dědění

- při dědění se mění přístupová práva proměnných a metod bázové třídy v závislosti na způsobu dědění
- class C: public A – A je bázová třída, public značí způsob dědění a C je název nové třídy
- způsob dědění u třídy je implicitně private (a nemusí se uvádět), u struktury je to public (a nemusí se uvádět) class C: D

tabulka ukazuje, jak se při různém způsobu dědění mění přístupová práva bázové třídy (A) ve třídě zděděné

class A	**class B:private A**	**class C:protected A**	**class D:public A**
public a	private a	protected a	public a
private b	-	-	-
protected c	private c	protected c	protected c

- postup volání konstruktorů - konstruktor bázové třídy, konstruktory lokálních proměnných (třídy) v pořadí jak jsou uvedeny v hlavičce, konstruktor (tělo) dané třídy
- destruktory se volají v opačném pořadí než konstruktory

```
class Base {
   int x;
  public:
  float y;
  Base( int i ) : x ( i ) { };
// zavolá konstruktor třídy a pak proměnné,
// x(i) je kontruktor pro int          };

class Derived : Base {
 public:
 int a ;
  Derived ( int i ) : a ( i*10 ) : Base (a) { }
// volání lokálních konstruktoru umožní
// konstrukci podle požadavků, ale nezmění
// pořadí konstruktorů
using Base::y;  // je možné takto vytáhnout
// proměnnou (zděděnou zde do sekce private)
// na jiná přístupová práva (zde public)         };
```

dědění

- - - - příklad 2 - - - - - - - - -

```
class base {
public:
base (int i=10) {…}
}

class derived:base {
complex x,y; ….

public: derived() : y(2,1) {f()  … }

}
```

volá se base::base()
complex::complex(void) - pro x
complex::complex(2,1) – pro y
f() … - vlastní tělo konstruktoru

dědění

- nedědí se: konstruktory, destruktory, operátor =
- ukazatel na potomka je i ukazatelem na předka
- implicitní konstruktor v private zabrání dědění, tvorbu polí
- destruktor v private zabrání dědění a vytváření instancí
- kopykonstruktor v private zabrání předávání (a vracení) parametru hodnotou
- operátor = v private zabrání přiřazení