**Тема18.Стандартні консольні потоки введення/виведення. Використання файлів для введення/виведення даних. Класи ofstream та ifstream. Бінарні та текстові файли. Обробка виключень.**

Зі стандартними консольними потоками введення/виведення ми вже давно працюємо, але не в повній мірі використовуючи їх потенціал. На минулих лекціях ми розглядали роботу з файлами та оператори введення-виведення "<<" та ">>", узагальнимо отримані знання та розглянемо більш детально деякі можливості, оскільки вважається, що у мові C++ бажано здійснювати потокові введення-виведення даних..

**Операції введення і виведення.**

У мові C++ дії, що пов’язані з операціями введення і виведення, виконуються за допомогою функцій бібліотек. Функції ведення і виведення бібліотек мови дозволяють читати дані з файлів та пристроїв і писати дані у файли і на пристрої.

Бібліотека мови C++ підтримує три рівня введення-виведення даних:

* введення-виведення потоку;
* введення-виведення нижнього рівня;
* введення-виведення для консолі і порту.

При введенні-виведенні потоку всі дані розглядаються як потік окремих байтів. Для користувача потік — це файл на диску або фізичний пристрій, наприклад, дисплей чи клавіатура, або пристрій для друку, з якого чи на який направляється потік даних. Операції введення-виведення для потоку дозволяють обробляти дані різних розмірів і форматів від одиночного символу до великих структур даних. Програміст може використовувати функції бібліотеки, розробляти власні і включати їх у бібліотеку. Для доступу до бібліотеки цих класів треба включити в програму відповідні заголовні файли.

За замовчуванням стандартні введення і виведення повідомлень про помилки відносяться до консолі користувача (клавіатури та екрана). Це означає, що завжди, коли програма очікує введення зі стандартного потоку, дані повинні надходити з клавіатури, а якщо програма виводить дані — то на екран. У мові C++ існує декілька бібліотек, які містять засоби введення-виведення, наприклад: **stdio.h, iostream.h.**

**Для потокового введення даних** вказується операція «**>>**» («читати з»). Це перевантажена операція, визначена для всіх простих типів і покажчика на **char**. Кожна операція «>>» передбачає введення одного значення. При такому введенні даних необхідно дотримуватись конкретних вимог:

* для послідовного введення декількох чисел їх слід розділяти символом пропуску (« ») або **Enter** (дані типу **char** роздiляти пропуском необов’язково);
* якщо послідовно вводиться символ і число (або навпаки), пропуск треба записувати тільки в тому випадку, коли символ (типу **char**) є цифрою;
* потік введення ігнорує пропуски;
* для введення великої кількості даних одним оператором їх можна розташовувати в декількох рядках (використовуючи Enter);
* операція введення з потоку припиняє свою роботу тоді, коли всі включені до нього змінні одержують значення.

Оскільки в цьому прикладі пропуск є роздільником між значеннями, що вводяться, то при введенні рядків, котрі мiстять пропуски у своєму складі, цей оператор не використовується. У такому випадку треба застосовувати функції **getline(), get()** тощо (див. *розділ 7*). У мові C++ бажано здійснювати потокові введення-виведення даних.

**Для потокового виведення даних** необхідна операція **«<<»** («записати в»), що використовується разом з ім’ям вихідного потоку **cout**. Як відомо,

формат запису операції виведення представляється як:

**cout << data [<< data1];**,

де **data, data1** — це змінні, константи, вирази тощо.

Потокова операція виведення може мати вигляд:  
**cout << “у =” << x + а – sin(x) << “\n“;.**

Застосовуючи логічні операції, вирази треба брати в дужки:  
**cout << “р =” << (а && b || с) << “\n“;.**

Символ переведення на наступний рядок записується як рядкова константа, тобто **“\n”**, інакше він розглядається не як символ керуючої послідовності, а як число 10 (код символу). Таких помилок можна уникнути шляхом присвоювання значення керуючих символів змінним, тобто:

**#define << sp " "**

**#define << ht "\t"**

**#define << hl "\n".**

Тепер операцію виведення можна здійснити так:  
**cout <<** **“у =” << x** **+ а – sin(x)<<** **hl;** .

Потрібно пам’ятати, що ***при виведенні даних з використанням «cout <<» не виконується автоматичний перехід на наступний рядок, для реалізації такого переходу застосовується так переведення рядка*** ***“\n” або операція*** ***endl.***

**Використання прапорців стану**

Для додаткового керування даними, що виводяться, використовують маніпулятори **setw(w)** та **setprecision(d)**. Маніпулятор **setw(w)**призначений для зазначення довжини поля, що виділяється для виведення даних (**w** — кількість позицій). Маніпулятор **setprecision(d)** визначає кількість позицій у дробовій частині дійсних чисел. Маніпулятори змінюють вигляд деяких змінних в об’єкті **cout**, що у потоці розташовані за ними. Ці маніпулятори називають *прапорцями стану*. Коли об’єкт посилає дані на екран, він перевіряє прапорці, щоб довідатися, як виконати завдання

