Потоци

Данните в компютърните системи се обработват не на куп, а под формата на поток - четенето и писането от файлове се извършват бит по бит, а също и това към и от конзолата. Навигацията в даден поток се случва чрез местене на специален указател за текущата позиция.

Ще разгледаме два примера за потоци - файлове и потоци за вход/изход

Файлове

Четенето и писането на файлове в езика C++ се извършва с помощта на следните три обект от библиотеката `fstream`

- `ofstream` - писане към файлове (**o**utput **stream**)
- `ifstream` - четене от файлове (**i**nput **stream**)
- `fstream` - четене и писане

Отваряне на файл

Отварянето на файл се случва чрез метода `open()`, който приема следните аргументи: път към файла и режим на отваряне.

Съществуват няколко режима на отваряне:

- ios::in - отваря файла за четене
- ios::out - отваря файла за писане
- ios::binary - отваря файла в бинарен режим
- ios::ate - началната позиция на файла се задава в края на файла
- ios::app - всяко писане се случва в края на файла
- ios::trunc - ако файлът вече съществува, неговото съдържание се изтрива и се започва отначало

```
ofstream testFile;
testFile.open("test.txt", ios::in);
```

Същото може да се напише и по следния начин:

```
ofstream testFile("test.txt", ios::in);
```

Преди да направим каквото и да е с файла, трябва да проверим дали той е отворен правилно. Това се случва със следната проверка:

```
if(testFile.is_open()){ //Можем да работим с файла}
```

Затваряне на файл

След като приключим работа с файла, трябва задължително да го затворим. Това става със следната операция:

```
testFile.close();
```

След като файла е затворен, можем да използваме същия обект да отворим друг файл.

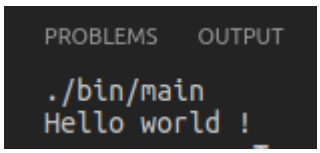
Четене от файл

Следният фрагмент от код чете символ по символ файла `test.txt`

```
std::ifstream testFile;
char c;
testFile.open("test.txt", std::ios::out);

if(testFile.is_open())
{
    while(testFile.get(c))
        std::cout << c;
}

testFile.close();
```



Писане към файл

Писането на файл се случва по следният начин:

```
std::ofstream testFile;
testFile.open("test.txt", std::ios::in | std::ios::app);

if(testFile.is_open())
{
    testFile << "\nThis is a new line !\n";
}
testFile.close();
```

Задаваме и флаг ios::app с цел да пишем в края на файла, без да изтриваме старото му съдържание.

Флагове

Освен флага is_open(), съществуват и други флагове за проверка на състоянието на файла. Всички флагове връщат boolean.

- bad() - връща `true` ако операцията по четене или писане се провали

- fail() - също като `bad()`, но също и ако има грешка във форматирането (прочитане на string като число)
- eof() - връща `true`, ако е достигнат края на файла
- good() - връща `false` ако някой от горните флагове са `true`

Допълнително четене :

http://www.cplusplus.com/doc/tutorial/files/

Потоци за вход и изход

Понякога искаме да използваме `std::cout <<` или `std::in >>` върху обекти на класове (примерно, да изведем елементите в `MyVector`, или пък да ги въведем)

Можем да предифинираме операторите за потоци. Това става по следният начин:

Създаваме следните friend функции:

```
friend std::ostream& operator << (std::ostream& out, const MyVector&);
friend std::istream& operator >> (std::istream& in, MyVector&);
```

Защо приятелски функции ?

Когато извикаме `std::cout << object`, обектът `ostream` търси съответен метод за обработка на подадения му обект, или като глобална функция. Също така, това да модифицираме класа ostream не е добра идея (ostream/istream са зависими от компилатора, и това е проблем ако искаме кода ни да работи на различни среди)

Чрез приятелските функции ние можем да имаме достъп до член-данните на `MyVector` от глобална функция.

Защо типът на функцията е `ostream&`

`osteam` и `istream` са единствени инстанции, т.е. не съществуват други `osteam` и `istream` и трябва да работим със същите.

