

Petle

Czytelnik w tym rozdziale zapozna się z możliwością iteracyjnego wykonywania operacji. Operacja iterowania, czyli wykonywana w kolejnych powtórzeniach w trakcie całego algorytmu, jest jedną z podstawowych procedur w informatyce. Języki programowania dają wiele możliwości wykorzystania różnych procedur iterowania operacji w kodzie. Najczęściej iteracje wykorzystujemy w rozbudowanych algorytmach. Jeśli w trakcie algorytmu wartości, na których będzie operował komputer będą się zmieniać w zależności od postępujących warunków zewnętrznych to zmiany te będą reprezentowane w kodzie programu właśnie w formie kolejnych iteracji. Każda iteracja będzie rozumiana jako zmiana w czasie warunków zadania. W ten sposób możemy opisać zmieniające się kolejno warunki wpływające na poszukiwane przez nas rozwiązanie. Istotny wpływ na sposób iterowania operacji ma warunek kończący proces zmian. Jeśli będziemy w stanie jednoznacznie określić ile operacji musimy wykonać użyjemy odpowiednio deklaracji procedury na skończonej liczbie kroków. Jeśli nie znamy takiej liczby to operacja będzie powtarzana do momentu jakiegoś warunku określającego koniec zmian.

Instrukcja while i tablice znakowe

Na początku przedstawmy instrukcję *while()*, która jest w pewien sposób iteracyjnym wykonywaniem zadanej operacji, aż do momentu spełnienia określonego kryterium. Postać instrukcji *while()* jest następująca.

```
while ( wyrażenie )  
    instrukcja;
```

Instrukcja *while()* jest powtarzana, tak długo jak długo wyrażenie kryterialne zadane w nawiasie jest prawdziwe. Instrukcja ta może również być blokiem kolejnych lub powiązanych instrukcji, w takim przypadku umieszczamy go w klamrach { }.

Przykład 10

Napisać program wczytujący jedną linię tekstu ze standardowego wejścia klawiatury i wypisujący ilość wystąpienia w tekście małej litery 'a'.

Do wprowadzenia linii potrzebna będzie nam tablica znakowa. Ponieważ nie znamy ilości wprowadzonych znaków, założymy maksymalną ilość 1023. Dodatkowo musimy zadeklarować wielkość tablicy o jeden znak więcej, gdyż wprowadzony znak *Enter* jest zamieniany na znak '\0'. Tak, więc we wprowadzonej tablicy znaków określanej jako bufor, ostatnim znakiem jest znak *NULL*.

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

```
int main()
```

```
{
```

```
    char s[1024];
```

```
    int i,t=0;
```

```
    gets(s);
```

```
    printf("Wprowadzona linia to: %s\n",s);
```

```
    i=0;
```

```
    while(s[i]!='\0')
```

```
{
```

```
        if(s[i] == 'a')    t++;
```

```
        i++;
```

```
}
```

```
    printf
```

```
        ("Ilosc wprowadzonych 'a' = %d\n",t);
```

```
    _getch();
```

```
    return 0;
```

```
}
```

```
// Deklaracja biblioteki
```

```
/* Deklaracja tablicy na 1023
```

```
znaki + '\0, */
```

```
/* Niebezpieczeństwo przekroczenia  
limitu wczytywanych znaków. */
```

```
/* Deklaracja pętli while z kryterium  
wybrania klawisza 'Enter'. */
```

```
/* Warunek sprawdzający czy litera 'a'  
jest znakiem wprowadzonym. Jeśli tak  
jest, to następuje zsumowanie jej  
wystąpienia w zmiennej t oraz  
przejsie do kolejnej komórki tablicy  
poprzez zmienną i. */
```

Czytelnik zechce prześledzić przedstawione rozwiązanie problemu. Wprowadzane znaki są umieszczane w kolejnych elementach tablicy.

Ostatnim znakiem jest znak *NULL*. Za pomocą pętli *while()* przeglądamy kolejne elementy tablicy i sprawdzamy, czy została wpisana w nią litera 'a'. Zwiększamy wówczas licznik *i* o jeden. Na koniec programu drukujemy ilość wystąpienia poszukiwanej litery. Działanie takie wynika z faktu, że tablice w C/C++ rozpoczynają się od adresu 0 i kończą na wartości o jeden mniej od zadeklarowanej (w naszym przykładzie 1023).

Do napisania programu zliczającego ilość wystąpienia litery 'a' w tekście możemy również wykorzystać funkcję *getchar()* pobierającą następny znak ze strumienia wejściowego, aż do wystąpienia znaku końca zbioru. Uwaga: Czytelnik zechce zauważyć różnicę pomiędzy systemami operacyjnymi. CTR + d – dla systemu Linux i CTR + SHIFT + Z – dla systemu Windows. Czytelnik zechce przeanalizować rozwiązanie problemu zliczania wystąpienia liter w tekście. W przykładzie zaproponowane zostało rozwiązanie oparte o pobieranie poszczególnego znaku funkcją *getchar()*, a następnie jego kwalifikacja.

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

```
int main()
```

```
{
```

```
    char c;
```

```
    int t = 0;
```

```
    c = getchar();
```

```
    while(c != EOF)
```

```
    {
```

```
        if(c == 'a')    t++;
```

```
        c = getchar();
```

```
    }
```

```
    printf("Ilosc wprowadzonych 'a' =%d\n",t);
```

```
    _getch();
```

```
    return 0;
```

```
}
```

```
// Deklaracja biblioteki
```

```
// Deklaracja zmiennej typu char
```

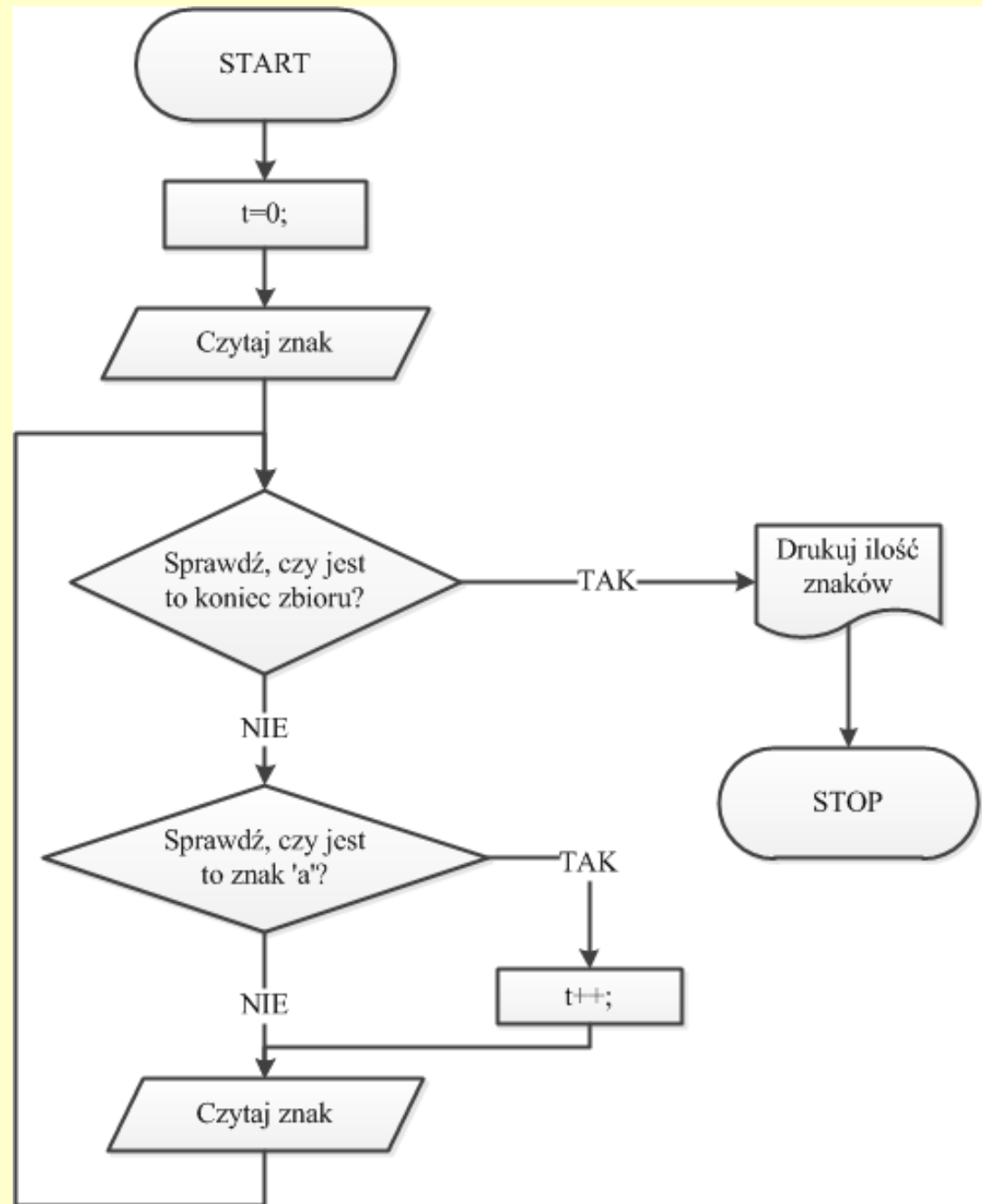
```
/* Deklaracja pobierania do  
zmiennej c znaku */
```

```
/* Jeśli pobrane c nie jest  
znakiem końca pliku */
```

