# Тема 4. Полиморфизъм и виртуални функции.

## (множествено наследяване)

# ПОЛИМОРФИЗЪМ И ВИРТУАЛНИ ФУНКЦИИ

***Дефиниции:***

***Полиморфизъм*** означава "многообразие".

Полиморфизъмът се проявява когато няколко дефинирани класа споделят общ интерфейс, защото са наследници на един и същ базов клас(ове). Имплементациите могат да се различават, защото класовете са различни. Обектите на тези класове се използват по еднакъв начин, тъй като те споделят общия интефейс.

# ПОЛИМОРФИЗЪМ И ВИРТУАЛНИ ФУНКЦИИ

***Дефиниции:***

***Виртуална функция*** е член функция на класа, адресът на която не е фиксиран до момента на изпълнение.

Тази техника е известна също и като “**късно свързване**”.

За разлика от другите функции, обръщението към виртуална функция винаги е свързано с реализацията на тази функция за даден обект, дори и когато обектът е достъпен чрез указател (референция).

# ПОЛИМОРФИЗЪМ И ВИРТУАЛНИ ФУНКЦИИ

Разлики между виртуална и невиртуална:

Невиртуална функция:

- Създават се няколко реализации на една и съща функция;
- Типът (класът) на обекта се определя по време на компилация и е еднозначен (един и същ);
- При обръщение чрез указател не могат да възникнат недетерминираности (невъзможност да се определи адреса).

# ПОЛИМОРФИЗЪМ И ВИРТУАЛНИ ФУНКЦИИ

Разлики между виртуална и невиртуална:

Виртуална функция:

- При динамичното (късно) свързване (late binding) методът, който ще се извика, се определя по време на изпълнение;
- Извиква се методът на този клас, от който всъщност е даденият обект;
- Това е така независимо че указателят може да е дефиниран към базов клас.

# ПОЛИМОРФИЗЪМ И ВИРТУАЛНИ ФУНКЦИИ

Пример:

Програма за създаване и изобразяване на прости геометрични фигури:

- Програмата трябва да осигурява изобразяването на множество от примитивни графични обекти като например окръжност, правоъгълник и квадрат;
- Потребителят на класа създава желаните обекти и извиква функциите за да се изобразят, изтрият или преместят.

# ПОЛИМОРФИЗЪМ И ВИРТУАЛНИ ФУНКЦИИ

Особености:

- Всички графични обекти поддържат еднакво множество от операции - изтриване и преместване.
- Начинът на имплементация на операцията изобразяване е различен за всеки един от тях.

# ПОЛИМОРФИЗЪМ И ВИРТУАЛНИ ФУНКЦИИ

Имплементация:

Дефинира се клас CGraphicalObject:

- Предоставя общ интерфейс, споделян от всички графични обекти-функциите, които имплементират общите операции;
- Общ за всички фигури е центърът, спрямо който се изобразяват, представен чрез класа CPoint;
- CGraphicalObject се наследява от класове, които представляват конкретните фигури-кръг, правоъгълник и квадрат.

# ПОЛИМОРФИЗЪМ И ВИРТУАЛНИ ФУНКЦИИ

*Пример за инплементация*:

```
class CPoint {
int m_ix, m_iy;
public:
        CPoint(void);
        ~CPoint(void);
       CPoint( int xarg, int yarg);
        CPoint(const CPoint& xy);
// четене на координата ix
       int Getx(void) const;
// четене на координата iy
       int Gety(void) const;
// Предекларация на оператора за присвояване
       const CPoint& operator =(const CPoint& xy) {
         m_ix = xy. m_ix;
         m_iy = xy. m_iy;
         return *this;
       }
};
```

# ПОЛИМОРФИЗЪМ И ВИРТУАЛНИ ФУНКЦИИ

*Пример (продължение):*

```
class CGraphicalObject {
protected:
  CPoint m_center; // вграден обект
  CGraphicalObject( CPoint const& p ) :
 m_center(p) {}
public:
  virtual void Draw() const = 0;
  virtual void Erase();
  virtual void MoveTo(CPoint const& p ) ;
  void SetPenColor(unsigned int uiColor);
};
```