***Приклад 1.***  Використання прапорців стану для виведення даних.

**#include <iostream>**

**#include <iomanip>**

**#include <conio.h>**

**#include <Windows.h>**

**using namespace std;**

**main ( )**

**{system("color F0");**

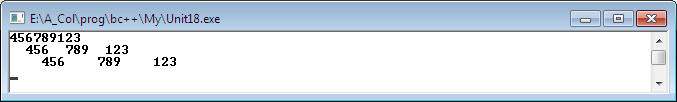
**cout << 456 << 789 << 123 << endl;**

**cout << setw(5) << 456 << setw(5) << 789 << setw(5) << 123 << endl;**

**cout << setw(7) << 456 << setw(7) << 789 << setw(7) << 123 << endl;**

**getch (); }**

Результати виконання програми:



У цьому використано заголовний файл **iomanip.h**, що дозволяє застосовувати функції маніпуляторів. При використанні функції **setw()** число вирівнюється вправо в межах заданої ширини поля виведення. Якщо ширина недостатня, то вказане значення ігнорується.

Функція **setprecision(2)** повідомляє про те, що число з плаваючою крапкою виводиться з двома знаками після крапки з округленням дробової частини, наприклад, при виконанні операції

**cout** **<< setw(7) << setprecision(2) << 123.456789;**

буде отримано такий результат: **123.46.**

Функції **cout.width(w)** та **cout.precision(d)**, які потребують підключення тільки заголовного файла **iostream.h**, виконують дії, подібні тим, що і функції **setw(w)** та **setprecision(d)**.

Операція введення використовує ті ж самі маніпулятори, що й операція виведення. Список змінних, в які будуть поміщені дані, визначений у **values**.

***Приклад 4.3.*** Написати програму обчислення податку на продаж.

**#include <iostream>**

**#include <iomanip>**

**#include <conio.h>**

**#include <Windows.h>**

**using namespace std;**

**main ( )**

**{system("color F0");**

**float prod\_sum; // prod\_sum – сума продаж**

**float nalog;**

**//--------------- виведення підказки для користувача**

**cout << "Enter the amount of sales ";**

**cin >> prod\_sum;**

**//............... обчислення**

**nalog = prod\_sum\* 0.7;**

**cout << " " << setprecision(2) << prod\_sum;**

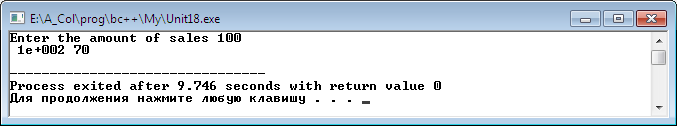
**cout << " " << setprecision(2) << nalog << "\n";**

**getch(); // затримка екрану**

**return 0;**

**}**

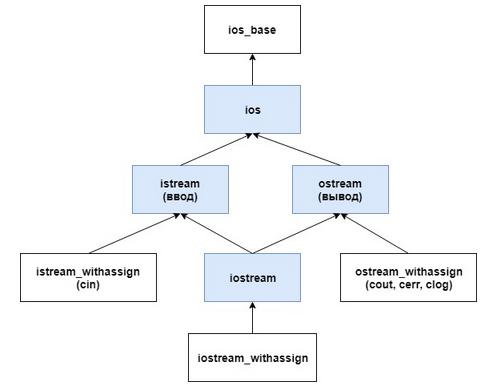
Результати виконання програми:



Унаслідок того, що у першому операторі **cout** відсутня інструкція переведення рядка, відповідь користувача на підказку (тобто введене значення змінної **prod\_sum**) з’явиться відразу праворуч за самою підказкою.

**Бібліотека стандартних консольних потоків введення/виведення.**

При підключенні заголовного файлу iostream отрмуємо доступ до всієї ієрархії класів бібліотеки iostream, які відповідають за функціонал введення/виведення даних (включно з класом, який називається iostream). Ієрархія цих класів виглядає так:



Як бачимо, ця ієрархія побудована на множинному наслідуванні. Бібліотека iostream була розроблена і ретельно протестована для унеможливлення типових помилок, які виникають при роботі з множинним наслідуванням.

## Потоки в С++

У всіх назвах класів присутнє слово «stream» ( «потік»). По суті, введення/виведення в C++ реалізоване за допомогою потоків. Абстрактно, **потік** — це послідовність символів, до якої можна отримати доступ. В часі потік може виробляти або споживати потенційно необмежені обсяги даних.

**Потік введення** (або «**вхідний потік**») використовується для збереження даних, отриманих від джерел даних: клавіатури, файлу, мережі тощо. Наприклад, користувач може натиснути клавішу на клавіатурі в той час, коли програма не очікує введення. Замість ігнорування натискання клавіши, дані поміщуються до вхідного потоку, де очікують відповіді від програми.

Та навпаки, **потік виведення** (або «**вихідний потік**») використовується для збереження даних, які надаються конкретному споживачу даних: монітору, файлу, принтеру тощо. При запису даних на пристрій виведення, цей пристрій може бути не готовим негайно отримати дані, наприклад, принтер все ще може прогріватися, коли програма вже записує дані до вихідного потоку. В цьому випадку дані будуть знаходитися в потоці виведення доки принтер не почне їх використовувати.

Деякі пристрої, як файли та мережі, можуть бути джерелами як введення, так і виведення даних. Але програмісту не потрібно знати деталі взаємодії потоків з різними пристроями та джерелами даних, а тільки вміти взаємодіяти з цими потоками для читання та запису даних.

