8 Wskaźniki i referencje

- *Wskaźnik* (ang. *pointer*) to zmienna specjalnego typu, w której można przechowywać adres pamięci przypisanej obiektowi określonego typu (np. zmiennej typu `int`, zmiennej typu `double`, zmiennej typu `char`).

- Wskaźniki występują *zarówno* w języku C++ jak i w C.

- *Referencja* (odniesienie, ang. *reference*) to *inna nazwa* obiektu (zmiennej).

- 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 - zamiast kopiowania całej struktury, do funkcji przesyłany jest jej adres – zwiększamy w ten sposób efektywność programu.

- Pozwalają rezerwować pamięć dynamicznie czyli podczas wykonywania programu wtedy, kiedy z góry nie wiemy ile pamięci będzie potrzeba na przechowywanie danych programu.

- Pozwalają przechowywać połączenia pomiędzy elementami złożonych struktur danych, takich jak na przykład listy dowiązaniowe, drzewa.

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 i zmiany te będą widoczne po powrocie z funkcji.

8.1 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 w danym komputerze.
- Każdy bajt jest identyfikowany za pomocą *adresu* (ang. *address*). Adres jest to liczba całkowita wskazująca położenie bajtu w pamięci.
- Jeden znak zajmuje jeden bajt. Adresem znaku jest adres zajmowanego przez niego bajtu.
- Zmienne innych typów zajmują więcej bajtów. Adresem takiej zmiennej jest *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*.
- Przykład:

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

- Typ obiektu na który wskaźnik może wskazywać nazywany jest *typem podstawowym wskaźnika*. Jest to informacja potrzebna kompilatorowi, aby wiedział, jaki jest rozmiar tego obszaru i jak interpretować obszar pamięci znajdujący się pod konkretnym adresem.
- Po zadeklarowaniu wartość wskaźnika jest nieokreślona. Aby z niego korzystać, trzeba mu przypisać wartość.

Podstawowe działania na wskaźnikach

- Wskaźnik służy do przechowywania adresu obiektu. Adres obiektu jest pobierany za pomocą *operatora adresu* (ang. *address-of operator*) - znaku `&` .

- Żaden obiekt nie będzie umieszczony pod adresem zero. Przyjęto więc, że przypisanie 0 wskaźnikowi oznacza, że nie pokazuje on na żaden obiekt.

- Przykład:

```
// deklaracja zmiennych
      int a, b, c;
      int *wi, *wj, *wk;
// przypisanie adresu
      wi = &a; // wi zawiera adres zmiennej a
      wj = wi; // obydwa wskaźniki wskazują na to samo miejsce
      wk = 0; // wk nie wskazuje na żaden obiekt
```

- 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 *wartości pośredniej* - ang. *indirection operator*). Jest nim gwiazdka `*`.

- Przykład:

```
// przypisanie wartości zmiennej wtedy, kiedy znamy jej nazwę
      a=5;
// przypisanie wartości zmiennej wtedy, kiedy znamy jej adres
       wi = &a;
     *wi = 10; // przypisanie zmiennej, której adres zawiera wi wartości 10
     b = *wi;   // przypisanie zmiennej b wartości spod adres wi
     wj = &b;   // przypisanie wj adresu zmiennej b
     *wj = *wi; // przypisanie zmiennej, której adres zawiera wj
                // wartości spod adresu wi
```

8.2 Zastosowanie wskaźników - przekazywanie adresów zmiennych do funkcji

- Przykład: Chcemy zamienić w funkcji wartości dwóch zmiennych, tak aby zmiana była widoczna po powrocie z funkcji:

```
// Nieprawidłowa funkcja zamien1 w8p1
      #include <iostream>
      using namespace std;

      void zamien1(int x, int y) {
           int tymcz;
           tymcz=x; x=y; y=tymcz;
      }
      int main() {
           int i=10, j=20;
           cout << "Przed zamiana \ti=" << i << "\tj=" << j << endl;
           zamien1(i,j);
           cout << "Po zamianie \ti=" << i << "\tj=" << j << endl;
           return 0;
      }

     [bl]$ ./zamienA
     Przed zamiana     i=10  j=20
     Po zamianie       i=10  j=20

// Prawidłowa funkcja zamien1 w8p2
      #include <iostream>
      using namespace std;
      void zamien2(int *x, int *y) {
           int tymcz;
           tymcz=*x; *x=*y; *y=tymcz;
      }
      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;

      }

     [bl]$ ./zamienB
     Przed zamiana     i=10  j=20
     Po zamianie       i=20  j=10
```

