Тема 2.6 Конструкторы и деструкторы

Поскольку инициализация объектов – очень распространенное требование, в языке C++ предусмотрен особый механизм, позволяющий инициализировать объекты в момент их создания. Эта инициализация осуществляется конструктором.

Конструктор – это особая функция, являющаяся членом класса. Ее имя должно совпадать с именем класса. Для объявления конструктора по умолчанию пишется следующий код, где представлен конструктор без аргументов, то есть никаких аргументов он не принимает:

```
class Account {
private:
   char *_name;
   unsigned int _acct_nmbr;
   double _balance;
public:
   Account(); // конструктор
};
```

Для объявления конструктора вне класса используется спецификатор доступа "::", как показано на примере:

```
Account::Account(int a, double b){}
```

Противоположностью конструктора является *деструктор*. Во многих ситуациях объект должен выполнить некоторое действие или действия, которые уничтожат его. При разрушении объекта автоматически вызывается его деструктор. Деструктор объявляется так же как и конструктор, только перед его названием используется символ "~". Пример:

```
public:
   Account(){cout<<"Конструктор";};
   ~ Account(){cout<<"Деструктор";};
```

Деструкторы можно вызывать явным образом, указав полностью уточненное имя, например:

```
Man *m; ...
m -> ~Man();
```

Единственное синтаксическое ограничение, налагаемое на конструктор и деструктор, состоит в том, что они не должны иметь тип возвращаемого значения, даже void. Поэтому следующие объявления ошибочны:

```
// ошибки: у конструктора не может быть типа возвращаемого
значения
void     Account::Account() { ... }
Account* Account::Account( const char *pc ) { ... }
```

C++ позволяет указывать значения по умолчанию для параметров функции. Если пользователь не указывает каких-либо параметров, функция будет использовать значения по умолчанию. Конструктор не является исключением; ваша программа может указать для него значения по умолчанию так же, как и для любой другой функции. Например, следующий конструктор Account использует по умолчанию значение баланса равным 10000.0, если программа не указывает оклад при создании объекта. Однако программа должна указать имя и номер. На данном примере уже показан конструктор с аргументами, то есть конструктору передаются какие-либо параметры и он производит необходимые действия с ними, инициализирует поля объекта:

```
public:
   // конструктор
   Account(char *name, int acct, double balance = 10000.00)
   {
     strcpy(_name, name);
     _acct_nmbr = acct;
     if (balance < 50000.0)
     _balance = balance;
   };
```

Доступность конструктора и деструктора определяется тем, в какой секции класса они объявлены. Мы можем ограничить или явно запретить некоторые формы создания объектов, если поместим соответствующий конструктор или деструтор в неоткрытую секцию. В примере ниже конструктор по умолчанию класса Account объявлен закрытым, а с двумя параметрами – открытым, так же показывается, что может создаваться несколько конструкторов, отличаются только параметры:

```
class Account {
   friend class vector< Account >;
public:
   Account( const char*, double = 0.0 );
   // ...
private:
   Account();
   // ...
};
```

Обычная программа сможет теперь определять объекты класса Account, лишь указав как имя владельца счета, так и начальный баланс. Однако функции-члены Account и дружественный ему класс могут создавать объекты, пользуясь любым конструктором.

Конструкторы, не являющиеся открытыми, в реальных программах C++ чаще всего используются для:

– предотвращения копирования одного объекта в другой объект того же класса;

– указания на то, что конструктор должен вызываться только в случае, когда данный класс выступает в роли базового в иерархии наследования, а не для создания объектов, которыми программа может манипулировать напрямую.

При создании объекта, если у класса имеется конструктор с одним параметром, или у всех параметров, кроме одного, имеются значения по умолчанию, тип параметра можно неявно преобразовать в тип класса. Например, если у класса Box имеется конструктор, подобный следующему: `Box(int size)`, то возможно инициализировать объект Box следующим образом: `Box b = 42;`

Если же конструктор имеет несколько параметров без значений по умолчанию, то в основном коде программы передаются значения всем параметрам в явном виде, или посредством переменных.

```
class Account {
private:
   char *_name;
   unsigned int _acct_nmbr;
   double _balance;
public:
   Account(char *name, unsigned int acct,double balance)
  {
       strcpy(_name,name);
       _acct_nmbr=acct;
       _balance=balance;
  };
};
//основная программа
Account A1("Name1", 40, 123.321)
```

Инициализация объекта другим объектом того же класса называется почленной инициализацией по умолчанию. Копирование одного объекта в другой выполняется путем последовательного копирования каждого нестатического члена. Проектировщик класса может изменить такое поведение, предоставив специальный копирующий конструктор. Если он определен, то вызывается всякий раз, когда один объект инициализируется другим объектом того же класса.

Часто почленная инициализация не обеспечивает корректного поведения класса. Поэтому мы явно определяем копирующий конструктор. В нашем классе Account это необходимо, иначе два объекта будут иметь одинаковые номера счетов, что запрещено спецификацией класса.

Копирующий конструктор принимает в качестве формального параметра ссылку на объект класса. Вот его реализация:

```
Account(Account &a){} //конструктор копирования

//основная программа
Account A1;
Account A2=A1 //создание объекта A2 идентичного A1
```