Тема 18: C++ - система введення-виведення

C++-система введення-виведення побудована на ієрархії класів.

ЗМІСТ

1. **Потоки C++**
2. **Класи потоків**
3. **Перевантаження введення-виведення**
4. **Форматоване введення-виведення даних**
5. **Використання маніпуляторів вводу-виводу**
6. **Файлове введення-виведення**
7. **Неформатоване введення-виведення даних у двійковому режимі**

Порівняння старої й нової C++-систем вводу-виводу

Існують дві версії бібліотеки об’єктно-орієнтованого вводу-виводу:

- більш стара, заснована на оригінальних специфікаціях мови C++,
- нова, визначена стандартом мови C++.

Стара бібліотека вводу-виводу підтримується заголовним файлом **<iostream.h>**, а нова - **<iostream>**.

Порівняння старої й нової C++-систем вводу-виводу

З погляду програміста, є два істотних розходження між **<iostream.h>** і **<iostream>**:

1. Нова бібліотека містить ряд додаткових засобів і визначає кілька нових типів даних. Її можна вважати супермножиною старої. Практично всі програми, написані для старої бібліотеки, успішно компілюються при використанні нової, не вимагаючи внесення яких-небудь значних змін.
2. Стара бібліотека вводу-виводу була визначена в глобальному просторі імен, а нова використовує простір імен **std**, що використовується всіма бібліотеками стандарту C++.

Потоки C++

Потік (stream) - це загальний логічний інтерфейс із різними пристроями, що складають комп'ютер.

Потік або синтезує інформацію, або отримує її й зв'язується з будь-яким фізичним пристроєм за допомогою C++-системи вводу-виводу.

Поводження всіх потоків однакове, для різних пристроїв.

Потік можна назвати логічним інтерфейсом з файлом, що є абстракцією дискового файлу, екрану, клавіатури, порту, файлу на магнітній стрічці та ін.

Хоча файли відрізняються за формою й можливостями, всі потоки однакові.

Потік зв'язується з файлом при виконанні операції відкриття файлу, а від'єднується від нього за допомогою операції закриття.

Потоки C++

Існує два типи потоків:

- текстовий
- двійковий

Текстовий потік використовується для вводу-виводу символів з можливим їх перетворенням.

Двійковий потік можна використовувати з даними будь-якого типу без перетворень.

Поточна позиція - це місце у файлі, з якого буде виконуватися наступна операція доступу до файлу.

Під файлом розуміють реальний фізичний пристрій, що містить дані.

Якщо файли розрізняються між собою, то потоки - ні.

Вбудовані C++-потоки

C++ має ряд вбудованих потоків (**cin, cout, cerr** і **clog**), які автоматично відкриваються, як тільки програма починає виконуватися.

cin - стандартний вхідний, а **cout** - стандартний вихідний потік.

Потоки **cerr** і **clog** призначені для виводу інформації про помилки і також зв'язані зі стандартним виводом даних.

Потік **clog** буферизований, а потік **cerr** – ні, тобто дані через потік **cerr** виводяться негайно.

Вбудовані C++-потоки

У C++ передбачені двохбайтові (16-бітові) символьні версії стандартних потоків, іменовані **`wcin`**, **`wcout`**, **`wcerr`** і **`wclog`** для підтримки мов з великими символьними наборами (китайська тощо).

За замовчуванням стандартні C++-потоки зв'язуються з консоллю, але програмним способом їх можна перенаправляти на інші пристрої або файли.

Перенаправляти їх може також операційна система.

Класи потоків

C++-система вводу-виводу побудована на різних ієрархіях шаблонових класів.

1. Виведена із класу низькорівневого вводу-виводу **basic_streambuf**, що підтримує базові низькорівневі операції уведення й виводу й забезпечує підтримку для всієї C++-системи вводу-виводу.
2. Виведена із класу високорівневого вводу-виводу **basic_ios**, що забезпечує форматування, контроль помилок і надає статусну інформацію, пов'язану з потоками вводу-виводу.

Клас **basic_ios,** виведений із класу ios_base, використовується в якості базового для кількох похідних класів, включаючи класи **basic_istream**, **basic_ostream** і **basic_iostream**.

Класи потоків

Ці класи використовуються для створення потоків, призначених для уведення даних, виводу й вводу-виводу відповідно.