Освен това, подаваме аргумент самия поток, за да можем да имаме достъп до входно/изходните функции

Пример

```
std::ostream& operator << (std::ostream& out, const MyVector& v)
{
    for(int i = 0; i < v.size; i++)
    {
        out << v.values[i] << " ";
    }
    return out;
}
```

```
std::istream& operator >> (std::istream& in, MyVector& v)
{
    int n, t;

    std::cout << "Enter amount of items to enter: ";
    in >> n;

    for(int i = 0; i < n; i++)
    {
        in >> t;
        v.push_back(t);
    }
}
```

Забелязваме, че вместо std::cout използваме out, и std::cin използваме in. Това е защото, във съответните оператори, cout/cin са заменени от out/in

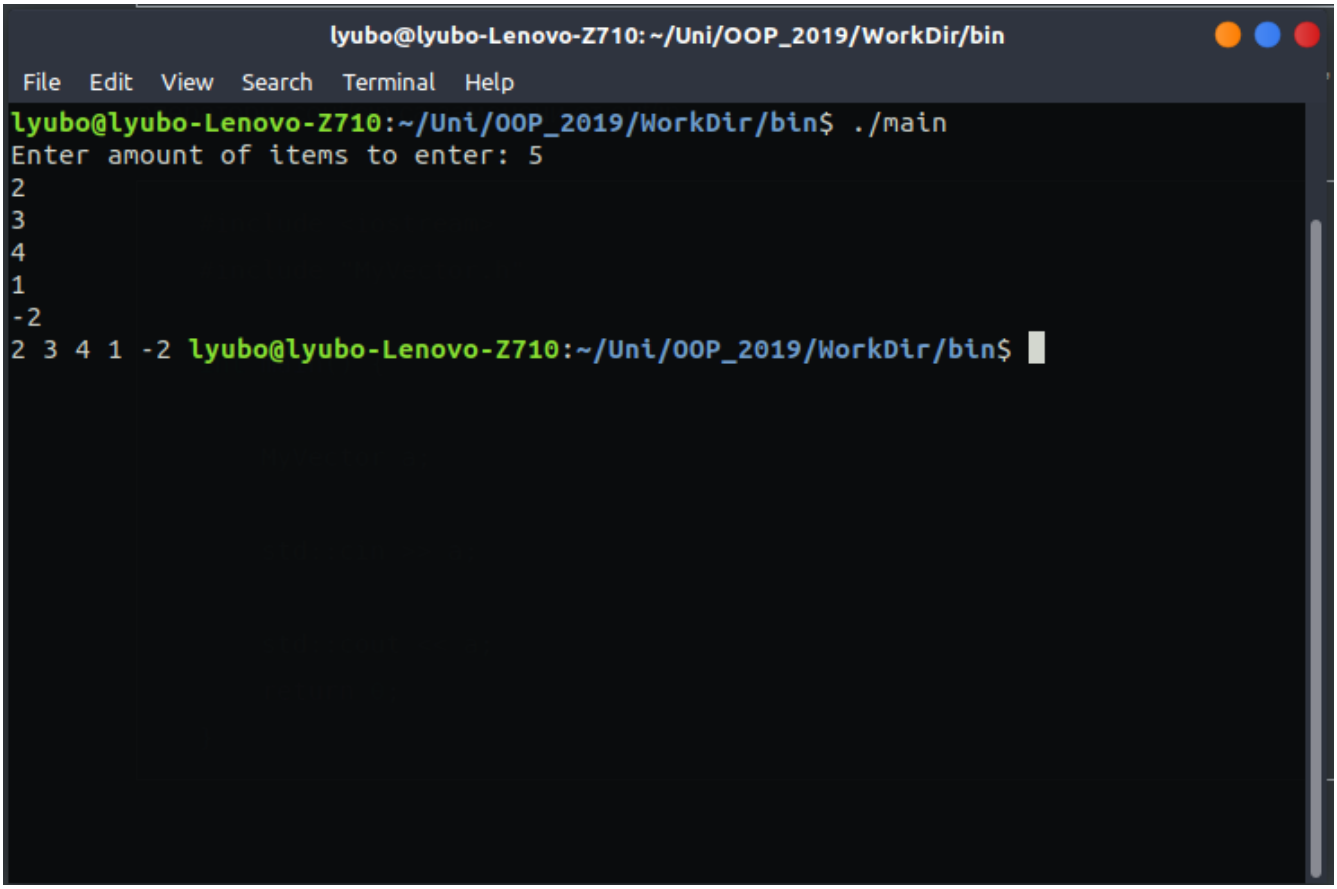
```
#include <iostream>
#include "MyVector.h"

int main() {

    MyVector a;

    std::cin >> a;

    std::cout << a;
    return 0;
}
```


lyubo@lyubo-Lenovo-Z710:~/Uni/OOP_2019/WorkDir/bin
File Edit View Search Terminal Help
lyubo@lyubo-Lenovo-Z710:~/Uni/OOP_2019/WorkDir/bin$ ./main
Enter amount of items to enter: 5
2
3
4
1
-2
2 3 4 1 -2 lyubo@lyubo-Lenovo-Z710:~/Uni/OOP_2019/WorkDir/bin$