Wstęp do Informatyki i Programowania
Elementy języków C i Ada

Przemysław Kobylański

Elementy języka C

Główna funkcja

Każdy program w C musi zawierać przynajmniej funkcję o nazwie `main()`:

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

Deklaracje zmiennych

Każda zmienna jaką używamy w funkcji do przechowywania wartości musi być zadeklarowana na początku funkcji:

```
int main(void)
{
  int x;
  int y;
  int z;
  ...
  return 0;
}
```

Czytanie danych

Aby możliwe było czytanie danych i drukowanie wyników, trzeba dołączyć plik nagłówkowy biblioteki standardowego wejścia i wyjścia (stdio). Wykonuje się to umieszczając na początku programu dyrektywę `include`:

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

Do przeczytania wartości służy funkcja `scanf()`:

```c
#include <stdio.h>
int main(void)
{
  int x;
  int y;
  int z;
  scanf("%d", &x);
  scanf("%d", &y);
  ...
  return 0;
}
```

Wartości całkowite typu `int` czyta się formatem `%d`, natomiast wartości rzeczywiste typu `float` (pojedynczej dokładności) formatem `%f`.

Instrukcja podstawienia

Aby podstawić wyliczoną wartość pod zmienną należy użyć instrukcję podstawienia:

```c
#include <stdio.h>
int main(void)
{
  int x;
  int y;
  int z;
  scanf("%d", &x);
  scanf("%d", &y);
  z = x+y;
  ...
  return 0;
}
```

Drukowanie wyników

Wartość dowolnego wyrażenia (w tym tak prostego jak złożonego z pojedynczej zmiennej) można wydrukować za pomocą funkcji `printf()`:

```c
#include <stdio.h>
int main(void)
{
  int x;
  int y;
  int z;
  scanf("%d", &x);
  scanf("%d", &y);
  z = x+y;
  printf("Suma %d i %d wynosi %d.\n", x, y, z);
  return 0;
}
```

Wartości całkowite drukuje się przy użyciu formatu `%d`, natomiast rzeczywiste formatem `%f`.

Instrukcja warunkowa

Instrukcja warunkowa w języku C ma następującą postać:

```
if(Warunek)
{
  // instrukcje wykonywane gdy warunek jest spełniony
}
else
{
  // instrukcje wykonywane gdy warunek nie jest spełniony
}
```

Jeśli w którymś z dwóch bloków ujętych w nawiasy klamrowe jest jedna instrukcja, to nawiasy otaczające tę jedną instrukcję można pominąć:

```
if(Warunek)
  // jedna instrukcja wykonywana gdy warunek jest spełniony
else
{
  // instrukcje wykonywane gdy warunek nie jest spełniony
}
```

Jeśli blok po słowie kluczowym **else** nie zawiera żadnej instrukcji, to można go pominąć:

```
if(Warunek)
{
  // instrukcje wykonywane gdy warunek jest spełniony
}
```

Przykłady

```
if(x > 0.0)
  y = sqrt(x);
else
  y = 0.0;

if(n > 0)
  printf("liczba jest dodatnia");

if(wiek >= 18)
  printf("osoba jest pelnoletnia");
else
  printf("osoba nie jest pelnoletnia");
```

Instrukcja pętli

Jeśli pewien fragment programu ma wykonać się n razy, to należy użyć pętli **for**:

```
for(int i = 0; i < n; i++)
{
  // instrukcje powtarzane n razy dla i = 0, 1, ..., n - 1
}
```

Podobnie jak w przypadku instrukcji warunkowej, nawiasy klamrowe otaczające jedną instrukcję można pominąć.

Instrukcje pętli mogą być zagnieżdżone jedna w drugiej.

