W1 Programowanie obiektowe (OOP)

Dr. Inż. Hab. Siergiej Fialko, IMK-PK,

http://torus.uck.pk.edu.pl/~fialko
sfialko@riad.pk.edu.pl

Bibliografia

1. B. Stroustrup. Jezyk C++. Wydawnictwo Naukowo-Techniczne, Warszawa,
2. Herbert Schildt. Programowanie C++. Wydawnictwo RM, Warszawa, 2002.
3. S. B. Lippman, J. Lajoie. Podstawy języka C++, Wydawnictwo Naukowo-Techniczne, Warszawa.

Rozmiar kodu	Mały ilość linij kodu (ilk) < 10 000	Średni 10 000 < ilk < 100 000	Duży ilk > 100 000
Typ języka programowania	asembler, FORTRAN	Języki strukturalne C, Pascal	Języki obiektowe C++

Cechy języków strukturalnych: obsluga samodzieljnych procedur, zmiennych lokalnych, duża liczba konstrukcji warunkowych, możliwość rezygnacji ze stosowania instrukcji *goto*

Program, napisany w języku strukturalnym (na przykład, w C), jest tworzony poprzez definiowanie funkcji, z których każda może operować na dowolnym typie danych,wykorzystywanych przez program.

W każdym języku obiektowym definiuje się dane oraz procedury, które mogą operować wyłącznie na tych danych. Typ danych określa, jakiego rodzaju operacje mogą być dokonywane.

W1

Cechy języków obiektowych: hermetyzacja (inkapsulacja), polimorfizm, dziedziczenie.

Hermetyzacja – to mechanizm, łączący w całość kod i dane, na których on operuje, zabezpieczający je przed zewnętrzną ingerencja i niepoprawnym użyciem.

• Kod i dane są połączone w *obiekcie*, który i realizuje hermetyzacje

• Znajdujące się wewnątrz obiektu kod i dane mogą być prywatne lub publiczne

• Prywatny kod i dane są dostępne tylko w obszarze obiektu. Zewnątrz obiektu składniki prywatne są ukryte – ani prywatny kod, ani prywatne dane nie mogą być wykorzystywane przez fragmenty programu, znajdujące się poza obiektem

• Publiczny kod i dane są dostępne dla innych elementów programu, znajdujących się jak wewnątrz obiektu, tak i poza nim. Najczęściej składniki publiczne obiektu są wykorzystywane do tworzenia interfejsu do prywatnych elementów obiektu.

Polimorfizm – realizuje generalną idee „jeden interfejs – wiele metod”. Nadaje możliwość jednemu interfejsowi kontrolować dostęp do ogólnej klasy akcji. O tym, jaka konkretnie akcja pozostanie wybrana, decyduje sytuacja.

Na przykład, w języku C, który nie wspiera polimorfizmu, dla obliczenia wartości bezwzględnej liczb całkowitych, liczb całkowitych typu long i liczb zmiennoprzecinkowych używamy różne funkcje abs(), labs(), fabs(). W języku C++ przy użyci polimorfizmu te funkcje mogą mieć taką samą nazwę (jedyny interfejs) i zupełnie różne realizacji

```
int abs(int val);
long abs(long val);
double abs(double val);
```

Kompilator automatycznie wybierze, która z trzech funkcji ma być wywołana po typu argumentu faktychnego: pierwsza, jeśli typ argumentu faktycznego jest int, drugą, jeśli long, trzecią, jeśli double.

W1

W języku C++ można wykorzystywać tą samą nazwę funkcji dla wielu różnych akcji – przeciążenie funkcji (function overloading), a również zastosowywać te same operatory dla działań nad różnymi obiektami – przeciążenie operatorów (operator overloading).

Polimorfizm zmniejsza złożoność programu i polepsza jej czytelność, zezwala jeden interfejs na różne akcji.

Dziedziczenie to proces, w którym obiekt otrzymuje właściwości innego obiektu.

• Jest to bardzo ważne, gdyż pozwala realizować zasadę *klasyfikacji*. Okazuje się, że większość informacji można uporządkować w sposób *hierarchiczny*. Na przykład, autobus i pociąg są częścią klasy transport, klasa transport jest częścią klasy maszyny.

• Gdyby nie dziedziczenie, cechy każdego obiektu trzeba by było definiować każdorazowo w sposób jawny. Jednak, dzięki klasyfikacji, dla obiektu trzeba zdefiniować te cechy, które czynią go niepowtarzalnym w obrębie klasy. Mechanizm dziedziczenia sprawdza, że obiekt może być specyficznym egzemplarzem bardziej ogólnej klasy.