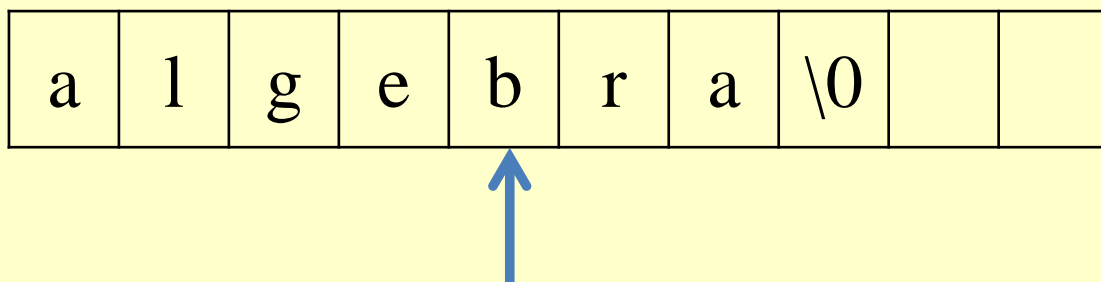
```
/* Warunek sprawdzający czy  
litera 'a' jest znakiem  
wprowadzonym. Jeśli tak jest, to  
następuje zsumowanie jej  
wystąpienia w zmiennej t oraz  
pobranie kolejnego znaku. */
```

```
// Wydruk obliczeń
```

W programie wykorzystano funkcję wczytującą znak ze strumienia wejściowego. Trzeba jednak pamiętać, że pobieranie znaków odbywa się przez bufor. Powoduje to, że faktycznie znaki przekazywane są do programu po przyciśnięciu klawisza *Enter*. Faktycznie przekazywany jest cały bufor, a nie pojedynczy znak. Daje to jedynie poczucie użytkownikowi, że operuje on na pojedynczych znakach wczytywanych ze strumienia.



Opisany problem możemy spróbować rozwinąć dla szukania całych słów i zliczania ich wystąpień. Jednak najpierw Czytelnik powinien poznać nowe pojęcie. Bardzo ważnym pojęciem w informatyce są tzw. białe znaki.



Białe znaki umieszczone w tekście to znaki niewidoczne na ekranie. Najczęściej używane to znak spacji ' ', nowej linii '\n' i znak tabulacji '\t'. Ponieważ znaki apostrofów, ukośnika i cudzysłowu są znakami specjalnymi, dlatego w celu wpisania ich w programie stosujemy zapis podany w tabeli.

Tablica białych znaków	
"\"	Ukośnik
"'	Apostrof
"'"	Cudzysłów
"\0"	Znak NULL zapisany na jednym bajcie

Przykład 11

Napisać program liczący ilość słów występujących w tekście. Aby wykonać to zdanie musimy podać definicję słowa. Dla naszych celów możemy przyjąć, że słowa oddzielają białe znaki, a wszystkie pozostałe znaki tworzą słowo. Jest to oczywiste uproszczenie, ale pozwala wytłumaczyć, w jaki sposób algorytm zliczający rozpoznaje pojawiające się na wejściu wyrazy. Przykładowe zdanie, analizowane przez napisany program, „C/C++ jest językiem programowania.” możemy zilustrować następująco.



Łatwo zauważyć, że ilość wyrazów jest równa ilości strzałek. Strzałka powstaje w miejscu, kiedy poprzedni znak był znakiem białym, a wczytany znak jest znakiem tekstu. Kod programu został przedstawiony poniżej.

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

```
int main()
```

```
{
```

```
    char c;
```

```
    int t = 0, t1 = 1, t2;
```

```
    while((c = getchar()) != EOF)
```

```
    {
```

```
        t2 = (c==' ' || c == '\n' || c == '\t');
```

```
        if(t1 && !t2) t++;
```

```
        t1=t2;
```

```
    }
```

```
    printf
```

```
        ("W tekście są %d słowa\n",t);
```

```
    _getch();
```

```
    return 0;
```

```
}
```

```
// Deklaracja biblioteki
```

```
    /* Deklaracja zmiennej typu  
char */
```

```
    /* Deklaracja pobierania do  
zmiennej c znaku */
```

```
    /* Jeśli pobrane c nie jest  
znakiem końca pliku */
```

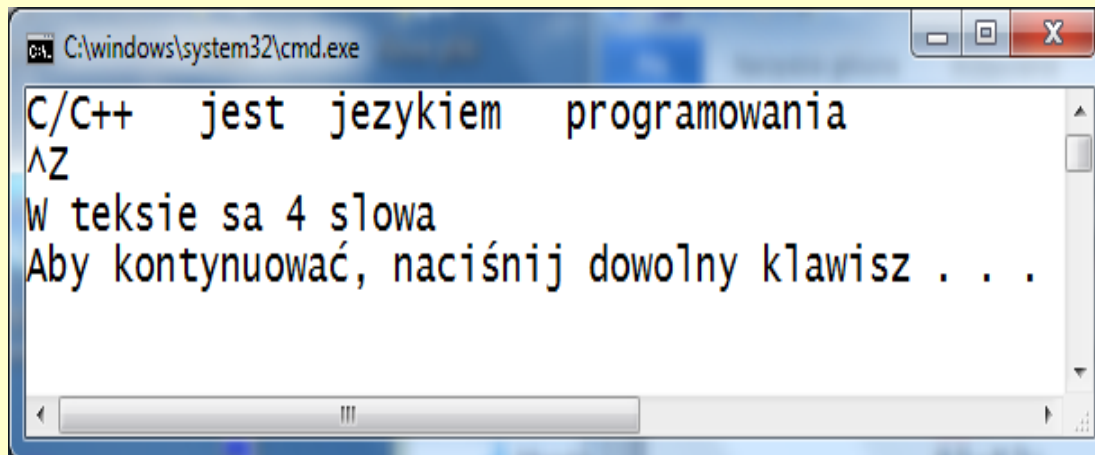
```
    /* Warunek sprawdzający czy  
litera 'a' jest znakiem
```

```
wprowadzonym. Jeśli tak jest,  
to następuje zsumowanie jej
```

```
wystąpienia w zmiennej t oraz
```

```
pobranie kolejnego znaku */
```

t	Ilość słów naliczonych w tekście
t1	Zmienna logiczna przyjmująca wartość 1 w przypadku, gdy poprzedni z wczytanych znaków był znakiem białym. Na początku zakładamy, że znak przed wczytanym tekstem był znakiem białym i ustawiamy wartość zmiennej domyślnie na 1.
t2	Zmienna logiczna przyjmująca wartość jeden w przypadku, gdy aktualnie wczytany znak jest znakiem białym.



