Wskaźniki i referencje

- *Wskaźnik* (ang. *pointer*) to zmienna specjalnego typu, w której można przechowywać adres pamięci przypisanej obiektowi określonego typu (np. zmiennej typu `int`).
- *Referencja* (odniesienie, ang. *reference*) to inna nazwa obiektu (zmiennej). Uwaga: referencja występuje w języku C++, w C nie ma referencji.

Podstawowe zastosowania wskaźników

- Ułatwiają współdzielenie danych pomiędzy różnymi częściami programu - jeśli prześlemy adres zmiennej do wywoływanej funkcji, możemy zmienić wartość tej zmiennej w funkcji i zmiana będzie widziana również w funkcji wywołującej
- Pozwalają w sposób zwarty odwoływać się do dużych struktur danych - bez względu na to, jak są duże znajdują się gdzieś w pamięci i mają adres; można na przykład przesyłać ich adres do funkcji zamiast kopiowania całej struktury – zwiększamy w ten sposób efektywność programu.
- Pozwalają rezerwować pamięć dynamicznie podczas wykonywania programu.
- Pozwalają przechowywać połączenia pomiędzy elementami złożonych struktur danych takich jak na przykład listy.

Podstawowe zastosowania referencji

- Ułatwiają współdzielenie danych pomiędzy różnymi częściami programu - jeśli zostaną użyte jako argumenty formalne funkcji, można wtedy zmieniać w funkcji wartości przekazywanych do niej obiektów.

Wskaźniki

Bity, bajty, słowa, adres

- Najmniejszą jednostką pamięci jest *bit*. Może on przyjmować dwa stany: 0 i 1.
- Bity są łączone w *bajty* (ang. *byte*). Każdy bajt zawiera tyle bitów, ile jest potrzebnych do przechowania jednego znaku. Obecnie najczęściej jest to 8 bitów. W językach C++/C bajt jest reprezentowany za pomocą typu `char`.
- Bajty są łączone w *słowa* (ang. *word*). Najczęściej słowo zawiera dwa, cztery lub osiem bajtów. Typ `int` jest tak definiowany, aby jego rozmiar odpowiadał jednemu słowu.
- Każdy bajt jest identyfikowany za pomocą *adresu* (ang. *address*). Adres jest to liczba całkowita.
- Jeden znak zajmuje jeden bajt. Adresem znaku jest adres zajmowanego przez niego bajtu. Jednak zmienne innych typów zajmują więcej bajtów. Adresem takiej zmiennej będzie adres pierwszego bajtu obszaru zajmowanego przez zmienną.

Deklarowanie zmiennych wskaźnikowych

- Postać deklaracji zmiennej wskaźnikowej:
  *typ_obiektu *nazwa_zmiennej_wskaźnikowej*;
- Mówimy, że *wskaźnik wskazuje na obiekt określonego typu*.
- Typ obiektu na który wskaźnik może wskazywać nazywany jest *typem podstawowym wskaźnika*. Jest to informacja potrzebna kompilatorowi, aby wiedział jak interpretować obszar pamięci znajdujący się pod konkretnym adresem i jaki jest rozmiar tego obszaru.
- Przykład:

```
char *w1, *w2; // wskaźnik do zmiennej typu char
int *wd;       // wskaźnik do zmiennej typu int
double *x, y;  // x jest wskaźnikiem, y zwykłą zmienną
```

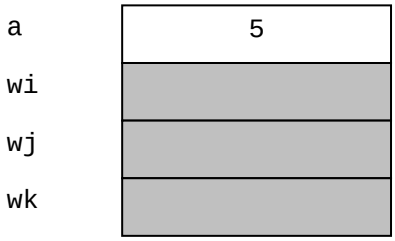
- Po zadeklarowaniu wartość wskaźnika jest nieokreślona.