- Przykład: rozwiązywanie równania kwadratowego – chcemy znać wartości wyznaczone w funkcji (w8p3)

```
#include <iostream>
#include <cmath>
using namespace std;

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)
// a,b,c zmienne tylko czytane w funkcji
// x1, x2 wartości są wyznaczane w funkcji i przekazywane na zewnątrz

{
 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;
}
```

8.3 Tablice i wskaźniki

- Załóżmy, że mamy tablicę:

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

- Związek między tablicą a wskaźnikiem: nazwa tablicy jest adresem jej 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; // załóżmy, że typ int zajmuje 4 bajty
p1=&x;   // załóżmy, że adresem x jest 1000 i tę wartość przypisujemy
p1
p1++;    // wartość p1 to 1004 - zwiększenie o 4
p1=p1+9; // wartość p1 to 1004 + 9*4

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)`

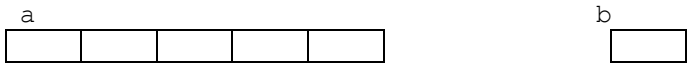
- Inna operacja to odejmowanie jednego wskaźnika od drugiego, kiedy oba wskaźniki wskazują na obiekt tego samego typu, na przykład tablicę. Wynikiem tej operacji jest liczba elementów typu bazowego oddzielających jeden wskaźnik od drugiego.

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 tyle miejsca w pamięci, ile w danej implementacji potrzeba na adres.

- 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;
```

  Do czego odnoszą się poniższe zapisy z użyciem zmiennej wsk?

Zapis z użyciem wskaźnika	Zapis z użyciem indeksu
`wsk`	adres `&a[2]` - (`a+2` w inicjacji)
`*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

- Fragment tabeli priorytetów operatorów:

Klasa	Operatory w klasie	Łączność	Przykład	Priorytet
przyrostkowe zwiększanie o 1 przyrostkowe zmniejszanie o 1	`++` `--`	lewostronna	`a++;` `a--;`	NAJ- WYŻSZY

adres
wyłuskanie
przedrostkowe zwiększanie
przedrostkowe zmniejszanie
	`&`
	`*`
	`++`
	`--`
		prawostronna	`&a`
			`*wsk`
			`++a`
			`--a`	

```
#include <iostream>  //w8p4
using namespace std;
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]={1,2,3}, b[10]={0};
int *wsk1, *wsk2;
wsk1=a; wsk2=b;
int i=0;
while (i++ < 10)
{
      *wsk2++ = *wsk1++;
}
```

Uwaga: instrukcja

```
x=*p++;
```

jest równoważna:

```
{ x=*p;
  p++;
}
```

8.4 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);
```

lub

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

Należy jednak 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>
using namespace std;

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>
using namespace std;

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>
using namespace std;

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>
using namespace std;

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> //w8p5
using namespace std;

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;
   }
   cout << endl;
}
```

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

8.5 Wskaźniki i indeksy

- Skoro do wartości elementów tablicy można odwoływać się zarówno za pomocą indeksów jak i wskaźników, czy są jakieś 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 wykonuje to tylko raz, podczas kompilacji.

```
// Wersja C
int tablica[10], *wsk;
for (wsk=tablica+9; wsk>=tablica; wsk--)
   *wsk=0;

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

- 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()
{
      int *wsk1, *wsk2;
      for (wsk1=x, wsk2=y; wsk1<x + MAX;)
            *wsk1++ = *wsk2++;
}
```

Deklarowanie argumentów typu tablicy

```
int fun(char *napis);
int fun(char napis[]);
```

- Jest to jedyne miejsce, w którym obydwie deklaracje są równoważne.

- Dla kompilatora *każdy* z zapisów oznacza wskaźnik. Jeżeli zastosowalibyśmy operator `sizeof(napis)`, otrzymalibyśmy rozmiar zmiennej wskaźnikowej, a nie rozmiar 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.

