Jacek Matulewski
WWW: http://www.fizyka.umk.pl/~jacek/cpp/
wersja z dnia 3 listopada 2006

Najnowsza wersja: http://www.fizyka.umk.pl/~jacek/cpp/cpp-wskazniki2.pdf
Źródła: http://www.fizyka.umk.pl/~jacek/cpp/cpp-wskazniki2.zip

Wskaźniki do funkcji i metod

Wskaźnik do funkcji

Funkcja, jej typ i funkcja wywołująca funkcję o sygnaturze TFunkcja:

```
int Funkcja(int argument){return argument;}

typedef int (*TFunkcja)(int);

void FunkcjaWywolujacaFunkcje(TFunkcja funkcja,int argument)
{
    int wf=funkcja(argument);
    ShowMessage(IntToStr(wf));
}
```

Testy w metodzie zdarzeniowej przycisku:

```
void __fastcall TForm1::Button1Click(TObject *Sender)
{
    int wf=Funkcja(1);
    ShowMessage(IntToStr(wf));

    FunkcjaWywolujacaFunkcje(Funkcja,3);
}
```

Wskaźnik do metody

Zob. „Wskaźnik do elementu klasy nie jest wskaźnikiem” Stephen C. Dewhurst „C++. Kanony wiedzy programistycznej, s. 67 i nn.

Wskaźnik do metody zawiera wskaźnik do obiektu i offset do konkretnego elementu składowego.

Jednak wskaźnik do metody statycznej jest zwykłym wskaźnikiem do funkcji!

```
class TKlasa
{
    public:
        int Metoda(int argument) const{return argument;}
        static MetodaStatyczna(int argument){return argument;}
};

//tu jest ukryty argument - wskaznik do instancji obiektu
typedef int (TKlasa::*TMetoda)(int) const;

void FunkcjaWywolujacaMetode(TKlasa* pobiekt,TMetoda metoda,int argument)
{
    int wm=(pobiekt->*metoda)(argument);
    ShowMessage(IntToStr(wm));
}

void FunkcjaWywolujacaMetode(TKlasa obiekt,TMetoda metoda,int argument)
{
    int wm=(obiekt.*metoda)(argument);
    ShowMessage(IntToStr(wm));
}
```

Testy w metodzie zdarzeniowej przycisku:

```
void __fastcall TForm1::Button1Click(TObject *Sender)
{
    int wf=Funkcja(1);
    ShowMessage(IntToStr(wf));

    TKlasa obiekt;
    int wm=obiekt.Metoda(2);
    ShowMessage(IntToStr(wm));

    FunkcjaWywolujacaFunkcje(Funkcja,3);
    FunkcjaWywolujacaFunkcje(TKlasa::MetodaStatyczna,4); //wskaznik do metody statycznej
= wskaznik do funkcji

    FunkcjaWywolujacaMetode(&obiekt,TKlasa::Metoda,5);
    FunkcjaWywolujacaMetode(obiekt,TKlasa::Metoda,6);

    TKlasa* pobiekt=new TKlasa();
    FunkcjaWywolujacaMetode(pobiekt,TKlasa::Metoda,7);
}
```

„Rzutowanie” metody na funkcję

Problem: przekazac wskaznik do metody do funkcji FunkcjaWywolujacaFunkcje

Najprostsze rozwiązanie:

```
int FunkcjaOtaczajacaMetode(int argument)
{
    TKlasa obiekt; //to moglby byc obiekt globalny
    return obiekt.Metoda(argument);
}
```

Testy:

```
void __fastcall TForm1::Button1Click(TObject *Sender)
{
    FunkcjaWywolujacaFunkcje(FunkcjaOtaczajacaMetode,8);
}
```

Ta sama filozofia, ale z użyciem szablonu:

```
template<typename T> int SzablonFunkcjaOtaczajacaMetode(int argument)
{
    T obiekt; //to moglby byc obiekt globalny
    return obiekt.Metoda(argument);
}
```

Testy:

```
void __fastcall TForm1::Button1Click(TObject *Sender)
{
    FunkcjaWywolujacaFunkcje(SzablonFunkcjaOtaczajacaMetode<TKlasa>,11);
}
```

„Rzutowanie” funkcji na metodę

Problem: przekazac wskaznik do funkcji do funkcji FunkcjaWywolujacaMetode

Rozwiązanie 1: dodać do klasy metodę wywołującą funkcję:

```
class TKlasa
{
    public:
        int Metoda(int argument) const{return argument;}
        static MetodaStatyczna(int argument){return argument;}

    public:
        int MetodaOtaczajacaFunkcje(int argument) const
        {
            return Funkcja(argument);
        }
};
```

