Przeciążenie operatora indeksującego i funkcyjnego, klasy Wektor i Macierz

Bogdan Kreczmer

bogdan.kreczmer@pwr.edu.pl

Katedra Cybernetyki i Robotyki
Wydziału Elektroniki
Politechnika Wrocławska

Kurs: Programowanie obiektowe

Niniejszy dokument zawiera materiały do wykładu dotyczącego programowania obiektowego. Jest on udostępniony pod warunkiem wykorzystania wyłącznie do własnych prywatnych potrzeb i może on być kopiowany

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

Modelowanie pojęcia wektora z wykorzystaniem tablicy

```
class Wektor { //.....
   double   _Wsp[2];
  public:


}; //.....

int  main( )
{
   Wektor  W1, W2;
   double  Liczba;



 }
```

Współrzędne wektora możemy reprezentować w postaci wartości tablicy.

Modelowanie pojęcia wektora z wykorzystaniem tablicy

```
#define ROZMIAR_WEKTORA      2

class Wektor { //.....
   double    _Wsp[ROZMIAR_WEKTORA];
  public:


}; //.....


int  main( )
{
   Wektor  W1, W2;
   double   Liczba;

   Liczba = W1[0];


 }
```

Jej rozmiar wygodnie jest zapisać w postaci stałej symbolicznej. Zaletą jest to, że widząc ją w kodzie programu rozumiemy jej znaczeni. Liczba natomiast w różnych kontekstach może mieć różne znaczenie. Ponadto takie rozwiązanie pozwala łatwo zwiększyć wymiar wektora.

Modelowanie pojęcia wektora z wykorzystaniem tablicy

```
#define ROZMIAR_WEKTORA      2

class Wektor { //.....................................................
   double   _Wsp[ROZMIAR_WEKTORA];
  public:
   double   operator  [ ] (int Ind) const { return  _Wsp[Ind]; }

}; //.....................................................................


int  main( )
{
   Wektor  W1, W2;
   double   Liczba;

   Liczba = W1[0];


 }
```

Jako element interfejsu klasy, który umożliwia odczyt wartości danej współrzędnej, możemy wykorzystać przeciążenie operatora indeksującego.

Modelowanie pojęcia wektora z wykorzystaniem tablicy

```
#define ROZMIAR_WEKTORA      2

class Wektor { //.....................................................
   double   _Wsp[ROZMIAR_WEKTORA];
  public:
   double   operator  [ ] (int Ind) const { return  _Wsp[Ind]; }
   double& operator  [ ] (int Ind)        { return  _Wsp[Ind]; }
}; //.....................................................................
```

```
int  main( )
{
   Wektor  W1, W2;
   double  Liczba;

   Liczba = W1[0];
   W1[1] = Liczba;
   W2[1] = W1[0];
 }
```

Aby umożliwić zapis możemy wykorzystać kolejne przeciążenie operatora indeksującego, które będzie zwracało referencję do zadanego pola tablicy.

Modelowanie pojęcia wektora z wykorzystaniem tablicy

```
#define ROZMIAR_WEKTORA      2

class Wektor { //.....................................................
   double   _Wsp[ROZMIAR_WEKTORA];
  public:
   double   operator  ( ) (int Ind) const { return  _Wsp[Ind]; }
   double& operator  ( ) (int Ind)        { return  _Wsp[Ind]; }
}; //.....................................................................
```

```
int  main( )
{
   Wektor  W1, W2;
   double  Liczba;

   Liczba = W1(0);
   W1(1) = Liczba;
   W2(1) = W1(0);
 }
```

Interfejs klasy Wektora można również zrealizować w oparciu o przeciążenie operatora funkcyjnego. Zalecane jest jednak stosowanie przeciążeń operatora indeksującego.

Plan prezentacji

Dostęp do pól klasy Macierz – operator funkcyjny

```
#define  ROZMIAR_MAC      2

class Macierz { //.....................................................
      double    _Tab[ROZMIAR_MAC][ROZMIAR_MAC];
   public:


}; //.....................................................................
```

Do przechowywania wartości elementów macierzy możemy wykorzystać tablicę.

```
int  main( )
{
   Macierz   M;
   double    Liczba;

}
```

Dostęp do pól klasy Macierz – operator funkcyjny

```
#define ROZMIAR_MAC      2

class Macierz { //.....................................................
      double    _Tab[ROZMIAR_MAC][ROZMIAR_MAC];
    public:
      double    operator( ) (int Wie, int Kol) const { return  _Tab[Wie][Kol]; }
      double&   operator( ) (int Wie, int Kol)       { return  _Tab[Wie][Kol]; }
}; //.....................................................................
```

```
int  main( )
{
    Macierz   M;
    double    Liczba;

    Liczba = M(0,0);   M(0,2) = 3;
}
```

Na potrzeby stworzenia interfejsu klasy dogodnie jest użyć przeciążeń operatorów funkcyjnych.

Koniec prezentacji
Dziękuję za uwagę