

Wskaźniki i tablice

Rozdział ten jest poświęcony problemom magazynowania i przekazywania informacji między procedurami w trakcie realizacji kodu programu. Programista zazwyczaj musi rozwiązać nie tylko problem sposobu realizacji samego rozwiązania, ale również aranżacji danych w kodzie. Ponieważ kod programu swoje działanie opiera o posiadane dane, kluczowym zagadnieniem w procesie programowania jest zrozumienie, jaki charakter mają posiadane dane i odpowiednie ich użycie. W literaturze [21, 28, 43-45, 65, 84, 91] znajdziemy wiele podejść do sposobu archiwizacji danych. Przedstawione metody magazynowania danych pokazują, że odpowiednie rozwiązanie problemu magazynowania danych w procedurze czy całym programie znacznie podnosi efektywność. W informatyce w zasadzie podstawowym pojęciem pomagającym opisać charakter powiązań pomiędzy posiadanymi informacjami jest tablica. Tablica w swoich kolejnych komórkach może zawierać informacje powiązane ze sobą jakąś konkretną zależnością lub wyniki operacji przeprowadzonych w trakcie kolejnych iteracji.

Tablica w różnej formie, jedno czy dwu wymiarowej, jest elementem kodu praktycznie każdego programu, dlatego dobre zrozumienie tego pojęcia leży u podstaw opanowania zdolności programistyczny. Kolejnym elementem często wykorzystywanym w trakcie pracy z danymi jest wskaźnik. Bardzo często przechowujemy w nim informację opisującą wybrane cechy omawianych elementów. Dzięki operacjom na wskaźnikach możemy zrealizować wiele z pozoru skomplikowanych rozwiązań. Choć sam wskaźnik wydać się może Czytelnikowi pojęciem abstrakcyjnym jest on w istocie jednym z kluczowych elementów efektywnego programowania.

Informacje zawarte w kolejnych podpunktach rozdziału pokażą, w jaki sposób należy budować a następnie wypełniać tablice. W praktycznych przykładach pokażemy również zastosowanie wskaźników w rozwiązywaniu postawionego przed oprogramowaniem problemu.

Adresy i arytmetyka na wskaźnikach

Ważnym elementem rzemiosła programisty są wskaźniki. W języku C zmienne wskaźnikowe służą do przechowywania adresów obiektów (tj. zmiennych, tablic, struktur i funkcji). Możliwości operowania na adresach różnych obiektów jest wiele, jednak na początku wprowadźmy następujące deklaracje.

```
char *s;  
int *p;  
double *a;
```

Oznaczają one wskaźniki do zmiennej typu *char*, *int* i *double*.

Przykład 18

Napisać program podstawiający wartość zmiennej *a* do zmiennej *b* za pomocą arytmetyki na wskaźnikach.

```
#include<stdio.h>
int main()
{
    int a, b;
    int *p;
    scanf("%d",&a);

    p = &a;

    b = *p;

    printf("%d\n",b);
    _getch();
    return 0;
}
```

```
// Deklaracja biblioteki
```

```
/* Deklaracja wskaźnika p dla zmiennej
typu int */
```

```
// Pobranie adresu zmiennej a
```

```
/* Podstawienie do zmiennej b wartości
przechowywanej pod adresem p. W
rezultacie wykonana zostanie instrukcja
a=b; */
```

```
// Wydruk wartości
```

Wskaźników możemy używać w wyrażeniach arytmetycznych typu $y = *p + 1;$. W przykładzie tym wskaźnik zwiększa wartość zmiennej zapisanej pod adresem p o jeden i podstawia obliczoną wartość do zmiennej y . Operatory jednoargumentowe $*$ i $\&$ mają wyższy priorytet niż operatory arytmetyczne. Oznacza to, że najpierw jest pobierana wartość obiektu z adresu p , a następnie wykonywana jest operacja arytmetyczna. W przykładzie $*p = 0;$ wskaźnik podstawia zero do zmiennej wskazanej przez zmienną p . Wyrażenie $*p += 1;$ zwiększa zmienną zapisaną pod adresem p o jeden podobnie jak instrukcja znana już Czytelnikowi $(*p)++;$. Jeżeli w programie dokonaliśmy deklaracji $int\ pa, pb;$ to poprawne jest podstawienie na adresach $pa=pb;$

Tablice i zmienne wskaźnikowe

Tablica jest elementem wykorzystywanym praktycznie w każdym programie. Dzięki wykorzystaniu tablic możemy zapisywać wiele ciągów różnych informacji. W pewnym sensie tablicę w informatyce możemy rozumieć jako określony ciąg danych, w którym każdy jego element ma swój własny i niepowtarzalny indeks. Podobne pojęcie prezentują autorzy książek [21, 28, 43-45]. Deklaracja *int a[4]*; definiuje tablice o elementach *a[0]*, *a[1]*, *a[2]*, *a[3]*. Jeżeli *p* jest wskaźnikiem typu *int* to podstawienie

```
p = &a[0];
```

wstawia adres zerowego elementu tablicy, tak samo jak poniższa instrukcja

```
p = a;
```

Zatem zapis $*(p + i)$ jest równoważny z $a[i]$.
Istnieje zasadnicza różnica pomiędzy
wskaźnikiem odnoszącym się do tablicy i nazwą
tablicy, która jest stałą. O ile instrukcje $p = a$;
 $p++$; są poprawne, to niepoprawne są instrukcje
 $a++$; $p = \&a$; , gdyż a jest stałą.

