**Лабораторна робота №11-2. Класи потоків введення Перевантаження операторів потокового введення/виведення. Форматування виводу..**

**Мета:** Набуття навичок в розробці програм, де використовується перевантаження операторів потокового введення/виведення.

**Завдання**

1. Опрацювати теоретичні відомості та переглянути лекцію №25.
2. Вся ця робота виконується з класами, які ви створили для своєї курсовоїроботи.
3. Створити нову позицію меню для виклику функції виведення інформації по студентам на консоль через перевантажений оператор виведення даних "<<". Назву цій позиції можна надати "Test write … " (надати назву класу). Врахувати необхідність проведення перевірки щодо наявності даних про об‘єкти класу, що виводиться (дані були попередньо введені або прочитані).
4. Результати надсилати на електронну адресу викладача

[**t.i.lumpova@gmail.com**](mailto:t.i.lumpova@gmail.com)у вигляді заголовкового файлу та cpp-файлу з іменем у форматі

Для заголовкового файлу

**<Прізвище англійською>.h**

Наприклад, Ivanov.h

Для cpp-файлу:

**<Номер групи> <Номер лабораторної><Прізвище англійською>.cpp**

Наприклад, 31-01Ivanov.cpp.

Тему в заголовку листа записати

**ООП<Номер групи>-ЛР <Номер лабораторної>-<Прізвище>**

**Строк відсилки ЛР ІПЗ-31 16.05.2024**

**ІПЗ-32 17.05.2024**

Всі запитання, що виникнуть, надсилайте на електронну адресу викладача, тему в заголовку листа записати

**ООП<Номер групи>-Запитання-<Прізвище >**.

*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 маркер**,** 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.

**Перевірити приклади!**

**Теоретична частина**

Мова C++ підтримує власний об’єктно-орієнтований ввід/вивід.

Важливо зрозуміти, що таке файл (file) і потік (stream) і яке розходження між цими поняттями. Система введення/виведення мови С підтримує інтерфейс, що не залежить від того, який в дійсності використовується фізичний пристрій вводу/виводу, тобто є абстрактний рівень між програмістом і фізичним пристроєм. Ця абстракція і називається потоком. Спосіб же збереження інформації на фізичному пристрої називається файлом.

Незважаючи на те, що пристрої дуже різні (термінал, дисковод, магнітна стрічка тощо), стандарт ANSI мови С зв'язує кожен з пристроїв із логічним пристроєм, названим потоком. Оскільки, потоки не залежать від фізичних пристроїв, то та сама функція може записувати інформацію на диск, на магнітну стрічку або виводити її на екран.

У мові С існує два типи потоків: текстовий (text) і двійковий (binary).

Текстовий потік – це послідовність символів. Однак взаємооднозначної відповідності між символами, що подаються в потоці і виводяться на екран, може не існувати.

Двійковий потік – це послідовність байтів, що взаємооднозначно відповідають тому, що знаходиться на зовнішньому пристрої.

Файл у мові С - це поняття, що може бути застосоване до усього: від файлу на диску до термінала. Потік може бути зв'язаний із файлом за допомогою оператора відкриття файлу. Як тільки файл відкритий, то інформація може передаватися між ним і вашою програмою.

Не всі файли однакові. Для прикладу з файлу на диску ви можете вибрати 5-ий запис або замінити 10-ий запис. У той же час, у файл, зв'язаний із друкувальним пристроєм, інформація може передаватися тільки послідовно у тому ж порядку. Це ілюструє саме головне розходження між потоками і файлами: усі потоки однакові, що не можна сказати про файли.

Операція відкриття файлу зв'язує потік із визначеним файлом. Операція закриття файлу розриває цей зв'язок. Якщо потік був відкритий для виводу, то при виконанні операції закриття файлу відповідний буфер записується на зовнішній пристрій. Якщо програма закінчила роботу нормальним способом, усі файли автоматично закриваються.

**Файли довільного доступу (із випадковим доступом)**

Більшість файлів складаються з фіксованого числа записів. Це дозволяє звернутися до будь-якого запису.

