# Программирование

Лекция 7

# Наследование

1. Особенности.
2. Открытые, защищенные и закрытые элементы.
3. Переопределение функций базового класса
4. Конструкторы и деструкторы при наследовании.

# Наследование

- **Наследование** – это способ повторного использования кода, при котором новые классы создаются на баз существующих классов путем заимствования их элементов (данных и функций).
- Новый класс объявляется **наследником** ранее определенного класса (**базового класса**).
- Новый класс называют **производным классом**.
- Производный класс может стать базовым для

# Наследование

- Наследование может быть:
  - **Простое** (производный класс – прямой потомок только одного базового класса).
  - **Множественное** (производный класс – прямой потомок нескольких базовых классов).

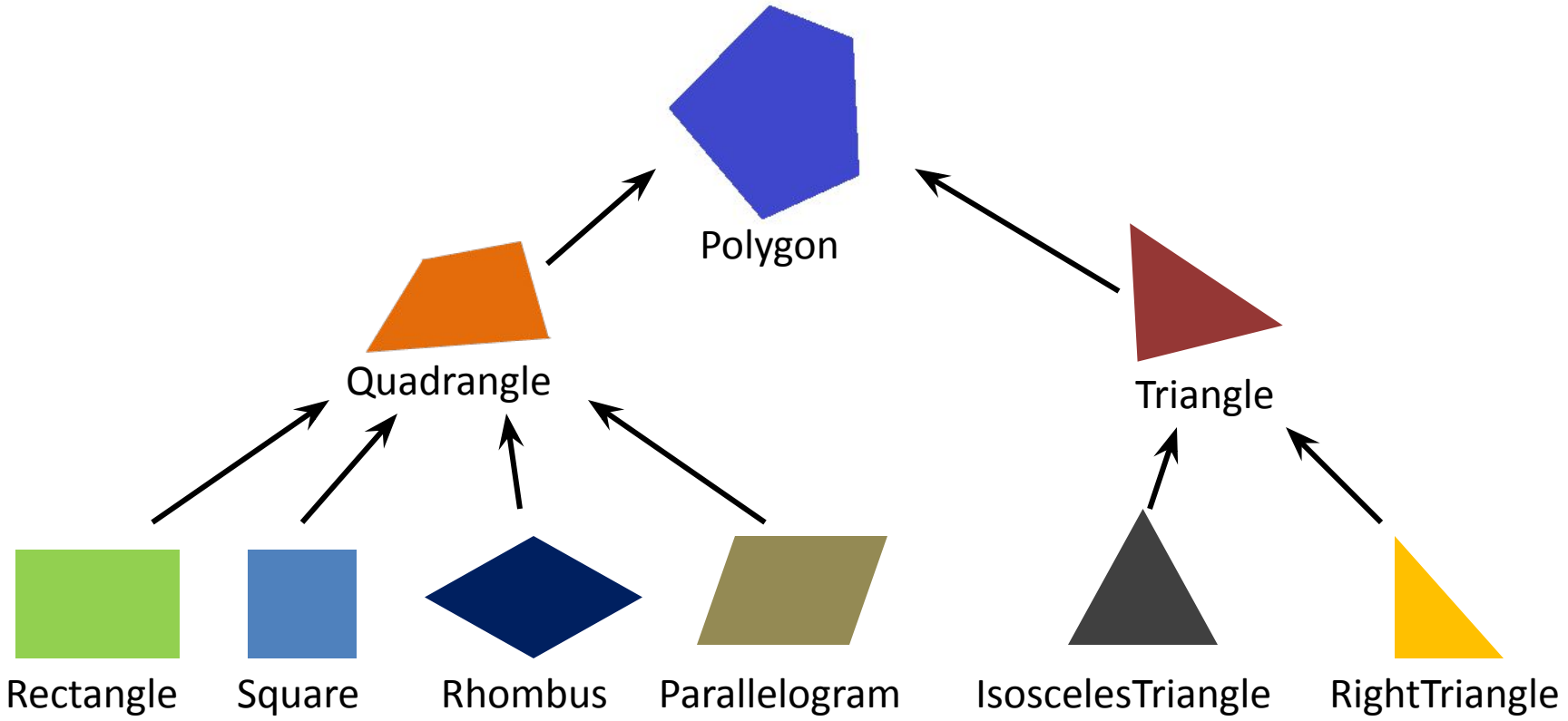
# Особенности производного класса

- Обычно производный класс (ПК) содержит больше данных и функций, чем базовый класс (БК). Он наследует элементы БК и добавляет собственные.
- ПК более специализирован, специфичен в своей области, представляет меньшее количество объектов.
- Объект производного класса также является объектом базового класса. Но объект базового класса не является объектом производного класса.
- В C++ возможны три вида наследования: открытое (**public**), защищенное (**protected**) и закрытое (**private**).