Dynamiczna deklaracja tablic

Niestety nie zawsze znamy wymiar tablicy, którą będziemy potrzebowali. Czasem wymiar tablicy jest rozwiązaniem równania, które swoje miejsce ma w dalszej części kodu. W takiej sytuacji wygodnym rozwiązaniem jest dynamiczna deklaracja tablicy. Pierwszym ze sposobów zadeklarowania tablicy dynamicznej jest skorzystanie z funkcji *malloc()* z biblioteki *stdlib*. Czytelnik zechce zapamiętać, co oznaczają związane z dynamicznymi tablicami wyrażenia.

malloc – zwraca wskaźnik do przydzielonej pamięci albo NULL, jeśli nie jest dostępna wystarczająca ilość pamięci. Należy użyć rzutowania, aby został zwrócony wskaźnik odpowiedniego typu.

free – zwalnia przydzieloną pamięć.

Zobaczmy teraz, jak możemy wykorzystać dynamiczne przydzielanie miejsca w pamięci tablicom, których wymiar będziemy znali dopiero po wykonaniu określonej operacji.

Przykład 19

Napisać program deklarujący n –elementową tablicę typu *double*, a następnie wczytujący do niej elementy i wypisujący sumę przeczytanych liczb.

```
#include<stdio.h>
#include<conio.h>
#include<stdlib.h>
int main()
{
    int n,i;
    double s=0;
    printf("Podaj n:");
    scanf("%d",&n);

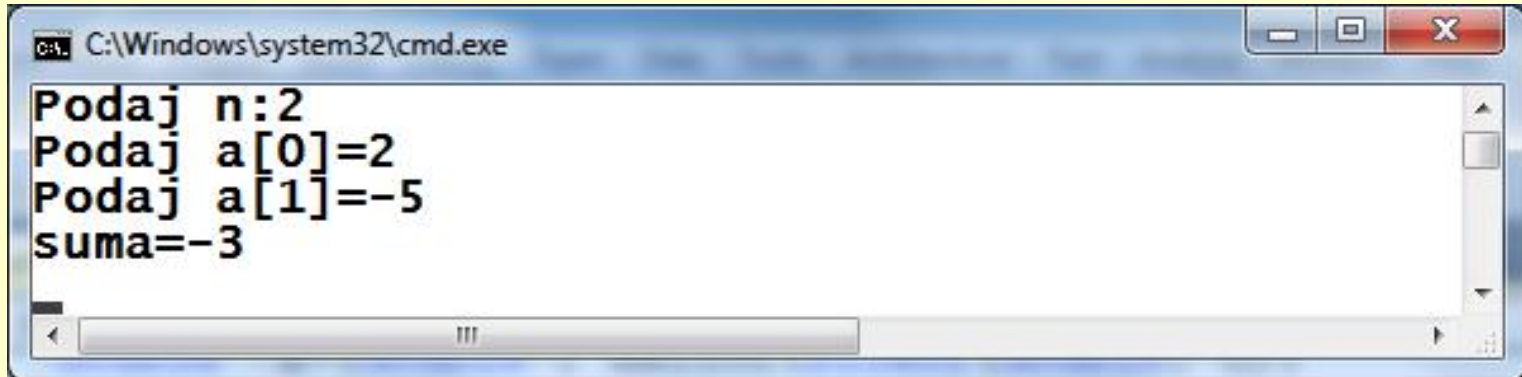
    double *a=(double*)
        malloc(sizeof(double)*n);
```

```
// Deklaracja bibliotek.
```

```
/* Funkcja malloc wymaga
rzutowania adresu (double *) i
podania wielkości rezerwowanej
pamięci
w bajtach. Funkcja sizeof(double)
zwraca ilość zajmowanego
miejsca przez typ double.
Mnożymy go przez wymiar
zadania n.*/
```

```
for(i=0;i<n;i++)
{
    printf("Podaj a[%d]=",i);
    scanf("%lf",a+i);
}
for(i=0;i<n;i++)
    s+=a[i];
printf("suma=%g\n",s);
free(a);
_getch();
return 0;
}
```

/* Sumowanie elementów tablicy
przez adres, następnie
wydrukowanie sumy.*/
// Uwolnienie zmiennej a.



```
C:\Windows\system32\cmd.exe
Podaj n:2
Podaj a[0]=2
Podaj a[1]=-5
suma=-3
```

realloc()

Zastanówmy się teraz nad możliwością dynamicznej rezerwacji miejsca w tablicy nie znając z góry liczby elementów do wczytania. Do rozwiązania w ten sposób postawionego zadania użyjemy funkcji *realloc()*, którą możemy opisać następującą definicją.

realloc – zwraca wskaźnik do rozszerzonego albo przesuniętego bloku pamięci albo NULL, jeśli nie jest dostępna wystarczająca ilość pamięci. Należy użyć rzutowania, aby został zwrócony wskaźnik odpowiedniego typu. W przypadku niemożliwości zmiany przydziału pamięci zarezerwowany blok nie zostanie zmieniony.

Przykład 20

Napisać program deklarujący tablicę typu *duble*, a następnie wczytujący do niej elementy i wypisujący sumę przeczytanych liczb. Wczytywanie liczb zostanie przerwane przez wpisanie końca zbioru w linii poleceń.