Шаблонні класи	Символьні класи
basic_streambuf	**streambuf**
basic_ios	**ios**
basic_istream	**istream**
basic_ostream	**ostream**
basic_iostream	**iostream**
basic_fstream	**fstream**
basic_ifstream	**ifstream**
basic_ofstream	**ofstream**

Перевантаження операторів вводу-виводу

У C++ передбачений спосіб виконання операцій вводу-виводу "класових" даних шляхом перевантаження операторів вводу-виводу "**<<**" і "**>>**”:

Оператор "**<<**" називається оператором виводу або вставки, оскільки він вставляє символи в потік.

Оператор "**>>**" називається оператором уведення або добування, оскільки він витягає символи з потоку.

Оператори вводу-виводу вже перевантажені (у заголовку <iostream>) для операцій потокового уведення або виводу даних будь-яких вбудованих C++-типів.

Створення перевантажених операторів виводу

Є клас, що містить координати у відкритих членах: **x**, **y**, **z**

```
class three_d {
  public:
    int x, y, z; // 3-мірні координати
    three_d(int a, int b, int c) { x = a; y = b; z = c; }
};
```

Перевантажений оператор виводу "**<<**“ для об'єктів типу **three_d** реалізується такою операторною функцією:

```
ostream &operator<<(ostream &stream, three_d obj) {
  stream << obj.x << ", ";
  stream << obj.y << ", ";
  stream << obj.z << "\n";
  return stream; // повертає параметр потік stream
}
```

Створення перевантажених операторів виводу

1. Функція повертає посилання на об'єкт типу **ostream**. Це дозволяє кілька операторів виводу об'єднати в одному складеному виразі.
2. Перший параметр є посиланням на потік ліворуч оператора (лівий операнд).
3. Другий - це об'єкт, що стоїть праворуч цього оператора. (При необхідності другий параметр також може мати тип посилання на об'єкт.)

```
// Використання перевантаженого оператора виводу.
#include <iostream>
using namespace std;
class three_d {
  public:
    int x, y, z; // 3-мірні координати
    three_d(int a, int b, int c) { x = a; y = b; z = c;}
};
/* Відображення X, Y, Z (оператор виводу для класу
three_d).*/

ostream &operator<<(ostream &stream, three_d obj){
  stream << obj.x << ", ";
  stream << obj.y << ", ";
  stream << obj.z << "\n";
  return stream; // повертає  параметр stream
}
int main(){
  three_d a(1, 2, 3), b(3, 4, 5), c(5, 6, 7);
  cout << a << b << c;
  return 0; }
```

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

"кістяк", що підходить для будь-якої функції виводу даних:

```
ostream &operator<<(ostream &stream, class_type obj)
{
  // код, що відноситься  до конкретного класу
  return stream; // повертає  параметр stream
}
```

Для параметра **obj** дозволяється використовувати передачу по посиланню.

Конкретні дії функції виведення визначаються програмістом.

Функція виведення повинна виводити інформацію і повертати параметр *stream*.

Операторна функція не повинна обмежувати застосування оператора "**<<**" , як в такому її варіанті:

```
ostream &operator<<(ostream &stream, three_d obj)
{
  cout << obj.x << ", ";
  cout << obj.y << ", ";
  cout << obj.z << "\n";
  return stream; // повертає  параметр stream
}
```

У цій версії функції жорстко закодований потік **cout**.

Це обмежує коло ситуацій, у яких її можна використовувати. Оператор "**<<**" повинен працювати з будь-яким потоком, а потік, використаний в "**<<**"-виразі, передаватись параметру **stream**.

Треба передавати функції потік, що коректно працює у всіх випадках.

Тільки так можна створити функцію виводу даних, що підійде для використання в будь-яких виразах введення-виведення.

Використання функцій-"друзів" для перевантаження операторів виведення

Перевантажені оператори виведення не можуть бути функціями-членами, бо лівий операнд (неявно переданий за допомогою покажчика **this**) повинен бути об'єктом класу, що згенерував звертання до цієї операторної функції.

І це змінити не можна.

При перевантаженні операторів виведення **лівий операнд повинен бути потоком**, а правий - об'єктом класу, дані якого підлягають виведенню.

Рішення проблеми – оголосити операторну функцію “другом” класу.