# ПОЛИМОРФИЗЪМ И ВИРТУАЛНИ ФУНКЦИИ

Имплементация на Erase: като се промени цветът на маркера с този на фона на изобразяващата повърхност и преизобразяване.

```
void CGraphicalObject::Erase(){
unsigned int backgroundColor=0;
unsigned int foregroundColor=0xffff;
  SetPenColor(backgroundColor);
  Draw();
  SetPenColor(foregroundColor);
}
void CGraphicalObject::SetPenColor(unsigned int uiColor) {
// имплементация в зависимост от използваната графична среда
}
void CGraphicalObject::MoveTo(CPoint const& center ) {
  Erase();
  m_center = center;
  Draw();
}
```

# ПОЛИМОРФИЗЪМ И ВИРТУАЛНИ ФУНКЦИИ

```
// клас наследник на CGraphicalObject
class CCircle : public CGraphicalObject {
int m_iradius;
public:
    CCircle( CPoint const& p, int r ) :
    CGraphicalObject(p), m_iradius(r) {}
    void Draw() const;
};
```

# ПОЛИМОРФИЗЪМ И ВИРТУАЛНИ ФУНКЦИИ

```
// клас наследник на CGraphicalObject
class CRectangle : public CGraphicalObject
{
int m_iheight;
int m_iwidth;
public:
  CRectangle( CPoint const& p, const int h, const
      int w ) :
  CGraphicalObject(p), m_iheight(h),
      m_iwidth(w) {}
  void Draw() const;
};
```

# ПОЛИМОРФИЗЪМ И ВИРТУАЛНИ ФУНКЦИИ

```
// клас наследник на CRectangle
class CSquare : public CRectangle {
public:
        CSquare( CPoint const& p, const int w ) :
        CRectangle( p, w, w ) {}
};
void CRectangle::Draw() const {
        // конкретна имплеметация на
        // изобразяването на правоъгълника
}
void CCircle::Draw() const {
        // конкретна имплеметация на
        // изобразяването на окръжност
}
```

# ПОЛИМОРФИЗЪМ И ВИРТУАЛНИ ФУНКЦИИ

Примери за създаване на обекти:

```
CCircle oc (CPoint (0, 0), 5);
oc.Draw ();
oc.MoveTo (CPoint (10, 10));
oc.Erase ();
CSquare os(CPoint (0, 0), 5);
os.Draw ();
os.MoveTo (CPoint (10, 10));
os.Erase ();
CRectangle or(CPoint (0, 0), 5, 10);
or.Draw ();
or.MoveTo (CPoint (10, 10));
or.Erase ();
```

# ПОЛИМОРФИЗЪМ И ВИРТУАЛНИ ФУНКЦИИ

**Разлика между невиртуални (предефинирани) и виртуални функции**

- Не-виртуални член функции са определени **статично-функцията се избира статично** (по време на компилация) на базата на типа на указателя (или референцията) на обекта;
- Виртуалните член функции се определят динамично (по време на изпълнение) - **член функцията се избира динамично** (по време на изпълнение) в зависимост от типа на обекта, а не от типа на указателя (или референцията) към този обект "**динамично свързване**"

# ПОЛИМОРФИЗЪМ И ВИРТУАЛНИ ФУНКЦИИ

## Имплементация на виртуални функции

- Повечето компилатори използват някакъв вариант на следната техника - ако обектът има една или повече виртуални функции, компилаторът поставя скрит указател в обекта, наречен "**виртуален указател**" (v-pointer) Този "v-pointer" сочи глобална таблица, наречена "**виртуална таблица**" (V-table);
- Компилаторът създава V-table за всеки клас, който има най-малко една виртуална функция;
- По време на изпълнение става зареждане на указателя на функцията от обекта във виртуалната таблица на класа (в примера тройката функции), след което от V-таблица определя кода на метода за извикване.

# ПОЛИМОРФИЗЪМ И ВИРТУАЛНИ ФУНКЦИИ

**Имплементация на виртуални функции**

**За примера:**

класът Circle има 3 виртуални функции - Draw(), MoveTo() и Erase();

- Има една (глобална) таблица, свързана с класа кръг с V-указатели за всяка виртуална функция;
- V-таблицата на класа ще има три указателя към функции:
  - указател към Circle::Draw();
  - указател към Circle::MoveTo();
  - указател към  Circle::Erase();

# ПОЛИМОРФИЗЪМ И ВИРТУАЛНИ ФУНКЦИИ

**Виртуални член функции**

*Дефиниция:Член функция на класа, започваща с ключовата дума* **virtual** *определя тази функция като* ***виртуална член функция***

***Функция, представена чрез адрес.***

# ПОЛИМОРФИЗЪМ И ВИРТУАЛНИ ФУНКЦИИ

Draw, Erase и MoveTo са декларирани като виртуални член функции на класа CGraphicalObject ( **virtual** ):

- Декларирайки член функция като виртуална се променя начинът, по който се определя коя член функция да се изпълни.