**Основні властивості потоків**

Потоки мають деякі властивості, які визначають, які функції можуть бути використані для організації введення / виводу і яким чином буде здійснюватися обмін даними через потоки введення або виведення. Більшість з цих властивостей визначаються в момент, коли потік, пов'язаний з файлом, відкритий за допомогою функції fopen.

* **Доступ потоку до читання або запису.** Ця властивість визначає, чи має даний потік доступ до читання і (або) записи на фізичних носіях.
* **Текст або двійковий код.** Потоки, як вважається, являють собою набір текстових рядків, кожна з яких закінчується символом нового рядка. Залежно від середовища, в якій додаток запускається, символи нового рядка можуть відрізнятися, тому виникає необхідність адаптувати деякі спеціальні символи в текстовому файлі, згідно специфікаціям використовуваної системи. З іншого боку, двійковий потік - це послідовність символів, записувана або прочитується з фізичного середовища без всякого перетворення даних.
* **Буфер тимчасового зберігання даних.** Буфер блоку пам'яті, де дані накопичуються, перш ніж фізично зчитуються або записуються на відповідний файл або пристрій. Потоки можуть бути або з повною буферизацією, або без буферизації. Якщо повна буферизація, то дані читання / запису фізично переносяться або змінюються, коли буфер заповнюється. Буфер вважається заповненим, якщо в потік потрапляє символ нового рядка. Небуферізованних потоки символів, також призначені для читання / запису, але буферизація в них виконується, по можливості, якомога швидше.

## Введення/виведення даних в C++

Хоч клас ios є похідним класу ios\_base, дуже часто саме з цим класом найчастіше працюють/взаємодіють напряму. Клас ios визначає багато спільних аспектів для потоків введення/виведення.

**Клас istream** використовується для роботи з вхідними потоками. **Оператор витягування** >> використовується для витягування значень з потоку. Коли користувач натискає на клавішу клавіатури, код цієї клавіши поміщується до вхідного потоку, потім програма витягує це значення з потоку та використовує його.

**Клас ostream** використовується для роботи з вихідними потоками. **Оператор вставлення <<** використовується для поміщення значень до потоку. Коли програма вставляє свої значення до потоку, а потім споживач даних (наприклад, монітор) використовує їх.

**Клас iostream** може обробляти як введення, так і виведення даних, що дозволяє йому здійснювати двоспрямоване введення/виведення.

3 класи, що закінчуються на «\_withassign», є потоковими класами похідними від класів istream, ostream и iostream (відповідно). Зазвичай з ними напряму не працюють.

Бібліотека С++ забезпечує програміста механізмами для отримання даних з потоків і для приміщення даних в потоки. Кожен потік (за винятком рядкових) асоціюється за допомогою операційної системи з певним зовнішнім пристроєм. При обміні з потоком використовується допоміжний ділянку пам'яті, званий буфером потоку. При введенні даних вони спочатку поміщаються в буфер і тільки потім передаються виконуваної програмі. При виведенні - дані заповнюють буфер перед передачею зовнішнього пристрою. Заповнення та очищення буферів операційна система виконує без явного участі програміста, тому потік в прикладній програмі можна розглядати як послідовність байтів, не залежну від конкретних пристроїв, з якими ведеться обмін даними.

Всі потоки бібліотеки послідовні, тобто в кожен момент для потоку визначені позиції записи і (або) читання, які після обміну переміщуються по потоку на довжину переданої порції даних.

В залежності від реалізованого напрямку передачі даних потоки ділять на три групи:

・ Вхідні, з яких читається інформація;

・ Вихідні, в які вводяться дані;

・ Двонаправлені, що допускають як читання, так і запис.

Відповідно до особливостей обслуговується пристрою потоки прийнято ділити на наступні групи:

・ Стандартні, для передачі даних від клавіатури і до дисплея (у всіх попередніх програмах використовувалися потоки даної групи);

・ Файлові, при розміщенні інформації на зовнішніх носіях (наприклад, диск, магнітна стрічка);

・ Рядкові, що дозволяють розміщувати дані потоку в пам'яті (символьний масив або рядок) і користуватися при цьому всіма засобами, наданими бібліотекою потоків (наприклад, форматний введення-виведення даних).

**Стандартні потоки в C++**

**Стандартний потік** — це попередньо підключений потік, який надається програмі її оточенням. C++ постачається з 4-ма попередньо визначеними стандартними об’єктами потоків, які можна використовувати:

**cin** — клас istream\_withassign, зв’язаний зі стандартним введення (зазвичай це клавіатура);

**cout** — клас ostream\_withassign, зв’язаний зі стандартним виведенням (зазвичай це монітор);

**cerr** — клас ostream\_withassign, зв’язаний зі стандартною помилкою (зазвичай це монітор) та, який забезпечує небуферизоване виведення;

**clog** — класс ostream\_withassign, зв’язаний зі стандартною помилкою (зазвичай це монітор) та, який забезпечує буферизоване виведення.

Небуферизоване виведення зазвичай оброблюється одразу, тоді як буферизоване зазвичай зберігається та виводиться як блок. Оскільки clog використовується рідко, то його зазвичай ігнорують.

