MENNÉ PRIESTORY (NAMESPACES)

Menné priestory

- Ďalšia konštrukcia, ktorá zabraňuje kolízií mien v programe.

- Cieľom je umožniť spoločné linkovanie programu pozostávajúceho z viacerých nezávislých častí a to aj v prípade, že programátori (nevedomky) použili to isté globálne meno pre dve rôzne veci v dvoch rôznych častiach.

Riešenie C

- Používať dlhé mená s dlhým prefixom

```
void moj_dlhy_prefix_fun1(...) {...}
int  moj_dlhy_prefix_var5;
```

```
void tvoj_dlhy_prefix_fun1(...) {...}
int  tvoj_dlhy_prefix_var5;
```

C preprocessor sa teoreticky tiež dá použiť

```
#define  X(meno) moj_prefix_##meno

int X(var);
void X(initVar)() {X(var) = 0; }

int main() {
  X(initVar)();
}
```

Menné priestory

- Budeme mená kvalifikovať jedinečným názvom (menným priestorom), ale chceme, aby sme tento jedinečný názov nemuseli používať pokiaľ nehrozí konflikt.

```
namespace X {
  int var;
  void initVar() {  var = 0;  }
};

int main() {
  X::initVar();
}
```

Príklad

```
namespace XX {
  void fun() ;

  void fun() { ... }
}

namespace YY {
  void fun() ;
}

void YY::fun() {
  ...
  fun();       // rekurzivne volanie
  XX::fun();   // vyvolanie fun z XX

}
```

Namespace alias

- C++ umožňuje definovať viacero mien pre ten istý namespace. Predpokladá sa, že knižnica bude exportovať dlhé meno a užívateľ si ho potom predefinuje na pohodlnejšie krátke meno.

```
namespace American_Telephone_and_Telegraph {
    ...
    void fun() {...}
}
```

```
#include <the_library.h>

namespace ATT = American_Telephone_and_Telegraph;
...
    ATT::fun();
...
```

Menné priestory môžu byť vnorené

```
namespace XX {
   int i;
   namespace YY {
      int j;
   }
}

int main() {
   XX::i = 0;
   XX::YY::j = 1;
}
```

Menné priestory sú obdobou konštrukcie **package** v Jave.

Import mien z menných priestorov: "using directive"

- Using directive umožňuje používať všetky mená z uvedeného menného priestoru bez kvalifikátora (ako **`import XX.*`** v Jave)

```
namespace XX {
  int i;
  void fun() {...}
}

int main() {
  using namespace XX;
  i = 0;
  fun();
}
```

Import mien z menných priestorov: "using declaration"

- Using declaration umožňuje vložiť jedno meno do aktuálneho scope-u.

```
namespace XX {
  int i;
  void fun() {...}
}

int main() {
  using XX::i;
  i = 0;
  XX::fun();
}
```

Príklad 1

```
namespace X {
 int i, j, k;
}

int k;

int main() {
  int i;
  using namespace X;
  i ++;                    // local i
  j ++;                    // X::j
  k ++;                    // error
  ::k ++;                  // global k
  X::k ++;                 // X::k
}
```

Príklad 2

```
namespace X {
 int i, j, k;
}

int k;

int main() {
  int i;
  using X::i;              // error
  using X::j;
  using X::k;               // o.k.
  j ++;                     // X::j
  k ++;                     // X::k
}
```

Príklad 3

```
namespace X {
  int i;
}

namespace Y {
  int i;
}

int main() {
  using namespace X;
  using namespace Y;
  i ++;                          // error
}
```

Global scope

- Ak nie je definovaný žiaden namespace, symbol patrí do takzvaného "global scope" a môže sa jednoznačne referovať ako

```
::k
```

Anonymný namespace

- Symboly z anonymného namespacu sa nelinkujú. Anonymný namespace umožňuje definovať symboly lokálne pre daný súbor. Je to konštrukcia ekvivalentná s deklaráciou local.

```
namespace {
   int i, j, k;
}
```

je ekvivalentné s:

```
static int i, j, k;
```

Namespace std

- Namespace **`std`** je namespace v ktorom sú definované všetky štandardné funkcie jazyka C++.
- C++ definuje pre každý štandardný hlavičkový súbor dve verzie: jednu s implicitnou klauzulou **`using std;`** a druhú verziu bez tejto klauzuly

```
#include <stdio.h>

#include <stdio>
```

Súbor definuje prototypy štandardných funkcií v mennom priestore std a končí riadkom:
`using namespace std;`

Len definuje prototypy štandardných funkcií v mennom priestore **`std`**

Namespace verzus trieda

- Teoreticky by sme mohli namiesto namespacu použiť triedu so všetkými členmi deklarovanými ako static.
- Jeden z rozdielov medzi menným priestorom a triedou je, že menný priestor je otvorený v zmysle, že sa dá dodatočne (a postupne) rozširovať.

```
namespace XX { int i; ... }
...
namespace XX { int j; ... }
...
```

Namespace verzus trieda

- Ďalší z rozdielov je, že mechanizmus preťaženia funguje naprieč mennými priestormi

```
namespace XX { void fun(char c) {...} }
namespace YY { void fun(int c) {...} }

using namespace XX;
using namespace YY;

int main() {
   fun('a');              // XX::fun
   fun(5);                // YY::fun
}
```

Using deklarácia sa dá použiť na preťažovanie cez hierarchiu tried

```
class A {
public:
  void f(int i) {...}
};

class B {
public:
  using A::f;
  void f(double d) { ... }
};

int main() {
  B b;
  b.f(1);          // vyvola A::f, nie B::f
}
```

Podobne aj na preťažovanie cez viacnásobné dedenie

```
class A {
public:
  void f(int i) { ... }
};

class B {
public:
  void f(double d) { ... }
};

class C : public A, public B {
  using A::f;
  using B::f;
  void g() {
     f(1);          // vyvola A::f
     f(1.0);        // vyvola B::f
  }
};
```