# Особенности производного класса

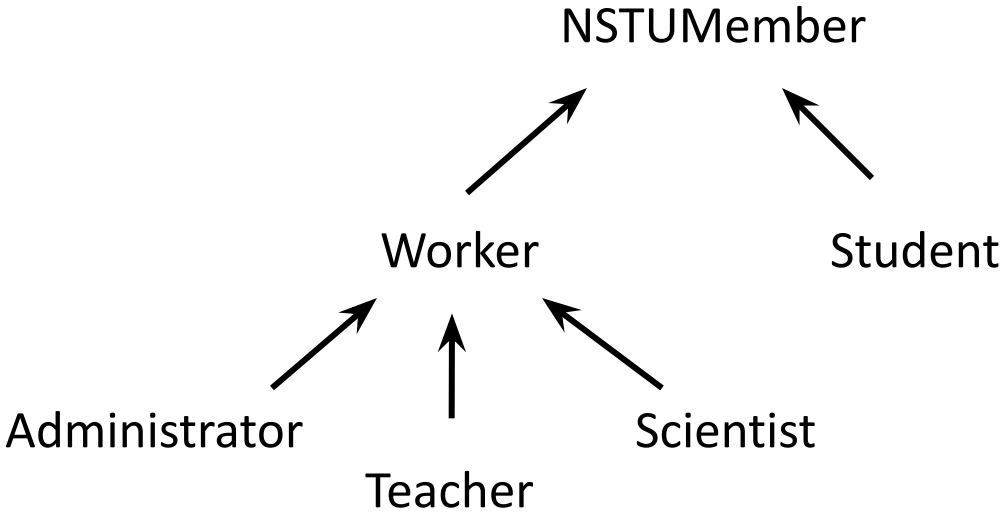
- В БК можно использовать спецификатор доступа **protected**. Защищенные элементы доступны только производным классам и их друзьям.
- ПК **не** имеет прямого доступа к закрытым элементам БК.
- ПК имеет прямой доступ к открытым и защищенным элементам БК.
- Если у БК есть друзья, то производный класс их не наследует.
- Производный класс может переопределить реализацию функций-элементов базового

# Базовые и производные классы




- Наследование порождает иерархические древовидные структуры данных. В них всегда можно добавить новый класс в требуемом месте.

# Базовые и производные классы



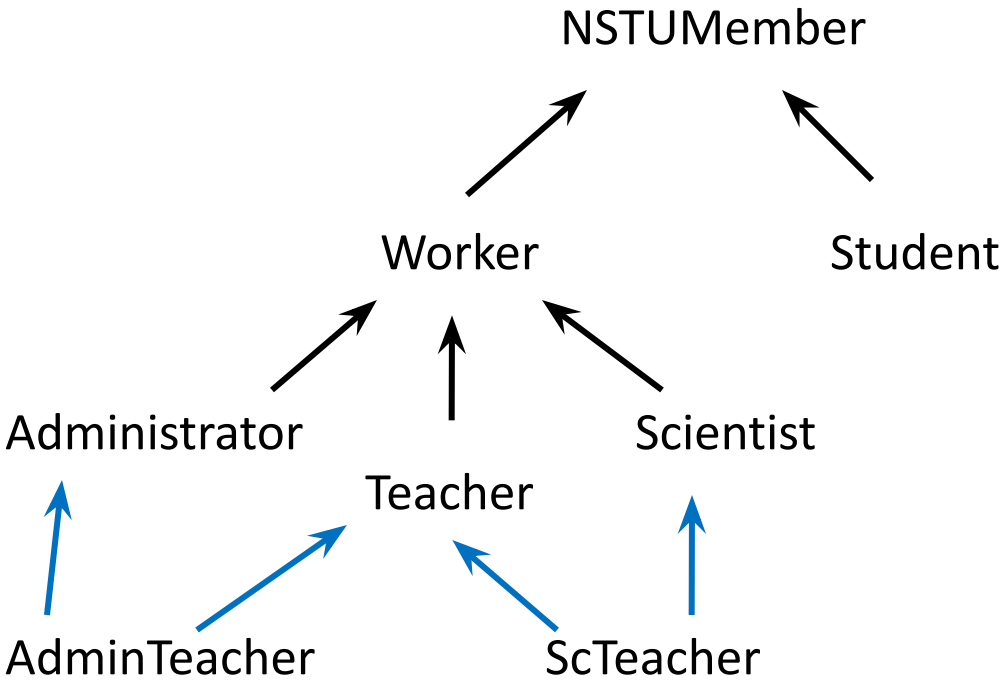

- Синтаксис определения наследования:

  class Worker **: public NSTUMember** { …

- Класс может быть базовым прямо и косвенно:

Для Teacher прямым базовым классом является Worker,
а косвенным – NSTUMember.