Przykłady

W wyniku wykonania poniższego fragmentu programu:

```
int n = 3;
for(int i = 0; i < n; i++)
{
  for(int j = 0; j <= i; j++)
  {
    printf("i = %d, j = %d\n", i, j);
  }
}
```

zostaną wydrukowane następujące wiersze:

```
i = 0, j = 0
i = 1, j = 0
i = 1, j = 1
i = 2, j = 0
i = 2, j = 1
i = 2, j = 2
```

Zwróć uwagę, że wywołanie funkcji **printf** jest jedyną instrukcją w bloku w wewnętrznej pętli **for**, zatem nawiasy klamrowe wokół niej można pominąć. Podobnie wewnętrzna pętla jest jedyną instrukcją w bloku zewnętrznej pętli **for**, zatem nawiasy klamrowe wokół niej można również pominąć:

```
int n = 3;
for(int i = 0; i < n; i++)
  for(int j = 0; j <= i; j++)
    printf("i = %d, j = %d\n", i, j);
```

Kompilacja programu w języku C

Do kompilacji programów w języku C używać będziemy polecenia `cc` (ang. C compiler)

Załóżmy, że następujące źródła programu znajdują się w pliku `witaj.c`:

```
#include <stdio.h>
int main(void)
{
  printf("witaj swiecie!\n");
}
```

Aby skompilować program wystarczy wydać polecenie: `cc witaj.c`

Skompilowany program wykonywalny znalazł się w pliku `a.out`

Można teraz go uruchomić wydając polecenie: `./a.out`

Zwróć uwagę na kropkę w powyższym poleceniu, która wskazuje, że plik `a.out` znajduje się w bieżącym katalogu.

Jeśli chcemy aby wykonywalny program znalazł się w pliku `witaj`, zamiast `a.out`, należy wydać polecenie: `cc -o witaj witaj.c`

Opcja `-o` służy do określenia nazwy pliku wykonywalnego.

Programy kompilować będziemy z dodatkową opcją `-Wall` nakazującą kompilatorowi informować o wszystkich ostrzeżeniach:

```
$ cc -Wall -o witaj witaj.c
witaj.c: In function ‘main’:
witaj.c:5: warning: control reaches end of non-void function
$
```

Ostrzeżenie **control reaches end of non-void function** oznacza, że w funkcji main (dokładniej w wierszu 5 pliku `witaj.c`) sterowanie, tj. wykonywanie, osiąga klamrę zamykającą definicję funkcji, mimo tego, że została ona zadeklarowana jako zwracająca wartość typu całkowitego (`int`).

Na program należy poprawić dopisując między instrukcją `printf(...)` a klamrą `}` instrukcję `return 0;`

Program po poprawieniu wygląda następująco:

```c
#include <stdio.h>
int main(void)
{
  printf("witaj swiecie!\n");
  return 0;
}
```

Tym razem kompiluje się beż żadnych ostrzeżeń:

```
$ cc -Wall -o witaj witaj.c
$
```

Jeśli w programie korzysta się z funkcji bibliotecznych, to należy za pomocą opcji `-l` podać nazwę wykorzystywanej biblioteki.

Dla przykładu, jeśli program `calkowanie.c` korzysta z funkcji matematycznych, to kompilujemy go poleceniem: `cc -o calkowanie calkowanie.c -lm`

Opcja `-lm` w powyższym poleceniu zawiera nazwę `m` biblioteki funkcji matematycznych.

Elementy języka Ada

Główna procedura

Każdy program w języku Ada musi zawierać główną procedurę, przy czym może ona nazywać się dowolnie. Ważne aby procedura znajdowała się w pliku o takiej samej nazwie ale z rozszerzeniem `.adb`.

Załóżmy, że główna procedura będzie nazywać się **Moj_Program**.

Wówczas najprostszym programem z taką procedurą jest:

```
procedure Moj_Program is
begin
  null;
end Moj_Program;
```

Zwróć uwagę, że nie można zostawić pustego bloku **begin end Moj_Program** dlatego należało w nim umieścić pustą instrukcję **null**.

Deklaracja zmiennych

Wszystkie zmienna użyte w procedurze muszą być zadeklarowane między słowem kluczowym **is** a słowem kluczowym **begin**:

```
procedure Moj_Program is
  X : Integer;
  Y : Integer;
  Z : Integer;
begin
  null;
end Moj_Program;
```

Po dwukropku podaje się typ zmiennej. Wsród dostępnych typów są między innymi:

- **Integer** - liczba całkowita,
- **Float** - liczba rzeczywista pojedynczej precyzji,
- **Long_Float** - liczba rzeczywista podwójnej precyzji.

