

# СТАТИЧНО И ДИНАМИЧНО СВЪРЗВАНЕ. ВИРТУАЛНИ ФУНКЦИИ.

гл.ас., д-р. Нора Ангелова

# СТАТИЧНО И ДИНАМИЧНО СВЪРЗВАНЕ

# СТАТИЧНО СВЪРЗВАНЕ

- ⦿ Изборът на функцията, която трябва да се изпълни става по време на компилация.
- ⦿ Изборът на функцията не може да се променя по време на изпълнение на програмата.

# СТАТИЧНО СВЪРЗВАНЕ

Пример:

Да се дефинира йерархия, определяща точка в равнината, точка в тримерното пространство и точка с цвят в тримерното пространство.

Координатите на точките са цели числа.

# СТАТИЧНО СВЪРЗВАНЕ


- Точка в равнината

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

```
class point2 {
public:
    point2(int x1 = 0, int y1 = 0) {
        x = x1;
        y = y1;
    }

    void print() const {
        cout << x << ", " << y;
    }

private:
    int x, y;
};
```

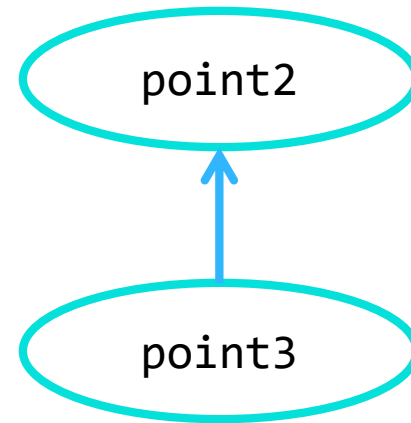


point2

# СТАТИЧНО СВЪРЗВАНЕ

- Точка в тримерното пространство

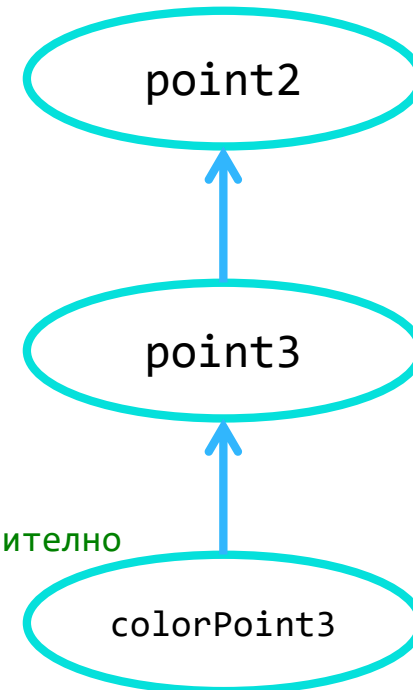
```
class point3 : public point2 {  
    public:  
        point3(int x1 = 0, int y1 = 0, int z1 = 0) : point2(x1, y1) {  
            z = z1;  
        }  
  
        void print() const {  
            point2::print();  
            cout << ", " << z << endl;  
        }  
  
    private:  
        int z;  
};
```



# СТАТИЧНО СВЪРЗВАНЕ

## Точка с цвят в тримерното пространство

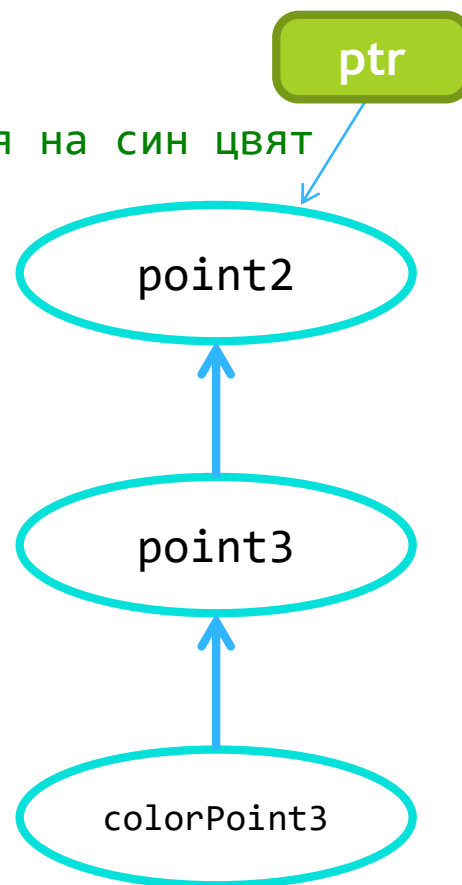
```
class colorPoint3 : public point3 {  
    public:  
        colorPoint3(int x1 = 0, int y1 = 0, int z1 = 0, int c = 0) : point3(x1, y1, z1) {  
            color = c;  
        }  
  
        void print() const {  
            point3::print();  
            cout << "color: " << color << endl;  
        }  
  
        private:  
            int color; // число, което отговаря на цвят от предварително  
                      // дефинирана таблица с цветове  
};
```