Podstawowe działania na wskaźnikach

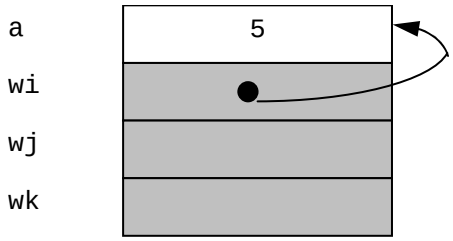
- Wskaźnik służy do przechowywania adresu obiektu. Adres obiektu jest pobierany za pomocą operatora *adresu* (ang. *address-of operator*) - znaku & .
- Wskaźnik pozwala na *pośredni* dostęp do wartości zmiennej. Aby odczytać lub zmienić wartość zmiennej przechowywanej pod adresem zawartym we wskaźniku należy posłużyć się operatorem *wyłuskania* (ang. *dereference operator*) (inna nazwa to operator *adresowania pośredniego* - ang. *indirection operator*). Jest nim gwiazdka *.
- Przykład:

```
// deklaracja zmiennych zmiennej
int a, *wi, *wj, *wk;
```

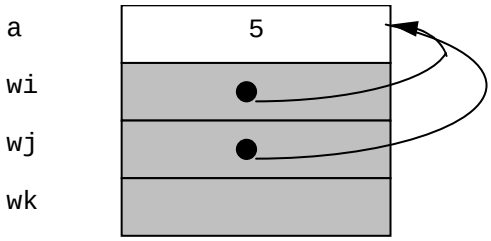

```
// przypisanie wartości
a=5;
```



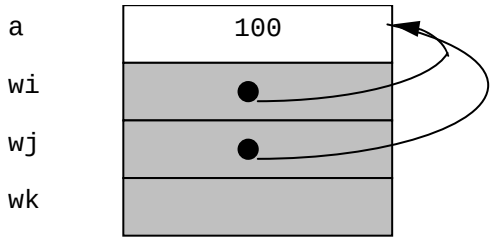

```
// przypisanie adresu
wi = &a;
```



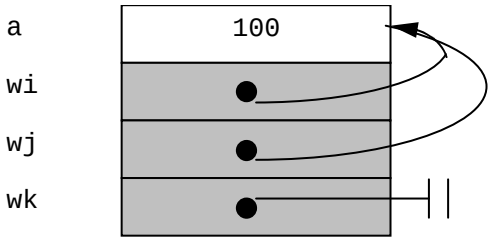

```
// przypisanie adresu
wj = wi;
```




```
// przypisanie nowej wartości
*wj = 100;
```




```
// wk nie wskazuje na żaden obiekt
wk = 0;
```



- Uwaga:

```
wj  = wi;   // przypisanie wskaźników: obydwa wskazują na to samo miejsce

*wj = 10;   // przypisanie wartości
a   = *wj;  // przypisanie wartości
*wj = *wi;  // przypisanie wartości
```

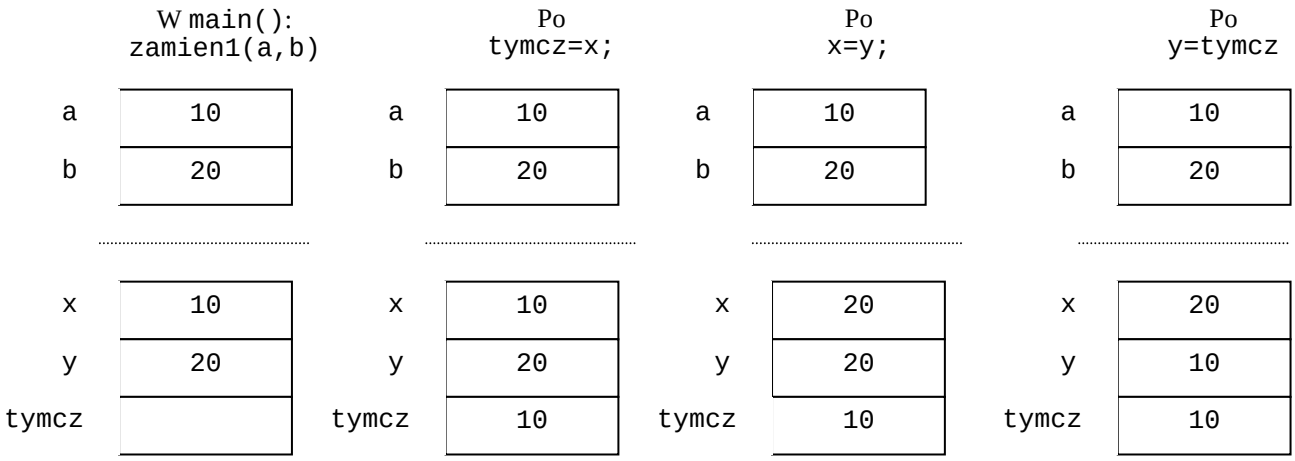
- Żaden obiekt nie będzie umieszczony pod adresem zero. Przyjęto więc, że przypisanie 0 wskaźnikowi oznacza, że nie pokazuje on na żaden obiekt.
- Dla zwiększenia czytelności programu, często definiowana jest stała (szczególnie w języku C) reprezentująca zerowy wskaźnik. Nosi ona nazwę `NULL` i jest definiowana w wielu plikach nagłówkowych, np. `stdlib.h`, `string.h`, `stddef.h`, `stdio.h`. Można również zdefiniować ją samemu za pomocą instrukcji:

```
const int NULL=0;
```