**Переваги системи вводу-виводу С++**

Бібліотека вводу-виводу розроблена засобами об'єктно- орієнтованого програмування і є бібліотекою класів. Основні переваги такого підходу полягають в наступному.

По-перше, класам потоків доступний надійний механізм контролю типів переданих даних, заснований на перевантаженні операцій (що забезпечує для кожного типу даних виклик відповідної функції обробки). Таким чином, саме класами інтерпретуються ≪сирі≫ послідовності байтів відповідних потоків.

По-друге, класи потоків, завдяки поліморфізму, дозволяють одним і тим же функцій працювати з потоками різних типів, а бібліотеці в цілому -підтримувати однаковий інтерфейс вводу-виводу. Наприклад, інтерфейс, використовуваний стандартним вводом-виводом, застосовуємо також до файлових і строкових потоків.

По-третє, бібліотека вводу-виводу розширювана. Це означає, що, з одного боку, класи бібліотеки можуть працювати з новими типами даних, визначеними програмістом, використовуючи згаданий механізм перевантаження операцій, а з іншого - програміст має можливість доповнювати бібліотеки.

На перший погляд, система вводу-виводу С++ є досить громіздкою. Проте, “класовий” ввід-вивід має ряд переваг:

1) *надійність* пов'язана з підтримкою контролю типів. Сімейства функцій printf/scanf не передбачають ніякого контролю типів даних. Компілятор не попереджає про невідповідність типів значень та відповідних специфікацій. Механізм потоків у С++ , який ґрунтується на перевантаженні функцій та операторів, забезпечує для кожного типу даних свою перевантажену функцію.

2) *розширюваність*. Завдяки поліморфізму одні i ті ж самі оператори можуть взаємодіяти з потоками різних типів. Легко можна написати і перевантажити функції , які будуть працювати з специфічними типами даних користувача.

3) *простота та послідовність*. Використання перевантажених функцій дає можливість використовувати однотипний інтерфейс вводу/виводу.

Розглянемо, як можна використовувати класи вводу-виводу.

***Класи потоків C + +***

Узагальнимо наші відомості про класипотоків C + +:

ios - базовий потоковий клас;

strstreambase - базовий клас строкових потоків;

fstreambase - базовий клас файлових потоків;

istream - клас вхідних потоків;

ostream - клас вихідних потоків;

iostream - клас двонаправлених потоків вводу-виводу;

ifstream - клас вхідних файлових потоків;

ofstream - клас вихідних файлових потоків;

fstream - клас двонаправлених файлових потоків;

istrstream - клас вхідних строкових потоків;

ostrstream - клас вихідних строкових потоків;

strstream - клас двонаправлених строкових потоків.

Клас ios є базовим для всіх класів потоків і, таким чином, містить всі загальні засоби потоків. Наприклад, за допомогою методів і даних класу ios здійснюється управління процесом передачі даних з буфера і в буфер. При виконанні цих дій необхідні, наприклад, відомості про потрібний підставі числення, про точність подання дійсних чисел і т.д. Клас ios містить ці відомості, тобто дані, що відносяться до станів потоків, і методи, що дозволяють змінювати ці властивості.

Буферизація даних потоків реалізована за допомогою наступного механізму: клас ios, а отже, і похідні класи містять покажчик на об'єкт класу streambuf бібліотеки - базового класу, що забезпечує свої похідні класи загальними методами для буферизації даних. У свою чергу його похідні класи (filebuf і strstreambuf) забезпечують взаємодію створюваних потоків з фізичними пристроями. Дана паралельна ієрархічна структура класів-буферів не показана на малюнку через досить рідкісною необхідності звернення до методів і даних класу streambuf безпосередньо з прикладних програм. Натомість частіше відбувається опосередковане їх використання викликом методів класів - спадкоємців ios.

Класи потоків, їх дані і методи стають видимими і доступними в програмі, якщо в неї включений відповідний заголовний файл:

iostream.h -для класів ios, istream, ostream, stream;

fstream.h -для класів fstreambase, ifstream, ofstream, fstream;

strstrea.h - для класів strstreambase, istrstream, ostrstream, strstream.

Оскільки клас ios є базовим для інших потокових класів, то включення в текст програми будь-якого з заголовних файлів fstream.h або strstrea.h автоматично підключає до програми і файл iostream.h. Відповідні перевірки виконуються на етапі препроцесорну обробки.

**Файлові потоки**

Включається файл:

# іnclude <fstream>

Файлові потоки реалізовані наступними класами:

/ / Файлові потоки введення

typedef basic\_ifstream <char> ifstream;

typedef basic\_ifstream <wchar\_t> wifstream;

/ / Файлові потоки виводу

typedef basic\_ofstream <char> ofstream;

typedef basic\_ofstream <wchar\_t> wofstream;

/ / Файлові потоки введення / виводу

/ / Спадкоємці від попередніх

typedef basic\_fstream <char> fstream;

typedef basic\_fstream <wchar\_t> wfstream;

***Методи файлових потоків***

**close ()** - закриває файл;

**open (const char \* s, ios\_base::openmode n, long protection = 0666)** -

відкриває файл;

**bool is\_open ()** - визначає чи відкритий файл.

Конструктор файлового потоку може замінити метод open. Значення режиму відкриття файлу задаються вкладеним типом **open\_mode**:

typedef int openmode; / / режим відкриття потоку

enum open\_mode {

app = 0x01, / / завжди писати в кінець файлу

binary = 0x02, / / введення / виведення в двійковому режимі

in = 0x04, / / відкрити для введення

out = 0x08, / / відкрити для виведення

trunc = 0x10, / / знищити дані після відкриття існуючого потоку

ate = 0x20 / / відкрити позиціонуючи покажчик на кінець файлу

};

**Текстовий та двійковий режим**

В двійковому режимі при операціях введення / виводу особливі символи не замінюються (типу кінця рядка), що дозволяє правильно обробити двійкові файли (наприклад, файли \*. Bmp). При цьому для читання даних використовується метод read, а для запису - метод write.

