Объектно-ориентированное программирование

Лекция 16

Объектно-ориентированное программирование

- Объектно-ориентированное программирование - это один из способов подхода к программированию.

Объектно-ориентированное программирование

- Класс – это определенный пользователем тип данных. Он содержит описания типов членов класса и набор операций для манипулирования такими объектами.
- Объект - это логическая единица, которая содержит данные и правила обработки этих данных. Объект является экземпляром класса.

Класс

- В языке C++ для того, чтобы определить объект (object), надо сначала определить его **форму** (**шаблон**) с помощью ключевого слова class. Понятие класса напоминает понятие структуры.

Класс

```
class имя класса{
    описание полей класса
};
```

Спецификаторы доступа

- **Public**: этот спецификатор делает поля и методы класса общедоступными.
- **Private**: данный спецификатор информирует о том, что следующие за ним компоненты класса доступны лишь только внутри класса.
- **Protected**: компоненты описанные с таким спецификатором могут быть доступны лишь функциям класса или классов-наследников

Пример

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

Объявление объектов

queue int1,int2;.

- классы это не **объекты**, а **шаблоны** для создания объектов.

Объявление объектов

- Можно создать объект и во время объявления класса, поместив имя объекта после закрывающейся фигурной скобки, как это делали со структурами. То есть в C++ класс создает новый тип данных, который может использоваться для создания объектом этого типа.
- Объявление класса в общем виде следующее:

```
class имя класса {
    приватные данные и функции
public:
    общие данные и функции
} список объектов;
```

Объявление функций вне класса

- Когда же требуется создать реальную функцию-член класса, необходимо указать к какому классу относится эта функция. Например, можно определить функцию init() класса queue следующим образом.

```
void queue::init(int i){
    if (i==10) {
      printf("Введите число");
      scanf("%d",&i);
      return;
    }
    printf(" Работает функция init()“);
}
```

Вызов функций

- Чтобы вызвать функцию-член класса в той части программы, которая не является частью класса, надо использовать имя объекта и оператор точка (.), как делается с элементами структур. Для вызова этой функции нужно написать:

  a.init();

- Следует помнить, что a и b два разных объекта. Хотя содержание функции init() в объектах одинаковое, сами функции разные.

  a.init();

  b.init();

Вызов функций

- С другой стороны функция член класса может вызвать другую функцию член того же класса непосредственно, не используя операцию точка (.). Операцию точка надо использовать только тогда, когда функция член класса вызывается в части программы, не принадлежащей классу.

```
{
if (sloc==rloc) {
cout<<" Очередь пуста ";
return 0;
}
rloc++;
return q[rloc];
}
void queue::qput(int i){
if(sloc==100){
cout<<"Очередь полна";
return;
}
q[++sloc]=i;
}
void queue::init(void) //объявление функиии членов класса
{ sloc=rloc=0; }
int queue::qget(void)
{
if (sloc==rloc) {
cout<<" Очередь пуста ";
return 0;
}
rloc++;
return q[rloc];
}
void queue::qput(int i){
if (sloc==100){
cout<<"Очередь полна";
return;
}
q[++sloc]=i;
}
main(void)
{ clrscr();
queue a,b; // Созданы два бъекта класса queue
a.init(); b.init();
a.qput(10); b.qput(9); a.qput(20); b.qput(1);
cout<<a.qget()<< " ";
cout << a.qget()<< " ";
cout << b.qget()<< " ";
cout<<b.qget()<< "\n";
return 0;
}
```


```
#include <iostream.h>
#include <conio.h>
//введим класс queue
class queue{
int q[100];
int sloc, rloc;
void qput(int i);
int qget(void);
void init(void);
};
void queue::init(void) //объявление функиии членов класса
{ sloc=rloc=0; }
int queue::qget(void)
{
if (sloc==rloc) {
cout<<" Очередь пуста ";
return 0;
}
rloc++;
return q[rloc];
}
void queue::qput(int i){
if (sloc==100){
cout<<"Очередь полна";
return;
}
q[++sloc]=i;
}
main(void)
{ clrscr();
queue a,b; // Созданы два бъекта класса queue
a.init(); b.init();
a.qput(10); b.qput(9); a.qput(20); b.qput(1);
cout<<a.qget()<< " ";
cout << a.qget()<< " ";
cout << b.qget()<< " ";
cout<<b.qget()<< "\n";
return 0;
}
```

ООП

- Объектно-ориентированные языки обладают тремя важнейшими характеристиками:
  - **инкапсуляция** (encapsulation),
  - **наследование** (inharitance) и
  - **полиморфизм** (polymorphism).

Инкапсуляция

- Понятие инкапсуляции означает, что в качестве единицы целого рассматривается объединение некоторой структуры данных и некоторой группы функций.
- Сущность этого принципа заключается в сокрытии данных. Для этого используют спецификаторы доступа.

Наследование

- Наследование позволяет одним объектам приобретать атрибуты и свойства других объектов. Наследование поддерживает иерархическую классификацию.

Наследование

class A: [спецификатор доступа] B{

.....

}

```
class B { protected: int t;
          public: char u;
        };
class E: B ( ... );
struct S: B ( ... );
```

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*

Полиморфизм

- Буквально означает многообразие форм. Он заключается в способности объекта динамически менять свойства и особенности поведения.
- Одно и то же имя может использоваться для логически связанных, но разных целей. Т.е. имя определяет класс действий, которые в зависимости от типа данных могут существенно отличаться.

Полиморфизм

```
class A{
      ……
      public:
        …
        virtual void Sum();
        …
};
class B:public A{
      ……
      public:
        ……
        virtual void Sum();
        ……
void main
      A *a = new B();
      a->Sum();
}
```