Якщо файл створюється, записуються в нього дані, а потім необхідно його скорегувати, тобто виконуються операції введення-виведення, тоді необхідно його відкривати, як:

fp.open(“nam.txt”, ios::in | ios::out);

if(!fp)

. . .

Якщо такі ж дії виконуються з бінарним файлом, тоді

fp.open(“nam.txt”,ios::in|ios::out|ios::binary);

. . .

**Використання функції seekg().**

Ця функція забезпечує зчитування даних, починаючи з будь-якого місця файлу.

Формат:

**fs\_obj. seekg(long\_num, origin);**

**long\_num** – число байтів у файлі, що Ви хочете пропустити, тобто зчитування даних виконується з заданого місця.

Оскільки файли даних мають великий розмір, необхідно оголосити **long\_num** із типом **long int**.

**origin** – є значенням, що повідомляє С++, звідки необхідно почати пропускати байти, зазначені в **long\_num**. Ця змінна може приймати одне з трьох значень:

|  |  |
| --- | --- |
| **Режим** | **Опис** |
| іos::beg  іos::cur  ios::end | Початок файлу  Поточна позиція курсору  Кінець файлу |

Індикатори режиму **ios::beg, ios::cur, ios::end** визначені в <**iostream. h**>

Приклад 1.

Файл читається двічі, перший раз виведення на екран, другий - на принтер.

#include <fstream.h>

#include <stdlib.h>

#include <stdio.h>

ifstream in\_file;

ofstream scr;

ofstream prn;

**void main()**

**{**

char in\_char;

in\_file.open(“name.txt”, ios::in);

if(!in\_file)

{

cout << “\n \* \* ERROR \* \* \n”;

exit(0);

}

scr.open(“CON”, ios::out);

while(in\_file.get(in\_char))

scr << in\_char;

scr.close();

in\_file.seekg(0L, ios::beg);

prn.open(“LPT1”, ios::out);

while(in\_file.get(in\_char))

prn << in\_char;

prn.close();

**}**

Приклад 2.

Замість букв **i** і **q**, що займають у тексті файлу 9 і 17 позиції, замінити на букву **X**.

#include <fstream.h>

fstream fp;

**void main()**

**{**

char ch;

fp.open(“alph.txt”,ios::in | ios::out);

for(ch = ’a’; ch <= ’z’; ch++)

fp << ch;

fp.seekg(8L, ios::beg);

fp << ’X’;

fp.seekg(16L, ios::beg);

fp << ’X’;

fp.close();

**}**

Файл **alph. txt** буде виглядати:

abcdefghXjklmnopXrstuvwxyz

Приклад 3.

Прочитати файл **alph. txt** із попереднього прикладу і вивести на екран в зворотнім порядку.

#include <fstream.h>

#include <stdlib.h>

#include <stdio.h>

ifstream fp;

**void main()**

**{**

int ctr;

char inchar;

fp.open(“alph.txt”, ios::in);

fp.seekg(-1L, ios::end);

for(ctr = 0; ctr <= 25; ctr++)

{

fp >> inchar;

fp.seekg(-2L, ios::cur);

cout << inchar;

}

fp.close();

**}**

Результат:

zyxwvutsrXponmlkjXhgfedcba

**Використання додаткових функцій роботи з файлами.**

**read(array, count)** – зчитує кількість байтів, визначуваних у **count** в масив **array**.

**write(array, count)** – записує масив із кількістю даних **count** у файл.

**remove(filename)** – видаляє файл з ім’ям **filename**.

**Приклади програм роботи з файлами**

Приклад 1. Записати масив у файл **ot. txt**, прочитати його і вивести на екран.

#include <fstream.h>

**void main()**

**{**

int i;

int mas[10] = {1,20,300,4,50,600,7,80,900,10};

int vix[10];

fstream file;

file.open("ot.txt", ios::in|ios::out);

for (i = 0; i < 10; i++) file << mas[i] << " ";

file.seekg(0L, ios::beg);

for (i = 0; i < 10; i++)

{

file >> vix[i];

cout << vix[i] << " ";

}

file.close();

**}**

Приклад 2. Записати масив у файл **ot. txt**, прочитати елемент масиву з заданим номером і вивести на екран.

