Семинар #2: Наследование.

Наследование (англ. *inheritance*) – это концепция объектно-ориентированного программирования, согласно которой абстрактный тип данных может наследовать данные и функциональность некоторого существующего типа, способствуя повторному использованию компонентов программного обеспечения.

Наследование в C++ позволяет создавать классы на основе уже существующих классов. Класс от которого происходит наследование принято называть *базовым*, *родительским* или *суперклассом*, в то время как класс, который наследует от него, именуется *производным*, *дочерним* или *подклассом*. Всё это эквивалентные определения. В английском языке используются аналогичные термины *base*/*derived*, *parent*/*child* и *superclass*/*subclass*. Производный класс наследует от базового класса его поля и методы и может пользоваться ими. Наследование также применимо и к структурам, так как классы и структуры в C++ почти ничем не отличаются.

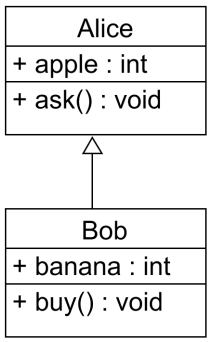
Рассмотрим пример в котором есть базовый класс `Alice` и производный класс `Bob`. То от какого класса наследуется данный класс, прописывается в определении класса через двоеточие. В этом примере класс `Bob` наследует от класса `Alice` поле `apple` и метод `ask` и может их использовать.

```
#include <iostream>
struct Alice
{
    int apple = 10;
    void ask() {std::cout << "ask" << std::endl;}
};

struct Bob : Alice
{
    int banana = 20;
    void buy() {std::cout << "buy" << std::endl;}
};

int main()
{
    Bob b;
    std::cout << b.apple << " " << b.banana << std::endl;
    b.ask();
    b.buy();
}
```

Зависимомость между классами `Alice` и `Bob` часто представляют в виде следующей диаграммы:



Это так называемая UML Class диаграмма. Наследование на такой диаграмме обозначается стрелкой с полым треугольным наконечником. Направлена стрелки – от производного класса к базовому. Знак плюс перед членами класса означает, что они публичные.

В памяти классы `Alice` и `Bob` можно представить следующим образом:

Производный класс `Bob` будет содержать внутри себя объект базового класса `Alice`. Это полностью корректный объект базового класса. Даже если создать указатель типа `Alice*`, указать им на объект типа `Bob` и разыменовать, то это будет корректно и в результате разыменования получится объект типа `Alice`, хранящийся внутри `Bob`.

Примеры наследования

Затенение членов базового класса

Название полей и методов в базовом и производном классах может совпадать. Это не приведёт к ошибке. В этом случае, члены базового класса будут *затенены* членами производного класса.

```
#include <iostream>
struct Alice
{
    int x = 10;
    void func() {std::cout << "alice func" << std::endl;}
};

struct Bob : Alice
{
    int x = 20;
    void func() {std::cout << "bob func" << std::endl;}
};

int main()
{
    Bob b;
    std::cout << b.x << std::endl;          // Напечатает 20
    b.func();                               // Напечатает bob func

    std::cout << b.Alice::x << std::endl;   // Напечатает 10
    b.Alice::func();                        // Напечатает alice func
}
```

В этом случае получить доступ к членам базового класса можно с помощью специального синтаксиса:

```
Bob b;
b.Alice::func(); // Вызываем затенённый метод базового класса Alice,
                 // используя объект производного класса Bob
```

Если методы в родительском классе перегружены, то метод с тем же именем в дочернем классе затенит сразу все перегрузки родительского класса:

```
#include <iostream>
struct Alice
{
    void func(float x)  {std::cout << "float" << std::endl;}
    void func(double x) {std::cout << "double" << std::endl;}
};
```

```
struct Bob : Alice
{
    void func(int x)    {std::cout << "int" << std::endl;}
};

int main()
{
    Bob b;
    b.func(1.5);         // Напечатает int, так cat из Bob затеняет все func из Alice
    b.Alice::func(1.5);  // Напечатает double, выбирает func из Alice по правилам перегрузки
}
```

Другими словами `Alice::func` и `Bob::func` это разные функции не являющиеся перегрузками друг друга.

Затенение в наследнике работает похожим образом на затенение в новой области видимости или в новом пространстве имён. Например, при работе с пространствами имён похожая ситуация выглядит так:

```
#include <iostream>
void func(float x)   {std::cout << "float" << std::endl;}
void func(double x)  {std::cout << "double" << std::endl;}
namespace mipt
{
    void func(int x) {std::cout << "int" << std::endl;}
    void test() {func(1.5);} // func из mipt затеняет func из глобального пространства
}
int main()
{
    mipt::test();   // Напечатает int
    func(1.5);      // Напечатает double
}
```

Модификация доступа

Защищённые члены класса. Модификатор доступа `protected`.

