

Funkcje

Rozdział ten poświęcony jest tematyce budowy i zastosowania funkcji. Funkcja jest jednym z podstawowych narzędzi wykorzystywanych w pisaniu programów. Funkcje mogą być dwojakiego pochodzenia. W większości przypadków stosuje się gotowe rozwiązania zawarte w bibliotekach językowych. Do nich zaliczamy takie funkcje jak *printf()*, *scanf()* i wiele innych, które Czytelnik poznał już w poprzednich rozdziałach. Jednak równie często programista musi stworzyć nowe, własne funkcje, które będą w stanie wykonać powierzone im zadania. W praktyce programowania praktycznie niemożliwym jest niestosowanie funkcji. Czytelnik analizując kolejne przykłady będzie mógł nauczyć się nie tylko zastosowania możliwości funkcji wbudowanych, ale również budowania funkcji własnych. Umiejętności te są kluczowe dla zrozumienia trzech ostatnich rozdziałów, które omawiają zaawansowane algorytmy sortowania danych i metody macierzowe.

Warto również zwrócić uwagę, w jaki sposób definiują funkcje autorzy publikacji [3, 5, 7, 18-20, 31, 35- 39, 42-48]. Opanowanie przedstawionej w kolejnych przykładach wiedzy pozwoli na pogłębienie umiejętności programowania i nabranie cennego doświadczenia. Zasady budowy i wykorzystania różnych funkcji w kodzie programu są zbliżone we wszystkich językach, zatem umiejętności omówione w tym rozdziale mają duże znaczenie w poszerzeniu wiedzy Czytelnika.

Funkcja w języku C/C++

Spróbujemy teraz wyjaśnić pojęcie funkcji. Naturalnym skojarzeniem jest odwołanie do matematyki i znanego z niej pojęcia funkcji. Jest to dobre porównanie. Różnica tkwi w samym działaniu funkcji.

Klasyczne, matematyczne rozumienie funkcji opiera się na wyznaczaniu wartości funkcji dla argumentów. Zatem takie podejście do funkcji jest numerycznym rozumowaniem, gdyż dla określonych wartości funkcja zwraca, wyznaczone przez odpowiedni wzór wartości. W programowaniu i szeroko rozumianej informatyce funkcja działa podobnie, jednak tutaj nie ma ona jedynie charakteru numerycznego. W kodzie programu możemy zaimplementować funkcję, której zadaniem będzie przepisywanie podanego tekstu w odwrotnej kolejności lub operowanie na danych wyszukując w nich odpowiedniego elementu. Często też mamy do czynienia z funkcjami dostępnymi już natywnie w systemie czy kompilatorze danego języka. W programowaniu na ogół używamy funkcji dostarczanych w bibliotekach standardowych. W języku C/C++ będą to najczęściej funkcje z bibliotek *stdio.h*, *math.h*, *stdlib.h* czy też *string.h*, omówionych na początku książki.

. Istnieje również możliwość tworzenia nowych funkcji, jakie są nam potrzebne w konkretnym przypadku. Programista może sam zdefiniować funkcję, którą zastosuje w programie. Najczęstszym powodem implementowania i pisania własnych funkcji jest:

- wielokrotnie wykonywanie tych samych obliczeń czy operacji,
- czytelne wyróżnienie instrukcji w kodzie programu, ułatwiająca późniejszą reedycję kodu lub możliwość wielokrotnego użycia tej samej procedury w całym programie.

Implementowanie własnej funkcji polega na określeniu wejścia do funkcji i wyniku, jaki ma ona zwracać. W wielu językach programowania spotykamy określenia funkcji jako procedury, która nie zwraca określonych wartości w sposób jawny (funkcja typu *void*) a jedynie operuje na przekazanych obiektach z funkcji wywołującej. Autorzy chcą pokazać funkcje operujące na wartościach i zwracające określone wyniki. Zastanówmy się nad umieszczaniem funkcji w kodzie programu.

Na ogół program składa się z wielu małych funkcji, które możemy deklarować przed funkcją główną w takiej kolejności, aby w momencie deklaracji wszystkie użyte funkcje były uprzednio zadeklarowane. Drugim sposobem jest umieszczenie w nagłówku programu predefinicji funkcji informującej kompilator, że danego typu funkcja wystąpi i zostanie zadeklarowana w późniejszej części programu. Formalna definicja funkcji wygląda następująco:

<pre>typ_funkcji nazwa_funkcji (jeżeli istnieją to lista argumentów) { Deklaracje zmiennych i tablic używanych przez funkcję; Instrukcje programu; return wyrażenie; }</pre>	<pre>/* Zmienne lokalne, dostępne tylko wewnątrz funkcji, są one niewidoczne w innych funkcjach programu.*/ /* Instrukcja <i>return</i> przekazuje wartość obliczonego wyrażenia do funkcji, z której została wywołana. Przekazanie wartości następuje przez nazwę funkcji.*/></pre>
---	--

Przedstawmy działanie funkcji na praktycznym przykładzie.

Przykład 25

Napisać program obliczający wartość największą spośród zadanych n liczb przy pomocy osobnej funkcji

<pre>#include<stdio.h> #include<conio.h> int max(int n,int *a) { int i, m; m = a[0]; for(i = 1; i < n; i++) if(a[i]>m) m=a[i]; return m; }</pre>	<pre>// Deklaracja biblioteki. /* Implementacja funkcji własnej liczącej maksimum n elementów. Funkcja <i>max()</i> jest wywoływana na dwóch argumentach.*/ /* Algorytm obliczania największej wartości.*/ // Zwracana wartość funkcji.</pre>
--	---

```
int main()
{
    int n, i, naj;

    printf("Podaj n:");
    scanf("%d", &n);

    int *a = new int [n];

    for(i = 0; i < n; i++)
    {
        printf("Podaj a[%d]:", i);
        scanf("%d", a+i);
    }
    naj=max(n,a);
    printf("max=%d\n", naj);

    _getch();
    return 0;
}
```

// Główna funkcja programu.