# Множественное наследование




class ScTeacher **: public Teacher, public Scientist** { …

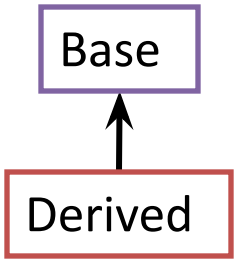
class AdminTeacher **: public Teacher,**

**public Administrator** { …

# Наследование

```
class Base {
  int x;
public:
  void setX(int n) { x = n; }
  void showX() {cout << x << ' '; }
};

class Derived : public Base {
  int y;
public:
  void setY(int n) { y = n; }
  void showY() {cout << y << ' '; }
};
```




```
int main() {
  Base ob;
  ob.setX(5);
  ob.showX();

  Derived od;
  od.setX(10);
  od.setY(20);
  od.showX();
  od.showY();
  return 0;
}
```

# Открытые, защищенные и закрытые базовые классы

## Статус элемента БК в ПК в зависимости от типа наследования и статуса в БК

| Спецификатор доступа для элемента в БК | Тип наследования | | |
|---|---|---|---|
| | **public** | **protected** | **private** |
| **public** | public | protected | private |
| **protected** | protected | protected | private |
| **private** | Не доступен | Не доступен | Не доступен |

# Переопределение функций базового класса в производном классе

- Если функция-элемент повторно описана в производном классе (перегружена), то именно эта новая версия будет выполняться при вызове.
- При этом сохраняется возможность вызова варианта функции, определенного в базовом классе.

```
<Имя_БК> :: <Имя_ФЭ> ();
```

# Файл Worker.h

```
#ifndef WORKER_H
#define WORKER_H

class Worker {
protected:
    char *fam, *name;
public:
    Worker(char *, char *);
    ~Worker();
    void print();
};

#endif
```

# Файл Worker.cpp

```
#include "Worker.h"
#include <string.h>
#include <iostream>

Worker::Worker(char *f, char *n) {
    fam=new char[strlen(f) + 1];
    if(!fam) return;
    strcpy(fam, f);
    name=new char[strlen(n) + 1];
    if(!name) return;
    strcpy(name, n);
}
```

# Файл Worker.cpp

```
Worker::~Worker() {
    std::cout<<"Деструктор Worker"<<std::endl;
    delete []fam;
    delete []name;
}

void Worker::print() {
    std::cout<<fam<<" "<<name<<std::endl;
}
```

# Файл Teacher.h

```
#ifndef TEACHER_H
#define TEACHER_H

#include "Worker.h"

class Teacher : public Worker {
    float hours; // Количество часов в неделю
    float wage;  // Почасовая ставка
public:
    Teacher(char *, char *, float, float);
    float getPay();
    void print();
};

#endif
```

# Файл Teacher.cpp

```
#include "Teacher.h"
#include <iostream>

Teacher::Teacher(char *f, char *n,
                 float h, float w) : Worker(f, n) {
    hours=h;
    wage=w;
}
```

# Файл Teacher.cpp

```
float Teacher::getPay() {
    return hours*wage;
}

void Teacher::print() {
    std::cout<<"Teacher::print()"<<std::endl;
    Worker::print();
    std::cout<<"Преподаватель с оплатой ";
    std::cout<<getPay()<<" руб. в неделю"<<std::endl;
}
```

