НАСЛЕДОВАНИЕ

Наследование

- процесс, в ходе которого один объект может приобрести свойства другого

Наследование в C++

- Один класс приобретает свойства другого класса в момент своего объявления
- Создание иерархии классов с уточнением их свойств от самых общих до более конкретных
- Процесс наследования:
  - определение базового класса, свойства которого будут общими для всех его наследников
  - определение производных классов, наследующих свойства базового класса

Управление доступом к членам базового класса

class *имя-производного-класса* : уровень_ доступа *имя-базового-класса*

{

// тело класса

}

уровень_ доступа – определяет статус членов базового класса в производном классе:

- **public**
- **private**
- **protected**

Если *уровень_ доступа* не указан, о для производного класса по умолчанию используется спецификатор **private**, а для производной структуры - **public**

Уровень доступа **public**

- Все открытые и защищенные члены базового класса становятся открытыми и защищенными членами производного класса
- Закрытые члены базового класса не меняют своего статуса и остаются недоступными членам производного

Объекты класса derived могут непосредственно ссылаться на открытые члены класса base

```
#include <iostream>
using namespace std;

class base {
  int i, j;
public:
  void set(int a, int b) { i=a; j=b; }
  void show() { cout << i << " " << j << "\n"; }
};

class derived : public base {
  int k;
public:
  derived(int x) { k=x; }
  void showk() { cout << k << "\n"; }
};

int main()
{
  derived ob(3);

  ob.set(1, 2); // Обращение к члену класса base
  ob.show(); // Обращение к члену класса base

  ob.showk(); // Обращение к члену класса derived

  return 0;
}
```

Уровень доступа **private**

- Все открытые и защищенные члены базового класса становятся закрытыми членами производного класса
- Они остаются доступными членам производного класса, но недоступны остальным элементам программы, не являющимся членами базового или производного класса

Пример уровня доступа **private**

```
// Эта программа не будет скомпилирована.
#include <iostream>
using namespace std;

class base {
  int i, j;
public:
  void set(int a, int b) { i=a; j=b; }
  void show() { cout << i << " " << j << "\n";}
};

// Открытые члены класса base являются
// закрытыми членами класса derived.
class derived : private base {
  int k;
public:
  derived(int x) { k=x; }
  void showk() { cout << k << "\n"; }
};

int main()
{
  derived ob(3);

  ob.set(1, 2); // Ошибка, доступ к функции set() запрещен.
  ob.show(); // Ошибка, доступ к функции show() запрещен.

  return 0;
}
```

Задание

- Разработать объектную модель кошелька
- Использовать:
  - конструкторы
  - дружественные функции
  - подставляемые функции
  - наследование: открытое и закрытое

Наследование и защищенные члены - **protected**

- Повышение гибкости механизма наследования
- Защищенный член класса также как и закрытый недоступен вне класса

Отличие

- При открытом наследовании
  - закрытые члены класса остаются закрытыми
  - защищенные члены базового класса становятся защищенными членами производного класса
- То есть защищенные члены класса являются закрытыми, но могут наследоваться производным классом

Открытое наследование защищенных членов

```
#include <iostream>
using namespace std;

class base {
protected:
  int i, j; // Закрыты по отношению к классу base,
            // но доступны классу derived.
public:
  void set(int a, int b) { i=a; j=b; }
  void show() { cout << i << " " << j << "\n"; }
};

class derived : public base {
  int k;
public:
  // Класс derived имеет доступ к членам i и j из класса base
  void setk() { k=i*j; }

  void showk() { cout << k << "\n"; }
};

int main()
{
  derived ob;

  ob.set(2, 3); // Все в порядке, этот член доступен классу derived
  ob.show(); // Все в порядке, этот член доступен классу derived

  ob.setk();
  ob.showk();

  return 0;
}
```

Если переменные i и j объявлены защищенными и класс derived наследует свойства с помощью открытого наследования, то функция seek() имеет доступ к i и j