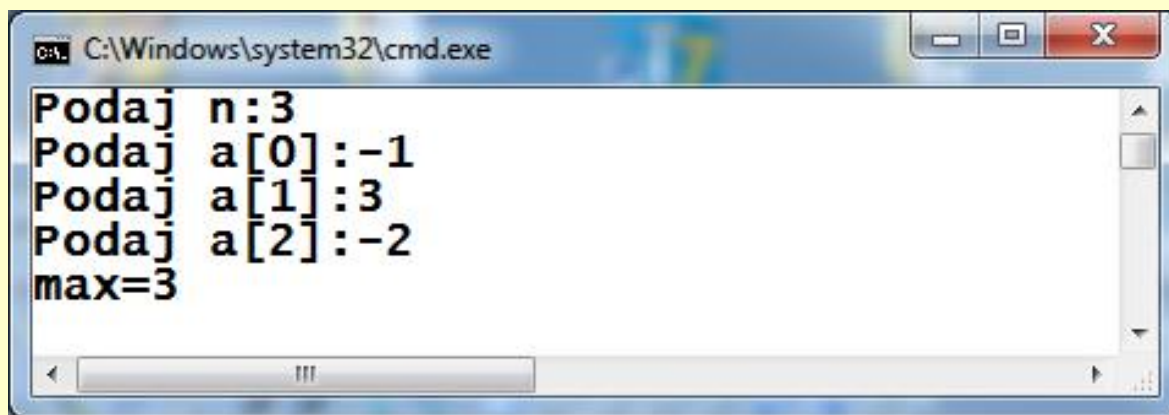
// Wczytanie liczby elementów.

/* Utworzenie tablicy dla n
pobranych elementów.*/

/* Pobranie n elementów do
tablicy.*/

/* Wywołanie funkcji i jednocześnie
zapamiętanie zwróconej wartości w
zmiennej naj.*/

Działanie naszego programu na konsoli systemowej przedstawia Rys. 64. Przekazanie argumentów wywołania funkcji odbywa się za pomocą przekazania wartości dla zmiennych i przekazania adresu tablicy. Dla zmiennych używanych w funkcji tworzone są zmienne lokalne, do których kopiowana jest wartość zmiennych. Tak więc zmiana wartości zmiennych lokalnych funkcji nie powoduje zmiany wartości zmiennych funkcji wywołującej.



```
C:\Windows\system32\cmd.exe

Podaż n:3
Podaż a[0]:-1
Podaż a[1]:3
Podaż a[2]:-2
max=3
```

The image shows a screenshot of a Windows command prompt window. The title bar indicates the path 'C:\Windows\system32\cmd.exe'. The window contains the following text: 'Podaż n:3', 'Podaż a[0]:-1', 'Podaż a[1]:3', 'Podaż a[2]:-2', and 'max=3'. The text is displayed in a monospaced font, and the window has standard Windows window controls (minimize, maximize, close) in the top right corner.

Gdybyśmy przykładowo dokonali zmiany wartości wymiaru zadania n w funkcji obliczającej największą wartość, to i tak w programie głównym wartość n pozostanie niezmienną, gdyż w funkcji wywoływanej wszystkie obliczenia są wykonywane na zmiennych lokalnych. Dotyczy to również zmiennych wskaźnikowych.

Funkcje rekurencyjne

Często odwołujemy się do powtarzania pewnej operacji przez zastosowanie rekurencji. Oznacza to, że funkcja wykonuje zadaną operację w pewien powtarzalny sposób. W języku C funkcja może wywoływać samą siebie. Podczas wywołania rekurencyjnego sporządzana jest kopia wszystkich zmiennych używanych przez funkcję wywołującą. Następnie przekazywane są argumenty do funkcji wywołanej. W przykładach pokażemy podstawowe możliwości wykorzystania rekurencji. Czytelnika zachęcamy do zapoznania się z przykładami przedstawionymi w literaturze [20-23, 29, 34, 39, 42-48, 58]. Obecnie przedstawimy rekurencyjny algorytm obliczania silni i porównajmy go z algorytmem tradycyjnym wykorzystującym pętlę.

Przykład 26

Napisać program obliczający silnię podanej liczby.

Metoda rekurencyjna ma następującą postać:

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

int silnia(int n)
{
    if(n>0)
        return n*silnia(n-1);
    else
        return 1;
}
```

```
// Deklaracja bibliotek.
/* Deklaracja funkcji liczącej
   silnię w sposób
   rekurencyjny.*/

/* Wywołanie działania funkcji
   silnia w zwracanej zmiennej n
   w sposób rekurencyjny.*/
```

```
int main()
{
    int k, r;

    printf("Podaj n=");
    scanf("%d",&k);

    r = silnia(k);

