**Лекція 25. Класи потоків введення Перевантаження операторів потокового введення/виведення. Форматування виводу.**

З потоковим введенням-виведенням файлів ми познайомилися в загальних рисах ще на лабораторні роботі №6. Тепер ми поглибимо і систематизуємо ці знання.

**Поняття файлу**

Під час розв’язання задач на комп’ютері часто виникає необхідність у використанні даних, які записані на зовнішніх носіях інформації (дисках) і оформлені у вигляді файлів даних. Незалежно від того, які дані містять файли (числа, символи, рядки, масиви, структури тощо), в мові С++ вони трактуються як потоки даних **(stream),** які являють собою послідовність байтів, що зчитуються або записуються.

За замовчуванням у кожній програмі С++ можна користуватися такими стандартними потоками: стандартного введення **(сіn),** стандартного виведення **(cout)** та виведення помилок **(сеrr).** Щоб користуватися файлами, потоки повинні бути створені і закріплені за цими файлами.

**Використання файлів даних у програмі передбачає виконання таких операцій**:

* створення потоку обміну даними між файлом і пам’яттю комп’ютера;
* зв’язування цього потоку з конкретним ім’ям файлу на диску і відкриття файлу;
* запис даних у файл або читання їх з файлу;
* закриття файлу.

Для реалізації цих операцій існують спеціальні класи, які містять конструктори створення необхідних потоків:

* **ifstream** — для створення потоку читання даних;
* **ofstream** — для створення потоку запису даних у файл;
* **fstream** — використовується як для запису даних у файл, так і їх читання.

**Конструктори для роботи з файлами**

Конструктори записуються як з параметрами, так і без параметрів. Конструктори з параметрами одночасно створюють відповідний потік, зв’язують його з файлом на диску, відкривають файл для роботи і мають такі форми запису:

**ofstream іп (“іф”, ios::out); або ofstream (“іф”);**

**ifstream іп (“іф”, ios::in); або ifstream (” “);**

**fstream іп (“іф”, ios::in | ios::out);**

де **іп** — ім’я потоку, який створюється для роботи з файлом;

**іф** — константа або змінна типу **char[ ],** її значення — ім’я файлу на диску.

Перший з конструкторів використовується для запису даних у файл, другий — для читання даних з файлу, а третій — як для запису, так і для читання даних, наприклад:

**ofstream fout( “myfile.dat”, ios::out);**

Цей запис створює потік з ім’ям **fout**, зв’язує його з файлом на диску, який має ім’я **myfile.dat** і відкриває цей файл для запису даних. Файл **myfile.dat** буде створено у тому ж каталозі, що і програма. Якщо треба створити файл у другому місці, то для запису його імені треба вказати шлях, наприклад, **a:\\pvp\\myfile.dat**. Тепер цей файл буде записано на диску **а:** в каталозі **pvp**.

Зверніть увагу на те, що для запису шляху треба використовувати подвійні зворотні косі риски.

Можна також для роботи з файлами застосувати конструктори без параметрів:

**ofstream іп;**

**ifstream іп;**

**fstream іп;**

де **іп** — ім’я відповідного потоку, тоді для зв’язування потоку з ім’ям файлу на диску і відкриття його для роботи треба додатково використовувати функцію-член відповідного класу, тобто:

**іn.ореn(“іф”, ios :: ознака | ios :: ознака);**

(в цьому запису  **|** означає "або", тобто можливість завдання декількох ознак відкриття файлу).

Наприклад, відкриття файлу для запису до нього даних матиме вигляд:

**ofstream fout;**

**fout.open ( “a:\\pvp\\my file.dat”, ios::out);**

Конструктори як з параметрами, так і без них, виконують однакову роботу, тому яким з них надати перевагу — справа користувача.

***Приклад 1.***  Створити файл на диску і записати до нього масив чисел. Прочитати цей файл і вивести його компоненти на екран.

