Лекція 10 : Простори імен (Namespaces)

Потреба в просторах імен та їх визначення

Реальні програми або програми складаються з багатьох вихідних файлів. Ці файли можуть бути авторськими та підтримуватися більш ніж одним розробником або програмістом. Врешті-решт, окремі файли організовуються та зв'язуються для отримання остаточної програми.
Традиційно організація файлів вимагає, щоб усі імена, які не інкапсульовані у визначеному просторі імен (наприклад, у тілі функції чи класу чи блоку перекладу), повинні міститись в одному спільному глобальному просторі імен, тобто бути доступними для всіх прогам та модулів даних файлів із тими самими іменами. Отже, під час пов'язування окремих модулів може виникнути кілька однакових визначень імен або зіткнень імен.
Механізм простору імен у C ++ долає проблему співпадінь імен у глобальному масштабі. Механізм простору імен дозволяє програмі розподілити на кількість підсистем. Кожна підсистема може визначати та функціонувати в межах своєї сфери.
Декларація просторів імен ідентифікує та присвоює унікальне ім'я оголошеному користувачем простору імен. Це буде використано для вирішення зіткнення чи конфлікту імен при великій розробці програм і бібліотек, де є багато програмістів або розробників, які працюють у різних частинах програми.
Для використання просторів імен C ++ передбачено два етапи:
1. Однозначно ідентифікувати простір імен за допомогою ключового слова namespace.
2. Отримати доступ до елементів ідентифікованого простору імен, застосувавши за допомогою ключового слова using або назви простору імен та оператору доступу(::).

Загальна форма визначення простору імен:

```
namespace <indentifier>{
  //тіло простору імен
 }
```

Приклад:

```
namespace  NewOne{
    int p;
    long q;
    class A { …};
    void foo {…};
}
```

де p та q звичайні змінні, A – визначений користувачем клас, а foo – деяка функція, але всі вони інтегровані в межах простору імен NewOne. Для доступу

до цих змінних за межами простору, потрібно використовувати оператор доступу (scope operator) ::. Зокрема, для попереднього прикладу:

```
NewOne::p;
NewOne::q;
NewOne::A;
NewOne::foo;
```

Визначення простору імен може бути вкладено в інше визначення простору імен. Кожне визначення простору імен повинно з'являтися або в межах файлу, або відразу в іншому визначенні простору імен. Наприклад:

Приклад.

```
//  Використання простору імен з допомогою директиви using
#include <iostream>
using namespace std; // підключаємо стандартний простір імен

namespace SampleOne{ // перший простір імен
    float p = 10.34;
}

namespace SampleTwo{ // другий простір імен

    using namespace SampleOne; // використовуємо перший простір всередині другого
    float q = 77.12;

    namespace InSampleTwo   { // вкладений простір імен
        float r = 34.725;
    }
}

int main(){
    // ця директива підключає декларації змінних з SampleTwo
    using namespace SampleTwo;

    // ця директива підключає лише декларації змінних внутрішнього простору InSampleTwo
    using namespace SampleTwo::InSampleTwo;

    // локальна декларація, має пріоритет
    float p = 23.11;
```

```
    cout<<"p = "<<p<<endl;
    cout<<"q = "<<q<<endl;
    cout<<"r = "<<r<<endl;
}
```

Результат роботи:
C++ Namespace using directive

Приклад (без директиви using)

```
// використання простору імен без директиви using
#include <iostream>
using namespace std;

namespace NewNsOne{
    // декларація змінної в NewNsOne
    int p = 4;
    // декларація функції в NewNsOne
    int funct(int q);
}

namespace NewNsTwo{
    // декларація змінної в NewNsTwo
    int r = 6;

    // декларація функції в NewNsTwo
    int funct1(int numb);

    // декларація вкладеного простору імен
    namespace InNewNsTwo    {
        // декларація змінної в InNewNsTwo
        long tst = 20.9456;
    }
}

int main(){
    /* наступний код згенерує помилку  тому що це не загальний простір імен а головна функція  main */
   // namespace local {
    //      int k;
    // }

    cout<<"NewNsOne::p is "<<(NewNsOne::p)<<endl;
```

```
    cout<<"NewNsTwo::r is "<<(NewNsTwo::r)<<endl;
    cout<<"NewNsTwo::InNewNsTwo::tst is" <<
(NewNsTwo::InNewNsTwo::tst)<<endl;
}
```