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

```
int main()
{
    int n = 0, i;
    double *a, e;
    a = (double *)malloc(0);
```

```
// Deklaracja bibliotek.
```

```
/* Zarezerwowanie wskaźnika
dla tablicy
o zerowej liczbie elementów.*/
```

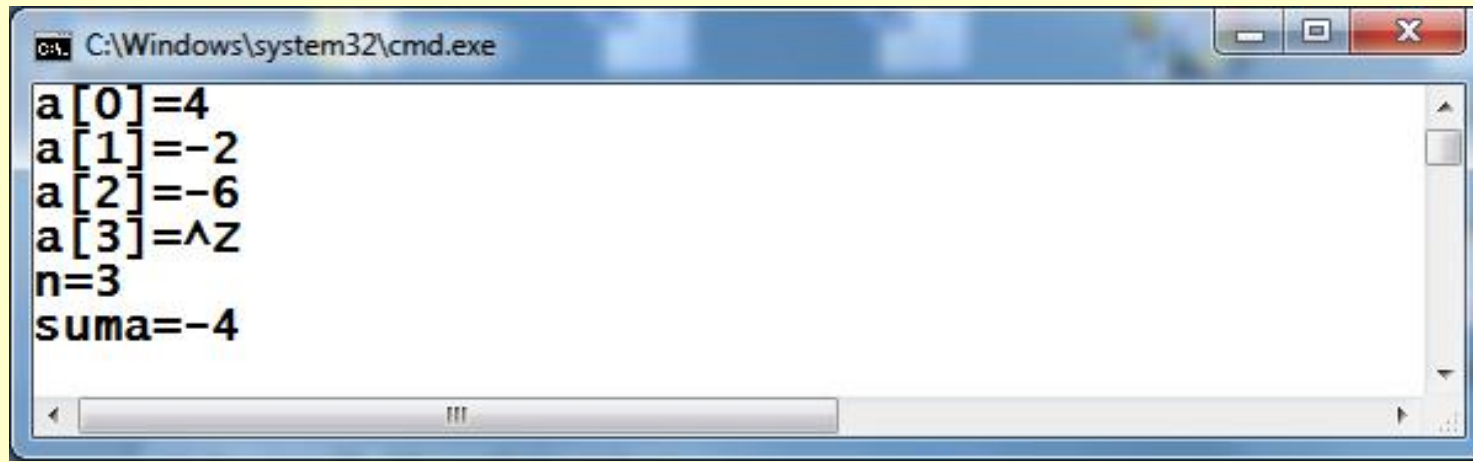
```
printf("a[0]=");  
while(scanf("%lf",&e)!=EOF)  
{  
    n++;  
    a=(double  
*)realloc(a,sizeof(double)*n);  
    a[n-1]=e;  
    printf("a[%d]=",n);  
}  
printf("n=%d\n",n);  
e=0;  
for(i=0;i<n;i++)  
    e+=a[i];  
printf("suma=%g\n",e);  
_getch();  
return 0;  
}
```

```
/* Funkcja scanf zwraca liczbę  
przeczytanych elementów albo  
EOF.*/
```

```
/* Relokacja pamięci tablicy.*/
```

```
/* Wstawienie przeczytanego  
elementu do tablicy.*/
```

Czytelnik porówna otrzymane rezultaty z przedstawionym oknem konsoli realizującej omawiany przykład sumujący elementy wczytane do tablicy na zasadzie arytmetyki wskaźników do momentu otrzymania białego znaku końca zbioru, co przedstawia Rys.

A screenshot of a Windows command prompt window. The title bar at the top reads "C:\Windows\system32\cmd.exe". The window contains the following text:

```
a[0]=4  
a[1]=-2  
a[2]=-6  
a[3]=^Z  
n=3  
suma=-4
```

The text is displayed in a monospaced font. The window has standard Windows window controls (minimize, maximize, close) in the top right corner.

Do konstrukcji tablicy możemy posłużyć się również kreatorem obiektów, dostępnym w rozszerzeniu języka ANSI C. Nie będziemy go mogli kompilować za pomocą *gcc* pod systemem operacyjnym Linux. W dalszej części książki zrezygnujemy z optymalności programów po kompilacji dla wielu systemów operacyjnych i będziemy posługiwali się jedynie elementami języka obiektowego. Nasz program będzie wyglądał bardzo podobnie, co Czytelnik zechce porównać z przedstawionym wcześniej kodem.

Przykład 21

Napisać program budujący tablicę typu *duble* za pomocą kreatora obiektów. Program wczytuje do tej tablicy elementy i wypisuje sumę przeczytanych liczb. Wczytywanie liczb zostaje przerwane przez wpisanie znaku końca zbioru w linii poleceń.

```
#include<stdio.h>
#include<conio.h>
int main()
{
    int n,i;
    double s=0;
    printf("Podaj n:");
    scanf("%d",&n);
```

```
    double *a=new double[ n ];
```