# СТАТИЧНО СВЪРЗВАНЕ

- Указатели към клас от йерархията

```
int main() {  
    point2 p2(15, 10);  
    point3 p3(21, 41, 63);  
    colorPoint3 p4(12, 24, 36, 11); // 11 отговаря на син цвят  
  
    point2 *ptr = &p2;  
    ptr->print();  
    cout << endl;  
  
    ptr = &p3; // атрибутът на point2 е public  
    ptr->print();  
    cout << endl;  
  
    ptr = &p4; // атрибутът на point3 е public  
    ptr->print();  
    cout << endl;  
  
    return 0;  
}
```

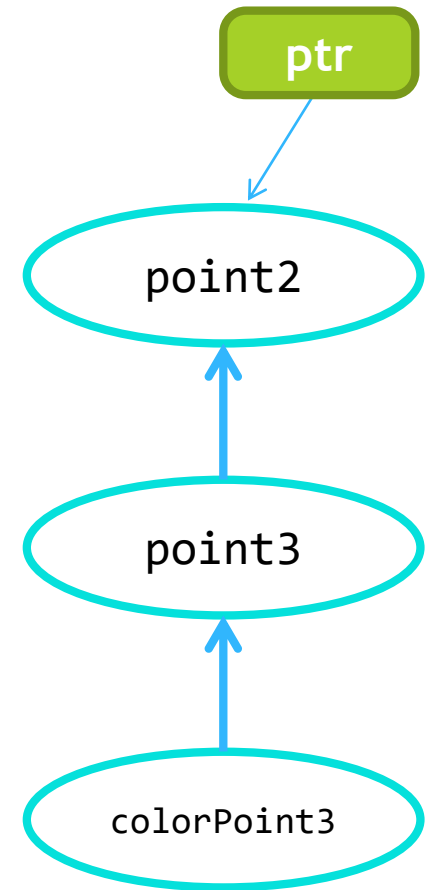




# СТАТИЧНО СВЪРЗВАНЕ

- Указатели към клас от йерархията

```
int main() {  
    point2 p2(15, 10);  
    point3 p3(21, 41, 63);  
    colorPoint3 p4(12, 24, 36, 11);  
  
    point2 *ptr = &p2;  
    ptr->print(); point2::print() - 15, 10  
    cout << endl;  
  
    ptr = &p3; // атрибутът на point2 е public  
    ptr->print(); point2::print() - 21, 41  
    cout << endl;  
  
    ptr = &p4; // атрибутът на point3 е public  
    ptr->print(); point2::print() - 12, 24  
    cout << endl;  
  
    return 0;  
}
```



# СТАТИЧНО СВЪРЗВАНЕ

- Указатели към клас от йерархията

```
int main() {  
    point2 p2(15, 10);  
    point3 p3(21, 41, 63);  
    colorPoint3 p4(12, 24, 36, 11);
```

