Наследяване

Въведение в тестването:

is-a realationships, създаане на йерархични структури, спестяване на код,

Достъп до член-данните на клас:

https://www.tutorialspoint.com/cplusplus/cpp_inheritance.htm

Access	public	protected	private
Same class	yes	yes	yes
Derived classes	yes	yes	no
Outside classes	yes	no	no

Какво се наследява?

Производният клас наследява всички методи на базовия клас със следните изключения:

- *Конструктор, деструктор и copy конструктор*
- *Предефинирани оператори*
- *Приятелски функции на базовия клас*

Public, Private и Protected наследяване:

Важно е да не се бъркат трите вида наследяване с идентификаторите за достъп на компоненти на клас.

```
class Derivative : public Base {
    //...
}
class Derivative : private Base {
    //...
}
class Derivative : protected Base {
    //...
}
```

В случая на английски им казват *access-specifier*, на български може да го срещнете *атрибут за област*.

При изпускане на *атрибута за област* пред някой от базовете класове в декларацията на производен клас, се подразбира, че се използва *private* атрибут.

```
class Derivative : Base {
    //...
}
```

е еквивалентно на:

```
class Derivative : private Base {
    //...
}
```

Достъп до наследените компоненти:

Атрибут за област:	Компонента на основен клас:	Наследавя се като:
public	public, protected, private	public, protected, private
private	public, protected, private	private, private, private
protected	public, protected, private	protected, protected, private

Не е много ОК да се забравя *access-specifier* на класа, като виждаме че по този начин няма да можем да достъпваме никакви компоненти на производния клас. Затова не бива да се забравя най-често използваният атрибут *public* в декларацията на производен клас.

Ще ги наричам основни, базови, родители, base.

Другите ще ги наричам наследници, производни, деца, derived.

В C++ съществува и множествено наследяване.

Данните, които са дефинирани за производния клас се наричат собствени.

Атрибутът за достъп *protected* - достъп до член-данни и методи на класа имат само вътрешни методи на класа, приятелски функции и наследници.

Тук трябва да се акцентира на следното: компонентите на класа наследник има достъп до *public* и *protected* наследени компоненти, а до *private* наследени компоненти нямат.

Най-често атрибута за област, ако е *protected* или *private* се използва, когато не трябва да се използва интерфейсът на базовия клас, а се налага да се предефинират всички негови интерфейсни функции.

Процес на наследяване:

- наследяват се член-данните и методите на основните класове;
- получава се достъп до някои от наследените компоненти на основните класове;
- производният клас "познава" реализациите само на основните класове, от които произлиза
- производният клас може да е основен за други класове

Производният клас може да дефинира допълнително:

- свои член-данни
- свои член-функции (методи), аналогични на тези на основните класове, а също и нови

Може *основни* класове да са *директни* или *индиректни*:

- *директни* - тези, които се изброяват в заглавието на производния клас, предшествани от двуеточие `:`;
- *индиректни* - те са *основни* за *директните основни* класове на наследника;

Правила за достъп при наследяване:

- **При достъп на компоненти на производен клас до компоненти на основен клас:**
  - Методите на производен клас (без значение атрибута на област) нямат директен достъп до *private* компонентите на основния клас;
  - Методите на производен клас (бз значение атрибута на област) имат пряк достъп до *public* и *protected* компонентите на основния клас;
- **При достъп на обекти до компонентите на основния и производния клас:**
  - Обект на основен клас има пряк достъп до всички свои компоненти, обявени като *public* и няма пряк достъп до компонентите, обявени като *private* и *protected*.
  - Обект на производен клас има пряк достъп до *public* компонентите на класа, от който произлиза

Наследяване и copy конструктор

Три случая:

Винаги, когато се работи с динамична памет в класа е необходимо да се декларира и дефинира copy конструктор за класа

Имаме следните варианти:

- В базовия клас имаме дефиниран copy конструктор, а в производния клас **нямаме** дефиниран - компилаторът генерира конструктор за копиране на производният клас, който преди да се изпълни, активира и изпълнява copy конструктора на базовия клас. (Това е изключение, понеже при обикновените конструктори не работеше по този начин освен ако нямаме зададен конструктор по подразбиране);
- В базовия клас **нямаме** дефиниран copy конструктор и в производния клас **нямаме** дефиниран - в този случай и в базовия, и в производния клас се генерират от компилатора copy конструктори и този на производния преди да бъде изпълнен активира този на базовия клас;
- В производния клас имаме дефиниран copy конструктор - copy конструкторът на производния клас определя как ще се инициализира наследената част. В неговия инициализиращ списък може да има или да няма обръшение към конструктор (copy или обикновен) на базовия клас. Препоръчва се в инициализиращия спписък на производния клас да им обръщение към конструктора за присвояване на основния клас, ако такъв е дефиниран. Ако не е указано обръщение към конструктор на основния клас, инициализирането на наследените членове се осъществява от default-ния конструктор на базовия клас. Ако няма такъв дефиниран се съобщава за грешка.

Наследяване и operator=

Два случая:

Винаги, когато се работи с динамична памет в класа е необходимо да се декларира и дефинира operator= за класа. Ако не се направи компилаторът създава собствен

Имаме следните варианти:

- В производния клас не е дефиниран оператор= - компилаторът създава операторна функция за присвояване в производния клас. Тя изпълнява операторната функция за присвояване на базовия клас (дефинираната или създадената от компилатора).
- В производния клас е дефинирана оператор= - тя трябва да се погрижи за присвояването на наследените компоненти. Налага се в тялото на нейната дефиниция да се извика дефинирания оператор равно на производния клас, ако има такъв.