Результат роботи:
C++ Namespace without using directive

Примітка. Всі стандартні функції, константи та глобальні змінні, що використовуються в Сі++ починаючи з 98-го стандарту визначені та містяться в стандартному глобальному просторі імен. Тому для використання, наприклад, методу потокового виведення cout<< нам потрібно або підключити для нашої програми за допомогою директиви using стандартний простір імен std:

```
using namespace std;
… //
cout<<  “Using cout”;
```

або кожен раз при використанні класу cout вказувати за допомогою оператору доступу(::) що він належить стандартному простору імен std:

```
std::cout<<“Using cout”;
```

Примітка. Потреба у якихось інших просторах імен, що відмінні від стандартних може зустрітись коли ми використовуємо якусь велику нестандартну бібліотеку, наприклад, boost, llvm, opencv і таке інше.

Псевдоніми просторів імен (Namespace Alias)

Альтернативне ім’я може використовуватися для позначення ідентифікатору простору імен. Псевдонім корисний, коли вам потрібно спростити довгий ідентифікатор простору імен.

Приклад:

```
//  псевдоніми просторів імен  namespace alias
#include <iostream>

using namespace std;

 namespace TheFirstLongNamespaceSample{

    float p = 23.44;
    namespace TheNestedFirstLongNamespaceSample {
```

```
   int q = 100;    }
}

// псевдонім: an alias namespace
namespace First = TheFirstLongNamespaceSample;

// псевдонім для вкладеного простору імен
namespace                                    Second                                    =
TheFirstLongNamespaceSample::TheNestedFirstLongNamespaceSample;

int main(){

      using namespace First; // використання першого псевдоніму
      using namespace Second; // використання другого псевдоніму
      cout<<"p = "<<p<<endl;
      cout<<"q = "<<q<<endl;
}
```

Результат роботи:
C++ Namespace alias

Розширення просторів імен (Namespace Extension)

Визначення простору імен можна розділити на кілька частин однієї одиниці перекладу.
Якщо ви оголосили простір імен, ви можете розширити вихідний простір імен, додавши нові декларації.
Будь-які розширення, які вносяться до простору імен після використання заяви, не будуть відомі в точці, в якій відбувається декларація використання.

Наприклад:

```
// Розширення простору імен
// не може бути зкомпільовано — лише приклад створення

// оригінальний простір імен
namespace One{
      // змінні цього простору імен
      int p;
      int q;
}

 namespace Two{
```

```
    float r;
    int s;
}

// Розширення простору One
namespace One{
    // декларація функції простору One
    void funct1();
    int funct2(int p);
}

int main(){
   return 0;
}
```

Приклад 2:

```
// namespace extension
#include <iostream>

using namespace std;

struct SampleOne
{   };

struct SampleTwo
{   };

// оригінальний простір імен namespace
namespace NsOne{

    // оригінальна функція
    void FunctOne(struct SampleOne){
            cout<<"Processing the struct argument..."<<endl;
     }
}

using NsOne::FunctOne;  // використання using-declaration...

// розширення простору NsOne
namespace NsOne{
```

```
    // перевантажена (overloaded) функція...
    void FunctOne(SampleTwo&)
    {   cout<<"Processing the function argument..."<<endl;   }
}

int main(){

    SampleOne TestStruct;
    SampleTwo TestClass;

    FunctOne(TestStruct);

    // Ця функція не запрацює, бо нема її перевантаженого екземпляру
   // FunctOne(TestClass);
    return 0;
}
```

Результат роботи:
C++ Namespace extension

Анонімні простори імен (Unnamed/anonymous Namespace)

Можна використовувати простір імен ключових слів без ідентифікатора перед заключною дужкою. Це може бути кращою альтернативою використанню глобальної статичної декларації змінної.
Кожен ідентифікатор, який укладений у неназваному просторі імен, є унікальним у блоці трансляції, у якому визначено неназваний простір імен.