Przekazywanie prostych zmiennych do funkcji

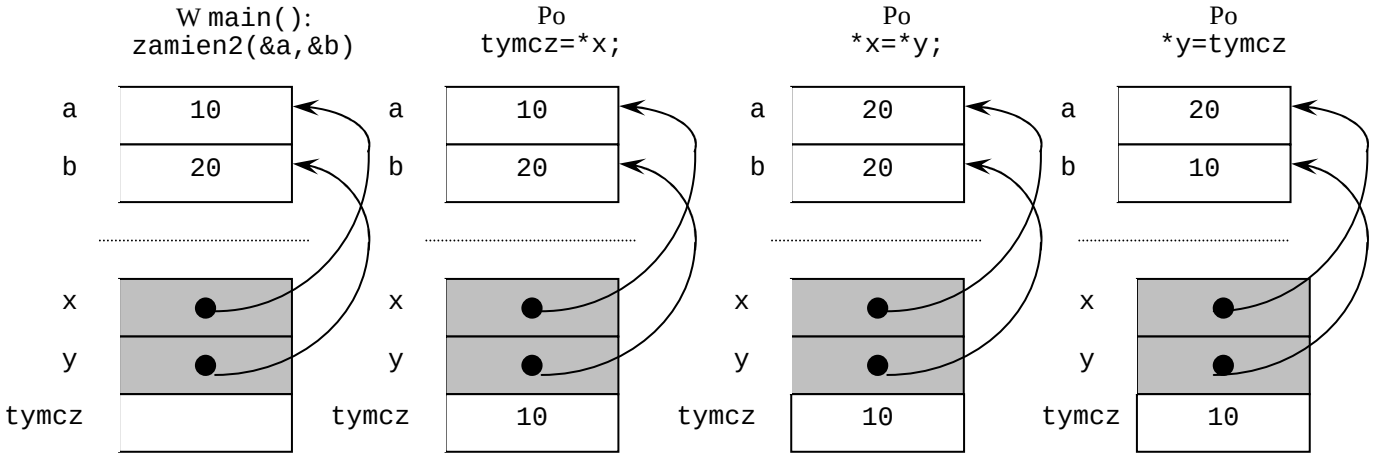
- Przykład: Chcemy zamieniać w funkcji wartości dwóch zmiennych.
- *Nieprawidłowa* funkcja `zamien1()`

```
void zamien1(int x, int y) {
  int tymcz;
  tymcz=x; x=y; y=tymcz;
}
```



- *Prawidłowa* funkcja `zamien2()`

```
void zamien2(int *x, int *y) {
  int tymcz;
  tymcz=*x; *x=*y; *y=tymcz;
}
```




```
// użycie funkcji zamien2()
 int main() {
   int i=10, j=20;
   cout << "Przed zamiana \ti=" << i << "\tj=" << j << endl;
   zamien2(&i,&j);
   cout << "Po zamianie \ti=" << i << "\tj=" << j << endl;
   return 0;
 }
```