// Створення файлу та запис до нього масиву

**#include <iostream>**

**#include <fstream>**

**#include <conio.h>**

**#include <Windows.h>**

**using namespace std;**

**int main()**

**{ int i, mas[5];**

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

**// запис елементів масиву до файлу:**

**ofstream fout("massiv.txt"); /\* створення потоку fout та відкриття файлу з іменем massiv. txt для запису \*/**

**if (! fout) cout <<"Cannot open file\n";**

**for (i = 0; i <5; i++)**

**{**

**cout << " Enter "<< i << " element\n";**

**cin >> mas[i]; // введення елементу масиву з клавіатури**

**fout << mas[i] << " "; //запис елементу до файлу**

**}**

**fout.close();**

**//---------- читання компонентів файлу та виведення на екран**

**ifstream fin("massiv.txt"); /\* створення потоку fin для читання файлу\*/**

**if (!fin) cout << " Cannot open file fo reading\n";**

**cout <<"REZULTAT \n";**

**for (i = 0; i <5; i++)**

**{**

**fin >> mas[i];**

**// читання поточного елементу масиву з файлу**

**cout << "mas[" << i << "]=" << mas[i] << " ";**

**}**

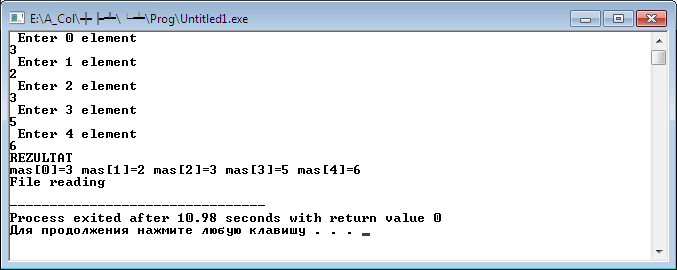
**cout << "\nFile reading\n";**

**fin.close();**

**return 0;**

**}**

Результат розв’язання прикладу має вигляд:



**Ознаки відкриття файлу**

Якщо не використовувати ознаки відкриття файлу для читання чи запису, на диску буде створений новий файл із зазначеним ім'ям, або перезаписаний файл із таким же ім'ям, якщо він вже існує.

Наприклад, для того, щоб записати потрібну інформацію в кінець файлу text1.dat, необхідно скористатися фрагментом програмного коду (тут файлова змінна flags):

ofstream flags(“text1.dat”, ios::app);

Таблиця 1 – Ознаки відкриття файлу

| **Ознака** | **Призначення** |
| --- | --- |
| ios::in | Відкриває файл для читання. Вміст файлу зберігається |
| ios::out | Відкриває файл для запису. Якщо файл не існує, то буде створений |
| ios::app | Відкриває файл для дозапису. Дані будуть записані у кінець файлу |
| ios::trunk | Якщо  файл, який відкривають для запису, вже існує, то його вміст буде видалено. |
| ios::nocreate | Забороняє створювати файл, який відкривають |
| ios::inoreplace | Забороняє перезаписувати існуючий  файл |

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 бінарний – це лінійна послідовність байтів, що відповідає внутрішньому поданню даних без поділу на рядки. Для завдання бінарного файлу використовується ознака ios:binary (двійковий). Якщо задається декілька ознак (як в прикладі), то вони розділяються вертикальною рискою ( | ).

***Приклад 2.*** Записати у файл 5 прізвищ, потім прочитати їх і вивести на екран.

**/\* *програма запису до файлу та читання з файлу масиву з 5 прізвищ* \*/**

**#include <iostream>**

**#include <fstream>**

**#include <conio.h>**

**#include <Windows.h>**

**using namespace std;**

**int main()**

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

**char st[5][15];**

**int i;//------------------ запис до файлу**

**ofstream fout("st\_file.dat"); // відкриття файлу**

**if (!fout) cout << "Cannot open file\n";**

**for (i = 0; i <5; i++)**

**{**

**cout << " Enter " << (i+1) << " name\n";**

**cin.getline(st[i],15); // введення поточного прізвища**

**fout << st[i] << '\n'; // запис прізвища до файлу**

**}**

**fout.close();//------------читання файлу та виведення на екран**

**cout << "\nReading file\n\n";**

**ifstream fin("st\_file.dat");**

**if (!fin)cout << "Cannot open file.dat\n";**

**for (i = 0; i < 5; i++)**

**{**

**fin.getline(st[i],15);**

**cout << st[i] << " ";**

**}**

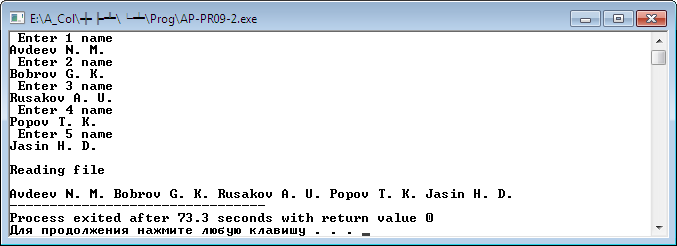
**fin.close();**

**getch();**

**return 0;**

**}**

Результат роботи програми має вигляд:

**.**

У попередніх програмах запис даних у файл та їх читання з файлу здійснювалось послідовно поелементно. Але записати або прочитати декілька даних (наприклад, масив чисел) можна однією операцією. Для цього використовують функції-члени відповідних класів, які мають вигляд:

**in.write((char\*)&p,sizeof(p));** — для запису даних у файл,

**in.read((char\*)&p,sizeof(p));** — для читання даних з файлу,

де **іn** — ім’я потоку введення або виведення;

**р** — змінна будь-якого типу, якщо змінна **р** має тип **char[ ],** то операція її приведення не потрібна.

**Функції обробки файлів та режими роботи з файлами**

У мові С++ використовуються функції роботи з файлами, запозичені з мови програмування C. Загальновживані функції роботи з файлами наведено в табл. 2.