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: 'C/C++ jest jezykiem programowania', '^Z', 'W teksie sa 4 slowa', and 'Aby kontynuować, naciśnij dowolny klawisz . . .'. The text is displayed in a monospaced font. The window has standard Windows window controls (minimize, maximize, close) in the top right corner and a scrollbar on the right side.

```

C:\windows\system32\cmd.exe
C/C++ jest jezykiem programowania
^Z
W teksie sa 4 slowa
Aby kontynuować, naciśnij dowolny klawisz . . .

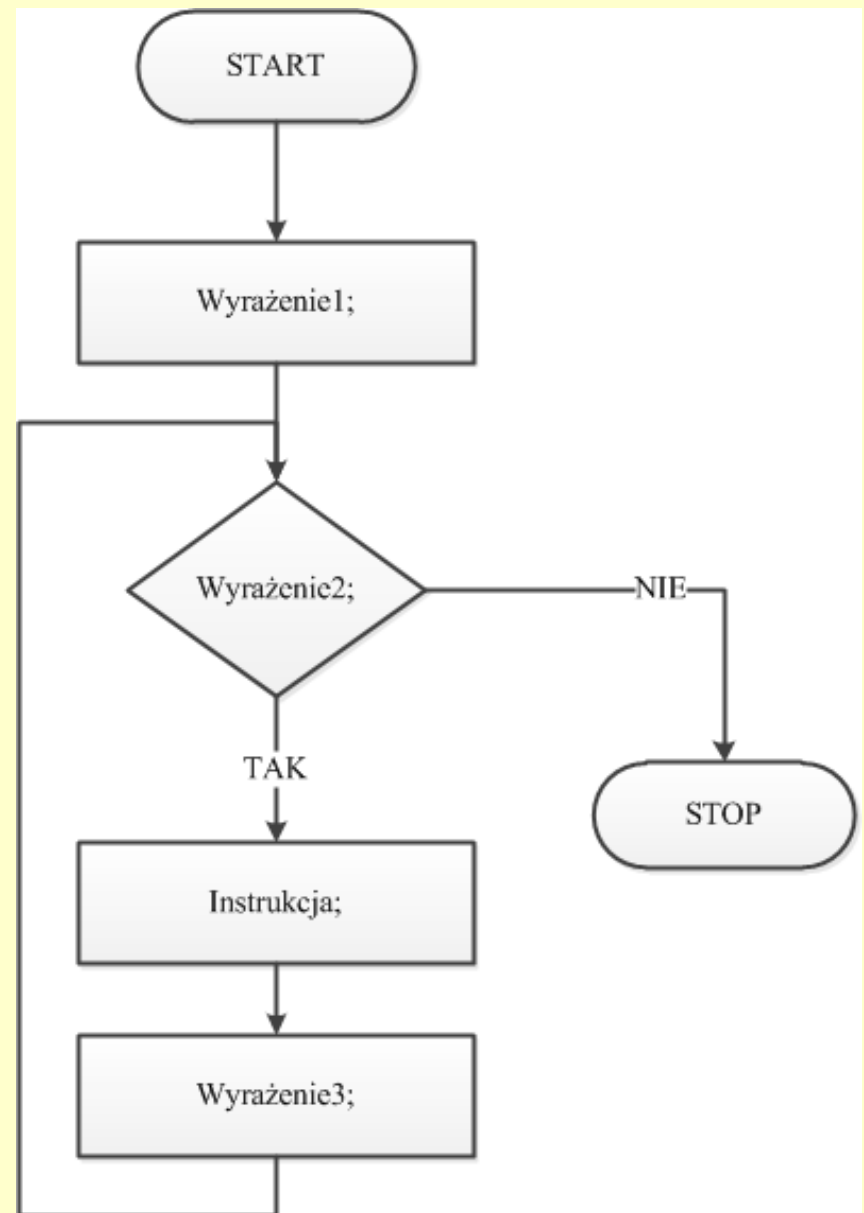
```

Pętla for

W życiu codziennym bardzo często napotykamy operacje, które należy powtórzyć kilka razy. Najczęściej jesteśmy zmuszeni przeszukać kilka razy nasze dokumenty, schowek czy zwyczajnie wykonać kilka razy pod rząd tę samą operację. Język programowania C/C++ daje możliwość wykonywania operacji powtarzalnych. Programista może zrealizować operacje powtarzane stosując odpowiednią pętlę. Pętla w rozumieniu języka jest to instrukcja, którą komputer będzie wykonywał określoną ilość razy z rzędu. Najpopularniejszą deklaracją powtarzania operacji jest pętla *for()* mająca następujący schemat budowy.

```
for ( Wyrażenie1 ; Wyrażenie2 ; Wyrażenie 3 )  
    Instrukcja;
```

Pętla *for()* jest tłumaczona przez kompilator języka C/C++ do pętli *while()*, w sposób przedstawiony na poniższym schemacie blokowym. Znajomość pętli *for()* pozwala na efektywne pisanie algorytmów.



Spróbujmy teraz wykonać program realizujący zadanie drukowania liczb naturalnych nie większych niż n . Czytelnik zechce się zapoznać z kodem procedury realizującej omawiane zagadnienie.

```
int i, n;  
printf("Podaj n:");  
scanf("%d", &n);  
for(i = 1; i <= n; i++)  
printf("%d\n", i);
```

Zachowanie strukturalności programu tak, aby każdy algorytm miał określone wejście (to, co musimy wprowadzić do zamkniętej części programu) i to, co otrzymujemy w wyniku wykonania algorytmu jest bardzo istotne. Ingerując w określoną strukturę kodu programu bez jej znajomości możemy w łatwy sposób doprowadzić do błędu.

Przykład 12

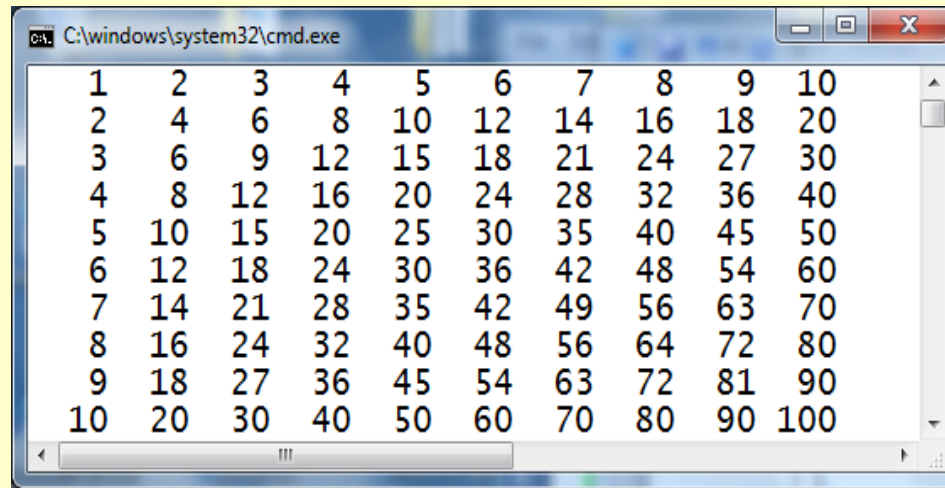
Napisać program drukujący tabliczkę mnożenia

```
#include<stdio.h>
int main()
{
    int i, j;
    for( i = 1; i <= 10; i++)
    {
        for( j = 1; j <= 10; j++)
            printf("%4d", i*j);
        printf("\n");
    }

    _getch();
    return 0;
}
```

```
//Dołączenie biblioteki.
// Funkcja główna programu.
/*    Deklaracja    zmiennych
Iteracyjnych */
// Iteracja wierszy.

// Iteracja kolumn.
/*    Wydruk    mnożenia    na
czterech znakach */
// Przedzielenie wierszy.
```

A screenshot of a Windows command prompt window titled "C:\windows\system32\cmd.exe". The window displays a 10x10 grid of numbers, starting from 1 in the top-left corner and ending at 100 in the bottom-right corner. The numbers are arranged in rows and columns, with each row containing 10 numbers and each column containing 10 numbers. The window has a standard Windows interface with a title bar, minimize, maximize, and close buttons, and a scrollbar on the right side.