- Если производный класс является базовым по отношению к другому производному классу, то любой защищенный член исходного базового класса, открыто наследуемый первым производным классом, также может наследоваться вторым производным классом как защищенный член

```
class base {
protected:
  int i, j;
public:
  void set(int a, int b) { i=a; j=b; }
  void show() { cout << i << " " << j << "\n"; }
};

// Переменные i и j наследуются как защищенные.
class derived1 : public base {
  int k;
public:
  void setk() { k = i*j; } // legal
  void showk() { cout << k << "\n"; }
};

// i and j inherited indirectly through derived1.
class derived2 : public derived1 {
  int m;
public:
  void setm() { m = i-j; } // Допускается
  void showm() { cout << m << "\n"; }
};

int main()
{
  derived1 ob1;
  derived2 ob2;

  ob1.set(2, 3);
  ob1.show();
  ob1.setk();
  ob1.showk();

  ob2.set(3, 4);
  ob2.show();
  ob2.setk();
  ob2.setm();
  ob2.showk();
  ob2.showm();

  return 0;
}
```

Закрытое наследование защищенных членов

- При закрытом наследовании все члены класса base становятся закрытыми членами класса derived1 и недоступны классу derived2

```
class base {
protected:
  int i, j;
public:
  void set(int a, int b) { i=a; j=b; }
  void show() { cout << i << " " << j << "\n"; }
};
```

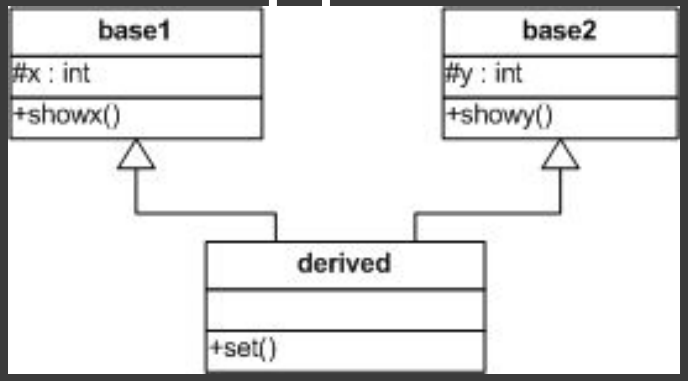
```
// Теперь все члены класса base являются закрытыми членами
// класса derived1.
class derived1 : private base {
  int k;
public:
  // Это делать можно, поскольку переменные i и j
  // являются закрытыми членами класса derived1.
  void setk() { k = i*j; } // OK
  void showk() { cout << k << "\n"; }
};

// Доступ к членам i, j, set() и show() не наследуется.
class derived2 : public derived1 {
  int m;
public:
  // Неправильно, так как переменные i и j являются закрытыми
  // членами класса derived1
  void setm() { m = i-j; } // Ошибка
  void showm() { cout << m << "\n"; }
};

int main()
{
  derived1 ob1;
  derived2 ob2;

  ob1.set(1, 2); // Ошибка, вызывать функцию set() нельзя.
  ob1.show(); // Ошибка, вызывать функцию show() нельзя.

  ob2.set(3, 4); // Ошибка, вызывать функцию set() нельзя.
  ob2.show(); // Ошибка, вызывать функцию show() нельзя.

  return 0;
}
```

Защищенное наследование

- Все открытые и защищенные члены базового класса становятся защищенным и членами производного

```
// Класс, полученный с помощью защищенного наследования.
class derived : protected base {
  int k;
public:
  // Класс derived имеет доступ к членам i, j и setij()
  // из класса base.
  void setk() { setij(10, 12); k = i*j; }

  // Отсюда можно вызвать функцию showij().
  void showall() { cout << k << " "; showij(); }
};

int main()
{
  derived ob;

//  ob.setij(2, 3); // Неверно, функция setij() является
//                     закрытым членом класса derived.

  ob.setk(); // Верно, вызывается открытый член класса derived.
  ob.showall(); // Верно, вызывается открытый член класса derived.

// ob.showij(); // Неверно, функция showij() является
                // защищенным членом класса derived

  return 0;
}
```

