КИЇВСЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ ІМЕНІ ТАРАСА ШЕВЧЕНКА

Іванов Є.О., Ліндер Я.М., Жереб К.А.

Основи мови програмування С++

Посібник першокурсника

Перший семестр

Київ – 2020

УДК 004.3/.4(075.8)
І-20

Рекомендовано до друку Вченою радою факультету комп'ютерних наук та кібернетики Київського Національного університету імені Тараса Шевченка (протокол № 7 від 10 лютого 2020 р.)

Рецензенти: О.І. Провотар, д-р фіз.-мат. наук; В.В. Пічкур, д-р фіз.-мат. наук.

І-20 Іванов Є.О., Ліндер Я.М., Жереб К.А.
Основи мови програмування С++: навчальний посібник. – К.: Логос, 2020. – 90 с.
ISBN 978-617-7631-24-7.

У посібнику подано матеріали для вивчення курсу «Основи програмування» студентами першого курсу факультету комп'ютерних наук та кібернетики спеціальності «Інженерія програмного забезпечення». Навчальний посібник містить загальні відомості про архітектуру комп'ютера, принципи роботи операційної системи та компілятора, а також основи синтаксису мови програмування С++. Крім того, у посібнику наведені приклади розв'язання типових задач та варіанти лабораторних робіт, рекомендованих до виконання студентами протягом першого семестру.

УДК 004.3/.4(075.8)

Підписано до друку 01.03.2020. Формат 60×84 1/16. Папір офс. № 1.
Гарнітура «Таймс». Ум. друк. арк. 5,1. Ум.-вид. арк. 6,3.
Наклад 100 прим. Зам.

Віддруковано в ТОВ-Видавництві «ЛОГОС» з оригіналів автора.
Свідоцтво про внесення суб'єкта видавничої справи Державного реєстру видавців, виготівників і розповсюджувачів видавничої продукції
серія ДК № 201 від 27.09.2000 р.
01030, Київ-30, вул. Богдана Хмельницького, 10, тел. 235-60-03

ISBN 978-617-7631-24-7

ЗМІСТ

Тема 1. Основи функціонування комп'ютера

Структура комп'ютера. Принципи зберігання даних та програм.

Для роботи будь-якого обчислювального пристрою (наприклад, персонального комп'ютера, сервера, мобільного пристрою, обчислювального кластера, суперкомп'ютера) необхідна правильна робота двох компонентів – 1) апаратної частини (hardware) і 2) програмного забезпечення (software).

Апаратна частина сучасного комп'ютера складається з багатьох взаємопов'язаних компонентів. Деякі типові компоненти зображено на Рис. 1

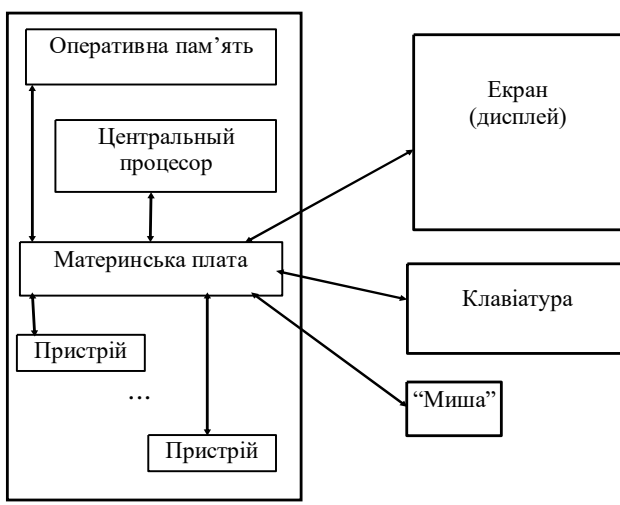


Рис. 1. Структура комп'ютера [7]

Центральний процесор (ЦП=Central Processing Unit, CPU), або **Мікропроцесор (МП)** — це, по суті, мініатюрна обчислювальна машина. Основними параметрами МП є: набір команд, розрядність, тактова частота.

Набір або система команд постійно вдосконалюється, з'являються нові команди, що замінюють серії найпримітивніших команд, — мікропрограми. На виконання нової команди потрібна менша кількість тактів, ніж на мікропрограму. Сучасні МП можуть виконувати до кількох сотень команд (інструкцій).

Розрядність показує, скільки двійкових розрядів (бітів) інформації обробляється (або передається) за один такт, а також скільки двійкових розрядів може бути використано у МП для адресації оперативної пам'яті, передачі даних тощо.

Кількість пам'яті, що адресується, або адресний простір, залежить від числа ліній шини адреси МП. Якщо цих ліній 20, то адресний простір становитиме 2^{20} = 1 Мбайт; якщо ліній 24, то — 2^{24} = 16 Мбайт тощо.

Тактова частота вказує, скільки елементарних операцій (тактів) МП виконує за секунду, вимірюється в мегагерцях (1 МГц = 1 000 000 Гц). Вона є лише відносним показником продуктивності МП. Через архітектурні відмінності МП у деяких з них за один такт виконується робота, на яку інші витрачають кілька тактів.

Системна (материнська) плата це велика друкована плата одного із стандартних форматів, яка несе на собі головні компоненти комп'ютерної системи: ЦП; оперативну пам'ять; кеш-пам'ять; комплект мікросхем логіки, що підтримують роботу плати, — чіпсет (chipset); центральну магістраль, або шину; контролер шини і кілька слотів, які служать для підключення до материнської плати інших плат (контролерів, плат розширення та ін.). Частина слотів у початковій комплектації ПК залишається вільною. У слоти іншої конфігурації встановлюють модулі оперативної пам'яті. Розрізняють постійну (постійний запам’ятовуючий пристрій - ПЗП), зовнішню та оперативну (оперативний запам’ятовуючий пристрій - ОЗП) пам’ять.

Постійна пам’ять (ROM — Read Only Memory) – це енергонезалежна пам’ять, яка використовується для тривалого зберігання сталої інформації та технічних програм ПК.

Зовнішня пам’ять До неї відносяться різні пристрої: жорсткий диск, компакт-диски, накопичувачі на флеш-пам’яті та інші.

Оперативна пам’ять (RAM— random access memory — пам'ять прямого доступу) – це енергозалежна пам’ять, яка використовується під час роботи комп’ютера. Характерною є велика швидкість виконання операцій. Після вимикання живлення інформація в пам’яті не зберігається. Оперативна пам'ять розподілена на елементарні області — байти. Кожний байт має свою адресу.

Оперативна пам’ять використовується для зберігання як даних, з якими працює програма, так і копії самої програми.

У більшості сучасних ПК розглядається наступна ієрархія пам'яті:

- Регістри процесора, організовані в регістровий файл — найбільш швидкий доступ (близько 1 такту), але розміром лише в декілька сотень або, рідко, тисяч байт.
- Кеш процесора 1го рівня (L1) — час доступу порядку декількох тактів, розміром в десятки кілобайт.
- Кеш процесора 2го рівня (L2) — більший час доступу (від 2 до 10 разів повільніше L1), близько половини мегабайта або більше.
- Кеш процесора третього рівня (L3) — час доступу близько сотні тактів, розміром в декілька мегабайт (у масових процесорах використовується з недавнього часу).
- ОЗП системи — час доступу від сотень до, можливо, тисячі тактів, але величезні розміри в кілька гігабайт, аж до десятків. Час доступу до оперативної пам'яті може змінюватись для різних його частин у випадку комплексів класу NUMA (з неоднорідним доступом в пам'ять).
- Дискове сховище — багато мільйонів тактів, якщо дані не були розміщені у кеші або буфері заздалегідь, розміри до декількох терабайт.

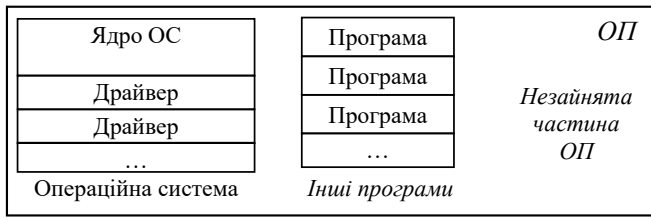


Рис. 2. Програми у оперативній пам'яті комп'ютера [7]

Програмне забезпечення (ПЗ) складається з багатьох програм, кожна з яких виконує свою функцію. Важливою частиною програмного забезпечення сучасного комп'ютера є операційна система (ОС). ОС надає базову функціональність, зокрема керує апаратними пристроями, завантажує програми, дозволяє виконувати багато програм одночасно та організовує перемикання між ними. Тому прикладні програми можуть фокусуватись лише на тій функціональності, яка потрібна користувачу.

Програми зберігаються у постійній пам'яті комп'ютера у вигляді одного чи декількох файлів. Зазвичай хоча б один з цих файлів є виконуваним файлом (executable). В ОС Windows більшість виконуваних файлів має розширення .EXE. В ОС на основі Unix у кожного файлу присутній спеціальний атрибут, що вказує, чи є цей файл виконуваним.

Компіляція та виконання програми.

Для виконання програми вона має бути завантажена в ОП (оскільки швидкість доступу до постійної пам'яті набагато повільніша). Це робить ОС. Завантажена копія програми, або процес, далі працює з якимись даними, які можуть завантажуватись з диску, або отримуватись від користувача через інтерфейс користувача, або бути отримані іншим шляхом (наприклад, завантажені з мережі). Деякі програми лише щось показують користувачу на екрані, але більшість програм також зберігає результати своєї роботи в постійну пам'ять та/або передають їх по мережі на зовнішні сервери.

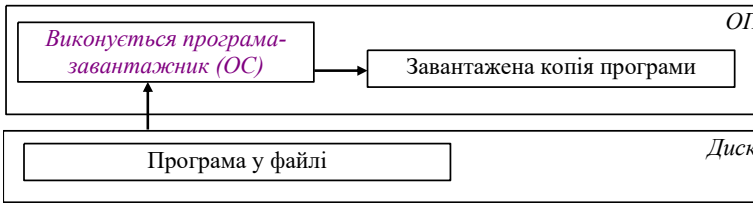


Рис. 3. Схема завантаження програми [7]

Виконувані програми мають містити інструкції для виконання на АЧ, зокрема на ЦП. Врешті будь-яка програма зводиться до набору інструкцій машинного коду. Проте перетворення програмного коду, написаного розробником, у машинний код, може виконуватись по-різному, залежно від обраної мови та технології програмування.

Трансляція – перетворення програми, представленої на якій-небудь мові програмування в еквівалентну форму на іншій мові.

Транслятори бувають двох видів: компілятори й інтерпретатори.

Компілятор перетворює початковий (вихідний) код програми в машинну мову, тобто мову нулів і одиниць. До отриманого коду підключаються стандартні процедури, використані програмістом. У результаті виходить працююча програма – її називають робочим кодом. Файли таких програм мають розширення .exe або .com і представляють собою машинний код. Компілятор – тип транслятора, що виконує перетворення всієї програми цілком з якої-небудь мови програмування на мову машинних кодів (абсолютний модуль) або близький до нього (об'єктний модуль).

Інтерпретатори обробляють текст не заздалегідь, а безпосередньо під час виконання програми. Інтерпретатори є програмами-посередниками, які читають команди з файлу і перекладають їх на мову процесора під час виконання програми. Інтерпретатор – тип транслятора, що здійснює покомандний (порядковий) переклад і виконання програми, написаної на якій-небудь мові програмування.

Існують мови програмування, що спочатку транслюють програму у байт-код, а потім виконують його.

Байт-код – машинно-незалежний код низького рівня, що генерується транслятором і виконуваний інтерпретатором. Більшість інструкцій байт-коду еквівалентні одній або кільком командам асемблера. Трансляція в байт-код займає проміжне положення між компіляцією в машинний код і інтерпретацією.

Байт-код називається так тому, що довжина кожного коду операції — один байт, але довжина коду команди різна. Кожна інструкція є однобайтовим кодом операції від 0 до 255, за яким слідують такі параметри, як регістри або адреси пам'яті. Це в типовому випадку, але специфікація байт-коду значно відрізняється в мовах програмування.

Програма на байт-коді зазвичай виконується інтерпретатором байт-коду (зазвичай він називається віртуальною машиною, оскільки подібний до комп'ютера). Перевага — в портивності, тобто один і той же байт-код може виконуватися на різних платформах і архітектурі. Ту ж саму перевагу дають мови, що інтерпретуються. Проте, оскільки байт-код зазвичай менш абстрактний, компактніший і більш «комп'ютерний» ніж початковий код, ефективність байт-коду зазвичай вища, ніж чиста інтерпретація початкового коду, призначеного для правки людиною. З цієї причини багато сучасних інтерпетованих мов насправді транслюють в байт-код і запускають інтерпретатор байт-коду. До таких мов відносяться Perl, PHP і Python. Програми на Java зазвичай передаються на цільову машину у вигляді байт-коду, який перед виконання транслюється в машинний код «на льоту» — за допомогою JIT-компіляції

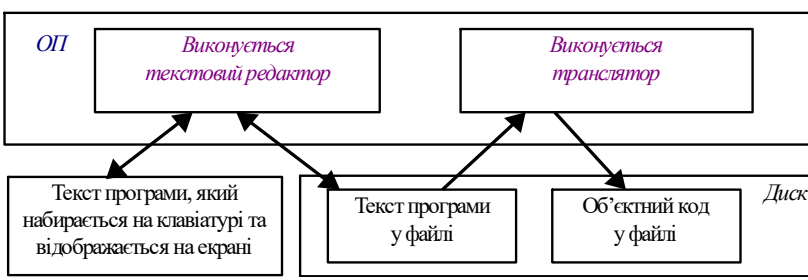


Рис. 4. Процес перетворення тексту програми на екрані у об'єктний код [7]

Компонувальник – програма, яка виконує компонування (англ. linking) — приймає на вхід один або кілька об'єктних модулів і збирає їх в один виконуваний модуль.

Для зв'язування модулів компонувальник використовує таблиці імен ідентифікаторів, створені компілятором в кожному з об'єктних модулів. Такі імена можуть бути двох типів:

- Певні або експортовані назви функцій та змінних, визначені в даному модулі і надані для використання іншим модулям
- Невизначені або імпортовані імена — функції та змінні, на які посилається модуль, але не визначає їх в середині себе

Робота компонувальника полягає в тому, щоб в кожному модулі конкретизувати посилання на невизначені імена. Для кожного імпортованого імені, визначення якого перебуває в інших модулях, згадування імені замінюється на його адресу.

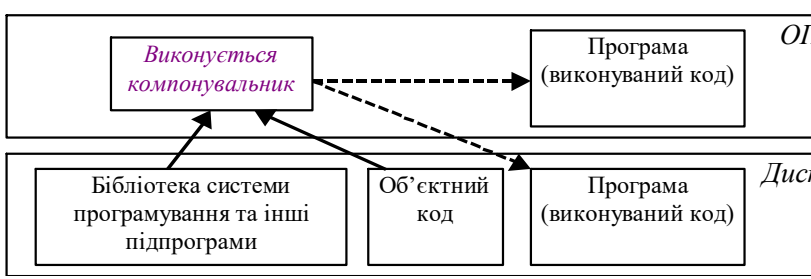


Рис. 5. Схема роботи компонувальника [7]

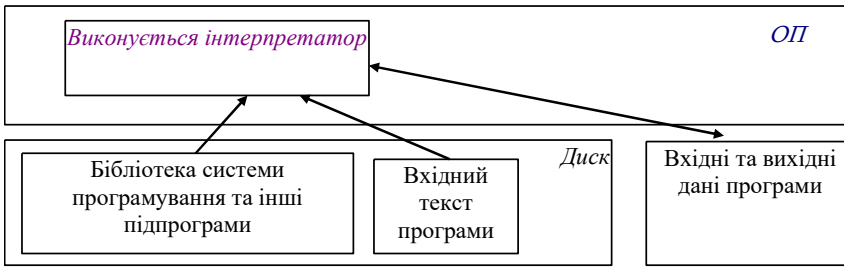


Рис. 6. Схема компіляції та виконання програми [7]

Результат компіляції програмного коду на C, C++, Python, Pascal, D та багатьох інших у машинний код можна інтерактивно одержати, використовуючи сайт Compiler Explorer https://godbolt.org/ . На цьому сайті можна ввести код високорівневою мовою (наприклад, C або C++) і подивитись машинний код, який генерує компілятор. Машинний код зазвичай подається у вигляді коду на асемблері (assembly language). Мова асемблер використовується через те, що вона однозначно відповідає машинному коду, але при цьому є більш читабельною, ніж набір нулів та одиниць. Сайт Compiler Explorer дозволяє подивитись, як насправді буде виконуватись написаний код, а також порівняти різні компілятори та різні налаштування компіляторів.

