Wstęp do wskaźników i dynamicznej alokacji pamięci w językach ANSI C i C++

/ Materiał dydaktyczny pomocniczy do przedmiotu Informatyka sem. III kier. Elektrotechnika/

Spis treści

1.Wprowadzenie....3
2. Wskaźniki i adresy....4
3. Przykładowe działania z użyciem wskaźników....6
4. Wskaźniki i tablice....7
5. Wskaźniki do znaków i tablice znaków....8
6. Wskaźnik do `void`....9
7. Tablice i arytmetyka wskaźników....9
8. Zakres wskaźników do komórek tablicy i działania na wskaźnikach....11
9. Mieszana składnia z użyciem wskaźników i tablic....13
10. Konstrukcja `typedef`....13
11. Złożone wskaźniki....14
    11.1 Wskaźnik do wskaźnika do int....14
    11.2 Tablica wskaźników do `int`....15
    11.3 Wskaźnik do tablicy jednowymiarowej typu `int`....15
    11.4 Wskaźnik do tablicy dwuwymiarowej typu `int`....16
    11.5 Przykłady innych wskaźników związanych z tablicami....18
12. Dynamiczna alokacja pamięci w ANSI C
    12.1 Funkcje `malloc` i `calloc` (stdlib.h) 18
    12.2 Dynamiczna alokacja tablic trójwymiarowych....21
    12.3 Dynamiczna alokacja tablic trójwymiarowych....26
13. Przekazywanie tablic do funkcji - standardy ANSI C i C99....27
14. Dynamiczna alokacja pamięci w C++....30
15. Literatura....34

1. Wprowadzenie

W materiale przedstawiono wybrane informacje na temat prostych wskaźników, wskaźników związanych z tablicami oraz dynamicznej alokacji pamięci.

W językach ANSI C i C++ dla każdego typu **X** (wbudowanego, pochodnego, zdefiniowanego przez użytkownika) istnieje skojarzony z nim typ wskaźnikowy **X***. Zbiorem wartości typu **X*** są wskaźniki do obiektów typu **X**. Do zbioru wartości typu **X** należy również wskaźnik pusty, oznaczany jako **0** lub **`NULL`**. Jednym z możliwych wystąpień typu wskaźnikowego jest zmienna wskaźnikowa, której definicja ma postać następującą:

nazwa_typu_wskazywanego *nazwa_zmiennej_wskaźnikowej;

Wartościami zmiennej wskaźnikowej mogą być wskaźniki do uprzednio zdefiniowanych obiektów (zmiennych, stałych). Jeżeli nie chcemy, aby zmienna wskaźnikowa wskazywała na jakiś obiekt, przypisuje się jej wartość zero lub **`NULL`**.

Przykład 1. Definicja wskaźnika do **`int`.**

```
int*  wski; /*  wski jest zmienną wskaźnikową mogącą wskazywać na
                obiekty typu int    */
```

Jeżeli typ wskazywanego obiektu nie jest znany w momencie definicji wskaźnika lub też typ wskazywanego obiektu może się zmieniać w trakcie działania programu, stosuje się wskaźnik do typu **`void`,** np.

```
void *wsk;
```

Zmienne wskaźnikowe mogą być definiowane lokalnie wewnątrz bloku lub na zewnątrz wszystkich bloków. Jeżeli zewnętrzna zmienna wskaźnikowa nie jest jawnie zainicjowana, to kompilator nadaje jej niejawnie wartość **0** (**`NULL`**) **.**

2. Wskaźniki i adresy

Wskaźnik z punktu widzenia programisty jest grupą komórek pamięci (rozmiar wskaźnika zależy od architektury procesora, zwykle są to dwa lub cztery bajty), które mogą pomieścić adres początkowy pewnego obiektu. Jeżeli **c** jest obiektem typu **char,** a **pc** ma być wskaźnikiem, który wskazuje na ten obiekt, to wskaźnik taki mógłby być zdefiniowany w sposób następujący:

```
char c;
char *pc;
pc=&c;
```

lub też

```
char c;
char pc=&c;
```

Operator adresowy **&** może być stosowany tylko do zmiennych i elementów tablic. Nie może być stosowany do stałych, wyrażeń i zmiennych typu **register**.

Zastosowanie do wskaźnika jednoargumentowego operatora *, oznacza tzw. **adresowanie pośrednie** zwane też odwołaniem pośrednim, daje ono wartość obiektu wskazywanego przez ten wskaźnik. Nie jest to jednak wyłączna interpretacja wyrażenia ***p**, gdzie **p** jest wskaźnikiem.

W języku ANSI C obiekt (zmienna) jest nazwanym obszarem pamięci; **l-wartość** (ang**. l-value**) jest wyrażeniem odnoszącym się do obiektu. Przykładem **l-wartości** jest identyfikator o odpowiednim typie i klasie pamięci.

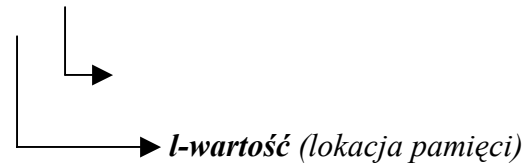
Niektóre operatory w wyniku swego zastosowania dają **l-wartość**, np. jeżeli **W** jest wyrażeniem wskaźnikowym, to ***W** jest **l-wartością** odnosząca się do obiektu wskazywanego przez **W**. Określenie **l-wartość** wywodzi się od instrukcji przypisania,

```
W1=W2;
```

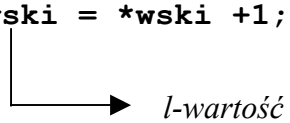
w której lewy argument musi **l-wartością** (reprezentować pewną lokację w pamięci).
Stosując poszczególne operatory należy zważać czy zastosowanie danego operatora daje w wyniku **l-wartość**.
Wyrażenie, które może się znaleźć wyłącznie po prawej stronie instrukcji podstawienia zwane jest **r-wartością** (ang. **r-value**).

Przykład 2. **(l-wartość)**

```
int x=0;
int *wski;
 x=x+1;
```




```
wski = &x; /* wskaźnik wski wskazuje na x */
*wski = *wski +1;
```



Przykład 3. Proste definicje wskaźników.

```
int x=1;
int y=2;
int z[10];
int * ip;
ip=&x;      /* wskaźnik ip wskazuje na komórkę x */
*ip=0;      /* odwołanie do komórki x za pośrednictwem
            wskaźnika, odpowiada x=0 */
ip=&z[0];  /* ip wskazuje na pierwszy element tablicy z */
*ip=0;       /* odwołanie do pierwszego elementu tablicy z,
             odpowiada z[0]=0  */
```

Składnia definicji wskaźnika jest taka jak składnia wyrażenia, w którym wskaźnik może wystąpić.