- Przykład: rozwiązywanie równania kwadratowego

```
#include <iostream.h>
#include <math.h>

bool Rozwiaz(double a, double b, double c, double *x1, double *x2);
void Drukuj(double x1, double x2);

int main()
{
  double a,b,c,x1,x2;
  cout << "Podaj wspolczynniki rownania kwadratowego ax**2+bx+c=0" <<
endl;
  cout << "a b c: ";
  cin >> a >> b >> c;
  if (Rozwiaz(a,b,c,&x1,&x2))
  {
    Drukuj(x1,x2);
    return 0;
  }
  else
    return 1;
}

bool Rozwiaz(double a, double b, double c, double *x1, double *x2)
{
  double delta;
  if (a==0) {
    cout << "Wspolczynnik a musi byc rozny od 0" << endl;
    return false;
  }
  delta = b*b - 4*a*c;
  if (delta < 0) {
    cout << "Brak pierwiastkow rzeczywistych" << endl;
    return false;
  }
  delta=sqrt(delta);
  *x1=(-b + delta)/(2*a);
  *x2=(-b - delta)/(2*a);
  return true;
}

void Drukuj(double x1, double x2)
{
  if (x1==x2)
    cout << "Podwojny pierwiastek o wartosci " << x1 << endl;
  else
    cout << "Pierwiastki to " << x1 <<" i " << x2 << endl;
}
```

Wartości tych zmiennych są wyznaczane w funkcji

Te zmienne są tylko czytane w funkcji

Tablice i wskaźniki

- Załóżmy, że mamy tablicę:

```
int t[]={1,2,2,5,8};
```

- Związek między tablicą a wskaźnikiem: nazwa tablicy jest adresem jego pierwszego elementu:

```
t  ≡ &t[0]          *t ≡ t[0];
```

- Równoważne zapisy:

	Adresy		Wartości	
pierwszy element	`t;`	`&t[0]`	`*t`	`t[0]`
drugi element	`t+1`	`&t[1]`	`*(t+1)`	`t[1]`

Arytmetyka wskaźnikowa

- Na wskaźnikach można wykonywać tylko dwa działania arytmetyczne: dodawanie i odejmowanie.
- Po każdym *zwiększeniu* zmiennej wskaźnikowej wskazuje ona na obszar pamięci zajmowany przez *następny obiekt jej typu bazowego*. Czyli:

```
int x;
int *wsk;
wsk=&x;
wsk++;
```

  oznacza, że `wsk` zawierać będzie adres: `&x + 1*sizeof(int)`

- Przykłady:

```
int x;
int *p1; // typ int zajmuje 2 bajty
p1=&x;   // załóżmy, że p1 ma wartość 1000
p1++;    // wartość p1 to 1002 - zwiększenie o 2
p1=p1+9; // wartość p1 to 1002 + 9*2

char znak;
char *p; // typ char zajmuje 1 bajt
p=&znak; // załóżmy, że p ma wartość 2000
p++;     // wartość p to 2001
p=p+9;   // wartość p to 2001 + 9*1
```

- Po każdym *zmniejszeniu* zmiennej wskaźnikowej wskazuje ona na obszar pamięci zajmowany przez poprzedni obiekt jej typu bazowego.
- W przypadku tablic oznacza to następujące zależności:

```
int a[10], *wsk;
wsk=a;           // wsk zawiera adres elementu a[0]
wsk=wsk+1;       // wsk zawiera adres elementu a[1]:
wsk++;           // wsk zawiera adres elementu a[2]
```

  Liczbowo oznacza to:

  `wsk+1` jest równe `&a[0]+1*sizeof(int)`
  `wsk+9` jest równe `&a[0]+9*sizeof(int)`

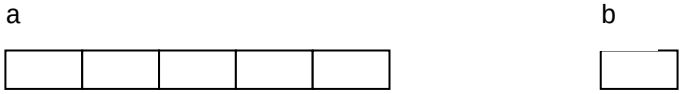
- Inna operacja to odejmowanie jednego wskaźnika od drugiego, kiedy oba wskaźniki wskazują na obiekt tego samego typu, na przykład tablicę. Wynikiem tej operacji jest liczba elementów typu bazowego oddzielających jeden wskaźnik od drugiego.

Nazwa tablicy

- Załóżmy, że mamy deklaracje:

```
int a[5];
int *b;
```

- Można je przedstawić następująco:



- Deklaracja tablicy oznacza przydzielenie miejsca w pamięci na określoną liczbę elementów, następnie utworzenie nazwy tablicy, która jest stałą wskazującą początek obszaru tablicy.
- Deklaracja wskaźnika oznacza przedzielenie miejsca w pamięci tylko na wskaźnik. Nie jest on zainicjalizowany.