```
#include <iostream> //w8p6
using namespace std;

const int MAX=10;

void funkcja(int t[]) {
      int i;
      cout << "W funkcja: " << sizeof(t) << endl;
      for (i=0; i<MAX; ++i)
            t[i]=0;
}

int main() {
      int a[MAX];
      cout << "W main: " << sizeof(a) << endl;
      funkcja(a);
      return 0;
}
```

Wynik wykonania:

```
W main: 40
W funkcja: 4
```

8.6 Referencje

- *Referencja* (odniesienie, ang. *reference*) to inna nazwa obiektu (zmiennej).
- Technicznie można powiedzieć, że referencja jest pewnym rodzajem wskaźnika używanego bez specjalnej składni właściwej dla wskaźnika.
- Referencję deklarujemy za pomocą operatora `&` umieszczonego między nazwą typu i nazwą referencji:

```
int a=123;    // obiekt typu int
int *wsk=&a;  // wskaźnik do obiektu
int &ra=a;    // referencja do obiektu
```

- Referencja MUSI być zainicjowana:

```
int a=1;
int &r1=a; // OK.
int &r2;   // BŁĄD! Brak inicjatora
int &r3=&a // BŁĄD! Nie można inicjować adresem
int *wsk=&a, &r4=*wsk; // OK.
```

- Raz zdefiniowana referencja nie może być później zmieniana tak, aby odnosiła się do innego obiektu.
- Wszystkie działania wykonywane na obiekcie referencyjnym działają na tym obiekcie, do którego odnosi się dany obiekt referencyjny.

```
int a=0,b=1,c=1,*wa=&a; // deklaracja wskaźnika
int x=0,y=1,z=1,&rx=x; // deklaracja referencji
wa=&b; // zmiana wskaźnika
rx=y; // przypisanie zmiennej x wartości zmiennej y, czyli x=y
```

- W praktyce rzadko stosuje się samodzielne obiekty referencyjne. Najczęściej używa się referencji jako argumentów formalnych funkcji.

Referencje jako argumenty formalne funkcji

- Deklaracja referencyjnego argumentu formalnego unieważnia mechanizm domyślnego przekazywania argumentów przez wartość. Funkcja wie, gdzie dany argument znajduje się w pamięci i może dzięki temu zmienić jego wartość.

- *Przykład 1*. Referencja może zastąpić wskaźniki, jeżeli chcemy aby zmiana wartości była widoczna w funkcji wywołującej.

```
# include <iostream>
using namespace std;

// funkcja korzysta ze wskaźników
void swap1(int *x, int *y) {
      int temp;
      temp=*x;
      *x=*y;
      *y=temp;
}
int main() {
      int a=6,b=3;
      swap1(&a,&b);
      cout<<"Zamienione liczby
      6,3\t"
            <<a <<'\t' <<b <<endl;
      return 0;
}
```

```
# include <iostream>
using namespace std;

// funkcja korzysta z referencji
void swap2(int &x, int &y) {
      int temp
      temp=x;
      x=y;
      y=temp;
}
main() {
      int c=4,d=8;
      swap2(c,d);
      cout <<"Zamienione liczby
      4,8\t"
            <<c <<'\t' <<d <<endl;
      return 0;
}
```

- *Przykład 2*. Wykorzystanie referencji do przekazywania dodatkowego wyniku do funkcji wywołującej. Funkcja `szukaj()` przeszukuje tablicę liczb całkowitych `t` o rozmiarze `n` i poszukuje w niej liczby `x`. Liczby w tablicy mogą się powtarzać. Jeśli liczba zostanie znaleziona, funkcja zwraca indeks wskazujący pierwsze wystąpienie liczby `x`. Jeśli liczba nie zostanie znaleziona przekazywana jest liczba przeszukanych elementów tablicy. Dodatkowo za pośrednictwem parametru referencyjnego przekazywana jest informacja o liczbie wystąpień szukanej wartości.