За замовчуванням потоки відкриваються в текстовому режимі і в залежності від типу потоку для введення, виведення або введення / виводу. Так у виклику ofstream fout ("out.txt", ios\_base :: out | ios\_base :: app);

ios\_base :: out | можна опустити.

**Бінарні та текстові файли.**

Коли ми говоримо про бінарні дані, тип char виступає в ролі типу byte, але

в С++ нема типу byte , тому замість типу byte використовують тип char. За правилам мови С++, тип char є найменшою одиницею інформації. Це може спотворювати нам сприйняття ситуації, оскільки ми бачимо char\*, а на справді це як byte\*.

В наших комп’ютерах використовується така модель пам’яті, де одна комірка займає 1 байт. А один символ char гарантовано згідно зі стандартом повинен вміщуватися в цю комірку. Тому, відповідно до ситуації, тип char можна трактувати або як символьний тип, або як байтовий тип. Файли, які зберігають послідовність байтів, називають ***бінарними файлами***. Це пов’язано з тим, що байт ділиться на біти, а біти – це двійкові цифри. Слово бінарний перекладається як двійковий.

Щоб записати будь-яке значення в бінарному представленні, потрібно спочатку вивести це бінарне представлення, а щоб записалося правильна кількість байтів, потрібно явно вказати цю кількість. Це може виглядати так:

***(char\*)&x***

Так ми створюємо рядок байтів для того, щоб віддати потоку, відкритому у двійковому режимі. Через ***sizeof(x)*** ми обмежуємо кількість байтів, що уходять в потік, потрібним числом.

Таким чином, при запису бінарного файлу потрібно виконувати: 1) виводите рядок байтів; 2) слідкуєте за його довжиною.

Тут є аналогія із символьним масивом, але оскільки у нас не символьний рядок, а байтовий, то обмеження виконується не за рахунок нуль-символу, а явною вказівкою кількості байтів. В залежності від оголошеного тиру змінної, вона може займати від одного до декількох байтів в пам’яті комп’ютера, и при записи змінних ми повинні уважно слідкувати, щоб число байтів, вказане нами, строго співпадало с числом байт, які займає змінна в пам’яті. Ми переміщуємо байти з пам’яті до файлу. Щоб байти переносилися саме як байти, ми інтерпретуємо змінну як рядок байтів.

Приклад.

//clang     Работа с бинарными файлами    Чтение текстового файла

#include <fstream>

#include <iostream>

using namespace std;

int main() {

   const char\* FName = "C:\\MyFiles\\text.txt";  //Путь к файлу

   char S[255] = {};     //Символьный массив в 255 символов

    /\*НАЧАЛО РАБОТЫ С ФАЙЛОМ\*/

   ifstream in(FName,ios::binary);

        in.read((char\*)&S, sizeof(S));

//перенос байтов из файла в "х"

        //in.read((char\*)S, sizeof(S));

//в случае с массивом можно и так

   in.close();

   /\*КОНЕЦ РАБОТЫ С ФАЙЛОМ\*/

    cout << S << '\n';

    cin.get();

}

Для того, щоб правильно читати дані з файлів, які зберігають ці дані в бінарному вигляді, потрібно знати загальну структуру збереження в середині файлу.

Підводний камінь може очікувати при спробі записати до файлу значення, на яке посилається покажчик. В цьому випадку до файлу буде записуватися не значення змінної, на яку вказує покажчик, а значення покажчика, тобто адреса.

Це одне зауваження щодо запису до файлу. Якщо до файлу об’єкт записується цілком, то читати його краще цілком, а не частинами. Сучасні комп’ютери вирівнюють поля класів, щоб робота зі значеннями, які зберігаються в об’єктах, виконувалась швидше. Таким чином, коли ви записуєте об’єкт цілком, то читаючи його цілком, не потрібно замислюватися о вирівнюванні. А. якщо спробувати витягнути окремі частини збереженого об’єкту, то можливо зробити помилку із-за неспівпадіння вашого уявлення про структуру збережених даних з реальною ситуацією.

Потрібно зауважити, що для комп’ютерів нема різниці між бінарними файлами та текстовими файлами. Текстові файли — це лише особливий випадок бінарних файлів.