Przykład 4. Inicjacja wskaźnika.

```
int ii=38;
```

```
int wsk1=&ii;        /* wskaźnik wski jest inicjowany adresem
                        zmiennej ii, wskazuje na zmienną ii  */

int *wsk2=wsk1; /* inicjalizacja wskaźnika wsk2 wskaźnikiem
                   wsk1, obydwa wskaźniki  będą wskazywać na
                   zmienną ii */

int **wsk3=&wsk1;/* wsk3 jest wskaźnikiem wskazującym na
                    obiekty typu wskaźnikowego int*,
                    inicjowanym adresem wskaźnika wsk1 */
```

3. Przykładowe działania z użyciem wskaźników

```
int i=1,j;
int *wski=&i;
*wski=*wski+3;
```

Instrukcja w trzecim wierszu zwiększa wartość zmiennej `i` o `3`, funkcjonuje ona poprawnie, gdyż operator * ma wyższy priorytet niż operator arytmetyczny `+`. Podobnie instrukcja

```
j= ++ * wski;
```

spowoduje zwiększenie wartości **`*wski`** o `1` i następnie przypisanie do zmiennej *j*.

Instrukcja

```
j=(*wski) ++;
```

przypisze bieżącą wartość `*`**`wski`** do **`j`**, a następnie zwiększy wartość komórki wskazywanej przez **`wski`** o `1`.

Natomiast instrukcja

```
j=*wsk ++;
```

przypisze do zmiennej bieżącą wartość **`*wski`** i następnie zwiększy wskaźnik **`wski`** o 1.

Wyrażenie

```
*wski ++
```

jest równoważne wyrażeniu

```
*(wski++),
```

wynika to stąd, że operatory **`*`** i `++` mają ten sam priorytet i łączność prawostronną.

Gdy obliczane jest wyrażenie **wsk ++,** zwiększany jest wskaźnik **wsk**, jednak wartością wyrażenia jest oryginalna (początkowa) wartość wskaźnika **wsk**. A więc operator ***** odnosi się do oryginalnej wartości wskaźnika, a nie zwiększonej.

Zadanie 1. Zinterpretować wyrażenia ***++wski**, **++*wsk**, ***wski++,(*wski)++** oraz wyjaśnić i uzasadnić, które z wyrażeń mogą być l-wartością,

4. Wskaźniki i tablice

W języku C istnieje ścisła zależność między wskaźnikami a tablicami. Każdą operację na tablicy, którą można przedstawić przy użyciu indeksowania można też zrealizować przy użyciu wskaźników.

Definicja

```
int a[20];
```

definiuje tablicę 20 elementów **a[0],a[1],...,a[19].** Zapis **a[*i*]** oznacza **i**-ty element tablicy.

Jeżeli zdefiniujemy wskaźnik do obiektów całkowitych

```
int *pa;
```

wtedy przypisanie

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

powoduje, że wskaźnik **pa** wskazuje na zerowy element tablicy **a** (wskaźnik **pa** zawiera adres zerowego elementu tablicy **a**). Wtedy instrukcja przypisania

```
x=*pa;
```

skopiuje zerowy element tablicy **a** do zmiennej **x**.

Jeżeli wskaźnik **pa** wskazuje w pewnym momencie na pewien element tablicy, to wyrażenie **pa+1** wskazuje na następny element tablicy (z wyjątkiem przypadku, gdy **pa** wskazuje na ostatni element tablicy). Ogólnie **pa+i** wskazuje na **i**-ty element tablicy. A więc do **i**-tego elementu tablicy można się odwołać w sposób następujący

```
*(pa+i)
```

,

gdzie **pa**+**i** jest adresem elementu **a[i],** a ***(pa+i)** jest zawartością **a[i].**

Ponieważ nazwa tablicy jednowymiarowej reprezentuje jej zerowy element (faktycznie jest wskaźnikiem do zerowego elementu) można napisać zamiast

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

instrukcję następującą

```
pa=a;
```

Warto jednak zauważyć istotną różnicę między **pa,** a nazwą tablicy **a**; wskaźnik **pa** jest zmienną, a nazwa tablicy nie jest zmienną (jest stałym wskaźnikiem), można więc zastosować działanie

```
pa++,
```

natomiast nie jest dozwolone działanie

```
a++;
```

gdyż oznacza próbę zwiększenia stałego wskaźnika.

Przykład 5. Wczytywanie tablicy jednowymiarowej przy użyciu wskaźnika.

```
int a[20];
int *pa, i;
pa=a;
for (i=0;i<20;i++)
  {
     printf("\n Element %d =",i);
     scanf("%d",pa++);
     fflush (stdin);
  }
```

5. Wskaźniki do znaków i tablice znaków

Stała łańcuchowa

```
"Tekst"
```

jest przechowywana jako tablica znaków z ostatnim elementem równym '\0'.

Ponieważ **"Tekst"** jest typu wskaźnik do **char** , można zastosować następującą inicjalizację wskaźnika

```
char *s="Tekst";
```

Jest ona równoważna parze instrukcji

```
char *s;
s="Tekst";
```

Należy zauważyć, że wartość przypisywana jest do **s** , a nie do ***s**.

Definicja

```
char *s="Tekst";
```

mówi, że **s** jest wskaźnikiem, a nie nazwą tablicy znaków i może być zmieniany, natomiast tablicę znaków można zdefiniować w sposób podobny i zainicjować ją łańcuchem

```
char s1[ ]="Tekst";
```

wtedy jednak **s1** jest stałym wskaźnikiem, który nie może być zmieniany.

6. Wskaźnik do void

Typ **void** oznacza wartość nieistniejącą, której nie można wykorzystać ani w sposób jawny ani niejawny. Można jednak wykorzystywać wskaźnik do **void,** określany jako **void*.** Każdy wskaźnik do obiektu w C może być przekształcony do typu **void*** bez utraty informacji. Przekształcenie wyniku tej operacji z powrotem na wskaźnik typu, z którego dokonano przekształcenia do typu **void*,** przywraca oryginalną wartość wskaźnika.

Przykład 6. Użycie typu **void *.**

```
#include <stdio.h>
#include <conio.h>
int main()
  {
    int x=48;
    int *px=&x;
    void *pv;
    char *pc1,*pc2;
    pc1=(char*) px;          /* rzutowanie wskaźnika do int na
                                wskaźnik do char */
    pv=pc1;
    pc2=(char *)pv;         /* rzutowanie wskaźnika do void na
                               wskaźnik do char*/
    printf("\n c=%c", *pc2);
    getch();
  }
```

7. Tablice i arytmetyka wskaźników

Jak zaznaczono powyżej, do tablicy można się odwoływać bezpośrednio stosując indeksowanie lub też wskaźniki. W języku C stosując wskaźniki można posługiwać się tzw. ograniczoną arytmetyką na wskaźnikach. Oznacza to, że można stosować niektóre operacje arytmetyczne, by obliczyć odpowiedni wskaźnik. Dopuszczalne są dodawanie, odejmowanie,

mnożenie przez stałą, inkrementacja i dekrementacja. Należy jednak w każdym przypadku zważać na to, co może oznaczać wynik danego działania.