- Przykłady:

```
int a[10], b[10], *wsk;
wsk=a;          // OK
a=a+1;          // błędne, tablica jest wskaźnikiem stałym
wsk=a+1;        // OK
b=a;            // błędne, tablica jest wskaźnikiem stałym
wsk=b;          // OK
wsk=&b[0];      // OK
```

Priorytet i wiązanie

- Operatory adresu (&) i wyłuskania (*) to operatory jednoargumentowe.
- Mają łączność prawostronną.
- W tabeli priorytetów znajdują się na nad wszystkimi operatorami arytmetycznymi.
- Przykład
  Załóżmy, że dane są deklaracje:

```
int a[10];
int *wsk=a+2;
```

Zapis z użyciem wskaźnika	Zapis z użyciem indeksu
`wsk`	adres `&a[2]` - `a+2` w inicjalizacji)
`*wsk`	wartość `a[2]` - `*(a+2)`
`wsk[0]`	wartość `a[2]` - `*(wsk+0)`
`wsk+6`	`&a[8]`
`*wsk+6`	`a[2]+6` - dodawanie ma niższy priorytet
`*(wsk+6)`	`a[8]`
`wsk[6]`	`a[8]`
`wsk[-1]`	`a[1]`
`wsk[9]`	adres spoza zakresu tablicy

- Operatory adresu (&) i wyłuskania (*) mają taki sam priorytet co operatory `++` i `--` .

`wsk++`	zwiększ wskaźnik o 1 (wskaż następny adres)
`*(wsk++)`	użyj zmiennej wskazywanej przez `wsk`; zwiększ wskaźnik o 1
`*wsk++`	to samo co `*wsk++`, bo obowiązuje wiązanie prawostronne
`(*wsk)++`	zwiększ o 1 to, na co wskaźnik `wsk` wskazuje
`*++wsk`	zwiększ o 1 wartość wskaźnika, użyj wskazywanej zmiennej

```
#include <iostream.h>
#include <stdio.h>
int main() {
  int a[5]={1,5,10,15,20}
  int z;
  int *wsk;
  wsk=a;
  cout << *wsk << endl;   /* drukuj a[0]: 1 */

  *wsk += 1;              /* zwiększ a[0] o 1 */
  cout << *wsk << endl;   /* drukuj a[0]: 2 */

  ++*wsk;                 /* zwiększ a[0] o 1 */
  cout << *wsk << endl;   /* drukuj a[0]: 3 */

  z=*wsk++;               /* przypisz a[0], zwiększ wskaźnik o 1 */
  cout << z << endl;      /* drukuj z: 3 */

  (*wsk)++;               /* zwiększ a[1] o 1 */
  cout << *wsk << endl;   /* drukuj a[1]: 6 */
  return 0;
}
```

- Co robi poniższy fragment programu?

```
int a[10], b[10]={1,2,3};
int *wsk1, *wsk2;
wsk1=a;
wsk2=b;
int i=0;
while (i++ < 10)
  *wsk2++ = *wsk1++;
```

Przekazywanie tablic do funkcji

- W językach C++/C tablica *zawsze* jest przekazywana do funkcji jako wskaźnik do jej pierwszego elementu.
- W deklaracji funkcji należy dla tablicy przewidzieć zmienną odpowiedniego typu do odebrania adresu tablicy:

```
void czysc(int *t, int l_elementow);
```

- Jednakże w *kontekście prototypu funkcji* (*nagłówka*) można również napisać:

```
void czysc(int t[], int l_elementow);
void czysc(int t[10], int l_elementow);
```

- Należy pamiętać, że rozmiar tablicy podany jako część argumentu formalnego czyli `[10]` nie jest brany przez kompilator pod uwagę. Dla kompilatora ważny jest tylko typ elementów oraz informacja, że jest to tablica. *Program musi uwzględnić jakiś sposób przekazywania informacji o rozmiarze tablicy do funkcji*.