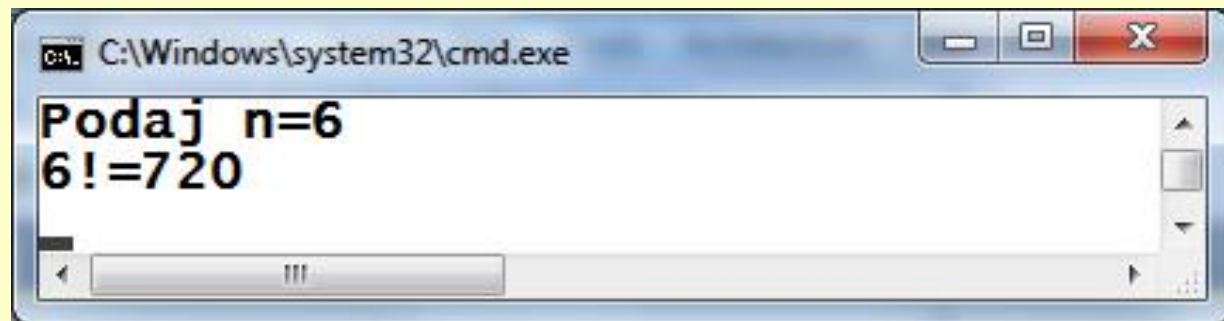
    printf("%d!=%d\n",k,r);

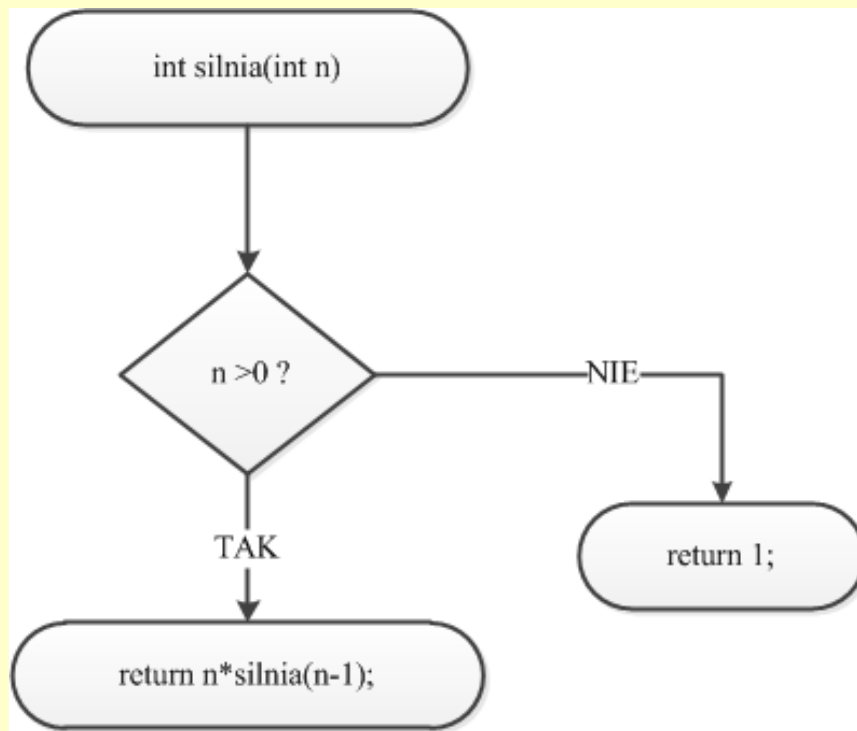
    _getch();
    return 0;
}
```

// Silnik programu.

/* Wczytanie podstawy
wykonania operacji silni.*/

/* Wykonanie obliczenia silni
poprzez wywołanie funkcji silnia
na wczytanym argumencie.*/





Jak widać, schemat blokowy metody rekurencyjnej przyjmuje bardzo uproszczoną postać. Metoda rekurencyjna pozwala na uproszczenie kodu. Program wyznaczający silnię można również zrealizować używając klasycznej pętli. Metoda klasyczna z użyciem pętli przyjmie postać:

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

```
int silnia(int n);
```

```
int main()
```

```
{
```

```
    int k, r;
```

```
    printf("Podaj n=");
```

```
    scanf("%d", &k);
```

```
    r=silnia(k);
```

```
    printf("silnia(%d)=%d\n", k, r);
```

```
}
```

```
int silnia(int n)
```

```
{
```

```
    int i, s;
```

```
    s = 1;
```

```
    for(i = 1; i <= n; i++)
```

```
        s*=i;
```

```
    return s;
```

```
}
```

```
// Deklaracja bibliotek
```

```
// Deklaracja występowania funkcji  
własnej.
```

```
// Silnik programu
```

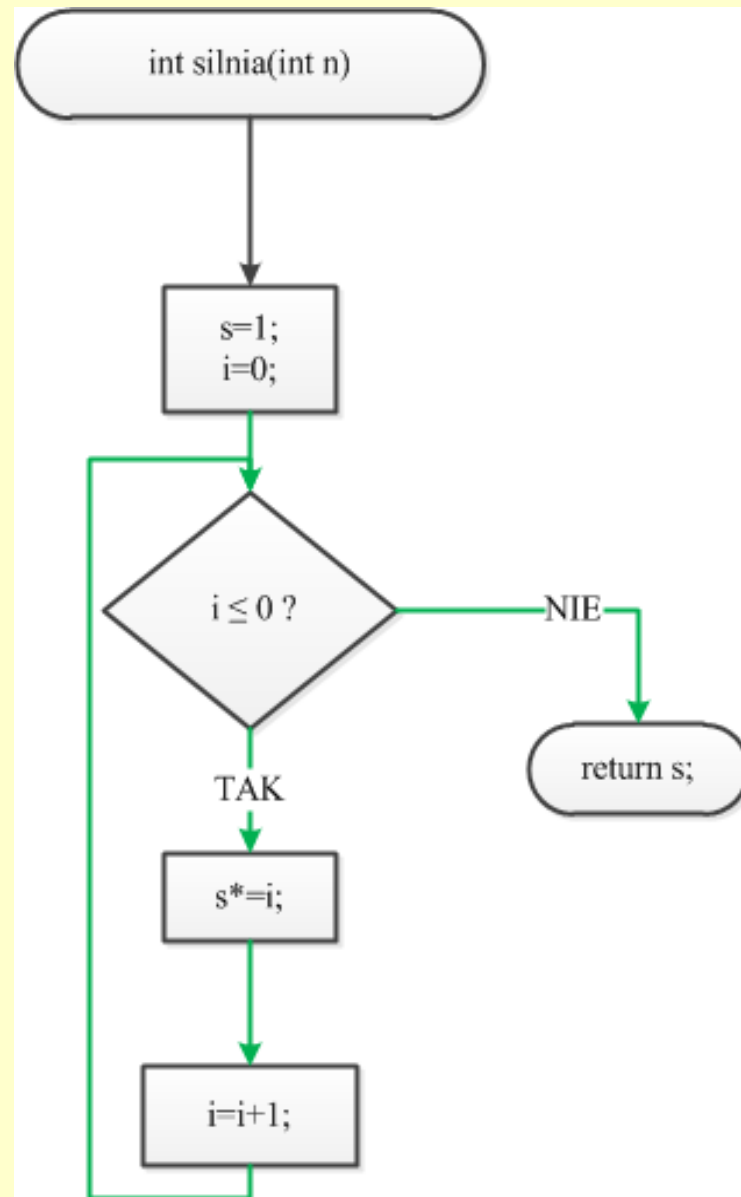
```
/ * Wczytanie podstawy wykonania  
operacji silni. */
```