Тема 2. Історія та особливості мови C/C++

Іноді в ресурсах можна знайти згадки про мову програмування C/C++, хоча формально є дві різні мови C та C++. Мова C з'явилась раніше – перша версія в 1972 р., в 1978 р. вийшла книга Б. Кернігана та Д. Рітчі «Мова програмування C» (яка була неофіційним стандартом мови), в 1989 р. з'явився перший офіційний стандарт мови (ANSI C або C89).

Мова C розроблялась як мова високого рівня, проте з можливістю доступу до низькорівневих можливостей (робота з пам'яттю, вставки коду на асемблері, підказки компілятору). Тому мова C дозволяє писати код, який з одного боку є досить зрозумілим для розробника і дозволяє легше вносити зміни, знаходити помилки – а з іншого боку може досягати високої ефективності під час виконання. Також мова C орієнтована на підтримку різних платформ (архітектур процесорів, операційних систем) – правильно написаний код має компілюватись під будь-яку підтримувану платформу.

Незважаючи на свій поважний вік, мова C залишається актуальною для сучасних розробників – вона зазвичай займає високі позиції в рейтингу мов програмування https://www.tiobe.com/tiobe-index/ , за останні роки не опускаючись нижче другої позиції в рейтингу. Мова C використовується, зокрема, для системного програмування – наприклад, розробки операційних систем, драйверів апаратних пристроїв, написанні коду для пристроїв з обмеженими апаратними можливостями (embedded development), реалізацій інших мов програмування (зокрема Python, Perl, PHP). Існує багато бібліотек та фреймворків, які дозволяють використовувати мову C і для створення прикладних програм. Проте для прикладної розробки часто використовують інші, більш високорівневі мови програмування.

Мова C++ з'явилась пізніше, перші версії в 1985 р., перший офіційний стандарт в 1998 р. (C++98). В 2011 р. з'явився новий стандарт (C++11), який суттєво розширив можливості мови – як синтаксичні, так і стандартну бібліотеку. Після

цього нові стандарти мови С++ виходять регулярно, кожні три роки (версії мови С++14, С++17, С++20).

Основною метою створення мови С++ було розширити існуючу мову С, додавши в неї нові можливості, зокрема засоби об'єктно-орієнтованого програмування (ООП). Крім засобів ООП, були додані також інші можливості (простори імен, шаблони, посилання, обробка винятків, лямбда-вирази та інші). Це дозволяє писати більш високорівневий код, вводити нові абстракції – проте не поступаючись ефективністю виконання коду. Мова С++ є майже надмножиною мови С – тобто більшість програм мовою С або будуть компілюватись в С++ без змін, або вимагатимуть лише мінімальних змін. Зокрема, в коді на С++ можна використовувати бібліотеки для мови С – як стандартні, так і написані іншими розробниками. Для багатьох задач в стандартних бібліотеках присутні як засоби С, так і засоби С++ – зокрема це стосується консольного вводу/виводу, роботи з файлами, роботи з текстовими рядками, структур даних.

Мова С++ є досить популярною серед розробників, зазвичай входить в верхні 5 позицій рейтингу мов програмування https://www.tiobe.com/tiobe-index/ . Її часто використовують для розробки великих та складних програмних продуктів – зокрема, веб-браузерів, офісних продуктів, систем керування базами даних, систем для роботи з графікою, моделювання, ігрових систем. Багато серверних компонентів реалізовано з використанням С++ для досягнення максимальної ефективності коду. Мова С в рейтингу займає вищі позиції, ніж С++; це можна пояснити тим, що для системного програмування фактично немає альтернатив мові С, тоді як для прикладного програмування існує декілька популярних мов (С++, Java, C#, Python, JavaScript та інші).

Багато сучасних мов програмування засновані на мовах С/С++, зокрема використовують схожий синтаксис, оператори та конструкції керування. При цьому семантика мови може бути різною – використовуються різні підходи розробки, різні моделі пам'яті, мови підтримують різний рівень абстракції, деякі з цих мов компілюються в машинний код, інші в байткод, деякі є інтерпретованими (скриптовими). До таких С-подібних мов належать Java, C#, JavaScript, PHP, Swift,

Objective-C, Go, D, Rust та інші. Також більшість сучасних мов програмування дозволяють підключити певні компоненти (модулі, функції), написані мовами C/C++. Тому розробники можуть писати більшу частину коду на більш високорівневій мові, але для найбільш складних фрагментів використовувати бібліотеки, реалізовані на C/C++. Можливості мов C/C++, наближені до апаратної реалізації, дозволяють розробнику зрозуміти, як насправді виконується код на процесорі. Вивчення мов C/C++ , які належать до мов програмування з «жорсткою типізацією», привчає до певної дисципліни в роботі з даними й закладає якісні основи для оволодіння іншими мовами, навіть з суттєво іншими принципами побудови. Тому знати мови програмування C/C++ корисно всім сучасним розробникам – навіть тим, хто надалі планує писати код на інших мовах програмування.

Мови C/C++ підтримують багато механізмів написання структурованого коду. Тобто програмний код розбивається на невеликі фрагменти, які взаємодіють між собою. Це дозволяє простіше реалізовувати такі фрагменти, а також спрощує розуміння коду, виправлення помилок, внесення змін, реалізацію ефективного та безпечного коду. Мова C підтримує розбиття коду на функції, винесення фрагментів коду в окремі файли, користувацькі типи даних (структури, об’єднання, перелічення). Мова C++ підтримує всі ці можливості, і також додає нові можливості (класи, простори імен).

Тема 3.Директиви препроцесора

Препроцесор є складовою системи програмування C++, першою фазою компіляції. Інструкції препроцесору називають директивами. Вони повинні розміщуватись у окремих рядках й починатися символом # . Область дії директиви – від неї до кінця файлу.

Основні директиви препроцесора:

- #include – підключення файлу;
- #define , #undef – визначення підстановки та обмеження дії підстановки;
- #if , #elif , #else , endif , #ifdef , #ifndef – умовної компіляції.

Директива підключення файлу дозволяє включити в програму вміст вказаного файлу. Має два варіанти:

- #include <file> – для файлу з системної бібліотеки, пошук (якщо не вказаний шлях) здійснюється у стандартних каталогах системних файлів;
- #include “file” – пошук файлу здійснюється у поточному каталозі, а потім у стандартних каталогах системних файлів.

Приклади:

```
#include <cstdio>
#include “c:\Pr_C\Pr_1\myfile.h”
```

Директива підстановки дозволяє замінити заданий ідентифікатор, можливо параметризований, на вказану послідовність символів. Використовується для визначення:

- констант: #define ідентифікатор текст_підстановки;
- макросів: #define ідентифікатор(параметри) текст_підстановки;
- символів, що керують умовною компіляцією: #define ідентифікатор.

Обмежити область дії підстановки можна за допомогою директиви #undef, яка відміняє дію підстановки.

Приклади:

```
#define VERS 5
#define BEGIN {
#define END }
#define VS "Visual Studio"
#define MIN(x,y) (((x)<(y))?(x):(y))
#define SQR(x) ((x)*(x))
#define MMM
#define SIZE 100
float vect[SIZE];
#undef SIZE
```

Директиви умовної компіляції надають можливість вибірково компілювати частини коду програми. Можуть бути вкладеними. Використовують, наприклад, при налагодженні, а також для підтримки декількох версій програми для різних платформ.

Формат директиви:

```
#if константний_вираз
...
[#elif константний_вираз
...]
[#elif константний_вираз
...]
[#else
...]
#endif
```

Приклад 1:

```
#if VERS == 1
  #define INCFILE "vers1.h"
#elif VERS == 2
  #define INCFILE "vers2.h"
```

```
#elif VERS == 3
  #define INCFILE "vers3.h"
#else
  #define INCFILE "versN.h"
#endif
#include INCFILE
```

Приклад 2

```
#if 0
  int i, j;
  double x, y;
#endif

#if defined(MMM)
  ...
#else
  ...
#endif
#ifdef MMM          // еквівалентно if defined(MMM)
```

Приклад 3

```
#define TED 10
int main() {
  #ifdef TED
    printf("Hello, Ted\n");
  #else
    printf("Hello, ???\n");
  #endif
  #ifndef RALPH
    printf("A RALPH not defined\n");
  #endif
  return 0;
}
```

- Заголовочні файли (за традиціями) не повинні містити визначень функцій та даних (за винятком ситуацій, коли використовуються шаблони (templates)). Зазвичай містять об`яви типів, констант, функцій (прототипи), шаблонів, перелічень, просторів імен.
- Кількість фактичних параметрів у виклику макросу повинна збігатись з кількістю формальних параметрів. Відсутність круглих дужок може привести до проблем з неправильним порядком обчислень, оскільки препроцесор на здійснює контролю синтаксису.
- Директиви умовної компіляції можуть бути вкладеними.
- Імена макросів рекомендують записувати великими буквами, щоб візуально відрізняти від імен функцій та змінних.
- Макроси й константи є спадком від C. В C++ рекомендують ними не зловживати. Рекомендують константи визначати користуючись const , enum , а функції – вбудованими функціями, шаблонами.

Задачі

- Реалізувати математичну функцію $y = x^2$ у вигляді макросу, функції, вбудованої функції. Порівняти результати звернень з різними аргументами, в тому числі з аргументом a++ .
- Порівняти на конкретних прикладах результати звернень до макросів #define N (-2) та #define N -2 . Пояснити отримані результати.
- Визначити власні макроси для дій з числовими, логічними, символьними даними, наприклад, знаходження мінімального значення для двох чисел, суми за модулем два для двох логічних даних, модуля комплексного числа. Перевірити працездатність макросів.

Тема 4. Скалярні типи даних

Будь-які дані у пам'яті комп'ютера зберігаються у вигляді послідовності біт. Проте для виконання операцій біти інтерпретуються як різні типи даних.

Мета програми – обробка даних. Данні різних типів представляються та обробляються по різному. Кожна константа, змінна, результат обчислення виразу, або функції повинні мати певний тип.

Тип даних – фундаментальне поняття, визначає:

- множину значень;
- операції та функції, що застосовують до значень;
- внутрішнє представлення даних.

Типи даних C++:

- ***основні*** (базові): цілі, дійсні, символьні, логічні;
- ***похідні*** (складені): масиви, структури, перелічення, об`єднання, посилання, покажчики, функції.

Важливими типами даних є числові типи.

Більшість мов програмування підтримують окремо цілі числа (integers) та дійсні числа з плаваючою комою (floating point real numbers)

Дійсні числа, що подаються в комп'ютері, за будь-якого подання утворюють обмежену підмножину множини раціональних чисел.

Ключові слова (визначають тип)***:***

- **int** (цілий);
- **char** (символьний);
- **wchar_t** (розширений символьний);
- **bool** (логічний);
- **float** (дійсний);
- **double** (дійсний з подвійною точністю).

Специфікатори типу (уточнюють представлення)***:***

- **short** (короткий);
- **long** (довгий);
- **signed** (знаковий);
- **unsigned** (беззнаковий).

Розмір типу **int** та інших цілих типів не визначається стандартом, а залежить від компілятора та цільової архітектури процесора:

- 16-розрядний – 2 байти (в сучасних комп'ютерах майже не зустрічається, використовувався у версіях Windows раніше за Windows 95);
- 32-розрядний – 4 байти;
- 64-розрядний – 4 байти.

Специфікатори додатково уточнюють:

- **short** – оптимізований під розмір, не менше 2 байти;
- **long** – не менше 4 байти (починаючи з C++11, може вказуватись двічі - **long long**, тоді не менше 8 байтів).

Розмір скалярних типів для систем різної архітектури наведено в таблиці:

Тип	**Повне ім'я**	**16-бітна архітектура, гарантований мінімум**	**32-бітна архітектура**	**64-бітна архітектура (Windows)**	**64-бітна архітектура (Unix-подібні)**
short	signed short int	16 біт (2 байти)	16 біт (2 байти)	16 біт (2 байти)	16 біт (2 байти)
int	signed int	16 біт (2 байти)	32 біт (4 байти)	32 біт (4 байти)	32 біт (4 байти)
long	signed long int	32 біт (4 байти)	32 біт (4 байти)	32 біт (4 байти)	64 біт (8 байтів)
long long	signed long long int	64 біт (8 байтів)	64 біт (8 байтів)	64 біт (8 байтів)	64 біт (8 байтів)

У 16-розрядній системі типи **int** та **short int** є еквівалентними. У 32-розрядній системі типи **int** та **long int** є еквівалентними.

- **signed** – знакові типи, старший біт – знаковий (0 відповідає знаку +, 1 відповідає знаку –);

- **unsigned** – беззнакові типи, старший біт – частина коду числа, тому максимальне значення більше. Розмір такий самий, як у відповідного signed типу.

За замовчуванням– **signed** та **int**, ці ключові слова можна пропускати, ***скорочення*** – short = signed short int, long = signed long int, signed = int = signed int, unsigned = unsigned int .

size_t – спеціальний беззнаковий цілий тип, який гарантовано вмістить максимальний розмір будь-якого об'єкта в пам'яті, в тому числі структур або масивів. Правильно використовувати саме цей тип, наприклад, для індексів масивів (оскільки індекс типу int може не вмістити максимально можливий індекс, наприклад на 64-розрядних архітектурах)

Літерали, 16, 8, бінарні

char – зберігання окремих символів та невеликих цілих – займає 1 байт.

char – може бути signed (значення: від -127 до 128), або unsigned (значення від 0 до 255).

wchar_t – робота з набором символів (наприклад Unicode), де не достатньо 1 байту (як правило – 2 байти).

bool – значення: **false** та **true**. Займає 1 байт, хоча для збереження інформації достатньо 1 біт. Причина у тому, що сучасні комп'ютери можуть адресувати пам'ять лише з точністю до байта

Внутрішнє представлення **false** – 0. Інше значення інтерпретується - **true .**

При перетворенні до цілого типу **true** дає 1.

Дійсні типи зберігаються інакше, ніж цілі. Внутрішнє представлення складається з мантиси (точність) та порядку (діапазон):

- **float** – 4 байти: 1б – знак, 8б – порядок, 23б – мантиса (старша цифра мантиси завжди -1, не зберігається). Діапазон значень: 3.4e-38 - 3.4e+38;
- **double** – 8 байтів: 1б – знак, 11б – порядок, 52б – мантиса. Діапазон значень: 1.7e-308 - 1.7e+308;

- **long double** – зазвичай 10 байтів. Діапазон значень: 3.4e-4932 - 3.4e+4932.

За замовчуванням дійсні константи - double : 2e6, 1.82. Можна явно вказувати тип (f, F, l, L): 2e+6l , 1.82f

Інформацію про розмір пам`яті типу або змінної (в байтах) можна отримати операцією **sizeof**. Наприклад: sizeof(int) або int var=12345; size_t size = sizeof(var).

В стандарті ANSI діапазони значень не задаються (визначаються реалізацією), але передбачається:

sizeof(float) <= sizeof(double) <= sizeof(long double)

sizeof(char) <= sizeof(short) <= sizeof(int) <= sizeof(long)

Тема 5. Структури керування

В теорії програмування доведено, що програму для розв`язання довільної задачі можна скласти використовуючи тільки три структури керування, які називають: ***слідування***, ***розгалуження***, ***циклу*** (Бойм, Якопіні 1966 р.). Ці структури керування називають ще ***базовими конструкціями структурного програмування***.

Слідування – послідовне виконання двох або більше операторів.

Розгалуження – виконання або одного, або іншого оператора в залежності від виконання умов.

Цикл – задає багатократне виконання оператору.

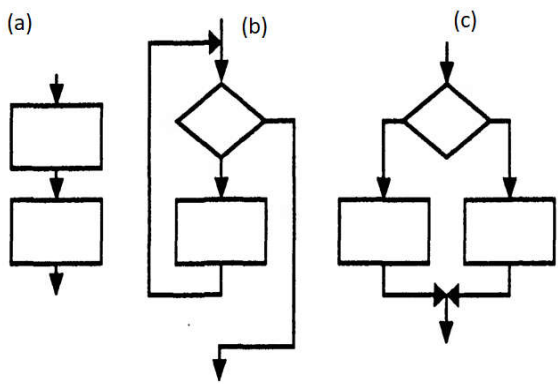


Рис. 7. Базові конструкції структурного програмування; (a) – слідування, (b) – цикл, (c) – розгалуження

Базові конструкції можуть довільним чином вкладатись одна в одну. Метою при використанні базових конструкцій є отримання програми з простою (прозорою) структурою. Такі програми легко читати, налагоджувати, змінювати.