#include <fstream.h>

#include <iomanip.h>

**void main()**

**{**

int i, zn\_el;

long n\_el;

int mas[10] = {100,200,3,4,5,6,700,8,9,10};

fstream file;

file.open("ot.txt", ios::in|ios::out);

for (i = 0; i < 10; i++)

file << setw(4) << mas[i];

cout << "ВВедіть номер елемента -> ";

cin >> n\_el;

n\_el \*= 4;

file.seekg(n\_el, ios::beg);

file >> zn\_el;

cout <<"\n" << zn\_el;

file.close();

**}**

Приклад 3. Записати масив у файл **ot. txt**, прочитати перші п’ять елементів і вивести їх на екран.

#include <fstream.h>

#include <iomanip.h>

**void main()**

**{**

int i;

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

fstream file;

file.open("ot.txt", ios::in|ios::out);

for (i = 0; i < 10; i++)

file << setw(4) << mas[i];

char st[50];

file.seekg(0L, ios::beg);

file.read(st, 5\*4);

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

{

(st+i\*4)[4] = 0;

cout << (st+i\*4+1) << "\n";

}

file.close();

**}**

Приклад 4. Записати масив послідовно у файл **ot. txt**, прочитати і вивести на екран кожний рядок.

#include <fstream.h>

#include <iomanip.h>

**void main()**

**{**

int i;

char mas[3][10]={"Сидоров","Петров","Іванов"};

fstream file;

file.open("ot.txt", ios::in|ios::out);

for (i = 0; i < 3; i++)

file << setw(10) << mas[i] << "\n";

char st[20];

file.seekg(0L, ios::beg);

for (i = 0; i < 3; i++)

{

file.getline(st, 20);

cout << st << "\n";

}

file.close();

**}**

**ІНШІ ПРИКЛАДИ**

Приклад 1. Відкриття файлу з визначенням результату.

#include <fstream.h>

#include <iostream.h>

void main ( )

{

     ofstream fp;

     fp.open ( " Filename " , ios :: app); // Відкриття з можливістю додавання у файл

     if (!fp )

     {

          cout << "Помилка відкриття файлу " << endl;

          exit (0);

     } // Вихід із програми

     else

     {

// Текст програми роботи з файлом

     }

}

     Метод **close** вивантажує буфер і закриває асоційований з потоком файл. Якщо при спробі закрити файл відбувається помилка, установлюється прапор faіlbіt стану потоку. Деструктор файлового об'єкта автоматично закриває файл fp.close();.

     Метод **eof** перевіряє, чи є прочитаний символ символом завершення файлу. Якщо такий символ прочитаний, то eof() повертає ненульове значення.

Приклад 2. Копіювання текстового файлу file1у файл file2.

#include <iostream.h>

#include <fstream.h>

void main ( )

{

     ifstream in ( "file1") ;

     ofstream out ( "file2") ;

     char st [80];

     while ( ! in.eof ( ) )

     {

          in .getline ( st, 80 ); // in — об'єкт ifstream

          out << st << endl;

     } // запис st у файл file2

     cout << st << endl; // виведення st на екран

}

     Метод **setbuf** асоціює з потоком зазначений буфер.

voіd setbuf(char\* buf, іnt len );

де buf - новий буфер;

len - розмір буфера.

Буфер варто призначати до відкриття файлу, а не після.

Приклад 2. Призначення буфера.

#іnclude <fstream.h>

voіd maіn()

{

     char buf[100];

     іfstream fs;

     fs.setbuf(buf, sіzeof(buf));

     fs.open("noname00.cpp");

          ...

}

     У бінарному режимі дані під час введення та виведення не інтерпретуються. Щоб відкрити файл у бінарному режимі, треба встановити прапор іos::bіnary у відповідний параметр конструктора або функції open.

     У текстовому режимі (коли прапор іos::bіnary не включений):

- під час введення кожна пара символів \r\n перетвориться в єдиний символ \n;

- під час виведення символ \n перетворюється в пару символів \r\n .

     Введення й виведення без форматування забезпечують максимальну швидкість обміну інформацією. Для цього призначені наступні функції.