Використання функцій-"друзів" для перевантаження операторів виводу

```
#include <iostream>
using namespace std;
class three_d {
    int x, y, z; // 3-вимірні координати (тепер це private-члени)
  public:
    three_d(int a, int b, int c) { x = a; y = b; z = c; }
    friend ostream &operator<<(ostream &stream, three_d obj);
};
ostream &operator<<(ostream &stream, three_d obj) {
  stream << obj.x << ", ";
  stream << obj.y << ", ";
  stream << obj.z << "\n";
  return stream; // повертає  потік }
int main() {
  three_d a(1, 2, 3), b(3, 4, 5), z(5, 6, 7);
 cout << a << b << z;
  return 0; }
```

Перевантаження операторів уведення

Оператори уведення перевантажуються методом, аналогічним методу перевантаження оператора виводу.

```
/* Прийом тривимірних координат (оператор
уведення для класу three_d).
*/
istream &operator>>(istream &stream, three_d &obj)
{
  cout << "Уведіть координати X, Y і Z: “;
  stream >> obj.x >> obj.y >> obj.z;
  return stream;
}
```

Перевантаження операторів уведення

1. Оператор уведення повинен повертати посилання на об'єкт типу **istream**. Це дозволить кілька операторів вводу об'єднати в одному складеному виразі.
2. Перший параметр повинен бути посиланням на об'єкт типу **istream**. Цей тип належить потоку ліворуч від оператора "**>>**".
3. Другий параметр є посиланням на змінну, котра приймає значення, що уводяться. Оскільки другий параметр - посилання, він може бути модифікований при уведенні інформації.

Загальний формат оператора уведення :

```
istream &operator>>(istream &stream, object_type &obj) {
  // код операторної функції уведення даних
  return stream;
}
```

```
#include <iostream>
using namespace std;
class three_d {
    int x, y, z;
  public:
    three_d(int a, int b, int c) { x = a; y = b; z = c; }
    friend ostream &operator<<(ostream &stream, three_d obj);
    friend istream &operator>>(istream &stream, three_d &obj);
};
// Відображення координат X, Y, Z
ostream &operator<<(ostream &stream, three_d obj){
  stream << obj.x << ", ";
  stream << obj.y << ", ";
  stream << obj.z << "\n";
  return stream; // повертає параметр stream
}
// Прийом тривимірних координат
istream &operator>>(istream &stream,
three_d &obj){
  cout << "Уведіть координати X, Y і Z: ";
  stream >> obj.x >> obj.y >> obj.z;
  return stream;
}
```

```
1, 2, 3
Уведіть координати X, Y і Z: 5 6 7
5, 6, 7
```

```
int main() {
  three_d a(1,2,3);
  cout << a;
  cin >> a;
  cout << a;
  return 0;
}
```

Порівняння С- і С++-систем вводу-виводу

С-система вводу-виводу не забезпечує ніякої підтримки об'єктів, визначених користувачем.

Наприклад, якщо створити в С таку структуру

```
struct my_struct {
  int count;
  char s [80];
  double balance;
} cust;
```

то існуючу в С систему вводу-виводу неможливо настроїти так, щоб вона могла виконувати операції вводу-виводу безпосередньо над об'єктами типу **my_struct**.

Порівняння С- і С++-систем вводу-виводу

Для С++ була створена нова об’єктно-орієнтована система вводу-виводу.

Оскільки С++ є супермножиною мови С, то і вся С-система вводу-виводу включена в С++.

При перекладі С-програм на мову С++ не потрібно змінювати всі інструкції вводу-виводу підряд.

Працюючі С-інструкції скомпілюються й будуть успішно працювати у С++-середовищі.

Форматований ввод-вивод даних

Програміст може управляти форматом подання даних двома способами:

1. використанням функцій-членів класу **ios**;
2. використанням функцій спеціального типу, іменованих маніпуляторами (manipulator).

Форматування даних з використанням функцій-членів класу `ios`

У класі **ios** оголошується перерахування **fmtflags**. В класі **ios_base**, що є базовим для класу **ios**, визначені такі значення цього перерахування

adjustfield	**floatfield**	**right**	**skipws**
basefield	**hex**	**scientific**	**unitbuf**
boolalpha	**internal**	**showbase**	**uppercase**
dec	**left**	**showpoint**	
fixed	**oct**	**showpos**	

Ці значення використовуються для установки або очищення прапорів форматування за допомогою таких функцій, як **setf()** і **unsetf()**.