Структурне програмування часто називають ще програмуванням без “go to”. Так зловживання використанням оператору передачі керування суттєво ускладнює відслідковування логіки роботи програми. Але з іншого боку існують ситуа-

ції, де поміркованe використання “go to” є виправданим й не створює проблем. Тому в мовах програмування остаточно не відмовляються від цієї конструкції, але суттєво обмежують її застосування.

В C++ існує декілька реалізацій для базових конструкцій: два варіанти розгалуження, три – циклу.

Блок (складений оператор) використовується щоб записати кілька операторів там, де за правилами мови може бути тільки один.

Загальний вигляд:

```
{
    <оператор1>;
        …
    <операторN>;
}
```

Умовний оператор if

Розгалуження процесу обчислень на два напрямки.

```
if (<вираз>) <оператор1>; [else <оператор2>;]
```

Оператор-перемикач switch-case

Розгалуження процесу обчислень на декілька напрямків.

```
switch (<вираз>) {
  case <константній вираз1> : <послідов_ операторів1>
  case <константній вираз2> : <послідов_ операторів2>
                                                …
  case <константній виразN> : <послідов_ операторівN>
  [default : <послідов_ операторів>]
}
```

Для виходу з switch використовується оператор break (або return). Якщо не писати цей оператор, то буде послідовно виконуватись код з кількох варіантів.

Вираз (селектор варіантів) – цілого типу, або такого, що приводиться до цілого.

Оператори циклу використовуються для багатократного повторення обчислень. Оператори циклу складаються з “тіла циклу” (оператори, що повторюються)

та “заголовку циклу” (перевірки умов, початкових установок, модифікації параметру). Один прохід тіла циклу – називають ітерацією.

Різновиди операторів циклу:

- з “передумовою” (while);
- з “постумовою” (do-while);
- з “параметром” (for).

Цикл з передумовою

```
while (<вираз>) <оператор>;
```

Тіло циклу виконується, поки вираз є істинним. Вираз обчислюється *до* виконання відповідної ітерації циклу. Тобто можлива ситуація, коли вираз одразу є хибними, і тоді тіло циклу не виконується жодного разу.

Цикл з постумовою (do-while)

```
do <оператор> while (<вираз>);
```

Тіло циклу виконується, поки вираз є істинним. Вираз обчислюється *після* виконання відповідної ітерації циклу. Тобто завжди буде виконана хоча б одна ітерація циклу.

Довільний цикл **do** можна замінити на цикл **while**.

```
 do <оператор> while (<вираз>);
```

// виконується так само як

```
<оператор>; while (<вираз>) <оператор>;
```

Цикл з параметром (for)

```
for ( [<вираз-ініціалізація>]; [<вираз-умова>];
        [<вираз-модіфікація>]) <оператор>;
```

// виконується так само як

```
<вираз-ініціалізація>;
while (<вираз-умова>)
  {
     <оператор>;
     <вираз-модіфікація>;
   }
```

Типова форма циклу for (цикл з лічильником)

```
    for(i=0;i<N;i++) { <тіло циклу>} // і приймає значення 0,1,2, ..., N-1
    for(i=A;i<B;i++) { <тіло циклу>} // і приймає значення A,A+1,A+2, ..., B-1
    for(i=0;i<N;i+=step) { <тіло циклу>} // з кроком step
    for(i=N-1;i>=0;i--) { <тіло циклу>} // і приймає значення N-1, N-2, ...,1,0
```

У стандарті С++11 є додаткова форма циклу, яка є придатна для циклічного аналізу та обробки послідовностей (vector, list), або статичних масивів.

```
    for(i:items) { <тіло циклу>} // і приймає значення з items
```

Оператори передачі керування:

- break – завершує виконання найближчого оператора-перемикача (switch), або операторів циклу (while, do, for). Виходить з циклу, виконуються наступні оператори після циклу.
- continue – завершує поточну ітерацію найближчого (внутрішнього) циклу й змушує розпочати наступну ітерацію циклу (while, do, for).
- return <result>– обчислює значення виразу-результату, з перетворенням до типу результату функції, пересилає обчислене значення в місце виклику, передає керування до точки продовження. Якщо функція не повертає результат (наприклад void do_something() {…}, то оператор return пишеться без результату: if(condition) {return;}.
- goto <label> – здійснює безумовну передачу керування на оператор помічений міткою. Не рекомендується зловживати оператором безумовного переходу, в більшості випадків можна реалізувати код з використанням інших операторів, і такий код буде легше читати та розуміти. В деяких ситуаціях використання goto є доцільним, наприклад для виходу з глибоко вкладених внутрішніх циклів, або для переходу на завершення функції з різних частин коду функції.

Нескінченний цикл:

```
while(true) { <тіло циклу>};
for ( ; ; ) { <тіло циклу>};
```

Зазвичай у випадку нескінченного циклу, в тілі циклу має бути механізм виходу з використанням break / return / goto .

Тема 6. Статичні масиви та рядки

Масив – структура даних (статична), що складається з фіксованої кількості елементів, одного типу.

Змінна масив:

- складається з елементів (компонент);
- всі елементи одного типу;
- кількість елементів фіксується в означенні;
- кожен елемент ідентифікується номером (індексом);
- займає неперервну область пам`яті при розміщенні;
- доступність елемента (час) не залежить від номеру.

Математика, аналогії:

- вектори, матриці;
- функція A: <індекси> $\rightarrow$ <елементи> .

Визначення:

<тип> <ім`я>[<кількість елементів>] [<ініціалізатор>];

Пам`ять:

<об`єм пам`яті> = sizeof(<тип>) * <кількість елементів>

Звернення:

<ім`я> [<номер елемента>], або через покажчики та адресну арифметику.

Зауваження:

- нумерація елементів (індекси) починаються з 0;
- для елементів глобальних масивів автоматично – ініціалізація 0;
- відсутній контроль на вихід індексу за межі.

Багатовимірні масиви

Визначення:

<тип> <ім`я>[<кількість1>] …[<кількістьN>] [<ініціалізатор>];

Звернення:

<ім`я> [<номер1>] …[< номерN>]

Зауваження:

- при розташуванні швидше змінюється останній індекс (“рядками”);
- для ініціалізації значення вказуються згідно з порядком розташування;
- при зверненні кожний індекс у власних дужках.

Символи та рядки

Змінні для збереження символів:

```
char ch1, ch2;
ch1 = ‘w’;  //буква w
ch2 = 119;  //буква w
```

Можна присвоювати – число (код символу), або символ в ‘’. Використання “w” буде означати не один символ w, а рядок з двох символів: w та нульовий символ. Можна в ‘’ вказувати спеціальні символи (керівні послідовності).

Основні операції з символами – порівняння.

С++ пропонує два способи роботи з рядками:

- спадщина С – “рядки у стилі С”, основані на використанні масивів;
- оснований на бібліотечному класі **string**.

Переваги першого – ефективність. Переваги другого – зручність роботи, захищеність даних.

Рядок (“у стилі С”) – послідовність символів, що зберігаються у послідовних байтах пам`яті. Іншими словами, рядок це масив символів, що закінчується *нуль символом* (з кодом 0 – записується – ‘\0’). Розмір масиву повинен враховувати *нуль символ*.

```
char str[7];
char str1[7] = {'S', 't', 'r', 'i', 'n', 'g', '\0'};
char str2[7] = "String";
char str3[] = "String";
```

Але:

```
char str[7];
str = "String";  //Помилка !!!
char *str = "String"; //покажчик на рядкову константу
```

Рядок літерал може містити спеціальні символи (керівні послідовності). Символи “ та \ у рядку-літералі супроводжують символом \.

```
char str1[] = "Group \"IS\"\n";
char str2[] = "C:\\temp\\new\\file.txt"; //правильно
char str3[] = "C:\temp\new\file.txt"; //неправильно
```

значення str3*:* C:<табуляція>emp<перевод рядка>ew<перевод сторінки>ile.txt

Доступ до символів рядка

Змінна – адреса першого символу рядка. Доступ або через індекси, або через покажчики та адресну арифметику.

```
char str[] = "String";
//еквівалентні
cout << str[1] << endl;
cout << *(str + 1) << endl;
//можна змінювати
str[0] = 's';
*(str + 1) = 'T';
cout << str << endl;
```

Рядкова змінна - масив символів, тому операції присвоювання одного рядка іншому, порівняння рядків та інші операції над рядками не визначені. Це можна здійснювати або використанням циклу, або функцій бібліотеки.

Стандартна бібліотека надає можливості копіювання, порівняння, об`єднання рядків, пошуку підрядка, визначення довжини, та інші, а також містить функції введення рядків та окремих символів з клавіатури, або з файлу.

Тема 7. Покажчики та динамічні масиви

Покажчики (вказівники, pointer) призначені для збереження адрес областей пам`яті й дозволяють маніпулювати об`єктом, що розташований в цій пам`яті. В С++ розрізняють покажчики: на об`єкт, на функцію, на void. Масиви та покажчики в С++ дуже тісно пов`язані. Покажчики дозволяють виконати довільну операцію з масивами. Ім`я масиву розглядається як константний покажчик на початок відповідної області пам`яті. Покажчики використовуються в С++ дуже інтенсивно й не тільки для роботи з масивами. Одне з основних застосувань покажчиків полягає в роботі з динамічно створеними об`єктами: динамічними масивами, списками, деревами. Покажчики дозволяють передавати в функції “великі” об`єкти, а також отримувати результати обчислень.

Кожний покажчик асоціюється з деяким типом даних. Область пам`яті, що адресується покажчиком, інтерпретується як значення відповідного типу.

Визначення покажчика має вигляд <тип> *<ім`я змінної>;

Наприклад:

```
int *a,  b,  *c; //покажчики a, c та ціла змінна b
```

Покажчик може бути змінною, або константою, а також вказувати на змінну, або константу.

Приклади:

```
int i;  //ціла змінна
const int ci=1;  //ціла константа
int *pi;  //покажчик на цілу змінну
const int *pci;   //покажчик на цілу константу
int *const cp = &i;  //покажчик-константа на цілу змінну
const int *const cpc=&ci; //покажчик-константа на цілу константу
```

Ініціалізація покажчиків

При визначенні покажчика бажано здійснити його ініціалізацію. Способи ініціалізації:

1) Присвоювання адреси існуючого об`єкта:

   a. операцією &

```
int a = 5;  //ціла змінна a
int *p = &a; //покажчик - адреса змінної a
int *p (&a);  //теж саме, інший спосіб
```

   b. іншим ініційованим покажчиком

```
int *r = p;  //p отримав значення
```

   c. за допомогою масиву

```
int b[10]; //масив
int *p = b;  //присвоювання адреси початку масиву
```

2) Присвоювання адреси пам`яті в явному вигляді:

```
char *p = (char *)0xB80000000;
```

3) Присвоювання порожньої адреси:

```
int *pnt = nullptr;
int *pp = 0;
```

4) Виділенням динамічної пам`яті:

```
int *pnt = new int;
int *pp = new int (10);
int *qq = new int [10];
```

Останній спосіб вимагає явного звільнення пам`яті за допомогою операції delete

```
delete pnt; delete [] qq;
```

Можна описувати складні типи. Діють правила:

- За означенням (), [] мають однаковий пріоритет більший за пріоритет *, розглядаються зліва-направо.
- Якщо праворуч від імені [] – це масив, якщо () – функція.

- Якщо ліворуч від імені * - це покажчик на проінтерпретовану раніше конструкцію.
- Якщо праворуч) – потрібно застосувати правила для внутрішньої частини () , а потім переходити до зовнішньої частини.
- В останню чергу інтерпретується специфікатор типу.

Наприклад:

```
int *(*p[10])();  // ???
```

Операції з покажчиками

- * – розіменування,
- & – отримання адреси,
- new – виділення та delete – звільнення пам`яті:

```
*pi = abs(*pi);
```

- присвоювання:

```
pi = pnt;
```

- порівняння:

```
pi == pnt;
```

- арифметичні мають сенс при роботі з структурами послідовно розташованими в пам`яті (наприклад - масиви) :
  - додавання константи (pi + 5)
  - віднімання константи (pi - 2)
  - різниця покажчиків (pi - ri)
  - інкременту (++) (pi++) (++pi)
  - декременту (--) (pi--) (--pi)
  - фактично дії відбуваються з одиницями виміру sizeof(<тип>).

В результаті визначення масиву у змінній зберігається адреса його першого елементу. Ім`я змінної-масиву – покажчик на перший елемент. Тому звернення до елементів можливі як за індексами, так й за результатами “адресної арифметики”.

Наприклад:

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

наступні вирази еквівалентні:

```
arr[1];
*(arr + 1);
*(1 + arr);
1[arr];
```

Проте рекомендується використовувати лише перший варіант, оскільки він є більш зрозумілим.

Приклад роботи з масивами через покажчики:

```
const short size = 3;
int *p = 0;  int arr[size] = {1, 2 ,3};
p = arr;    p = &arr[0];  //еквівалентні
//перебір елементів
for (int i=0; i<size; i++) {
cout << *p << endl;  ++p;  }
//теж перебір елементів
for (int *q=arr; q<arr+size; ++q) {
cout << *q << endl; }
//теж перебір елементів
p = arr; i = size; while (i-- > 0) {cout << *p++;}
const short size = 3;
int *p = 0;  int arr[size] = {1, 2 ,3};
p = arr;    p = &arr[0];  //еквівалентні
//перебір елементів
for (int i=0; i<size; i++) {
cout << *p << endl;  ++p;  }
//теж перебір елементів
for (int *q=arr; q<arr+size; ++q) {
cout << *q << endl; }
//теж перебір елементів
p = arr; i = size; while (i-- > 0) {cout << *p++;}
```

Приклад бінарного пошуку у впорядкованому масиві:

```
int bin_search(int key, const int *arr, int count) {
 if (count < 1 || !arr) return -1;
 int beg = 0, end = count - 1, i = 0;
 while (beg <= end) {
   i = (beg + end) / 2;
   if (arr[i] == key) return i;
   if (arr[i] < key) beg = i + 1;
   else end = i - 1;
  }
 return -1;
}
```

Можна визначати масиви покажчиків:

<тип> *<ім`я масиву> [<кількість елементів>];

Наприклад:

```
int *p[3];
int x=10, y=20, z=30;
p[0] = &x;
p[1] = &y;
p[2] = &z;
cout << *p[0] << "  " << *p[1] << "  " << *p[2] << endl;
```

Використовуючи механізм покажчиків, є можливість динамічного виділення пам`яті і роботи з динамічними масивами:

```
<покажчик> = new <тип>[<кількість елементів>];
```

Звільнення пам`яті:

```
 delete [] <покажчик>;
```

При звільненні пам`яті кількість елементів не вказується. Відповідальним за повернення пам’яті є програміст. Приклад:

```
int main(){
 int n;
```

```
 int *p = 0,  *q = 0;
 cout << "n= ";   cin >> n;  cout << endl;
 p = new int [n];   q = p;
 cout << "array A: ";
 for (int i=0; i<n; i++)   cin >> *q++;
 cout << endl;     //q==?
 q = p;
 //виведення даних
 cout << "array A: ";
 for (int i=0; i<n; i++)  cout << *q++ << "  " ;
 delete [] p;  p = 0;
return 0; }
```

Потрібно враховувати, що при виділенні пам`яті може виникнути ситуація неможливості надати заявлений обсяг пам`яті. В сучасному стандарті C++ активується виключення bad_alloc , що визначено у файлі <new.h>.