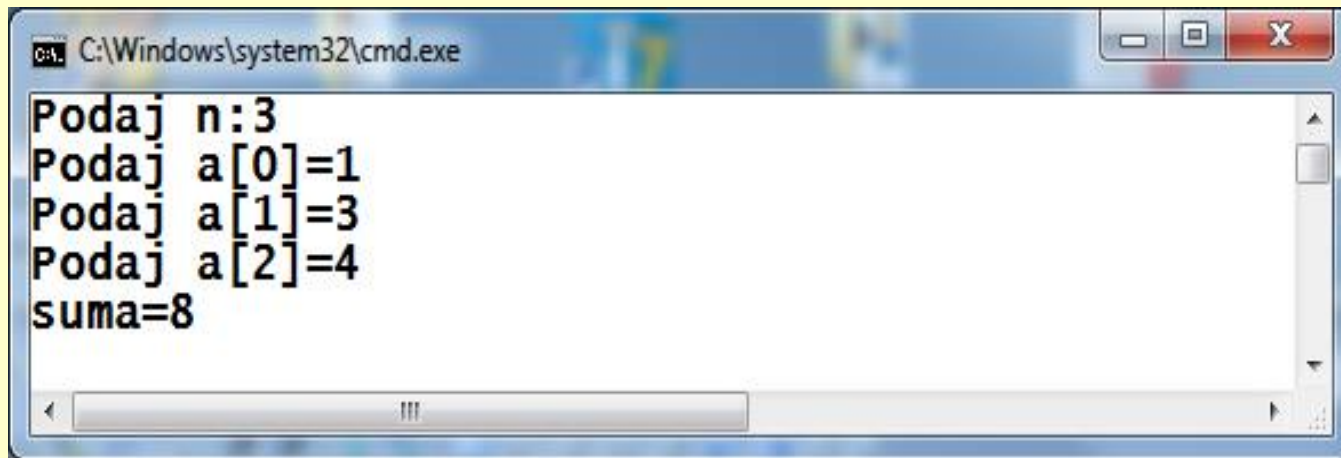
```
// Deklaracja biblioteki.
```

```
/* Kreator new buduje tablicę n-  
elementową przypisując ją do  
wskaźnika a.*/
```


<pre>for(i=0;i<n;i++) { printf("Podaj a[%d]=", i); scanf("%lf", a + i); } for(i=0;i<n;i++) s+=a[i]; printf("suma=%g\n",s); delete a; _getch(); return 0; }</pre>	<pre>/* Destruktor niszczy niepotrzebną tablicę.*/</pre>
--	--

Przedstawiony przykład powinniśmy skompilować kompilatorem *g++* pod systemem Linux. Obydwie z przedstawionych metod deklaracji tablicy działają poprawnie w Visual Studio.

Ekran konsoli wykonującej kod, który pokazuje Przykład 21



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:

```
Podaj n:3  
Podaj a[0]=1  
Podaj a[1]=3  
Podaj a[2]=4  
suma=8
```

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.

Tablice dwuwymiarowe i nadawanie wartości początkowych

Zastanówmy się teraz na możliwością poszerzenia własności poznanych już tablic. Ważnym rodzajem tablic, mającym ogromne zastosowanie w programowaniu, są tablice dwuwymiarowe, które najczęściej służą do przechowywania danych o charakterze złożonym. Tablice dwuwymiarowe deklarujemy tak samo jak jednowymiarowe, podając w nawiasach kwadratowych liczbę elementów. Składniki tablicy otrzymują numery rozpoczynające się od 0 i kończące na elemencie o indeksie o jeden mniejszym od zadeklarowanej liczby. Istnieje możliwość deklarowania tablic dynamicznych przez przydzielenie pamięci w czasie wykonywania programu, którą omówimy w dalszej części rozdziału. Deklaracja tablicy dwuwymiarowej jest postaci:

Typ nazwa[N][M]

Stałe N i M oznaczają wymiary tworzonej tablicy.

Zobaczmy, jak można wykorzystać tablice dwuwymiarowe w praktyce. Wykorzystamy do tego celu kalendarz. Spróbujemy napisać program, który będzie wypisywał nazwy miesięcy. Aby ułatwić zadanie, będziemy operować nazwami w języku angielskim, w którym nie występują znaki rozszerzone jak czcionki polskich samogłosek czy spółgłosek.

Przykład 22

Napisz program, który wczytuje nazwę miesiąca w postaci liczb od 1 do 12 i wyświetla pełną nazwę miesiąca w języku angielskim (January, ..., December).

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

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

```
char mi[12][10] = {  
    "January",  
    "February",  
    "Mach",  
    "April",  
    "May",  
    "June",  
    "July",  
    "August",  
    "September",  
    "October",  
    "November",  
    "December" };
```

```
int main()  
{  
    int m;  
    printf("Podaj miesiac :");  
    scanf("%d",&m);  
    printf("Miesiacem %d jest %s\n",m,&mi[m-1][0]);  
    _getch();  
    return 0;  
}
```

```
// Deklaracja biblioteki.
```

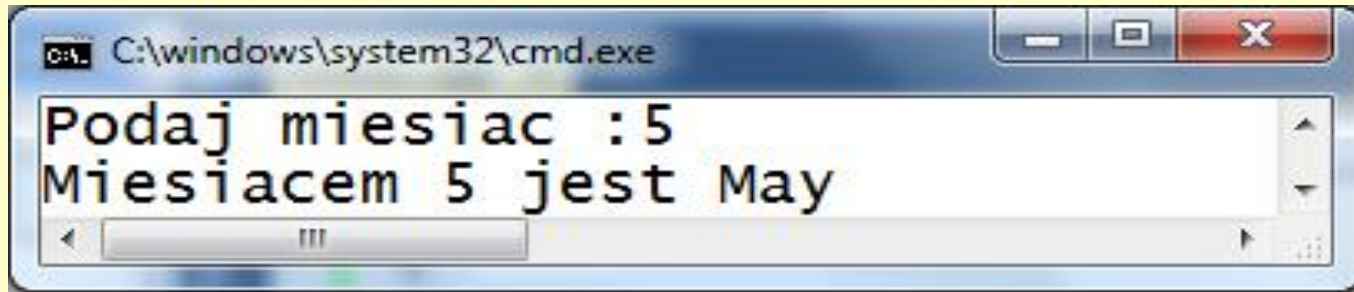
```
/* Deklaracja tablicy mającej  
12 wierszy, które odpowiadają  
liczbie miesięcy w roku.  
Tablica posiada 10 kolumn,  
które odpowiadają liczbie liter  
w najdłuższej nazwie miesiąca  
roku  
w języku angielskim.*/
```

```
/* Procedura wczytania  
numeru porządkowego  
miesiąca, a następnie wydruk  
nazwy miesiąca znajdującej  
się w wierszu o wczytanym  
numerze.*/
```