```
/* Wykonanie obliczenia silni  
poprzez wywołanie funkcji silnia na  
wczytanym argumencie. */
```

```
/* Deklaracja funkcji liczącej silnię  
poprzez pętlę. */
```

Schemat blokowy takiego rozwiązania jest bardziej skomplikowany niż przedstawiony na poprzedniej ilustracji.

Czytelnik zechce porównać obie metody i przeanalizować różnice między nimi na podstawie przedstawionych schematów blokowych. Obie przedstawione metody są zaimplementowane w kodzie głównym programu jako funkcja, do której odwołujemy się w trakcie realizacji procedury głównej.



Argumenty wywołania funkcji `main()`

Programista może korzystać z wielu własności języka. Jedną z nich jest możliwość wywoływania funkcji z bezpośrednimi argumentami. W języku C można przekazywać argumenty do wywoływanego programu bezpośrednio z linii poleceń. Pierwszy argument w funkcji `main()` określa liczbę argumentów znajdującą się w linii poleceń. Drugi jest wskaźnikiem do tablicy znakowej, której wierszami są argumenty uruchomionego programu zapamiętane jako tekst i zakończone znakiem `'\0'`. W postaci kodu argumenty te są zawarte w nawiasie wywoływanej funkcji.

Przykład 27

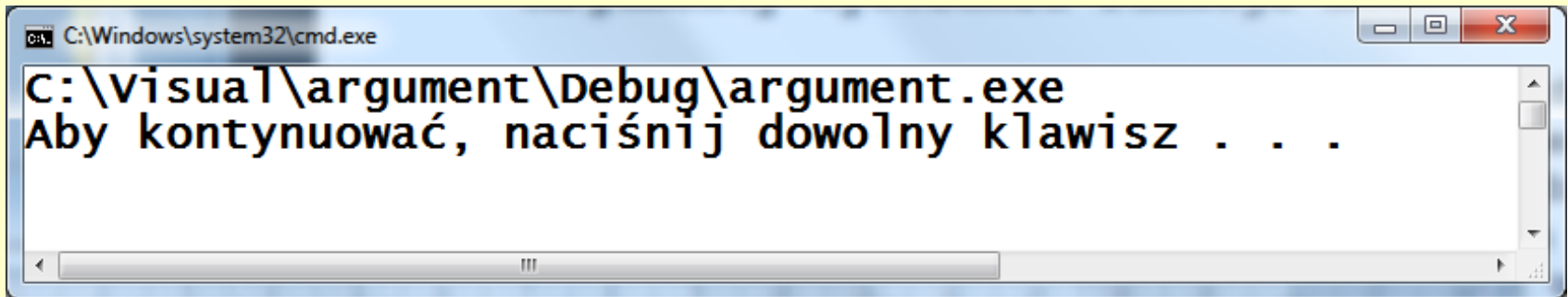
Napisać program, który będzie wypisywał argumenty pobrane bezpośrednio z linii poleceń.