```
int szukaj(int *t, int n, int x, int &wystepuje)
{
    int indeks=n;
    wystepuje=0;
    for (int i=0; i<n; ++i)
      if (*(t+i) == x)
      {
          if (indeks==n)
              indeks=i;
          ++wystepuje;
      }
    return indeks;
}
```

8.7 Wskaźniki i tablice wielowymiarowe

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

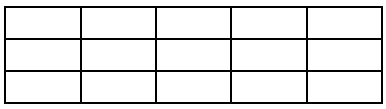
  `a` `&a[0]` są równoważne

- Dostęp do elementu a_2 tablicy:

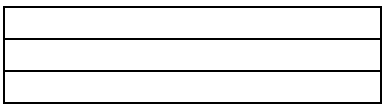
```
x = a[2];
x = *(a+2);
```

- Tablica *wielowymiarowa*:

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



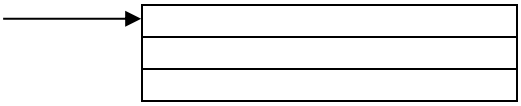
jest równoważne



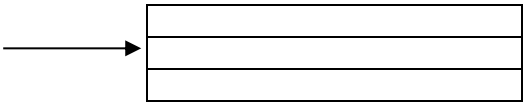
Zapisy:
`a` `&a[0]` `&a[0][0]` są równoważne

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

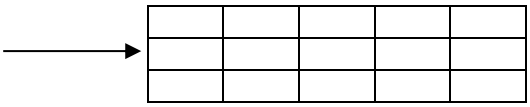
  `a` jest wskaźnikiem do tablicy o 5 wartościach `int`:



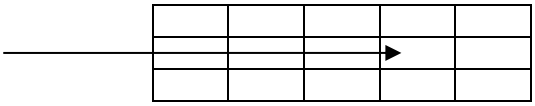
  `a+1` jest wskaźnikiem do tablicy o 5 wartościach `int`:



  `*(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

```
// tablica jednowymiarowa –
int wektor[10];
int *wsk1=wektor;
wsk1++; // dodanie 1 do wskaźnika oznacza przesunięcie o jeden element

// tablica wielowymiarowa –
// dodanie 1 do wskaźnika może oznaczać różne przesunięcia, w zależności
// od tego jaki wskaźnik został zdefiniowany

int b[3][5];
int *wsk2=&b[0][0]; // wsk3 wskazuje pierwszy element macierzy
wsk2++; // dodanie 1 oznacza przesunięcie o jeden element

int c[3][5];
int *wsk3=c[0]; // wsk4 wskazuje pierwszy element macierzy
wsk3++; // dodanie 1 oznacza przesunięcie o jeden element

int a[3][5];
int (*wsk4)[5]=a ; // wsk2 wskazuje pierwszy wiersz macierzy,
wsk++; // dodanie 1 oznacza przesunięcie o wiersz
```

8.7.1 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, zachodzi 1 z 3 przypadków:
  - 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>
#include <iomanip>
using namespace std;
void podw(int t[], int n);
int main() {
  int a[3][4] = {
      {2,4,5,8},
      {3,5,6,9},
      {12,10,8,6}
      };
  for (int i = 0; i < 3 ; i++)
      podw(a[i], 4);
  for (int i = 0; i < 3; i++) {
        for (int j = 0; j < 4; j++)
          cout << setw(4) << a[i][j] ;
        cout << endl;
  }
  return 0;
}
void podw(int t[], int n)
{
  for (int i = 0; i < n; i++)
     t[i] *= 2;
}
```

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

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

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

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

```
#include <iostream>
#include <iomanip>
using namespace std;

void podw(int *t, int n);

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

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

C. Funkcja jest dostosowana do tablicy dwuwymiarowej

```
#include <iostream>
#include <iomanip>
using namespace std;

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}
  };
  podw(a, 3, 4);
  for (int i = 0; i < 3; i++) {
    for (int 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>
#include <iomanip>
using namespace std;