Таблиця 2 – Основні функції роботи з файлами

| **Функція** | **Призначення** |
| --- | --- |
| fopen() | Відкриває потік |
| fclose() | Закриває потік |
| fputc() | Записує символ у потік |
| fgetc() | Зчитує символ з потоку |
| fwrite() | Записує блок даних у потік |
| fread() | Зчитує блок даних з потоку |
| fseek() | Встановлює індикатор позиції файлу на заданий байт у потоці |
| fprintf() | Робить для потоку те, що функція printf() робить для консолі |
| fscanf() | Робить для потоку те, що функція scanf() робить для консолі () |
| feof() | Повертає значення true, якщо досягнуто кінець файлу |
| ferror() | Повертає значення true, якщо виникла помилка |
| rewind() | Встановлює індикатор позиції файлу у початок файлу |
| remove() | Видаляє файл |

**Функції, призначені для відкриття / закриття файлів**

Функція **fopen()** виконує наступні дії:

* відкриває потік;
* пов'язує файл з потоком;
* повертає покажчик типу FILE на цей потік.

Прототип функції fopen():

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

У цьому записі параметр filename вказує на ім'я файлу, що відкривається, параметр mode – на рядок, що містить потрібний режим відкриття файлу. Можливі значення режима відкриття файлу наведені у табл. 3. Параметр filename повинен представляти рядок символів, який визначає ім'я файлу, допустиме у даній операційній системі. Цей рядок може містити специфікацію шляху до файлу, якщо діюче програмне середовище підтримує таку можливість.

Таблиця 3– Режими роботи з файлами

| **mode** | **Призначення** |
| --- | --- |
| "r" | Відкриває текстовий файл для зчитування |
| "w" | Створює текстовий файл для запису |
| "а" | Відкриває текстовий файл для запису у кінець файлу |
| "rb" | Відкриває двійковий файл для зчитування |
| "wb" | Створює двійковий файл для запису |
| "ab" | Відкриває двійковий файл для запису у кінець файлу |
| "r+" | Відкриває текстовий файл для зчитування і запису |
| "w+" | Створює текстовий файл для зчитування і запису |
| " a+ " | Відкриває текстовий файл для зчитування і запису у кінець файлу |
| "r+b" | Відкриває двійковий файл для зчитування і запису |
| "w+b" | Створює двійковий файл для зчитування і запису |
| "a + b" | Відкриває двійковий файл для зчитування і запису у кінець файлу |

У випадку успішного відкриття вказаного файлу функція fopen() повертає покажчик FILE. Цей покажчик ідентифікує файл. Якщо файл не вдається  відкрити, наприклад, якщо файл із заданим іменем відсутній на диску, повертається нульовий покажчик.

**Приклад 3.** Відкриття текстового файлу test для запису**:**

fp = fopen("test", "w");

Функція fopen() має режими роботи як з текстовими, так і з двійковими (бінарними) файлами.

Закриття файлу здійснюється функцією **fclose()**, що має наступний прототип:

**Int fclose (FILE \* Stream);**

У якості параметра функції виступає покажчик на файл, що закривається. Функція повертає 0 якщо файл був успішно закрито, і  значення 1 – в іншому випадку.

Приклад виклику функції закриття файлу:

fclose(p1);

Під час виклику функції  fclose()  звільняється  блок  керування файлом, що  пов'язаний  з  потоком,  що  робить  його  доступним  для  повторного використання.

**Введення/виведення даних**

Функцію **fputc()** використовують для виведення символів у потік, який заздалегідь відкрито для запису за допомогою функції fopen().

Прототип функції fputc():

**int fputc(int ch, FILE \*fp);**

У цьому  записі параметр  fp представляє собою файловий покажчик, що повертається функцією fopen(), а параметр  ch –  символ, що виводиться. Файловий покажчик повідомляє функції fputc(), в який дисковий файл необхідно записати символ. При успішному виконанні операції виведення функція  fputc()  повертає записаний у файл символ, інакше – значення EOF.

Функція  **fgetc()**  використовують  для  зчитування  символів  з потоку, відкритого в режимі  зчитування  за допомогою функції  fopen().

Прототип функції fgetc():

**int fgetc(FILE \*fp);**

У цьому  записі параметр  fp означає файловий покажчик, що повертається fopen(). Під  час  виникненні помилки  або  досягнення кінця файлу функція fgetc() повертає значення EOF. Отже, для того, щоб зчитати весь вміст текстового файлу можна використати фрагмент програмного коду:

**ch = fgetc(fp);**

**while(ch != EOF)**

**{**

**ch= fgetc(fp);**

**}**

Функція **fprintf()** форматує і друкує у вихідний потік набори символів і значень. Кожен аргумент (якщо він є) перетворюється і виводиться згідно заданої специфікації формату.