# ПОЛИМОРФИЗЪМ И ВИРТУАЛНИ ФУНКЦИИ

Пример:

```
CGraphicalObject& g1 = *new CCircle (CPoint (0,0), 5);
CGraphicalObject& g2 = *new CSquare (CPoint (0,0), 5);
g1.Draw ();
g2.Draw ();
```

Ако функцията Draw не е декларирана като виртуална, тогава и g1.Draw() и g2.Draw() биха изпълнили CGraphicalObject::Draw. Но тъй като Draw е виртуална функция, g1.Draw() изпълнява CCircle::Draw, а g2.Draw() изпълнява CRectangle::Draw.

**Забележка: Класът CSquare няма предефинирана функция Draw,поради което се търси в базовия му клас**

- **CRectangle**

# ПОЛИМОРФИЗЪМ И ВИРТУАЛНИ ФУНКЦИИ

Действие при създаване на обект от наследника на абстрактен клас:

I. Създаване на базовия клас:

- Създаване на инстанция на виртуалната таблица на базовия клас __vfptr;
- Създаване на инстанциите на член променливите на базовия клас;
- Изпълнение на тялото на конструктора на базовия клас.

# ПОЛИМОРФИЗЪМ И ВИРТУАЛНИ ФУНКЦИИ

II. Създаване на наследник (ци):

- Създаване на инстанция на виртуалната таблица на наследника __vfptr;
- Създаване на инстанциите на член променливите на наследника;
- Изпълнение на тялото на конструктора на наследника.

# ПОЛИМОРФИЗЪМ И ВИРТУАЛНИ ФУНКЦИИ

Действие при унищожаване на обект:

I. Унищожаване на наследник (ци):

- Изпълнение на деструктора на наследника;
- Унищожаване на променливите на наследника - стартиране на деструкторите им.

# ПОЛИМОРФИЗЪМ И ВИРТУАЛНИ ФУНКЦИИ

II. Унищожаване на базовия клас (рекурсивно):

- Изпълнение на деструктора на базовия клас;
- Унищожаване на променливите на базовия клас - стартиране на деструкторите им.

# ПОЛИМОРФИЗЪМ И ВИРТУАЛНИ ФУНКЦИИ

“Правилността” на извикването на член функция се осигурява, независимо от това как се осъществява достъпа към обекта.

Пример:

```
CSquare s (CPoint (0,0), 5);
CRectangle& r = s;
CGraphicalObject& g = r;
```

- Тук s, r и g реферират към един и същ обект, въпреки че са от различен клас. Поради това че Draw е декларирана като виртуална функция, s.Draw(), r.Draw() и g.Draw() ще изпълнят CRectangle::Draw

# ПОЛИМОРФИЗЪМ И ВИРТУАЛНИ ФУНКЦИИ

Абстрактни класове и конкретни класове

*Дефиниции*:

***Напълно (чисто) виртуална функция*** е функция на клас, която има имплементация 0 (нула). Тази нулева имплементация се задава чрез спецификацията = *0* в дясната част на функцията (вместо тяло).

В C++ ***абстрактен клас*** е такъв клас, който дефинира интерфейс, но не задава задължително имплементации на всичките си член функции:

- Абстрактен клас е клас, който има поне една ***напълно виртуална функция***.

# ПОЛИМОРФИЗЪМ И ВИРТУАЛНИ ФУНКЦИИ

- **Абстрактният клас** е предназначен да бъде използван като базов клас, от който се правят производни класове. Очаква се производният клас да предостави имплементации на функциите, които не са имплементирани в базовия клас.
- **Производен клас,** който имплементира цялата липсваща функционалност на базовия се нарича ***конкретен клас*** .

# ПОЛИМОРФИЗЪМ И ВИРТУАЛНИ ФУНКЦИИ

Например, класът CGraphicalObject е абстрактен клас:

- Няма имплементация за виртуалната член функция Draw. Фактът, че не е зададена имплементация се вижда от =0 следващо прототипа на функцията Draw в дефиницията на класа;
- Обектът съдържа указател към съответстващата функция за всяка виртуална такава.
- Когато има нулева спецификация ( =0 ) това означава, че указателя към функцията Draw не е дефиниран в класа CGraphicalObject и затова той трябва да бъде дефиниран в негов производен клас.