Przykład 7. Zastosowanie wskaźnika do wczytania tablicy jednowymiarowej z użyciem działań na wskaźnikach.

```
#include <stdio.h>
int main ()
  {
/*1*/ char litery[5];
/*2*/ char *ptr;
/*3*/ int liczba;
/*4*/ptr=litery;/* inicjalizacja wskaźnika adresem elementu
                   litery[0] */
/*5*/for(liczba=0;liczba<5;liczba++)
       {
/*6*/    *ptr=getchar();
/*7*/    ++ptr;
       }
/*8*/ptr=litery+4; /* wskaźnik ptr wskazuje na 5-ty element
tablicy litery*/
/* drukowanie znaków w odwrotnej kolejności */
/*9*/for(liczba=0;liczba<5;liczba++)
       {
         putchar(ptr);
/*10*/   --ptr;
       }
  system ("pause");
}
```

Linie 6 i 7 mogą być połączone jak niżej :

```
*ptr++=getchar();
```

Warto zauważyć, że wyrażenie ***ptr++** równoważne ***(ptr++)** jest interpretowane w sposób następujący: operatory ***** i ++ mają ten sam poziom priorytetu i łączność od prawej do lewej. Najpierw oblicza się wyrażenie

```
ptr++,
```

którego wartością jest początkowa wartość **ptr** i jednocześnie zwiększany jest wskaźnik **ptr** o **1**. Operator ***** stosowany jest do wyrażenia, czyli do początkowej wartości wskaźnika, a nie zwiększonej o 1. Rozważymy jeszcze, co oznacza wyrażenie

```
(*ptr)++;
```

Zapis ten oznacza sięgnięcie do komórki adresowanej przez **ptr** i zwiększenie jej wartości o 1.

Przykład 8. Różne sposoby użycia operatora ++ z operatorem *.

```
#include <stdio.h>
#include <conio.h>
int main()
  {
    int *ptr;
    int x[5];
    ptr=&x[0];
    *ptr++=5; /* wstawienie 5 do x[0], a ptr po wykonaniu
                 instrukcji wskazuje na x[1]*/
     x[1]=2;
     ++*ptr; /*  zwiększenie zawartości komórki x[1] o 1 */
      printf("\n x=%d",x[1]);
     *++ptr=5; /*  wyrażenie ptr wskazuje na x[2] i instrukcja
                   realizuje wpisanie 5 do x[2]*/
     printf("\n x=%d", x[2]);
     getch();
     return 0;
  }
```

8. Zakres wskaźników do komórek tablicy i działania na wskaźnikach

Standard ANSI C wymaga, aby wskaźnik stosowany do wskazywania tablic wskazywał na komórkę tablicy albo też na jedną komórkę poza tablicą. Dla definicji następującej

```
double tablica [10],*ptr;
```

dopuszczalnymi wartościami wskaźnika **ptr** są **tablica+1, tablica+2,..., tablica+10,** nie można jednak dokonywać dereferencji wskaźnika, gdy ma on wartość **tablica +10.**

Działania na wskaźnikach

Na wskaźnikach wskazujących na ten sam typ można dokonywać następujących operacji:

-inkrementacji (++)

-dekrementacji (--)

-dodawanie do wskaźnika liczby całkowitej

-odejmowanie od wskaźnika liczby całkowitej

-odejmowanie wskaźników od siebie

-porównywania dwóch wskaźników.

Pierwsze cztery operacje zilustrowano powyżej. Poniżej zostanie rozważone tylko porównywanie wskaźników w odniesieniu do tablic. Jeżeli dany wskaźnik wskazuje na element tablicy o wyższym indeksie niż drugi wskaźnik to jest on uważany za większy.

Przykład 9. Ilustracja porównania wskaźników

```
double tab1[4]={20.5,30.0,5.1,2.5};
double *pf1, *pf2;
pf1=pf2=tab1;
pf1++;
pf1++;
pf2++;
if (pf1>pf2)
     printf("\n Wskaźnik pf1 na element dalszy niż pf2");
```

Ponadto, jeżeli **pf1** i **pf2** są wskaźnikami wskazującymi na elementy tablicy, to różnica **pf1-pf2** mierzy odległość między elementami tablicy, jeśli zaś **pf1=pf2** (**pf1** równy **pf2**), to wskaźniki wskazują na tę samą komórkę oraz jeżeli **pf1 < pf2** to **pf1** wskazuje na element bliższy.

9. Mieszana składnia z użyciem wskaźników i tablic.

Mieszana składnia w tym przypadku, to konstrukcje, w których stosuje się nazwę tablicy jako wskaźnik, należy jednak brać pod uwagę, że nazwa tablicy jest stałym wskaźnikiem, który może być elementem operacji arytmetycznych na wskaźnikach, jednak nie może zmieniany. Przykładowo, przy definicji

```
char tablica[20];
```

do 10-tej komórki tablicy można odwołać się na dwa sposoby tzn.

```
tablica [9]=100;
```

lub

```
*(tablica+9)=100; /* w tym odwołaniu użyto nazwy tablicy jako wskaźnika */
```

Przykład 10. Ilustracja operacji na wskaźnikach.

```
char alfa[ ]={'A','B','C','D','E'};
char *p1,*p2;
char x;
p1=alfa;    /* wskaźnik p1 wskazuje na alfa[0], czyli komórkę zawierającą
znak 'A' */
p2= p1+2;   /* wskaźnik p2  wskazuje na komórkę zawierającą  znak 'C' */
x= *p2;     /*  do zmiennej  x wstawiony zostaje znak 'C'  */
        /* alternatywnie  można użyć instrukcji x= *(p1+2) lub alfa[2] */
p2=p2-1;    /* wskaźnik p2 wskazuje na komórkę zawierającą  znak 'B' */
x=*p2;      /* do zmiennej x  wstawiony zostaje znak 'B'  */
```

Stosując arytmetykę wskaźników nie trzeba uwzględniać fizycznych rozmiarów elementów, gdyż język C bierze pod uwagę rozmiary komórek przy obliczaniu nowego adresu (definiując wskaźnik definiujemy go, jako wskazujący na określony typ danych, a typ danych ma znany rozmiar).