Множественное наследование




```
class base1 {
protected:
  int x;
public:
  void showx() { cout << x << "\n"; }
};
```

```
class base2 {
protected:
  int y;
public:
  void showy() {cout << y << "\n";}
};

// Множественное наследование.
class derived: public base1, public base2 {
public:
  void set(int i, int j) { x=i; y=j; }
};

int main()
{
  derived ob;

  ob.set(10, 20); // Эта функция принадлежит классу derived.
  ob.showx(); // Эта функция принадлежит классу base1.
  ob.showy(); // Эта функция принадлежит классу base2.

  return 0;
}
```

Конструкторы, деструкторы и наследование

- Вызов конструкторов и деструкторов
- При создании объекта производного класса сначала вызывается конструктор базового класса, а затем – производного
- При уничтожении объекта производного класса сначала вызывается конструктор производного класса, а затем – базового

```
class base {
public:
  base() { cout << "Создается объект класса base\n"; }
  ~base() { cout << "Уничтожается объект класса base\n"; }
};

class derived: public base {
public:
  derived() { cout << "Создается объект класса derived\n"; }
  ~derived() { cout << "Уничтожается объект класса derived\n"; }
};
```

```
int main()
{
  derived ob;

  // Кроме создания и уничтожения объекта, ничего не происходит

  return 0;
}
```

Как указано в комментарии к функции **main()**, программа просто создает, а затем уничтожает объект **ob** класса **derived**. В ходе выполнения программа выводит на экран следующие сообщения.

```
Создание объекта класса base
Создание объекта класса derived
Уничтожение объекта класса derived
Уничтожение объекта класса base
```

Вызов конструкторов и деструкторов при иерархическом наследовании

- Конструкторы вызываются в иерархическом порядке
- деструкторы – в обратном

```
class base {
public:
  base() { cout << "Создание объекта класса base\n"; }
  ~base() { cout << "Уничтожение объекта класса base\n"; }
};

class derived1 : public base {
public:
  derived1() { cout << "Создание объекта класса derived1\n"; }
  ~derived1() { cout << "Уничтожение объекта класса derived1\n"; }
};

class derived2: public derived1 {
public:
  derived2() { cout << "Создание объекта класса derived2\n"; }
  ~derived2() { cout << "Уничтожение объекта класса derived2\n"; }
};
```

```
int main()
{
  derived2 ob;

  // Создаем и уничтожаем объект ob

  return 0;
}
```

В результате на экран выводятся следующие строки.

```
Создание объекта класса base
Создание объекта класса derived1
Создание объекта класса derived2
Уничтожение объекта класса derived2
Уничтожение объекта класса derived1
Уничтожение объекта класса base
```

Множественное наследование

- Конструкторы вызываются в иерархическом порядке
- Деструкторы – в обратном

```
#include <iostream>
using namespace std;

class base1 {
public:
  base1() { cout << "Создание объекта класса base1\n"; }
  ~base1() { cout << "Уничтожение объекта класса base1\n"; }
};

class base2 {
public:
  base2() { cout << " Создание объекта класса base2\n"; }
  ~base2() { cout << " Создание объекта класса base2\n"; }
};

class derived: public base1, public base2 {
public:
  derived() { cout << " Создание объекта класса derived\n"; }
  ~derived() { cout << " Уничтожение объекта класса derived\n"; }
};

int main()
{
  derived ob;

  // Создание и уничтожение объекта ob

  return 0;
}
```

Эта программа выдает на экран следующие сообщения.

```
Создание объекта класса base1
Создание объекта класса base2
Создание объекта класса derived
Уничтожение объекта класса derived
Уничтожение объекта класса base2
Уничтожение объекта класса base1
```