# ПОЛИМОРФИЗЪМ И ВИРТУАЛНИ ФУНКЦИИ

В C++ **не е възможно** да се

създаде обект от абстрактен клас!!!!!

Например, следната декларация е невалидна:

```
CGraphicalObject g(CPoint (0,0)); //error!!!
```

# ПОЛИМОРФИЗЪМ И ВИРТУАЛНИ ФУНКЦИИ

Причини:

1. Ако е възможно да се декларира g по този начин, след това би могла да се извика несъществуващата член функция g.Draw().
2. Конструкторът на класа CGraphicalObject не е дефиниран като публичен (public).

# ПОЛИМОРФИЗЪМ И ВИРТУАЛНИ ФУНКЦИИ

**Ефективност:**

**Памет:** допълнително указател към обекта (но само за обекти, които изискват задължително динамично свързване), плюс допълнителен указател към функция (но само за виртуалните функции).

**Бързодействие:**

Времето се увеличава допълнително- 15 до 50%:

- За извличане на указател, състоящо се в:
  - получаване на стойността на V-указателя;
  - получаване адреса на член функцията.

Тези действия не се осъществяват с не-виртуални функции, тъй като компилатора ги определя по време на компилация на основата на типа на указателя.

# АБСТРАКТНИ АЛГОРИТМИ

**Алгоритмична абстракция**

*Дефиниция*:

Една от от най-полезните идеи (парадигми) на обектното програмиране е използването на абстрактни класове за ***алгоритмична абстракция***.

# АБСТРАКТНИ АЛГОРИТМИ

В примера:

Функциите Erase и MoveTo дефинирани в примера са ***алгоритмична абстракция***, тъй като осигуряват общ алгоритъм за всички наследници на базовия клас:

- Функциите Erase, SetPenColor и MoveTo са имплементирани в абстрактния клас CGraphicalObject;
- Имплементираните алгоритми са предвидени да работят с всяка конкретна имплементация на клас, призводен на CGraphicalObject, независимо дали е CCircle, CRectangle или CSquare;
- Представените функции работят независимо от реалния клас на обекта. Затова такива функции и съответните им алгоритми се наричат ***абстрактни алгоритми***.

# АБСТРАКТНИ АЛГОРИТМИ

**Абстрактните алгоритми най-често са виртуалните член функции.**

В примера:

Функцията **MoveTo** изпълнява **Erase и Draw**, за да извърши по-голямата част от конкретната работа:

- Производните класове наследяват абстрактния алгоритъм MoveTo;
- Предефинират напълно виртуалната функция Draw.

# АБСТРАКТНИ АЛГОРИТМИ

**Действие:**

- Производният клас променя поведението на абстрактния алгоритъм **чрез предефиниране на съответните виртуални член функции;**
- Механизъмът за определяне на адреса на виртуалната функция осигурява извикването на съответната “вярна” виртуална член функция, **чрез динамична промяна на нейния адрес по време на изпълнение.**

# МНОЖЕСТВЕНО НАСЛЕДЯВАНЕ

Всеки клас на C++ може да бъде производен на един или повече базови класове. В този случай всички базови класове трябва да са различими, т.е. следната декларация не е разрешена:

```
class D : public B, public B { ... };
// грешка.
```

Въпреки това е възможно един и същ базов клас да се *наследи индиректно* повече от веднъж.

# МНОЖЕСТВЕНО НАСЛЕДЯВАНЕ

Пример, дадени са следните декларации на класове:

```
class A { ... };
class B1 : public A { ... };
class B2 : public A { ... };
class D : public B1, public B2 { ... };
```

Производният клас D наследява две инстанции на базовият клас A:

- една индиректно през B1;
- другата, също индиректно през B2.

Това означава че D съдържа две копия на всяка член променлива на клас A.