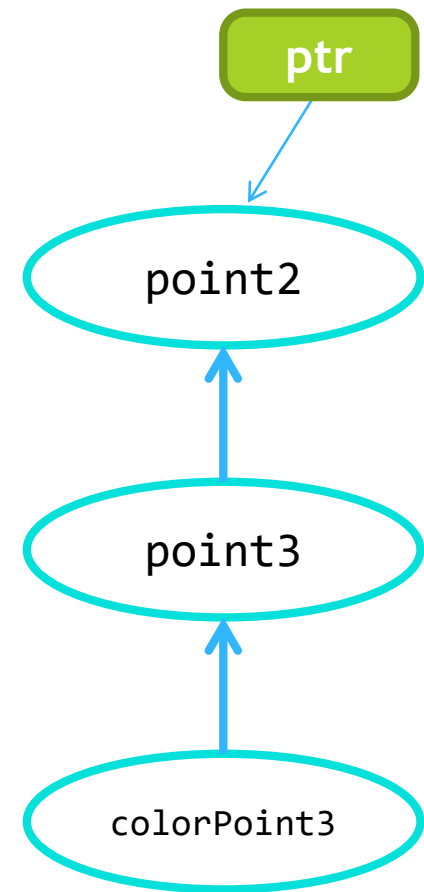
```
    point2 *ptr = &p2;  
    ptr->print(); point2::print() - 15, 10  
    cout << endl;
```

```
    ptr = &p3; // атрибутът на point2 е public  
    ((point3*)ptr)->print(); point3::print() - 21, 41, 63  
    cout << endl;
```

```
    ptr = &p4; // атрибутът на point3 е public  
    ((colorPoint3*)ptr)->print(); colorPoint3::print() - 12, 24, 36  
                                color: 11  
    cout << endl;
```

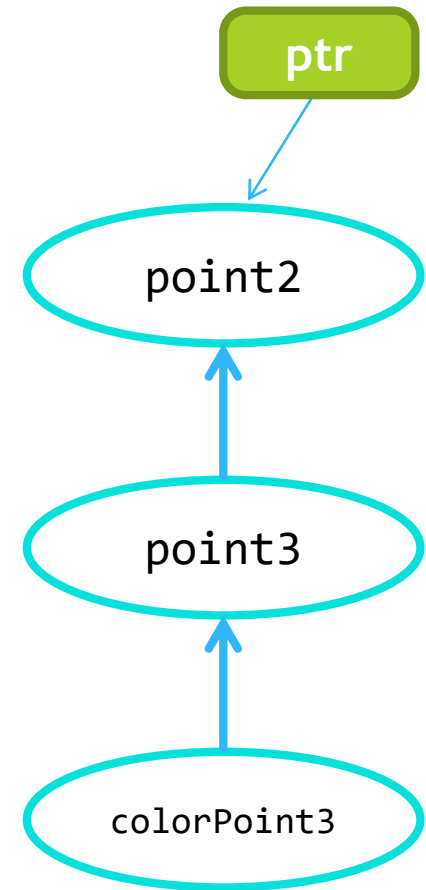
```
    return 0;
```

```
}
```



# СТАТИЧНО СВЪРЗВАНЕ

- За да се извикат правилните функции трябва да се предвидят възможните обекти, указатели и псевдоними на обекти, чрез които ще се извикват член-функциите.
- Какво ще стане при по-сложни йерархии ?!?



# ДИНАМИЧНО СВЪРЗВАНЕ

- ◉ Изборът на функцията, която трябва да се изпълни става по време на изпълнение на програмата.
- ◉ Определянето е в зависимост от типа на обекта.
- ◉ Не се налага явно преобразуване на типове.
- ◉ Опростяват се текстовете на програмите.
- ◉ Разширяването на йерархията не е проблем.
- ◉ Усложняване на кода и забавяне на програмата.
- ◉ Реализира се с виртуални член-функции.