Прототип функції fprintf():

**int fprintf ( FILE \* stream, const char \* format, ... );**

Функція **fscanf()** зчитує дані з поточної позиції потоку в місце, обумовлене значенням аргументів (якщо вони є). Кожен аргумент повинен бути покажчиком на змінну і тип, який відповідають типу, заданому в рядку формату.

Прототип функції fscanf():

**int fprintf (stream, format-string [, argument ...]);**

Таким чином призначення функцій fprintf() та fscanf() аналогічно функціям printf()  і scanf(), за винятком  того, що вони працюють з файлами.

**Приклад 4.** Дано два текстових файли, що містять цілі числа, впорядковані за зростанням. Необхідно об'єднати ці файли в третій, що будуть містити впорядковані за зростанням числа з перших двох файлів.

**#include <iostream>**

**#include <fstream>**

**#include <Windows.h>**

**using namespace std;**

**int main()**

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

**FILE \*f1,\*f2,\*r;**

**int a1,a2;**

**f1= fopen("file1.txt","r");**

**if(f1==NULL) return 1;**

**f2 = fopen("file2.txt","r");**

**if(f2==NULL)**

**{**

**fclose(f1);**

**return 1;**

**}**

**r = fopen("result.txt","w");**

**if(r==NULL){fclose(f1);fclose(f2);return 1;}**

**fscanf(f1,"%d",&a1);**

**fscanf(f2,"%d",&a2);**

**cout << a1 << '\t'<< a2 << endl;**

**while(1)**

**{**

**if(a1<a2)**

**{**

**fprintf(r,"%d ",a1);**

**if(fscanf(f1,"%d",&a1)==EOF) break;**

**}**

**else**

**{**

**fprintf(r,"%d ",a2);**

**if(fscanf(f2,"%d",&a2)==EOF) break;**

**}**

**}**

**if(feof(f1))**

**{**

**while(!feof(f2))**

**{**

**fprintf(r,"%d ",a2);**

**fscanf(f2,"%d",&a2);**

**}**

**fprintf(r,"%d ",a2);**

**}**

**else**

**if(feof(f2))**

**{**

**while(!feof(f1))**

**{**

**fprintf(r,"%d ",a1);**

**fscanf(f1,"%d",&a1);**

**}**

**fprintf(r,"%d ",a1);**

**}**

**fclose(f1);**

**fclose(f2);**

**fclose(r);**

**return 0;**

**}**

Функцію **ferror()** використовують  для  визначення факту виникнення помилки у процесі виконання операції з файлом.

Її прототип має такий вигляд:

**int ferror(FILE \*fp);**

У цьому записі параметр  fp дійсний файловий покажчик. Функція ferror() повертає значення  true, якщо у процесі виконання останньої файлової операції відбулася  помилка;  інакше  значення  false.  Оскільки  виникнення  помилки можливе  у  процесі  виконання  будь-якої  операції  з файлом, функцію  ferror() необхідно викликати відразу після кожної функції оброблення файлів; інакше інформацію про помилку можна просто втратити.

Функція  **rewind()**  переміщує  індикатор  позиції  файлу у початок файлу, що задається як аргумент.

Її прототип має такий вигляд:

**void rewind(FILE \*fp);**

У цьому записі параметр fp представляє собою дійсний файловий покажчик.

Функція **fseek()** призначена для встановлення індикатора позиції файла у необхідному місці. Таким чином, система введення-виведення у мові С++ подібно до мови С дає змогу виконувати операції зчитування і запису даних з довільним доступом.

Прототип функції fseek():

**int fseek(FILE \*fp, long numbytes, int origin);**

У цьому  записі параметр  fp означає файловий покажчик, що повертається функцією fopen(), параметр numbytes визначає кількість байтів щодо початкового положення, які задаються параметром origin. Параметр origin може приймати одне наступних макроімен (визначених у заголовку stdio.h):

* SEEK\_SET –константа 0 – пошук з початку файлу;
* SEEK\_CUR– константа 1 – пошук з поточної позиції;
* SEEK\_END– константа 2 –пошук з кінця файлу.

Нульове  значення  результату  функції  свідчить  про  успішне  виконання функції  fseek(),  а  ненульове  –  про  виникнення  збою.  Як  правило,  функцію fseek() не рекомендується використовувати для файлів, відкритих у текстовому режимі, оскільки перетворення символів може призвести до помилкових переміщень індикатора позиції у файлі. Тому краще використовувати цю функцію для файлів, відкритих у двійковому (бінарному) режимі.

**Приклад 5.** Дано бінарний файл, що містить дійсні числа. Поміняти в ньому місцями максимальний і мінімальний елементи**.**

**#include <iostream>**

**#include <fstream>**

**#include <Windows.h>**

**using namespace std;**

**int main()**

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

**FILE \*f;**

**int i=0, pmin=0, pmax=0;**

**double max,min,v;**

**f = fopen("file\_digits.dat","r+b");**

**if(f==NULL) return 1;**

**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);**

**return 0;**

**}**

**Особливості механізмів перевантаження операторів введення-виведення даних**

У мові програмування C++ передбачено спосіб виконання операцій введення-виведення даних класів шляхом перевантаження операторів введення-виведення "<<" та ">>".