Ponieważ drukujemy wiersz tablicy zapisany w postaci ciągu znaków, możemy użyć przekształcenia `%s` w instrukcji `printf()`. Istotny jest fakt, że kompilator automatycznie dopisuje pusty znak na końcu tekstu, przez co nie musimy drukować wiersza znak po znaku. Tablica `mi` jest zadeklarowana w nagłówku jako tablica globalna, przez co jest widoczna dla wszystkich instrukcji i procedur w całym programie. W poprzednich przykładach deklarowaliśmy tablice na poziomie lokalnym procedury, przez co były one widoczne tylko w danej procedurze i nie można było operować nimi w dowolnym fragmencie kodu programu. W omawianym przykładzie zapamiętanie kolejnych znaków w tablicy możemy zilustrować w następujący sposób:

**mi	0	1	2	3	4	5	6	7	8	9
&mi[0][0]	J	a	n	u	a	r	y	\0		
&mi[1][0]	F	e	b	r	u	a	r	y	\0	
&mi[2][0]	M	a	r	c	h	\0				
&mi[3][0]	A	p	r	i	l	\0				
&mi[4][0]	M	a	y	\0						
&mi[5][0]	J	u	n	e	\0					
&mi[6][0]	J	u	l	y	\0					
&mi[7][0]	A	u	g	e	s	t	\0			
&mi[8][0]	S	e	p	t	e	m	b	e	r	\0
&mi[9][0]	O	c	t	o	b	e	r	\0		
&mi[10][0]	N	n	v	e	m	b	e	r	\0	
&mi[11][0]	D	e	c	e	m	b	e	r	\0	

Podając numer 5 miesiąca otrzymujemy następujący wydruk wyników na konsoli.

A screenshot of a Windows command prompt window. The title bar shows the path 'C:\windows\system32\cmd.exe'. The command prompt displays two lines of text: 'Podaj miesiac :5' and 'Miesiacem 5 jest May'. The window has standard Windows window controls (minimize, maximize, close) in the top right corner.

```
C:\windows\system32\cmd.exe
Podaj miesiac :5
Miesiacem 5 jest May
```

Ponieważ drukujemy wiersz tablicy zapisany w postaci ciągu znaków możemy użyć przekształcenia `%s` w instrukcji `printf()`. Istotny jest fakt, że kompilator automatycznie dopisuje znak pusty na końcu tekstu, przez co nie musimy drukować wiersz znak po znaku. Zastanówmy się teraz nad wykorzystaniem tablic w arytmetyce zadań obliczeniowych. Czytelnik zapewne zna układy równań i metody ich rozwiązywania. Zastosowanie tablic w rozwiązywaniu układów równań wydaje się naturalne i postaramy się teraz omówić to zagadnienie.

Przykład 23

Napisać program rozwiązujący zadany układ równań liniowych:

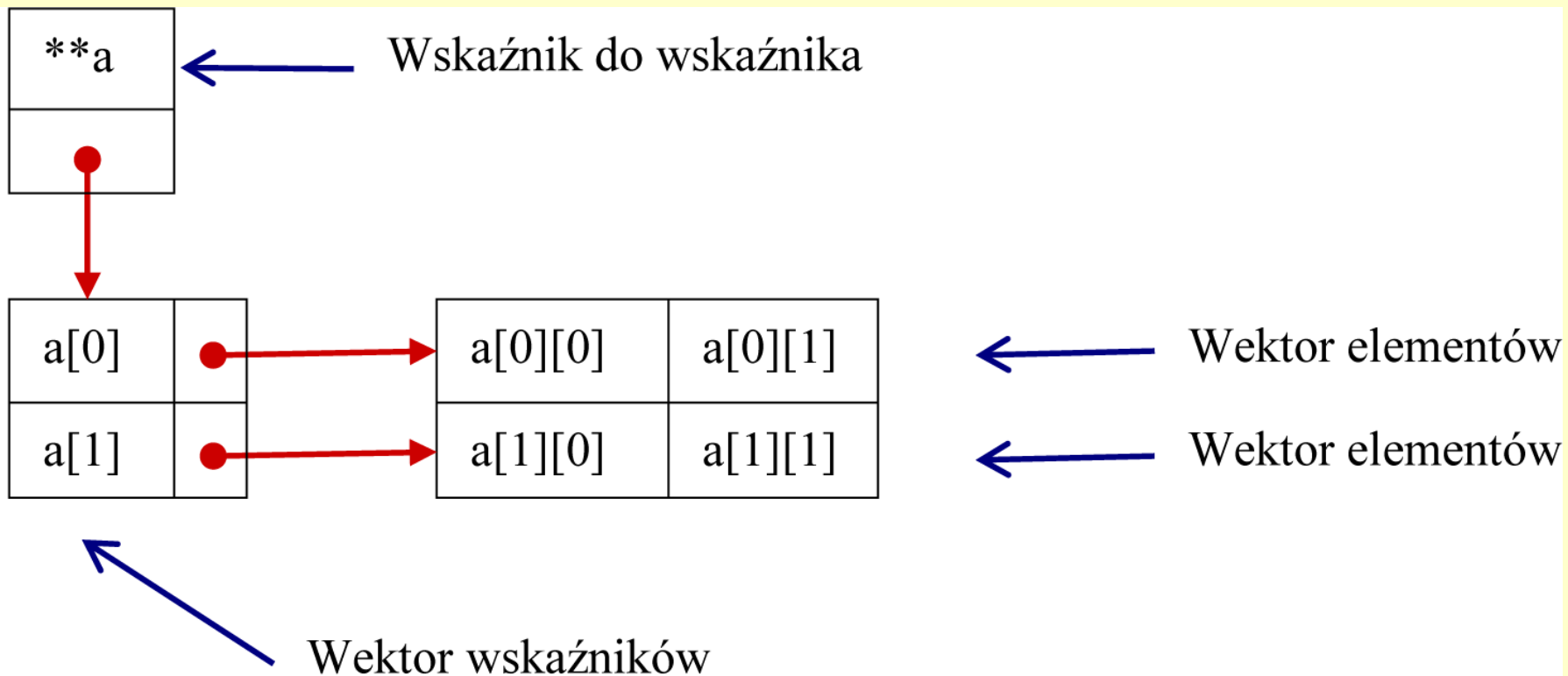
$$\begin{cases} a_{00}x_0 + a_{01}x_1 = b_0 \\ a_{10}x_0 + a_{11}x_1 = b_1 \end{cases}$$