# ДИНАМИЧНО СВЪРЗВАНЕ

- Виртуални функции

`virtual` [<тип\_на\_резултата>] <име\_на\_метод> (<параметри>) [`const`];

думата `virtual` се поставя пред декларацията на функцията

# ДИНАМИЧНО СВЪРЗВАНЕ

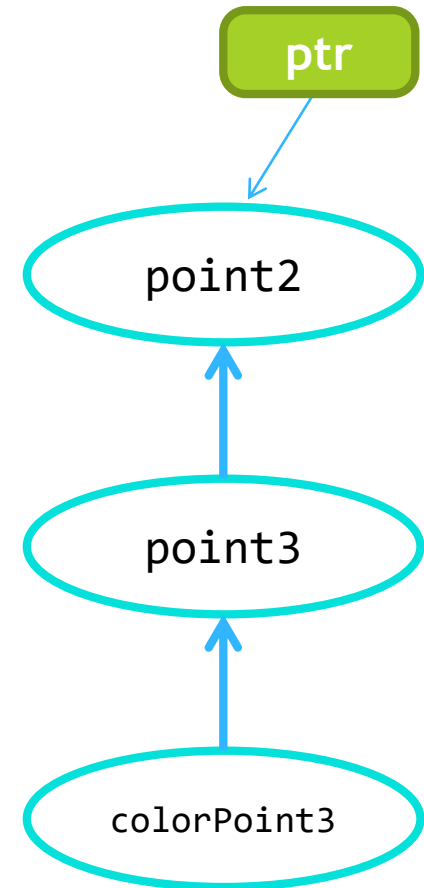
## ◉ Виртуални функции

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

```
class point2 {
public:
    point2(int x1 = 0, int y1 = 0) {
        x = x1;
        y = y1;
    }

    virtual void print() const {
        cout << x << ", " << y;
    }

private:
    int x, y;
};
```



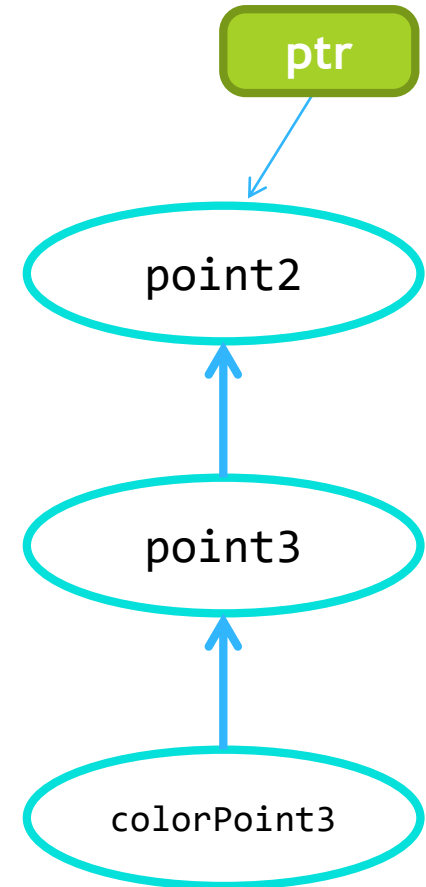
# ДИНАМИЧНО СВЪРЗВАНЕ

## Виртуални функции

```
class point3 : public point2 {  
    public:  
    point3(int x1 = 0, int y1 = 0, int z1 = 0) : point2(x1, y1) {  
        z = z1;  
    }  
};
```

```
virtual void print() const {  
    point2::print();  
    cout << ", " << z << endl;  
}
```