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}
                };
  podw(a, 3, 4);
  for (int i = 0; i < 3; i++) {
     for (int 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 (w8p7)

```
#include <iostream>
using namespace std;
void drukuj_wiersz(int *tablica, int nr_wiersza, int ile_kolumn);
void drukuj_tablice(int *tablica, int ile_wierszy, int ile_kolumn);
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]=i*4+j;
    /* wyświetlenie liczb w tablicy wiersz po wierszu */
    for (i=0; i<3; ++i)
      drukuj_wiersz(&a[0][0],i,4);
    cout << endl;
    /* wyświetlenie liczb w całej tablicy */
    drukuj_tablice(&a[0][0],3,4);
    return 0;
}
void drukuj_wiersz(int *tablica, int nr_wiersza, int ile_kolumn) {
    int j;
    tablica=tablica+nr_wiersza*ile_kolumn;
    for (j=0; j<ile_kolumn;++j) cout << *(tablica+j)<<' ';
    cout << endl;
}
void drukuj_tablice(int *tablica, int ile_wierszy, int ile_kolumn) {
    for(int i=0;i<ile_wierszy;++i)
      drukuj_wiersz(tablica, i, ile_kolumn);
}
```

8.8 Ochrona zmiennych przekazywanych do funkcji

- Jeśli do funkcji przekazywana jest prosta zmienna, na przykład typu `int`, programista może przekazać do funkcji wartość zmiennej lub wskaźnik do niej. Powszechnie stosuje się zasadę, że przekazuje się wartość, chyba, że występuje konieczność zmiany tej wartości, wtedy bądź przekazywany jest adres, bądź korzysta się z mechanizmu referencji.

- W przypadku tablic nie ma wyboru: zawsze przekazywany jest adres. W rezultacie funkcja przetwarzająca tablicę działa na danych pierwotnych i może je swobodnie zmieniać. Czasem jest to pożądane, ale wiele funkcji tylko czyta tablice i nie modyfikuje ich zawartości. Jednakże, jeśli programista popełni błąd, może zniszczyć dane zapisane w tablicy.

- Przykład:

```
#include <iostream>
using namespace std;

// Funkcja poprawna
int suma(int t[], int n) {
  int suma=0;
  for (int i=0; i< n; i++)
    suma += t[i];
  return suma;
}
int main() {
  int a[10]={1,3,2,5,4,8,9,2,4,2};
  cout << "suma = " << suma(a,10)
    << endl;
  return 0;
}
```

```
#include <iostream>
using namespace std;

// Funkcja błędna
int suma(int *t, int n) {
  int suma=0;
  for (int i=0; i< n; i++)
    suma += t[i]++;
  return suma;
}
int main() {
  int a[10]={1,3,2,5,4,8,9,2,4,2};
  cout << "suma =" << suma(a,10)
    << endl;
  return 0;
}
```

Obie wersje się skompilują, mimo, że w wersji II dodatkowo zmieniliśmy zawartość tablicy.

- Jeśli wiemy, że wartości elementów tablicy nie mogą być w funkcji zmieniane, możemy zadeklarować argument formalny funkcji z użyciem słowa kluczowego `const`. Czyli nagłówek funkcji może mieć postać:

```
int suma(const int t[], int n)
lub
int suma(const int *t, int n)
```

Oznacza to, że *w funkcji* tablica t ma być tak traktowana, jakby zawierała stałe dane. Próba kompilacji wersji 2 funkcji z tym nagłówkiem zakończy się komunikatem wskazującym na zmianę wartości, która nie powinna być zmieniana:

```
Cannot modify a const object.
```

8.9 Przekazywanie struktur do funkcji

Przekazywanie składowych struktury

- Składowe przekazuje się tak jak zmienne proste:
  - przez wartość
  - przez adres
  - przez referencję

- Przykład 1. Przekazywanie przez adres

```
#include <iostream>
using namespace std;

struct Ulamek {
   long licznik;
   long mianownik;
};

void DrukujUlamek(char *opis, long *x) {
   cout << opis << *x <<endl;
}

int main() {
   struct Ulamek u;
   u.licznik=1; u.mianownik=2;
   DrukujUlamek("Licznik:",&u.licznik);
   DrukujUlamek("Mianownik:",&u.mianownik);
   return 0;
}
```