# МНОЖЕСТВЕНО НАСЛЕДЯВАНЕ

**Възниква въпрос:**

**Когато бъде извикана член функция на A?**

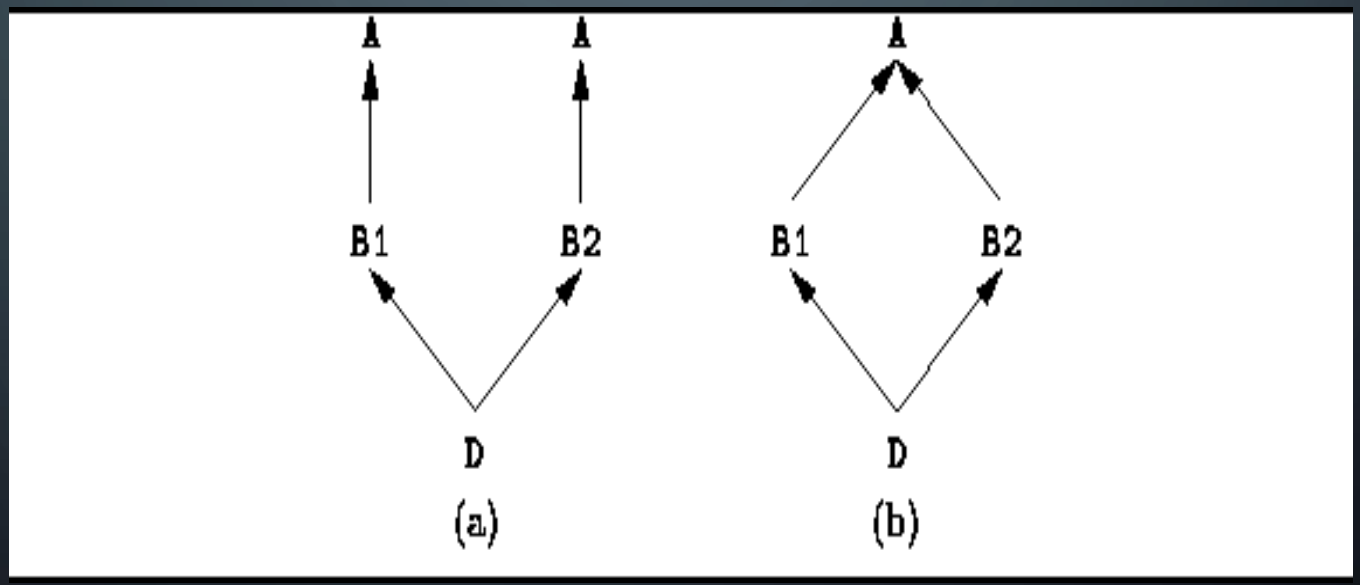
**Коя от двете инстанции на A ще се използва?**

- **Тази, която е наследил B1?**
- **Или тази от B2 ?**
- **Как програмно може да се декларира?**

# МНОЖЕСТВЕНО НАСЛЕДЯВАНЕ

**Видове множествено наследяване:**

- **невиртуални базови класове (a);**
- **виртуални базови класове (b)**



# МНОЖЕСТВЕНО НАСЛЕДЯВАНЕ

В C++ се допускат ***виртуални базови класове*** (b):

```
class A { ... };
class B1 : virtual public A { ... };
class B2 : virtual public A { ... };
class D : public B1, public B2 { ... };
```

- Производният клас D съдържа само една инстанция на базовия A;
- D съдържа само едно копие на член променливите на A;
- Няма двусмислие при извикването на член функциите на A.

# МНОЖЕСТВЕНО НАСЛЕДЯВАНЕ

Пример за можествено наследяване:

```
class CPolygon
{
protected:
	int m_iwidth, m_iheight;
public:
	void SetValues (int w, int h){
		m_iwidth=w;
		m_iheight=h;
	}
};
class CPrint
{
public:
	void printing (int value){    cout << value << endl;}
};
```

# МНОЖЕСТВЕНО НАСЛЕДЯВАНЕ

Пример за множествено наследяване (продължение):

```
class CRectangle: public CPolygon, public CPrint
{
public:
    int area ()    { return (m_iwidth * m_iheight);}
};

class CTriangle: public CPolygon, public CPrint
{
public:
    int area ()    {return (m_iwidth * m_iheight / 2);   }
};
```

# МНОЖЕСТВЕНО НАСЛЕДЯВАНЕ

```
int main ()
{
    CRectangle rectangle;
    CTriangle triangle;
    rectangle.SetValues (2,2);
    triangle.SetValues (2,2);
    rectangle.printing (rectangle.area());
    triangle.printing (triangle.area());
    return 0;
}
```

# Въпроси?