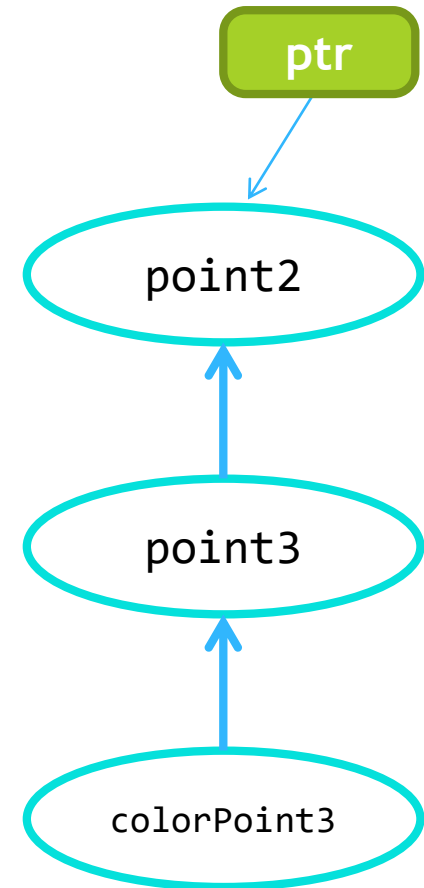
```
private:  
int z;  
};
```



# ДИНАМИЧНО СВЪРЗВАНЕ

## Виртуални функции

```
class colorPoint3 : public point3 {  
    public:  
    colorPoint3(int x1 = 0, int y1 = 0, int z1 = 0, int c = 0) : point3(x1, y1, z1) {  
        color = c;  
    }  
  
    virtual void print() const {  
        point3::print();  
        cout << "color: " << color << endl;  
    }  
  
    private:  
    int color; // число, което отговаря на цвят от предварително  
              // дефинирана таблица с цветове  
};
```

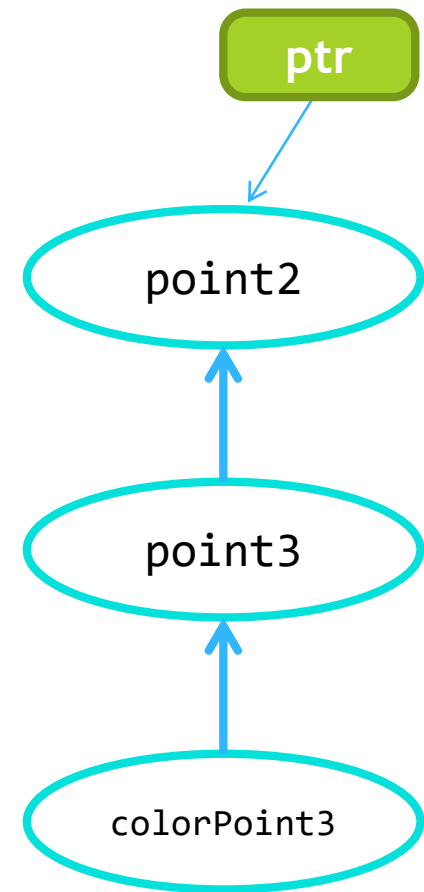




# ДИНАМИЧНО СВЪРЗВАНЕ

- Указатели към клас от йерархията

```
int main() {  
    point2 p2(15, 10);  
    point3 p3(21, 41, 63);  
    colorPoint3 p4(12, 24, 36, 11);  
  
    point2 *ptr = &p2;  
    ptr->print(); point2::print() - 15, 10  
    cout << endl;  
  
    ptr = &p3; // атрибутът на point2 е public  
    ptr->print(); point3::print() - 21, 41, 63  
    cout << endl;  
  
    ptr = &p4; // атрибутът на point3 е public  
    ptr->print(); colorPoint3::print() - 12, 24, 36  
                  color: 11  
    cout << endl;  
  
    return 0;  
}
```



# ДИНАМИЧНО СВЪРЗВАНЕ

- Декларирането на член-функциите print като виртуални причинява трите обръщания към print чрез указателя ptr да определят функцията, която ще бъде извикана по време на изпълнението на програмата.
- Определянето е в зависимост от типа на обекта.

# ДИНАМИЧНО СВЪРЗВАНЕ

- В случая `ptr = &p3`, указателят `ptr` е от класа `point2`, но сочи обекта `p3`, който е от класа `point3`.  
Затова обръщението `ptr->print()` ще изпълни `point3::print()`, ако е възможен достъп.
- В случая `ptr = &p4`, указателят `ptr` сочи обекта `p4`, който е от класа `colorPoint3`. Затова обръщението `ptr->print()` ще изпълни `colorPoint3 ::print()`, ако е възможен достъп.

Как се определя достъпът?

Достъпът се определя в зависимост от типа на указателя!

И в двата случая достъпът е възможен, тъй като `ptr` е от тип `point2*`, а в този клас член-функцията `print` е в `public` секцията му.

