Лабораторная работа № 4

Тема: Наследование в C++

Цель работы. Изучить принципы наследования и его использования в классах

Теоретическая часть

Наследование служит для создания нового класса на основе одного или нескольких уже существующих классов. Новый класс называют производным классом (или классом-потомком). Класс, элементы которого наследуются, называется базовым классом (родительским классом или классом-предком) для своего производного класса. При наследовании все характеристики (свойства, методы) класса-родителя присваиваются классу-потомку.

Наследование дает возможность некоторые общие черты поведения классов абстрагировать в одном базовом классе. Производные классы, наследуя это общее поведение, могут его несколько видоизменять, переопределяя некоторые функции-члены базового класса, или дополнять, вводя новые данные-члены и функции-члены. Поэтому определение производного класса значительно сокращается, поскольку нужно определить только отличающие его от производных классов черты поведения.

Синтаксис объявления производного класса имеет следующий вид:

```
//Базовый класс 1
class Base1
{
//.............
};

//Базовый класс 2
class Base2
{
//.............
};

//Базовый класс N
class BaseN
{
//.............
};

//Производный класс
class Derived:<спецификатор_доступа> Base1,
              <спецификатор_доступа> Base2,
              ...,
              <спецификатор_доступа> BaseN,
{
// ..........
};
```

Возможными спецификаторами доступа являются: public, protected или private. Этот параметр не является обязательным и по умолчанию принимает значение private для классов и public для структур. При наследовании спецификатор доступа определяет уровень доступа к элементам базового класса, который получают элементы производного класса.

Спецификатор наследуемого доступа устанавливает тот уровень доступа, до

которого понижается уровень доступа к членам, установленный в базовом классе. Если спецификатор наследуемого доступа установлен как public, то public-члены базового класса будут public-членами в производном классе, protected-члены базового класса будут protected-членами в производном классе, а private-члены базового класса будут недоступны. Если спецификатор наследуемого доступа установлен как protected, то в производном классе происходит понижение доступности до этого уровня (private-члены остаются недоступными), а в случае применения спецификатора private — понижение до уровня private (private-члены все также недоступны).

Однако можно сделать некоторые из членов базового класса открытыми в производном классе, объявив их в секции public производного класса. Например,

Пример 1

```
#include <cstdlib>
#include <iostream>

using namespace std;

class Base
{
   int x,y;
public:
   int GetX(){cout << "Method GetX()\n"; return x;}
   int GetY(){return y;}
};

class Derived : private Base
{
public:
   Base::GetX;
};

int main(int argc, char *argv[])
{
   int x;
   Derived ob;
   x = ob.GetX();
   cout << "Press the enter key to continue ...";
   cin.get();
   return EXIT_SUCCESS;
}
```

При таком наследовании открытые и защищенные члены базового класса будут доступны только для членов данного производного класса, и все члены базового класса, кроме явно объявленных в разделе public или protected, будут закрытыми для следующих производных классов. Этот прием позволяет отсечь доступ к членам базового класса при построении иерархии классов с отношением «родитель-потомок».

При любом способе наследования в производном классе доступны только открытые (public-) и защищенные (protected-) члены базового класса (хотя наследуются все члены базового класса). Иначе говоря, закрытые члены базового класса остаются закрытыми, независимо от того, как этот класс наследуется.

Если у производного класса имеется всего один базовый класс, то говорят о простом (одиночном) наследовании. Ниже приведен **пример 2**, в котором используется простое наследование. Класс OutCoord является производным от базового класса Coord. К членам класса Coord он добавляет один приватный атрибут, три открытые функции и конструктор, так как конструкторы не наследуются. Производный класс либо должен объявить свой конструктор, либо предоставить возможность компилятору сгенерировать

конструктор по умолчанию.

