Файловые потоки вывода

```
//<fstream>

class ofstream : public ostream{
        ofstream ();
        ofstream (const char *, openmode = out | trunc);
        void open (const char *, openmode = out | trunc);
        void close ();
        bool is_open () const;
};
```

Файловые потоки ввода

```
//<fstream>

class ifstream : public istream{
        ofstream ();
        ofstream (const char *, openmode = in);
        void open (const char *, openmode = in);
        void close ();
        bool is_open () const;
};
```

Режимы открытия

```
base_ios{
public:
        typedef … openmode;
        static const openmode
                ate,            //?
                app,            //добавление в конец
                binary,         //бинарный поток
                in,             //чтение
                out,            //запись
                trunc;          //пересоздание (файла)
};
```

Файловые двунаправленные потоки

```
//<fstream>

class fstream : public iostream{
        ofstream ();
        ofstream (const char *, openmode = in|out);
        void open (const char *, openmode = in|out);
        void close ();
        bool is_open () const;
};
```

Строковые потоки

```
//<sstream>

class stringstream : public iostream{
        ofstream ();
        ofstream (string&, openmode = in|out);
        void open (string&, openmode = in|out);
        void close ();
        bool is_open () const;
};
```

Манипуляторы

```
#include <iostream>
#include <iomanip>
using namespace std;
//...
int a = 64;

printf ("%d\n%o\n%x\n", a, a, a);
fflush (STDOUT);
//-----------------------------------------------
cout << a << endl << oct << a << endl << hex <<
        endl << dec << flush;
```

Реализация манипуляторов

```
class ostream{
//...
        ostream& operator << (ostream& (*f)(ostream&)){
                return f(*this);
        }
//...
};

ostream& flush (ostream& s){
        return s.flush ();
}
```

Манипуляторы с параметрами

```
int a = 64;

//вывод в системе счисления с основанием 6
cout << setw (6) << a << endl << flush;
```

setw (6) – это не указатель на функцию

Реализация манипуляторов с параметрами (способ #1)

```
class setw{
public:
        int param;
        setw (int _param);
        ostream& f (ostream& s, int param){
                return s.setw (param);
        }
};

ostream& operator << (ostream& s, setw& m){
        return m.f(s, m.param);
}
```

Реализация манипуляторов с параметрами (способ #2)

```
class Manip{
public:
        virtual ostream& operator () (ostream &) = 0;
        virtual ~manip () {}
};
ostream& operator << (ostream& s, Manip& m){
        return m(s);
}
//...
class setw : public Manip{
private:
        int param;
public:
        setw (int _param);
        virtual ostream& operator () (ostream& s) {
                return s.setw(param);}
};
```

Стандартные манипуляторы

#include <iomanip>

dec, hex, oct – вывод в различных системах
setbase(int b) – вывод в системе с основанием b

endl – конец строки (\n)
flush – сброс буфера

scientific, fixed – %e, %f
setprecision (int n) – n символов после точки

Шаблонный полиморфизм

Шаблонный полиморфизм – параметризация класса, функции или метода, при этом в зависимости от параметра изменяется поведение класса, функции или метода.

Параметрами шаблонов могут быть как объекты, так и классы (типы).

Функция сортировки

Задача:

- Необходима функция, которая бы реализовывала некоторый алгоритм сортировки.
- Алгоритм сортировки не зависит от природы сортируемых объектов.

Решения:

1. void sort (void *, int len, int size,
   bool (*less)(viod *, void *));
2. template <typename T>
   void sort (T *, int len);

Реализация списка

Задача:

- Необходим класс списка, который мог бы хранить произвольные типы данных

Решения:

1. typedef Complex ListLoad;
   class ListELement {private: ListLoad * elem; /*...*/};

2. class ListElement {private: void * elem; /*...*/};

3. template <typename T>
   class ListElement {private: T * elem; /*...*/};

Шаблоны функций

```
template <typename T> void sort (T *, int len); //1

template <typename T, int len> void sort (T *); //2

int main (){
        Complex arr [100];
        sort <Complex> (arr, 100);          //1
        sort (arr, 100);                    //1

        sort <Complex, 100>(arr);           //2
        return 0;
}
```