10. Konstrukcja `typedef`

Konstrukcja ta pozwala na wprowadzanie nowych nazw dla istniejących typów, nazwy te mają zwykle prostszą formę niż oryginalna nazwa typu, co ułatwia definiowanie typów, w

których typ wprowadzony przez **typedef** jest elementem składowym. jak również ułatwia wprowadzanie zmian w programach.

Przykład 11. Proste zastosowania **typedef.**

```
typedef int price;
     price x,y,z;
```

Definicje zmiennych, jak powyższa, mogą być stosowane w różnych miejscach w programie, i jeżeli chciałoby się zmienić ich typ, np. na **float**, wystarczy zmienić nazwę typu w konstrukcji **typedef** , czyli zastosować

```
typedef float price;
```

Inny przykład to

```
typedef int* WSKINT;
     WSKINT  pi1,pi2;
```

11. Złożone wskaźniki

Poza prostymi wskaźnikami, stosowanymi powyżej, istnieją też wskaźniki bardziej skomplikowane. Należą do nich wskaźniki do tablic, wskaźniki do funkcji, jak również wskaźniki stanowiące kombinację innych wskaźników.

11.1 Wskaźnik do wskaźnika do int

Definicja takiego wskaźnika ma postać:

```
int **pointerName;
```

Wskaźnik **pointerName** jest wskaźnikiem do wskaźnika do **int**

Przykład 12. Proste zastosowanie wskaźnika do wskaźnika do **int**

```
int ii;
int *wski1;
int **wski2;
wski1=  &ii;
wski2= &wski1;  /* inicjacja wskaźnika wski2 adresem
                   wskaźnika wski1 */
**wski2= 10;    /* zapis wartości 10 do zmiennej ii */
```

Podobnie można tworzyć wskaźniki do wskaźników dla innych typów podstawowych.

11.2 Tablica wskaźników do `int`

Definicja takiej tablicy ma postać:

```
int *tableName[5];
```

`tableName[5]` jest tablicą, której elementami są wskaźniki do **`int`**.

Przykład 13. Przykład użycia tablicy wskaźników do **`int`.**

```
#include <stdio.h>

int main()
{
  int *ptab[5];
  int a[2]={2,3};
  int x=1;
  ptab[0]=&x; /* przypisanie pierwszemu elementowi
                 tablicy wskaźników adresu zmiennej x */
  ptab[1]=a; /* przypisanie tablicy wskaźników adresu
                elementu a[0] */
  printf("\n x=%d", *ptab[0];
  printf("\n a[0]=%d",*ptab[1]);
  system("pause");
}
```

11.3 Wskaźnik do tablicy jednowymiarowej typu `int`

Przykładowa definicja takiego wskaźnika do tablicy o ilości elementów **`ROZMIAR`** (w ANSI C stała **`ROZMIAR`** musi być zdefiniowana przy użyciu **#define**) ma postać:

```
int (*wsk)[ROZMIAR];
```

a więc **`wsk`** jest wskaźnikiem do tablicy liczb typu int o rozmiarze **`ROZMIAR.`**

Aby zrozumieć istotę takiego wskaźnika można sprawdzić, co powoduje jego inkrementacja. Inkrementacja zwykłego wskaźnika do **`int`** powoduje zwiększenie jego wartości o 4 (czyli rozmiar **int** w bajtach), natomiast działanie **`wsk++`** spowoduje jego zwiekszenie o 4 * **`ROZMIAR`** bajtów, czyli o rozmiar całej tablicy. Można to sprawdzić uruchamiając prosty program. W programie tym zilustrowano również użycie zdefiniowanego wskaźnika do do tablicy jednowymiarowej liczb typu **`int`** do wczytania i drukowania tablicy dwuwymiarowej.

Przykład 14. Przykład badania własności wskaźnika do tablicy liczb typu **`int`** i użycia takiego wskaźnika do wczytywania tablicy dwuwymiarowej**.**

```
#include <stdio.h>
#define ROZMIAR 3
```

```
typedef int(*WSKTAB1D)[ROZMIAR];
int main ( )
  {
   WSKTAB1D wsk1;/* definicja z użyciem typu zdefiniowanego
   przy użyciu typedef*/
   int (*wsk)[ROZMIAR];

   int a[3][3];
   int l1,l2;
   int i,j;
   wsk=a;/* przypisanie wsk adresu zerowego wiersza tablicy a
      */
   wsk1=wsk; /* obydwa wskaźniki wskazują na a[0][0] */
   printf("\n wydruk wartosci wskaznika wsk=%p",wsk);
   l1=wsk;/* zapisanie w zmiennej typu int adresu zawartego we
            wskazniku wsk */
   wsk++;
   printf("\n wydruk wartosci wskaznika po zwiekszeniu\
             wsk=%p",wsk);
   l2=wsk;
   printf("\n  roznica wartosci wskaznikow=%d\n",l2-l1);


   for (i=0;i<3;i++)
     for (j=0;j<3;j++)
       {
        printf("\na[%d][%d]=",i,j);
        scanf("%d",&wsk1[i][j]);
        fflush(stdin);
       }
   for (i=0;i<3;i++)
     {
      for (j=0;j<3;j++)
        printf(" %d",wsk1[i][j]);
        printf("\n");
     }
   system("pause");
}
```

11.4 Wskaźnik do tablicy dwuwymiarowej typu int

Wskaźnik taki można zdefiniować w sposób następujący:

```
#define  N  2
int (*wsk2D)[ ][N];
```

wsk2D jest wskaźnikiem do tablicy dwuwymiarowej liczb typu **int.**

Przykład 15. Przykład użycia wskaźnika do tablicy dwuwymiarowej liczb typu int do wczytywania tablicy trójwymiarowej (w rozważanym przypadku użycie wskaźnika nie jest potrzebne, w przykładzie chodzi o pokazanie tworzenia i użycia wskaźnika)

```
#include <stdio.h>
#define N    2
typedef int(*WSKTAB2D)[ ][N] ;
int main ( )
 {
  int (*wsk2D)[N][N];/* wskaźnik do tablicy dwuwymiarowej*/

  WSKTAB2D  wsk2D1;/* alternatywna definicja wskaźnika do
                      tablicy dwuwymiarowej przy użyciu
                      wprowadzonej nazwy typu*/
  int b[N][N][N]; /* definicja tablicy trójwymiarowej*/
  int i,j,k;
  wsk2D=b;
/*wczytywanie tablicy trójwymiarowej przy użyciu wskaźnika*/
  for (i=0;i<N;i++)
    for (j=0;j<N;j++)
      for (k=0;k<N;k++)
       {
         printf("\n a[%d][%d][%d]=",i,j,k);
         scanf("%d",&wsk2D[i][j][k]);
         fflush(stdin);
       }
/*drukowanie tablicy trójwymiarowej przy użyciu wskaźnika*/
  for (i=0;i<N;i++)
   for (j=0;j<N;j++)
     for (k=0;k<N;k++)
       printf("\n b[%d][%d][%d]=%d",i,j,k,wsk2D[i][j][k]);
       printf("\n");
system("pause");
}
```

