Классы и объекты

Константы в описании класса

```
class MyClass{
private:
const int N=5;  //только int
const const int MAX=999;
/* создается для каждого объекта, поэтому не может
   использоваться для определения размеров
   массива но может использоваться в методах, хотя
   это накладно*/
int arr[N];     //ошибка
 public:
bool  wellFormed(){   return x<MAX; } // правильно
…
};
```

Статические константы

```
class MyClass{
static const int N=5;
static const int MAX=999;
// одна константа для всех объектов
int arr1[N];
 public:
bool  wellFormed(){   return x<MAX; }

};
```

```
class MyClass{
int arr1[N];
 public:
static const int N=5;
static const int MAX=999;
// открытые, но принадлежат не объектам, а
   классу
};

cout<< “max value=“<<MyClass::MAX<<endl;
```

Статические методы

```
class MyClass{
static const int N=5;
// одна константа для всех объектов
int arr1[N];
 public:
  int  sizeOfArr(){ return N; }
//независимо от объекта возвращает одинаковые
   значения
};
MyClass x;
cout<<“ArraySize=“<<x.sizeOfArr()<<endl;
```

Статические методы

```
class MyClass{
static const int N=5;
// одна константа для всех объектов
int arr1[N];
 public:
 static int  sizeOfArr() {return N;}
//принадлежит классу, а не объектам
//работает только со статическими элементами
   класса
};

cout<<“ArraySize=“<<MyClass::sizeOfArr()<<endl;
```

Константные методы

- Не изменяют *this
- Вернемся к примеру Timer

```
int   Timer :: period (Timer x)  // почему плохо?
{
   return timeInSec() - x.timeInSec();

}
```

Константные методы

- Пусть нам нужна не копия, а ссылка на больший

```
int   Timer :: period (const Timer& x)
{
   return timeInSec() –
            x.timeInSec(); //ошибка , может x изменится?

}
```

В результате преобразования теряются квалификаторы

Константные методы

- мы знаем, что timeInSec() не меняет вызвавший его объект

```
int    timeInSec() const ;
```

Композиция классов

- Классы могут содержать в качестве полей, объекты (указатели на объекты) других классов или структуры дынных, содержащие объекты (указатели на объекты) других классов.
- Такой способ построения сложных классов называется ***композицией***

Примеры

```
class Point {
   private:
         int x,y;
   public:
. . .};
class T {
  private:
     Point a, b, c;
  public :
. . .
}
```

Классы, использующие динамическую память

```
class Array {
private:
  int n;
  int *A;
public:
  Array();                          //конструктор без параметров
  Array(int _n, int x=0);           //конструктор с параметрами
  Array(const Array &B);            //конструктор копии
  int max( );                       // максимальный элемент
  void addInt (int x);
  void addArray (const  Array&  x);
  bool equal (const  Array&  x);
~Array ( );
};
```

Конструктор без параметров

```
Array :: Array( ) {
  n=10;
  A=new int[n]; // выделяем память
  for (int i=0; i<n; ++i)
    A[i]=0;
}
```

Конструктор с параметрами

```
Array::Array(int _n, int x) { // синтаксис!!!
  n=_n;
  A=new int[n];
  for (int i=0; i<n; ++i)
    A[i]=x;
}
```

Конструктор копии

```
Array::Array (const Array &x) {
  n=x.n;
  A=new int[n];
  for (int i=0; i<n; ++i)
    A[i]=x.A[i];
}
```

При отсутствии, будет использован конструктор копии по умолчанию, выполняющий поверхностное копирование. Память выделена не будет, а скопируется указатель на массив в оригинале.

Дектсруктор

```
Array::~Array( ){
  delete[] A;
}
```

Максимальный элемент

```
int Array::max( ) {
int maxi = A[0];
  for (int i=1; i<n; ++i)
    if (A[i]>maxi)
      maxi=A[i];
  return maxi;
}
```

Добавить целое

```
void Array::addInt ( int x) {
   for (int i=0; i<n; ++i)
   A[i]=A[i]+x;
}
```

Добавить другой массив

```
void Array::addArray (const Array &x) {
    if ( n!= x.n ) return;
    for ( int i=0; i<n; ++i)
    A[i]=A[i]+x.A[i];

}
```

Сравнить на равенство

```
bool Array::equal (const  Array& x){
    if ( ! this || !(&x)) return false;
    if (this == &x) return true;
    if (n!= x.n) return false;
    for (int i=0; i<n; ++i)
            if (A[i]!= x.A[i]) return false;
    return true;
}
```

Дружественные функции

- класс разрешает функции, а не методу иметь доступ к закрытым частям класса

```
class Array {
private:
  . . .
public:
 . . .
friend void print(const Array& x);
//хотя бы один параметр – объект класса
};
```

```
void print (const Array& x){
   for (int i=0; i<x.n;  ++i) cout<< x.A[i]<<"   ";
    cout<<endl;
 }
```

вызов – обычная функция

```
Array a1;
  print(a1);
```