**Введення / виведення бінарних файлів стандартними засобами**

У файловій системі мови С передбачено дві функції  fread()  і  fwrite(), які дають  змогу  зчитувати  і  записувати  блоки  даних. Ці функції подібні С++ функціям read() і write(). Їх прототипи мають такий вигляд:

**size\_t fread(void \*buffer, size\_t num\_bytes, size\_t count, FILE \*fp);**

**size\_t fwrite(const void \*buffer, size\_t num\_bytes size\_t count, FILE \*fp);**

Під  час  виклику  функції  fread()  параметр  buffer  є  покажчиком на область пам'яті, яка призначена для прийому даних, які зчитуються з файлу. Функція fread() повертає кількість зчитаних об'єктів, яка може опинитися менше заданого  значення  count,  якщо  у  процесі  виконання  цієї операції виникла помилка або був досягнуто кінець файлу.

Під час виклику функції  fwrite() параметр buffer є покажчиком на інформацію, яка підлягає запису у файл. Ця функція записує count об'єктів завдовжки um\_bytes у потік, яка адресується файловим покажчиком  fp. Функція fwrite() повертає кількість записаних об'єктів, яка буде дорівнює значенню count, як-що у процесі виконання цієї операції не було помилки.

Якщо під час виклику функцій fread() і fwrite() файл був відкрито для виконання двійкової операції, то вони можуть зчитувати або  записувати  дані будь-якого  типу.

**Приклад 6.** Запис значення типу float у двійковий (бінарний) файл.

#**include "stdafx.h"**

**int main()**

**{**

**#include <iostream>**

**#include <fstream>**

**#include <Windows.h>**

**using namespace std;**

**int main()**

**{system("color F0");**

**FILE \*fp;**

**float f = 12.23F;**

**if((fp=fopen("test","wb"))==NULL)**

**{**

**printf("File is not opened!" );**

**return 1;**

**}**

**fwrite(&f,sizeof(float), 1, fp);**

**fclose(fp);**

**Sleep(2000);**

**return 0;**

**}**

**Приклад 5. Запис масиву у бінарний файл.**

**#include <iostream>**

**#include <fstream>**

**#include <Windows.h>**

**int main()**

**{ system("color F0");**

**int i;**

**FILE \*fp;**

**float balance[100];**

**//Відкриваємо файл для запису.**

**if((fp=fopen("balance","w"))==NULL)**

**{**

**printf("Не вдається відкрити файл");**

**return 1;**

**}**

**for(int i=0; i<100; i++) balance[i] = (float) i;**

**//зберігаємо весь масив balance.**

**fwrite(balance, sizeof balance, 1, fp);**

**fclose(fp);**

**//Обнуляємо масив.**

**for(int i=0; i<100; i++) balance[i] = 0.0;**

**//Відкриваємо файл для зчитування.**

**if((fp=fopen("balance","r"))==NULL)**

**{**

**printf("Не вдається відкрити файл");**

**return 1;**

**}**

**//зчитуємо весь масив balance.**

**fread(balance, sizeof balance, 1, fp);**

**// Відображаємо вміст масиву.**

**for(int i=0; i<100; i++) printf("%f ", balance[i]);**

**fclose(fp);**

**return 0;**

**}**

**Робота з текстовим файлом як потоком**

Ці питання ми постійно опрацьовуємо на лабораторних, тому приклади розглядати не будемо.

Оголошення файлової змінної та відкриття текстового файлу для читання має наступний синтаксис:

**ifstream f\_in(“filename.txt”, ознака1|ознака2|…|ознакаN);**

У даному записі f\_in – назва файлової змінної, filename.txt –назва файлу на диску. Ознаки слугують для визначення прав доступу до файлу. Перелік ознак відкриття файлу та їх значень наведено в табл. 1.

Оголошення файлової змінної та відкриття файлу для запису має синтаксис:

**ofstream f\_out(“filename.txt”, ознака1|ознака2|…|ознакаN);**

У даному записі f\_out – назва файлової змінної, filename.txt –  назва файлу на диску.

Наприклад, для того, щоб створити текстовий файл і записати в нього рядок «Робота в С++» необхідно виконати наступні кроки:

1. створити об’єкт класу ofstream, наприклад, з назвою fout:
2. пов’язати об’єкт класу з файлом, в який буде проводитися запис:
3. записати рядок в файл;
4. закрити файл:

**ofstream fout;**

**fout.open("cppstudio.txt"); // це можна було зробити, записавши**

**// ofstream fout(“cppstudio.txt ”, app);**

**fout << “ Робота в С++ ”;**

**fout.close();**

Для того щоб прочитати створений файл, знадобитися виконати аналогічні кроки, що й при запису у файл з невеликими змінами:

1. створити об'єкт класу ifstream і пов'язати його з файлом, який відкриємо в режимі для читання:
2. прочитати файл;
3. закрити файл.

**ifstream fin("cppstudio.txt");**

**fin >> buff;**

**fin.close().**

**Виключні або виняткові ситуації**

У мові С++ існує механізм обробки виняткових ситуацій. У ході виконання програми можуть виявитися різні помилки. Вони можуть бути позв'язані з неправильним програмуванням (наприклад, вихід індексу масиву за межі припустимого чи переповнення пам'яті), а іноді їхня причина не залежить від програміста (скажемо, розрив зв'язку при мережевому з'єднанні). У кожній з цих ситуацій реакція програми непередбачена. Іноді вона завершує виконання, і лише після закінчення деякого інтервалу часу починають позначатися наслідку помилки, а частіше програма негайно припиняє роботу, піддаючи ризику дані, що знаходяться в пам'яті чи у файлі. Якщо не передбачити акуратне завершення роботи, використовуючи обробку виняткових ситуацій, результати можуть виявитися неприємними. В подальшому будемо називати винятковою ситуацією будь-яку подію, що вимагає особливої обробки. При цьому зовсім неважливо, чи є ця подія фатальною чи простою помилкою. Перевірка умов, що описують виняткову ситуацію, і реакція на її виникнення називається обробкою виняткової ситуації. Ця задача покладається на оброблювача виняткової ситуації.