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

W języku C/C++ można zrealizować wiele najważniejszych problemów matematycznych. Jednym z ważniejszych algorytmów, możliwym do zrealizowania w omawianym języku jest algorytm wyszukiwania w ciągu liczbowym największej i najmniejszej wartości oraz obliczania średniej arytmetycznej np. średniej oceny klasowej. Trzeba jednak pamiętać o tym, że każda liczba rozumiana przez komputer ma zawsze znak, może być dodatnia albo ujemna.

Przykład 13

Znaleźć największą liczbę w ciągu n elementowym, złożonym z liczb całkowitych.

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

```
int main()
```

```
{
```

```
    int n, i, max, a[1024];
```

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

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

```
    for(i=0;i<n;i++)
```

```
    {
```

```
        printf("Podaj liczbę %d:",i);
```

```
        scanf("%d",&a[i]);
```

```
    }
```

```
// Deklaracja bibliotek
```

```
// Deklaracja zmiennych i tablicy o stałym wymiarze.
```

```
// Wczytanie wymiaru ciągu
```

```
/* Wczytanie n elementowego ciągu liczb całkowitych wykonane pętlą wypisującą kolejne elementy. */
```

```
max=a[0];
```

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

```
    if(a[i] > max) max=a[i];
```

```
printf("max=%d\n",max);
```

```
_getch();
```

```
return 0;
```

```
}
```

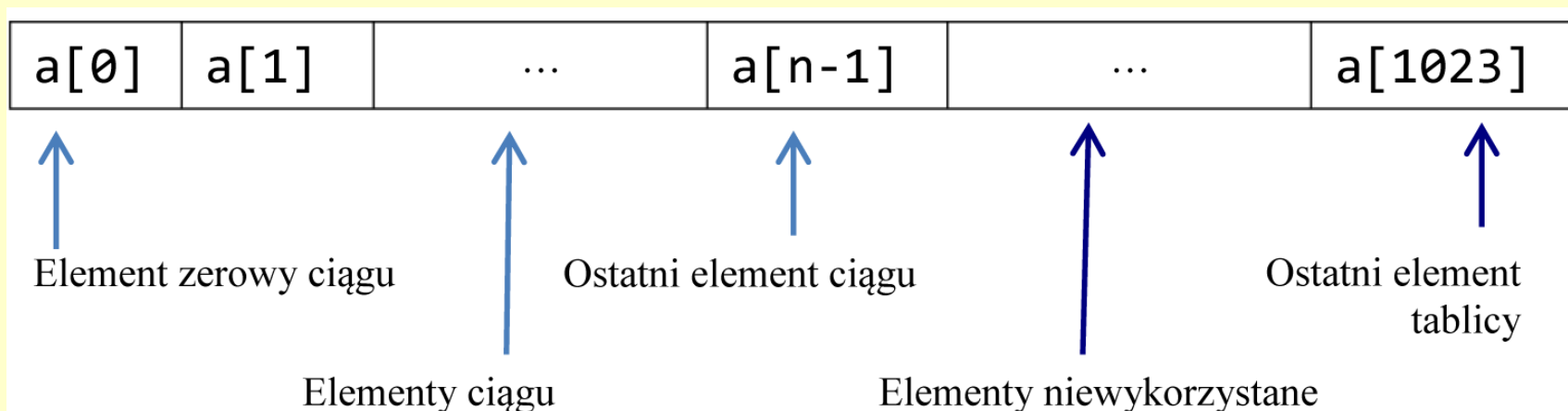
```
/* Przyjęcie za element maksymalny  
elementu zerowego.*/
```

```
/*      Porównanie      elementu  
maksymalnego      z      pozostałymi  
elementami ciągu. */
```

```
// Wydrukowanie      znalezionego  
elementu największego.
```

W programie przyjęto tablicę o największym możliwym do rozwiązania wymiarze. Poszukiwanie największej liczby może być wykonane dla ciągu nie większego niż 1024 elementy. Wykorzystujemy n początkowych elementów tablicy do zapisania ciągu liczbowego. Pozostałe elementy mają wartość przypadkową.

Uwaga: W języku C/C++ nie są zerowane zmienne ani tablice w momencie załadowania programu do pamięci komputera o ile programista sam nie ustalił wartości początkowych. Czytelnik programując musi pamiętać o tym fakcie. Aby uzyskać tablicę „pustą” należy ją wyzerować.



Jeżeli podany wymiar ciągu będzie większy niż wymiar zadeklarowanej tablicy wystąpi błąd podczas wykonania programu lub program zostanie wykonany i otrzymamy wynik losowy.

W starszych wersjach języka C nie istniała możliwość deklarowania tablic dynamicznych tzn. określania wymiaru tablicy podczas działania programu i zachodziła konieczność deklaracji tablic o maksymalnym wymiarze. Obecnie w dalszym ciągu korzystamy z tej formy deklaracji, jeżeli nie potrafimy określić ilości wczytywanych elementów np. wielkości bufora, jest ona określana z góry.

Przedstawiony algorytm można uzupełnić o procedurę wyszukiwania najmniejszego elementu w podanym ciągu oraz obliczanie średniej arytmetycznej z przeczytanych liczb. Czytelnik zechce przypomnieć wzór opisujący średnią arytmetyczną n liczb iterowaną w sensie rozumienia kompilatora.

$$s = \frac{a_0 + a_1 + \dots + a_{n-1}}{n}$$

Uzupełniony kod procedury został przedstawiony poniżej.

```
s=max=min=a[0];

for(i = 1;i < n;i++)
{
    if(a[i]>max)max=a[i];
    if(a[i]<min)min=a[i];

    s+=a[i];
}

s/=n;

printf("max=%d min=%d s=%g\n",
        max, min, s);
```

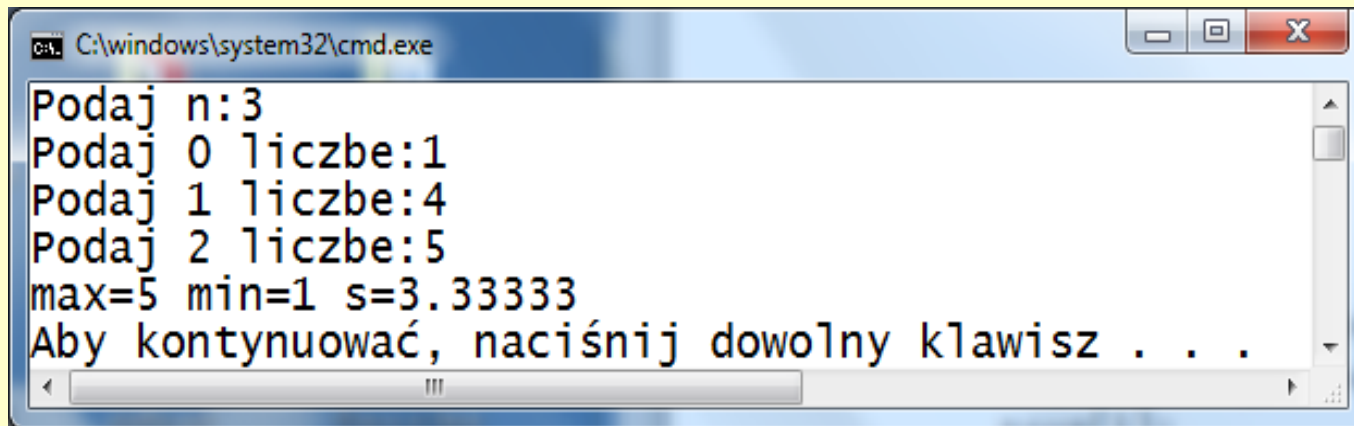
```
/* Wprowadzenie minimum i maksimum
jako elementu zerowego macierzy. */
/* Pętla wykonująca warunki sprawdzenia
czy kolejne elementy są większe od
poprzedniego maksimum oraz czy nie jest
mniejsze od minimum. */

/* Wykonanie dodawania do elementów
ciągu dla wyliczenia średniej. */
// Wyznaczenie sumy
/ Obliczenie średniej

// Wydruk wyników
```