Багатовимірні динамічні масиви

При створенні багатовимірних динамічних масивів в операції new потрібно вказувати всі виміри (лівий вимір може бути змінною). Наприклад:

```
int ind1 = 5;
int **parr = (int **) new int [ind1][10];
```

Більш безпечний та універсальний спосіб виділення пам`яті, без вказаного обмеження на виміри:

```
int ind1, ind2;
cout << "Size array n*m :";
cin >> ind1 >> ind2;
int **a = new int *[ind1];
for (int i = 0; i < ind2; i++)
  a[i] = new int[ind2];
```

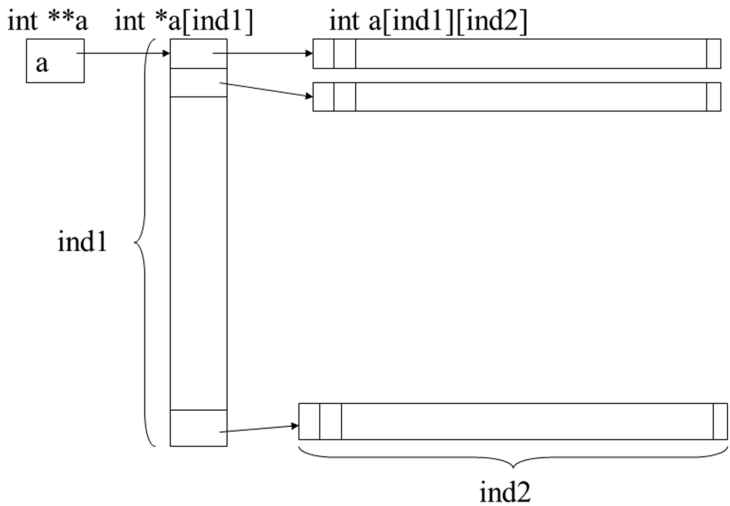


Рис. 8. Схема виділення пам'яті при створенні двовимірного динамічного масиву.

Зауваження та підсумки

- Варто дотримуватись розглянутих правил використання спефіцікаторів при визначенні покажчиків та масивів покажчиків (наприклад, int *p[10] – масив з 10 покажчиків).
- Використовуючи явні механізми для виділення пам`яті, не забувати про необхідність звільнення виділеної пам`яті. Враховувати, що при звільненні відповідний покажчик не змінює свого значення.
- При виділення пам`яті враховувати можливість відсутності потрібного обсягу вільної пам`яті.
- Не порушувати правил "адресної арифметики" та порівнянь покажчиків.
- Не виходити покажчику за межі пам`яті об`єкту.
- Не допускати ситуацій з "втраченою пам`яттю".

Задачі

1) Пошук заданого значення у масиві з цілих:
    - масив одновимірний;
    - масив одновимірний впорядкований;
    - масив двовимірний;
    - масив двовимірний впорядкований;
2) Перевірити, чи є задане число паліндромом.
3) Записати функцію для “бульбашкового” сортування масиву.
4) Записати функцію для визначення чи є квадратна матриця симетричною.
5) Записати функцію для транспонування квадратної матриці.
6) Для заданої дійсної матриці знайти індекси всіх її “сідлових точок” (елементи, що є одночасно найменшими у рядку й найбільшими у стовпчику, або навпаки.)
7) Визначити чи є ціла квадратна матриця “магічним квадратом” (суми елементів у всіх рядках та стовпчиках однакові).
8) В місті М діє р-ічна система числення, а номери тролейбусних квитків містять 2k розрядів. Квиток вважається щасливим, якщо сума перших k розрядів дорівнює сумі останніх k розрядів. Вхід: Значення p та k. Вихід: Кількість щасливих квитків.
9) Обчислити масив A[100], який містить 100 перших елементів (у зростаючому порядку) множини M, що визначається наступним чином:
    - 1 належить M;
    - якщо x належить M, то y=2*x+1 та z=3*x+1 належать M;
    - ніякі інші числа не належать M.
10) Побудувати перші N натуральних чисел дільниками яких є тільки числа 2, 3, 5.
11) Задані A, B, N. Знайти всі натуральні числа, що не перевищують N, які представляються у вигляді суми довільної кількості доданків, кожен з яких A або B.

Тема 8. Функції у мові C/C++

Більшість коду в програмі на C/C++ пишеться у вигляді функцій. Функція складається з сигнатури (ім'я функції, імена та типи параметрів, тип результату) та тіла функції (код, який реалізує функцію). Приклади функцій:

```
int square(int num) {
    return num*num;
}

double max(double x,double y) {
    if(x>y) {
        return x;
    } else {
        return y;
    }
}

void print_repeat(const char* message,
                  unsigned repeat_count,
                  const char* delimiter = " ") {
    for(unsigned i = 0; i < repeat_count; i++) {
        std::cout<<message<<" ";
    }
    std::cout<<std::endl;
}
```

Функція повинна бути оголошена до її виклику. Можна зробити за допомогою прототипу функції – заголовок функції, що закінчується ; і повідомляє про наявність функції з відповідними типами параметрів та результату. Прототип функції не замінює означення функції.

Традиція: розміщення прототипів на початку програми, а означень функцій після головної функції. Порядок слідування означень – довільний. У прототипі можна не вказувати імен параметрів.

Функція може не повертати результат – в цьому випадку для результату використовується спеціальний тип даних **void**. Функція може викликати інші функ-

ції, в тому числі ту саму функцію (рекурсивні виклики). Під час виклику функції розробник вказує значення параметрів, які будуть передані у функцію. Під час визначення функції можна вказати значення деяких параметрів за замовчуванням – в цьому випадку можна буде не передавати значення цих параметрів під час виклику функції (але можна і передати інші значення, ніж параметри за замовчуванням). Будь-яку функцію (як визначену з типом результату void, так і визначену з іншим типом результату) можна викликати без отримання результату.

Функції на C/C++ можуть повертати лише один результат. В деяких випадках розробнику необхідно, щоб функція повертала декілька результатів. Це можна реалізувати двома способами: 1) повернути більш складний тип даних, який містить в собі декілька значень (наприклад структуру або масив), або 2) передати декілька параметрів за вказівником, або за посиланням (лише C++), так щоб функція могла записати в них нові значення.

Передача за посиланням дозволяє повернути з функції відразу кілька значень. Також передача параметрів по посиланню є більш ефективною при передачі дуже великих об'єктів, оскільки в цьому випадку не відбувається копіювання значень, а функція використовує сам об'єкт, а не його значення.

Приклад використання структури для отримання результатів із функції:

```
struct max_result {
     int index;
     int value;
};
max_result max_struct(int* array, int length) {
     int max_index = -1;
     int max_value = INT_MIN;
     for (int i = 0; i < length; i++) {
          if (array[i] > max_value) {
               max_value = array[i];
               max_index = i;
          }
     }
     return {max_index,max_value};
 }
```

Приклад використання вказівників для отримання результатів із функції:

```
    void max_pointer(int* array, int length, int* max_index,
int* max_value) {
        *max_index = -1;
        *max_value = INT_MIN;
        for (int i = 0; i < length; i++) {
            if (array[i] > *max_value) {
                *max_value = array[i];
                *max_index = i;
            }
        }
    }
```

Приклад використання посилань для отримання результатів із функції:

```
    // лише для C++
    void max_reference(int* array, int length, int& max_index,
int& max_value) {
        max_index = -1;
        max_value = INT_MIN;
        for (int i = 0; i < length; i++) {
            if (array[i] > max_value) {
                max_value = array[i];
                max_index = i;
            }
        }
    }
```

В програмі на C/C++ є спеціальна функція main(), яка називається також точкою входу в програму. Ця функція викликається першою, і всі інші функції мають викликатись з неї – або напряму, або через декілька інших функцій. Функція main() має повертати результат типу int – код результату виконання. Результат виконання 0 свідчить про успішне виконання програми, будь-який ненульовий результат – про помилку. Функція main() може або не приймати параметрів, або

приймати два параметри int main(int argc, char* argv[]). В цьому випадку параметр argc описує кількість аргументів, які передали під час виклику програми, а масив argv – самі значення цих параметрів. Наприклад, якщо скомпілювати програму у виконуваний файл з іменем hello.exe, то можна буде викликати цю програму наступним чином: hello.exe C++ world. При цьому значення параметрів будуть наступними: argc ==3, argv[0] ==”hello.exe”, argv[1] ==”C++”, argv[2] ==”world”. Тобто першим в масиві буде ім’я виконуваного файлу.

В C/C++ виклики функції можуть знаходитись в коді лише після визначення відповідної функції. Якщо це з якихось причин неможливо або небажано, можна використати попередню декларацію (forward declaration). При цьому сигнатури функцій (без тіл) пишуться до їх виклику, а повна реалізація (сигнатура та тіло функції) може бути розташовано де завгодно. Приклад:

```
double calculate(double value, int steps);

void run() {
    //...
    result = calculate(x,steps);
    //...
}

double calculate(double value, int steps) {
    //...
    return result;
}
```

Невеликі програми мовами C/C++ можуть містити весь код в одному файлі. Але зі збільшенням розміру коду це стає незручно, тоді доцільно розділити код на кілька файлів. Файли коду C/C++ бувають двох видів: файли з кодом (source files, зазвичай використовують розширення .c для мови C та розширення .cpp для мови C++) та файли-заголовки (header files, зазвичай використовують розширення *.h для C та C++, або розширення *.hpp лише для C++). Файли-заголовки містять лише декларації – визначення типів, сигнатури функцій. Реалізація функціональнос-

ті міститься у файлах з кодом. Файли можуть включати інші файли (зазвичай лише заголовки) з використанням директиви препроцесора **#include**.

Для використання можливостей стандартної бібліотеки зазвичай необхідно включити відповідні файли-заголовки. Які саме файли-заголовки потрібно включати, можна подивитись в документації, зокрема на сайті https://cppreference.com .

Мова C++ додала можливість використання просторів імен (namespace) для уникнення колізій імен функцій, типів, глобальних змінних між різними бібліотеками чи компонентами. Більшість імен зі стандартної бібліотеки C++ знаходяться в просторі імен std. Для використання цих функцій, типів, змінних треба вказувати простір імен. Це можна робити, використовуючи повністю кваліфіковані імена, наприклад **std::cout**, **std::string**, **std::vector**. Можна використати інструкцію **using namespace std;** яка дозволяє використовувати всі імена зі стандартної бібліотеки без префіксу. Але так робити не рекомендується, оскільки таким чином додається дуже багато імен зі стандартної бібліотеки, деякі з яких використовують розповсюджені англомовні слова, тому збільшується шанс колізії імен і зникають переваги від використання просторів імен. Краще використовувати інструкції на зразок **using std::cout;** які дозволять писати без префіксів лише ті імена, які часто використовуються в коді. Цю інструкцію можна писати як на початку файлу з кодом, так і в окремій функції.

Для того, щоб код програми був зрозумілим, щоб було легше знаходити помилки та вносити зміни, треба давати функціям, типам, змінним, файлам зрозумілі імена. Імена певних сутностей в коді мають коротко описувати призначення цих сутностей і бути зрозумілими в контексті їх використання. Чим ширшим є контекст використання, тим більш зрозумілим має бути ім’я. Тому можна використовувати досить короткі імена для локальних змінних, які використовуються лише в одному блоці коду – проте варто використовувати більш зрозумілі імена для глобальних змінних, функцій, типів даних. Не варто використовувати занадто короткі імена, наприклад з однієї літери (за винятком деяких особливих випадків, наприклад індекси циклів). Не варто зловживати скороченнями, оскільки іноді їх можна розуміти неоднозначно.

Варто структурувати код програми таким чином, щоб він виглядав одноманітно і в ньому було легко знайти потрібні фрагменти. Наприклад, можна використовувати таку структуру для головного файлу з кодом:

```
<підключення файлів>
<декларації глобальних змінних>
<декларації функцій, класів, …>
int main([<параметри>])
{
     <інструкції>;
     return 0; // успішне завершення виконання
}
< визначення функцій, класів, …>
```

Тема 9. Типи даних, що визначає програміст

Структури

У реальних задачах інформація, що потребує обробки, може мати досить складну структуру. Для її представлення використовують типи даних побудовані на основі простих типів даних, масивів, покажчиків.

Мова С++ дозволяє програмісту визначати свої типи даних та правила роботи з ними. Історично для основних таких типів склалися свої власні назви, але їх можна об`єднати під назвою ***типів, що визначає програміст***.

Структури (записи) – гнучкий структурований тип даних, що дозволяє поєднувати в єдине ціле елементи різних типів.

Структури – використовують для адекватного представлення реальних об`єктів, де елементи (поля) визначають різні характеристики об`єктів.

Використання:

- таблиці із різнотипною структурованою інформацією;
- файли, бази даних;
- динамічні структури даних тощо;

```
struct [ім`я типу]
{
<тип_1> <елемент_1>;
     <тип_2> <елемент_2>;
     ...
     <тип_N> <елемент_N>;
} [<список змінних>];
```

Елементи структури називають ***полями***.

Типи полів можуть бути різними, але не типом самої структури (хоча може бути покажчиком на нього). Розмір пам`яті не обов`язково дорівнює сумі розмірів полів (за рахунок вирівнювання на границі слова).

Приклад:

```
struct date {
```

```
  int day;
  int month;
  int year;
};
date dt_brt[10], *pdt;
```

Стандартом С дозволяється декларувати і неіменовані структури. Недоліком такого підходу є неможливість подальшого використання типу структури.

```
struct {
float x, y;
} position;
```

Приклади ініціалізації структур

```
struct person {
 char name[20];
 date b_d;};
 person man = {"Bill", {7, 11, 1935}};
struct complex {
  float re, im;
}  compl [2][3] = {
{{1,1}, {1,2}, {1,3}},
{{2,1}, {2,2}, {2,3}}
};
```

Операції зі структурами:

- Для змінних одного структурного типу – *операція присвоювання* (поелементне копіювання).
- Структури можна передавати у функцію через параметри та повертати як значення функції.

Доступ до полів структури здійснюється операціями:

- . (вибору) через ім`я змінної-структури;
- -> через покажчик на структуру.

До структур можна застосовувати операцію &.

Приклад доступу до полів структури

```
dt_brt[1].day = 15;
pdt -> month = 6;
if (pdt->year == 2012 && pdt->month == 2)  pdt->day = 29;
compl [1][2].re = 2.4;   compl [1][2].im = 5.55;
```

Приклад передачі структури у функцію

```
struct complex {
  float re, im;
};
complex add_complex (complex c1, complex c2){
 complex c3 = c1;
 c3.re += c2.re;
 c3.im += c2.im;
 return c3;
}
...
complex x, y, z;
...
z = add_complex(x, y);
cout << "z = x+y  = " << z.re << " + " << z.im << "i" << endl;
```

При використанні структур в якості параметрів функцій потрібно враховувати можливості параметрів-значень та параметрів-посилань (Тема 8). Якщо функція не змінює значення полів структури її можна передавати як параметр-значення. Якщо функція повинна змінювати значення полів структури передавати структуру як параметр-посилання.

Бітові поля

Особливий вид полів структури. Використовується для щільної упаковки невеликих даних. При визначенні поля вказується його розмір у бітах.

```
struct options {
 bool centerX:1;
 bool centerY:1;
 unsigned int shadow:2;
 unsigned int palette:4;
} pd;
pd.centerX = 0; pd.centerY = 0;
pd.shadow = 3; pd.palette = 7;
```

Бітові поля можуть бути довільного цілого типу. Доступ здійснюється традиційним чином через ім`я поля. Фактично такі поля надають доступ до окремих частин байтів – бітів пам`яті. Отримати адресу такого поля не можна. Не можна створювати масиви бітових даних.

Ім`я поля може бути відсутнім. Такі поля використовуються для вирівнювання на апаратні границі. Доступ до самого такого поля відсутній. Дозволяють заощадити пам`ять, але слід враховувати, що операції з окремими бітами реалізуються суттєво менш ефективно ніж операції з байтами та словами. Такі програми значною мірою є машино-залежними.

Об`єднання

Змінні типу об`єднання (суміш) можуть набувати значень кількох різних типів. Можна трактувати їх як структури, всі поля яких розташовані за однією адресою, в одній спільній ділянці, достатній для збереження найбільшого за розміром поля.

Фактично вони надають можливість доступу до однієї й тієї самою ділянки пам`яті за допомогою змінних різних типів. В об`єднанні допускаються й структури з бітовими полями, що надає можливості доступу до окремих бітів ділянки. Зрозуміло що в кожен момент часу у такій змінній зберігається лише одне значення. Відповідальність за правильність використання покладається на програміста.

