Programowanie strukturalne i obiektowe

Język C – część I

Opracował:
Grzegorz Flesik

Literatura:

- A. Majczak, *Programowanie strukturalne i obiektowe*, Helion, Gliwice 2010
- P. Domka, M. Łokińska, *Programowanie strukturalne i obiektowe*, WSiP, Warszawa 2010

Staszów 2010

Spis treści

1. Struktura programu w C / C++

```
#include <biblioteki_standardowe>  //   dyrektywa preprocesora (dołączanie
                                   //   bibliotek do programu)
...
#include "pliki_nagłówkowe"     // dołączenie własnych plików nagłówkowych
                                // do programu
...

// Deklaracje globalnych zmiennych i stałych
...
// Deklaracje funkcji
...
int main()         // funkcja główna programu
{
  ...
  // instrukcje programu
  ...
  return 0;        // wyjście z programu, program zwraca wartość 0
}
```

2. Wielkie i małe litery

Kompilator C/C++ rozróżnia wielkie i małe litery, zatem ważne jest konsekwentne stosowanie nazw zmiennych, np. `wynik`, `Wynik`, `WYNIK` oznaczają trzy różne zmienne. Należy również pamiętać, że polecenia języka C/C++ piszemy małymi literami.

3. Komentarze

Istnieją dwa rodzaje komentarzy w języku C. Pierwszy z nich używany jest do oznaczenia pojedynczej linii w kodzie programu – znakiem komentarza jest **`//`**. Kiedy chcemy umieścić komentarz składający się z kilku wierszy, używamy znaków **`/*`** do rozpoczęcia komentarza, oraz **`*/`** do jego zakończenia. Przykłady:

```
#include <stdio.h>
int main()
{
 // Komentarz zajmujący jedną linijkę
 // printf("Ten tekst nie zostanie wyświetlony");

 /* komentarz
    zajmujący
    kilka wierszy */
    printf("Ta linijka jest poza komentarzem i zostanie wykonana");

 /* printf("To też się nie wyświetli"); */

 return 0;
}
```

4. Słowa kluczowe

Słowa kluczowe są identyfikatorami zarezerwowanymi dla określonych zadań i nie możemy ich używać jako nazw zmiennych w programie. W przypadku używania zintegrowanego środowiska programistycznego (np. DEV C++) słowa kluczowe w edytorze tekstu są zazwyczaj wyróżnione innym stylem czcionki. Poniżej wyszczególniono listę słów kluczowych języka C:

Instrukcje	**Typy danych**	**Specyfikatory klasy zmiennej**
break	char	auto
case	double	extern
continue	enum	static
do	float	register
else	int	volatile
for	long	
goto	short	
if		
return	**Specyfikatory typów danych**	**Inne**
switch	signed	const
typedef	struct	default
while	union	sizeof
	unsigned	
	void	

5. Funkcja return

Wywołanie instrukcji `return` powoduje natychmiastowe wyjście z funkcji wraz ze zwróceniem wartości do funkcji nadrzędnej. Wywołanie `return` w funkcji głównej programu powoduje jego zakończenie. Przykład:

```
#include <stdio.h>

int suma(int x, int y)   // funkcja obliczająca sumę liczb x i y
{
  return x+y; // wyjście z programu i przekazanie wartości x+y
}

int main()               // funkcja główna programu
{
  printf("3 + 4 = %i", suma(3,4));  // wywołanie funkcji suma()

  return 0;             // wywołanie return w funkcji głównej
                        // - wyjście z programu

  printf("Ten tekst się nie wyświetli, po funkcji return");
}
```

6. Dyrektywa #include

Dyrektywę `#include` stosujemy w pierwszych linijkach kodu źródłowego programu. Służy ona do dołączania zewnętrznych plików źródłowych (tzw. pliki nagłówkowe) zawierających dodatkowe funkcje, które chcemy wykorzystać w programie.
Przy pomocy dyrektywy `#include` możemy dołączać pliki nagłówkowe zawierające podstawowe funkcje języka C/C++, które zainstalowane zostały wraz z kompilatorem. Nazwy standardowych bibliotek zapisujemy w nawiasach kątowych < >. Na przykład:

```
#include <stdio.h>
```