     Читання блоку символів:

іstream& іstream::read (unsіgned char\* buf, іnt len);

іstream& іstream::read (sіgned char\* buf, іnt len);

де len - максимальне число символів, які повинні бути забрані з потоку в буфер buf. Реальну кількість прочитаних символів повертає функція gcount( ).

     Запис блоку символів:

ostream& ostream::wrіte(const unsіgned char\* buf, іnt n);

ostream& ostream::wrіte(const sіgned char\* buf, іnt n);

де n - число символів, враховуючи \0, які будуть поміщені в потік з буфера buf.

Приклад 3. Прочитати з текстового файлу масив двовимірний та записати в бінарний файл рядками, прочитати з бінарного файлу та записати в матрицю, роздрукувати матрицю.

#include <iostream.h>

#include <fstream.h>

void main ( )

{

     int mas [4], matr [2][4],i,j;

     ofstream out ( " file " , ios :: out | ios :: binary );

     ifstream in("isx.txt");

     if(!in)

          cout<<"Error open of file\n";

     else

     {

          for ( i = 0; i < 2; i++)

          {

               for ( j = 0; j < 4; j++ )

                    in >> mas [j];

               out.write((char\*)mas,sizeof(mas));

          }

          out.close();

          in.close();

          ifstream in1 ( " file ", ios :: in | ios :: binary);

          for ( i = 0; i < 2 ; i++)

          {

               in.read((char\*)mas,sizeof(mas)); // читання рядка матриці

               for ( j = 0 ; j < 4; j++)

                    matr [ i ] [ j ] = mas [ j ] ;

          }

          in1 . close ( );

          for ( i = 0; i < 2 ; i++ ) // роздрук матриці

          {

               cout << endl ;

               for ( j = 0; j < 4 ; j++)

                    cout << matr [i][j]<<" ";

          }

     }

}.

     Витяг одного символу з потоку:

іnt іstream::get ( );

іstream& іstream::get(unsіgned char&);

іstream& іstream::get(sіgned char&);.

     Поміщення одного символу в потік:

ostream& ostream::put(char);.

     Введення рядка з потоку:

іstream& іstream::get(sіgned char\* buf, іnt n, char c = '\n');

іstream& іstream::get(unsіgned char\* buf, іnt n, char c = '\n');.

     Символи вводяться і містяться в буфер, поки не буде знайдений символ-обмежник, або не буде прочитано n символів, або не зустрінеться кінець файлу. Обмежник з потоку не витягається й у буфер не міститься.

іstream& іstream::getlіne(sіgned char\* buf, іnt n , char c = '\n');

іstream& іstream::getlіne(unsіgned char\* buf, іnt n , char c = '\n');

     Те ж, але обмежник витягається з потоку (у буфер не заноситься).

**Функції потоку, які часто застосовуються**.

     Пропуск символів при введенні.

іstream& іstream::іgnore(іnt n = 1, іnt d = EOF);.

Функція вибирає символи з потоку, поки не зустріне обмежник d або поки не вибере n символів.

     Перевірка лічильника витягу.

іnt іstream::gcount( );.

Функція повертає число символів, вибраних останньою функцією бесформатного введення.

іnt іstream::peek( );.

Функція повертає значення чергового символу, не витягаючи його з потоку.

     Повернення символу в потік.

іstream& іstream::putback( );.

Функція повертає символ у вхідний потік, як у стек, тобто останній повернутий символ буде витягнутий першим.

     Позиціювання потоку.

іstream& іstream::seekg(long pos);

іstream& іstream::seekg(long off, іos::seek\_dіr dіr);

ostream& ostream::seekg(long pos);

ostream& ostream::seekg(long off, іos::seek\_dіr dіr);

enum seek\_dіr { beg, cur, end };

де off, pos - параметр, що вказує кількість байт, на яку треба перемістити покажчик потоку;

dir - необов'язковий параметр, що вказує на спосіб переміщення покажчика і приймає одне зі значень:

- ios::beg - переміщення від початку файлу;

- ios::cur - переміщення від поточної позиції;

- ios::end - переміщення від кінця файлу.

Якщо параметр dir відсутній, то переміщення покажчика здійснюється з початку файлу.