Приклади

```
union onefrom {
  int int_val;
  long long_val;
  double double_val;
};
onefrom var;
…
var.int_val = 15;
cout << var.int_val;
…
var.double_val = 1.38;
cout << var.double_val;
```

```
struct widget { //структура реєстру
  char brand[20];
  int type;
  union id {  //залежить від типу
    long id_num;
    char id_char[20];
  } id_val;
};
...
widget prize;
...
if (prize.type == 1) cin >> prize.id_val.id_num;
else cin >> prize.id_val.id_char;
```

Перелічення

Спосіб створення іменованих констант, зв`язаних між собою. Сприяє наочності програмних рішень, а також забезпечує контроль з боку системи програмування за значеннями відповідних змінних. ***Формат:***

```
enum [<ім`я_типу>] {<список_констант>} [<змінні>];
```

<ім`я_типу> – визначається, якщо потрібно окремо визначати змінні цього типу. Змінні можуть приймати значення тільки з <список_констант>. Константи з

<список_констант> – цілі. Можуть бути ініціалізовані звичайним чином, або за умовчанням у першої константи значення – 0, у кожної наступної значення – на 1 більше ніж у попередньої. Операції: присвоювання, порівняння. При виконанні арифметичних операцій перетворюються на цілі.

Приклади

```
enum spectrum {red, orange, yellow, green,
blue, violet, indigo, ultraviolet};
```

Визначили новий тип spectrum, а також іменовані константи з значеннями 0 – 7.

```
spectrum band; //змінна з типом spectrum
band = blue;
band = 2000; //не припустимо: значення змінної band обмежені 8 переліченими.
int color = blue; //припустимо
band = spectrum(3); //приведення 3 до типу spectrum
band = spectrum(40003); //невизначеність
```

```
enum paytype {CARD, CHECK};
paytype ptype;
union payment {
  char card[25];
  long check;
} info;
... //отримання значень info та ptype
switch (ptype) {
  case CARD: cout << "Оплата карткою: " << info.card; break;
  case CHECK: cout << "Оплата чеком: " << info.check; break;
  }
```

Визначення типів

Можна створювати додаткові імена для існуючих та нових типів.

Формат:

```
typedef <тип> <нове_ім`я> [<розмірність>];
```

Приклади

```
typedef int LENGTH;
typedef unsigned int UINT;
typedef char Msg[100];
typedef struct {
char fio[30];
int date, code;
double salare;
} Worker;
typedef int LENGTH;
typedef unsigned int UINT;
typedef char Msg[100];
typedef struct {
  char fio[30];
  int date, code;
  double salare;
} Worker;
LENGTH n, m, k;
UINT i, j;
Msg str[10];
Worker staff[100];
```

```
typedef struct tnode {
  char *word;
  struct tnode *left;
  struct tnode *right;
} TREENODE, *TREEPTR;
TREEPTR talloc(){
```

```
  return ((TREEPTR) calloc(1, sizeof(TREENODE)));
}
TREEPTR p, t;
p = talloc();
```

Задачі

1) Реалізувати тип даних, що задає матеріальну точку на площині у гравітаційному полі, що направлене вниз. Реалізувати операції: ввід , вивід, обчислення положення точки через t секунд.
2) Реалізувати тип даних, що задає раціональне число у нескоротному запису. Реалізувати операції: ввід числа, вивід числа, добуток, сума та різниця двох чисел
3) Реалізувати тип даних, що задає комплексне число у тригонометричній формі. Реалізувати операції: ввід числа, вивід числа, добуток, сума та різниця двох чисел, піднесення у степінь.
4) Реалізувати тип даних, що задає комплексне число у декартовому запису. Реалізувати операції: ввід числа, вивід числа, добуток, сума та різниця двох чисел, піднесення у степінь.
5) Реалізувати тип даних, що задає многочлен від однієї змінної. Реалізувати операції: ввід, вивід, сума, різниця, добуток двох многочленів, частку та остачу від ділення многочленів. Реалізувати похідну та невизначений інтеграл від многочлена. Реалізувати обчислення значення многочлену у точці.
6) Реалізувати тип даних, що задає цілочисельну арифметику за модулем N. Операції: ввід N, ввід числа, сума, різниця, множення, ділення.
7) Реалізувати тип даних, що задає вектор у N-вимірному просторі. Операції: ввід, вивід, сума, скалярний добуток, обчислення норми.
8) Реалізувати тип даних, що задає відрізок на площині. Операції: ввід, вивід, поворот та масштабування відносно центру, обчислення довжини.

Тема 10. Файли

Мова С++ підтримує дві системи введення/виведення. Перша – спадок С. Друга – власна, об`єктно-орієнтована. Дозволяється поєднувати в одній програмі.

До переваг об`єктно-орієнтованої системи введення/виведення можна віднести: простоту використання у простих випадках, можливості перевизначення для власних класів. Але вона вимагає розуміння об`єктів та класів.

Введення/виведення в "*стилі С*": зручне при форматованому обміну, дозволяє побачити й зрозуміти роботу з файлами, потрібне для розуміння та підтримки накопиченого програмного забезпечення.

Для використання функцій введення/виведення необхідно підключити відповідний заголовочний файл:

```
#include <cstdio>  //або <stdio.h>
```

Наявні функції бібліотеки дозволяють працювати й з стандартними вхідним та вихідним потоками: stdin – клавіатура, stdout – екран, stderr – помилки.

Наприклад:

- getchar(), putchar() – читання та запис символу;
- gets(), puts() – читання та запис рядка;
- scanf(), printf() – форматоване введення та виведення даних

Обробка інформації, що зберігається у вигляді файлу передбачає наступні дії:

- визначення змінної – файлового покажчика;
- відкривання та закривання потоку;
- введення-виведення (символів, рядків, форматованих даних, порцій даних певної довжини);
- аналіз можливих помилок операцій введення-виведення;
- керування буферізацією потоку (розміром буферу);
- керування буферним покажчиком.

Потоки бувають двох типів: текстові та двійкові.

Кожний потік має керівну структуру типу FILE , що містить усю необхідну інформацію для роботи з ним. Змінна, що буде представляти потік визначається як покажчик на структуру типу FILE.

Наприклад:

FILE *fp;

Змінна fp зображує потік у подальшій роботі з файлом.

Опис типу FILE , а також прототипи більшості функцій, макросів та констант файлової системи містяться у заголовному файлі <cstdio> (та <stdio.h>).

Потік можна відкрити для читання або/та запису в текстовому або двійковому режимі. Функція відкриття потоку має формат:

FILE *fopen(const char *filename, const char *mode);

Якщо відкривання було успішним, функція повертає *покажчик файлу*, що містить всю необхідну для роботи з потоком інформацію; інакше функція повертає NULL.

Параметр filename – задає ім`я файлу у вигляді рядка в *стилі С*.

Параметр mode – визначає режим відкривання файлу.

- “r” – відкриття існуючого файлу для читання;
- “w” – створення нового файлу для запису (якщо файл з таким ім`ям існує – він перезаписується);
- “a” – відкриття існуючого файлу для додавання в його кінець нової інформації;
- “r+” – відкриття існуючого файлу для читання й запису;
- “w+” – створення нового файлу для читання й запису (перезаписується файл з таким ім`ям, якщо існує);
- “a+” – відкриття існуючого файлу для читання та додавання в його кінець нової інформації.

Режим може містити символи t – текстовий, або b – двійковий. По умовчанню передбачається текстовий режим (t).

Потік закривається або при завершенні програми, або явно функцією fclose: int fclose(FILE *);

Якщо потік, заданий параметром-покажчиком файлу, був відкритий для запису, то перед закриттям у файл записуються данні, що містяться у буферах потоку. У випадку успішного закриття функція повертає 0, інакше – EOF.

Рекомендується завжди явно закривати потоки, що відкриті для запису, щоб уникнути втрати даних.

int fflush(FILE *); - примусово скидає буфер у файл.

Максимальна кількість одночасно відкритих файлів визначається макросом FOPEN_MAX.

Введення/виведення можна здійснювати у вигляді послідовності байтів, символів, рядків, або з використанням форматних перетворень. Існують відповідні функції бібліотеки.

Операції введення/виведення завжди виконуються починаючи з поточної позиції потоку, що визначається *покажчиком потоку*. *Покажчик потоку* встановлюється при відкритті на початок, або кінець (в залежності від режиму відкриття) й змінюється автоматично при виконанні операцій введення/виведення. Поточне положення *покажчика потоку* можна отримати функціями ftell, fgetpos й задати функціями fseek, fsetpos . Ці функції не можна використовувати для *стандартних потоків*.

На початку виконання програми автоматично відкриваються три стандартних потоки: stdin (введення), stdout (виведення), stderr (помилок). Ці ідентифікатори є файловими покажчиками, що зв`язані за умовчанням з вікном консолі. Їм не можна присвоювати значення, не потрібно закривати наприкінці роботи, але можна вказувати у функціях призначених для роботи з файлами. *Наприклад*:

```
fputs("String1\nString2", stdout);
fflush(stdout);
```

Стандартні потоки можна перенаправити, щоб данні зчитувались/записувались з файлу. Для цього існує два способи.

Перенаправлення потоків з командного рядка не вимагає змін у програмі. У командному рядку можна скористатись однією з команд:

test.exe > file.txt – результат виконання програми test.exe записується у файл file.txt. Якщо такого файлу не існує – він створюється, інакше – перезаписується.

test.exe >> file.txt – відбувається дозаписування в кінець файлу file.txt .

test.exe < file.txt – використовується для введення даних з файлу file.txt .

Можна поєднувати: test.exe < file.txt1 > file.txt2

Перенаправлення потоків за допомогою функції freopen() вимагає підключення #include <cstdio> .

Формат:

```
FILE *freopen(const char *Filename, const char *Mode, FILE *fp);
```

Filename, Mode – ідентичні параметрам fopen() . Якщо виникли помилки – повертає NULL.

Приклад:

```
FILE *fp = 0;
fp = freopen("file.txt", "w", stdout);
if (!fp) exit (1);
printf("%s\n", "Записуємо у файл");
```

Перенаправлення стандартних потоків може бути корисним, наприклад, при налагодженні й тестуванні. Однак виконання дискових операцій введення/виведення на перенаправлених потоках є менш ефективним у порівнянні з традиційними операціями з файлами.

Задачі

1. Написати функцію для злиття вмісту двох заданих текстових файлів.
2. Написати функцію для копіювання текстового файлу.
3. Порівняти два текстові файли. Надрукувати перший рядок в якому вони відрізняються.
4. Написати функцію для підрахунку числа рядків у текстовому файлі.

5. Є текстовий файл та рядок S. Записати у інший файл всі його рядки, що містять як фрагмент S.

6. Написати програму для пошуку у текстовому файлі вказаного слова. Слово та ім`я файлу подаються у командному рядку.

7. Для текстового файлу f прибрати всі пропуски, що розташовані в кінці рядків. Результат отримати у файлі g.

8. Написати функцію для дописування до елементів першого файлу елементів другого, зі зберіганням результату в першому файлі. Всі елементи мають фіксовану довжину.

9. Множину цілих представлено файлом, дані якого упорядковано за зростанням. За двома такими файлами створити третій файл, який також є упорядкованим і подає їх

   a. об’єднання;
   b. перетин;
   c. різницю;
   d. симетричну різницю.

10. F файл, що містить цілі числа. Записати в G всі парні, а в H всі непарні числа з збереженням порядку.

11. F файл, що містить цілі числа. Обчислити їх середнє арифметичне значення.

12. Файл містить інформацію про книжки (автор, назва, видавництво, рік, обсяг).

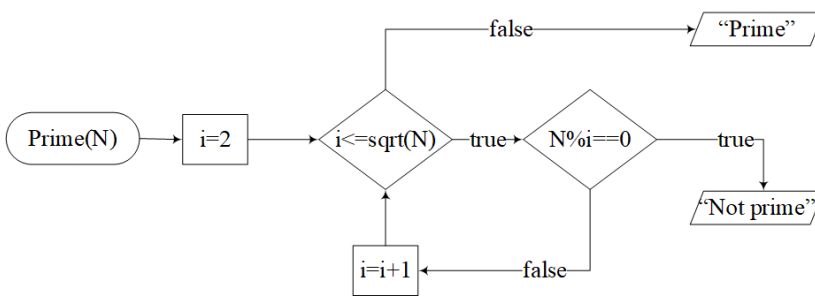
   a. Знайти назви книжок зазначеного автора, виданих з 2000 р.;
   b. Відібрати книжки з програмування;
   c. Підрахувати загальний обсяг по кожному видавництву.

Лабораторна робота №1. Цілочисельна арифметика.

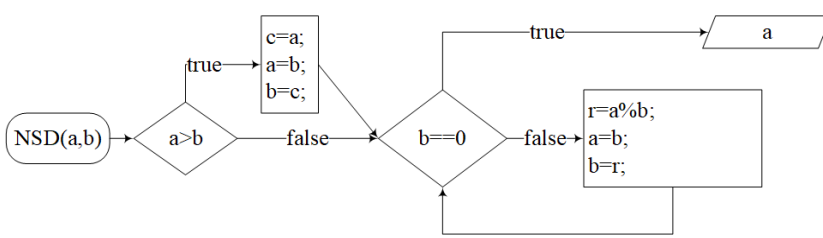
Для ефективного впровадження алгоритмів, пов'язаних з перевіркою властивостей цілих чисел, або зі знаходженням чисел, що задовольняють певній умові, необхідно вміти розбивати задачу на менші підзадачі.

Вибрані алгоритми

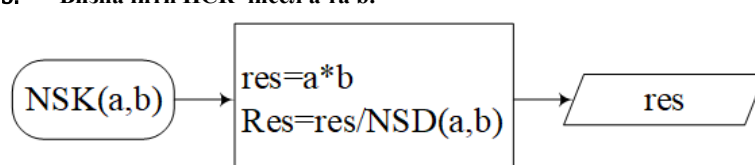
1. **Перевірити, чи є число N простим.**



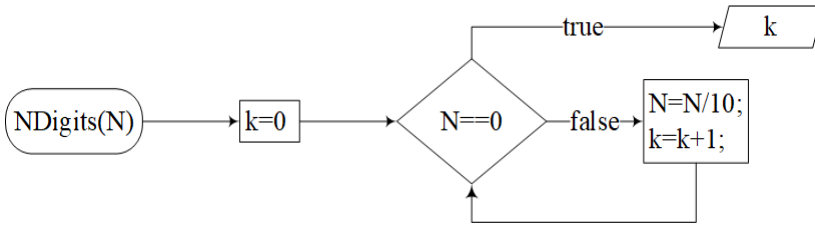
2. **Визначити НСД чисел a та b.**



3. **Визначити НСК чисел a та b.**



4. Визначити кількість цифр числа N.



Варіанти завдань

1) Вводиться N. Знайти, на скільки нулів закінчується $N!$
2) Розкласти на прості множники біноміальний коефіцієнт $C_n^k = \frac{n!}{k!(n-k)!}$.
3) Реалізувати переведення числа з десяткової системи числення у систему з основою m<=16.
4) Реалізувати переведення числа з системи числення з основою m<=16 у десяткову.
5) Знайти всі трійки чисел $x < y < z \leq N$ такі, що $x^2 + y^2 = z^2$.
6) Знайти всі прості числа вигляду $2^n - 1$ (прості числа Мерсена)
7) Число Армстронга – це натуральне число, яке дорівнює сумі своїх цифр у степені, що дорівнює кількості цифр цього числа. Знайти всі числа Армстронга , менші за N.
8) Користувач задає натуральне N. Знайти суму $\frac{1}{2} + \frac{1}{3} + \frac{1}{5} + \cdots + \frac{1}{p_n}$, де p_n – n-те просте число, у вигляді раціонального дробу.
9) Дано два натуральних числа a та b. Вивести на екран період дробу a/b.
10) Прості числа-близнюки це пара простих чисел, різниця між якими становить два. Знайти всі пари простих чисел-близнюків, менших за N.
11) Досконале число це натуральне число, яке дорівнює сумі всіх своїх дільників крім самого числа. (Перші 4 досконалі числа 6, 28, 496, 8128) Перевірити, чи число є досконалим.

12) Число є паліндромом, якщо воно читається однаково і зліва направо і справа наліво. Знайти всі числа – паліндроми, менші за N.

13) 2n-розрядне число називається щасливим, якщо сума перших n цифр дорівнює сумі останніх n цифр цього числа. Знайти всі щасливі числа, менші за N.

14) Розглянемо деяке натуральне число N. Якщо воно парне, то розділимо його на 2, інакше - помножимо на 3 і додамо 1. Будемо повторювати такі дії (кроки), поки не вийде 1. Отримана послідовність називається послідовністю Хейеса, а найбільше з чисел цієї послідовності - її вершиною. Потрібно скласти програму, яка обчислює для заданого N послідовність Хейеса, підраховує число кроків в ній і знаходить її максимальний елемент.