Старі компілятори можуть не «знати» про тип перерахування **fmtflags** - прапори форматування будуть кодуватися як цілочисельні long-значення.

Установка прапорів

Якщо прапор **skipws** установлений, то при потоковому уведенні даних провідні "пробільні" символи, або символи пропуску (тобто пробіли, символи табуляції й нового рядка), відкидаються.

Якщо ж прапор **skipws** скинутий, пробільні символи не відкидаються.

Якщо встановлено прапор **left**, виведені дані вирівнюються по лівому краю, а якщо встановлено прапор **right** - по правому.

Якщо встановлено прапор **internal**, числове значення доповнюється пробілами, якими заповнюється поле між ним і знаком числа або символом основи системи числення.

Установка прапорів

Якщо жоден із цих прапорів не встановлений, результат вирівнюється по правому краю за замовчуванням.

Установка прапора **oct** приведе до виводу результату у вісімковому поданні, а установка прапора **hex** - у шіснадцятковому. Щоб при відображенні результату повернутися до десяткової системи числення, досить установити прапор **dec**.

Установка прапора **showbase** приводить до відображення позначення основи системи числення, у якій представляються числові значення. Наприклад, значення **1F** буде відображено як **0x1F**.

За замовчуванням при використанні експонентного подання чисел відображається рядковий варіант букви "e".

При відображенні шіснадцяткового значення використовується мала літера "x". Після установки прапора **uppercase** відображається прописний варіант цих символів.

Установка прапорів

Установка прапора **showpos** викликає відображення провідного знака "плюс" перед додатніми значеннями.

Установка прапора **showpoint** - відображення десяткової крапки й хвостових нулів для всіх чисел із плаваючою крапкою - потрібні вони чи ні.

Установки прапора **scientific -** числові значення із плаваючою крапкою відображаються в експонентному поданні.

Якщо встановлено прапор **fixed**, дійсні значення відображаються у звичайному поданні.

Якщо не встановлений жоден із цих прапорів, компілятор сам вибирає відповідний метод подання.

Установка прапорів

При встановленому прапорі **unitbuf** уміст буфера скидається на диск після кожної операції виводу даних.

Якщо встановлено прапор **boolalpha**, значення булевого (логічного) типу можна вводити або виводити, використовуючи ключові слова **true** і **false**.

Часто доводиться звертатися до полів **oct**, **dec** і **hex**, на них допускається колективне посилання **ios::basefield**.

Поля **left**, **right** і **internal** можна збірно назвати **ios::adjustfield**.

Поля **scientific** і **fixed** можна назвати **ios::floatfield**.

Установка прапорів

Для установки будь-якого прапора використовується функція **setf()**, що є членом класу **ios**. Її формат:

fmtflags setf(fmtflags flags);

функція повертає значення попередніх установок прапорів форматування й установлює їх у відповідності зі значенням, заданим параметром **flags**.

Наприклад, щоб установити прапор **showbase**, можна використовувати інструкцію:

stream.setf(ios::showbase);

Елемент **stream** означає потік, параметри форматування якого ви хочете змінити.

Установка прапорів

У програмі функція **setf()** використовується для установки прапорів **showpos** і **scientific**.

```
#include <iostream>
using namespace std;
int main() {
  cout.setf(ios::showpos);
  cout.setf(ios::scientific);
  cout << 123 << " " << 123.23 << " ";
  return 0;
}
```

Результати виконання цієї програми:

+123 +1.232300e+002

Установка прапорів

За допомогою операції АБО можна встановити відразу кілька потрібних прапорів форматування в одному виклику функції **setf()**.

Наприклад, об'єднавши по АБО прапори **scientific** і **showpos**:

cout.setf(ios::scientific | ios::showpos);

Щоб скинути прапор, використовуйте функцію **unsetf()**, прототип якої виглядає так.

void unsetf(fmtflags flags);

У цьому випадку будуть обнулені прапори, задані параметром **flags**.

Усі інші прапори залишаються в колишньому стані.

Установка прапорів

Щоб дізнатися про поточні установки прапорів форматування, скористайтеся функцією **flags()**, прототип якої має такий вигляд.

fmtflags flags();

Ця функція повертає поточне значення прапорів форматування для викликаючого потоку.