```
#include<stdio.h>

int main(int argc,char *argv[])
{
    int i;
    for(i = 0; i < argc; i++)
        printf("%s\n", argv[i]);
}
```

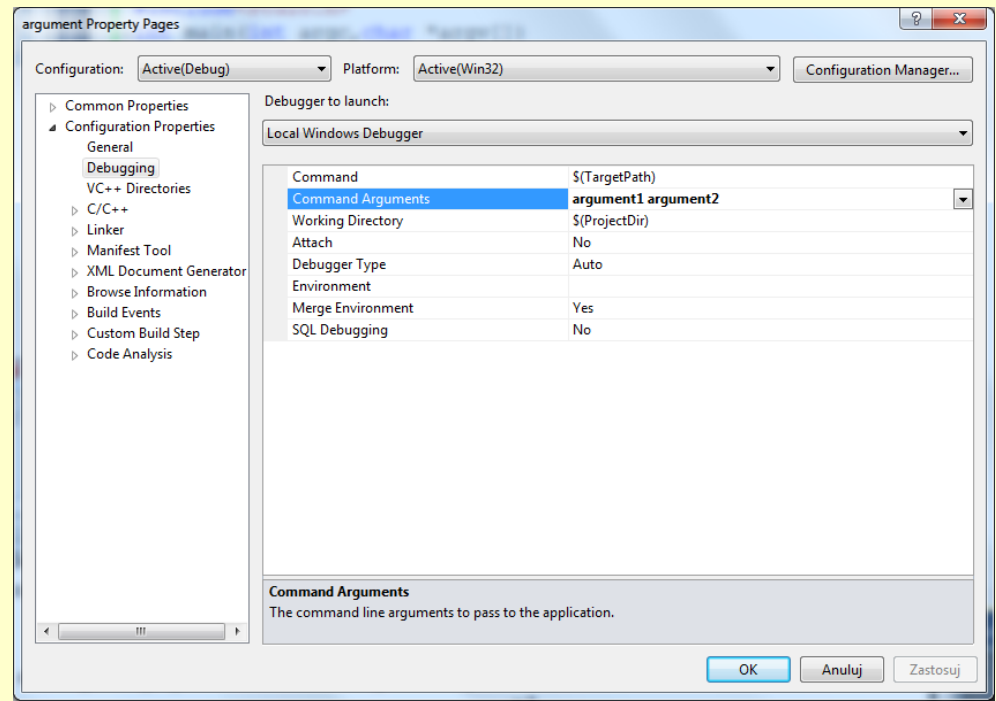
```
// Wywołanie biblioteki
// Funkcja main() wywołana z
// argumentami
/*    Wykorzystanie argumentu
pobranego z pierwszego miejsca
funkcji main() jako znaku „stopu”
dla pętli wyświetlającej znaki z
tablicy. */
```

Mając kod programu zapisany w powyższej formie Czytelnik zechce zauważyć, jakie różnice występują w przypadku różnego sposobu wywołania programu po kompilacji. Jeżeli uruchomimy program nie wpisując w linii poleceń argumentów wywołania funkcji to skompilowany program wyświetli tylko argument zerowy, którym jest nazwa programu ze ścieżką dostępu do wywołanego programu, co pokazuje kolejna ilustracja.

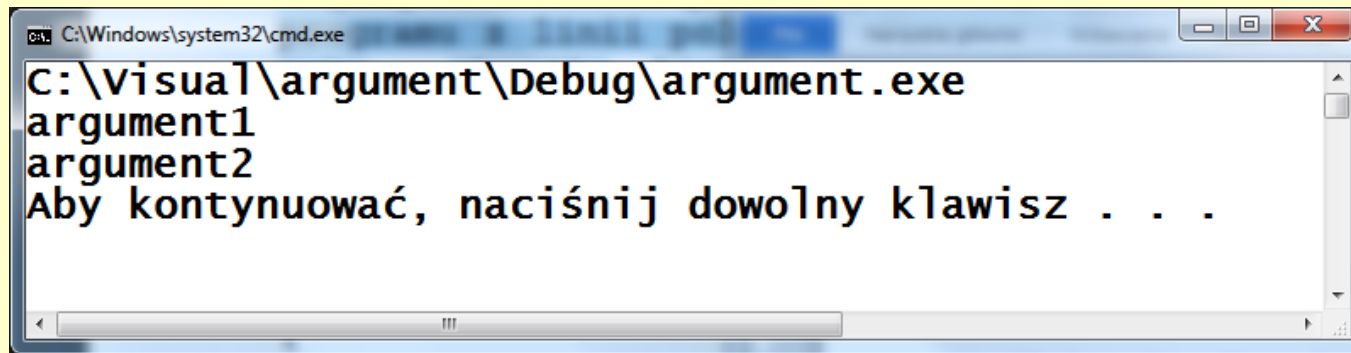
A screenshot of a Windows command prompt window. The title bar at the top reads "C:\Windows\system32\cmd.exe". The command prompt shows the command "C:\Visual\argument\Debug\argument.exe" entered. Below the command, the output is "Aby kontynuować, naciśnij dowolny klawisz . . .". The window has a standard Windows XP-style interface with a scroll bar on the right and a taskbar at the bottom.

