Przestrzenie nazw

Bogdan Kreczmer

bogdan.kreczmer@pwr.edu.pl

Katedra Cybernetyki i Robotyki
Politechnika Wrocławska

Kurs: Programowanie obiektowe

Niniejsza prezentacja została wykonana przy użyciu systemu składu LaTeX *oraz stylu* `beamer`, *którego autorem jest Till Tantau.*

Strona domowa projektu `Beamer`:

`http://latex-beamer.sourceforge.net`

Plan prezentacji

Plan prezentacji

Zarys problemu

Należy stworzyć zbiór klas modelujących pojęcie prostej w przestrzeni i na płaszczyźnie. Ich metody powinny umożliwić w łatwy sposób określenie, czy dany punkt w przestrzeni lub na płaszczyźnie należy do danej prostej czy też nie.

Te same nazwy, różne definicje

prosta-plaszczyzna.hh

```
class Wektor {
        double   _x, _y;
        ...
};

class Prosta {
        Wektor   _WekNorm;
        double        _c;
        ...
};
```

prosta-przestrzen.hh

```
class Wektor {
        double   _x, _y, _z;
        ...
};

class Prosta {
        Wektor   _WekOsiowy;
        Wektor   _PnkProstej;
        ...
};
```

```
#include "prosta-plaszczyzna.hh"
#include "prosta-przestrzen.hh"

int main( )
{
    Prosta  Pr2;
    Prosta  Pr3;
    ...
}
```

Te same nazwy, różne definicje

prosta-plaszczyzna.hh

```
class Wektor {
        double   _x, _y;
        ...
};

class Prosta {
        Wektor   _WekNorm;
        double        _c;
        ...
};
```

prosta-przestrzen.hh

```
class Wektor {
        double   _x, _y, _z;
        ...
};

class Prosta {
        Wektor   _WekOsiowy;
        Wektor   _PnkProstej;
        ...
};
```

```
#include "prosta-plaszczyzna.hh"
#include "prosta-przestrzen.hh"

int  main( )
{
    Prosta  Pr2;
    Prosta  Pr3;
    ...
}
```

Zroznicownie nazw

prosta2d.hh

```
class Wektor2D {
        double   _x, _y;
        ...
};

class Prosta2D {
        Wektor2D   _WekNorm;
        double     _c;
        ...
};
```

prosta3d.hh

```
class Wektor3D {
        double   _x, _y, _z;
        ...
};

class Prosta3D {
        Wektor3D   _WekOsiowy;
        Wektor3D   _PnkProstej;
        ...
};
```

```
#include "prosta2d.hh"
#include "prosta3d.hh"

int main( )
{
    Prosta2D  Pr2;
    Prosta3D  Pr3;
    ...
}
```

Zroznicownie nazw w stylu C

d2_prosta.hh

```
class D2_Wektor {
        double   _x, _y;
        ...
};

class D2_Prosta {
        D2_Wektor   _WekNorm;
        double      _c;
        ...
};
```

d3_prosta.hh

```
class D3_Wektor {
        double   _x, _y, _z;
        ...
};

class D3_Prosta {
        D3_Wektor   _WekOsiowy;
        D3_Wektor   _PnkProstej;
        ...
};
```

```
#include "d2_prosta.hh"
#include "d3_prosta.hh"

int main( )
{
    D2_Prosta  Pr2;
    D3_Prosta  Pr3;
    ...
}
```

Przestrzenie nazw

d2_prosta.hh

```
namespace D2 {
   class Wektor {
         double   _x, _y;
         ...
   };

   class Prosta {
         Wektor   _WekNorm;
         double         _c;
         ...
   };
}
```

d3_prosta.hh

```
namespace D3 {
   class Wektor {
         double   _x, _y, _z;
         ...
   };

   class Prosta {
         Wektor   _WekOsiowy;
         Wektor   _PnkProstej;
         ...
   };
}
```

```
#include "d2_prosta.hh"
#include "d3_prosta.hh"

int main( )
{
    D2::Prosta  Pr2;
    D3::Prosta  Pr3;
    ...
}
```

Plan prezentacji

Blok przestrzeni nazw

```
int  ZmGlobalna;

namespace D3 {
      class Wektor {
            double   _x, _y, _z;
            ...
      };


      class Prosta {
            Wektor   _WekOsiowy;
            Wektor   _PnkProstej;
            ...
       };
}

int  main( )
{
       ZmGlobalna = 4;
       D3::Wektor   Wk;
       D3::Prosta   Pr;
}
```

W danej jednostce translacyjnej można stworzyć jednolity blok grupujący wszystkie definicje w wybranej przewstrzeni nazw.

Blok przestrzeni nazw

```
int  ZmGlobalna;

namespace D3 {
      class Wektor {
            double  _x, _y, _z;
            ...
      };
}
namespace D3 {
      class Prosta {
            Wektor  _WekOsiowy;
            Wektor  _PnkProstej;
            ...
       };
}

int  main( )
{
       ZmGlobalna = 4;
       D3::Wektor   Wk;
       D3::Prosta   Pr;
}
```

Tę samą przestrzeń nazw można jednak rozbić na dowolną liczbę bloków.