11.5 Przykłady innych wskaźników związanych z tablicami

```
#define   N    2

typedef int(*WSKTAB1D)[N] ;  /* definicja typu:  wskaźnik do tablicy N-elementowej
                                    typu int*/

int (* ( *wskpp[N]) ) [N];          /* N-elementowa tablica typu wskaznik do
                                    N-elementowej tablicy liczb typu int*/
WSKTAB1D *tab_wskpp [N]; //  ta sama definicja co powyżej z użyciem nazwy typu
```

12. Dynamiczna alokacja pamięci

W języku ANSI C istnieją zasadniczo dwa rodzaje zmiennych:

- **zmienne zwykłe** posiadające nazwę definiowane w funkcji main() lub na zewnątrz wszystkich funkcji lub w pewnym bloku. Każda zmienna tego rodzaju posiada swoją nazwę oraz określony typ.
- **zmienne dynamiczne** tworzone i usuwane w trakcie działania programu; taki sposób przydzielania pamięci zwany jest alokacją w trakcie działania programu (ang. run-time allocation). Zmienne te nie posiadają nazw, znane są wyłącznie adresy przydzielonej im pamięci (wskaźniki do tej pamięci).

Do przydzielania pamięci zmiennym dynamicznym służą w ANSI C funkcje **malloc** i **calloc.** Do usuwania zmiennych dynamicznych stosuje się funkcję **free**.

12.1 Funkcje `malloc` i `calloc` (stdlib.h)

Każda z tych funkcji alokuje przydziela pamięć i zwraca adres tej pamięci (wskaźnik do tej pamięci). Rozmiar przydzielanej pamięci nie musi być znany podczas kompilacji.

Funkcja `malloc`

Nagłówek funkcji tej ma postać następującą:

```
void * malloc (int);
```

Funkcja **`malloc`** oczekuje, jako swojego argumentu, liczby bajtów, które mają być przydzielone w danym wywołaniu funkcji. Jeżeli przydzielenie pamięci jest możliwe, funkcja

zwraca wskaźnik do tej pamięci, jeśli nie, funkcja zwraca **NULL** (zerowy wskaźnik). Zwracany wskaźnik jest typu **void*,** czyli jest to wskaźnik do **void**. Wskaźnik ten musi być przekształcony na wskaźnik do żądanego typu. Język C gwarantuje, że wskaźnik do **void** może być przekształcony na wskaźnik do każdego innego typu (w języku C++ konieczne jest wykonanie rzutowania). W języku ANSI C to przekształcenie może być niejawne.

Przykład 16. Zastosowanie funkcji malloc do alokacji pamięci dla zmiennej dynamicznej typu *int*.

```
#include <stdio.h>
#include <conio.h>
int main()
  {
    int *ptr;
    ptr=(int*) malloc( sizeof(int));
    if (ptr==NULL)
      {
        printf("\n Przydzielenie pamięci nie byłomożliwe");
        getch();
        return 1;
      }
    printf(" Podaj wartosc zmiennej :");
    scanf("%d", ptr);
    printf("\n Wartosc to %d:", *ptr);
    free(ptr);
    getch();
    return 0;
  }
```

Funkcja calloc

Nagłówek funkcji tej ma postać następującą:

```
void * calloc (int,int);
```

Funkcja **calloc** oczekuje dwóch argumentów typu **int**. Pierwszy argument oznacza liczbę bloków pamięci, które mają zostać przydzielone, a drugi rozmiar pojedynczego bloku. Funkcja zwraca wskaźnik do pierwszego bajtu pierwszego bloku. Wskaźnik ten jest typu **void*** i musi być rzutowany na wskaźnik do wymaganego typu, obowiązuje ta sama zasada co dla **malloc.**

Przykład 17. Zastosowanie funkcji calloc do alokacji pamięci dla tablicy liczb typu **double**.

```
#include <stdio.h>
#include <conio.h>
int main()
  {
     double *ptr;
     int i;
     ptr=(double *) calloc(5, sizeof(int));
     for (i=0;i<5; i++)
       {
         printf(”\n Podaj element %d”, i);
         scanf("%lf", ptr++);
         fflush(stdin);
       }
     ptr=ptr-5;/* przesuniecie wskaznika na początek tablicy*/
     for (i=0;i<5;i++) {
       printf("\n Element [%d]=%f", *ptr++);
     ptr-=5;
     free(ptr);
     getch();
     return 0;
  }
```

Przykład 18. Zastosowanie funkcji **calloc** do alokacji pamięci dla tablicy liczb typu **double**.

bez zmiany pozycji wskaźnika. Efekt taki można uzyskać stosując do odwoływania się do elementów tablicy wyrażenia **ptr[i]** lub **(ptr+i).**

```
#include <stdio.h>
#include <conio.h>
int main()
  {
     double *ptr;
     int i;
     ptr=(double *) calloc(5, sizeof(int));
     for (i=0;i<5; i++)
       {
         printf(”\n Podaj element %d”, i);
         scanf("%lf", ptr[i]);
      // scanf("%lf", ptr+i); 2- wariant
         fflush(stdin);
       }
     for (i=0;i<5;i++) {
       printf("\n Element [%d]=%f",ptr[i]);
     //printf("\n Element [%d]=%f",*(ptr+i)); 2-gi wariant
     free(ptr);
     getch();
     return 0;
  }
```

Przykład 19. Dynamiczna alokacja tablicy z użyciem funkcji zwracającej wskaźnik.

```
#include <stdio.h>
#include <conio.h>
#include <stdlib.h>
int * wczyt_tab(int n);
int main(int argc, char **argv)
 {
   int *pa,i,n=5;
   pa=wczyt_tab(5);
   for (i=0;i<n;i++)
     printf("\n Element[%d]=%d",i, *pa++);
   pa-=5;
   free(pa);
   getch();
   return 0;
}
// definicja funkcji zwracającej wskaźnik do wczytanej tablicy
int * wczyt_tab(int n)
  {
    int i,*px;
    px=(int*) malloc(n*sizeof(int));
    for (i=0;i<n;i++)
      {
        printf("\n Podaj element[%d]=",i);
        scanf("%d",px++);
        fflush(stdin);
      }
    px=px-n;
    return px;
  }
```

Warto zauważyć, że funkcja **wczyt_tab** zwraca wskaźnik będący zmienną lokalną, ale wskazujący na zmienną dynamiczną, która jest alokowana na stogu (ang. heap) poza obszarem zmiennych lokalnych funkcji i alokowany obszar istnieje po zakończeniu działania funkcji.