```
C:\Windows\system32\cmd.exe  
C:\Visual\argument\Debug\argument.exe  
Aby kontynuować, naciśnij dowolny klawisz . . .
```

Zastanówmy się teraz nad zadaniem argumentu w sposób bezpośredni. Czytelnik zechce porównać zadanie argumentu w MS Visual Studio oraz po kompilacji w systemie Linux. Jeżeli chcemy wpisać dodatkowe argumenty w MS Visual Studio, które następnie zostaną pobrane przez wywołany program należy postąpić w następujący sposób. Wybieramy opcję z menu MS Visual Studio *Project* → *argumentProperties...* → *Debugging* → *CommandArguments*, a następnie wpisujemy argumenty wywołania funkcji *main()* przedzielone spacją, tak jak obrazuje to poniższa ilustracja. W naszym przypadku wpisaliśmy: *argument1 argument2*.

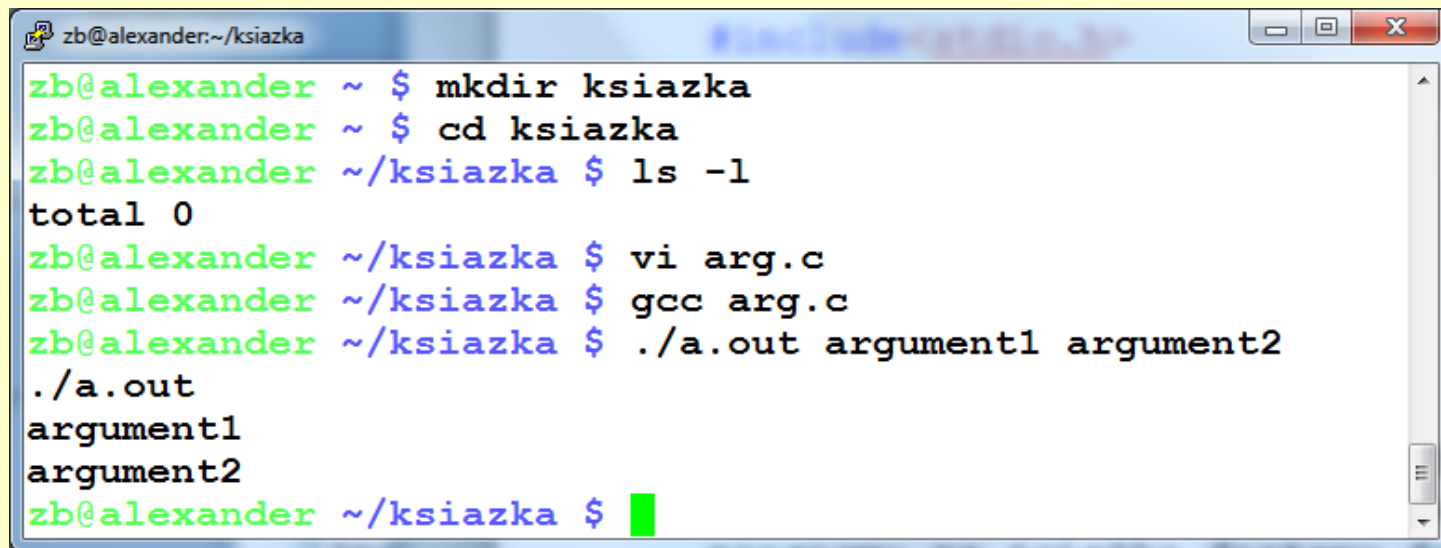


Po uruchomieniu programu skompilowanego z ustawionymi argumentami wyświetli się nazwa programu ze ścieżką dostępu oraz wpisane w debuggerze nasze argumenty:

A screenshot of a Windows command prompt window. The title bar shows 'C:\Windows\system32\cmd.exe'. The command prompt displays the following text:

```
C:\Visual\argument\Debug\argument.exe  
argument1  
argument2  
Aby kontynuować, naciśnij dowolny klawisz . . .
```

Prześledźmy teraz analogiczną sytuację pod systemem Linux. Postępowanie wywołania funkcji *main()* z argumentami bezpośrednimi jest możliwe. Jednak sytuacja kompilacji i wywołania wygląda trochę inaczej. Pod systemem operacyjnym Linux należy skompilować kod, a następnie uruchomić program z linii poleceń wpisując argumenty za nazwą wywołanego programu.

A terminal window titled 'zb@alexander:~/ksiazka' with standard Linux window controls. The terminal shows a sequence of commands and their outputs. The user creates a directory 'ksiazka', changes to it, lists its contents (showing it's empty), creates a file 'arg.c' with 'vi', compiles it with 'gcc', and runs the resulting 'a.out' with two arguments. The output shows the arguments being printed.

```
zb@alexander ~ $ mkdir ksiazka
zb@alexander ~ $ cd ksiazka
zb@alexander ~/ksiazka $ ls -l
total 0
zb@alexander ~/ksiazka $ vi arg.c
zb@alexander ~/ksiazka $ gcc arg.c
zb@alexander ~/ksiazka $ ./a.out argument1 argument2
./a.out
argument1
argument2
zb@alexander ~/ksiazka $
```

Jak widać na powyższej ilustracji, kompilując kod pod systemem operacyjnym Linux używamy odpowiednich opcji. Czytelnik zapewne zechce sprawdzić ich działanie samodzielnie. Po kompilacji kompilatorem *gcc* *argc.c* otrzymujemy domyślną nazwę programu *a.out* dlatego argumentem zerowym jest *./a.out*. Program powtarzający argumenty możemy napisać inaczej korzystając z operatora inkrementacji. Kod ten został zamieszczony poniżej.

<pre>#include<stdio.h> int main(int t,char **a) { while(--t > 0) printf("%s\n", *++a); }</pre>	<pre>// Deklaracja funkcji /* Funkcja main() wywołana z argumentem t */</pre>
--	--

Otrzymany wynik jest taki sam. Niemniej należy zauważyć, że język C/C++ jest językiem kontekstowym. Symbol `*` może oznaczać operację mnożenia dwóch liczb, pobranie wartości z zadanego adresu albo deklarację zmiennej wskaźnikowej. Stąd np. operacja `*++a` jest poprawną operacją pobrania kolejnego wiersza z tablicy argumentów. Inkrementacja wykonana jest przed pobraniem adresu wiersza, więc zostaną wypisane tylko argumenty podane przez użytkownika bez wypisania nazwy programu ze ścieżką dostępu, co pokazuje kolejna ilustracja.

A screenshot of a Windows command prompt window. The title bar shows the path C:\Windows\system32\cmd.exe. The window contains the following text: argument1, argument2, and a prompt line 'Aby kontynuować, naciśnij dowolny klawisz . . .' with three dots. The text is in a monospaced font.

Możemy również spróbować napisać ten sam program wypisujący argumenty w jednym wierszu.