Możemy również dołączać własne pliki nagłówkowe, które zawierają stworzone przez nas funkcje. Zapisywanie funkcji w odrębnych plikach nagłówkowych poprawia przejrzystość kodu programu. Przy dołączaniu własnych plików nagłówkowych stosujemy znaki cudzysłowiu " ". Na przykład:

```
#include "funkcje.h"
```

W odniesieniu do przykładu z poprzedniej strony dla funkcji `return`, w pliku `funkcje.h` moglibyśmy umieścić nagłówek funkcji `suma().`

Plik **funkcje.h**:
```
#ifndef FUNKCJE_H
#define FUNKCJE_H

int suma(int x, int y);

#endif
```

Plik **funkcje.c**:
```
int suma(int x, int y)
{
  return x+y;
}
```

Plik główny programu **program.c**:
```
#include <stdio.h>
#include "funkcje.h”

int main()
{
  printf("3 + 4 = %i”, suma(3,4));
}
```

Należy trzymać się następującego schematu:

- W pliku nagłówkowym z rozszerzeniem **.h** umieszczamy dyrektywy `#ifndef, #define, #endif` tak jak na przykładzie. Zapobiega to przypadkowemu wielokrotnemu przetwarzaniu pliku. Po dyrektywie `#define` umieszczamy tylko typ, nazwę i parametry funkcji (tzw. deklaracje funkcji).
- W pliku źródłowym z rozszerzeniem **.c** umieszczamy zawartości funkcji zadeklarowanych w pliku **.h** (tzw. definicje), czyli jaki kod one wykonują.

- W programie głównym umieszczamy dyrektywę `#include` dołączającą tylko plik nagłówkowy **.h**. Możemy wówczas stosować funkcje zadeklarowane w dołączanym pliku.

7. Zmienne i stałe

7.1. Podstawowe typy zmiennych w C

Typ zmiennej	Opis
`void`	nieokreślony
`char`	pojedynczy znak
`int`	liczba całkowita (16 lub 32 bity)
`float`	liczba zmiennoprzecinkowa pojedynczej precyzji (4 bajty)
`double`	liczba zmiennoprzecinkowa podwójnej precyzji (8 bajtów)
Modyfikatory:	
`unsigned`	nieujemna (zakres liczb od 0)
`signed`	ze znakiem (mogą wystąpić wartości ujemne, domyślnie)
`const`	stała (brak możliwości zmiany wartości)

7.2. Deklaracja zmiennych i stałych

Zmienną deklarujemy podając w pierwszej kolejności jej typ a następnie nazwę:

```
typ zmienna;
```

Można deklarować kilka zmiennych tego samego typu jednocześnie, a także określać ich wartości początkowe:

```
typ zmienna1, zmienna2, zmienna3 = wartosc3, zmienna4 = wartosc4;
```

Dodanie słowa kluczowego `const` przed typem powoduje deklarację stałej, trzeba od razu podać jej wartość. Nie będzie można później jej zmienić.

```
const typ zmienna = wartość;
```

Wykorzystując typ znakowy `char` można tworzyć łańcuchy znaków, które służą do przechowywania tekstu. W nawiasach kwadratowych określamy maks. długość łańcucha.

```
char imie[30] = "Grzegorz";
```

8. Funkcje wejścia-wyjścia w języku C

Funkcja, przykłady	Znaczenie
`printf()`	**wyświetlenie danych na ekranie**
`printf("Tekst\n");`	wyświetla tekst i przechodzi do następnej linii
`printf("x = %i", x);`	wyświetla tekst oraz wartość zmiennej całkowitej x
`scanf()`	**wczytanie danych z klawiatury**
`scanf("%i", &x);`	wczytuje wartość dla zmiennej całkowitej x
`getch()`	**wczytanie pojedynczego znaku z klawiatury**
`c = getch();`	wczytuje znak dla zmiennej c
`puts()`	**wyświetla łańcuch znaków**
`puts("Tekst");`	
`gets()`	**wczytuje łańcuch znaków**
`gets(tekst);`	