Синтаксис:
namespace { namespace_body }

Цей код поводиться так само як:
namespaceunique { namespace_body}
using namespace unique;

Приклад :

```
// анонімний простір імен
#include <iostream>
using namespace std;

// Простір імен без назви  (anonymous namespace)
namespace{
```

```
  int p = 1;  // unique::p
}

void funct1(){
     ++p;  // unique::++p
}

namespace One{
     // вкладений анонімний простір
     namespace      {
          int p;        // One::unique::p
          int q = 3;  // One::unique::q
     }
}

// використання  using-declaration
using namespace One;

void testing(){
    // ++p;            // помилка:  unique::p  або One::unique::p?
    // One::++p;     // помилка , One::p  - невизначене (undefined)

     cout<<"++q = "<<++q<<endl;
}

int main(){
     testing();
}
```

Результат роботи:
C++ Namespace anonymous un-named

Доступ до елементів простору імен (Accessing Namespace Elements)

Існують такі шляхи доступ до елементів простору імен:

- Використовуючи повний шлях імені
- Використання директиви using
- Використання декларації using

Використання повного шляху імені

В цьому варіанті (найбільш уживаному) потрібно просто вказувати повний шлях імені розділяючи імена за допомогою оператору доступу(::):
Приклад.

```
namespace NewNsOne{
    int p = 4;
    int funct(int q);
}

namespace NewNsTwo{
    int r = 6;
    int funct1(int numb);
    namespace InNewNsTwo     {
        long tst = 20.9456;
    }
}

 int main(){
    cout<<"NewNsOne::p is "<<(NewNsOne::p)<<endl;
    cout<<"NewNsTwo::r is "<<(NewNsTwo::r)<<endl;
    cout<<"NewNsTwo::InNewNsTwo::tst                    is"                    <<
(NewNsTwo::InNewNsTwo::tst)<<endl;
}
```

Директива using

Цей метод корисний, коли ви хочете отримати доступ до кількох або всіх членів простору імен. Директива using вказує, що всі ідентифікатори в просторі імен знаходяться в області застосування в тому місці, коли зроблено оператор-використання.
Цей метод також є транзитивним; це означає, що коли ви застосовуєте директиву using до простору імен, яка містить використання директиви всередині себе, ви також отримуєте доступ до цих просторів імен.

Приклад:

```
// Директива using
#include <iostream>

using namespace std;

namespace One
{    float p = 3.1234;    }
```

```
namespace Two {
     using namespace One;
    float q = 4.5678;
    namespace InTwo
     {   float r = 5.1234;   }
}

int main(){
     // використання всіх змінних з Two
     using namespace Two;

     // доступ лише до вкладеного підпростору імен InTwo
     using namespace Two::InTwo;

    // локальна змінна
     float p = 6.12345;

     cout<<"p = "<<p<<endl;
     cout<<"q = "<<q<<endl;
     cout<<"r = "<<r<<endl;
}
```

Результат:
C++ Namespace using directive access element

Декларація using

Ви можете отримати доступ до елементів простору імен індивідуально, застосувавши using-декларацію. Тут ви додаєте оголошений ідентифікатор до локального простору імен.

Синтаксис:
using::unqualified-identifier;

Приклад:

```
// Декларція using :
// функція funct2() визначена в двох різних просторах

#include <iostream>
using namespace std;

 namespace One{
```

```
    float funct1(float q) {
        return q;
      }
    // перше визначення funct2()
    void funct2()  {
        cout<<"funct2() function, One namespace..."<<endl;
    }
}

namespace Two{
    // друге визначення funct2()
    void funct2()
    {   cout<<"funct2() function, Two namespace..."<<endl;   }

}

int main(){
    // використання using визначає версію funct2()
    using One::funct1;    // визначає версію
    using Two::funct2;    // визначає версію

    float p = 4.556;          // локальна змінна
    cout<<"First p value, local = "<<p<<endl;
    p = funct1(3.422);

    cout<<"Second p value, by function call = "<<p<<endl;
    funct2();
}
```