Układ równań możemy zapisać w postaci macierzowej

$$\begin{bmatrix} a_{00} & a_{01} \\ a_{10} & a_{11} \end{bmatrix} \begin{bmatrix} x_0 \\ x_1 \end{bmatrix} = \begin{bmatrix} b_0 \\ b_1 \end{bmatrix}$$

lub prościej $Ax = b$, gdzie $A \in R^{2 \times 2}$ i $x, b \in R^2$.

Zapis macierzy w kodzie programu faktycznie jest macierzą deklarowaną za pomocą wskaźników, co ilustruje zamieszczony rysunek



Ilustracja 1. Ilustracja zapisu macierzy przy pomocy wskaźników.

Tak naprawdę w języku C/C++ nie istnieje macierz kwadratowa. Deklarujemy dwa wektory typu *double*, które w programie będą widoczne jak macierz o indeksach zaczynających się od zera i kończących się na indeksie o jeden mniejszym niż wymiar zadania. Takie właśnie podejście do zagadnienia tablic pokazuje literatura [21, 44-45, 59, 72-73, 79]. Stąd kod naszego programu będzie wyglądał następująco:

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

```
int main()
```

```
{  
    double W,Wx,Wy;  
    double **a = new double * [2];  
    a[0] = new double [2];  
    a[1] = new double [2];  
  
    double *b = new double [2];  
    int i, j;  
    for(i = 0; i < 2; i++)  
    {  
        for(j = 0; j < 2; j++)  
        {  
            printf("Podaj a[%d][%d]:", i, j);  
            scanf("%lf", a[i] + j);  
        }  
        printf("Podaj b[%d]:", i);  
        scanf("%lf", b + i);  
    }  
    W = a[0][0]*a[1][1] - a[0][1]*a[1][0];  
    Wx= b[0]*a[1][1] - a[0][1]*b[1];  
    Wy= a[0][0]*b[1] - b[0]*a[1][0];
```

```
// Deklaracja biblioteki
```

```
/* Deklaracja macierzy A */
```

```
/* Deklaracja wektora b */
```

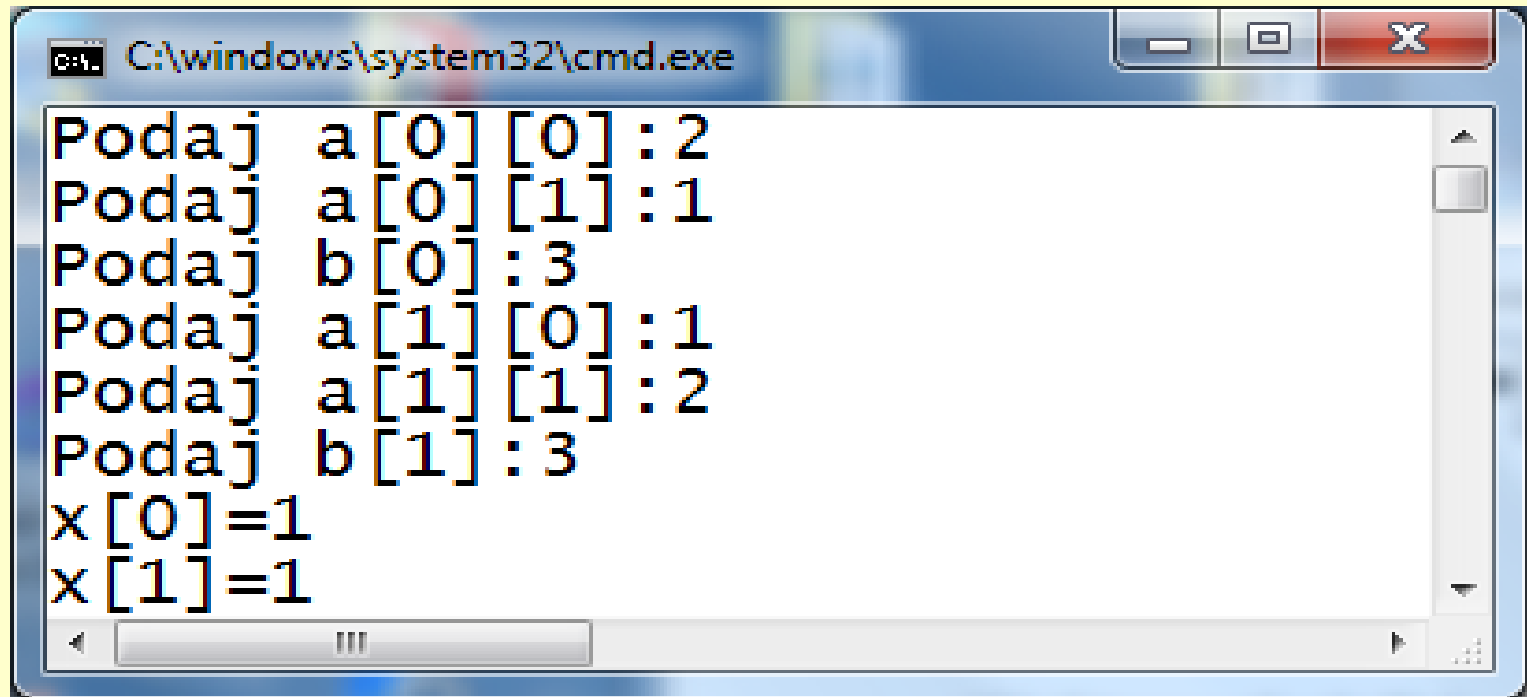
```
// Wczytywanie elementów macierzy A  
układu równań zapisanego wzorem (6.4.1)  
*/
```

```
// Wczytywanie elementów wektora b
```

```
// Wyliczenie wyznaczników układu  
zapisanego macierzowo wzorem (6.4.2) */
```