У мові C++ оператор "<<" називається оператором виведення або вставлення, оскільки він вставляє символи у потік. Аналогічно оператор ">>" називається оператором введення або вилучення, оскільки він вилучає символи з потоку. Оператори введення-виведення вже перевантажені (у заголовку ) для того, щоби вони могли виконувати операції потокового введення або виведення даних будь-яких вбудованих С++-типів. Розглянемо як можна визначити ці оператори для створення власних класів.

**Створення перевантажених операторів виведення даних**

Як простий приклад розглянемо механізм створення оператора виведення даних для класу **kooClass** на наступній програмі.

**#include <iostream>**

**#include <conio.h>**

**#include <string.h>**

**#include <stdio.h>**

**#include <Windows.h>**

**using namespace std;**

**class kooClass { public: int x, y, z; kooClass(int a, int b, int c)**

**{ x = a; y = b; z = c; };**

**};**

/**/ Перевизначений оператор введення даних**

**ostream &operator<<(ostream &stream, const kooClass &obj)**

**{ stream << obj.x << ", ";**

**stream << obj.y << ", ";**

**stream << obj.z << endl;**

**return stream;**

**}**

**main()**

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

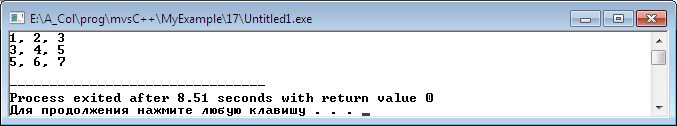
**kooClass ObjA(1, 2, 3), ObjB(3, 4, 5), ObjC(5, 6, 7);**

**cout << ObjA << ObjB << ObjC;**

**getch();**

**}**

Результат



Щоб створити операторну функцію виведення даних для об'єктів типу kooClass, необхідно перевантажити оператор виведення даних "<<". Один з можливих способів його реалізації має вигляд наданий після тексту "//ºПеревизначений оператор введення даних ". Ця операторна функція, вміст якої характерний для багатьох операторних функцій виведення даних, згідно з оголошенням, повертає посилання на об'єкт типу ostream. Це дає змогу декілька звичайних операторів виведення даних об'єднати в одному складеному виразі. Ця функція має два параметри. Перший є посиланням на потік, який використовується в лівій частині оператора "<<". Другим є об'єкт, який знаходиться у правій частині цього оператора. Саме тіло операторної функції складається з настанов виведення трьох значень координат, що містяться в об'єкті типу kooClass, і настанови повернення потоку stream.

Для будь-якій операторній функції виведення даних оператор перевантаження задається за схемою:

**ostream &operator<<(ostream &stream, class\_type obj)**

**{ // Код операторної функції виведення даних**

**return stream; // Повертає посилання на параметр stream**

**}**

У такій версії операторної функції жорстко закодований потік cout. Це обмежує перелік ситуацій, в яких її можна використовувати. Отже, програміст повинен передавати операторній функції виведення даних потік, який коректно працює в усіх конкретних випадках.

**Використання функцій-"друзів" класу для перевантаження операторів виведення даних**

У попередній програмі операторну функцію виведення даних не було визначено як член класу kooClass. Насправді ні будь-яка операторна функція виведення даних, ні функція їх введення не можуть бути членами класу. Причина в тому, що якщо операторна функція є членом класу, то лівий операнд (що опосередковано передається за допомогою показника this) повинен бути об'єктом класу, який генерує звернення до цієї операторної функції. І це змінити не можна. Проте при перевантаженні операторів виведення даних лівий операнд повинен бути потоком, а правий – об'єктом класу, дані якого підлягають виведенню. Отже, перевантажені оператори виведення даних не можуть бути функціями-членами класу. У зв'язку з тим, що операторні функції виведення даних не можуть бути членами класу, для якого вони визначаються, то виникає серйозне запитання: як перевантажений оператор виведення даних може отримати доступ до закритих членів класу?

У попередній програмі змінні x, y і z були визначені як відкриті, тому оператор виведення даних без перешкод міг отримати до них доступ. Для закритих членів класу оператор виведення даних можна зробити "другом" класу, і функція - "друг" отримує легальний доступ до його private-даних. Оголошення "другом" класу операторної функції виведення даних продемонструємо на прикладі класу kooClass.

Приклад. Демонстрація механізму використання функцій-"друзів" класу для перевантаження оператора виведення даних – опис класу kooClass змінено.

