Научно-исследовательский практикум

ДИНАМИЧЕСКИЙ И СТАТИЧЕСКИЙ ПОЛИМОРФИЗМ.

Базовый класс

```
class Base
{
public:
          Base();
          ~Base();
          void NonVirtualMethod();
          virtual void VirtualMethodA();
          virtual void VirtualMethodB();
};
Base::Base()
{
          cout << "Base default c-tor called" << endl;
}

Base::~Base()
{
          cout << "Base d-tor called" << endl;
}

void Base::NonVirtualMethod()
{
          cout << "Non-virtual method of Base class called" << endl;
}

void Base::VirtualMethodA()
{
          cout << "Virtual method A of Base class called" << endl;
}

void Base::VirtualMethodB()
{
          cout << "Virtual method B of Base class called" << endl;
}
```

Первый наследник

```
class HeirA : public Base
{
public:
       void NonVirtualMethod();
       virtual void VirtualMethodA();
};

void HeirA::NonVirtualMethod()
{
       cout << "Non-virtual method of Heir A class called" << endl;
}

void HeirA::VirtualMethodA()
{
       cout << "Virtual method A of Heir A class called" << endl;
}
```

Второй наследник

```
class HeirB : public Base
{
public:
    void NonVirtualMethod();
    virtual void VirtualMethodB();
};

void HeirB::NonVirtualMethod()
{
    cout << "Non-virtual method of Heir B class called" << endl;
}

void HeirB::VirtualMethodB()
{
    cout << "Virtual method B of Heir B class called" << endl;
}
```

main() – первый вариант

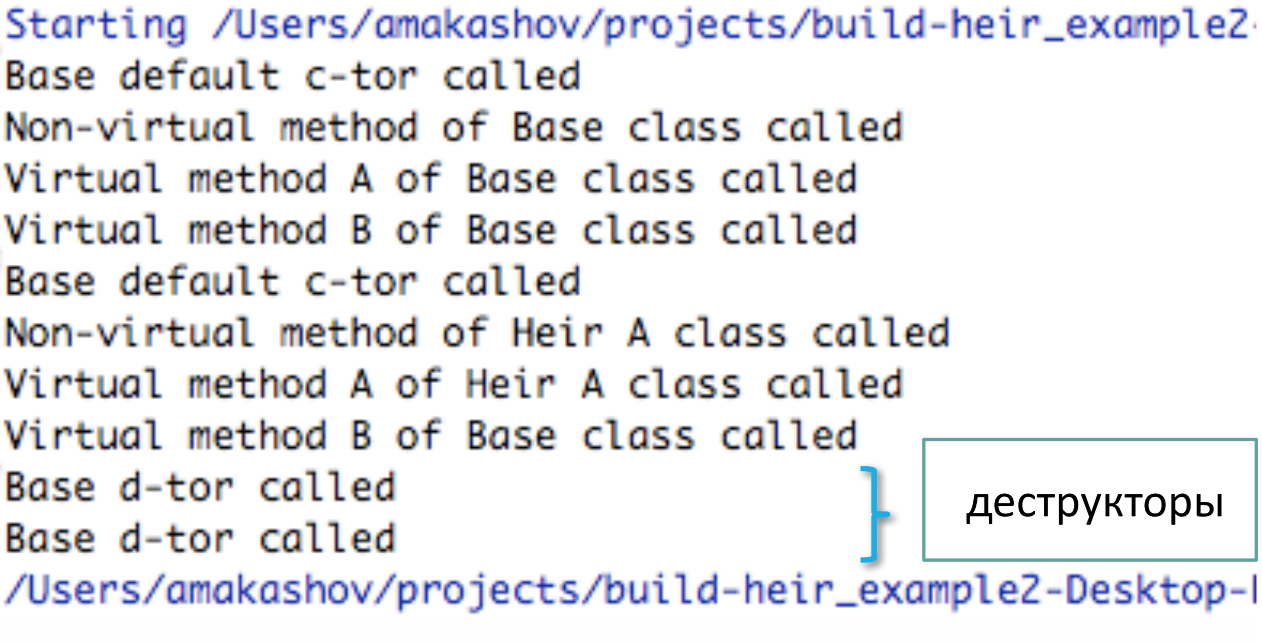
Мы создадим размещённые в стеке экземпляры объектов

```
int main(int argc, char *argv[])
{
        Base baseObject;
        baseObject.NonVirtualMethod();
        baseObject.VirtualMethodA();
        baseObject.VirtualMethodB();

        HeirA aObject;
        aObject.NonVirtualMethod();
        aObject.VirtualMethodA();
        aObject.VirtualMethodB();

        return 0;
}
```