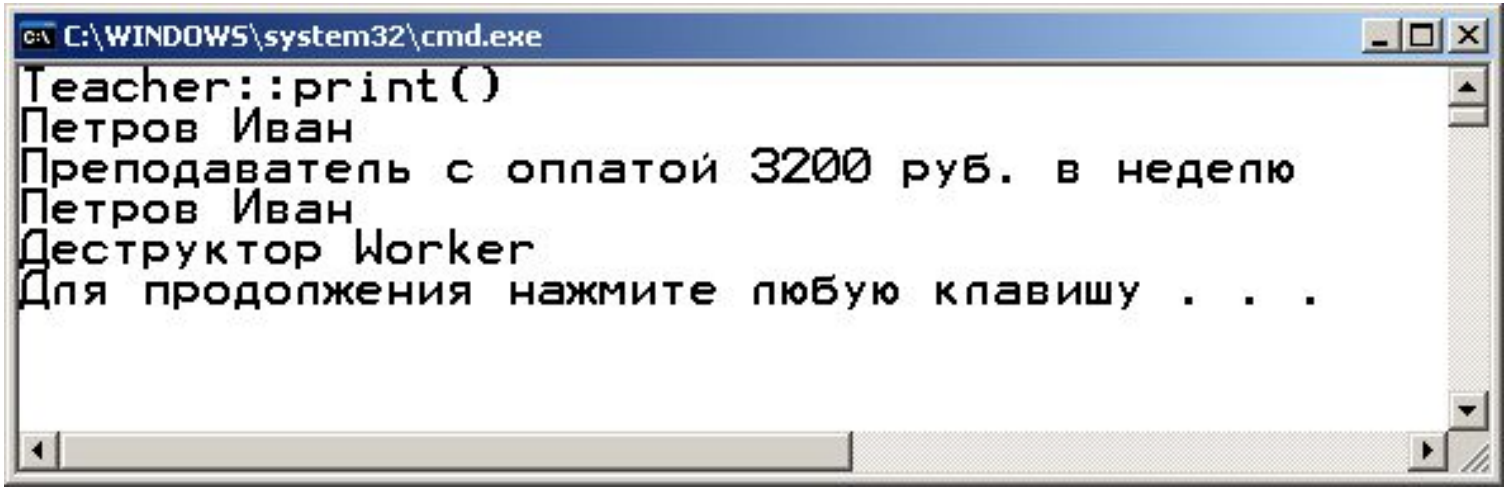
# Файл main.cpp

```
#include "Teacher.h"

#include <iostream>
using namespace std;

int main() {
	setlocale(LC_ALL, "Russian");
	Teacher t("Петров", "Иван", 40, 80);
	t.print();
	Worker *w=&t;
	w->print();
	return 0;
}
```

# Результат выполнения



# Конструкторы и деструкторы при наследовании

- Производные классы не наследуют конструкторы базовых классов, но могут их вызвать.
- Конструктор производного класса всегда в начале вызывает конструктор базового класса, чтобы инициализировать унаследованные элементы. Этот вызов может быть явным или неявным.
- При явном вызове конструктор БК записывается в списке инициализаторов элементов.
- Неявный вызов происходит при отсутствии списка инициализаторов элементов или если отсутствует конструктор ПК.
- Деструкторы вызываются в обратном порядке по отношению к вызовам конструкторов (сначала деструктор ПК, потом - БК).

# Конструкторы и деструкторы при наследовании

- При создании объекта производного класса первым вызывается конструктор БК, затем – конструктор элементов ПК, а потом – конструктор самого ПК. Деструкторы вызываются в обратном порядке.
- Конструкторы элементов ПК вызываются в порядке их перечисления в определении ПК.
- При множественном наследовании конструкторы БК вызываются в порядке их объявления в определении ПК. Порядок записи инициализаторов элементов не влияет на последовательность вызовов конструкторов.

# Файл Point.h

```
#ifndef POINT_H
#define POINT_H

class Point {
protected:
    int x,y;
public:
    Point(int =0, int =0);
    ~Point();
};

#endif
```

Файл
Point.cpp

```
#include "Point.h"
#include <iostream>

Point::Point(int a, int b) {
    x=a; y=b;
    std::cout<<"Конструктор Point: ";
    std::cout<<"["<<x<<","<<y<<"]";
    std::cout<<std::endl;
}

Point::~Point() {
    std::cout<<"Деструктор Point: ";
    std::cout<<"["<<x<<","<<y<<"]";
    std::cout<<std::endl;
}
```

# Файл Circle.h

```
#ifndef CIRCLE_H
#define CIRCLE_H

#include "Point.h"

class Circle : public Point {
protected:
    float radius;
public:
    Circle(float =0, int =0, int =0);
    ~Circle();
};

#endif
```

# Файл Circle.cpp

```
#include "Circle.h"

#include <iostream>

Circle::Circle(float r, int a, int b) : Point(a, b) {
    radius=r;
    std::cout<<"Конструктор Circle: ";
    std::cout<<"Радиус = "<<radius<<", Центр: ";
    std::cout<<"["<<x<<","<<y<<"]";
    std::cout<<std::endl;
}
```

# Файл Circle.cpp

```
Circle::~Circle() {
    std::cout<<"Деструктор Circle: ";
    std::cout<<"Радиус = "<<radius<<", Центр: ";
    std::cout<<"["<<x<<","<<y<<"]";
    std::cout<<std::endl;
}
```

Файл
main.cpp

```
#include "Circle.h"

#include <iostream>

int main() {
    setlocale(LC_ALL, "Russian");
    {
        Point p(1, 2);
    }
    Circle circle1(4.5, 7, 9);
    Circle circle2(10, 5, 5);
    return 0;
}
```

# Результат выполнения