- Przykład 2. Przekazywanie przez referencję

```
#include <iostream>
using namespace std;

struct Ulamek {
   long licznik;
   long mianownik;
};

void DrukujUlamek(char *opis,long &x) {
    cout << opis << x << endl;
}

int main() {
     struct Ulamek u;
     u.licznik=1; u.mianownik=2;

     DrukujUlamek("Licznik: ",u.licznik);
     DrukujUlamek("Mianownik: ",u.mianownik);
     return 0;
}
```

Przekazywanie całych struktur

- Użycie struktury jako argumentu funkcji powoduje przekazanie struktury *przez wartość* i *nie jest zalecane* szczególnie w przypadku dużych struktur.

- Przykład 1. Przekazywanie przez wartość NIEZALECANE

```
#include <iostream>
using namespace std;

struct Ulamek {
   long licznik;
   long mianownik;
};

void DrukujUlamek(char *opis, Ulamek x) {
   cout << opis << x.licznik << '/' << x.mianownik << endl;
}

int main() {
   struct Ulamek u;
   u.licznik=1; u.mianownik=2;
   DrukujUlamek("Licznik/Mianownik ",u);
   return 0;
}
```

- Przykład 2. Przekazywanie przez adres

```
#include <iostream>
using namespace std;

struct Ulamek {
    long licznik;
    long mianownik;
};

void DrukujUlamek(char *opis, Ulamek *x) {
    cout << opis << x->licznik << '/' << x->mianownik << endl;
}

int main() {
    struct Ulamek u;
    u.licznik=1; u.mianownik=2;
    DrukujUlamek("Licznik/Mianownik: ",&u);
    return 0;
}
```

- Przykład 3. Przekazywanie przez referencję

```
#include <iostream>

struct Ulamek {
    long licznik;
    long mianownik;
};

void DrukujUlamek(char *opis, Ulamek &x) {
    cout << opis << x.licznik << '/' << x.mianownik << endl;
}

int main() {
    struct Ulamek u;
    u.licznik=1; u.mianownik=2;
    DrukujUlamek("Licznik/Mianownik: ", u);
    return 0;
}
```

- Przykład:

```
#include <iostream>
using namespace std;

struct Pomiary {
  char nazwa;
  int a,b;
};

void drukuj(Pomiary proba);

int main() {
  Pomiary proba;
  proba.nazwa='A';
  proba.a=100;
  proba.b=20;
  drukuj(proba);
  return 0;
}

// Przekazywanie przez wartość NIEZALECANE
void drukuj(Pomiary proba) {
  cout << "Nazwa: "<< proba.nazwa
    <<" Wyniki: " << proba.a << '\t' << proba.b << endl;
}
```

- *Uwaga*: należy unikać takiego przekazywania, ponieważ niepotrzebnie zwiększa się czas wykonywania oraz może wystąpić przepełnienie stosu.

- Zalecane jest posługiwanie się referencją lub wskaźnikiem do pierwszego elementu struktury.

```
#include <iostream> //(w8p9)
using namespace std;

struct Pomiary {
char nazwa;
int a,b;
};

void drukuj(Pomiary &proba);

int main() {
   Pomiary proba;
   proba.nazwa='A';
   proba.a=100;
   proba.b=20;
   drukuj(proba);
   return 0;
}

// Przekazywanie przez referencję
void drukuj(Pomiary &proba) {
   cout << "Nazwa: "<< proba.nazwa
       <<" Wyniki: " << proba.a << '\t' << proba.b << endl;
}
```

8.10 Zwracanie struktur z funkcji

- Funkcja może zwracać strukturę lub wskaźnik do struktury.