Что получилось



```
Starting /Users/amakashov/projects/build-heir_example2-
Base default c-tor called
Non-virtual method of Base class called
Virtual method A of Base class called
Virtual method B of Base class called
Base default c-tor called
Non-virtual method of Heir A class called
Virtual method A of Heir A class called
Virtual method B of Base class called
Base d-tor called
Base d-tor called
/Users/amakashov/projects/build-heir_example2-Desktop-
```

baseObject

aObject

деструкторы

Новый main()

Теперь мы для обращения к элементам будем использовать указатель

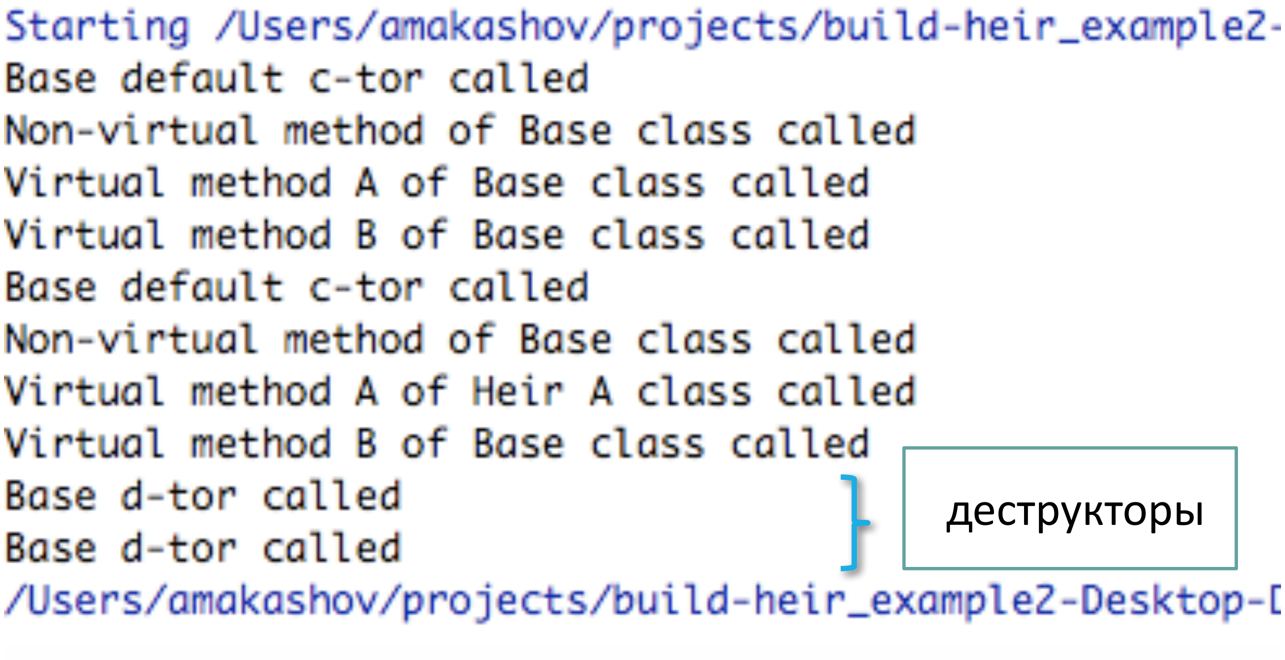
```
int main(int argc, char *argv[])
{
      Base* ptr = nullptr;

      Base baseObject;
      ptr = &baseObject;
      ptr->NonVirtualMethod();
      ptr->VirtualMethodA();
      ptr->VirtualMethodB();

      HeirA aObject;
      ptr = &aObject;
      ptr->NonVirtualMethod();
      ptr->VirtualMethodA();
      ptr->VirtualMethodB();

      return 0;
}
```

Что получилось



```
Starting /Users/amakashov/projects/build-heir_example2-
Base default c-tor called
Non-virtual method of Base class called
Virtual method A of Base class called
Virtual method B of Base class called
Base default c-tor called
Non-virtual method of Base class called
Virtual method A of Heir A class called
Virtual method B of Base class called
Base d-tor called
Base d-tor called
/Users/amakashov/projects/build-heir_example2-Desktop-D
```

baseObject

aObject

деструкторы

Указатель на функцию

С точки зрения C++ функция – это тоже разновидность данных

У функции есть адрес, позволяющий определить, где находится её реализация

Можно реализовать указатель на функцию можно:

```
Возвращаемый_Тип (*имя_указаетля) (Входной_тип1, Входной_тип2,  …);
```

Например,

```
double (*someFunct) (double);
```

создаёт указатель на функцию, которая принимает на вход один double и возвращает double

