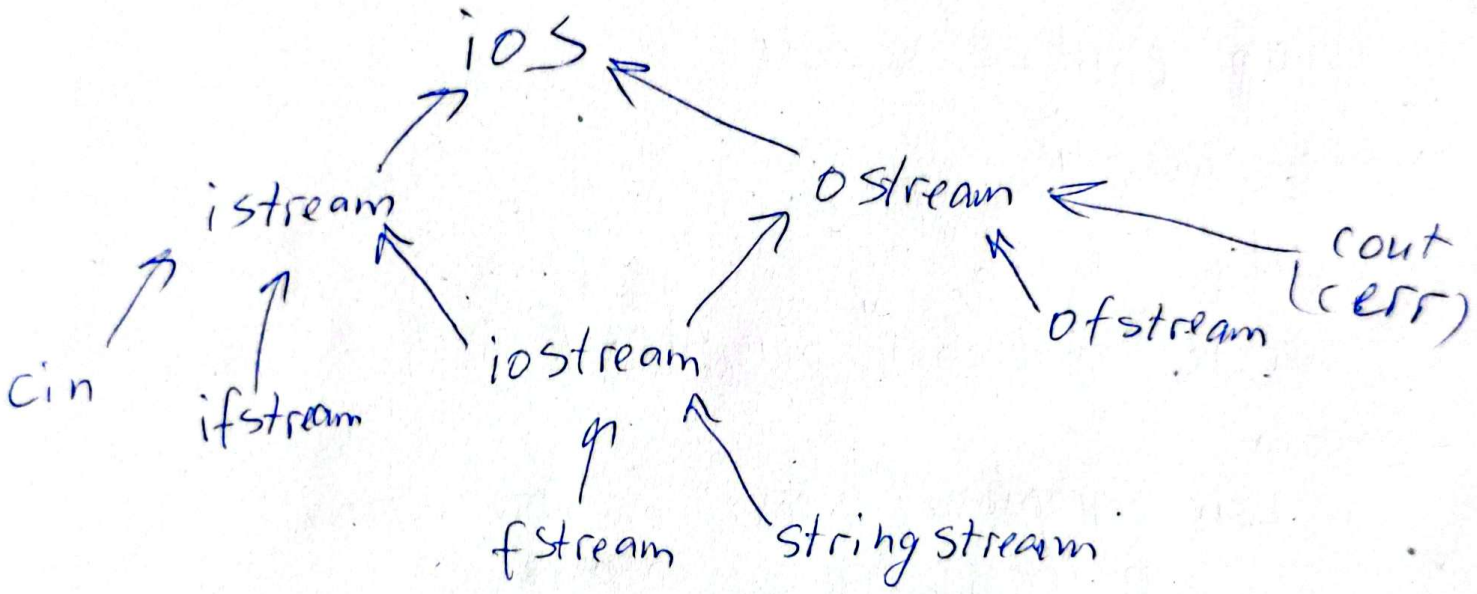
Тема 2

Поток. Стандартни потоци. Текстови файлове. Йерархия на потоците. Интерфейс на потоци. Потоци за вход/изход от файл. Режими на работа. Флагове на състоянията на потока. Позициониране във файл.

Поток

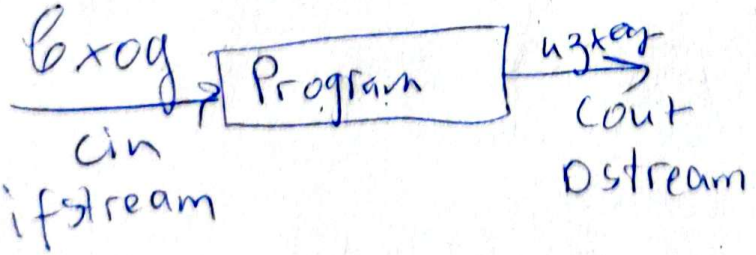
- наредена последователност от байтове
- с origin и destination

Йерархия на потоците



Стандартни потоци

Диагр.



- вход
  - байтовете идват от източник за вход (клавиатура, файл, мрежа, програмата)
- изход
  - байтовете излизат от програмата и отиват във външен приемник (конзола, файл, ...)

- потоците работят като посредници между програмиста и I/O устройствата, така освобождават програмиста от работа с I/O операции на ниско ниво.

- стъпки с работа върху поток в C++:
  създаване на поток → свързване с I/O устройства → извършване на I/O операции → прекъсване на връзката (освобождава се потока)

Потоци за изход: ostream
- форматиран: оператор <<
  Пример: cout << 3 << 4;
- неформатиран
  put(char ch)
  .write(const char* str, size_t size);
- работа
  а) към конзолата (обект от тип ostream)
  ostream os << <обект>
  б) към файл (обект от тип ofstream)
  ofstream ofs(<име_на_файл>, режим на работа);

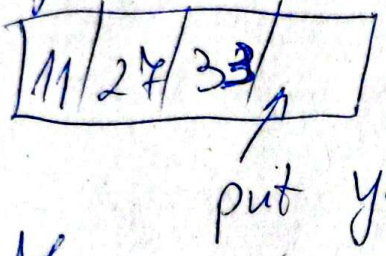
⚠ Проверка дали потока е отворен
```
if(!ofs.is_open())
{ throw std::runtime_error("Test");
}
```

⚠ За затварянето се извиква функцията close() или автоматично се извиква при края на scope-а

- Синхронизиране
  .flush() - записва буфера във файла

⚠ close() извиква ~~flush()~~;

- позициониране
  имаме put указател, който сочи към настоящия адрес, към който ще се записва в паметта.



put указател.