Pierwszy program na C++

```
# include <iostream>
using namespace std;

int main()
{
        int i;
        cout  << “hello, C++” << “\n”;   //komentarz jednoliniowy
        /* Wieloliniowe komentarze są dopuszczalne*/

        //wprowadź liczbe, stosując instrukcje >>
        cin >> i;

        //wyświetli i
        cout  <<  ”i = ” << i << ”i *i = ” << i*i << ”\n ” ;

        return 0;
}
```

W1

 # include <iostream>

To jest analog C - #include <stdio.h>. W standardzie C++ używają nowy styl nagłówkowy. Przecież stary styl #include <iostream.h> też będzie działało.

using namespace std;

Będzie wykorzystana przestrzeń nazw *std*. Przestrzeń nazw – tworzy deklaratywny obszar, w którym umieszczone są różne elementy programu. Pomaga uporządkować duże programy.
Instrukcja *using* mówi, że chcemy posłużyć się przestrzenią nazw std, w której jest zadeklarowana cała biblioteka standardu C++.
Biblioteki C nie wymagają instrukcji *using* dla tego że domyślnie dostępne w globalnej przestrzeni nazw.

int main() – lista parametrów jest pusta, przecież *void* nie jest potrzebne. W języku C to by wyglądało tak: int main(void);

```
cout  << “hello, C++” << “\n”   //komentarz jednoliniowy
        /* Wieloliniowe komentarze są dopuszczalne*/
```

W1

W języku C brakuje operatorów wejścia\wyjścia, więc ich role przy organizacji we\wy z monitora\na monitor wykonują funkcje printf(...), scanf(...). W języku C++ dla wykonania wyjścia istnieje operator << , a wejścia - >> . W języku C to są operatory przesunięcia w lewo, w prawo. W C++ operatory te wykonują jak przesunięcie w lewo\prawo, tak i wyjście\wejście, jeśli są użyte z słowem cout\cin .

Insertion to stream cout: **Extraction from stream cin:**

cout << a; **cin >> b;**

Instrukcja C++

cout << “ten wiersz będzie wyświetlony na monitorze” << “\n”;

dokonuje wyprowadzenie wiersza tekstowego w potok standardowy cout , który jest związany z monitorem, jak tylko program C++ zaczyna działanie. Standardowy potok C++ cout jest podobny do potoku C stdout. W języku C to działanie wyglądało by tak:

```
printf(“ten wiersz będzie wyświetlony na monitorze”);
printf(“\n”);
```

Analogicznie, instrukcja

cin >> i

wykonuje czytanie danych z standardowego potoku wejściowego cin , związanego z monitorem, i przypisuje te dane zmiennej o nazwie i. Przetworzenie formatowe jest wykonane automatycznie na podstawie typu zmiennej i . W języku C to działanie (odczyt z standardowego potoku stdin) wyglądało by tak:

scanf(“%d”, &i); //jeśli i – to zmienna typu int
scanf(“%lf”, &i); //jeśli i – to zmienna typu double
.....

Operatory << , >> nadają możliwość wyprowadzać i wprowadzać (nie koniecznie na\z monitor(a)) dane dowolnego typu podstawowego.

W1

Przykłady:

```
double a = 0.3;
int i = 15;
char ch = 'a';
char str[] = “abcde”;

cout << “ a =  “ << a << “ i = “ << i << “ ch = ” << ch << “ str: “ << str << endl;

double b;
cout << “wprowadż dane: “ << endl;
cin >> b;
.....
```

Przykład W0

Przy wprowadzeniu wierszy tekstowych operator >> zatrzymuje się po wykryciu pierwszej spacji. Będzie wprowadzone tylko pierwsze słowo, a pozostała cześć wiersza – nie. Działa to dokładnie tak samo, jak pobieranie wiersza tekstowego funkcją scanf(“%s”, str) ;

Komentarze

/*To jest komentarz wieloliniowy –
wieloliniowe komentarze są dopuszczalne */

//To jest komentarz jednoliniowy

/* To jest komentarz wieloliniowy //To jest włożenie komentarza jednoliniowego w komentarz wieloliniowy*/

Dekłarowanie zmiennych lokalnych

W języku C zmienne lokalne muszą być deklarowane na początku bloku. W C++ zmienne lokalne można deklarować w dowolnym miejscu bloku.

Porada: deklarować zmienne nie na początku kodu tylko w tych wypadkach, kiedy to jest uzasadnione, na przykład, w funkcjach z dużą ilością linij kodu.