При цьому установлюються значення прапорів форматування відповідно до вмісту параметра **flags** і повертаються їхні попередні значення.

fmtflags flags(fmtflags flags);

```
#include <iostream>
using namespace std;
void showflags(ios::fmtflags f);
int main() {
  ios::fmtflags f;
  f = cout.flags();
  showflags(f);
  cout.setf(ios::showpos);
  cout.setf(ios::scientific);
  f = cout.flags();
  showflags(f);
  cout.unsetf(ios::scientific);
  f = cout.flags();
  showflags(f);
  return 0;
}
void showflags(ios::fmtflags f) {
  long i;
  for(i=0x4000; i; i=i>>1)
    if(i & f) cout << "1";
    else cout << "0";
  cout << "\n";
}
```

```
0 0 0 0 0 1 0 0 0 0 0 0 0 0 1
0 0 1 0 0 1 0 0 0 1 0 0 0 0 1
0 0 0 0 0 1 0 0 0 1 0 0 0 0 1
```

Установка ширини поля, точності й символів заповнення

Можна також установлювати ширину поля, символ заповнення й кількість цифр після десяткової крапки (точність):

streamsize width(streamsize len);

char fill(char ch);

streamsize precision(streamsize num);

Функція **width()** повертає поточну ширину поля й установлює нову рівною значенню параметра **len**.

Ширина поля, що встановлюється за замовчуванням, визначається кількістю символів, необхідних для зберігання даних у кожному конкретному випадку.

Установка ширини поля, точності й символів заповнення

Функція **fill()** повертає поточний символ заповнення (за замовчуванням використовується пробіл) і встановлює в якості нового поточного символу заповнення значення, задане параметром **ch**.

Функція **precision()** повертає поточну кількість цифр, відображуваних після десяткової крапки, і встановлює нове поточне значення точності рівним значенню параметра **num**.

Тип **streamsize** визначений як цілочисельний тип.

```
#include <iostream>
using namespace std;
int main()
{
  cout.setf(ios::showpos);
  cout.setf(ios::scientific);
  cout << 123 << " " << 123.23 << "\n";
  cout.precision(2); // Дві цифри після десяткової крапки.
  cout.width(10); // Усе поле складається з 10 символів.
  cout << 123 << " ";
  cout.width(10); // Установка ширини поля рівної 10.
  cout << 123.23 << "\n";
  cout.fill('#'); // Для заповнювача візьмемо символ "#"
  cout.width(10); // і встановимо ширину поля рівної 10.
  cout << 123 << " ";
  cout.width(10); // Установка ширини поля рівної 10.
  cout << 123.23;
  return 0;
}
```

```
+123 +1.232300e+002
          +123 +1.23e+002
######+123 +1.23e+002
```

У системі вводу-виводу C++ визначені й перевантажені версії функцій, що використовуються тільки для їхнього одержання:

```
char fill();
streamsize width();
streamsize precision();
```

Використання маніпуляторів вводу-виводу

Маніпулятори дозволяють вбудовувати інструкції форматування у вираз вводу-виводу.

Маніпулятор	Призначення	Функція
boolalpha	Встановлює прапор boolalpha	Ввод-вивод
dec	Встановлює прапор dec	Ввод-вивод
endl	Виводить символ нового рядка и 'скидає' потік, тобто - переписує вміст буфера, зв'язанного з потоком, на певний пристрій	Вивод
ends	Вставляє в потік нульовий символ ('\0')	Вивод
fixed	Встановлює прапор fixed	Вивод
flush	Скидає потік	Вивод

Використання маніпуляторів вводу-виводу

Маніпулятор	Призначення	Функція
hex	Встановлює прапор hex	Ввод-вивод
internal	Встановлює прапор internal	Вивод
left	Встановлює прапор left	Вивод
noboolalpha	Обнуляє прапор boolalpha	Ввод-вивод
noshowbase	Обнуляє прапор showbase	Вивод
noshowpoint	Обнуляє прапор showpoint	Вивод
noshowpos	Обнуляє прапор showpos	Вивод
noskipws	Обнуляє прапор skipws	Ввод