Różne sposoby przekazywania do funkcji informacji o rozmiarze tablicy:

A. Stała globalna MAX oznaczająca rozmiar tablicy

```
#include <iostream.h>

const int MAX=10;

void czysc(int t[]) {
  int i;
  for (i=0; i<MAX; ++i)
    t[i]=0;
}

int main() {
  int a[MAX];
  czysc(a);
  // to samo:
  // czysc(&a[0]);
  return 0;
}
```

```
#include <iostream.h>

const int MAX=10;

void czysc(int *t) {
  int i;
  for (i=0; i<MAX; ++i)
    *(t+i)=0;
}

int main() {
  int a[MAX];
  czysc(a);
  // to samo:
  // czysc(&a[0]);
  return 0;
}
```

B. Jednym z parametrów funkcji `suma()` jest rozmiar tablicy.

```
#include <iostream.h>
int suma(int a[], int n);

int main()
{
  int t[5]={0,3,4,6,7};
  cout << suma(t,5) << endl;
  return 0;
}

int suma(int a[], int n)
{
  int i,suma=0;
  for (i=0;i<n;i++)
    suma += a[i];
  return suma;
}
```

```
#include <iostream.h>
int suma(int *wa, int n);

int main()
{
  int t[5]={0,3,4,6,7};
  cout << suma(t,5) << endl;
  return 0;
}

int suma(int *wa, int n)
{
  int i,suma=0;
  for (i=0;i<n;i++)
    suma += *wa++; // czyli *(wa++)
  return suma;
}
```

C. Do funkcji przekazywany jest adres końca tablicy.

```
#include <iostream.h>
void drukuj(int *poczatek, int *koniec);

int main()
{
  int t[5]={0,3,4,6,7};
  drukuj(t,t+5);
  return 0;
}

void drukuj(int *poczatek, int *koniec)
{
  while (poczatek != koniec)   {
    cout << *poczatek << ' ';
    ++poczatek;
  }
}
```

Komentarz: adres końcowy wskazuje pierwszy adres *za* ostatnim adresem tablicy. Pętla jest powtarzana dopóty, dopóki zmienna `początek` nie przyjmie wartości poza obszarem tablicy.

Wskaźniki i indeksy

- Skoro do wartości elementów tablicy można odwoływać się zarówno za pomocą indeksów jak i wskaźników, czy są zalecenia związane z wyborem zapisu?
  - Zapis z indeksem jest na ogół bardziej czytelny, ale często program w ten sposób napisany może być mniej efektywny.
  - Można powiedzieć, że zapis z użyciem indeksów *nigdy* nie jest bardziej efektywny niż zapis ze wskaźnikami, zaś zapis ze wskaźnikami *czasem* jest bardziej efektywny.
- Przykład, kiedy użycie wskaźnika jest bardziej efektywne - przesuwanie się po kolejnych elementach tablicy:

```
// Wersja A
int tablica[10], i;
for (i=0; i<10;, i++)
  tablica[i]=0;
```

Komentarz: aby obliczyć indeks kompilator musi wstawić instrukcje, które wartość indeksu `i` pomnożą przez rozmiar typu `int` (np. 2 lub 4). Powtarzane jest to w każdym wykonaniu pętli.

```
// Wersja B
int tablica[10], *wsk;
for (wsk=tablica; wsk<tablica+10; wsk++)
  *wsk=0;
```

Komentarz: kompilator skaluje wartość 1 dodawaną do wskaźnika mnożąc ją przez rozmiar typu `int`, ale jest to wykonywane tylko raz, podczas kompilacji.

- Przykład, kiedy nie ma różnicy między użyciem wskaźnika i indeksu:

```
i=ustal_indeks();
tablica[i]=0;

i=ustal_indeks();
*(tablica+i)=0;
```

- *Posługiwanie się wskaźnikami wcale nie oznacza, że program będzie bardziej efektywny*. Napisanie wersji, która będzie bardziej efektywna nie zawsze jest proste.
- Przykład: funkcja kopiowania dwóch tablic

```
const int MAX=100;
int x[MAX], y[MAX];
int i;
int *wsk1, *wsk2;
```

```
// Wersja pierwsza
void kopiuj1()
{
  for (i=0; i<MAX; i++)
    x[i]=y[i];
}
```

```
// Wersja optymalizowana pod kątem
// efektywnosci
void kopiuj2()
{
  register int *wsk1, *wsk2;
  for (wsk1=x, wsk2=y; wsk1<&x[MAX];)
    *wsk1++ = *wsk2++;
}
```

Deklarowanie argumentów typu tablicy

```
int strlen(char *string);
int strlen(char string[]);
```

- Jest to jedyne miejsce, w którym obydwie deklaracje są równoważne.
- Dla kompilatora *każdy* z zapisów oznacza wskaźnik. Jeżeli zastosowalibyśmy operator `sizeof(string)`, otrzymalibyśmy długość zmiennej wskaźnikowej, a nie tablicy.