15) ● Дано два натуральних числа a та b. Вивести на екран період дробу a/b.

16) ● Число Кармайкла – це додатне складене число n, що задовольняє умову $b^{n-1} \equiv 1\ (mod\ n)$ для всіх цілих b, взаємно простих з n. (Приклади 561, 1105, 1729, 2465, 2821). Перевірити чи є введене число числом Кармайкла.

17) ● (Критерій Корсельта) Складене число n є числом Кармайкла тоді і тільки тоді, коли n не ділиться на квадрат жодного числа і, крім того, для всіх простих дільників p числа n число $n-1$ ділиться націло на $p-1$. Перевірити чи є введене число числом Кармайкла.

18) ● Вивести на екран число 2^N для заданого N<1000

19) ● Реалізувати «довге» додавання на множині натуральних чисел.

20) ● Реалізувати «довге» віднімання на множині натуральних чисел.

21) ● Реалізувати «довге» множення на множині натуральних чисел.

Лабораторна робота №2. Геометрія на площині

Для ефективного впровадження алгоритмів обчислювальної геометрії на площині доцільно використовувати структури даних. Так, точку з дійсними координатами можна подати структурою

```
struct point
{
float x,
float y,
};
```

Відрізок, у свою чергу, визначається структурою

```
struct interval
{
point startpt, //початок відрізка
point endpt, //кінець відрізка
} ;
```

Багатокутники та ломані лінії можна подати структурою

```
struct polygon
{
unsigned int n //кількість вершин
pt* vertices, //вказівник на самі вершини
};
```

Вибрані алгоритми

1. Визначити точку перетину двох прямих, які задаються двома точками

Нехай перша пряма задається точками (x_1^1, y_1^1) та (x_2^1, y_2^1), а друга – точками (x_1^2, y_1^2) та (x_2^2, y_2^2). Рівняння прямих мають вигляд

$$\frac{x - x_1^i}{x_2^i - x_1^i} = \frac{y - y_1^i}{y_2^i - y_1^i}, i \in \{1,2\}$$

Щоб уникнути можливого ділення на нуль, помножимо рівності на $\left(x_2^i - x_1^i\right) \cdot \left(y_2^i - y_1^i\right)$, одержимо

$$\left(x - x_1^i\right) \cdot \left(y_2^i - y_1^i\right) - \left(y - y_1^i\right) \cdot \left(x_2^i - x_1^i\right), i \in \{1,2\}$$

$$x \cdot \left(y_2^i - y_1^i\right) - y \cdot \left(x_2^i - x_1^i\right) - x_1^i \cdot \left(y_2^i - y_1^i\right) + y_1^i \cdot \left(x_2^i - x_1^i\right) = 0.$$

Введемо нові змінні

$$A_i = y_2^i - y_1^i, \qquad B_i = x_1^i - x_2^i,$$

$$C_i = y_1^i \cdot \left(x_2^i - x_1^i\right) - x_1^i \cdot \left(y_2^i - y_1^i\right) = y_1^i \cdot x_2^i - y_2^i \cdot x_1^i, \qquad i \in \{1,2\}$$

Тоді точка перетину буде розв'язком системи

$$\begin{cases} A_1 x + B_1 y + C_1 = 0 \\ A_2 x + B_2 y + C_2 = 0 \end{cases}$$

Користуючись формулою Крамера, знаходимо розв'язок системи, який і буде шуканою точкою перетину:

$$x = -\frac{\begin{vmatrix} C_1 & B_1 \\ C_2 & B_2 \end{vmatrix}}{\begin{vmatrix} A_1 & B_1 \\ A_2 & B_2 \end{vmatrix}} = -\frac{C_1 B_2 - C_2 B_1}{A_1 B_2 - A_2 B_1},$$

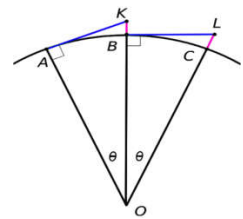
$$y = -\frac{\begin{vmatrix} A_1 & C_1 \\ A_2 & C_2 \end{vmatrix}}{\begin{vmatrix} A_1 & B_1 \\ A_2 & B_2 \end{vmatrix}} = -\frac{A_1 C_2 - A_2 C_1}{A_1 B_2 - A_2 B_1},$$

Якщо знаменник $A_1 B_2 - A_2 B_1$ дорівнює нулю, то прямі паралельні чи співпадають.

2. Намалювати коло, не використовуючи циклічних викликів тригонометричних функцій (Алгоритм Брезенхема)

```
void DrawCircle(float cx, float cy, float r, int
num_segments)
{
float theta = 2*M_PI / float(num_segments); //обчислюємо θ
float tangetial_factor = tanf(theta);  //обчислюємо тангенс θ
  float radial_factor = cosf(theta);
// обчислюємо коефіцієнт поправки
  float x = r;float y = 0; //починаємо з нульового кута
```

```
  glBegin(GL_LINE_LOOP);
  for(int ii = 0; ii < num_segments; ii++) {
    glVertex2f(x , y); //поточна точка на колі A
    float tx = -y; float ty = x;
  //Обчислюємо ортогональний вектор до (x,y)
    //Обчислюємо точку K.
    x += tx * tangetial_factor;
    y += ty * tangetial_factor;
    //Коректуємо та знаходимо точку B
    x *= radial_factor;
    y *= radial_factor;
  }
  glEnd();
}
```



Варіанти завдань

1) Задаються три пари чисел, які задають трикутник на площині. Крім того, задається ще одна пара чисел, яка задає точку на площині. Визначити, чи лежить точка усередині трикутника, чи на одному з його ребер, чи зовні.
2) Задаються три пари чисел, які задають трикутник на площині. Реалізувати поворот та масштабування трикутника відносно його центру мас.
3) Задаються n пар чисел, які задають багатокутник. Написати програму перевірки його опуклості
4) Задаються n пар чисел, які задають багатокутник. Написати програму перевірки наявності перетинів його ребер.
5) Задаються два відрізки. Якщо вони перетинаються, то вивести їх точку перетину
6) Задаються коло та лінія. Якщо вони перетинаються, то вивести точки перетину.
7) Задаються два кола. Вивести координати дотику дотичної лінії до цих кіл.
8) Задаються n пар чисел, які задають багатокутник без самоперетинів. Знайти його площу та периметр. Формула

$$S = \sum_{i=1}^{n} \frac{y_{i+1} + y_i}{2} (x_{i+1} - x_i)$$

9) Задаються два трикутника. Визначити, чи є вони подібними, якщо так то визначити кут повороту, вектор масштабування і вектор зсуву для їх суміщення

10) Дано трикутник. Визначити центр та радіус вписаного та описаного кола.

11) Задаються числа a, b, крок по куту θ та число n. Відобразити перші n точок спіралі $r(\theta) = ae^{bi\theta}, i \in \{0, N\}$ (формула дана в полярних координатах).

12) Задається число a, крок по куту θ та число n. Відобразити перші n точок кривої $r(\theta) = 2a(1 + \cos(i\theta)),\ i \in \{0, N\}$ (формула дана в полярних координатах).

13) ● Дано n точок на площині. Побудувати їх опуклу лінійну оболонку

14) ● Дано опуклий багатокутник. Описати навколо нього коло.

15) ● Не використовуючи тривимірну графіку, змоделювати поворот об'ємного куба в ортогональній системі координат.

16) ● Не використовуючи тривимірну графіку, змоделювати поворот тетраедра в ортогональній системі координат.

17) ● Змоделювати «броунівський» рух кульок. Кульки мають стикатися між собою за законами фізики.

18) ● Задається еліпс. Змоделювати рух матеріальної точки, що рухається всередині еліпса та відбивається від його стінок.

19) ● Задано горизонтальну лінію. По ній котиться коло, всередині якого зафіксована точка. Змоделювати траєкторію цієї точки.

20) ● Задано коло. По ньому котиться інше коло (всередині або зовні). На колі, що рухається, зафіксована точка. Змоделювати траєкторію цієї точки.

21) ● Задано правильний трикутник. По його поверхні котиться коло, на ободі якого зафіксована точка. Змоделювати траєкторію цієї точки.

Лабораторна робота №3. Наближені обчислення

Для наближених математичних обчислень використовують дійсні типи даних, зазвичай double. Але такі числа не можуть точно подати значення – як отримані результати вимірів якихось фізичних значень, так і математичні константи (наприклад, π, e, $\sqrt{2}$). Точність стандартного типу double достатня для подання більшості значень (до 15 цифр подаються точно; в C++ це можна перевірити, подивившись значення std::numeric_limits<double>::digits10). Проте під час обчислень, особливо ітеративних може накопичуватись похибка. Тому треба акуратно будувати алгоритми розрахунків, щоб уникнути втрати точності.

Дійсні числа не варто порівнювати з використанням оператора ==. Часто значення, які алгебраїчно мають бути однаковими, під час наближених обчислень виходять трохи різними. Більш правильним є порівняння з певною заданою точністю. Наприклад:

```
    double one_third = 1.0/3;
    double two_thirds = 1.0 - one_third;
    double one = (1.0 – two_thirds) * 3;

    if (one == 1.0) { std::cout<< “exact equal”<<std::endl;}
    else { std::cout<< “not exact equal”<<std::endl;}

    double max_error = 1e-8;
    if (std::abs(one-1.0)<max_error) { std::cout<< “approx. equal”<<std::endl;}
    else { std::cout<< “not approx. equal”<<std::endl;}
```

Деякі операції знижують точність результату. Наприклад, віднімання або ділення двох близьких (досить великих) чисел. Варто уникати таких операцій, наприклад алгебраїчно перетворюючи вирази. Особливо якщо такі операції повторюються багато разів, наприклад в циклі.

Варіанти завдань

1) Реалізувати чисельне обчислення кореня n-того ступеня. Формула

$$x_{i+1} = \frac{1}{n}\left((n-1)x_i + \frac{A}{x_i^{n-1}}\right)$$

2) Обчислити визначений інтеграл математичного виразу.

3) Методом градієнтного спуску визначити екстремум однозначної функції.. Формула

$$x_{i+1} = x_i - \varepsilon_i \cdot f'(x_i)$$

4) Розв'язати наближено нелінійне рівняння методом дотичних. Формула

$$x_{i+1} = x_i - \frac{f(x_i)}{f'(x_i)}$$

5) Визначити довжину кривої, заданої математичним виразом. Початкова і кінцева точка задаються користувачем. Формула

$$L = \int_a^b \sqrt{f'(t)^2 + 1}$$

6) Знайти площу сектора кривої, заданої у вигляді полярних координат. Початковий і кінцевий кути a, b задаються користувачем. Формула

$$S = \int_a^b [r(\phi)]^2 d\phi$$

7) Знайти чисельний розв'язок системи лінійних рівнянь $Ax = b$ за формулою

$$\begin{pmatrix} x_1 \\ x_2 \\ \dots \\ x_n \end{pmatrix}^{i+1} = \begin{pmatrix} a_{11}+1 & a_{12} & \dots & a_{1n} \\ a_{21} & a_{22}+1 & \dots & a_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ a_{n1} & a_{n2} & \dots & a_{nn}+1 \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \\ \dots \\ x_n \end{pmatrix}^{i} - \begin{pmatrix} b_1 \\ b_2 \\ \dots \\ b_n \end{pmatrix}$$

8) Визначити довжину кривої, заданої у полярних координатах. Початковий і кінцевий кути a, b задаються користувачем. Формула

$$L = \int_a^b \sqrt{r'(t)^2 + r(t)^2}$$

9) Розв'язати наближено нелінійне рівняння методом бісекції. Алгоритм: користувач вводить дві точки, a, b на яких функція приймає різні значення. Обчислюється значення функції в точці $c = (a + b)/2$. Якщо $f(a), f(c)$ різних знаків, то b=c, інакше a=c. Повторюємо, доки модуль значення функції не буде меншим якогось заданого наперед значення.

10) Розв'язати наближено нелінійне рівняння методом хорд. Формула

$$x_{i+1} = x_i - \frac{f(x_i)(x_i - x_{i-1})}{f(x_i) - f(x_{i-1})}$$

11) Дано різницеве рівняння $x_{i+1} = f(i, x_i)$. Знайти суму перших n його елементів

12) Задана функція. Перевірити, чи має вона на відрізку (a, b) обернену, якщо так, то обчислити визначений інтеграл функції $f^{-1}(y)$ на цьому відрізку

13) Знайти наближену суму нескінченного ряду

$$S = \sum_{i=a}^{\infty} f(i)$$

14) (*) Дано функцію $f(x)$. Знайти лінійну функцію $ax + b$, яка найкращим чином апроксимує її. Використати метод найменших квадратів.

15) (*) Дано функцію $f(x)$. Визначити, чи є вона періодичною, якщо так – визначити період.

16) (*) Дано функцію $f(x)$. Розкласти її в частковий ряд Тейлора до n-того члена

$$T(x) = \sum_{k=0}^{n} \frac{f^{(k)}(a)}{k!}(x - a)^k$$

Знайти інтегральну похибку $\int_a^b |f(x) - T(x)|dx$

17) (*) Дано функцію $f(x)$. Розкласти її в частковий ряд Фур'є до n-того члена

$$\Phi(x) = \frac{a_0}{2} + \sum_{k=1}^{n} (a_n \cos(nx) + b_n \sin(nx))$$

Знайти інтегральну похибку $\int_a^b |f(x) - \Phi(x)|dx$

18) (*) Дано функцію $f(x, y)$. Вивести на екран функцію $g(x) = \int_a^b f(x, y)\, dy$

19) (*) Дано функцію $f(x)$, що всюди на [a,b] має похідну. Визначити всі нулі функції на відрізку [a,b]

20) (*) Дано пари $(x_i, y_i), i \in \{1..N\}$. Інтерполювати ці точки поліномом, тобто знайти поліном (N-1)-ого ступеня, що проходить через всі ці точки.

Лабораторна робота №4. Обчислення математичних виразів та обернена польська нотація

Обернена польська нотація – вид запису алгебраїчних та логічних виразів, при якому символ операції ставиться після операндів.

Вираз	Традиційна (інфіксна) нотація	Зворотна польська (постфіксна) нотація
a + b × c	a + b*c	a b c * +
(a + b) × (z + x)	(a + b)*(z + x)	a b + z x + *
(a + t) × (b × (a + c))$^{(c + d)}$	(a + t)*(b * (a + c))^(c + d)	a t + b a c + * c d + ^ *

Таблиця 4.1. Приклади виразів у традиційній та оберненій польській нотації

Алгоритм для перетворення звичайного запису в бездужковий

Необхідні типи даних: рядок або масив чисел, функцій та операцій, що визначає вхідний вираз, який треба перетворити; вихідний масив, де в кінці роботи алгоритму зберігатиметься вхідний вираз у оберненій польській нотації; допоміжний стек.

Стек – різновид лінійного списку, структура даних, яка працює за принципом «останнім прийшов — першим пішов» (LIFO, англ. last in, first out). Всі операції (наприклад, видалення елементу) в стеку можна проводити тільки з одним елементом, який знаходиться на верхівці стека та був введений в стек останнім.

Зчитуємо символи поки не закінчаться і для кожного робимо наступне:

- Якщо A_i є числом то записуємо його у вихідний масив;
- Якщо A_i є функцією то завантажуємо його в стек;
- Якщо $A_i = '('$, то поміщаємо його в стек;
- Якщо $A_i = ')'$, то до тих пір, поки верхнім елементом стека не стане відкриваюча дужка, виштовхуємо елементи з стека у вихідний масив. При цьому відкриваюча дужка видаляється з стека, але у вихідний масив не додається. Якщо після цього кроку на вершині стека виявляється символ функції, виштовхуємо його у

вихідний масив. Якщо стек закінчився раніше, ніж ми зустріли відкриваючу дужку, це означає, що у виразі або невірно розставлені оператори, або не узгодженні дужки.

Якщо символ A_i є **асоціативним** або **ліво-асоціативним** оператором то до тих пір, поки пріоритет оператора менше або дорівнює пріоритету оператора, що знаходиться на вершині стека, виштовхуємо верхні елементи стека у вихідний масив. (В С++ всі оператори, крім присвоювання, ліво-асоціативні);

- Якщо символ A_i є **право-асоціативним** оператором, то до тих пір, поки пріоритет оператора менше пріоритету оператора, що знаходиться на вершині стека, виштовхуємо верхні елементи стека у вихідний масив;
- Коли вхідний рядок закінчився, виштовхуємо всі символи зі стеку у вихідний масив. У стеці повинні залишитися тільки символи операторів; якщо це не так, значить у виразі не узгоджені дужки.