Приватные члены класса доступны только в самом классе и в друзьях класса, но не в дочерних классах. Защищённые члены класса доступны в самом классе, в друзьях и в дочерних классах.

```
#include <iostream>
struct Alice
{
public:
    int x = 10;  // Публичное поле
protected:
    int y = 20;  // Защищённое поле
private:
    int z = 30;  // Приватное поле
};

struct Bob : Alice
{
    void func()
    {
        std::cout << x << std::endl;  // OK, доступ есть, так как поле x публичное
        std::cout << y << std::endl;  // OK, доступ есть, так как поле y защищённое
        std::cout << z << std::endl;  // Ошибка, нет доступа к z, так как поле z приватное
    }
};

int main()
{
    Alice a;
    std::cout << a.x << std::endl;  // OK, доступ есть, так как поле x публичное
    std::cout << a.y << std::endl;  // Ошибка, нет доступа к y, так как поле y защищённое
    std::cout << a.z << std::endl;  // Ошибка, нет доступа к z, так как поле z приватное
}
```

Получать доступ к защищённому полю в производном классе можно только через объект производного класса:

```
struct Alice
{
protected:
    int y = 20;
};

struct Bob : Alice
{
    void func(Alice& a, Bob& b)
    {
        std::cout << y   << std::endl;  // OK
        std::cout << a.y << std::endl;  // Ошибка
        std::cout << b.y << std::endl;  // OK
    }
};
```

В данном примере класс `Bob` при наследовании получает поле `y` и имеет доступ только к этому полю, чьё полное название `Bob::y`. Но класс `Bob` не имеет доступа к полю `Alice::y`.

Публичное, защищённое и приватное наследование

В C++ есть три типа наследования: публичное, защищённое и приватное.

- *Публичное наследование* – поля, которые в родительском классе были публичными или защищёнными, остаются такими же в дочернем классе. Самый распространённый тип наследования. В предыдущих примерах использовался именно этот тип наследования.
- *Защищённое наследование* – поля, которые в родительском классе были публичными или защищёнными, становятся защищёнными в дочернем классе. Почти никогда не используется.
- *Приватное наследование* – поля, которые в родительском классе были публичными или защищёнными, становятся приватными в дочернем классе. Иногда используется.

Тип наследования указывается в определении класса сразу после двоеточия. Для типа наследования используются те же ключевые слова, что и для модификаторов доступа членов класса. Это не случайно, можно считать, что базовый класс является часть производного класса с модификатором доступа соответствующему типу наследования.

```
#include <iostream>
struct Alice
{
public:
    int x = 10;  // Публичное поле
protected:
    int y = 20;  // Защищённое поле
private:
    int z = 30;  // Приватное поле
};

// Приватно наследуем Bob от Alice
// Поля, которые были в Alice публичными или защищёнными, в Bob станут приватным
struct Bob : private Alice
{
    void func()
    {
        std::cout << x << std::endl;  // OK, доступ есть, так как поле x в Alice публичное
        std::cout << y << std::endl;  // OK, доступ есть, так как поле y в Alice защищённое
        std::cout << z << std::endl;  // Ошибка, нет доступа, так как поле z в Alice приватное
    }
};

int main()
{
    Alice a;
    std::cout << a.x << std::endl;  // OK, доступ есть, так как поле x в Alice публичное
    std::cout << a.y << std::endl;  // Ошибка, нет доступа, так как поле y в Alice защищённое
    std::cout << a.z << std::endl;  // Ошибка, нет доступа, так как поле z в Alice приватное

    Bob b;
    std::cout << b.x << std::endl;  // Ошибка, нет доступа, так как поле x в Bob приватное
    std::cout << b.y << std::endl;  // Ошибка, нет доступа, так как поле y в Bob приватное
    std::cout << b.z << std::endl;  // Ошибка, нет доступа, так как поле z в Bob приватное
}
```

Как запомнить, что делает тот или иной тип наследования

Запомнить, что делает тот или иной тип наследования, можно, если представлять, что базовый класс не наследуется, а просто является полем производного класса с модификатором доступа, соответствующим типу наследования:

```
struct Bob : private Alice
{
public:
    void func()
    {
    // x, y, z тут будут иметь тот же доступ
    }
};
```

```
struct Bob
{
private:
    Alice a;

public:
    void func()
    {
    // что и a.x, a.y, a.z вот тут
    }
};
```

Различие между определение класса с помощью `class` и `struct`

Классы в языке C++ можно создавать как с помощью ключевого слова **`class`**, так и с помощью ключевого слова class. Есть ровно два отличия между классами созданными с использованием **`struct`** и классами, созданными с использованием **`class`**:

1. У классов, созданных с использованием **`struct`**, все члены по умолчанию публичны, в то время как у классов, определённых с помощью **`class`**, члены по умолчанию являются приватными.

   ```
   struct Alice
   {
       int x; // Это публичное поле
   };
   ```

   ```
   class Alice
   {
       int x; // Это приватное поле
   };
   ```

2. Если класс, созданный с использованием **`struct`**, наследует без указания типа наследования, то по умолчанию выбирается публичное наследование. У классов, созданных с использованием **`class`**, по умолчанию выберется приватное наследование.

   ```
   struct Bob : Alice  // Публичное
   {
       ...
   }
   ```

   ```
   class Bob : Alice  // Приватное
   {
       ...
   }
   ```

Наследование и друзья

Самое главное, что нужно знать про друзей в контексте наследования:

- Используя друзей, можно обойти любые запреты, созданные модификаторами доступа.
- Друзья не наследуются. Друзья родительского класса не обязательно являются друзьями дочернего класса.