Niektóre znaki specjalne, które można zastosować przy wyświetlaniu:

Znak specjalny	Opis
\a	symbol dźwiękowy
\n	przejście do następnego wiersza
\t	tabulator poziomy
\v	tabulator pionowy
\r	powrót na początek bieżącego wiersza
\”	cudzysłów
\’	apostrof
\0	znak zerowy
\\	ukośnik odwrotny (*backslash*)

Funkcja `printf()` umożliwia wyświetlanie wartości zmiennych. Aby to uczynić należy oznaczyć, w którym miejscu wyświetlane i jakiego typu będą to zmienne. Oznaczenia typów zmiennych zamieszczono poniżej.

Oznaczenie	Opis
`%d lub %i`	**liczba całkowita**
`%5d`	wyświetlenie liczby w polu 5-znakowym
`%f`	**liczba zmiennoprzecinkowa (float)**

`%.3f`	wyświetlenie liczby z dokładnością 3 miejsc po przecinku
`%lf`	**liczba zmiennoprzecinkowa (double)**
`%c`	**znak**
`%s`	**łańcuch znaków**

9. Operatory

Operator	Przykład	Znaczenie
Arytmetyczne		
+	x + y	dodawanie
-	x – y	odejmowanie
*	x * y	mnożenie
/	x / y	dzielenie
%	x % y	reszta z dzielenia
++	x++ ++x	inkrementacja
--	x-- --x	dekrementacja
Przypisania		
=	x = 5	przypisanie wartości
+= -= *= itd.	x += 5	złożone operatory przypisania x += 5 – to samo co x = x+5
Relacyjne		
<	x < y	mniejszy
<=	x <= y	mniejszy lub równy
>	x > y	większy
>=	x >= y	większy lub równy
!=	x != y	różny
==	x == y	równy
Logiczne		
||	(a == 0) || (a == 1)	alternatywa (suma logiczna)
&&	(a > 0) && (a < 10)	koniunkcja (iloczyn logiczny)
!	!(a==3)	negacja
Bitowe logiczne		
|	x | y	alternatywa bitowa
&	x & y	koniunkcja bitowa
^	x ^ y	bitowa różnica symetryczna
<<	x << y	bitowe przesunięcie w lewo
>>	x >> y	bitowe przesunięcie w prawo
~	~x	negacja bitowa

10. Instrukcje sterujące

10.1. Instrukcja warunkowa `if`

Składnia instrukcji warunkowej `if` wygląda następująco:

```
if (warunek)
  instrukcja1; //wykonywana, gdy warunek jest spełniony
else
  instrukcja2; // wykonywana, gdy warunek nie jest spełniony
```

W przypadku gdy jest więcej do wykonania niż jedna instrukcja, wtedy zamykamy je w nawiasach klamrowych { }. Nie jest konieczne stosowanie polecenia `else`. Przykład:

```
#include <stdio.h>
int main()
{
  int x;
  printf("Podaj liczbę x: "); scanf("%i", &x);
  if (x > 0 && x < 100)
    printf("Liczba mieści się w przedziale (0,100)");
  else
  {
    printf("Liczba znajduje się poza przedziałem (0, 100)\n");
    printf("Uruchom program ponownie");
  }
  return 0;
}
```

10.2. Instrukcja `switch`

Instrukcja `switch` umożliwia wybranie jednego spośród wielu wariantów. Podawane jest wyrażenie warunkowe (zmienna) oraz jego wartości, jakie może ono przyjmować. W zależności od wartości wyrażenia realizowana jest odpowiednia instrukcja. Składnia instrukcji `switch` wygląda następująco:

```
switch (wyrażenie)
{
  case wartosc1: instrukcja1;
                 ...
  case wartosc2: instrukcja2;
                 ...
  ...
  default: instrukcja_domyslna;
}
```