Czytanie danych

Do czytania wartości zmiennej używać będziemy procedury **Get**. W zależności od typu czytanej wartości należy skorzystać z odpowiedniego pakietu:

- **Ada.Text_IO** gdy czytam napisy,
- **Ada.Integer_Text_IO** gdy czytamy liczbę całkowitą,
- **Ada.Float_Text_IO** gdy czytamy liczbę rzeczywistą pojedynczej precyzji,
- **Ada.Long_Float_Text_IO** gdy czytamy liczbę rzeczywistą podwójnej precyzji.

Oto jak przeczytać wartości zmiennych całkowitych:

```
with Ada.Integer_Text_IO;
procedure Moj_Program is
  X : Integer;
  Y : Integer;
  Z : Integer;
begin
  Ada.Integer_Text_IO.Get (X);
  Ada.Integer_Text_IO.Get (Y);
end Moj_Program;
```

Przy wywołaniu procedury **Get** można pominąć prefiks **Ada.Integer_Text_IO** ale trzeba wtedy użyć frazy **use** po frazie **with**:

```
with Ada.Integer_Text_IO; use Ada.Integer_Text_IO;
procedure Moj_Program is
  X : Integer;
  Y : Integer;
  Z : Integer;
begin
  Get (X);
  Get (Y);
end Moj_Program;
```

Instrukcja podstawienia

Aby podstawić wyliczoną wartość pod zmienną należy użyć instrukcji podstawienia:

```
with Ada.Integer_Text_IO; use Ada.Integer_Text_IO;
procedure Moj_Program is
  X : Integer;
  Y : Integer;
  Z : Integer;
begin
  Get (X);
  Get (Y);
  Z := X + Y;
end Moj_Program;
```

Drukowanie wyników

Do drukowania wartości używać będziemy procedury **Put**, która dla różnych typów wartości znajduje się w odpowiednich pakietach (tych samych co dla procedury **Get**).

```
with Ada.Integer_Text_IO; use Ada.Integer_Text_IO;
procedure Moj_Program is
  X : Integer;
  Y : Integer;
  Z : Integer;
begin
  Get (X);
  Get (Y);
  Z := X + Y;
  Put (Z);
end Moj_Program;
```

Szczególną procedurą jest **New_Line**, której wykonanie powoduje przejście z wydrukiem na początek nowego wiersza. Znajduje się ona w pakiecie **Ada.Text_IO**.

Instrukcja warunkowa

Instrukcja warunkowa w języku Ada ma następującą postać:

```
if Warunek then
  --  instrukcje wykonywane gdy warunek jest prawdziwy
else
  --  instrukcje wykonywane gdy warunek jest fałszywy
end if;
```

Jeśli blok instrukcji po słowie kluczowym **else** nie zawiera instrukcji, to trzeba go opuścić:

```
if Warunek then
  -- instrukcje wykonywane gdy warunek jest prawdziwy
end if;
```

albo wpisać w nim słowo kluczowe **null** (nie może być pusty jak to jest możliwe w języku C):

```
if Warunek then
  --  instrukcje wykonywane gdy warunek jest prawdziwy
else
  null;
end if;
```

Przykłady

```
if X > 0.0 then
  Y := Sqrt(X);
else
  Y := 0.0;
end if;

if N > 0 then
  Put ("liczba jest dodatnia");
end if;

if Wiek >= 18 then
  Put ("osoba jest pełnoletnia");
else
  Put ("osoba nie jest pełnoletnia");
end if;
```

Instrukcja pętli

Jeśli pewien blok instrukcji ma być wykonany wielokrotnie, to należy go umieścić wewnątrz instrukcji pętli. Ma ona w języku Ada następującą postać:

```
for Zmienna in Zakres loop
  -- instrukcje, których wykonanie należy powtórzyć
end loop;
```

Podczas kolejnych wykonań instrukcji zmienna będzie przyjmować kolejne wartości za danego zakresu.