12.2 Dynamiczna alokacja tablic dwuwymiarowych

Poniżej zostanie opisana alokacja pamięci dla tablic dwu- i trójwymiarowych, dla tablic o większej liczbie rozmiarów można postępować w sposób analogiczny definiując wskaźniki do odpowiednich typów. Poniżej w przykładzie podano dwa sposoby alokacji dla tablic dwuwymiarowych: alokację przy zastosowaniu tablicy wskaźników do **int** oraz wskaźnika do tablicy jednowymiarowej liczb typu **int .**

Przykład 20. Dynamiczna alokacja tablic dwuwymiarowych.

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

```
#include <stdlib.h>
#include <conio.h>
 #define  N  2
typedef  int (*PTAB1D)[N];
/* definicja typu wskaznik do int */

int main()
  {
   int *pointer_array[N];// pointer_array- tablica
                         // wskaznikow
   int(* pointer_to_array)[N]; // pointer_to_array -
                               //wskaznik  do tablicy

/*alternatywna definicja wskaznika do tablicy  z
      uzyciem typu PTAB1D  to
    PTAB1D pointer_to_array; */

   int i,j;
   int l1,l2;
   int  *p1,*p2;
   int x[2];

/*for (i=0;i<2;i++)
   pointer_array[i]=(int*)calloc(2,sizeof(int));*/

// alokacja dynamiczna dla pierwszego elementu tablicy
    // wskazników  pointer_array
   pointer_array[0]=(int*)calloc(1,sizeof(int));

//alokacja dynamiczna dla drugiego elementu tablicy
   wskazników  pointer_array

   pointer_array[1]=x;
/* Powyzej zastosowano odrębną alokację dla dwóch elementow,
   by pokazać, że te alokacje są niezależne*/

   pointer_to_array= (int (*)[N])calloc(N*N,sizeof(int));




      printf("\n Wczytywanie i drukowanie tablicy\
                dwuwymiarowej dynamicznej");
      printf("\n przy uzyciu tablicy wskaznikow do int");

      getch();
      for (i=0;i<N;i++)
        for (j=0;j<N;j++)
          {
```

```
            printf("\n element[%d][%d]=",i,j);
            scanf("%d",&(pointer_array[i])[j]);
            fflush(stdin);
          }


   for (i=0;i<N;i++)
     for (j=0;j<N;j++)
       printf("%d  ",(pointer_array[i])[j]);


   printf("\n Wczytywanie i drukowanie tablicy \
            dwuwymiarowej dynamicznej");
   printf("\n przy uzyciu wskaznika do tablicy liczb\
                          typu int");

   for (i=0;i<N;i++)
     for (j=0;j<N;j++)
       {
         printf("\n element[%d][%d]=",i,j);
         scanf("%d",&pointer_to_array[i][j]);
         fflush(stdin);
       }

   for (i=0;i<N;i++)
     for (j=0;j<N;j++)
           printf("%d  ",pointer_to_array[i][j]);


   printf("\n styl adresowamia z wykorzystaniem\
             nazwy tablicy jako wskaznika");

// przyklad liczbowy na adresach
   for (i=0;i<N;i++)
     for (j=0;j<N;j++)
       {
         printf("\n element[%d][%d]=",i,j);
         scanf("%d",*(pointer_to_array+i)+j);
         fflush(stdin);
       }

/*
       Pobranie wskaznika do tablicy int i przesuniecie go
, tak aby wskazywal adres i-tego wiersza, pobranie adresu
 i-tego wiersza  i przesunięcie go o j pozycji by
 wskazywal j-ta wartosc w wierszu */



   for (i=0;i<N;i++)
     for (j=0;j<N;j++)
```

```
      printf("%d  ",*(*(pointer_to_array+i)+j));
```

```
/* Dalsza część programu pokazuje związki między wskaźnikami a
adresami, m.in. jakie zmiany adresów powodują poszczególne
działania na wskaźnikach*/
```

```
   printf("\n Drukowanie wskaznikow zwiazanych z tablica\
             wskaznikow pointer_array");

   l1=(int)pointer_array[0];
   l2=(int)(pointer_array[1]);
   printf("\n Wartosc wskaznika pointer_array[0]=%p",
                                  pointer_array[0]);
   printf("\n Wartosc wskaznika pointer_array[1]=%p",
                                  pointer_array[1]);
   printf("\n Roznica wskaznikow=%d",l2-l1);



   l1=(int)(&pointer_array[0][0]);
   l2=(int)(&pointer_array[0][1]);
   printf("\n Wartosc wskaznika &pointer_array[0][0]=%p",\
                                 &pointer_array[0][0]);
   printf("\n Wartosc wskaznika &pointer_array[0][1]=%p",\
                                  &pointer_array[0][1]);
   printf("\n Roznica wskaznikow=%d",l2-l1);



   l1=(int)(&pointer_array[1][0]);
   l2=(int)(&pointer_array[1][1]);
   printf("\n Wartosc wskaznika &pointer_array[1][0]=%p",\
                                  &pointer_array[1][0]);
   printf("\n Wartosc wskaznika  &pointer_array[1][0]=%p",
                                  &pointer_array[1][1]);
   printf("\n Roznica wskaznikow=%d",l2-l1);
   getch();
```

```
 /* Wyniki przebiegu fragmentu programu

   printf("\n Drukowanie wskaznikow zwiazanych\
       z tablica wskaznikow pointer_array");
   Wartosc wskaznika pointer_array[0]=01BF4D6C
   Wartosc wskaznika pointer_array[1]=01BF4D7C
   Roznica wskaznikow=16
```

```
  Wartosc wskaznika &pointer_array[0][0]=01BF4D6C
  Wartosc wskaznika &pointer_array[0][1]=01BF4D70
  Roznica wskaznikow=4

  Wartosc wskaznika &pointer_array[1][0]=01BF4D7C
  Wartosc wskaznika &pointer_array[1][0]=01BF4D80
  Roznica wskaznikow=4

  */

   printf("\n Drukowanie wskaznikow zwiazanych z \
              wskaznikiem array_pointer");
  l1=(int)pointer_to_array;
  l2=(int)(pointer_to_array+1);
  printf("\n Wartosc wskaznika pointer_to_array=%p",\
                               pointer_to_array);
  printf("\n Wartosc wskaznika pointer_to_array+1=%p",\
                               pointer_to_array+1);
  printf("\n Roznica wskaznikow=%d",l2-l1);



  l1=(int)(&pointer_to_array[0][0]);
  l2=(int)(&pointer_to_array[0][1]);
  printf("\n Wartosc wskaznika &pointer_to_array[0][0]=%p",\
                                     &pointer_to_array[0][0]);

printf("\n Wartosc wskaznika &pointer_to_array[0][1]=%p",\
                                    &pointer_to_array[0][1]);
  printf("\n Roznica wskaznikow=%d",l2-l1);

  l1=(int)(&pointer_to_array[1][0]);
  l2=(int)(&pointer_to_array[1][1]);
  printf("\n Wartosc wskaznika  pointer_to_array[1][0]);
  printf("\n Wartosc wskaznika &pointer_to_array[1][0]=%p",\
                                   pointer_to_array[1][1]);
  printf("\n Roznica wskaznikow=%d",l2-l1);
  getch();
  /*Wyniki przebiegu fragmentu programu
  Drukowanie wskaznikow zwiazanych z  wskaznikiem
  array_pointer
  Wartosc wskaznika pointer_to_array=01BF4D8C
  Wartosc wskaznika pointer_to_array+1=01BF4D94
  Roznica wskaznikow=8
  Wartosc wskaznika &pointer_to_array[0][0]=01BF4D8C
  Wartosc wskaznika &pointer_to_array[0][1]=01BF4D90
  Roznica wskaznikow=4
  Wartosc wskaznika &pointer_to_array[1][0]=01BF4D94
  Wartosc wskaznika &pointer_to_array[1][0]=01BF4D98
  Roznica wskaznikow=4
```

```
      */
  free pointer_array;
  free pointer_to_array;
  system("pause");
}
```

