Jmenné prostory

Jmenné prostory umožňují entity, jako jsou deklarace, definice a funkce sloučit do jednoho celku, kterému je přiřazeno jméno. Jmenné prostory takto poskytují způsob, jak globální prostor rozdělit na více dílčích prostorů.

Deklarace jmenného prostoru:

```
namespace jmeno {
....
}
```

Chceme-li v nějaké části programu použít entitu obsaženou ve jmenném prostoru se jménem ***jmeno***, lze zde uvést direktivu **using**

```
using namespace jmeno;
```

nebo před entitu napsat jméno jmenného prostoru oddělené operátorem rozlišení

```
jmeno::
```

Příklad.

```
namespace A { int i=5;
              const int c=2;
              unsigned u;
}
namespace B { inline unsigned soucet(int i,int j)
              { int m=i+j;
                return m>=0 ? m : 0;
              }
}
{ A::u = B::soucet(A::i,A::c); }
{ using namespace A;
  using namespace B;
  u = soucet(i,c); }
```

Jmenné prostory mohou být vnořené – v jednom jmenném prostoru lze deklarovat další jmenné prostory, neboli jmenný prostor lze rozdělit na další dílčí jmenné prostory. Pro označení vnořených prostorů používáme opět operátor rozlišení.

Příklad.

```
namespace A { namespace A { namespace A { int i=5; }
                            namespace B { const int c=2; }
                          }
              namespace B { unsigned u; }
}
namespace B {
  inline unsigned soucet(int i,int j)
  { int m=i+j;
    return m>=0 ? m : 0; }
```

```
}
{ A::B::u = B::soucet(A::A::A::i, A::A::B::c); }
{ using namespace A::A::A;
  using namespace A::A::B;
  using namespace A::B;
  using namespace B;
  u = soucet(i,c); }
```

Konverze typu - `cast`

Jazyk C++ zavedl další 4 konverze typu, jejichž použití je bezpečnější nebo přehlednější než konverze typu obsažená v jazyce C.

Konverze `const_cast<typ>(`*`výraz`*`)`

Odstraňuje konstantnost ***výrazu***.

Příklad.

```
int a=5;
const int &ra=a;
ra=4;                          // chyba – reference na konstantní hodnotu !!
const_cast<int &>(ra)=4;       // v pořádku – odstraněna konstantnost hodnoty
const int *pb = new int(5);
*pb=4;                         // chyba – ukazatel na konstantní hodnotu !!
*const_cast<int *>(pb)=4;      // v pořádku – odstraněna konstantnost hodnoty
```

Konverze `reinterpret_cast<typ>(`*`výraz`*`)`

Změní interpretaci ***výrazu*** dle nového typu, nemůže odstranit konstantnost. Reprezentace hodnoty se nemění, mění se jen její interpretace.

Příklad.

```
const void *pb = new int(5);
cout << *reinterpret_cast<const int *>(pb);      // v pořádku
*reinterpret_cast<int *>(pb)=4;       // chyba – nelze odstranit konstantnost !!
```

Konverze `static_cast<typ>(`*`výraz`*`)`

Změní datový typ ***výrazu*** na nový typ, nemůže odstranit konstantnost.

Příklad.

```
int i=2; char c;
c = static_cast<char>(i);              // v pořádku
```

```
c = reinterpret_cast<char>(i);    // chyba – nelze změnit reprezentaci !!
```

Konverze `dynamic_cast<typ>(`*výraz*`)`

Změní datový typ ***výrazu*** na nový typ. Typ ***výrazu*** a nový ***typ*** musí být ukazatel nebo reference na polymorfní třídu (obsahující aspoň jednu virtuální funkci) nebo typ **`void *`**. Nemůže odstranit konstantnost.

Přetypování probíhá při výpočtu. Pokud je chybné, jeho výsledek je

- **`nullptr`** u ukazatele
- generuje se výjimka **`bad_cast`** u reference

Příklad.

```
class A { virtual void f() { } };
class B: public A { };
class C { virtual void f() { } };

A a;
B b;
C c;

A *pa;
B *pb;

pa = dynamic_cast<A *>(&b);
cout << pa << endl;
```

0041B1A4 // v pořádku – konverze na ukazatele na děděnou třídu

```
pb = dynamic_cast<B *>(&a);
cout << pb << endl;
```

00000000 // chyba – nelze konvertovat na ukazatele na dědící třídu !!

```
pa = dynamic_cast<A *>(&c);
cout << pa << endl;
```

00000000 // chyba – nelze konvertovat na ukazatele na jinou než děděnou třídu !!