W kodzie programu wykorzystano automatyczną zamianę liczby całkowitej na zmienną typu *double*. Stąd wartość sumy ciągu jest naliczana jako liczba rzeczywista.

Ekran konsoli z wyszukaniem największej liczby w ciągu liczb



```
C:\windows\system32\cmd.exe
Podaj n:3
Podaj 0 liczbe:1
Podaj 1 liczbe:4
Podaj 2 liczbe:5
max=5 min=1 s=3.33333
Aby kontynuować, naciśnij dowolny klawisz . . .
```

The image shows a Windows command prompt window titled "C:\windows\system32\cmd.exe". The window contains the following text: "Podaj n:3", "Podaj 0 liczbe:1", "Podaj 1 liczbe:4", "Podaj 2 liczbe:5", "max=5 min=1 s=3.33333", and "Aby kontynuować, naciśnij dowolny klawisz . . .". The text is displayed in a monospaced font. The window has a standard Windows XP-style title bar with minimize, maximize, and close buttons.

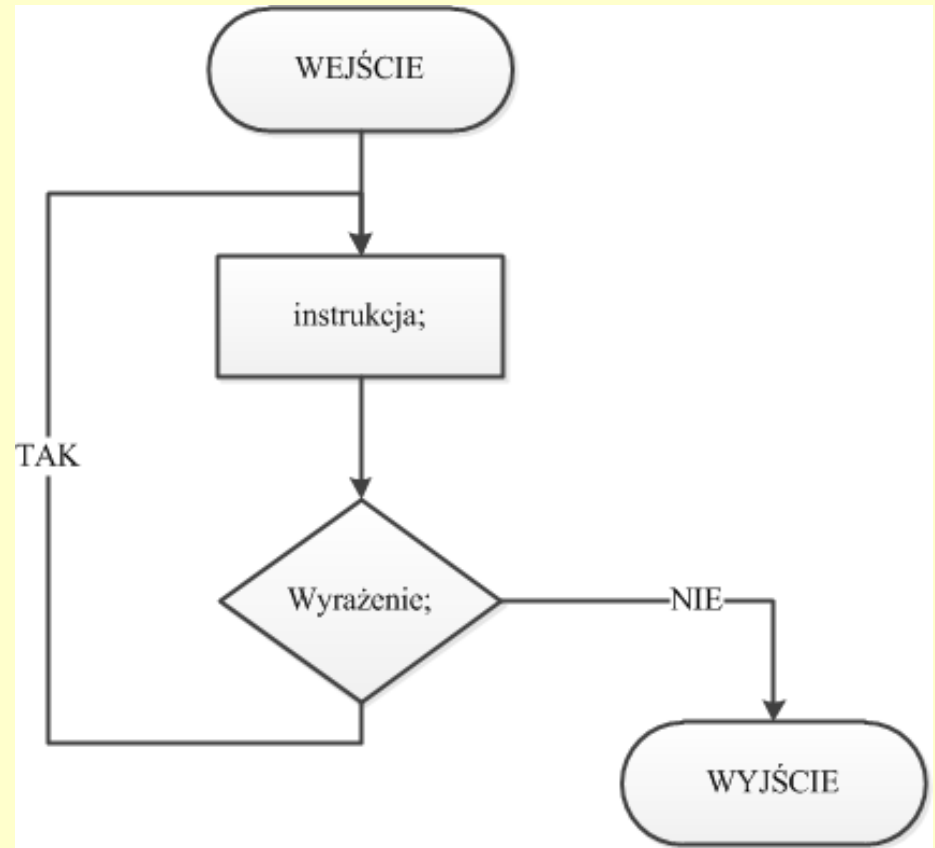
Pętla do...while

Zapisanie operacji w postaci pętli *do...while* będzie pomocne w przypadku, kiedy będziemy chcieli sprawdzać zachodzenie warunku kryterium dopiero po wykonaniu pewnej grupy operacji. Ta instrukcja wykonuje kolejne iteracje w algorytmie, aż do momentu spełnienia określonego warunku. Jest to instrukcja dająca możliwość przeprowadzenia wielu operacji nawet, jeśli nie wiemy, jaka będzie ich ostateczna liczba, a jesteśmy jedynie w stanie określić, kiedy komputer powinien przestać realizować określoną procedurę.

Nie jest to jednak instrukcja stworzona bezpośrednio na potrzeby programistów korzystających z języka C/C++. Składnia tej instrukcji została zaczerpnięta z języka Fortran, w którym warunek zatrzymania pętli był sprawdzany dopiero na końcu iteracji. Formalnie instrukcja ma następującą składnię

do
Instrukcja;
while(Wyrażenie) ;

Oznacza to, że
instrukcja zostanie
wykona
przynajmniej raz,
co możemy
przedstawić na
przedstawionym
schemacie
blokowym.



Przykład 14

Napisać program wypisujący kolejne liczby naturalne z wykorzystaniem pętli do...while

```
#include<stdio.h>
int main()
{
    int i, j=0;
    scanf("%d", &i);
    if(i > 0)
    {
        do
            printf("%d\n", ++j);
        while(--i > 0);
    }
    return 0;
    _getch();
}
```

// Deklaracja biblioteki

// Wczytanie ilości liczb do wypisania

/* Sprawdzanie poprawności wprowadzonej liczby */

/* Przedrostkowa postać inkrementacji powoduje zmniejszenie lub zwiększenie wartości zmiennej przed wzięciem jej do obliczeń */

Przykład 15

Uczniowie na koniec semestru otrzymują stopnie, są to liczby całkowite z przedziału {1,2,3,4,5,6}. Należy napisać program, który wczytuje oceny uczniów, następnie oblicza najwyższą i najniższą ocenę wystawioną przez nauczyciela oraz przeciętną ocenę klasy z danego przedmiotu.