Пример 2

```
#include <cstdlib>
#include <iostream>

using namespace std;

class Coord
{
   int x, y;
   public:
      Coord(int _x, int _y) {x = _x; y = _y;}
      Coord(){x = 0; y = 0;}
      int GetX(){return x;}
      int GetY(){return y;}
      void SetX(int _x){x = _x;}
      void SetY(int _y){y = _y;}
};

class OutCoord: public Coord
{
   int z;
   public:
      OutCoord(int _x, int _y, int _z) : Coord(_x, _y){z = _z;}
      void ShowX(){cout << GetX() << " ";}
      void ShowY(){cout << GetY() << " ";}
      void ShowZ(){cout << z << " ";}
};

main()
{
   OutCoord* ptr;
   ptr = new OutCoord(10,20,30);
   ptr->ShowX();
   ptr->ShowY();
   ptr->ShowZ();
   cout << "\n";
   delete ptr;
   system("pause");
   return 0;
}
```

Рассмотрим принципы, используемые программистом, при построении конструктора производного класса. Поскольку производный класс должен унаследовать все члены родительского, при построении объекта своего класса он должен обеспечить инициализацию унаследованных данных-членов, причем она должна быть выполнена до инициализации данных-членов производного класса, так как последние могут использовать значения первых. Поэтому для построения конструктора производного класса применяется следующая конструкция:

<конструктор_произв_класса>(<список параметров>) :
<конструктор_базового_класса> (<список_аргументов>) {<тело__конструктора>}

Используется список инициализации элементов, в котором указывается конструктор базового класса. Часть параметров, переданных конструктору производного класса, обычно используется в качестве аргументов конструктора базового класса. В теле

конструктора производного класса выполняется инициализация данных-членов, принадлежащих этому классу.

В приведенном примере эта конструкция используется для создания конструктора класса OutCoord и конструктор имеет вид:

```
OutCoord(int _x, int _y, int _z) : Coord(_x, _y){z = _z;}
```

В теле конструктора выполняется инициализация только одного атрибута, который относится к собственным данным-членам класса OutCoord. Остальные параметры конструктора производного класса просто передаются конструктору базового класса для осуществления инициализации. Аргументы передаются конструктору производного класса в операторе new, который вызывает этот конструктор. Затем программа вызывает функции-члены ShowX(), ShowY() и ShowZ() объекта (представителя производного класса), которые выводят на экран заданные значения координат. Оператор delete вызывает удаление объекта, для чего он неявно вызывает деструктор производного класса.

Если в производном классе используется конструктор по-умолчанию, то при создании экземпляра производного класса автоматически вызывается конструктор базового класса. Конструктор базового класса вызывается компилятором, только когда конструируется объект производного класса. Конструктор, в отличие от других унаследованных функций, вызвать явно нельзя.

В отношении деструкторов производных классов также действуют определенные правила. Деструктор производного класса, должен выполняться раньше деструктора базового класса (иначе деструктор базового класса мог бы разрушить данные-члены, которые используются и в производном классе). Когда деструктор производного класса выполнит свою часть работы по уничтожению объекта, вызывается деструктор базового класса. Вся работа по организации соответствующего вызова возлагается на компилятор, программист не должен заботиться об этом.

Следующий пример демонстрирует работу конструкторов и деструкторов базового и производного классов:

```
#include <cstdlib>
#include <iostream>

using namespace std;

class Base {
   public:
      Base() { cout << "Constructor " << "of Base-class" << "\n"; }
      ~Base(){ cout << "Destructor " << "of Base-class" << "\n"; }
};

class Derived: public Base {
   public:
      Derived() { cout << "Constructor of Derived-class" << "\n"; }
      ~Derived() { cout << "Destructor of Derived-class" << "\n"; }
};

int main(int argc, char *argv[])
{
   {Derived ob; }
   system("pause");
   return EXIT_SUCCESS;
}
```

Задания к работе

1. Изучите примеры, приведенные в теоретической части.
2. Используя теоретический материал о наследовании решите следующую задачу. Пусть создается некоторая игра, в которой будут задействованы различные герои. Напишите два простых класса. Первый класс служит для создания объектов типа «фермер», который характеризуется положением на игровой карте через задание координат X и Y, скоростью движения V, конечной точкой движения XK и YK, а также уровнем здоровья H. Второй класс служит для создания объектов типа «солдат», который характеризуется теми же параметрами, что и «фермер», но у него есть еще два дополнительных параметра — боевая сила F и уровень защиты S. Для создания второго класса используйте принципы наследования. Классы должны иметь конструкторы/деструкторы, а также функции GET и SET, для соответствующих параметров.
3. Составьте отчет, в который включите примеры теоретической части и скриншоты результатов их работы, а также код разработанных классов из пункта 2 и примеры их работы.