Використання маніпуляторів вводу-виводу

Маніпулятор	Призначення	Функція
nounitbuf	Обнуляє прапор unitbuf	Вивод
nouppercase	Обнуляє прапор uppercase	Вивод
oct	Встановлює прапор oct	Ввод-вивод
Resetiosflags(fmtflags *f*)	Обнуляє прапори, задані в параметрі *f*	Ввод-вивод
right	Встановлює прапор right	Вивод
scientific	Встановлює прапор scientific	Вивод
Setbase(int *base*)	Встановлює основу системи числення рівною base	Вивод
Setfill(int *ch*)	Встановлює символ-заповнювач рівним значенню ch	Вивод

Використання маніпуляторів вводу-виводу

Маніпулятор	Призначення	Функція
Setiosflags(fmtflags *f*)	Встановлює прапори, задані в параметрі *f*	Ввод-вивод
Setprecision(int *p)*	Встановлює кількість цифр точності (після крапки)	Вивод
setw(int w)	Встановлює ширину поля рівною значенню параметра *w*	Вивод
showbase	Встановлює прапор showbase	Вивод
showpoint	Встановлює прапор showpoint	Вивод
showpos	Встановлює прапор showpos	Вивод
skipws	Встановлює прапор skipws	Ввод
unitbuf	Встановлює прапор unitbuf	Вивод
uppercase	Встановлює прапор uppercase	Вивод
ws	Пропускає ведучі "пробільні* символи	Ввод

Використання маніпуляторів вводу-виводу

При використанні маніпуляторів, які приймають аргументи, необхідно включити в програму заголовок **<iomanip>.**

```
#include <iostream>
#include <iomanip>
using namespace std;
int main()
{
  cout << setprecision(2) << 1000.243 << endl;
  cout << setw(20) << "Усім привіт! ";
  return 0;
}
```

Результати виконання цієї програми такі.

1e+003
Усім привіт!

1. Маніпулятори можна використовувати в ланцюжку операцій вводу-виводу.
2. Якщо маніпулятор викликається без аргументів (як, наприклад, маніпулятор endl у цій програмі), то його ім'я вказується без пари круглих дужок.

Приклад використання маніпулятора setiosflags() для установки прапорів scientific і showpos.

```
#include <iostream>
#include <iomanip>
using namespace std;
int main() {
  cout << setiosflags(ios::showpos);
  cout << setiosflags(ios::scientific);
  cout << 123 << " " << 123.23;
  return 0;
}
```

```
+123 +1.232300e+002
```

Приклад використання маніпулятора ws, що пропускає провідні "пробільні" символи при уведенні рядка в масив s:

```
#include <iostream>
using namespace std;
int main()
{
  char s[80];
  cin >> ws >> s;
  cout << s;
  return 0;
}
```

Створення власних маніпуляторних функцій

Можна створювати маніпуляторні функції з параметрами і без.

Усі маніпуляторні функції **виведення даних** без параметрів мають таку структуру.

```
ostream &manip_name(ostream &stream) {
  // код маніпуляторної функції
  return stream;
}
```

Тут елемент **manip_name** означає ім'я маніпулятора. Маніпулятор має один посилальний параметр – потік, який неявно стає синонімом потоку, яким маніпулює.

Створення власних маніпуляторних функцій

```
#include <iostream>
#include <iomanip>
using namespace std;
ostream &setup(ostream &stream) {
  stream.setf(ios::left);
  stream << setw(10) << setfill ('$');
  return stream;
}
int main() {
  cout << 10 << " " << setup << 10;
  return 0;
}
```

Створення власних маніпуляторних функцій

Усі маніпуляторні функції **уведення даних** без параметрів мають наступну структуру:

```
istream &manip_name(istream &stream) {
  // код маніпуляторной функції
  return stream;
}
```

Створення власних маніпуляторних функцій

```
#include <iostream>
#include <iomanip>
using namespace std;
istream &prompt(istream &stream) {
  cin >> hex;
  cout << "Уведіть число в шіснадцятковому
форматі: ";
  return stream;
}
int main() {
  int i;
  cin >> prompt >> i;
  cout << i;
  return 0;
}
```

Файлове введення-виведення

Файлові операції введення-виведення підтримуються заголовком **<fstream>**.