Przykłady

W wyniku wykonania poniższego fragmentu kodu:

```
for I in 1 .. 3 loop
  Put ("TAK ");
end loop;
New_Line;
```

zostanie wydrukowany napis:

```
TAK TAK TAK
```

i bieżąca pozycja drukowania zostanie przeniesiona na początek kolejnego wiersza.

Poniższy program drukuje tabliczkę mnożenia:

```
with Ada.Integer_Text_IO; use Ada.Integer_Text_IO;
with Ada.Text_IO;         use Ada.Text_IO;
procedure Tabliczka_Mnozenia is
begin
  for I in 1 .. 10 loop
    for J in 1 .. 10 loop
      Put (Item => I * J, Width => 3);
    end loop;
    New_Line;
  end loop;
end Tabliczka_Mnozenia;
```

W wywołaniu procedury **Put** jawnie wskazano, że **Item**, czyli wartość do drukowania, to iloczyn **I * J** i ma być ona wydrukowana na polu szerokości trzech znaków (parametr **Width**).

Efekt działania powyższego programu:

```
  1   2   3   4   5   6   7   8   9  10
  2   4   6   8  10  12  14  16  18  20
  3   6   9  12  15  18  21  24  27  30
  4   8  12  16  20  24  28  32  36  40
  5  10  15  20  25  30  35  40  45  50
  6  12  18  24  30  36  42  48  54  60
  7  14  21  28  35  42  49  56  63  70
  8  16  24  32  40  48  56  64  72  80
  9  18  27  36  45  54  63  72  81  90
 10  20  30  40  50  60  70  80  90 100
```

Kompilacja programu w języku Ada

Do kompilowania programów w języku Ada będziemy używać polecenia **gnatmake**. W tym celu wystarczy wywołać to polecenie podając jako parametr nazwę pliku, w którym znajduje się główna procedura programu.

Dla przykładu, aby skompilować program:

```
with Ada.Integer_Text_IO; use Ada.Integer_Text_IO;
procedure Moj_Program is
  X : Integer;
  Y : Integer;
  Z : Integer;
```

```
begin
  Get (X);
  Get (Y);
  Z := X + Y;
  Put (Z);
end Moj_Program;
```

wywołujemy:

```
$ gnatmake moj_program.adb
gcc -c moj_program.adb
gnatbind -x moj_program.ali
gnatlink moj_program.ali
```

W wyniku kompilacji i konsolidacji postaje program wykonywalny o nazwie `moj_program`.

Efekt jego uruchomienia:

```
$ ./moj_program
12
36
         48
```

Po uruchomieniu programu wpisano dwie liczby całkowite i pojawił się wynik będący ich sumą.

A Podstawowe polecenia Linuxa

polecenie	**opis**	**przykłady**
`mkdir NAZWA` `mkdir ŚCIEŻKA/NAZWA`	stworzenie katalogu	`mkdir zadanie1` `mkdir zadanie1/src`
`rmdir NAZWA` `rmdir ŚCIEŻKA/NAZWA`	usunięcie katalogu (katalog musi być pusty)	`rmdir zadanie1` `rmdir ../zadanie2/src`
`cd ŚCIEŻKA`	przejście do katalogu	`cd zadanie1/src` `cd ../../zadanie2/src`

`ls`
`ls -l`
 	
`ls ŚCIEŻKA`
`ls -l ŚCIEŻKA`
	wyświetlenie zawartości
	bieżącego katalogu
	(opcja pełnego opisu plików);
	wyświetlenie zawartości
	katalogu
		`ls`
		`ls -l`
		
		`ls zadanie1/src`
		`ls -l ../../zadanie2/src`

`nano` `nano ŚCIEŻKA/NAZWA`	edycja pliku tekstowego (na dole ekranu opis skrótów klawiszowych dla podstawowych operacji, np. ^X oznacza Ctrl-x)	`nano` `nano src/main.c`
`man POLECENIE`	opis polecenia	`man man` `man nano`