Brak domyślnego przekształcania na typ int.

W języku C jeśli funkcje zwraca zmienne, typ której nie jest jawnie zadeklarowany, to domyślnie typ tej zmiennej jest traktowany jako int. W języku C++ typ zwracany przez funkcje, należy zadeklarować jawnie.

Typ bool

W C++ zdefiniowano wbudowany typ logiczny – bool. Obiekt typu bool może przechowywać tylko wartości true, false, które są słowami kluczowymi, zdefiniowanymi w C++. Wartości różne od 0 są przekształcane na wartość true, a zero – na false. Odwrotnie, true jest zamieniane na 1, a false – na 0 .

Stary styl i nowy

Istnieje dwie wersji C++ - stara, którą używali programiści w poprzednim dziesięcioleciu, i nowa – standard C++. Podstawowe założenia obu wersji są podobne, przecież standard C++ zawiera szereg rozszerzeń w stosunku do tradycyjnego C++.

Stary styl można jeszcze często spotkać w programach już napisanych. Nowe programy należy pisać w standardzie C++.

Różnica pomiędzy starym i nowym stylem dotyczy głównie dwóch rzeczy: nowej postaci nagłówków oraz instrukcji namespace.

Stary styl:

```
#include <iostream.h>
int main()
{
   return 0;
}
```

Nowy styl:

```
#include <iostream>
using namespace std;

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

W1

W języku C

#include <nazwa_pliku.h>

powoduje dołączenie pliku o nazwie nazwa_pliku.h .

W standardzie C++

#include <id_nazwa>

jest identyfikatorem standardowym, które mogą być przez kompilator odwzorowane na odpowiednie pliki, lecz nie jest to koniecznie. Używane w C++ nagłówki nowego typu to konstrukcje abstrakcyjne, który gwarantują że zostaną zadeklarowane odpowiednie prototypy i definicje wymagane przez biblioteki C++.

Nagłówki nowego typu nie są nazwami plików, więc nie zawierają rozszerzenia .h

<vector> <fstream> <string> ....

Można używać nagłówki starego typu <string.h> <math.h> .... To nie jest błąd, przecież standard C++ wymaga zastosowywać odpowiedniki : <cstring>, <cmath> ..... Odpowiedniki są konstruowane tak: dodajemy do nazwy nagłówku starego typu „c” i odrzucamy „.h”

Przestrzeń nazw

– to po prostu pewien zadeklarowany obszar. Przestrzenie nazw są stosowane w celu identyfikacji nazw i uniknięcia ich konfliktów. Elementy zadeklarowane w jednej przestrzeni nazw są odseparowane od elementów zadeklarowanych w innej.

W1

```
namespace MY_NAMESPACE_1
{
   int i;
   void fun(int j, char *str);
};

namespace MY_NAMESPACE_2
{
   int i;
   void fun(int j, char *str);
};

void MY_NAMESPACE_1::fun(int j, char *str)
{
   cout << “j =  “ << j << “ str:  “ << str << endl;
}

void MY_NAMESPACE_2::fun(int j, char *str)
{
   cout << “j*j =  “ << j*j << “ str:  “ << str << “ !!!” << endl;
}
```

W1

```
int i;                                          //należy do globalnej przestrzeni nazw
void fun(int j, char *str)
{
   cout << “j*j*j = “ << j*j*j << “str: “ << “[ “ << str << “] “ << endl;
}

int main()
{
   MY_NAMESPACE_1::i = 1;          //należy do przestrzeni nazw MY_NAMESPACE_1
   MY_NAMESPACE_2::i = 2;          //należy do przestrzeni nazw MY_NAMESPACE_2
.....................................................................................................
//To są różne zmienne – są umieszczone w różnych adresach pamięci
// Dla tego nie powstaje konfliktu nazw; :: - operator zakresu.

MY_NAMESPACE_1::fun(10, “abcd”);  //wywołujemy funkcje z przestrzeni nazw
                                                  // MY_NAMESPACE_1
MY_NAMESPACE_2::fun(15, “efgk”);  //wywołujemy funkcje z przestrzeni nazw
                                                  // MY_NAMESPACE_2
fun(20, “tunm”);  //wywołujemy funkcje z globalnej przestrzeni nazw
```

W1

Nazwy funkcji bibliotecznych C są zadeklarowane w globalnej przestrzeni nazw. Nazwy funkcji bibliotecznych C++ umieszczone w przestrzeni nazw std. Instrukcje *using namespace std* udostępnia nazwy, umieszczone w przestrzeni nazw std.