<pre>#include<stdio.h> int main() { int n=0, i, a[1024],d,max,min; double s; do { printf("a[%d]=",n); d = scanf("%d",&a[n]); n++; } while(d != EOF);</pre>	<pre>// Deklaracja biblioteki /* Deklaracja zmiennych oraz tablicy na 1024 elementy */ /* Próba wczytania stopnia ucznia do zadeklarowanej wcześniej 1024 elementowej tablicy. W przypadku wczytania znaku końca zbioru ang. End Of File nastąpi zakończenie pętli. */</pre>
---	--

```
n--;  
printf("?\\n");  
printf("Ilosc elemetow %d\\n",n);
```

```
s=max=min=a[0];  
for(i = 1; i < n; i++)  
{  
    if( a[i] > max ) max = a[i];  
    if( a[i] < min ) min = a[i];  
    s+=a[i];  
}  
printf("max=%d ", max);  
printf("min=%d ", min);  
printf("avg=%6.2f\\n", s/n);
```

```
_getch();  
return 0;
```

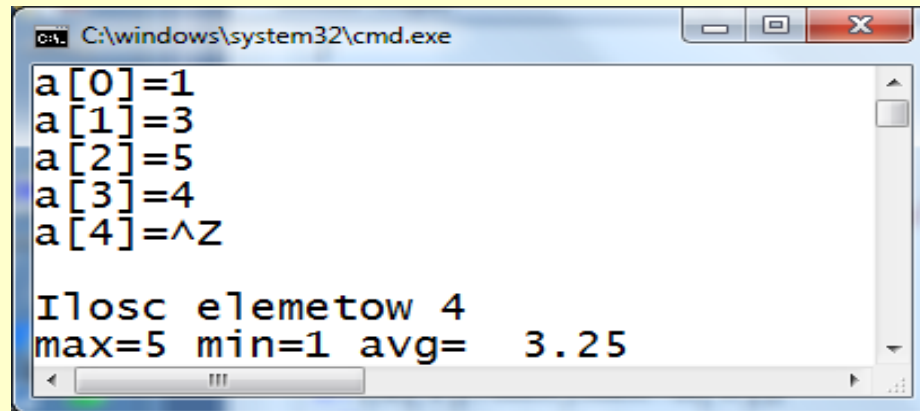
```
}
```

/* Zmniejszenie o jeden ilości
przeczytanych elementów i wydruk
wymiaru zadania (ilości przeczytanych
stopni */

// Licznik wzoru

/* Omówiony algorytm obliczania
największej i najmniejszej wartości we
wczytanym zbiorze średniej arytmetycznej
według wzoru */

Program czyta liczby całkowite, aż do wystąpienia znaku końca zbioru. Uwaga: Czytelnik zechce zauważyć, że znaki końca zbioru mają inną postać dla omawianych systemów operacyjnych. Jest to kombinacja klawiszy CTR + d dla systemu Linux oraz CTR + SHIFT + Z dla systemu Windows.



```
C:\windows\system32\cmd.exe
a[0]=1
a[1]=3
a[2]=5
a[3]=4
a[4]=^Z

Ilosc elementow 4
max=5 min=1 avg= 3.25
```

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 text representing an array 'a' with five elements. The first four elements are 1, 3, 5, and 4, each on a new line. The fifth element is '^Z', which represents the end-of-file (EOF) character in Windows. Below the array, the program outputs statistics: 'Ilosc elementow 4' (Number of elements 4), 'max=5 min=1 avg= 3.25' (maximum=5, minimum=1, average=3.25). The window has a scrollbar on the right side.

W większości programów nie poprawne jest przerwanie czytania przez wczytanie określonej litery czy liczby np. zera. Mimo, iż w omawianym zadaniu nie może zostać wystawiona ocena zero, takie rozumowanie byłoby błędne. Rozwiązanie zostało oparte o spostrzeżenie, że funkcja `scanf()` zwraca ilość przeczytanych liczb. Natomiast, jeśli napotka znak końca zbioru *EOF* odpowiedź będzie faktycznie równa liczbie -1. Natomiast należy podkreślić, że jest to wartość zwracana przez funkcję, a nie wartość znajdująca się faktycznie w zbiorze liczb.

Instrukcje switch i break

Postarajmy się teraz rozwiązać problem jednoczesnego nałożenia kilku warunków na rozwiązanie problemu. Takie zdarzenie może mieć miejsce, jeśli będziemy rozpatrywali przynależność badanego punktu do kilku przedziałów liczbowych. Oczywiście Czytelnik może zaproponować rozwiązanie takiego problemu na podstawie posiadanej już wiedzy. Stosując kilka zagnieżdżonych odpowiednio instrukcji warunkowych *if()* można sprawdzić kilka warunków jednocześnie. Należy jednak pamiętać, że kod programu powinien cechować się prostotą i przejrzystością opisywanych operacji. Zatem zagnieżdżanie nie jest dobrym rozwiązaniem i często może prowadzić do przeciążania systemu czy niepoprawnej interpretacji. Zatem w takim przypadku najlepszym rozwiązaniem byłaby instrukcja łącząca w sobie kilka warunków jednocześnie. W języku C/C++ istnieje możliwość zastosowania takiego swoistego multi warunku. Służy temu odpowiednio zbudowana instrukcja *case*. Jest ona rozszerzeniem instrukcji *switch*. Postać instrukcji *switch* jest następująca.

```
switch( Wyrażenie )  
    Blok_instrukcji
```

Wyrażenie jest wykonywane na liczbach całkowitych i stanowi wzorzec do poszukiwania instrukcji poprzedzonej warunkiem *case*.

```
case stała_n: Instrukcja_n;  
    default: określa domyślny wzorzec poszukiwania
```

Do przerywania poszukiwania służy instrukcja *break*. Zatem cała procedura wykonująca omawiany multi warunek będzie miała postać.


```
switch( Wyrażenie )  
{  
    case stała_1: Instrukcja_1;  
    break;  
    case stała_2: Instrukcja_2;  
    break;  
  
    case stała_n: Instrukcja_n;  
    break;  
    default: instrukcja_domyslna;  
    break;  
}
```

Spróbujemy teraz poznać działanie omawianej instrukcji na praktycznym przykładzie.

Przykład 16

Napisać program obliczający ilość wystąpienia w tekście małych liter a, b, c i pozostałych znaków.

```
#include<stdio.h>
int main()
{
    char c;
    int a[3], t;
    t = a[0] = a[1] = a[2] = 0;

    while((c = getchar()) != EOF)
    switch(c)
    {
        case 'a': a[0]++;
        break;
        case 'b': a[1]++;
        break;
        case 'c': a[2]++;
        break;
```

// Deklaracja biblioteki

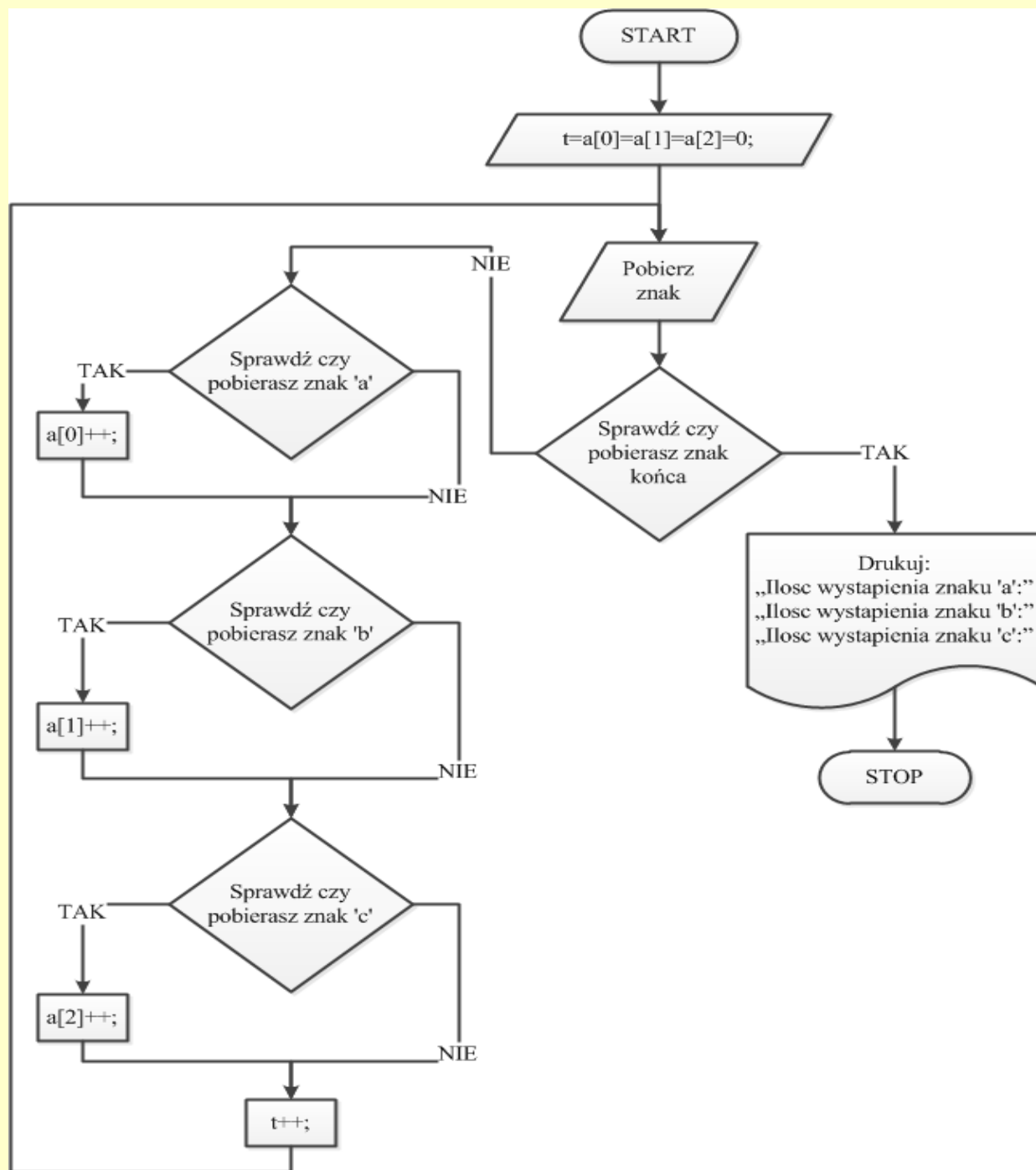
/* Deklaracja zmiennych i tablicy zawierającej ilości rozpoznanych znaków. */