12.3 Dynamiczna alokacja tablic trójwymiarowych

Jak to pokazano w przykładzie 15, aby zastosować wskaźnik do działań na tablicy trówymiarowej typu int , należy zdefiniować wskaźnik do tablicy dwuwymiarowej liczb typu **int(*)[ ][N]**, gdzie **N** oznacza pewną stałą.

Przykład 21 . Dynamiczna alokacja tablic trójwymiarowych.

```
#include <stdio.h>
#include <stdlib.h>
#define N    2
typedef int(*WSKTAB2D)[ ][N] ;
int main ( )
 {
  int (*wsk2D)[N][N];/* wskaznik do tablicy dwuwymiarowej*/
  WSKTAB2D  wsk2D1;/* alternatywna definicja wskaznika do
  tablicy dwuwymiarowej przy uzyciu wprowadzonej nazwy typu*/
  int i,j,k;

  wsk2D=(int(*)[N][N]) calloc(N*N*N,sizeof(int));
  //wsk2D= calloc(N*N*N,sizeof(int));
  wsk2D=( WSKTAB2D) calloc(N*N*N,sizeof(int));
/*wczytywanie tablicy trójwymiarowej przy uzyciu wskaznika*/
  for (i=0;i<N;i++)
    for (j=0;j<N;j++)
      for (k=0;k<N;k++)
       {
         printf("\n a[%d][%d][%d]=",i,j,k);
         scanf("%d",&wsk2D[i][j][k]);
         fflush(stdin);
       }
/*drukowanie tablicy trójwymiarowej przy uzyciu wskaznika*/

  for (i=0;i<N;i++)
   for (j=0;j<N;j++)
     for (k=0;k<N;k++)
       printf("\n b[%d][%d][%d]=%d",i,j,k,wsk2D[i][j][k]);
       printf("\n");

free(wsk2D);
system("pause");
}
```

13. Przekazywanie tablic do funkcji -standardy ANSI C i C99

Standard języka C wprowadzony w roku 1999 wprowadził istotne zmiany w stosunku do ANSI C. Nie jest on w pełni realizowany przez dostępne kompilatory, istotna część zmian wprowadzonych przez ten standard jest jednak uwzględniona w kompilatorze GNU C, który stanowi podstawę DevC++. W standardzie ANSI C rozmiar tablicy może być podany w postaci liczby naturalnej lub stałej definiowanej i musi być znany na etapie kompilacji. W C99 rozmiar tablicy w funkcji może być zwykłą zmienną lokalną lub parametrem funkcji (są to tzw. variable-length arrays),. Ma to szczególne znaczenie przy przekazywaniu tablic wielowymiarowych do funkcji, gdyż wszystkie rozmiary przekazywanej tablicy mogą być zmiennymi, a nie stałymi (poza pierwszym rozmiarem) jak to ma miejsce w ANSI C.

Poniżej w programie umieszczono przykładowe definicje tablic oraz ich przekazywanie do funkcji w obydwu standardach.

Przykład 22. Ilustracja metod przekazywania do funkcji tablic jedno- i dwuwymiarowych w standardach ANSI C i C99.

```
#include <stdio.h>
#include <stdlib.h>
#define   N    2
#define   N1   2
#define   N2    2
#define ROZMIAR 100
typedef int( *TYPWSK2D)[2];
typedef int(*WSKTAB1D)[ROZMIAR];
typedef int (*TYPWSK3D)[3];
void  wczyt1D( int x[ ], int n);
void  druk1D( int x[ ], int n);
void  wczyt2D( int x[ ][N2], int n);
void  druk2D( int x[ ][N2], int n);
void  wczyt2DC99(int n, int m, int x[n][m]);
void   druk2DC99(int n, int m, int x[n][m]);
```

```
int main () {
  int a[N], b[N1][N2];
  int n=2,m=2;
  int aa[n];// styl C99
  int bb[n][m];// styl C99
  int (*wskD2x2)[2];
  int (*wskD3x3)[3];
  printf("\nWczytywanie i drukowanie tablicy 1D-styl ANSI C");
  wczyt1D(a,N);
  druk1D(a,N);
  system("pause");
  wczyt1D(aa,n);
  druk1D( aa,n);
  printf("\nWczytywanie i drukowanie tablicy 2D-styl ANSI C");
  wczyt2D( b, N1);
  druk2D( b,N1);
  system("pause");
  printf("\n Wczytywanie i drukowanie tablicy 2x2- styl C99");
  //przekazywana jest tablica 2 x 2
  wczyt2DC99(n,m, bb);
  druk2DC99( n,m,bb);
  //przekazywana jest tablica 3 x 3
  printf("\n Wczytywanie i drukowanie tablicy 3x3- styl C99");
  wczyt2DC99(3,3, bb);
  druk2DC99( 3,3,bb);
  n=m=2;
  // dynamiczna alokacja tablicy i rzutowanie wskaźnika
  // na typ TYPWSK2D
  wskD2x2= (TYPWSK2D)calloc(n*m,sizeof(int));
  printf("\n Wczytywanie i drukowanie tablicy dynamicznej\
               2x2- styl C99");
  wczyt2DC99(n,m, wskD2x2);
  druk2DC99( n,m, wskD2x2);
  n=m=3;
```

```
  printf("\n Wczytywanie i drukowanie tablicy dynamicznej\
                3x3- styl C99");
  wskD3x3= (TYPWSK3D)calloc(n*m, sizeof(int));
  wczyt2DC99(n,m, wskD3x3);
  druk2DC99( n,m, wskD3x3);
  free(wskD2x2);
  free(wskD3x3);
  system("pause");
}
void  wczyt1D( int x[ ],int n)
  { int i;
    for (i=0;i<n;i++)
       { printf("\n Element [%d]=",i);
          scanf("%d",&x[i]); }
  }

void  druk1D( int x[ ],int n)
  {
    int i;
    for (i=0;i<n;i++)
       printf("\n element[%d]=%d",i,x[i]);
   }

void  wczyt2D( int x[ ][N2], int n)
  { int i,j;
        for (i=0;i<n;i++)
           for (j=0;j<N2;j++)
             { printf("\n Element [%d][%d]=",i,j);
                scanf("%d",&x[i][j]); }
   }

void  wczyt2DC99( int n, int m, int x[n][m])
   // ilosc kolumn nie musi być stałą, może być podawana
   // jako parametr
```

```
   {  int i,j;
      for (i=0;i<n;i++)
        for (j=0;j<m;j++)
          { printf("\n Element [%d][%d]=",i,j);
            scanf("%d",&x[i][j]);}
    }

void  druk2D( int x[ ][N2], int n)
  { int i,j;
        for (i=0;i<n;i++)
          { for (j=0;j<N2;j++)
                printf("  %d", x[i][j]);
            printf("\n"); }
   }

void  druk2DC99( int n, int m, int x[n][m])
  { int i,j;
      for (i=0;i<n;i++)
          { for (j=0;j<m;j++)
                printf("  %d", x[i][j]);
            printf("\n");}
  }
```