```
#include<stdio.h>
int main(int t,char **a)
{
    while(--t > 0)
        printf((t>1)? "%s ":"%s\n",
                *++a);
}
```

```
// Deklaracja biblioteki

/*  Wypisanie poprzez funkcję
printf() pobranych argumentów w
jednej linii. */
```


Preprocesor języka

Bardzo często chcemy zdefiniować jakąś wartość czy nazwę dla wszystkich funkcji i procedur w naszym programie. Taka operacja w rozumieniu kompilatora powinna być wykonana przed kompilacją i wywoływaniem funkcji w programie. W języku C istnieje możliwość włączenia do programu swojego makro preprocesora, jest to operacja zdefiniowania stałych i funkcji jeszcze przed kompilacją programu. Operacja wykonana przed funkcją główną nosi nazwę preprocesora lub nagłówka programu. Prosty przykładem jest instrukcja preprocesora włączająca obsługę podstawowych funkcji czy definicji zmiennych.

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

Makro rozwinięcie

```
#include N 1024
```

Zaprezentowany schemat powoduje zastąpienie w całym kodzie programu napotkanego symbolu N przez wartość 1024.

Przykład 28

Napisać wykonujący zadanie wyszukiwania mniejszej z dwóch liczb za pomocą funkcji nagłówka.

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

```
#define min(W,Z) ((W)<(Z)?(W):(Z))
```

```
int main()
```

```
{
```

```
    double a,b;
```

```
    printf("Podaj a b:");
```

```
    scanf("%lf %lf",&a,&b);
```

```
    printf("Min(%g,%g)=%g\n", a, b, min(a,b));
```

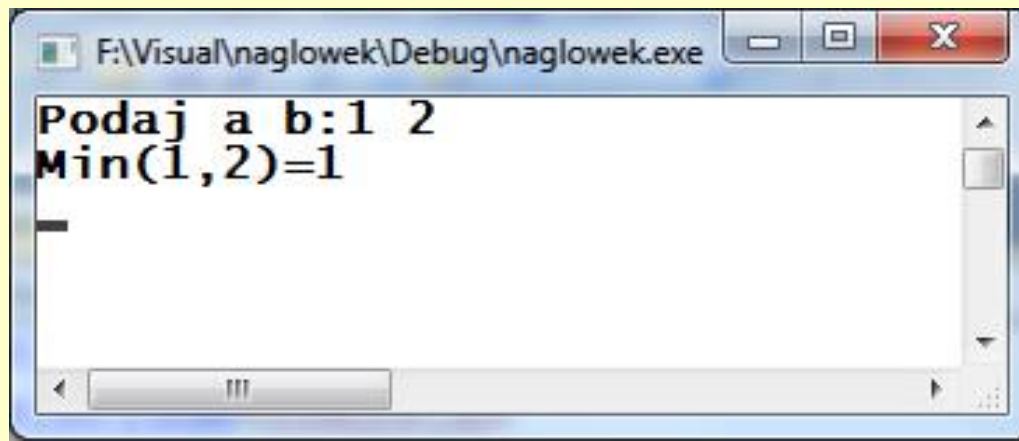
```
}
```

```
// Deklaracja biblioteki
```

```
/*      Definicja      warunku  
wyszukania minimum. Użycie  
nawiasów      w      makro  
rozwinięciu jest konieczne. */
```

```
/*      Pobranie      wartości,  
wyznaczanie i wydrukowanie  
minimum. */
```

Po kompilacji, uruchomieniu programu i wprowadzeniu liczb otrzymujemy następujący wynik.



```
F:\Visual\naglowek\Debug\naglowek.exe  
Podaj a b:1 2  
Min(1,2)=1
```

Uwaga: Użycie nawiasów w makro rozwinięciu jest konieczne. Definicja kwadratu liczby bez użycia nawiasów prowadzi do błędu w wykonaniu kodu, co pokazuje poniższy przykład.

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

