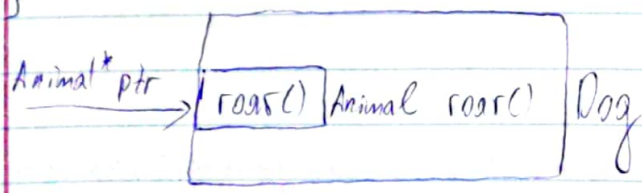
Тема 11

Статично и динамично свързване. Виртуални функции. Полиморфизъм. Абстрактни класове.

Нека разгледаме следния пример:

```
struct Animal
{
  void roar() const
  {
    std::cout<<"I am an animal"<<std::endl;
  }
};
struct Dog : Animal
{
  void roar() const
  {
    std::cout<<"Bow, bow"<<std::endl;
  }
};
int main()
{
  Dog d;
  d.roar(); // Bow, bow
  Animal* ptr = &d;
  ptr->roar(); // I am an animal
}
```



Картинка с илюстративна цел! Функцията не "живее" в класа/структурата

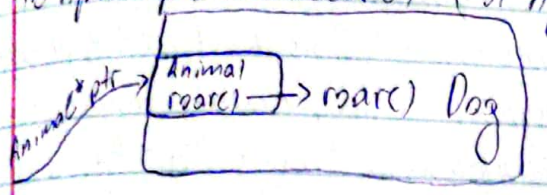
Имаме структура Animal, която дефинира функция roar(). Създаваме структура Dog, която наследява Animal и също дефинира функция roar(), но тя има различно поведение, т.е в случая предефинираме roar().

Този пример илюстрира така нареченото статично свързване. При него изборът на функцията, която трябва да се изпълни, става по време на

компилацията определя се директно от типа на указателя/референцията.

Съществува и динамично свързване. При него изборът на функцията, която трябва да се изпълни, се случва по време на изпълнение на програмата. Това се постига чрез така наречените виртуални функции. Те се декларират чрез запазената дума virtual пред типа на функцията.

Ако направим функцията roar() виртуална (virtual void roar() const), то при ptr->roar() (от предишния пример) вече се отпечатва "Bow, bow!"

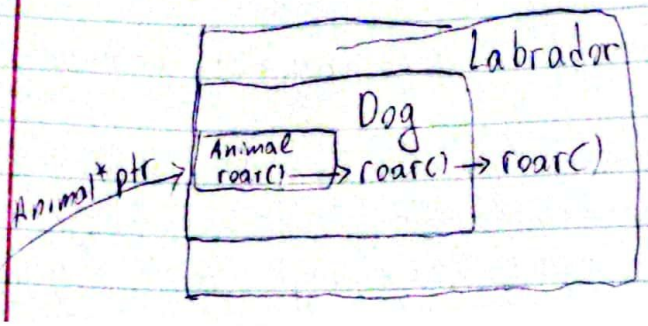


Картинка с илюстративна цел!

Това, което се случва е: указателят Animal* ptr извиква функцията roar(), но тъй като тя е виртуална, започва да търси някоя "по-конкретна" функция roar(). Установява, че класът Animal е част от по-голям клас Dog, който също дефинира функцията roar(). Въпреки че указателят е от тип Animal се извиква функцията roar() на класът Dog, защото присвоената стойност на ptr е от тип Dog (Animal* ptr = &d (Dog d))

Предефинираната виртуална функция в наследника също е виртуална. Тоест функцията roar() в класа Dog е виртуална.
Нека имаме следния клас:

```
class Labrador: public Dog
{
  void roar() const
  {
  std::cout << "I am a labrador";
  }
};
int main(){
Labrador labrador;
Animal* ptr2 = &labrador;
ptr2->roar(); → извиква функция roar() на клас Labrador, заради
                динамичното свързване
```

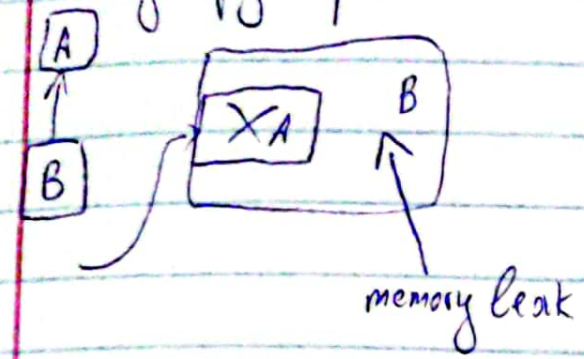


Илюстративна уел!

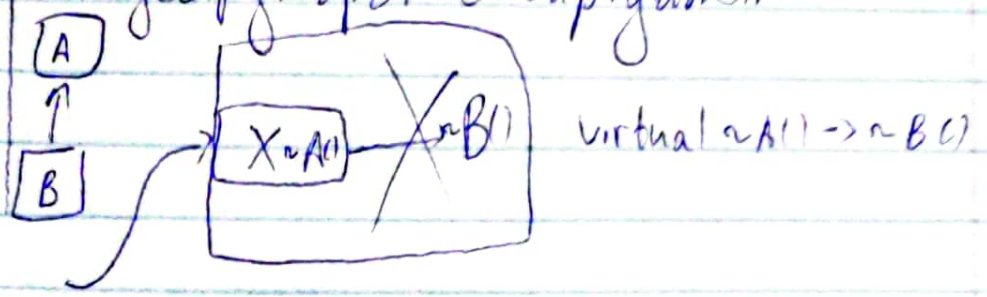
Един от основните принципи на ООП е полиморфизъм - едни и същи действия се реализират по различен начин в зависимост от обектите, върху които се прилагат. Действията се наричат полиморфни. Полиморфизмът се реализира чрез виртуални функции. Класовете, върху които ще се прилага чрез да имат общ родител или прародител, т.е. да са наследници на един и същ клас. В клас се дефинира виртуален метод, съответстващ на полиморфното действие. Всеки клас предефинира или не виртуалния метод. Активирането става чрез указател към базовия клас, на който може да се присвоят адресите на обекти на който и да е от базовите класове от йерархията.

Много важно: при полиморфна йерархия унищожаване обектите чрез указател от базовия клас. За да се избягват правилните деструктори, трябва деструкторът на базовия клас да е деклариран като виртуален!

Ако деструкторът не е виртуален



Ако деструкторът е виртуален



правилно поведение

Чисто виртуална функция (Pure virtual function) е виртуална функция без тяло. Тя е предназначена да бъде имплементирана от всеки наследник. Дефинира се по следния начин:

virtual f() = 0

Ако клас има поне една чисто виртуална функция, то той става абстрактен клас. От абстрактен клас не можем да правим обекти, но можем да създаваме указатели и референции. Абстрактният клас е предназначен за наследяване. Ако наследник на абстрактния клас не разпише чисто виртуалната функция, то той също става абстрактен.