Наведемо приклад перетворення виразу до оберненої польської нотації. Нехай маємо рядок «3+4*2/(1-5)^2».

Поточний символ	**Стек**	**Вихід**	**Коментар**
3		3	Додаємо «3» до виходу
+	+	3	Вставляємо «+» в стек
4	+	3 4	Додаємо «4» до виходу
*	+ *	3 4	Вставляємо «*» в стек
2	+ *	3 4 2	Додаємо «2» до виходу
/	+ /	3 4 2 *	Видаляємо «*» зі стеку і додаємо до виходу, вставляємо «/» в стек.
(	+ / (	3 4 2 *	Вставляємо «(» в стек
1	+ / (	3 4 2 * 1	Додаємо «1» до виходу
-	+ / (-	3 4 2 * 1	Вставляємо «-» в стек
5	+ / (-	3 4 2 * 1 5	Додаємо «5» до виходу
)	+ /	3 4 2 * 1 5 -	Видаляємо «-» зі стеку і додаємо до виходу, видаляємо «(» зі стеку
^	+ / ^	3 4 2 * 1 5 -	Вставляємо «^» в стек
2	+ / ^	3 4 2 * 1 5 - 2	Додаємо «2» до виходу
		3 4 2 * 1 5 – 2 ^ / +	Виштовхуємо усі елементи зі стеку і додаємо до виходу

Таблиця 4.2. Приклад перетворення виразу до оберненої польської нотації

Алгоритм для обчислення значення виразу у оберненій польській нотації

Для всіх символів робимо наступне:

- Якщо A_i число, то покласти його у стек;
- Якщо A_i є бінарним оператором, то:
  - Послідовно витягуємо із стеку два числа x_1 і x_2;
  - Виконуємо дію $x_2 A_i x_1$ і результат вкладаємо в стек;
- Якщо A_i є унарним оператором або функцією то:
  - Витягуємо із стеку одне число;
  - Визначаємо значення функції із відповідним аргументом та поміщаємо результат у стек;
- В кінці роботи в стеку знаходитиметься результат виразу.

Приклад 4.1

Наведемо приклад обчислення виразу, записаного у оберненій польській нотації. Нехай маємо вираз: 12 2 3 4 * 10 5 / + * +

Тоді порядок дій над ним буде наступний:

Крок	Елемент	Стек
1	12	12
2	2	12 2
3	3	12 2 3
4	4	12 2 3 4
5	*	12 2 **12**
6	10	12 2 12 10
7	5	12 2 12 10 5
8	/	12 2 12 **2**
9	+	12 2 **14**
10	*	12 **28**
11	+	**40**

Таблиця 4.3. Приклад обчислення виразу у оберненій польській нотації

Варіанти завдань

Необхідно перевірити правильність введеного виразу та обчислити його значення. У виразі можуть бути дужки.

№	Операнди	Бінарні оператори	Унарні оператори та функції
1	$\mathbb{N} \cup \{0\}$	+,-,*,/,^	+,-
2	$\mathbb{R}^+$	*,/,%,^	+,-
3	$\mathbb{N} \cup \{0\}$ та змінні	+,-,*	+,-,правий ++, правий --
4	$\mathbb{N} \cup \{0\}$	+,-,*,%	+,-,sin,cos,
5	$\mathbb{N} \cup \{0\}$ та змінні	+,-,/	+,-,exp
6	$\mathbb{R}^+$ та змінні	+,-	+,-,ln

- *Змінна – послідовність, що складається з латинських літер та цифр і починається з латинської літери. Якщо у виразі є змінні, то інтерпретатор повинен запросити у користувача їх значення.*

Лабораторна робота №5. Рекурсія.

Рекурсія – спосіб виклику, за яким підпрограма (процедура, функція) викликає сама себе, або напряму (пряма рекурсія), або через виклики інших функцій (непряма рекурсія).

Рекурсивні алгоритми:

- Метод «розділяй та володарюй» (divide and conquer) – коли задачу розділили на підзадачі, але меншого розміру;
- рекурсивні означення;
- рекурсивні структури даних.

Важливі поняття: глибини рекурсії, загальної кількості викликів.

Глибина рекурсії, на якій перебуває виклик підпрограми – кількість рекурсивних викликів, розпочатих та не незавершених на момент початку цього виклику

Глибина рекурсії, породжена викликом – максимальна кількість рекурсивних викликів, які розпочато й не завершено після початку цього виклику.

Загальна кількість рекурсивних викликів, породжених викликом рекурсивної функції - кількість викликів, які виконані між початком та завершенням.

Необхідно уникати ситуацій, коли рекурсія призводить до експоненційного зростання кількості викликів.

Іноді цього можна уникнути, запам'ятовуючи проміжні результати

Вибрані алгоритми

1. Написати рекурсивну програму обчислення факторіалу числа N

Визначимо функцію $int\ Fact(int\ i)$, яка:

- якщо $i > 1$, повертає значення $i \cdot Fact(i-1)$,
- інакше повертає 1.

Робимо початковий виклик функції $Fact$ із функції main: $Fact(N)$. Глибина рекурсії у цьому випадку дорівнює *N*.

2. Вивести на екран фрактальне дерево

Визначимо функцію $void\ F(float\ x, float\ y, float\ l, float\ \alpha, int\ d)$, яка:

- малює відрізок з точки (x, y) довжини l під кутом α,
- обчислює кінець відрізку (x^*, y^*),
- якщо $d > 0$, то викликає себе двічі:
  $F(x^*, y^*, l \cdot K, \alpha + \alpha_0, d-1)$ та $F(x^*, y^*, l \cdot K, \alpha - \alpha_0, d-1)$.
  (Параметри $\alpha_0 > 0$ та $K < 1$ – можна вибрати константами)

Робимо початковий виклик функції F із функції main: $F(0,0,1,\pi/2,N)$. Отже, дерево буде «рости» з точки (0,0) вверх. Параметр N визначає глибину рекурсії.

3. Написати рекурсивну програму генерації всіх перестановок чисел 1,2,…N.

Функція $void\ Gen(int\ pos, int*\ arr, int\ n)$:

- Якщо $pos < n-1$, то
  - у циклі, пробігаючи всі значення i від pos до $n-1$,
  - міняє місцями елементи масиву $arr[i]$ та $arr[pos]$,
  - викликає себе з параметрами $Gen(pos+1, int*\ arr, int\ n)$.
  - знову міняє місцями елементи масиву $arr[i]$ та $arr[pos]$,
- Кінець циклу.
- Якщо $pos = n-1$, то виводить на екран масив arr.

Робимо початковий виклик функції Gen із функції main: $Gen(0,\{1,2,\dots,N\},N)$.

```
void Gen(int k,int *a,int n)
{
if (k<n-1)
for (int i=k;i<n;i++)
    {
    swap<int>(a[i],a[k]);
    Gen(k+1,a,n);
    swap<int>(a[i],a[k]);
    }
else
    {
    for (int i=0;i<n;i++)
            cout<<a[i]<<" ";
    cout<<endl;
    }
```

```
}
```

4. Написати рекурсивну функцію генерації всіх K-елементних комбінацій з множини $\{1, 2, 3 \dots N\}$.

Функція $void\ in(int\ n, int\ ost)$ має параметр $int\ n$ – поточне число до можливого включення у комбінацію, $int\ ost$ – кількість елементів, що необхідно включити у комбінацію. Якщо $ost > 0\ \&\&\ n <= N$, то функція рекурсивно викликає себе двічі з параметрами $(n + 1, ost - 1)$ та $(n + 1, ost)$. Крім того, перед першим викликом заповнюється відповідна позиція масиву $comb\_set$, у якому зберігається поточна комбінація. Якщо $ost == 0$, то повністю заповнений масив $comb\_set$ виводиться на екран.

```
int N,K; int* comb_set;
void in(int n, int ost) {
if (ost > 0 && n <= N)
        {
        comb_set[K-ost] = n;
        in(n + 1, ost - 1);
        in(n + 1, ost);
        }
    if (ost == 0){
         for (int i = 0; i < K; i++)
         cout << comb_set[i] << ' ';
         cout << "\n";
}}
```

Варіанти завдань

1) Написати рекурсивну програму обчислення n-того члену послідовності Фібоначчі.
2) Ланцюговий дріб – це математичний вираз вигляду

$$a_0 + \cfrac{b_0}{a_1 + \cfrac{b_1}{a_2 + \cfrac{b_2}{a_3 + \cdots}}}$$

3) Послідовності $a = \{a_0, a_1, a_2, a_3, \dots, a_{n-1}\}$ та $b = \{b_0, b_1, b_2, b_3, \dots, b_{n-1}\}$ мають довжину n та задаються користувачем. Обчислити значення ланцюгового дробу.
4) Написати рекурсивну програму обчислення біноміального коефіцієнту C_n^k використовуючи правило додавання $C_n^k = C_{n-1}^k + C_{n-1}^{k-1}$ та початкові умови $C_n^0 = C_n^n = 1$ для всіх *n*.
5) Реалізувати швидке сортування масиву (Метод Хоара).
6) Написати рекурсивну програму генерації всіх перестановок чисел 1,2,…N.
7) Написати програму перевірки, чи задовольняє рядок масці. Символ «?» в масці означає довільний символ, а «*» означає послідовність символів довільної (можливо, нульової) довжини. Приклад: рядок «12345» задовольняє масці «?2*5*».
8) Написати рекурсивну програму заливки області кольором *A*. Границя області – клітинки з кольором, що відрізняються від кольору клітинок області.
9) Написати рекурсивну програму виводу на екран фрактального дерева.
10) Написати рекурсивну програму виводу на екран кривої Коха.
11) Написати рекурсивну функцію розбиття множини {1,2,3…N} на не пронумеровані підмножини.
12) Написати рекурсивну програму, що розкриває дужки у виразі

$$\prod_{i=0}^{n-1}(a_i x + b_i).$$

13) та виводить на екран коефіцієнти одержаного многочлена. Масиви a, b задаються користувачем.
14) Написати рекурсивну програму, що розкриває дужки у виразі $(ax^2 + bx + c)^n$ та виводить на екран коефіцієнти одержаного многочлена. Числа a, b, c, n задаються користувачем
15) Написати рекурсивну програму, що знаходить кількість шляхів коня на шаховому полі nxn з клітинки (1,1) у клітинку (n, n), причому кінь не може повторювати клітинки у маршруті.

16) Словом Діка називається довільне слово в алфавіті {0, 1}, що задовольняє такі вимоги: 1) воно містить однакову кількість символів 0 та 1; 2) довільний його префікс(тобто початок) містить символів 1 не більше ніж символів 0. У текстовому файлі зберігається деяка кількість рядків, які складаються з цифр “0” та “1”. Написати рекурсивну програму, що знаходить всі слова Діка довжини 2n.

17) Дано набір з n цілих чисел, де n-парне. Необхідно розділити множину на дві підмножини розміру n/2 таким чином, щоб абсолютне значення різниці сум двох підмножин було мінімально можливим.

Лабораторна робота №6. Динамічне програмування та пошук з поверненням.

Динамічне програмування – метод вирішення задачі шляхом її розбиття на кілька однакових підзадач, рекурентно пов'язаних між собою. Найпростішим прикладом будуть числа Фібоначчі – щоб обчислити деяке число в цій послідовності, нам потрібно спершу обчислити третє число, склавши перші два, потім четверте таким же чином на основі другого і третього, і так далі.

Рішення завдання динамічним програмуванням повинно містити наступне:

- Залежність елементів динаміки один від одного. Така залежність може бути прямо дана в умови (так часто буває, якщо це завдання на числові послідовності). В іншому випадку ви можете спробувати дізнатися якийсь відомий числовий ряд (на кшталт тих же чисел Фібоначчі), обчисливши перші кілька значень вручну. Якщо вам зовсім не пощастило – доведеться думати
- Значення початкових станів. В результаті довгого розбиття на підзадачі вам необхідно звести функцію або до уже відомих значеннях (як у випадку з Фібоначчі – заздалегідь визначені перші два члена), або до задачі, розв'язуваної елементарно.

Приклад задачі динамічного програмування: обчислення відстані Левенштейна між двома рядками. Відстань Левенштейна це міра відмінності двох послідовностей символів (рядків). Обчислюється як мінімальна кількість операцій вставки, видалення і заміни, необхідних для перетворення одної послідовності в іншу. Для розрахунку відстані Левенштейна найчастіше застосовують простий алгоритм, в якому використовується матриця розміром $(n+1)\times(m+1)$, де n і m - довжини порівнюваних рядків v та w. Вартість операцій вилучення, заміни та вставки вважається однаковою. Для конструювання матриці для рядків використовують таке рекурсивне рівняння з початковими умовами $D_{i0} = i, D_{0j} = j$:

$$D_{ij} = \begin{cases} D_{i-1,j-1} + (v_i = w_j) \\ D_{i-1,j} + 1 \\ D_{i,j-1} + 1 \end{cases}$$

```
    int levenshtein_distance(const string & src, const string
& dst) {
      int m = src.size();
      int n = dst.size();
      if (m == 0) return n;
      if (n == 0) return m;
      std::vector< std::vector<int>> matrix(m + 1);
      for (int i = 0; i <= m; ++i) {
        matrix[i].resize(n + 1);
        matrix[i][0] = i;
      }
      for (typename T::size_type i = 0; i <= n; ++i) {
        matrix[0][i] = i;
      }
      int above_cell, left_cell, diagonal_cell, cost;
      for (int i = 1; i <= m; ++i) {
        for(int j = 1; j <= n; ++j) {
          cost = src[i - 1] == dst[j - 1] ? 0 : 1;
          above_cell = matrix[i - 1][j];
          left_cell = matrix[i][j - 1];
          diagonal_cell = matrix[i - 1][j - 1];
          matrix[i][j]  =  std::min(std::min(above_cell  +  1,
left_cell + 1), diagonal_cell + cost);
        }
      }
      return matrix[m][n];
    }
```

Пошук з поверненням – загальний алгоритм для знаходження всіх (або деяких) розв'язків деякої обчислювальної задачі, який поступово будує кандидатів

на розв'язок, і відкидає кожного неповного кандидата («вертається») як тільки визначає, що кандидат не може бути доповненим до вірного розв'язку.

Класичний приклад використання пошуку з вертанням — це задача про вісім ферзів, в який потрібно знайти розташування восьми ферзів на стандартній шахівниці таким чином, щоб жоден ферзь не атакував іншого. В звичайному підході пошуку з повереннням, неповні кандидати це розташування k ферзів в перших k рядах дошки, всі в різних рядах і стовпцях. Будь-який неповний розв'язок, що містить два ферзі, які атакують один одного має бути відкинутий, бо він не може бути доповненим до повного правильного розв'язку.

Приклад задачі на пошук з поверненням: Згенерувати рядок довжини N, що складається з цифр {1,2,3} такий, що він не містить однакових підрядків, що йдуть підряд. Приклад: «12131231»

```
    bool prov(int pos,char *arr) //функція, що перевіряє чи
поточний символ не порушує умову
     {
       for (int i=1;i<=1+pos/2;i++)
       {
        bool equ=true;
        for (int j=pos;j>pos-i;j--)
        if (arr[j]!=arr[j-i]) equ=false;
        if (equ==true) return false;
       }
     return true;
     }
     char* get123seq(int len) //основна функція
     {
        char *res=new char[len];
        for (int i=0;i<len;i++) res[i]=0;
        int pos=0;
        while (pos<len)
        {
            do //послідовно перебираєм поточний символ поки
умова на рядок не виконається
            res[pos]++;
```

```
            while ((prov(pos,res)==false)&&res[pos]<=3);
            if (res[pos]==4) {res[pos]=0;pos--;} else pos++;
//якщо ми не змогли знайти символ, відкочуємся назад і міняємо
попередній символ.
        }
        return res;
    }
```

Варіанти завдань

Завдання на динамічне програмування

1) Знайти найдовшу спільну підпослідовність у двох масивах
2) Знайти найдовшу неспадаючу послідовність у масиві
3) Знайти найдовшу підпослідовність у масиві, що є паліндромом
4) Знайти максимальну за сумою підпослідовність у масиві.
5) Знайти найдовший спільний підрядок у двох рядках.
6) Перевірити, чи задовольняє рядок масці. Маска – це рядок, у якому крім звичайних символів є спеціальні – «?» – один довільний символ і «*» – будь-яка кількість довільних символів.
7) Знайти найдовшу пилоподібну (у який кожен елемент або менше своїх сусідів, або більше за них) підпослідовність у масиві.
8) Дається словник допустимих слів та рядок, що складається зі слів у словнику, але без пробілів. Необхідно розставити пробіли.