- Przekazywanie do funkcji adresu tablicy oznacza, że w funkcji nie potrafimy określić długości tablicy. Rozmiar tablicy *musi* być przekazany w jakiś sposób do funkcji.

Wskaźniki i tablice wielowymiarowe

- Tablica *jednowymiarowa* `int a[5]`:

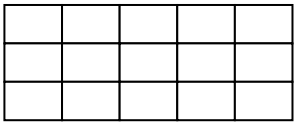
  `a` `&a[0]` są równoważne

  Dostęp do elementu a_2 tablicy:

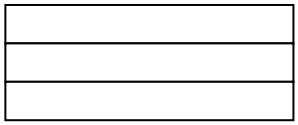
```
x = a[2];
x = *(a+2);
```

- Tablica *wielowymiarowa*:

```
int a[3][5];  /* tablica 3 tablic o 5 wartościach typu int */
```



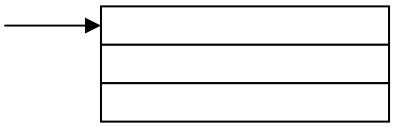
jest równoważne



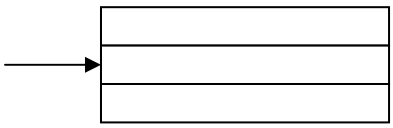
Zapisy:
`a` `&a[0]` `&a[0][0]` są równoważne

- Przykład: Jak zapisać a[1][3] za pomocą wskaźników?

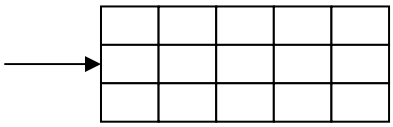
`a` jest wskaźnikiem do tablicy o 5 wartościach `int`:



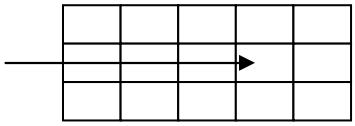
`a+1` jest wskaźnikiem do tablicy o 5 wartościach `int`:



`*(a+1)` jest wskaźnikiem do pierwszego elementu z drugiej tablicy



`*(a+1)+3` jest wskaźnikiem do czwartego elementu z drugiej tablicy



`*(*a+1)+3)` jest wartością `a[1][3]`

Dostęp do elementu $a_{1,3}$ tablicy:

```
x = a[1][3];
x = *((*(a+1)+3));
x = *(a[1])+3;
```

Wskaźnik do tablicy wielowymiarowej

```
int wektor[10], *wsk1=wektor;
int a[3][5], wsk2=(*a)[5]; // dodanie 1 oznacza przesunięcie o wiersz
int b[3][5], wsk3=&b[0][0]; // dodanie 1 oznacza przesunięcie o element
int c[3][5], wsk4=c[0]; // dodanie 1 oznacza przesunięcie o element
```

Przekazywanie tablicy wielowymiarowej do funkcji

- Przekazywany jest wskaźnik do pierwszego elementu tablicy.
- W funkcji otrzymującej tablicę dwuwymiarową trzeba określić przynajmniej rozmiar wyznaczający liczbę kolumn. Rozmiar wyznaczający liczbę wierszy można również podać, ale nie jest to konieczne.
- Przetwarzanie w funkcji tablic dwuwymiarowych:
  - funkcja jest dostosowana do tablicy jednowymiarowej – przekazujemy jej po jednej podtablicy - wierszu
  - funkcja jest dostosowana do tablicy jednowymiarowej – przekazujemy jej całą tablicę traktowaną jako tablicę jednowymiarową
  - funkcja jest dostosowana do tablicy dwuwymiarowej