**class kooClass { private: int x, y, z;**

**public: kooClass(int a, int b, int c)**

**{ x = a; y = b; z = c; };**

**friend ostream &operator<<(ostream &stream, const kooClass &obj);**

**};**

**Створення перевантажених операторів введення даних**

Для перевантаження операторів введення даних використовують аналогічний підхід, який було застосовано при перевизначенні операторів виведення даних. В теорії перевантажений оператор виглядає так:

// Прийняття тривимірних координат x, y, z

// Перевантажений оператор введення даних для класу

istream &operator>>(istream &stream, kooClass &obj)

{ cout << "Введіть координати x, y і z: ";

// Перевантажений оператор введення даних

stream >> obj.x >> obj.y >> obj.z;

return stream;

// Повертає посилання на параметр stream }

Перевантажений оператор введення даних повинен повертати посилання на об'єкт типу istream. Окрім цього, перший параметр повинен бути посиланням на об'єкт типу istream. Цей тип належить потоку, що вказується зліва від оператора ">>". Другий параметр є посиланням на об'єкт, який приймає значення, що вводяться. Оскільки другий параметр – посилання на об'єкт, то його можна модифікувати при введенні інформації. Подібно до операторних функцій виведення даних, операторні функції їх введення не можуть бути членами класу, для оброблення даних якого вони призначені. Вони можуть бути тільки "друзями" цього класу або просто незалежними операторними функціями. За винятком того, що операторна функція введення даних повертає посилання на об'єкт типу istream, тіло цієї функції може містити все те, що програміст вважає за потрібне у неї помістити. Але логічніше використовувати оператори введення, а також виведення, все ж таки за прямим призначенням, тобто для виконання операцій введення даних.

Загальний формат перевизначеного оператора введення даних має такий вигляд:

**istream &operator>>(istream &stream, objectType &obj)**

**{ // код операторної функції введення даних**

**return stream;**

**// Повертає посилання на параметр stream**

**}**

# Потрібно зробити зауваження, що запропонована теоретична конструкція може викликати помилку при компіляції " Ambiguous overload for operator>>", яка викликана проблемами перетворення типів .

Нижче представлений працюючий код.

**#include <iostream>**

**#include <fstream>**

**#include <stdlib.h>**

**#include <conio.h>**

**#include <string.h>**

**#include <math.h>**

**#include <stdio.h>**

**#include <Windows.h>**

**using namespace std;**

**class kooClass**

**{**

**private:**

**public: int x, y, z;**

**kooClass(int a, int b, int c)**

**{ x = a; y = b; z = c; };**

**friend ostream &operator<<(ostream &stream, const kooClass &obj);**

**friend istream &operator>>(istream &stream, const kooClass &obj);**

**};**

**ostream &operator<<(ostream &stream, const kooClass &obj)**

**{ stream << obj.x << ", ";**

**stream << obj.y << ", ";**

**stream << obj.z << endl;**

**return stream;**

**};**

**istream &operator>>(istream &stream, kooClass &obj)**

**{ cout << "Enter x, y , z: ";**

**// Бажаний перевантажений оператор виведення**

**//stream>>obj.x >> obj.y>>obj.z;**

**char tmp[10];**

**stream>> tmp;**

**obj.x=atoi(tmp);**

**stream>> tmp;**

**obj.y=atoi(tmp);**

**stream>> tmp;**

**obj.z=atoi(tmp);**

**return stream;**

**// повертає посилання на поток stream**

**};**

**main()**

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

**kooClass ObjA(1, 2, 3), ObjB(3, 4, 5), ObjC(5, 6, 7);**

**cout << ObjA << ObjB << ObjC;**

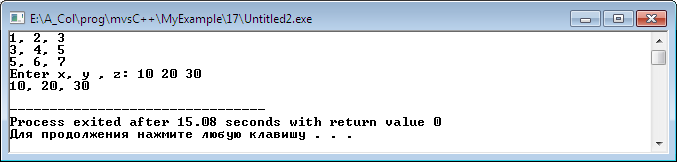
**cin >> ObjA;**

**cout << ObjA;**

**getch();**

**}**

Ось як виглядає результат виконання цієї програми:



**Форматоване виведення**.

Здійснюєтьсяфункцією:

* **printf** - виводить аргументи у стандартний потік stdout у відповідності із заданим форматом.

Щоб зв’язати програму користувача зі стандартною бібліотекою, де знаходяться ці функції, необхідно на початку програми включити заголовний файл **stdio.h**.

***Формат виклику функції printf*:**

**printf("форматний\_рядок", [список\_аргументів])**

Де *список\_аргумен*тів = аргумент {, аргумент}

*форматний\_рядок* = “”” ([літерал] | % [маркер][ширина][.точність] тип)

{( [літерал] | % [маркер][ширина][.точність] тип}”””

Форматний рядок завжди береться в подвійні лапки. Символ "%" є символом початку задання формату введення чергового аргумента. При описі формату виведення також використовуються:

*маркер* – специфікатор вирівнювання виведених знаків, пропусків, десяткових точок, вісімкових і шістнадцяткових префіксів (наприклад, вирівнювання результату перетворення по лівій межі, обов'язкове зображення знака числа тощо);

ширина– загальна ширина поля виведення;

*точність* – максимальне число цифр дробової частини числа, які будуть виведені після коми;

*тип* – специфікатор типу аргументу (наприклад, d – ціле десяткове число, f – дійсне число).