14. Dynamiczna alokacja tablic w C++

W języku C++ wprowadzono operatory **new** i **delete** związane z dynamiczną alokacją. Operator new służy do tworzenia obiektów dynamicznych, a operator delete do usuwania obiektów dynamicznych (dealokacji pamięci). Składnia stosowana dla operatora new ma postać nastepującą:

nazwa_wskaźnika=**new** typ;

lub

nazwa_wskaźnika=**new** typ(wartosc_poczatkowa);

Powyższe konstrukcje mogą też wystąpić przy inicjalizacji.

Przykład 23. Alokacja zmiennych dynamicznych przy użyciu operatora new.

```
int *px=new int;  // inicjalizacja wskaźnika  px  adresem
                  //zmiennej dynamicznej bez inicjalizacji
```

```
int *py=new int (10); // podobnie jak powyżej, ale
                      // inicjalizacją zmiennej dynamicznej
                      // wartością 10
double *pv,*pz;
pz=new double;
pv=new double(30);
```

Składnia operatora **delete** ma formę następującą:

```
delete nazwa_wskaźnika
```

Jeżeli próba przydzielenia pamięci jest udana i zostanie przydzielona pamięć, operator new zwraca wskaźnik do przydzielonego obszaru pamięci (wskaźnik do piewszego bajtu). Przydzielony obszar nie ma nazwy, która pełniłaby rolę l-wartości, stąd role tę pełni wyrażenie *nazwa_wskaźnika, np. ***px**.

Operatory **new** i **delete** mogą być też stosowane do dynamicznej alokacji tablic. Alokacja taka ma postać

```
nazwa_wskaźnika=new typ[rozmiar];
```

nazwa_wskaźnika jest nazwą zmiennej wskaźnikowej, **typ** jest typem elementów alokowanej tablicy, a **rozmiar** oznacza liczbę elementów.
Możliwa jest także alokacja przy inicjalizacji w następującej formie:

```
typ   nazwa_wskaźnika=new typ[rozmiar];
```

Dealokacja przy użyciu operatora delete ma postać

```
delete[ ]nazwa_wskaźnika;
```

Przykład 23. Użycie operatorów **new** i **delete**.

```
# include <iostream.h>

int main ()
{
int *px;
double  *ptab;
px= new int;  // alokacja pamięci dla zmiennej
ptab=new double[5];// alokacja pamięci dla tablicy
*px=5;
cout<<" Wartosc zmiennej="<<*px<<endl;
for (int i=0;i <5;i++)
  {
    cout<<" Podaj element["<<i<<"]=";
    cin>>*ptab++;
    cout<<endl;
```

```
    fflush(stdin);
   }

ptab-=5;// przesuniecie wskaźnika na początek alokowanego
        //obszaru
for (int i=0;i<5;i++)
        cout<<endl<<" Element["<<i<<"]="<<*ptab++;
ptab-=5;
delete px;
delete [ ]ptab;
cin. get();
return 0;
}
```

Przykład 24. Użycie operatora **new** do alokacji tablicy dwuwymiarowej.

```
#include <iostream>
using namespace std;
typedef int( *TYPWSK2D)[2];// definicja wskaznika do tablicy
// do tablicy dwuwumiarowej typu int o dwóch kolumnach
typedef int*  TYPWSK1D;
#define N2  2
void  wczyt2D( int x[ ][N2], int n);
void  druk2D( int x[ ][N2], int n);
int main()
{
  int n=2;
  TYPWSK2D wsk2D;
  wsk2D=(TYPWSK2D) new int[n*n];
   //wsk2D= new int[4];

  wczyt2D(wsk2D,n);
  druk2D ( wsk2D,n);
  cout<<"\n";
  delete [] wsk2D;
  system("pause");
  return 0;
}
void  wczyt2D( int x[ ][N2], int n)
  { int i,j;
        for (i=0;i<n;i++)
           for (j=0;j<N2;j++)
             {
                cout<<"\n Element["<<i<<"]["<<j<<"]=";
                cin>>x[i][j];
             }
   }

 void  druk2D( int x[ ][N2], int n)
  { int i,j;
        for (i=0;i<n;i++)
```

```
        { for (j=0;j<N2;j++)
              cout<< x[i][j];
          cout<<"\n";
        }
}
```

Literatura

[1] Reek K.A. ***Język C. Wskaźniki. Vademecum profesjonalisty,*** **Helion, Gliwice, 2003.**

[2] Porębski W. ***Język C++. Standard ISO*****. PWN, W-wa, 2008.**

[3] Gręboszj., ***Symfonia C++. Programowanie w języku C++ orientowane obiektowo*****, EDITION2000, Kraków, 2008.**

[4]Kernighan B.W., Ritchie D.M, ***Język ANSI C*****, WNT, W-wa, 1997.**