Завдання на пошук з поверненням

1) Розставити на шаховій дошці 8 ферзів так, щоб вони не били одна одну.
2) Кінь розміщується на першій клітинці порожньої шахової дошки і, рухаючись за правилами шахів, повинен один раз відвідати кожну клітинку. Вивести на екран маршрут коня.
3) У множині елементів, що задається масивом, знайти підмножину, сума елементів якої дорівнює K.

4) Квадратна матриця складається з нулів та одиниць. Знайти найдовший шлях у матриці між двома заданими точками, що складається тільки з одиниць. Доступні кроки вправо, вліво, вверх, вниз.
5) Дано послідовність додатних чисел. Розставити між ними знаки «+» та «-» так, щоб у результаті ми одержали 0. Якщо це не можливо, вивести відповідне повідомлення.
6) Згенерувати допустиме поле Судоку, що має один розв'язок https://www.101computing.net/sudoku-generator-algorithm/
7) Вивести на екран всі перестановки чисел від 1 до N, при яких $\forall i\ a[i] \neq i$
8) Розв'язати гру Судоку за допомогою пошуку з поверненням

Інші завдання

1) Дано рядок. Дописати до нього мінімальну кількість символів зліва або справа так, щоб він став паліндромом.
2) Реалізувати роботу нормального алгоритму Маркова.
3) Реалізувати перетворення рядка, що складається з {0,1} у код Грея і навпаки.
4) Дано масив натуральних чисел. Вивести на екран без повторень всі трійки чисел з масиву, які є сторонами гострокутного трикутника
5) Вивести значення елементів масиву по спіралі (скатертина Улама)
6) Дано два масива. Чи існує перестановка, що одночасно перетворює ці масиви на монотонно неспадаючі? Якщо існує, то вивести її на екран.

Лабораторна робота №7. Робота з файлами.

Файли класифікують за типом компонентів і за методом доступу до них. За типом компонентів розрізняють текстові та бінарні (двійкові) файли, а за методом доступу – файли послідовного і прямого доступу. Текстові файли призначені для збереження текстів (наприклад, текстів програм), а бінарні файли використовуються для збереження даних різних типів.

Файл текстовий є сукупністю символьних рядків змінної довжини. Кожен рядок завершується маркером кінця рядка – спеціальною парою керуючих символів: #13 (повернення каретки) та #10 (переведення рядка). Наприкінці файла записується маркер кінця файла – керуючий символ #26.

Файл бінарний – це лінійна послідовність байтів, що відповідає внутрішньому поданню даних без поділу на рядки.

Приклад 1. Вивести на екран вміст текстового файлу та його розмір

```
void openFile(char out[],char path[])
{
  char str[STRING_BUFFER];
  FILE *file;
  file=fopen(path,"r");
  while((fgets(str,STRING_BUFFER,file))!=NULL)
    {
      strcat(out,str);
    }
  fclose (file);
printf("%s\nSize=%i.\n\n",out,strlen(out));
}
```

Приклад 2. Дано бінарний файл, що містить дійсні числа. Поміняти в ньому місцями максимальний і мінімальний елементи.

```
void swap(char* fname)
  {
     FILE *f;
     int i=0, pmin=0, pmax=0;
```

```
        double max,min,v;
        f = fopen(fname,"r+b");
        if(f==NULL) return;
        fread(&v,sizeof(double),1,f);
        max = v;
        min = v;
        i=1;
        while(!feof(f))
          {
            fread(&v,sizeof(double),1,f);
            if(max<v) pmax = i;
            if(min>v) pmin = i;
            i++;
          }
        fseek(f,pmax*sizeof(double),0);
        fwrite(&min,sizeof(double),1,f);
        fseek(f,pmin*sizeof(double),0);
        fwrite(&max,sizeof(double),1,f);
        fclose(f);
    }
```

Варіанти завдань

1) Вивести текст на екран, прибравши зайві пробіли та вирівнявши текст по правому краю.
2) Вивести текст на екран, прибравши зайві пробіли та вирівнявши текст по лівому краю. Абзаци мають виділятися абзацним відступом.
3) Вивести текст на екран, вирівнявши його по ширині (шляхом дописування між словами пробілів)
4) В тексті зустрічаються числа від 0 до 999, але записані англійськими словами. Замінити їх на числа. Наприклад «… one hundred seventeen …» замінити на …117… Записати результуючий текст у інший файл.
5) Вилучити з тексту всі слова, які містять у собі паліндроми довжиною не менші за n. Записати результуючий текст у інший файл.

6) Вилучити з тексту всі слова, які задовольняють масці. Записати результуючий текст у інший файл.
7) Знайти у тексті всі пари дзеркальних слів (Наприклад, “abcde”, “edcba”) та вивести їх на екран.
8) У текстовому файлі зберігається послідовність довгих чисел, розділена знаками + та -. Вивести на екран результат виразу.
9) У бінарному файлі зберігається велика послідовність чисел. Вивести на екран середнє арифметичне всіх додатних елементів.
10) У бінарному файлі зберігається велика послідовність чисел. Вивести на екран середнє арифметичне добутків сусідніх чисел
11) У бінарному файлі зберігається n матриць розміром mxm. Вивести на екран їх добуток.
12) Записати всі слова, які є числами, з текстового у бінарний файл.
13) Вилучити з текстового файлу всі слова, яких немає у «словнику». Результат записати у інший файл.
14) У текстовому файлі зберігається деяка кількість рядків. Кожен рядок вхідного файлу перевірити на відповідність відкриваючих та закриваючих круглих дужок. Приклад: рядок “((abc))(d)()ef” є коректним, рядок “))a((“ є некоректним.
15) Словом Діка називається довільне слово в алфавіті {0, 1}, що задовольняє такі вимоги: 1) воно містить однакову кількість символів 0 та 1; 2) довільний його префікс(тобто початок) містить символів 1 не більше ніж символів 0. У текстовому файлі зберігається деяка кількість рядків, які складаються з цифр “0” та “1”. Кожен рядок файлу перевірити чи є він словом Діка.

- *Текст задається у текстовому файлі, кирилиця не допускається.*

Додаток А. Як читати блок-схеми?

Найменування	Позначення	Функція
Термінатор		Елемент відображає початок і кінець програми або функції. Всередині фігури записується назва функції та аргументи, що в неї передаються.
Процес		Виконання однієї або кількох операцій, обробка даних будь-якого виду. Всередині фігури записують безпосередньо самі операції.
Перемикання		Елемент перемикального типу з одним входом і двома або більше альтернативними виходами, з яких тільки один може бути вибраний після обчислення умов, визначених всередині цього елементу.
Зумовлений процес		Символ відображає виконання процесу, що складається з однієї або кількох операцій, що визначені в іншому місці програми (у підпрограмі, модулі). Всередині символу записується назва процесу і передані в нього дані.
Дані		Символ відображає обробку (введення) або відображення результатів обробки (виведення) даних.
Межа циклу		Символ складається з двох частин - відповідно, початок і кінець циклу - операції, що виконуються всередині циклу, розміщуються між ними. Умови циклу і збільшення записуються всередині символу початку або кінця циклу - в залежності від типу організації циклу. Часто для зображення на блок-схемі циклу замість цього символу використовують символ перемикання, вказуючи в ньому умову, а одну з ліній виходу замикають вище в блок-схемі (перед операціями циклу).
З'єднувач		Символ відображає вихід в частину схеми і вхід з іншої частини цієї схеми. Використовується для обриву лінії та продовження її в іншому місці (приклад: поділ блок-схеми, що не поміщається на листі). Відповідні сполучні символи повинні мати одне (при тому унікальне) позначення.
Коментар		Використовується для відображення більш детальної інформації про кроки процесу або групи процесів. Опис розташовується з боку квадратної дужки і охоплюється нею по всій висоті. Пунктирна лінія йде до описуваного елементу, або групи елементів (при цьому група виділяється замкнутою пунктирною лінією).

Додаток Б. Підключення зовнішніх бібліотек.

Заголовний файл – тип текстового файлу в деяких мовах програмування, зокрема C та C++, в якому містяться декларації макросів, змінних та прототипів функцій. Заголовні файли додаються у вихідний файл у тому місці, де розташована деяка директива (#include <file.h> для C++). У мовах програмування C та C++, заголовні файли - основний спосіб підключити до програми типи даних, структури, прототипи функцій, типи, і макроси, використовувані в іншому модулі. Має за замовчуванням розширення .h; іноді для заголовків файлів мови C++ використовують розширення .hpp.

Статичні бібліотеки – це об'єктні файли, що підключаються програмістом до своєї програми на етапі компіляції (у Microsoft Windows такі файли мають розширення .lib, у UNIX-подібних ОС - зазвичай .a). В результаті програма включає в себе всі необхідні функції, що робить її автономною, але збільшує розмір.

Динамічні бібліотеки – це об'єктні файли, які завантажується в ОС за запитом працюючої програми в ході її виконання, тобто динамічно. Один і той же набір функцій (підпрограм) може бути використаний відразу в декількох працюючих програмах, через що вони мають ще одну назву – бібліотеки загального користування (Shared Library). Динамічні бібліотеки зберігаються зазвичай у визначеному місці і мають стандартне розширення. Наприклад, в Microsoft Windows файли бібліотек загального користування мають розширення .dll; в UNIX - подібних ОС – зазвичай .so; в Mac OS - .dylib.

Місце розташування заголовних файлів та бібліотек за замовчуванням

Компілятор	Заголовні файли	Статичні бібліотеки	Динамічні бібліотеки
Microsoft Visual C++	…\VC\include	…\VC\lib	…\VC\bin
MinGW	…\MinGW\include	…\MinGW\lib	…\MinGW\bin

- Динамічні бібліотеки можуть розташовуватись, крім того, у теках C:\Windows\system32 або C:\Windows\SysWow64 залежно від розрядності компілятора.
- Місце розташування MS Visual C++ за замовчуванням *C:\Program Files \Microsoft Visual Studio **.*\VC або C:\Program Files (x86)\Microsoft Visual Studio **.*\VC*
- При встановленні MinGW разом з Code::Blocks місце розташування компілятора за замовчуванням *C:\Program Files\CodeBlocks\MinGW* або *C:\Program Files (x86)\CodeBlocks\MinGW*

Приклад: встановлення freeglut.

- Скачуємо Prepackaged Releases із сайту http://freeglut.sourceforge.net/ у залежності від компілятора (MSVC Package або MinGW Package)
- Переносимо заготовочні файли та бібліотеки у відповідні теки компілятора.
- Підключаємо бібліотеки у властивостях проекту
- MS Visual C++ 2012: Menu – Project – Properties – Configuration Properties – Linker – Input – Additional Dependencies. Потрібно додати бібліотеку freeglut.lib
- Code::Blocks: Menu – Project – Build Options – Linker Settings – Add. Потрібно додати бібліотеки freeglut, opengl32 та glu32.

Додаток В. Пріоритет операцій.

1) **(Найвищий пріоритет) Виклик функції і доступ до елемента масиву.**
2) **Операції з одним операндом:**
   - логічне заперечення (!) ;
   - побітовое заперечення (~) ;
   - інкремент (++);
   - декремент (--) ;
   - унарний мінус (-).
3) **Побітові операції:**
   - побітова операція «і» (&) ;
   - побітова операція «або» (|) ;
   - побітова операція виключаюче «або» (^) ;
   - побітовий зсув вправо (>>) ;
   - побітовий зсув вліво (<<) .
4) **Ряд арифметичних операцій:**
   - множення (*) ;
   - ділення (/) ;
   - отримання залишку від ділення (%) .
5) **Решта арифметичних операцій:**
   - додавання (+);
   - віднімання (-).
6) **Операції відношення:**
   - більше (>) ;
   - більше або дорівнює (> =) ;
   - менше (<) ;
   - менше або дорівнює (< =) ;
   - дорівнює (==) ;
   - не дорівнює (! =).
7) **Логічна операція «і» (&&) .**
8) **Логічна операція «або» (| |) .**
9) **Операція присвоювання та об'єднані арифметичні та побітові операції з операцією присвоювання (=, +=, -=, *=, \=, %=, &=, |=, ^=, >>=, <<=)**

Рекомендовані джерела

1. Прата С. Язык программирования С++. Лекции и упражнения. -М.: «И.Д. Вильямс», 2007.
2. Павловская Т.А. С/С++. Программирование на языке высокого уровня. - СПб.: Питер, 2003.
3. Ковалюк Т.В. Основи програмування. – К.: Видавнича група BHV, 2005.
4. Шилдт Г. С++. Базовый курс. 3-е изд. – М.: «И.Д. Вильямс», 2010.
5. Клюшин Д.А. Полный курс С++. Профессиональная работа. – М.: «И.Д. Вильямс», 2005.
6. Страуструп Б. Язык программирования С++. Специальное издание. – СПб.: Невск. Диалект, 2006.
7. Ставровський А.Б., Карнаух Т.О. Програмування. Перші кроки. –К.: «Діалектика», 2005.
8. Вирт Н. Алгоритмы и структуры данных – 2-е изд., испр. — СПб.: Невский Диалект, 2001.
9. Окулов С.М. Программирование в алгоритмах. - М.: Бином. Лаборатория знаний, 2004.
10. Зубенко В.В., Омельчук Л.Л. Програмування: навчальний посібник. – К.: ВПЦ: «Київський університет», 2011.
11. Эккель Б. Философия С++. Введение в стандартный С++. 2-е изд. СПб.: Питер, 2004.
12. Липпман С., Лажойе Ж. Язык программирования С++. Вводный курс. – СПб.: Невск. Диалект, 2001.
13. Керниган Б., Пайк Р. Практика программирования. – СПб.: Невск. Диалект, 2001.
14. Вступ до програмування мовою С++. Організація даних / Т. О. Карнаух, Ю. В. Коваль,М. В. Потієнко, А. Б. Ставровський. . К.: ВПЦ "Київський університет", 2015.

15. Вступ до програмування мовою С++. Структури даних / Р. А. Веклич, Т. О. Карнаух,А. Б. Ставровський. . К.: ВПЦ "Київський університет", 2018.
16. Майерс С. Эффективное использование С++. –М.: ДМК, 2000.
17. Майерс С. Эффективный и современный С++. – М.: «И.Д. Вильямс», 2016.
18. International Standard ISO/IEC 14882:2017(E) . Programming Language C++ : [Електронний ресурс]. Режим доступу: https://isocpp.org/std/the-standard.
19. Куликов С. Тестирование программного обеспечения. Базовый курс. ©EPAM Systems. Версия книги 2.1.3 от 21.08.2019. http://svyatoslav.biz/software_testing_book/.
20. Керниган Б., Ричи Д. Язык программирования Си. – СПб.: Невск. Диалект, 2001.
21. Коплиен Дж. Программирование на С++. – СбП.: Питер, 2004.
22. Харбисон С.П., Стил Г.Л. Язык программирования С. – М.: БИНОМ, 2004.
23. Сэджвик Р. Фундаментальные алгоритмы на С++. М.: ООО «ДиаСофтЮП», 2002.
24. Шилдт Г. Полный справочник по С. – М.: «И.Д. Вильямс», 2006.
25. Проценко В.С., Чаленко П.Й., Ставровський А.Б. Техніка програмування мовою Сі. - К.: Либідь, 1993.
26. Дж. Макконнел Основы современных алгоритмов. - М.: Техносфера, 2004.
27. Себеста Р. Основные концепции языков программирования. – М.: «И.Д. Вильямс»,, 2001.
28. Уэзерелл Ч. Этюды для программистов. - М.: Мир, 1982.
29. Липский В. Комбинаторика для программистов. - М.: Мир, 1988.
30. Порублев И.Н., Ставровский А.Б. Алгоритмы и программы. Решение олимпиадных задач.- - М.:ООО «И.Д.Вильямс», 2007.
31. Иванов Б.Н. Дискретная математика. Алгоритмы и программы. - М.: Лаборатория Базовых Знаний, 2001.
32. Новиков Ф.А. Дискретная математика для программистов. - СПб.: Питер, 2001.