Функція printf переводить дані з внутрішнього коду в символьне представлення відповідно до форматного рядка і виводить отримані символи на екран. Форматний рядок може включати довільний текст, керуючі символи та специфікації перетворення даних. Список аргументів є необов'язковим параметром даної функції. Дану функцію можна використовувати для виводу будь-якої комбінації символів, цілих та дійсних чисел, тощо.

Функція printf може використовуватися, наприклад, для виведення повідомлення на екран:

**printf ("**Enter the source data **\n");**

Для звертання до функції використовуються параметри, розташовані у круглих дужках. Найчастіше функція **printf** реалізується для виведення значень змінних. Першим аргументом у звертанні до функції ставиться рядок форматів (береться в лапки), а наступними, якщо вони є, — об’єкти, що виводяться.

Рядок форматів може включати звичайні символи, які копіюються при виведенні, і специфікації перетворення, що починаються із символу«% », за специфікаціями йде символ перетворення. Кожна специфікація перетворення відповідає одному з аргументів, що йдуть за форматним рядком, і між ними встановлюється взаємно однозначна відповідність, наприклад:

**printf (“Values a, b, c are equal: %d %d. %d \n”, а, b, с);**

тут літера **d** у специфікації перетворення вказує, що значення аргументу має бути представлено як десяткове ціле число.

Список форматних кодів має таку форму запису:

**% [прапорець] [довжина] [точність] [f | n] [h | l] тип ,**

де **прапорець** — символ, що керує вирівнюванням виведення і виведенням пропусків, десяткової крапки, ознак чисел вісімкової і шістнадцяткової систем числення. **Прапорець** може задаватися одним із символів:

**«-»** — вирівнювання вліво усередині заданого поля;

**«+»** — виведення знака числа;

**« »** (пропуск) — приєднання пропуску до виведеного числа, якщо число є додатним і має тип зі знаком;

**«#»** — виводиться ідентифікатор системи числення для цілих: **0** — для вісімкових чисел, **0х** чи **0Х** — для шістнадцяткових чисел;

**довжина** — визначає мінімальну кількість виведених символiв, якщо довжина більше виведеної кількості символів, то виведене значення доповнюється пропусками, у випадку, коли довжина менше кількості символів у виведеному значенні або вона не задана, виводяться всі символи значення (відповідно до поля точність, якщо воно є);

**точність** — задається цілим числом після крапки і визначає кількість виведених символів, кількість знаків після крапки; на відміну від поля довжини поле точність може привести до «зрізання» виведених даних.

**f | n** — дозволяють приглушити погодження за замовчуванням про використану модель пам’яті («далека», «близька» пам’ять);

**h | l** — предикати, що визначають відповідно аргументи типів **short і long**;

**тип** — задається одним із символів: **d** — десяткове ціле; **і** — десяткове, вісімкове чи шістнадцяткове ціле зі знаком; **с** — одиночний символ;**u** — беззнакове десяткове число; **х, X** — беззнакове шістнадцяткове число; **0** — вісімкове число; **s** — сприймає символи без перетворення до символу **«\n»** або пропуску, доки не буде досягнута задана довжина (при виведенні видає до потоку всі символи до символу **«\0»** або до досягнення специфікованої точності); **f, F** — значення з плаваючою крапкою; **е, Е** — значення у експоненціальній формі; **G, g** — значення зі знаком у формі **f** або **е**.

**Символи формату**

| **Символ**  **формату** | **Тип**  **виводимого значення** |
| --- | --- |
| c, С | **char; одиничний літерал (символ)** |
| d, i | **int**; цілі зі знаком |
| o | **int**; без знакові цілі у восьмирічній системі числення |
| u | **int**; цілі зі знаком |
| x, X | **int**; беззнакові цілі у шістнадцятирічній системі числення |
| f | **float**; дійсні числа типу **float** (фіксована крапка) |
| lf | **double**; дійсні числа типу **double** (фіксована крапка) |
| e, E | **double**; дійсні числа в експоненційній формі (плаваюча крапка) |
| g, G | **double**; дійсні числа |
| P | **void**; вказівник |
| % | знак % |

