Объектно-ориентированное программирование
Лекция 27

Пример

```
#include <iostrem.h>
// Создаем класс queue   (очередь)
class queue {
      int q[100];
      int sloc, rloc;
public:
      void init(void);
      void qput(int m);
      int qget(void);
};
```

```
void queue::init(void)
{
    rloc=sloc=0;
}
```

Конструкторы и деструкторы

- Так как необходимость инициализации объектов является общим требованием, то язык C++ предоставляет возможность делать это автоматически при создании объекта, т.е. при объявлении переменной.

Конструктор

- Конструктор - это специальная функция, являющаяся членом класса и имеющая то же самое имя, что и класс.

```
class queue;
    int q[100];
    int sloc, rloc;
    public:
    queue(void);      // конструктор класса queue
    void qput(int i);
    int qget(void);
}
```

Конструктор

```
queue::queue(void) // конструктор класса
   queue
{
    sloc=rloc=0;
    cout << " queue инициализирована \n";
}
```

- Конструктор исполняется в тот момент, когда создается объект. Для локального объекта это будет происходить при каждом объявлении переменной данного класса.

Деструктор

- Противоположностью конструктору является функция-деструктор (destructor). Во многих случаях необходимо, чтобы были произведены какие-либо действия перед тем как объект уничтожить.

Деструктор

- Деструктор имеет такое же имя как и конструктор, но перед ним ставится знак тильды (~).

```
#include <iostream.h>
#include <conio.h>
// объявление класса
class queue {
     int q[100];
     int sloc,rloc;
   public:
     queue(void);   // конструктор
     void qput(int i);
     int qget(void);
     ~queue(void);  // деструктор
};
```

```
// определение функции конструктор
queue::queue(void)
{
	sloc=rloc=0;
	cout << " очередь инициализирована\n";
}
// определение функции-деструктора
queue:: ~queue(void)
{
	cout << "очередь разрушена\n";
}
void queue::qput(int i)
{
	if (sloc==100){
		cout <<"очередь полна";
		return;
	}
	sloc++; q[sloc]=i;
}
int queue::qget(void)
{
	if (rloc==sloc){
		cout<<"очередь пуста\n";
		return 0;
	}
	return q[++rloc];
}
```

```
main()
{
    clrscr();
    queue a, b; // объявление двух объектов типа queue
    a.qput(10);
    b.qput(19) ;
    a.qput(20);
    b.qput(1);
    cout<<a.qget()<<" ";
    cout<<a.qget()<<" ";
    cout<<b.qget()<<" ";
    cout<<b.qget()<<"\n";
    cout<<b.qget()<<"\n";
    return 0;
}
```

Наследование

- Наследование - одна из главных черт объектно-ориентированного программирования.

Пример

```
class transport{
        int kol;
        int puss;
    public:
        void set_kol(int num);
        int get_kol(void);
        void set_pass(int num);
        int get_pass(void);
};
```

```
void transport::set_kol (int num)
{
    kol=num;
}
// объявление функции члена класса transport
int transport::get_kol(void)
{
    return kol;
} // объявление функции члена класса transport
void transport::set_pass(int num)
{
    pass=num;
}  // объявление функции члена класса transport
int transport::get_pass(void)
{
    return pass;
}
```

Наследование

- Это общее определение дорожного транспорта может быть использовано для того, чтобы определить конкретный объект. Например, определить класс, называемый trakt, с использованием класса transport

```
class trakt: public transport{
        int gruz;
    public:
        void set_gruz(int size);
        int get_gruz(void);
        void show(void);
}
```

```
void trakt::set_gruz(int num)
{
    gruz=num;
}
int trakt:: get_gruz(void)
{
    return gruz;
}
void trakt:: show(void)
{
    cout <<" колеса: "<<get_kol()<<"\n" ;
    cout <<" пассажиры: "<<get_pass( )<<"\n";
    cout <<" вес груза:  "<<gruz<<"\n";
}
```

Опция public

- Эта опция означает, что все элементы типа public предка будут типа protected для класса, который наследует его.

Доступ в базовом классе	Спецификатор доступа перед базовым классом	Доступ в производном классе	
		struct	class
public	отсутствует	public	private
protected	отсутствует	public	private
private	отсутствует	недоступны	недоступны
public	public	public	public
protected	public	protected	protected
private	public	недоступны	недоступны
public	protected	protected	protected
protected	protected	protected	protected
private	protected	недоступны	недоступны
private	public	недоступны	недоступны
public	private	private	private
protected	private	private	private
private	private	недоступны	недоступны

```
enum type {car, van, bus};
class avto:public transport{
        type car_type;
    public:
        void set_type(type t);
        type get_type(void);
        void show(void);
};//конец объяв-ия класса auto - наследника класса transport
```

```
// avto
void avto::set_type(type t)
{
	car_type=t;
}
type avto::get_type(void)
{
	return car_type;
}
void avto::show(void)
{
	cout<< " колеса: "<<get_kol()<<"\n";
	cout<< " пассажиры: "<<get_pass()<<"\n"; cout<< " type:";
	switch (get_type()){
		case van: cout<<"van \n"; break;
		case car: cout<<"car \n";  break;
		case bus: cout<<"bus \n";  break;
	}
} // конец объявления функции show члена класса avto
```

```
void main()
{
    trakt t1,t2; avto c;
    t1.set_kol(18); t1.set_pass(2); t1.set_gruz(3200);
    t2.set_kol(8);  t2.set_pass(3); t2.set_gruz(1000);
    t1.show(); t2.show();
    c.set_kol(4); c.set_pass(6); c.set_type(van);
    c.show();
}
```