/* Pobieranie znaku z klawiatury, aż do momentu wczytania znaku końca zbioru. */

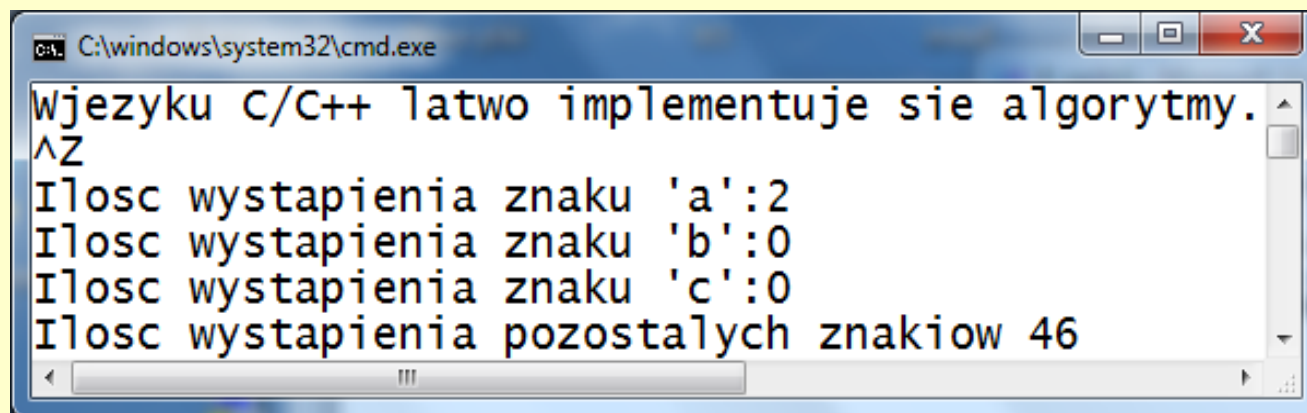
/* Instrukcja *switch...case*, której zadaniem jest sprawdzenie wystąpienia poszczególnej szukanej litery lub innego znaku, a następnie zliczenie ilości wystąpienia sprawdzonego w poszczególnym warunku znaku */

```
        default: t++;  
        break;  
    }  
    printf("Ilosc wystapienia znaku 'a':%d\n",  
          a[0]);  
    printf("Ilosc wystapienia znaku 'b':%d\n",  
          a[1]);  
    printf("Ilosc wystapienia znaku 'c':%d\n",  
          a[2]);  
    printf("Ilosc wystapienia pozostalych  
          znakow :%d\n", t);  
  
    _getch();  
    return 0;  
}
```

```
/* Instrukcje wypisujące  
naliczone ilości wystąpień  
zweryfikowanych znaków  
*/
```



Czytelnik zapewne zauważył, że po każdym warunku w instrukcji *case* następuje przerwanie wykonywanych operacji poprzez komendę *break*. Wystąpienie instrukcji *break* jest konieczne, jeżeli chcemy uniknąć podwójnego zliczania znaków. Po znalezieniu wzorca oraz wykonaniu instrukcji związanej ze zweryfikowanym wzorcem instrukcja *switch* nie przerywa poszukiwania. Oznacza to, że nastąpi ponowne wykonanie instrukcji domyślnej *default* zwiększające ilość wystąpień pozostałych znaków. Wykonanie kodu opisanego w przykładzie 16 na konsoli systemowej prezentuje ilustracja.



```
C:\windows\system32\cmd.exe
Wjezyku C/C++ latwo implementuje sie algorytmy.
^Z
Ilosc wystapienia znaku 'a':2
Ilosc wystapienia znaku 'b':0
Ilosc wystapienia znaku 'c':0
Ilosc wystapienia pozostałych znaków 46
```

Czytelnik zapewne zauważył, że działaniu programu zostało poddane zdanie „*W języku C/C++ łatwo implementuje się algorytmy.*” W zdaniu tym nie zastały użyte polskie znaki językowe ze względów opisanych we wcześniejszych rozdziałach. Powyższy kod programu można bardzo łatwo napisać bez wykorzystania instrukcji *switch*. Możliwość taka istnieje jednak tylko dla tekstów opartych o litery należące do alfabetu angielskiego.

Do napisania uproszczonego kodu można wykorzystać fakt, że kody kolejnych liter alfabetu angielskiego są reprezentowane w tablicy znaków w określonej kolejności. Zostało to omówione na początku książki. Kod litery b jest o jeden większy od kodu litery a. Jest to w pewien sposób kod naturalny dla języka angielskiego. Niestety nie ma on jednak zastosowania do języka polskiego i innych języków narodowych.

```
#include<stdio.h>
int main()
{
    char c;
    int a[3], t = 0, i;

    for(i=0;i<3;i++)
        a[i]=0;

    while((c = getchar() ) != EOF)
        if('a', <= c && c <= 'c')    a[c-'a']++;
        else t++;

    for(i=0;i<3;i++)
        printf("Ilosc wystapienia znaku
                '%c':%d\n", 'a'+i, a[i]);

    printf("Ilosc wystapienia
            pozostalych znakow %d\n", t);
    _getch();
    return 0;
}
```

```
// Deklaracja biblioteki

/* Deklaracja zmiennych i tablicy
zawierającej ilości rozpoznanych
znaków. */

/* Wypełnienie tablicy ilości znaków
początkowymi zerami, aby obliczenia
były prawidłowe */
/* Pobieranie znaku z klawiatury, aż do
momentu wczytania znaku końca
zbioru. Jednoczesna weryfikacja
wczytanego znaku instrukcją if() oraz
naliczanie wystąpień zweryfikowanych
znaków */
/* Instrukcje wypisujące naliczone
ilości wystąpień zweryfikowanych
znaków */
```

Uzyskany wydruk z obydwu programów jest identyczny, chociaż zostały napisane za pomocą innych instrukcji. Co jest ciekawe dla wielu programów napisanych zupełnie inaczej uzyskujemy ten sam kod wynikowy. Wynika to z faktu, że kompilatory dokonują optymalizacji kodu programu źródłowego w wyniku, czego możemy uzyskiwać identyczne programy. Działanie takie jest często wykorzystywane przez hackerów i osoby produkujące nielegalne oprogramowanie. Uniemożliwia to faktycznie ustalenie pochodzenia i autora programu, jeśli różne wersje działają w identyczny sposób. Niestety wszystkie niekorzystne operacje dla użytkownika nieświadomego są wykonywane w tle. Starają się temu przeciwdziałać antywirusy i różnego rodzaju skanery. Należy zawsze pamiętać o posiadaniu aktualnego antywirusa i starać się zawsze posiadać jedynie legalne oprogramowanie dostarczone przez jego bezpośredniego producenta. Niestety najczęściej to sami użytkownicy są główną przyczyną wielu zniszczeń i włamań dokonanych na ich komputerach poprzez korzystanie z nielegalnych kopii oprogramowania. Na zakończenie rozdziału przedstawmy program oparty o omówione już przykłady, który będzie w stanie znajdować słowa.