Пример

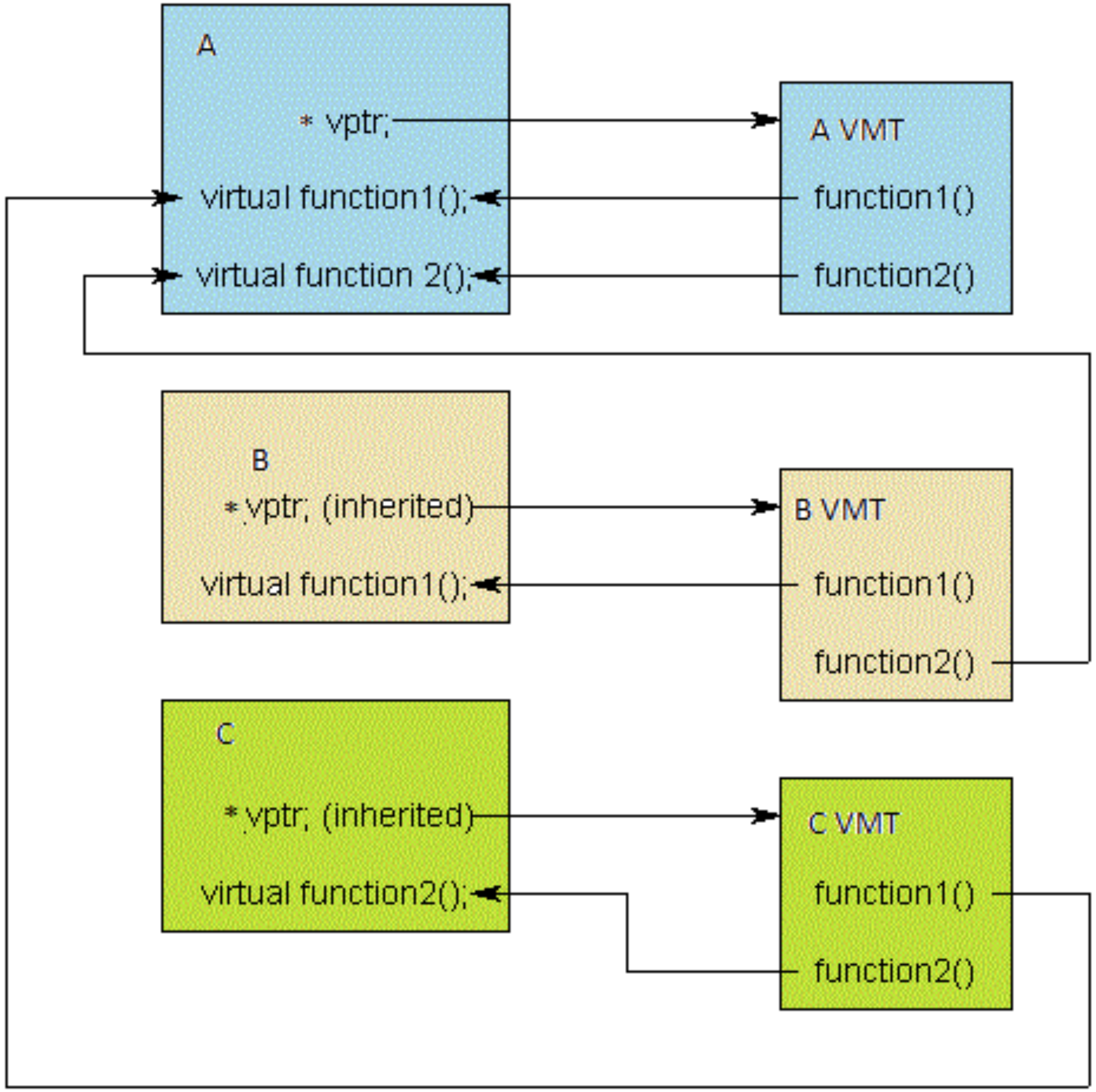
```
double Square(double input) {return pow(input,2);}
double Cube(double input) {return pow(input,3);}
double Root(double input) {return pow(input,0.5);}


int main(int argc, char *argv[])
{
  double (*someFunct) (double);

  someFunct = Square;
  cout << "Square of 2 is " << someFunct(2) << endl;
  someFunct = Cube;
  cout << "Cube of 2 is " << someFunct(2) << endl;
  someFunct = Root;
  cout << "Square root of 2 is " << someFunct(2) <<
endl;

  return 0;
}
```

Построение таблицы виртуальных функций



Какие функции бывают виртуальными?

- Любая функция-член может быть виртуальной
- Не бывает виртуальных конструкторов
- А вот деструкторы должны быть виртуальными
- Есть чистые виртуальный функции:

```
class Base
{
public:
        void NonVirtualMethod();
        virtual void VirtualMethodA() = 0; // pure virtual
        virtual void VirtualMethodB();
};
```

Самостоятельно

Реализуйте класс комплексных чисел:

1. Конструктор, принимающий действительную и мнимую части
2. Функция-геттер и сеттер для действительной и мнимой частей
3. Операторы сложения, вычитания, умножения и деления
4. Определите, нужны ли вам конструктор-копировщик, деструктор и оператор присваивания?