Ta sama przestrzeń nazw w różnych modułach

d3_wektor.hh

```
...
namespace D3 {
   class Wektor {
         double   _x, _y, _z;
         ...
   };
}
...
```

d3_prosta.hh

```
...
#include  "d3_wektor.hh"

namespace D3 {
   class Prosta {
         Wektor   _WekOsiowy;
         Wektor   _PnkProstej;
         ...
   };
}
...
```

```
#include  "d3_wektor.hh"
#include  "d3_prosta.hh"

int  main( )
{
    D3::Wektor   Wk;
    D3::Prosta   Pr;
    ...
}
```

Ta sama przestrzeń nazw może być *rozbita* na zbiór różnych jednostek translacyjnych (np. `3d_wektor.cpp` i `3d_prosta.cpp`) z własnymi plikami nagłówkowymi (np. `3d_wektor.hh` i `3d_prosta.hh`) zawierającymi definicje klas i zapowiedzi definicji metod.

Co dają przestrzenie nazw

- Definiowanie przestrzeni nazw pozwala rozwiązać konflikty nazw. Tym samym daje znacznie większą swobodę programiście.
- Definicja przestrzeni nazw może być „kontynuowana” w kilku jednostek translacyjnych.
- Jedną z ważniejszych konsekwencji wykorzystywania przestrzeni nazw jest możliwość *tworzenia modułów w sensie logicznym*, a nie tak jak to ma miejsce, np. w języku *C* tylko w sensie stricte fizycznym.

Co dają przestrzenie nazw

- Definiowanie przestrzeni nazw pozwala rozwiązać konflikty nazw. Tym samym daje znacznie większą swobodę programiście.
- Definicja przestrzeni nazw może być „kontynuowana” w kilku jednostek translacyjnych.
- Jedną z ważniejszych konsekwencji wykorzystywania przestrzeni nazw jest możliwość *tworzenia modułów w sensie logicznym*, a nie tak jak to ma miejsce, np. w języku *C* tylko w sensie stricte fizycznym.

Co dają przestrzenie nazw

- Definiowanie przestrzeni nazw pozwala rozwiązać konflikty nazw. Tym samym daje znacznie większą swobodę programiście.
- Definicja przestrzeni nazw może być „kontynuowana” w kilku jednostek translacyjnych.
- Jedną z ważniejszych konsekwencji wykorzystywania przestrzeni nazw jest możliwość *tworzenia modułów w sensie logicznym*, a nie tak jak to ma miejsce, np. w języku *C* tylko w sensie stricte fizycznym.

Dostęp do przesłoniętych zmiennych globalnych

```
int Zm = 7;

void Funkcja( )
{
    int Zm = 55;
    ::Zm = Zm;
}

int main( )
{
    cout << "Zm. Globalna: " << Zm << endl;
    Funkcja( );
    cout << "Zm. Globalna: " << Zm << endl;
}
```

Dostęp do przesłoniętych zmiennych globalnych

```
int Zm = 7;

void Funkcja( )
{
    int Zm = 55;
    ::Zm = Zm;
}

int main( )
{
    cout << "Zm. Globalna: " << Zm << endl;
    Funkcja( );
    cout << "Zm. Globalna: " << Zm << endl;
}
```

```
Wynik działania:
 Zm. Globalna: 7
 Zm. Globalna: 55
```

W nienazwanej przestrzeni nazw znajdują się wszystkie globalnie definiowane zmiennych, typy oraz funkcje. Zmienne te można zawsze "odsłonić" stosując nazwę kwalifikowaną.

Dostęp do przesłoniętych zmiennych globalnych

```
int  Zm = 7;

void  Funkcja( int  Zm )
{
    ::Zm = Zm;
}


int  main( )
{
    cout << "Zm. Globalna: " << Zm << endl;
    Funkcja( 10 );
    cout << "Zm. Globalna: " << Zm << endl;
}
```

```
Wynik działania:
 Zm. Globalna: 7
 Zm. Globalna: 10
```

W nienazwanej przestrzeni nazw znajdują się wszystkie globalnie definiowane zmiennych, typy oraz funkcje. Zmienne te można zawsze "odsłonić" stosując nazwę kwalifikowaną.

Dostęp do przesłoniętych typów

```
enum   Symbole { a, b, c};

void  Funkcja_LokalnyTyp( Symbole Param )
{     enum   Symbole { x, y, z };
      Symbole    Zm1 = x;
      ::Symbole    Zm2 = Param;
}     ::Symbole    Zm3 = ::a;

int  main( )
{     Symbole    x = a;
}     Funkcja_LokalnyTyp( x );
```

```
Wynik działania:
 Zm. Globalna: 7
 Zm. Globalna: 10
```

W nienazwanej przestrzeni nazw znajdują się wszystkie globalnie definiowane zmiennych, typy oraz funkcje. Zmienne te można zawsze "odsłonić" stosując nazwę kwalifikowaną.