```
#define kwadrat(x) x*x
```

```
int main()
```

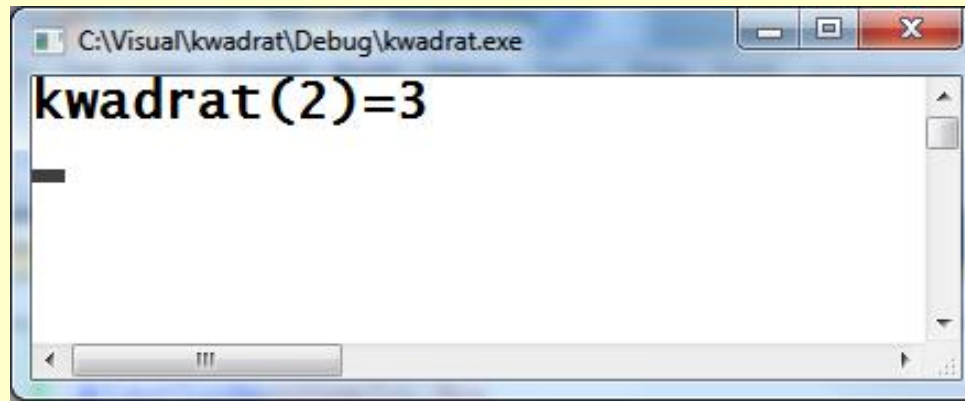
```
{
```

```
double x=1.0;
```

```
printf("kwadrat(%g)=%g\n", x+1.0,  
      kwadrat(x + 1.0));
```

```
}
```

/* Błędne zadanie
makroprocesora. */



The screenshot shows a Windows command prompt window with the title bar "C:\Visual\kwadrat\Debug\kwadrat.exe". The window contains the output of the program: "kwadrat(2)=3". The text is displayed in a monospaced font, and the window has standard Windows window controls (minimize, maximize, close) in the top right corner.

Funkcje matematyczne

Programując bardzo często korzystamy z określonych wartości liczbowych. Spełnianie różnych warunków czy wartości, jakie wyznaczamy w programie często zależy od wartości liczbowych wyznaczonych na podstawie funkcji matematycznych. W językach programowania dostępnych mamy wiele funkcji zdefiniowanych w bibliotekach językowych. W przypadku języka C funkcje matematyczne zdefiniowane są w bibliotece *math.h*. W języku C mamy dostępne następujące funkcje matematyczne operujące na argumentach double:

- $\sin(x)$ – sinus dla x , wartość podana w radianach,
- $\cos(x)$ – cosinus dla x , wartość podana w radianach,
- $\tan(x)$ – tangens dla x , wartość podana w radianach,

- $\text{atan}(x)$ – Funkcja oblicza arcus tangens dla zadanego x tj. liczbę, której tangens wynosi x ,
- $\text{pow}(x,y)$ – wartość x podniesiona do potęgi y , wartości argumentów funkcji są typu `double`,
- $\text{sqrt}(x)$ - pierwiastek kwadratowy z $x \geq 0$,
- $\text{fabs}(x)$ - wartość bezwzględna dla liczby zmiennoprzecinkowej,
- $\text{exp}(x)$ – wartość e podniesiona do potęgi x ,
- $\text{log}(x)$ - logarytm naturalny z x ,
- $\text{log10}(x)$ - logarytm dziesiętny z x .

Wymieniona biblioteka matematyczna zawiera definicje wielu funkcji. Przytoczone zostały tylko niektóre, najczęściej używane. Czytelnik zechce zapoznać się pełną treścią i możliwościami, jakie daje wykorzystanie funkcji oferowanych przez bibliotekę *math.h*.

Wymieniona biblioteka matematyczna zawiera definicje wielu funkcji. Przytoczone zostały tylko niektóre, najczęściej używane. Czytelnik zechce zapoznać się pełną treścią i możliwościami, jakie daje wykorzystanie funkcji oferowanych przez bibliotekę *math.h*.

Przykład 29

Napisać program obliczający: $\sqrt{2}$, $\sqrt[3]{2}$, ..., $\sqrt[9]{2}$.

```
#include<stdio.h>
#include<math.h>
int main()
{
    int i;
    double a=2,b,c,d;
    for(i=2;i<10;i++)
    {
        d=i;
        b=1/d;
        c=pow(a,b);
        printf("Pierwiastek %d-go stopnia z 2= ",i);
        printf("%17.15lf\n",c);
    }
}
```

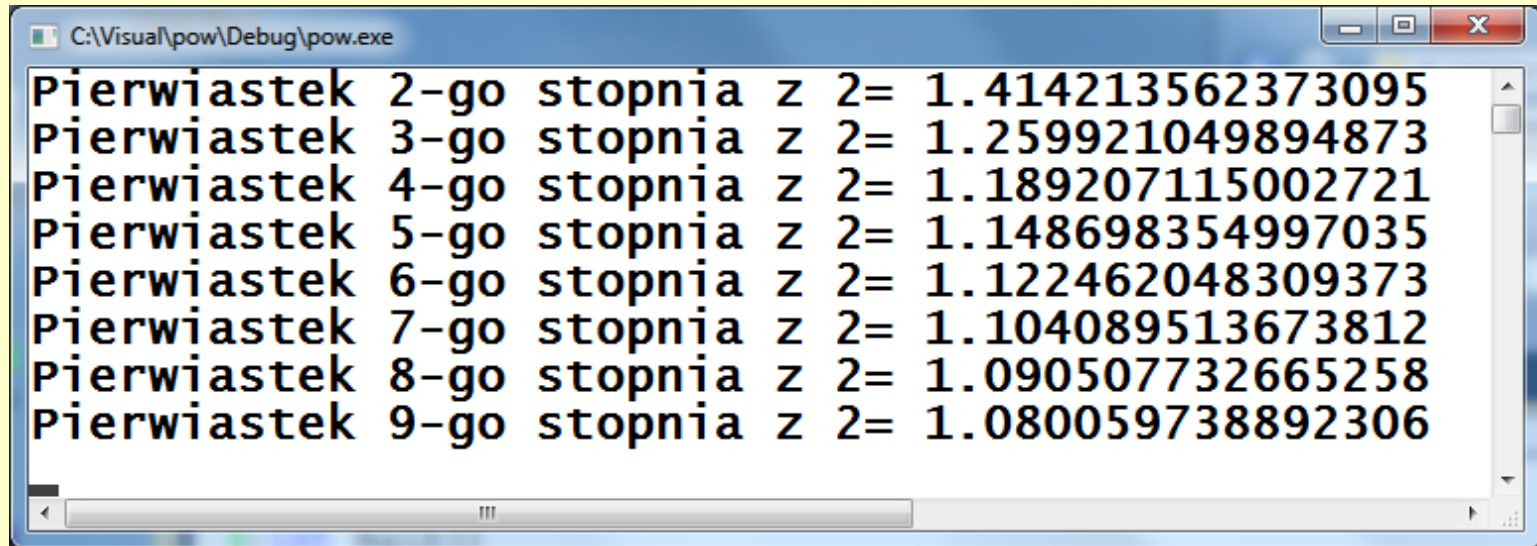
```
// Deklaracja bibliotek.
```

```
// Funkcja główna programu.
```

```
// Deklaracja zmiennych.
```

```
/* Procedura obliczania oraz  
wypisywania wartości. */
```


Wykonanie programu



```
C:\Visual\pow\Debug\pow.exe
Pierwiastek 2-go stopnia z 2= 1.414213562373095
Pierwiastek 3-go stopnia z 2= 1.259921049894873
Pierwiastek 4-go stopnia z 2= 1.189207115002721
Pierwiastek 5-go stopnia z 2= 1.148698354997035
Pierwiastek 6-go stopnia z 2= 1.122462048309373
Pierwiastek 7-go stopnia z 2= 1.104089513673812
Pierwiastek 8-go stopnia z 2= 1.090507732665258
Pierwiastek 9-go stopnia z 2= 1.080059738892306
```

Uwaga: przy użyciu kompilatora *gcc* musimy użyć opcji *-lm* w celu dołączenia biblioteki matematycznej.