Передача параметров конструктору базового класса

```
конструктор_производного-класса (список_аргументов) :base1(список_аргументов),
                                                      base2(список_аргументов),
                                                      . . .
                                                      baseN(список_аргументов)
{
    // Тело конструктора производного класса
}
```

```
#include <iostream>
using namespace std;

class base {
protected:
  int i;
public:
  base(int x) { i=x; cout << "Создание объекта класса base\n"; }
  ~base() { cout << "Уничтожение объекта класса base\n"; }
};

class derived: public base {
  int j;
public:
  // Класс derived использует переменную x;
  // переменная y передается базовому классу.
  derived(int x, int y): base(y)
    { j=x; cout << "Создание объекта класса derived\n"; }

  ~derived() { cout << "Уничтожение объекта класса derived\n"; }
  void show() { cout << i << " " << j << "\n"; }
};

int main()
{
  derived ob(3, 4);

  ob.show(); // Выводит на экран числа 4 3

  return 0;
}
```

```
#include <iostream>
using namespace std;

class base1 {
protected:
  int i;
public:
  base1(int x) { i=x; cout << "Создание объекта класса base1\n"; }
  ~base1() { cout << "Уничтожение объекта класса base1\n"; }
};

class base2 {
protected:
  int k;
public:
  base2(int x) { k=x; cout << "Создание объекта класса base2\n"; }
  ~base2() { cout << "Уничтожение объекта класса base2\n"; }
};

class derived: public base1, public base2 {
  int j;
public:
  derived(int x, int y, int z): base1(y), base2(z)
    { j=x; cout << "Создание объекта класса derived\n"; }

  ~derived() { cout << "Уничтожение объекта класса derived\n"; }
  void show() { cout << i << " " << j << " " << k << "\n"; }
};

int main()
{
  derived ob(3, 4, 5);

  ob.show(); // Выводит на экран числа 4 3 5

  return 0;
}
```

```
#include <iostream>
using namespace std;

class base1 {
protected:
  int i;
public:
  base1(int x) { i=x; cout << "Создание объекта класса base1\n"; }
  ~base1() { cout << "Уничтожение объекта класса base1\n"; }
};

class base2 {
protected:
  int k;
public:
  base2(int x) { k=x; cout << "Создание объекта класса base2\n"; }
  ~base2() { cout << "Уничтожение объекта класса base2\n"; }
};

class derived: public base1, public base2 {
public:
  /* Конструктор класса derived не имеет параметров,
 в его объявлении указываются параметры конструкторов
     базовых классов.
 */

  derived(int x, int y): base1(x), base2(y)
    { cout << "Создание объекта класса derived\n"; }

  ~derived() { cout << "Уничтожение объекта класса derived\n"; }
  void show() { cout << i << " " << k << "\n"; }
};

int main()
{
  derived ob(3, 4);

  ob.show(); // Выводит на экран числа 3 4

  return 0;
}
```

Предоставление доступа

базовый_класс::*член*;

Это объявление размещается внутри производного класса после заголовка соответствующего раздела. Обратите внимание на то, что тип переменной в объявлении уровня доступа указывать не следует.

```
class base {
public:
  int j; // Открытый член класса base
};

// Закрытый наследник класса base.
class derived: private base {
public:

  // Место для объявления уровня доступа.
  base::j; // Теперь переменная j снова открыта.
  .
  .
  .
};
```

Поскольку класс **derived** является закрытым наследником класса **base**, открытая переменная-член **j** из класса **base** становится закрытой переменной-членом класса **derived**. Однако в раздел **public** класса **derived** мы поместили объявление уровня доступа

```
base::j;
```

```
#include <iostream>
using namespace std;

class base {
  int i; // Закрытый член класса base
public:
  int j, k;
  void seti(int x) { i = x; }
  int geti() { return i; }
};

// Закрытый наследник класса base.
class derived: private base {
public:
  /* Следующие три оператора восстанавливают
     открытый статус членов j, seti() и geti(). */
  base::j; // Переменная j снова открыта, а переменная k — нет.
  base::seti; // Функция seti() снова открыта.
  base::geti; // Функция geti() снова открыта.

// base::i; // Неверно, уровень доступа поднимать нельзя.

  int a; // Открытый член.
};

int main()
{
  derived ob;

//ob.i = 10; // Нельзя, так как переменная i
             // является закрытым членом класса derived.

  ob.j = 20; // Можно, так как переменная j открыта в классе derived.
//ob.k = 30; // Нельзя, поскольку переменная k является
             // закрытым членом класса derived.

  ob.a = 40; // Можно, так как переменная a является открытым
             // членом класса derived.
  ob.seti(10);

  cout << ob.geti() << " " << ob.j << " " << ob.a;

  return 0;
}
```

Задание