**Керуючі символи(*Escape* – послідовності)**

|  |  |  |  |
| --- | --- | --- | --- |
| **Chr** | **Dec** | **Hex** | **Значення** |
| '\a' | '\07' | '\x07' | Звуковий сигнал |
| '\b' | '\08' | '\x08' | Повернення на одну позицію назад (Backspace) |
| '\f' | '\14' | '\x0c' | Перехід на нову сторінку |
| '\n' | '\12' | '\x0a' | Перехід на новий рядок |
| '\r' | '\15' | '\x0d' | Повернення каретки (повернення на початок рядка) |
| '\t' | '\11' | '\x09' | Горизонтальна табуляція |
| '\v' | '\13' | '\x0b' | Вертикальна табуляція |
| '\\' | '\134' | '\x5c' | Зворотна коса риска |
| '\'' | '\47' | '\x27' | Апостроф |
| '\"' | '\42' | '\x22' | Подвійні лапки |
| '\?' | '\77' | '\x77' | Знак питання |

Виведення результатів з використанням форматних кодів функції **printf** може мати вигляд:

**printf (” % 3.0 f % 6.1 f \ n ”, х, у);**

***Приклад .*** Обчислити значення функції **у = ах2 – sinx**, якщо **а =  10,5; х є [-1; 2]; hx = 0,5**.

**#include <stdio.h>**

**#include <math.h>**

**#include <conio.h>**

**main( )**

**{ float x, y, a(10.5);**

**printf ("\t Vivod rezultata\n");**

**for (x = -1; x <= 2; x += 0.5)**

**{** **у = a \* pow(x,2) - sin(x);     //у = a\*x\*x - sin(x);**

**printf (" \t x = % 4.1f    у = % 6.3f \n", x, y);**

**}**

**getch ();** *//затримка екрана*

**}**

Результати обчислення:  
**Vivod rezultata**  
**x = -1.0 у =11.342**  
**х = -0.5 у = 3.104**  
**х = 0.0  у = 0.000**  
**х = 0.5  у = 2.146**  
**х = 1.0  у = 9.659**  
**х = 1.5  у = 22.628**  
**х = 2.0  у = 41.091.**

*Для самостійного вивчення*: Поглибити матеріал лекції за наданою літературою. Вивчення лекційного матеріалу та додаткових джерел. Розгляд запитань і виконання завдань для самостійної роботи, запропонованих на лекції.

*Література*

1. Джейс Либерти Освой самостоятельно С++ за 21 день: 3-е изд. пер. с англ.: Уч. пос. – М.: Издательский дом „Вильямс”, 2001. – 816 с.: ил..
2. Павловская Т.А. С/С++. Программирование на языке высокого уровня. СПб.: Питер, 2003. – 461 с. URL:  <http://www.ph4s.ru/bookprogramir_1.html>
3. Липпман С. Б., Лажойе Ж. Язык программирования С++: Вводный курс. — М.: ДМК, 2001. URL: <http://www.insycom.ru/html/metodmat/inf/Lipman.pdf>
4. Дейтел Х., Дейтел П. Основы программирования на С++. – М.: Бином, 1999. – 1024 с. URL:  <http://ijevanlib.ysu.am/wp-content/uploads/2018/03/deytel.pdf>
5. Бадд Т. Объектно-ориентированное программирование в действии. [2-е изд.] – СПб.: Изд-во "Питер". 1997.  URL: <http://khizha.dp.ua/library/Timothy_Budd_-_Introduction_to_OOP_(ru).pdf>.
6. *Герб Саттер,Андрей Александреску* "Стандарты программирования на С++ ", [*Вильямс, 2005*](http://www.williamspublishing.com/)*; 304 с.*
7. *Скотт Мейерс.* Эффективное использование C++. 50 рекомендаций по улучшению ваших программ и проектов. *"ДМК", 2000; 240 с.*
8. *Скотт Мейерс.* Наиболее эффективное использование C++. 35 новых рекомендаций по улучшению ваших программ и проектов.*"ДМК",2000;304 с.*
9. Г.Буч. Об‘єктно-орієнтоване проектування з прикладами застосування. – К.: Видавничий центр "Академія". 2002. ­ 499 с.
10. Страуструп Б. Язык программирования С++· Киев: "ДиаСофт", 1993. - 256 с.

*Контрольні запитання*.

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. Назвіть класи для створення файлових потоків і їх призначення.
7. Які дії необхідно виконати для використання файлів у програмі?
8. Які існують режими відкриття файлів?
9. Для чого призначений метод open() і які він має параметри?
10. Які способи відкриття та закриття файлу ви знаєте?
11. Як записати дані у файл?
12. Як прочитати дані з файлу?
13. Опишіть механізмів перевантаження операторів введення-виведення даних.
14. Яка стандартна бібліотека застосовується при форматованому введенні-виведенні даних?
15. Запишіть специфікацію формату функції **printf**?
16. За що відповідають параметри **флаг, ширина** і **точність**? Чому вони вказані у квадратних дужках?
17. Як вказати кількість значущих розрядів цілої і дробової частини дійсного числа з фіксованою крапкою?
18. Для чого застосовуються керуючі символи (*Escape* – послідовності)? Наведіть приклади.