A. Funkcja jest dostosowana do tablicy jednowymiarowej, przekazujemy jej po jednej podtablicy - wierszu

```
#include <iostream.h>
#include <iomanip.h>

void podw(int t[], int n);

int main() {
  int a[3][4] = {
                  {2,4,5,8},
                  {3,5,6,9},
                  {12,10,8,6}
                 };
  int i, j;
  for (i = 0; i < 3 ; i++)
    podw(a[i], 4);
  for (i = 0; i < 3; i++) {
    for (j = 0; j < 4; j++)
      cout << setw(4) << a[i][j] ;
    cout << endl;
  }
  return 0;
}

void podw(int t[], int n)
{
  int i;
  for (i = 0; i < n; i++)
    t[i] *= 2;
}
```

```
#include <iostream.h>
#include <iomanip.h>

void podw(int *t, int n);

int main() {
  int a[3][4] = {
                  {2,4,5,8},
                  {3,5,6,9},
                  {12,10,8,6}
                 };
  int i, j;
  for (i = 0; i < 3 ; i++)
     podw(a[i], 4);
  for (i = 0; i < 3; i++) {
    for (j = 0; j < 4; j++)
      cout << setw(4) << a[i][j] ;
    cout << endl;
  }
  return 0;
}

void podw(int *t, int n)
{
  int i;
  for (i = 0; i < n; i++)
    *(t+i) *= 2;
}
```

B. Funkcja jest dostosowana do tablicy jednowymiarowej – przekazujemy jej całą tablicę traktowaną jako tablicę jednowymiarową

```
#include <iostream.h>
#include <iomanip.h>
void podw(int t[], int n);
int main()
{
  int a[3][4]={
                {2,4,5,8},
                {3,5,6,9},
                {12,10,8,6}
               };
  int i, j;
  podw(a[0], 3*4);
  for (i = 0; i < 3; i++) {
    for (j = 0; j < 4; j++)
      cout << setw(4) << a[i][j] ;
    cout << endl;
  }
  return 0;
}

void podw(int t[], int n)
{
  int i;
  for (i = 0; i < n; i++)
    t[i] *= 2;
}
```

```
#include <iostream.h>
#include <iomanip.h>
void podw(int *t, int n);
int main()
{
  int a[3][4]={
                {2,4,5,8},
                {3,5,6,9},
                {12,10,8,6}
               };
  int i, j;
  podw(a[0], 3*4);
  for (i = 0; i < 3; i++) {
    for (j = 0; j < 4; j++)
      cout << setw(4) << a[i][j] ;
    cout << endl;
  }
  return 0;
}

void podw(int *t, int n)
{
  int i;
  for (i = 0; i < n; i++)
    *(t+i) *= 2;
}
```

C. Funkcja jest dostosowana do tablicy dwuwymiarowej

```
#include <iostream.h>
#include <iomanip.h>
void podw(int t[][4], int m, int n);
int main()
{
  int a[3][4]={
                {2,4,5,8},
                {3,5,6,9},
                {12,10,8,6}
              };
  int i, j;
  podw(a, 3,4);
  for (i = 0; i < 3; i++) {
    for (j = 0; j < 4; j++)
      cout << setw(4) << a[i][j] ;
    cout << endl;
  }
  return 0;
}

void podw(int t[][4], int m, int n)
{
  for (int i = 0; i < m; i++)
    for (int j=0; j<n; j++)
      t[i][j] *= 2;
}
```

```
#include <iostream.h>
#include <iomanip.h>
void podw(int (*t)[4], int m, int n);
int main()
{
  int a[3][4]={
                {2,4,5,8},
                {3,5,6,9},
                {12,10,8,6}
              };
  int i, j;
  podw(a, 3,4);
  for (i = 0; i < 3; i++) {
    for (j = 0; j < 4; j++)
      cout << setw(4) << a[i][j] ;
    cout << endl;
  }
  return 0;
}

void podw(int (*t)[4], int m, int n )
{
  for (int i = 0; i < m; i++)
    for (int j=0; j<n; j++)
      t[i][j] *= 2;
}
```

- Przykład

```
#include <iostream.h>
void drukuj_wiersz(int i, int rozmiar_wiersza, int *tablica);

int main(void)
{
  int a[3][4],i,j;

/* wpisanie liczb do tablicy */
  for (i=0; i<3; ++i)
    for (j=0; j<4; ++j)
      a[i][j]=1;

/* wyświetlenie liczb w tablicy */
  for (i=0; i<3; ++i)
    drukuj_wiersz(i,3,&a[0][0]);
  return 0;
}

void drukuj_wiersz(int i, int rozmiar_wiersza, int *tablica)
{
  int j;
  tablica=tablica+i*rozmiar_wiersza;
  for (j=0; j<rozmiar_wiersza;++j)
    cout << *(tablica+j);
  cout << endl;
}
```