Przykład 17

Napisać program powtarzający słowa zawarte w przeszukiwanym wzorcu. Poszukiwany wzorzec nie może zawierać znaków białych, zgodnie z podaną definicją.

```
#include<stdio.h>

int main()
{
    char c, a[4096], s[4096];
    int t, t1 = 1, t2, t3, i, j;

    printf("Podaj wzorzec do");
    printf("poszukiwania\n");
    gets(s);
    /* Wprowadzamy wzorzec tekstu,
w którym komputer będzie poszukiwał zadanego słowa. */
```

```
printf("Wprowadz tekst\n");
while ((c = getc(stdin)) != EOF)
    // Czytanie kolejnego znaku.
{
    t2 = (c == ' ' || c == '\n' || c == '\t');
/* Sprawdzenie czy, przeczytany znak jest białym
znakiem. */

    if (t1&&!t2){ t = 0; a[t] = c; }
    if (!t2&&!t1){ t++; a[t] = c; }
/*Jeżeli znalazłeś słowo, to zapamiętaj znak w tablicy a.
*/

    if (!t1&&t2)
/* Jeżeli koniec słowa, to wstaw znak pusty do tablicy i
rozpocznij poszukiwanie wzorca. */
    {
        t++; a[t] = '\0';
        t3 = 0;
        i = 0;
```

```

while (a[i] != '\0')
{
for (j = 0; s[j] == a[i]; j++, i++)
    if (s[j + 1] == '\0')
    {
        printf("%s\n", a); t3 = 1;
        break;
    }
    if (t3)break;
    i++;
}

```

```

/* Algorytm poszukiwania zadanego wzorca. */

```

```

    }

```

```

    t1 = t2;

```

```

}

```

```

return 0;

```

```

}

```

Przedstawiony algorytm poszukiwania słów w tekście jest oparty na poprzednio prezentowanym algorytmie zliczania słów. Uzupełniony jest on o algorytm wyszukiwania wzorca w momencie zakończenia słowa w tekście. Za zakończenie słowa w tekście analizowanym przez komputer rozumiemy tutaj sytuację, kiedy przeczytany znak jest białym znakiem i jednocześnie poprzedni znak był znakiem tekstu. Sam algorytm wyszukiwania wzorca zastosowany w przykładzie ma następującą postać.

```
t3=0;
i=0;

while(a[i]!='\0')
{
    for(j=0;s[j]==a[i];j++,i++)
    if(s[j+1]=='\0')
    {
        printf("%s\n",a);t3=1;
        break;
    }
    if(t3)
        break;
    i++;
}
```

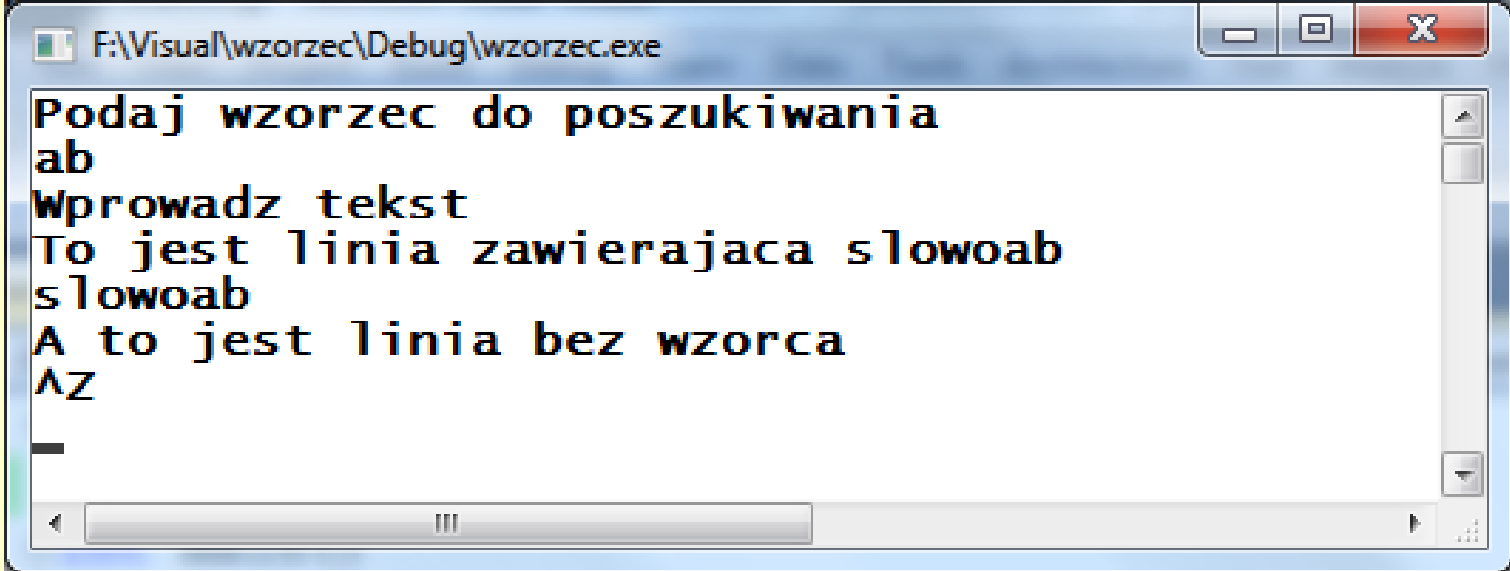
/* Do momentu napotkania znaku końca wykonuj operacje */

// Jeśli znak jest znakiem końca to wypisz

/* Jeśli znak jest innym znakiem wykonaj procedurę poszukiwania */

Zmienna `t3` jest zmienną logiczną i przyjmuje wartość jeden, jeżeli został znaleziony wzorzec. Blokuje ponowne wyświetlenie linii w przypadku powtórnego wystąpienia wzorca w słowie. Sam algorytm podpisuje pod każdym znakiem słowa wzorzec zwiększając wskaźnik wzorca i słowa jednocześnie. W przypadku, gdy wskaźnik wzorca wskazuje na pusty znak, to został znaleziony poszukiwany wzorzec i słowo zostanie wyświetlone. Do składni języka C/C++ zostało dodane słowo kluczowe *continue*. Słowo to jest zapożyczone z języka Fortran, w którym pełniło rolę kończącą pętlę w przypadku, gdy instrukcja kończąca pętlę była instrukcją wprowadzania, albo wypisywania danych. W języku C/C++ możliwość użycia słowa *continue* jest bardzo ograniczona, gdyż sama składnia programu zapewnia jednoznaczność zapisu pętli.

Okno konsoli wykonującej Przykład 17.



```
F:\Visual\wzorzec\Debug\wzorzec.exe

Podaj wzorzec do poszukiwania
ab
Wprowadz tekst
To jest linia zawierajaca slowoab
slowoab
A to jest linia bez wzorca
^Z
```

