо структуру створити неможливо (невірне ім’я файлу, хибний шлях тощо), наприклад:

```
…
char fName[]=″file.txt″;
FILE *f;
if((f=fopen(fName, ″r″))==NULL)
{
 printf(″Error: File %s not
 found″,fName); exit(1); /* stdlib.h */
}
… /* опрацювання відкритого файлу
*/ fclose(f);
…
```

Після нормального завершення програми всі відкриті потоки автоматично закриваються, а їх буфери вивільнюються, проте коректніше буде застосування для цього стандартної функції **fclose()** (див. останній приклад). Після успішного виконання ця функція повертає нуль, а у разі помилки – макроконстанту **EOF** - (End Of File – кінець файлу).

Читання і запис текстових файлів. Існують два фундаментальні засоби обробки текстових файлів: посимвольний і рядковий.

Основними функціями посимвольного зчитування з файлових потоків, відкритих для читання у текстовому режимі (″rt″), є стандартні функції **getc()** і **fgetc()**. У разі успішного виконання ці функції повертають зчитаний з потоку символ або EOF (у разі досягнення кінця файлу або неможливості зчитування даних).

Для посимвольного запису у файлові потоки, відкриті для запису у текстовому режимі (″wt″), використовуються стандартні функції **putc()** і **fputc()**.

Наприклад, наступний оператор відображає на екрані вміст довільного текстового файлу, зв’язаного у програмі з файловою змінною f1, та посимволь-

но копіює цей файл в інший, зв'язаний зі змінною f2:

```
while((c=fgetc(f1))!=EOF)
{
  putchar(c);
  fputc(c,f2);
}
```

Оскільки текстові файли організовано у вигляді символьних рядків, то для підвищення швидкодії алгоритмів роботи з ними більш придатним буде їх рядкове опрацювання. Для цього застосовуються стандартні функції **fgets()** (для читання) і **fputs()** (для запису) рядків у файли, відкриті для опрацювання у відповідному текстовому режимі.

Функція fgets() потребує у якості параметрів: рядкову змінну (буфер); кількість символів (включаючи завершальний нуль-символ), що необхідно прочитати у цей буфер; змінну файлового потоку, відкритого для читання у текстовому режимі. Після успішного виконання ця функція повертає покажчик на зчитаний з файлу рядок або NULL – у разі помилки.

Функція fputs() потребує лише два параметри – рядкову змінну і змінну файлового потоку, відкритого для запису у текстовому режимі. Після успішного виконання ця функція повертає ціле додатне значення або EOF – у разі помилки.

Наступний фрагмент демонструє рядкове копіювання вмісту одного текстового файлу в інший:

```
#define SIZE 255
…
FILE *f1,*f2;
char buffer[SIZE];
…
while(fgets(buffer,SIZE,f1)!=NULL) fputs(buffer,f2);
…
```

Необхідно зауважити, що рядки, розмір яких у вхідному файлі буде більшим за SIZE, «урізатимутся».

Форматне введення/виведення даних по відношенню до текстових файлів здійснюється за допомогою функцій **fscanf()** і **fprintf()** відповідно. Ці функції працюють аналогічно консольним функціям scanf() і printf(), наприклад:

```
fscaf(fin,″%i″,&x);
fprintf(fout,″%5.2f\n″,y);
```

але у якості першого параметру записується відповідний текстовий потік.

Читання і запис двійкових файлів здійснюється за допомогою стандартних функцій блочного обміну даними **fread()** і **fwrite()** відповідно.

Функція fread() потребує наступні чотири параметри: адресу буфера (змінної або масиву змінних), що приймає дані з файлового потоку; розмір у байтах одного елементу даних, що зчитуються з файлу; кількість таких елементів; змінну файлового потоку, відкритого для читання у двійковому режимі (″rb″). Функція повертає кількість реально зчитаних елементів або нуль (у разі помилки або коли досягнуто кінця файлу).

Функція fwrite() потребує наступні чотири параметри: адресу буфера (змінної або масиву змінних), що передає дані до файлового потоку; розмір у байтах одного елементу даних, що записуються у файл; кількість таких елементів; змінну файлового потоку, відкритого для запису у двійковому режимі (″wb″). Функція повертає кількість реально записаних елементів.

Наступний фрагмент програми демонструє блочне читання даних (″rb″) з файлу f1 довільного розміру у буфер-масив, побайтове опрацювання елементів цього масиву і подальший їх блочний запис (″wb″) у інший файл f2:

```
…
while(res=fread(&buffer,1,sizeof(buffer),f1))
{
  for(i=0;i<res–1;i++) … /* опрацювання масиву buffer
  */ fwrite(&buffer,1,res,f2);
}
…
```

Для розв’язку деяких задач обміну даними виникає необхідність у застосуванні функцій позиціювання потоків:

fseek() – встановлює покажчик поточної позиції файлу;
rewind() – встановлює покажчик поточної позиції файлу на його початок;
ftell() – повертає значення покажчика поточної позиції файлу;
fgetpos() – зберігає значення покажчика поточної позиції файлу;
fsetpos() – відновлює значення покажчика поточної позиції файлу, збереженого раніше fgetpos().

Більшість файлових функцій повертають результат, за яким можна перевірити коректність виконаної файлової операції. Проте встановити причину можливого збою можна за допомогою функцій аналізу помилок файлового обміну:

feof() – перевіряє досягнення кінця файлу;
ferror() – перевіряє ознаку помилки в операціях звертання до потоків;
perror() – виводить на екран інформацію про причини виникнення помилки в операціях звертання до потоків, наприклад:

```
if(ferror(f1)) perror(″Warning″);
```

Існує також багато інших файлових функцій. Наприклад, бібліотека dir.h містить спеціальні функції для роботи з файлами і каталогами.

Контрольні запитання до теми 19:

1. Що таке файл у мові програмування?
2. Що таке файловий потік і як він створюється засобами програми?
3. Які основні режими відкриття файлового потоку?
4. Як застосовується та працює функція fopen()?
5. Як закривається файловий потік?
6. Як засобами здійснюються посимвольне опрацювання текстових файлів?
7. Як засобами здійснюються рядкове опрацювання текстових файлів?
8. Якими засобами здійснюється форматний обмін даними між програмою і файлами?
9. Як засобами здійснюється опрацювання двійкових файлів?
10. Що таке позиціювання файлових потоків?
11. Які є засоби аналізу помилок файлового обміну?

Тема 20. Директиви препроцесора

(*тема для самостійного вивчення*)