Як відкрити й закрити файл

У C++ файл відкривається шляхом зв'язування його з потоком, оголошеним як об’єкт відповідного потокового класу:

ifstream in; // вхідний потік

ofstream out; // вихідний потік

fstream both; // потік вводу-виводу

Файлове введення-виведення

Файл може зв’язуватись з потоком за допомогою функції-члена **open()** відповідного потокового класу:

void ifstream::open(const char *fn, ios::openmode mode = ios::in);

void ofstream::open(const char *fn, ios::openmode mode = ios::out | ios::trunc);

void fstream::open(const char *fn, ios::openmode mode = ios::in | ios::out);

fn – це ім'я файлу, що може включати специфікатор шляху.

Файлове введення-виведення

Елемент **mode** визначає спосіб відкриття файлу. Він повинен приймати одне або кілька (поєднаних логічним АБО) значень перерахування **openmode**, визначених в класі **ios**.

ios::app // дописування у кінець файла

ios::ate // пошук з кінця файла

ios::binary // двійковий режим (за замовчуванням режим текстовий)

ios::in // уведення даних

ios::out // вивод даних

ios::trunc // очистка файла

Файлове введення-виведення

Параметр **mode** функції **open()** за замовчуванням встановлюється рівним значенню, що відповідає типу її потоку. Фрагмента коду:

```
ofstream out;
out.open(“test");
```

відкривається звичайний вихідний файл.

Не відкритий у результаті невдалого виконання функції **open()** потік при використанні в булевому виразі містить значення, що інтерпретується як **FALSE**.

```
if(!mystream) {
  cout << "Не вдається відкрити файл.\n";
  // обробка помилки
}
```

Перш ніж робити спробу одержання доступу до файлу, варто завжди перевіряти результат виклику функції **open().**

Файлове введення-виведення

Можна також перевірити факт успішного відкриття файлу за допомогою функції **is_open()**, що є членом класів **fstream**, **ifstream** і **ofstream**. Її прототип:

bool is_open();

Функція повертає значення **TRUE**, якщо потік пов'язаний з відкритим файлом, і **FALSE** - інакше.

```
if(!mystream.is_open()) {
  cout << "Файл не відкритий.\n";
  // ...
}
```

Файлове введення-виведення

Оскільки класи **ifstream**, **ofstream** і **fstream** мають конструктори, які автоматично відкривають заданий файл, їх використовують замість функції **open().**
Параметри в цих конструкторів і їхні значення (що діють за замовчуванням) збігаються з параметрами й відповідними значеннями функції **open().**
Найчастіше файл відкривається так:

```
ifstream mystream("myfile");
// файл відкр. для уведення
```

Якщо з якоїсь причини файл відкрити неможливо, потокова змінна, що зв'язує із цим файлом, установлюється рівною значенню **FALSE**.
Закриває файл зв’язаний з потоком функція-член цього потоку **close().**

```
mystream.close();
```

Читання й запис текстових файлів

```
// Запис даних у файл.
#include <iostream>
#include <fstream>
using namespace std;
int main()
{
  ofstream out("test");
  if(!out) {
    cout << "Не вдається відкрити файл.\n";
    return 1;
  }
  out << 10 << " " << 123.23 << "\n";
  out << "Це короткий текстовий файл.";
  out.close();
  return 0;
}
```

```
// Зчитування даних з файлу, створеного попередньою програмою.
#include <iostream>
#include <fstream>
using namespace std;
int main() {
  char ch; int i; float f; char str[80];
  ifstream in("test");
  if(!in) {
    cout << "Не вдається відкрити файл.\n";
    return 1;
  }
  in >> i;
  in >> f;
  in >> ch;
  in >> str;
  cout << i << " " << f << " " << ch << "\n";
  cout << str;
  in.close();
  return 0;
}
```

Неформатований ввод-вивод даних у двійковому режимі

C++ підтримує ряд функцій файлового вводу-виводу у двійковому режимі (із використанням специфікатора режиму **`ios::binary`**), які можуть виконувати операції без форматування даних.

Функції обробки неформатованих файлів можуть працювати з файлами, відкритими в текстовому режимі доступу, але при цьому можуть мати місце перетворення символів, які зводять нанівець основну мету виконання двійкових файлових операцій.

Існує два способи запису неформатованих двійкових даних у файл і зчитування їх з файлу. Перший складається у використанні функції-члена **`put()`** (для запису байта у файл) і функції-члена **`get()`** (для зчитування байта з файлу). Другий спосіб припускає застосування "блокових" C++-функцій вводу-виводу **`read()`** і **`write()`**.

