ТЕМА 2.2 ПАКЕТИРОВАНИЕ, НАСЛЕДОВАНИЕ И ПОЛИМОРФИЗМ

НАСЛЕДОВАНИЕ В C++

НАСЛЕДОВАНИЕ В C++

1. Определение наследования
2. Модификаторы доступа при наследование и их роль
3. Множественное наследование
4. Виртуальные базовые классы

1. ОПРЕДЕЛЕНИЕ НАСЛЕДОВАНИЯ

Наследование — механизм языка, позволяющий описать новый класс на основе уже существующего (родительского, базового) класса или интерфейса. Потомок может добавить собственные методы и свойства, а также пользоваться родительскими методами и свойствами. Позволяет строить иерархии. Является одним из основных принципов объектно-ориентированного программирования.

Наследование объекта:

`class имя_класса_наследника: модификатор_доступа имя_базового_класса {…}`

Пример наследования:

```
using namespace std;

class Class1{
protected:
     int znachenie;
public:
    Class1() {znachenie = 0;}
    Class1(int   vvod){znachenie   =
vvod;}
    void    vivod_znach(){cout    <<
znachenie << endl;}
};
class Class2 : public Class1{
public:
    Class2() : Class1(){}   //
конструктор
    Class2(int vvod_2) : Class1
(vvod_2){}
void
ZnachSqr(){znachenie*=znachenie;}
```

```
int main(){
    Class1 object1(3);
cout << "znachenie object1 = ";
    object1.vivod_znach();
    Class2 object2(4);
    cout << "znachenie object2 = ";
    object2.vivod_znach();
    object2.ZnachSqr();
    cout << "kvadrat znacheniya object
= ";
    object2.vivod_znach();

    //object1.ZnachSqr();  //  базовый
класс  не  имеет  доступа  к  методам
производного класса

    return 0;
}
```

Базовый класс – **Class1**; Он разрешил доступ к свой переменной **znachenie** своим наследникам.

Программа показывает различие между методами объектов классов **Class1** и **Class2** и невидимость методов наследника базовому классу.

2. МОДИФИКАТОРЫ ДОСТУПА ПРИ НАСЛЕДОВАНИИ И ИХ РОЛЬ

Класс – это тип данных, объединяющий данные и методы их обработки. Инкапсуляция позволяет скрыть информацию, к которой не предусмотрен прямой доступ из других классов. В языке C++ предусмотрено несколько модификаторов доступа, которые определяют кто имеет право использовать следующие за ними объявления членов класса и некоторых других элементов языка:

- **public** - означает, что следующие за ним определения доступны всем.
- **private** - делает следующие за ним определения доступными только внутри класса
- **protected** - защищённые методы или переменные доступны только внутри класса, где они были объявлены и из его производных классов.

PUBLIC - НАСЛЕДОВАНИЕ

При наследовании **public** в класс-потомок передаются все поля в таком виде в котором они записаны в родителе. **private** поле тоже туда передается, но напрямую наследник ничего с ней сделать не может.

Используя **public** наследование мы передаем потомкам всё что есть в основном классе в таком виде, как и записано в основном классе. Получаем клон основного класса. Разница в том что элементы основного класса к элементам своего клона отношения не имеют.

PRIVATE - НАСЛЕДОВАНИЕ

Используя **private** наследование можно создать первого потомка от которого дальнейшее наследование будет бессмысленно. Если первый потомок получает возможность работы с некоторыми элементами, переданными по механизму наследования, то потомки первого потомка таких возможностей не получают. Кроме того, потомки первого потомка даже лишены возможности узнавать кто их первый родитель. Предполагается, что потомки класса B не должны даже знать о существовании класса A (либо потомков класса B вообще не должно быть).

PROTECTED - НАСЛЕДОВАНИЕ

Используя **protected** наследование, программист предполагает, что внутри всех потомков и потомков потомков и потомков потомков потомков и т.д. будут использоваться только такие элементы, передаваемые механизмом наследования, которые будут защищены от внешнего воздействия извне своих классов.

Примеры последующего наследования одинаковых классов при различных модификаторах наследования:

Public-наследование:

```
Class A{
private: int x;
public: int y;
protected: int z;
};

Class B{
protected: int x;
public: int y;
protected: int z;
};

Class C{
private: int x;
public: int y;
protected: int z;
};
```

Private-наследование:

```
Class A{
private: int x;
public: int y;
protected: int z;
};

Class B{
private: int x;
private: int y;
private: int z;
};

Class C{
private: int x;
private: int y;
private: int z;
};
```

Private-наследование:

```
Class A{
private: int x;
public: int y;
protected: int z;
};

Class B{
private: int x;
protected: int y;
protected: int z;
};

Class C{
private: int x;
protected: int y;
protected: int z;
};
```

3. МНОЖЕСТВЕННОЕ НАСЛЕДОВАНИЕ

C++ позволяет порождать класс из нескольких базовых классов. Один класс может наследовать атрибуты двух и более классов одновременно. Для этого используется список базовых классов, в котором каждый из базовых классов отделен от других запятой. Общая форма множественного наследования имеет вид:

```
class имя_порожденного_класса: список базовых классов {...};
```

В следующем примере класс Z наследует оба класса X и Y:

```
class X {
protected:
int a;
public:
void make_a(int i) { a = i; }
};
class Y {
protected:
int b;
public:
void make_b(int i) { b = i; }
};
// Z наследует как от X, так и от Y
class Z: public X, public Y {
public:
int make_ab() { return a*b; }
};

int main(){
Z i;
i.make_a(10);
i.make_b(12);
cout << i.make_ab();
return 0;
}
```

Поскольку класс Z наследует оба класса X и Y, то он имеет доступ к публичным и защищенным членам обоих классов X и Y. В предыдущем примере ни один из классов не содержал конструкторов.

Стоить отметить, что при создании конструкторов базовых классов они вызываются в том порядке, в котором они указаны в списке при объявлении класса множественного наследования. В общем случае, когда используется список базовых классов, их конструкторы вызываются слева направо. Деструкторы вызываются в обратном порядке — справа налево.

4. ВИРТУАЛЬНЫЕ БАЗОВЫЕ КЛАССЫ

В C++ ключевое слово virtual используется для объявления виртуальных функций, которые будут переопределены в производных классах. Однако ключевое слово virtual также имеет другое использование, позволяющее определить виртуальный базовый класс.

Виртуальные классы используются для решения проблемы множественного наследования одного и того же класса.

Пробема: При множественном наследовании базовый класс не может быть задан в производном классе более одного раза:

```
class B { ... };
class D : B, B {…}: // недопустимо
```

Однако базовый класс может быть передан производному классу более одного раза косвенно:

```
class X : public B {...}
class Y : public B { ... }
class Z : public X, public Y {…} // допустимо
```

В данном случае каждый объект класса Z будет иметь два подобъекта класса B.

Решение: Если множественное наследование в таком случаи вызывает проблемы, к спецификатору базового класса может быть добавлено ключевое слово **virtual**, например:

```
class X : virtual public B {…}
class Y : virtual public B {…}
class Z : public X, public Y {…}
```

Теперь B является виртуальным базовым классом, а класс Z имеет только один подобъект класса B.