Słowo kluczowe `default` określa instrukcje, które mają się wykonać jeśli żaden z wymienionych wariantów nie został wybrany. W razie potrzeby po każdym wariancie należy dodać instrukcję `break`, aby przerwać wykonywanie kolejnych wyrażeń. Oto przykład:

```
#include <stdio.h>
int main()
{
```

```
  int x;
  printf("Podaj liczbe od 1 do 3: "); scanf("%i", &x);
  switch (x)
  {
    case 1: printf("Podałeś jeden"); break;
    case 2: printf("Podałeś dwa"); break;
    case 3: printf("Podałeś trzy"); break;
    default: printf("Podałeś inną liczbę!");
  }
  return 0;
}
```

10.3. Instrukcja `while`

Instrukcja `while` umożliwia wykonywanie instrukcji w pętli dopóki jest spełniony określony warunek. Warunek ten może być sprawdzany zarówno na początku, jak i na końcu pętli. Poniżej podano składnię dla obydwu wymienionych wariantów.

```
while (warunek) instrukcja;         // warunek sprawdzany na początku

do instrukcja while (warunek);      // warunek sprawdzany na końcu
```

Poniższy program przedstawia przykład zastosowania pętli `while` - wypisuje tekst dopóki użytkownik nie wciśnie klawisza ‘x’:

```
#include <stdio.h>
#include <stdlib.h>

int main()
{
  char c;
  do
  {
    printf("Trwa pętla\n");
    c = getch();
  } while (c != 'x');
  return 0;
}
```

10.4. Pętla `for`

Pętla `for` jest rozbudowaną wersją pętli `while`. W odróżnieniu do instrukcji `while`, `for` posiada 3 parametry – instrukcję inicjalizującą, warunek oraz instrukcję krokową. Składnia pętli `for` wygląda następująco:

```
for (inicjalizacja; warunek; instrukcja_krokowa)
{
...
}
```

Standardowo pętlę `for` wykorzystujemy, gdy określone instrukcje mają wykonać się określoną ilość razy. Wówczas składnia polecenia przypomina pętlę `for` napisaną w języku Pascal. Na przykład:

```
for (i = 0; i < 10; i++)
{
  printf("%i ", i);
}
```

Powyższy fragment można opisać następująco: przypisz zmiennej `i` wartość 0. Dopóki `i` jest mniejsze od 10, wykonuj instrukcję `printf`, za każdym razem zwiększając `i` o 1. Pętla wykona się 10 razy dając ostatecznie na wyjściu: 0 1 2 3 4 5 6 7 8 9.

10.5. Instrukcje `break` i `continue`

Instrukcja break, jak już opisano, może być wykorzystywana w obrębie instrukcji `switch` – powoduje wówczas bezwarunkowe wyjście poza jej obręb. Instrukcję `break` można też stosować w obrębie pętli `while` oraz `for` – jej działanie jest takie samo – powoduje zakończenie działania pętli. Przykład – przerwanie wykonywania pętli po naciśnięciu klawisza ‘q’:

```
#include <stdio.h>

int main()
{
  char c;
  int i;
  for (i = 0; i < 10; i++)
  {
    c = getch();
    if (c == 'q') break;
    printf("Naciśnięto klawisz: %c\n", c);
  }
  return 0;
}
```

Z kolei instrukcja `continue` zastosowana w obrębie pętli powoduje pominięcie instrukcji znajdujących się poniżej, ale pętla jest kontynuowana od nowa. Poniżej przedstawiono przykład zastosowania instrukcji `continue` – obliczanie w pętli sumy liczb nieparzystych z zakresu 0-20.

```
#include <stdio.h>

int main()
{
  int i, suma = 0;
  for (i = 0; i < 20; i++)
  {
    if (i % 2 == 0) continue;
    suma += i;
  }
  printf("Suma liczb parzystych od 0 do 10 wynosi %i\n", suma);
  return 0;
}
```