Zagnieżdżanie przestrzeni nazw

```
int  ZmGlobalna;

namespace D3 {
      class Wektor {
            ...
      };

      namespace Bryly {
            class Szescian {
                  ...
            };
       }
}
```

```
int  main( )
{
    ZmGlobalna = 4;
    D3::Wektor            Wk;
    D3::Bryly::Szescian   Sz;
}
```

Przestrzenie nazw mogą być zagnieżdżane.

Dyrektywa using namespace

```
int  ZmGlobalna;

namespace D3 {
      class Wektor {
            ...
      };

      namespace Bryly {
            class Szescian {
                  ...
            };
      }
}

using namespace D3;

int  main( )
{
   ZmGlobalna = 4;
   Wektor            Wk;
   Bryly::Szescian   Sz;
}
```

Użycie dyrektyw **using namespace** powoduje przeniesienie wszystkich nazw z danej podrzestrzeni, do której się ona odnosi, do przestrzeni nazw, w której jest zastosowana.

W przedstawionym przykładzie nazwy z podprzestrzeni D3 zostały przeniesione do przestrzeni nienazwanej związanej z tradycyjnymi zmiennymi globalnymi.

Użycie dyrektywy **using namespace**

Dyrektywy **using namespace** nie należy używać w plikach nagłówkowych. Jej użycie tam jest przejawem bardzo złego stylu programowania.

Dyrektywa using namespace

```
int  ZmGlobalna;

namespace D3 {
      class Wektor {
            ...
      };

      namespace Bryly {
            class Szescian {
                  ...
            };
       }
}

using namespace D3::Bryly;

int  main( )
{
    ZmGlobalna = 4;
    D3::Wektor      Wk;
    Szescian        Sz;
}
```

Dyrektywę **using namespace** możemy stosować selektywnie do wybranych podprzestrzeni.

W tym przykładzie została ona zastosowana do podprzestrzeni `D3::Bryly`. Do pozostałych elementów przestrzeni nazw `D3` musimy odwoływać się poprzez pełną nazwę kwalifikowaną.

Dyrektywa using namespace

```
namespace D3 {
    class Wektor {
        ...
    };
    namespace Bryly {
        class Szescian {
            ...
        };
        Szescian    SzesWzorcowy;
    }
    using namespace Bryly;
    Szescian    DrugiSzes;
}

int  main( )
{
   D3::DrugiSzes = D3::SzesWzorcowy;
}
```

Dyrektywę **using namespace** możemy stosować również w podprzestrzeniach. Jej działanie ograniczone jest tylko do danej podprzestrzeni.

W tym przykładzie dyrektywa **using namespace** powoduje przeniesienie wszystkich nazw z podprzestrzeni `Bryla` do przestrzeni nazw `D3`.

Dyrektywa using namespace

```
namespace D3 {
    class Wektor {
        ...
    };

    namespace Bryly {
        class Szescian {
            ...
        };
        Szescian    SzesWzorcowy;
    }

    Bryly::Szescian    DrugiSzes;
}

int main( )
{
   D3::DrugiSzes = D3::Bryly::SzesWzorcowy;
}
```

Bez tej dyrektywy nie *stracimy* podprzestrzeni `Bryly` i niezbędne jest wówczas odwoływanie się do jej poszczególnych elementów poprzez nazwy kwalifikowane.

Deklaracja **using**

```
namespace D3 {
    class Wektor {
        ...
    };

    namespace Bryly {
        class Szescian {
            ...
        };
        Szescian    SzesWzorcowy;
    }

    Bryly::Szescian    DrugiSzes;
}
using D3::Bryly::SzesWzorcowy;

int  main( )
{
   D3::DrugiSzes = SzesWzorcowy;
}
```

Elementy danej podprzestrzeni nazw możemy przenosić selektywnie stosując deklarację **using**.

Dyrektywa **using namespace** – ograniczanie zakresu

```
namespace D3 {
        class Wektor {
                ...
        };
        namespace Bryly {
                class Szescian {
                        ...
                };
                Szescian    SzesWzorcowy;
        }
        Bryly::Szescian    DrugiSzes;
}

int  main( )
{
    {
        using namespace D3;
        DrugiSzes = Bryly::SzesWzorcowy;
    }
    D3::DrugiSzes = D3::Bryly::SzesWzorcowy;
}
```

Ograniczenie użycia dyrektywy **using namespace** możemy zrealiwać poprzez jej wykorzystanie w bloku ciała funkcji. Ogranicza on wówczas ważność stosowania tej deklaracji do tego bloku.

Deklaracja **using** – ograniczanie zakresu

```
namespace D3 {
    class Wektor {
        ...
    };
    namespace Bryly {
        class Szescian {
            ...
        };
        Szescian    SzesWzorcowy;
    }
    Bryly::Szescian    DrugiSzes;
}

int main( )
{
    {
        using D3::Bryly::SzescWzorcowy;
        D3::DrugiSzes = SzesWzorcowy;
    }
    D3::DrugiSzes = D3::Bryly::SzesWzorcowy;
}
```

Analogicznie można ograniczyć zakres ważności deklaracji **using**.

Użycie deklaacji **using**

W plikach nagłówkowych nie powinno używać się zarówno dyrektywy **using namespace** jak też deklaracji **using**.

Koniec prezentacji
Dziękuję za uwagę