**Механізм обробки виняткових ситуацій**

Обробка виняткових ситуацій у мові С++ є об‘єктно-орієнтованою. Це значить, що виняткова ситуація є об'єктом, що генерується при виникненні незвичайних умов, передбачених програмістом, і передається оброблювачу, що неї перехоплює. Об'єктом, що описує природу виняткової ситуації, може бути будь-як сутність — літерал, рядок, об'єкт класу, число і т.д. Виняткова ситуація не обов'язково повинна бути об'єктом якого-небудь класу. В основі обробки виняткових ситуацій у мові С++ лежать три ключових слова: **try**, **catch** і **throw**.

Якщо програміст підозрює, що визначений фрагмент програми може спровокувати помилку, він повинний занурити цю частину коду в блок *try*. Необхідно мати на увазі, що зміст помилки (за винятком стандартних ситуацій) визначає сам програміст. Це значить, що програміст може задати будь-яку умову, що приведе до створення виняткової ситуації. Після цього необхідно вказати, у яких умовах варто генерувати виняткову ситуацію. Для цієї мети призначене ключове слово *throw*. І нарешті, виняткову ситуацію потрібно перехопити й обробити в блоці *catch*. Ось як виглядає ця конструкція.

**try {**

**// Тіло блоку**

**try if(умова)throw виняткова\_ситуація**

**}**

**catch(тип1 аргумент)**

**{**

**// Тіло блоку catch**

**}**

**catch(тип2 аргумент)**

**{**

**// Тіло блоку catch**

**}**

**. . .**

**catch(типN аргумент)**

**{**

**// Тіло блоку catch**

**}**

Розмір блоку try не обмежений. У нього можна занурити як один оператор, так і цілу програму. Один блок ***try*** можна зв'язати з довільною кількістю блоків ***catch***. Оскільки кожен блок catch відповідає окремому типу виняткової ситуації, програма сама визначить, який з них виконати. У цьому випадку інші блоки catch не виконуються. Кожен блок catch має аргумент, що приймає визначене значення. Цей аргумент може бути об'єктом будь-якого типу.

Якщо програма виконана правильно й у блоці **try** не виникло жодної виняткової ситуації, усі блоки ***catch*** будуть зігноровані. Якщо в програмі виникла подія, що програміст вважає небажаною, оператор ***throw*** генерує виняткову ситуацію. Для цього оператор **throw** повинний знаходитися усередині блоку ***try*** або усередині функції, викликуваної усередині блоку ***try***.

Якщо в програмі виникла виняткова ситуація, для якої не передбачені перехоплення й обробка, викликається стандартна функція ***terminate()***, що, у свою чергу, викликає функцію ***abort().*** Утім, іноді виняткова ситуація не є небезпечною. У цьому випадку можна виправити помилку (наприклад, привласнити нульовому знаменнику ненульове значення) і продовжити виконання програми.

**Приклади обробки виняткової ситуації**

Розглянемо найпростіший приклад обробки виняткової ситуації

**#include <iostream>**

**#include <Windows.h>**

**using namespace std;**

**int main()**

**{system("color F0");**

**int n = 10, m = 0;**

**printf("Begin\n");**

**try {**

**printf("In try block\n");**

**if(m==0) throw "Divide by zero";**

**else n=n/m;**

**printf("The rest of the block is not executed!");**

**}**

**catch (const char\* s)**

**{**

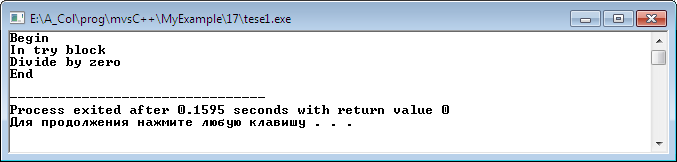
**printf("%s\n",s);**

**}**

**printf("End\n"); return 0;**

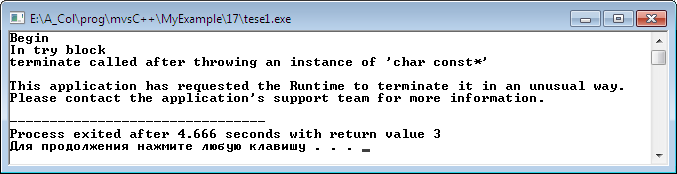
**}**

Ця програма виводить на екран наступні рядки.