Ф-ции от интерфейса:

.tellp() - показва къде в паметта се намира put указателя.

.seekp(size_t idx) - премества put указател на idx

.seekp(offset, dest) премества put указателя като:

- offset - с какво отместване да бъде put указателя от подадената dest.
  0 - цяла същата позиция
- dest - ios::begin - началото на потока
  - ios::curr - настоящата позиция
  - ios::end - на края на потока

Потоци за вход (istream)

- форматирано : operator>>

Пример
```
int i;
cin >> i;
```
istream cin >> <обект> i

- неформатиран

.get(); чете 1 символ

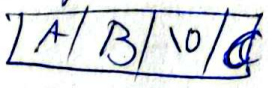
.get(buff, size) == get(buff size, '\n')

.get(buff, size, symb) - чете или size на брой байтове или докато не срещне <symb>, - всичко се записва в buff (без <symb>)

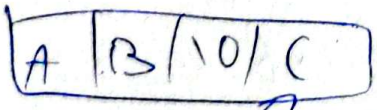
.getline(buff, size): - чете size на брой или докато не срещне нов ред, като прескача '\n' и поставя get указателя след това.

.getline(buff, size, symb);

Пример: имаме стринг

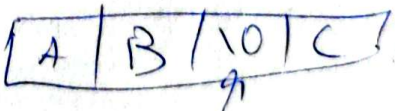


getline(buff, 32)
AB



get

get(buff, 32);
AB



get

- работа

а) към конзолата (обект от тип istream)

```
int i;
cin >> i;
```

FuncGetNumberOf NewLines.cpp

```
include <iostream>
#include <fstream>
# include <assert.h>
using std::cout;

namespace Const
 {
   const char* FILE_NAME = "file.txt";
 }
size_t GetNumberOfSymbol(std::ifstream& ifs, char ch)
 {
   size_t count=0;
   size_t currPos = ifs.tellg();
   ifs.seekg(0, std::ios::beg);
   while (true)
   {
     char _ch= ifs.get();
     if(ifs.eof())
       break;
     if(ch==_ch)
       count++;
   }
   ifs.seek(0, currPos);
   return count;
}
```

```
size_t GetNumberOfSymbol(const char* fileName, char ch)
{
    std::ifstream ifs(fileName, std::ios::in);
    assert(ifs.is_open());
    size_t count = GetNumberOfSymbol(ifs, ch));
    ifs.close();
    return count;
}

size_t GetNumberOfNewLines(const char* fileName)
{
    return GetNumberOfSymbol(fileName, '\n')+1;
}

int main()
{
    cout << GetNumberOfNewLines(Const::FILE_NAME);
}
```

Бъдещ файл

ifstream ifs(<име_на_файл>, <режим_на_работа>);

- Синхронизация
  - като ostream
- позициониране
  - аналогично на ostream, но имаме get указател, който сочи към следващия елемент за четене
  - .tellg();
  - .seekg(size_t idx)
  - .seekg(<Offset>, <dest>)
  - .ignore(size, <symb>) - прескача size на брой символи или докато няма разделител: no def size=1
  - .symb = eof();

Режими на работа

- ofstream/ifstream/fstream stream(<името файл>, <режим>)
- режима на работа е число
- в паметта:

0/1	0/1	0/1	0/1	0/1	0/1	0/1	0/1

- в зависимост от това дали битовете са вдигнати или не потока работи в режим
- ако имаме няколко режима за работа на работа искаме да ги обединяваме, чрез "|" - подходящо или

Опции:

Срещу всяко число съответства име на режима в ios namespace-а

- ios::in (1) - потока е отворен за четене/извличане
- ios::out (2) - потока е отворен за писане/вмъкване. Ако файлът съществува, то съдържанието му се трие
- ios::ate (4) - отваря потока, поставя get показалеца, позволява местене/позициониране
- ios::app (8) - отваря потока, поставя put показалеца, не позволява достъп до вече съществуващото съдържание
- ios::trunc (16) - по default ако файла съществува изтрива
- ios::binary (32) - отваря потока в двоичен режим на работа.
- ios::_NoCreate (64) - отваря за вмъкване, само ако съществува
- ios::_Noreplace (128) - отваря за вмъкване, само ако не съществува.

Състояние на потока
- поток има 3 бита, които показват състоянието му

- bad() - има загуба на информация → badbit (операция I/O не е била успешна)
- fail() - последната I/O операция е невалидна → badbit / failbit
- good() дали всички са сбъднати
- eof() - дали сме достигнали до края на потока → eofbit
- clear() - автоматично сбъдва всички

Func GetFileSize.cpp

```
#include <iostream>
#include <fstream>
using std::cout;
namespace Const
{
    const char* FILE_NAME = "file.txt";
}
size_t getFileLength(std::ifstream& ifs)
{
    size_t currPos = ifs.tellg();
    ifs.seekg(0, std::ios::end);
    size_t res = ifs.tellg();
    ifs.seekg(0, currPos);
    return res;
}
size_t getFileLength(const char* fileName)
{
    ifstream ifs(fileName, std::ios::in);
    assert(ifs.is_open());
    size_t res = getFileLenght(ifs);
    ifs.close();
    return res;
}
int main()
{
    cout << getFileLength(Const::FILE_NAME);
}
```