Testy:

```
void __fastcall TForm1::Button1Click(TObject *Sender)
{
    TKlasa obiekt;
    int wm=obiekt.Metoda(2);
    ShowMessage(IntToStr(wm));

    FunkcjaWywolujacaMetode(obiekt,TKlasa::MetodaOtaczajacaFunkcje,9);
}
```

Rozwiązanie 2: Szablony

Szablony funkcji wywołujących metodę:

```
//template<typename T> typedef int (T::*TSzablonMetoda)(int) const; //to nie dziala w
Borland


template<typename T> void SzablonFunkcjaWywolujacaMetode(T* pobiekt,int
(T::*metoda)(int) const,int argument)
{
    int wm=(pobiekt->*metoda)(argument);
    ShowMessage(IntToStr(wm));
}


template<typename T> void SzablonFunkcjaWywolujacaMetode(T obiekt,TMetoda metoda,int
argument)
{
    int wm=(obiekt.*metoda)(argument);
    ShowMessage(IntToStr(wm));
}
```

Szablon klasy potomnej (nie trzeba modyfikować oryginalnej klasy i można wskazać wywoływaną funkcję):

```
template<typename T> class TSzablonKlasaPotomna : public T
{
    private:
        TFunkcja funkcja;
    public:
        TSzablonKlasaPotomna(TFunkcja funkcja):funkcja(funkcja){};
        int NowaMetodaOtaczajacaFunkcje(int argument) const
        {
            return funkcja(argument);
        }
};
```

Testy:

```
void __fastcall TForm1::Button2Click(TObject *Sender)
{
    TSzablonKlasaPotomna<TKlasa> obiektPotomnej(Funkcja);
    FunkcjaWywolujacaMetode(
                  obiektPotomnej,
                  (TMetoda)(TSzablonKlasaPotomna<TKlasa>::NowaMetodaOtaczajacaFunkcje),
                  12);
}
```

C++Builder: słowo kluczowe `__closure`

W kompilatorach firmy Borland możliwe jest zadeklarowanie typu funkcji składowej (metody) w taki sposób, że metod tego typu można używac tak, jakby były funkcjami. Służy do tego niestandardowe słowo kluczowe `__closure`.

```
/*
class TKlasa
{
    public:
        int Metoda(int argument) const{return argument;}
};
*/

typedef int (__closure *TMetodaJakFunkcja)(int);

void __fastcall TForm1::Button3Click(TObject *Sender)
{
    TKlasa obiekt;
    TMetodaJakFunkcja metodajakfunkcja;
    metodajakfunkcja=obiekt.Metoda; //blad, gdy inicjacja razem z deklaracja!!!

    int wmjf=metodajakfunkcja(14);
    ShowMessage(IntToStr(wmjf));
}
```

Z definicji typu wynika, że modyfikator `__closure` uzupełnia typ wskaźnika do funkcji o wskaźnik do odpowiedniego obiektu, jednak tak, że typ nie jest związany z żadną konkretną klasą, lecz może przechowywać metody dowolnej klasy. Nie ma znaczenia, czy przy metodzie jest słowo kluczowe `const`. Mechanizm ten wykorzystywany jest przez bibliotekę VCL do przechowywania wskaźników do metod zdarzeniowych.

Nie oznacza to jednak, że wskaźnik typu `TMetodaJakFunkcja` może być przekazany do funkcji `FunkcjaWywolujacaFunkcje`. Nie może tez być przekazany do funkcji `FunkcjaWywolujacaMetode`. Możemy natomiast stworzyć nową wersję tych funkcji:

```
void FunkcjaWywolujacaMetodeJakFunkcje(TMetodaJakFunkcja metoda,int argument)
{
    int wmjf=metoda(argument);
    ShowMessage(IntToStr(wmjf));
}
```

Możemy to rozwiązanie przetestować:

```
void __fastcall TForm1::Button3Click(TObject *Sender)
{
    TKlasa obiekt;
    TMetodaJakFunkcja metodajakfunkcja;
    metodajakfunkcja=obiekt.Metoda; //blad, gdy inicjacja razem z deklaracja!!!

    int wmjf=metodajakfunkcja(14);
    ShowMessage(IntToStr(wmjf));

    FunkcjaWywolujacaMetodeJakFunkcje(metodajakfunkcja,15);
}
```