Простежимо за потоком керування при виконанні цієї програми. Спочатку з'являються і ініціалізуються дві цілочисельні змінні (одна з них дорівнює нулю). Потім виводиться повідомлення про початок виконання програми, і потік керування входить у блок ***try.*** Після виводу рядка повідомлення про вхід у блок try, потік керування переходить до перевірки рівності ***m==0***. Оскільки ця рівність є істиною, генерується виняткова ситуація (у даному випадку — константний рядок). Керування негайно передається блоку ***catch***, аргументом якого є константний символьний вказівник, ігноруючи всі інші оператори в блоці ***try***. У цій програмі блок **catch** не робить жодних спроб виправити помилку. Замість цього він просто видає повідомлення — рядок, отриманий як аргумент — і передає керування оператору, що слідує за блоком. На закінчення функція printf() виводить на екран рядок **End**, і програма завершує свою роботу. Тип виняткової ситуації повинний збігатися з типом аргументу розділу **catch**.

Поглянемо, що відбудеться, якщо цією умовою зневажити. Порушення угоди про тип виняткової ситуації можна викликати заміною оператору **catch (const char\* s)** на **(const char s).** Отримаємо результат:



У цій програмі ми зробили цілком ― природну‖ помилку — забули поставити зірочку в оголошенні аргументу. Тепер блок catch очікує виняткову ситуацію, що представляє собою константний символ, а не вказівник. Ця помилка приводить до аварійного завершення роботи програми. Покажемо, що відбудеться, якщо виняткова ситуація генерується усередині функції, яка викликається в блоці **try**. Виняткова ситуація, згенерована усередині функції **#include <iostream>**

**#include <Windows.h>**

**using namespace std;**

**int Denominator(int);**

**int main()**

**{ system("color F0");**

**int n = 10, m; printf("Begin\n");**

**try { printf("In try block\n");**

**m=Denominator(0);**

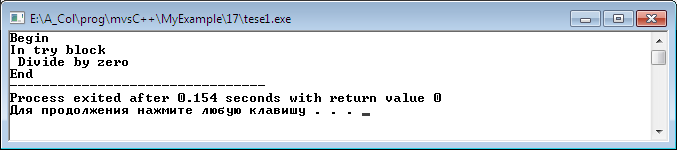
**printf("The rest of the block is not executed!"); }**

**catch (const char\* s) { printf("%s\n",s); }**

**printf("End"); return 0; }**

**int Denominator(int i) { if(i==0)throw " Divide by zero"; return i; }**

Результат:



У цій програмі виняткова ситуація генерується у функції Denominator(), яка викликається в блоці **try**. Завдяки цьому результати роботи програми цілком збігаються з попередніми. Якщо блок **try** знаходиться усередині функції, обробка виняткової ситуації виконується при кожнім виклику.

**Розміщення блоку try усередині функції**

Приклад.

**#include <iostream>**

**#include <stdio.h>**

**#include <Windows.h>**

**using namespace std;**

**int Denominator(int);**

**int main()**

**{ system("color F0");**

**int n = 10, m; printf("Begin\n");**

**m=Denominator(0);**

**n = Denominator(11);**

**printf("End"); return 0; }**

**int Denominator(int i)**

**{ printf("In Denominator \n");**

**try {**

**printf("In try block\n");**

**if(i==0) throw("Divide by zero!");**

**if(i>10) throw 10;**

**printf("The rest of the block is not executed!");**

**}**

**catch (const char\* s)**

**{**

**printf("%s\n",s);**

**}**

**catch (int n)**

**{**

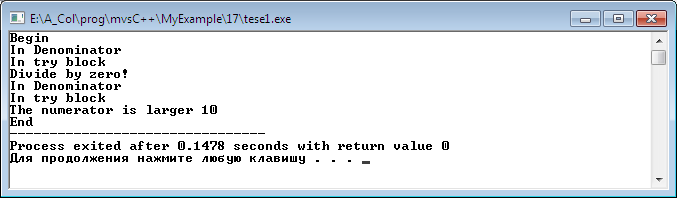
**printf("The numerator is larger %d\n",n);**

**}**

**return i;**

**}**

Результат



*Література* [1, 4, 6, 11, 12, 18]

*Контрольні запитання для самоперевірки*.

1. Які види потоків ви знаєте?
2. Для чого призначені стандартні потоки?
3. Який файл необхідно включати у програму для використання стандартних потоків?
4. Назвіть класи вхідних та вихідних потоків.
5. Який клас є базовим для потоків?
6. Для чого призначені маніпулятори потоків, і який файл необхідно включати у програму для їх використання?
7. Для чого призначені маніпулятори endl, setprecision, setw?
8. Які є файли за способом доступу?
9. Назвіть класи для створення файлових потоків і їх призначення.
10. Які дії необхідно виконати для використання файлів у програмі?
11. Які існують режими відкриття файлів?
12. Для чого призначений метод open() і які він має параметри?
13. Які способи відкриття та закриття файлу ви знаєте?
14. В чому різниця між бінарним та текстовим файлом?
15. Як записати дані у бінарний файл?
16. Як прочитати дані з бінарного файлу?
17. Що таке виключні ситуації.
18. Синтаксис виключних ситуацій.
19. Як контролюються виключні ситуації?
20. Як відбувається перехват виключних ситуацій?

*Контрольні запитання для надання письмових відповідей*.

1. Якщо Ви будете записувати до файлу дані об’єкту похідного класу, то чи будуть записані заповнені поля успадковані з базового класу?
2. Наведіть власний приклад запису даних до бінарного файлу та читання з нього.