<pre>if(W != 0) { printf("x[0]=%g\n",Wx/W); printf("x[1]=%g\n",Wy/W); } else if(Wx != 0) printf("Układ rownan jest sprzeczny\n"); else printf("Układ rownan jest nieoznaczony\n"); }</pre>	<pre>/* Sprawdzenie kolejnych warunków rozwiązalności układu */</pre>
--	--

UWAGA. Omówiony w kodzie sposób deklaracji macierzy przenosi się na dowolny wymiar.



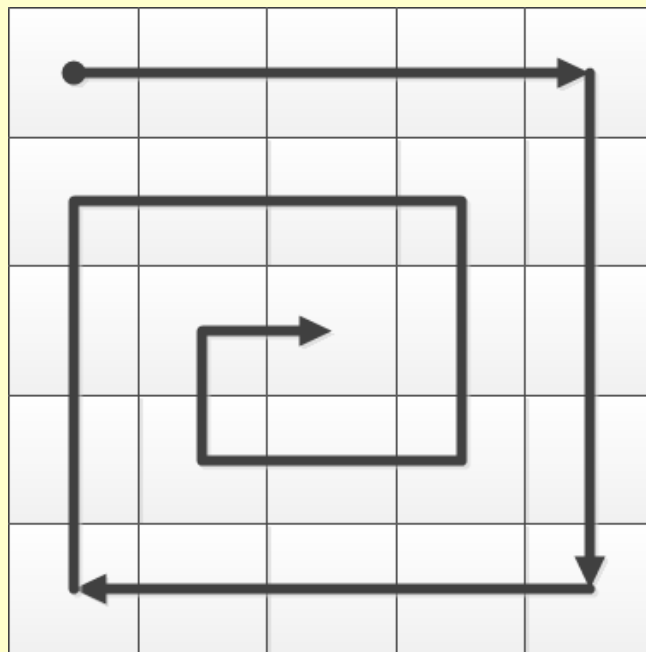
```
C:\windows\system32\cmd.exe

Podaaj a[0][0]:2
Podaaj a[0][1]:1
Podaaj b[0]:3
Podaaj a[1][0]:1
Podaaj a[1][1]:2
Podaaj b[1]:3
x[0]=1
x[1]=1
```

Prześledźmy teraz kolejny przykład pokazujący możliwości zastosowania tablic dwuwymiarowych w praktyce.

Przykład 24

W tablicę a o wymiarze $n \times n$ należy wpisać kolejne liczby naturalne od 1 do n^2 w spiralę rozpoczynając od elementu $a[0][0]$ i kończąc na elemencie $a[n/2][n/2]$ dla n -nieparzystego albo $a[n/2][n/2-1]$ dla n -parzystego. Sposób numeracji ilustruje rysunek.



Schemat blokowy pokazany na poprzedniej ilustracji pokazuje kolejność wykonywanych pętli. Powiązania pomiędzy poszczególnymi pętlami pokazują, w jaki sposób będzie wypełniana tablica, aby zachować kierunek zgodny ze strzałkami pokazanymi na ilustracji opisującej Przykład 24. Kod programu numerowania tablicy możemy napisać następująco:

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

```
int main()
```

```
{
```

```
    int n, i, j, t = 1, t1;
```

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

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

```
    int **a = new int* [n];
```

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

```
        a[i] = new int [n];
```

```
// Deklaracja biblioteki
```

```
// Deklaracja tablicy a[n][n]
```



```

for(i = 0; i < n/2; i++)
{
    t1=n - i - 1;
    for(j = i; j < t1; j++)
        a[i][j] = t++;
    for(j = i; j < t1; j++)
        a[j][t1] = t++;
    for(j = t1; j > i; j--)
        a[t1][j] = t++;
    for(j = t1; j > i; j--)
        a[j][i] = t++;
}
if(n%2 == 1) a[n/2][n/2] = t;
for(i=0;i<n;i++)
{
    for(j=0;j<n;j++)
        printf("%4d",a[i][j]);
    printf("\n");
}
}