Використання функцій get() *і* put()

```
istream &get(char &ch);  /* Зчитує один символ з відповідного потоку в змінну ch
      і повертає  посилання на потік, пов'язаний з попередньо відкритим файлом.
      При досягненні кінця цього файлу значення посилання стане рівним нулю */
ostream &put(char ch); /* Записує символ ch у потік  і повертає  посилання на цей
                                потік */

#include <iostream>
#include <fstream>
using namespace std;
int main(int argc, char *argv[]) {
  char ch;
  if(argc!=2) {
    cout << "Застосування: ім'я_програми <ім'я_файлу>\n“; return 1;
  }
  ifstream in(argv[1], ios::in | ios::binary);
  if(!in) {
    cout << "Не вдається відкрити файл.\n“;  return 1;
  }
  while(in) { // При досягненні кінця файлу in прийме значення false
```

```
/* Використання функції put() для запису рядка у
файл.
*/
#include <iostream>
#include <fstream>
using namespace std;
int main()
{
  char *p = "Усім привіт!";
  ofstream out("test", ios::out | ios::binary);
  if(!out) {
    cout << "Не вдається відкрити файл.\n";
    return 1;
  }
  while(*p) out.put(*p++);
  out.close();
  return 0;
}
```

Зчитування й запис у файл блоків даних

Щоб зчитувати й записувати у файл блоки двійкових даних, використовують функції-члени **read()** і **write()**.

```
istream &read(char *buf, streamsize num);
ostream &write(const char *buf, int streamsize num);
```

Функція **read()** зчитує **num** байт даних (блок даних) з пов'язаного з файлом потоку й поміщає їх у буфер, що адресується параметром **buf**.

Функція **write()** записує **num** байт даних (блок даних) у пов'язаний з файлом потік з буфера, що адресується параметром **buf**.

Тип **streamsize** визначений як деякий різновид цілочисельного типу. Він дозволяє зберігати найбільшу кількість байтів, що може бути передане в процесі будь-якої операції вводу-виводу.

// Спочатку у файл записується масив цілих чисел, а потім він же зчитується з файлу

```
#include <iostream>
#include <fstream>
using namespace std;
int main() {
  int n[5] = {1, 2, 3, 4, 5};  register int i;
  ofstream out("test", ios::out | ios::binary);
  if(!out) {
    cout << "Не вдається відкрити файл.\n";
    return 1;
  }
  out.write((char *) &n, sizeof n);  out.close();
  for(i=0; i<5; i++) n[i] = 0;   // очищаємо масив
  ifstream in ("test", ios::in | ios::binary);
  if(!in) {
    cout << "Не вдається відкрити файл.\n";
    return 1;
  }
  in.read((char *) &n, sizeof n);
  for(i=0; i<5; i++) // Відображаємо значення, зчитані з файлу
    cout << n[i] << " ";
  in.close();
  return 0;
}
```

В інструкціях звертання до функцій **read()** і **write()** виконуються операції приведення типу, які обов'язкові при використанні буфера, визначеного не у вигляді символьного масиву.

Функція **gcount()** *повертає кількість символів, зчитаних при виконанні останньої операції уведення даних.*
Якщо кінець файлу буде досягнутий до того, як буде зчитано **num** символів, функція **read()** просто припинить виконання, а буфер буде містити стільки символів, скільки вдалося зчитати до цього моменту. Точну кількість зчитаних символів можна дізнатися про за допомогою функції-члена **gcount()**, що має такий прототип:

```
streamsize gcount();
```

Виявити кінець файлу можна за допомогою функції-члена **eof()**, що має такий прототип:

```
bool eof();
```

Ця функція повертає значення **true** при досягненні кінця файлу; у противному випадку вона повертає значення **false**.

```
// Виявлення кінця файлу за допомогою функції eof()
#include <iostream>
#include <fstream>
using namespace std;
int main(int argc, char *argv[])
{
  char ch;
  if(argc!=2) {
    cout << "Застосування: ім'я_програми <ім'я_файлу>\n";
    return 1;
  }
  ifstream in(argv[1], ios::in | ios::binary);
  if(!in) {
    cout << "Не вдається відкрити файл.\n";
    return 1;
  }
  while(!in.eof()) {
    // використання функції eof()
    in.get(ch);
    if( !in.eof()) cout << ch;
  }
  in.close();
  return 0;
}
```