# ДИНАМИЧНО СВЪРЗВАНЕ

## ◎ Свойства

1. Само член-функциите на класовете могат да се декларират като виртуални. Конструкторите не могат да се декларират като виртуални.
2. Ако функция е обявена за виртуална в основния клас, декларираните член-функции в производните класове със същия прототип **също са виртуални дори ако запазената дума бъде пропусната**.
3. Ако в производен клас се дефинира виртуална функция, която има същия прототип като неvirtуална функция в основния клас, **двете функции се интерпретират като различни** член-функции.
4. Възможно е виртуална функция да се дефинира извън клас. Тогава не започва със запазената дума `virtual`.
5. Виртуалните член-функции се наследяват като останалите компоненти на класа.
6. Основния клас, в който член-функция е обявена за виртуална, трябва да е с атрибут `public` в производните от него класове.
7. Виртуалните член-функции се извикват чрез указател или псевдоним на обект на някакъв клас.
8. Виртуалната член-функция, която в действителност **се изпълнява, зависи от класа на обекта**, към който сочи указателят.
9. Локалният и външният достъпът до виртуална член-функция имат някои особености.

# ДОСТЪП ДО ВИРТУАЛНА ЧЛЕН-ФУНКЦИИ

- Пряк достъп (вътрешен, локален)

Виртуална член-функция на производен клас има пряк достъп до:

- собствените на производния клас компоненти;
- компонентите, декларирани в public и protected секциите на основните си класове.

Всяка член-функция на клас, в който е дефинирана виртуална член-функция, има пряк достъп както до виртуалната член-функция на самия клас, така и до виртуалните член-функции със същия прототип на производните му директни и индиректни класове без значение на вида на секциите, в които са дефинирани виртуалните член-функции.

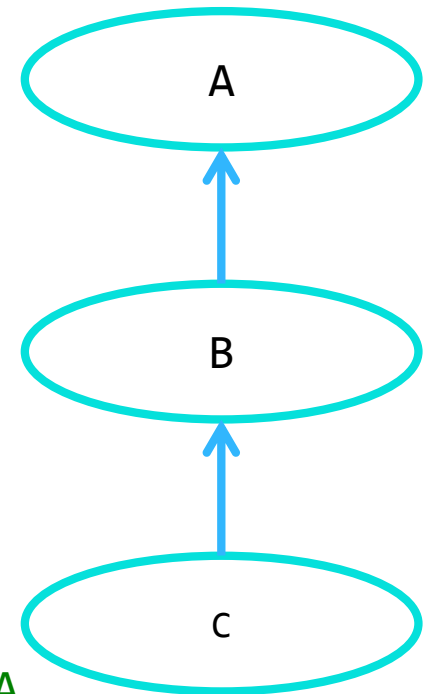
# ДОСТЪП ДО ВИРТУАЛНИ ЧЛЕН-ФУНКЦИИ

## ◎ Пряк достъп (вътрешен, локален)

```
class A {  
    private:  
    virtual void f() const {  
        cout << "A\n";  
    }  
  
    public:  
    void test() const {  
        f(); // разрешава се динамично  
    }  
};
```

```
class B : public A {  
    protected:  
    void f() const {  
        cout << "B\n";  
    }  
};  
  
class C : public B {  
    private:  
    virtual void f() const {  
        cout << "C\n";  
    }  
};
```

```
int main() {  
    A a; B b; C c;  
    A *ptr = &a;  
    ptr->test(); // A  
  
    ptr = &b;  
    ptr->test(); // B  
  
    ptr = &c;  
    ptr->test(); // C  
  
    return 0;  
}
```



# ДОСТЪП ДО ВИРТУАЛНА ЧЛЕН-ФУНКЦИИ

## ◉ Външен достъп

1. Външният достъп до виртуална член-функция на клас чрез обект на класа се определя по традиционните правила. Връзката се разрешава статично.
2. Външен достъп чрез указател

```
клас1* ptr = &обект;  
ptr -> виртуална_функция_на_клас(...);
```

Ако *виртуална\_функция\_на\_клас1* е в **public** секция в *клас1*.

3. Външен достъп чрез псевдоним

```
клас1& ptr = обект;  
ptr. виртуална_функция_на_клас(...);
```

Ако *виртуална\_функция\_на\_клас1* е в **public** секция в *клас1*.

КРАЙ