```

// Ponumerowanie górnego wiersza

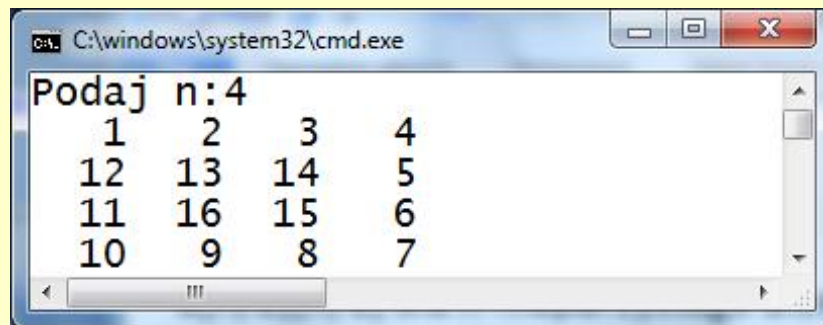
// Ponumerowanie prawej kolumny

// Ponumerowanie dolnego wiersza

// Ponumerowanie lewej kolumny

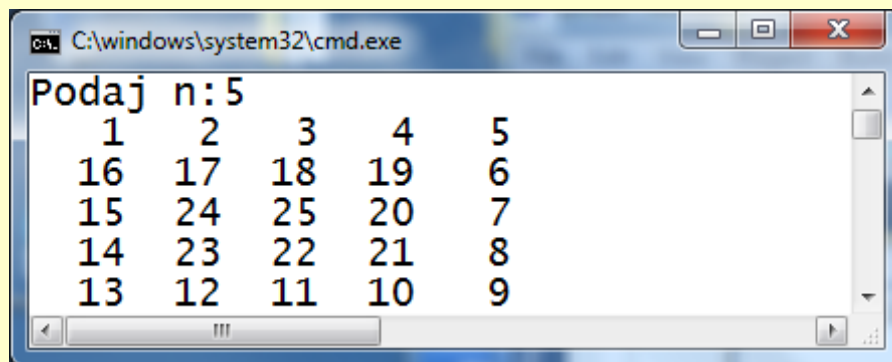
/* Wstawienie numeru w tarczy, jeśli n
jest nieparzyste */

Poniżej zostały przedstawione otrzymane wyniki dla zadanych w programie wymiarów tablicy. Dla $n=4$ numeracja ma następującą postać:



```
C:\windows\system32\cmd.exe
Podaj n:4
  1   2   3   4
 12  13  14   5
 11  16  15   6
 10   9   8   7
```

Dla $n=5$ numeracja ma centralny punkt w środku równy 25.

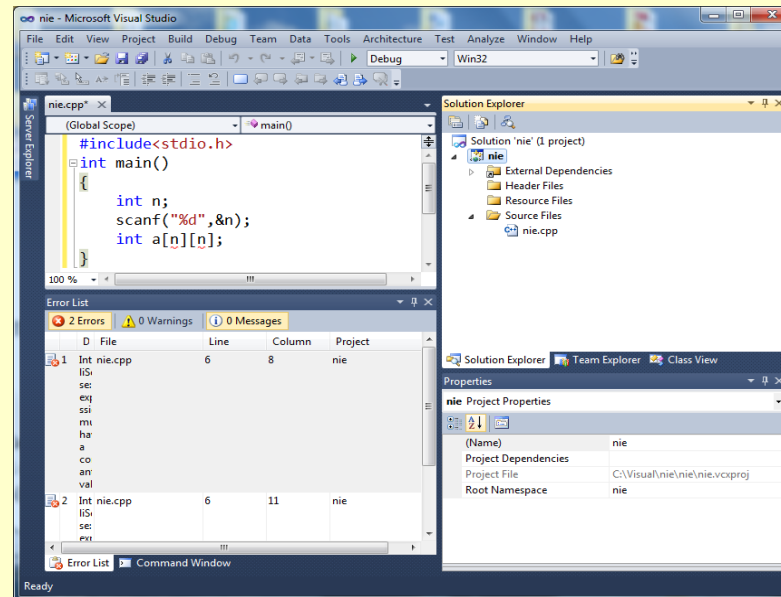


```
C:\windows\system32\cmd.exe
Podaj n:5
  1   2   3   4   5
 16  17  18  19   6
 15  24  25  20   7
 14  23  22  21   8
 13  12  11  10   9
```

Uwaga: Kompilatory *gcc* i *g++* pod systemem operacyjnym Linux zezwalają na konstrukcję tablicy dwuwymiarowej w następujący sposób.

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

Kompilator MS Visual Studio w takim przypadku sygnalizuje błąd, ze względu na brak jednoznaczności reprezentacji tablicy dwuwymiarowej w pamięci komputera.



W dalszej części książki będziemy starali się pokazać jedynie schematy blokowe zastosowanych funkcji, a nie całych programów, ze względu na to, że przedstawione algorytmy sortowania i metody macierzowe składają się z wielu powiązanych ze sobą procedur i funkcji. Takie działanie jest podyktowane złożonością prezentowanych procedur. Schematy blokowe takich programów byłyby bardzo rozległe. Zakładamy, że Czytelnik posiadając już wiedzę zgromadzoną w dotychczasowych przykładach jest w stanie bez większego trudu samodzielnie uzupełnić schematy blokowe funkcji i procedur, aby obrazowały cały kod programów. Wszystkie programy omawiane w dalszych rozdziałach książki są oparte o ogólny schemat blokowy pokazany na kolejnej ilustracji.