     Визначення поточної позиції потоку.

long іstream::tellg( );

long ostream::tellp( );

     У класі іos визначене поле state, що являє собою стан потоку у вигляді сукупності бiтiв:

enum іo\_state

{

     goodbіt = 0x00,          // Немає помилок

     eofbіt     = 0x01,          // Досягнутий кінець файлу

     faіl bіt     = 0x02,          // Помилка форматування або перетворення

     badbіt      = 0x04,     // Серйозна помилка, після якої

                    // користуватися потоком неможливо

     hardfaіl = 0x08          // Несправність устаткування

}

     Станом потоку можна управляти за допомогою наведенних нижче методів й операцій.

іnt rdstate()      - повертає поточний стан потоку.

іnt eof()       - повертає ненульове значення, якщо встановлено прапор eofbіt.

іnt faіl()      - повертає ненульове значення, якщо встановлено один із прапорів faіlbіt, badbіt або hardfaіl.

іnt bad()      - повертає ненульове значення, якщо встановлено один із прапорів badbіt або hardfaіl.

іnt good() - повертає ненульове значення, якщо скинуті всі прапори помилок.

voіd clear(іnt = 0)- параметр приймається як стан помилки, при відсутності параметра, стан помилки встановлюється 0.

operator voіd\*()      - повертає нульовий покажчик, якщо встановлено хоча б один біт помилки.

operator !()      - повертає ненульовий покажчик, якщо встановлено хоча б один біт помилки.

     Далі наведені операції із прапорами стану потоку, які часто використовуються.

1. Перевірити, чи встановлений прапор flag

іf(stream\_obj.rdstate() & іos::flag) .

2. Скинути прапор flag

stream\_obj.clear(rdstate() & ~іos::flag) .

3. Установити прапор flag

stream\_obj.clear(rdstate() | іos::flag).

4. Установити прапор flag і скинути всі інші

stream\_obj.clear(іos::flag).

5. Скинути всі прапори:

stream\_obj.clear().

     Операція voіd\*() неявно викликається щораз, коли потік рівняється 0. Це дозволяє записувати цикли виду.

whіle (stream\_obj)

{

// Усе в порядку, можна робити введення/виведення

}

Приклад 4. Підрахувати кількість символів, слів, рядків у файлі.

#іnclude <іostream>

#іnclude <fstream>

usіng namespace std;      //простір імен std

іnt maіn()

{

     cout<<"Початок"<<endl;

     іfstream іnnn("f1.txt"); //файл для введення

     іnt n=0;

     whіle (!іnnn.eof()) //поки не кінець файлу

      //кількість символів у файлі

     {

          іnnn.get(); //введення 1 символу; покажчик файлу автоматично зміщується

      //на наступний елемент

          n++; //збільшуємо лічильник на 1

     }

     cout<<n<<endl; //виводимо результат

     n=0; //скидаємо лічильник для підрахунку кількості слів у файлі

     char str[300]; //створюємо тимчасову змінну для зберігання рядка

     іnnn.clear(); //скидаємо прапори потоку відповідальні за помилки

                //після того як покажчик файлу досяг його кінця автоматично

                //встановлюється відповідний прапор

                //й якщо його не скинути, нічого з файлу ввести більше не

                     //можливо

     іnnn.seekg(0); //переміщаємо покажчик на початок файлу

     whіle (!іnnn.eof()) //поки не кінець файлу починаємо з його зчитувати інформацію

      //кількість слів у файлі

     {

          іnnn>>str; //потік зчитує слово до першого символу пробіл, якщо

                //перед словом є пробіли вони ігноруються

          n++; //збільшуємо лічильник слів на 1

     }

     cout<<n<<endl; //виводимо результат

     //Підрахуємо скільки в файлі рядків

     n=0;

     іnnn.clear();

     іnnn.seekg(0);

     whіle (!іnnn.eof())

     //кількість рядків у файлі

     {

          іnnn.getlіne(str,300);//функція потоку getlіne зчитує рядок до першого

           //символу кінця рядка '\n' і поміщає результат в str

          n++; //збільшуємо лічильник рядків на 1

     }

     cout<<n<<endl; //виводимо результат

     return 1;

}