- Przykład 1:

```
#include <iostream>
using namespace std;

struct Punkt {
   int x;
   int y;
};

Punkt wprowadz(char *tekst) {
   Punkt p;
   cout << tekst<<": ";
   cin >> p.x >> p.y;
   return p;
}

int main() {
   Punkt p1=wprowadz("Podaj wspolrzedne punktu p1");
   cout << "Punkt p1: " << p1.x << ',' << p1.y << endl;
   Punkt p2=wprowadz("Podaj wspolrzedne punktu p2");
   cout << "Punkt p2: " << p2.x << ',' << p2.y << endl;
   return 0;
}
```

- Przykład 2: funkcja zwraca wskaźnik do punktu o najmniejszej współrzędnej x (w8p8)

```
#include <iostream>
#include <cmath> // dla abs()
using namespace std;

struct Punkt {
   int x;
   int y;
};

Punkt *minX(Punkt *tablica, int ile)
{
   int p=0; // indeks punktu o najmniejszej współrzędnej x
   for (int i=1;i<ile;i++)
      if (abs(tablica[p].x) > abs(tablica[i].x)) p=i;
   return (tablica+p);
}

int main() {
   Punkt t[]={{1,2},{-3,5},{2,8}};
   Punkt *w=minX(t,sizeof(t)/sizeof(*t));
   cout << "Punkt o najmniejszej wsp. x: "
       << w->x << ',' << w->y << endl;
   return 0;
}
```

8.11. Wskaźniki do struktur

- Deklarowanie wskaźników do struktur

```
nazwa_struktury *nazwa_zmiennej;
```

- Przykład:

```
struct Data {
   int dzien;
   int miesiac;
   int rok;
};
Data data_biezaca, tydzien[7];
Data *wsk_data;
```

- Adres zmiennej strukturowej

```
wsk_data=&dzien_biezacy;
wsk_data=&tydzien[0];
```

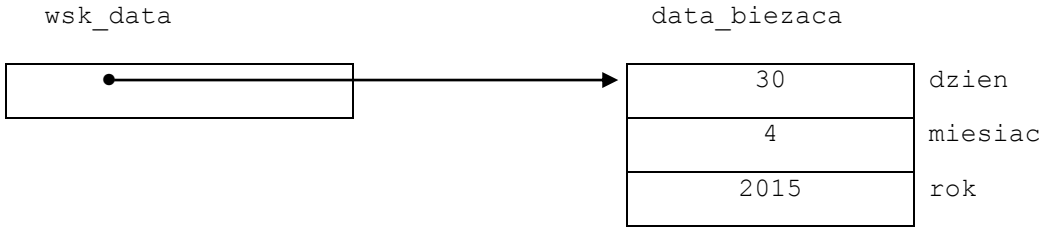
- Dostęp do wartości pól struktury uzyskiwany jest za pomocą operatora `->` .

Zamiast pisać:

```
(*wsk_data).dzien=30;
```

można:

```
wsk_data->dzien=30;
wsk_data->miesiac=4;
wsk_data->rok=2015;
```



- Przykład: przekazywanie struktury za pomocą wskaźnika (w9p9)

```
#include <iostream>
using namespace std;

struct Pomiary {
    char nazwa;
    int a,b;
};

void drukuj(Pomiary *proba);

int main() {
    Pomiary proba;
    proba.nazwa='A';
    proba.a=100;
    proba.b=20;
    drukuj(&proba);
    return 0;
}

void drukuj(Pomiary *proba) {
    cout << "Nazwa: "<< proba->nazwa << '\t'
      <<" Wyniki: " << proba->a << '\t'
      << proba->b << endl;
    }
```

8.11 Wskaźniki stałych i stałe wskaźniki

- Wskaźnik stałej: wskazywany obiekt nie może być modyfikowany, może wskazywać różne obiekty

```
int i;                // zmienna typu int
const int k = 13;     // stała typu int
const int *wk;        // wskaźnik stałej typu int

wk=&k                 // OK
wk=&i;                // OK

int i=1;              // zmienna typu int
const int k = 13;     // stała typu int
const int* wk;        // wskaźnik stałej typu int

wk=&k;                // OK
*wk += 1;             // blad
cout << *wk << endl; // OK

wk=&i;                // OK
cin >> *wk;           // blad
```

- Stały wskaźnik: zawsze wskazuje ten sam obiekt, można zmieniać wartość obiektu

```
int i=1;              // zmienna typu int
const int k = 13;     // stała typu int
int* const wk = &i;   // stały wskaźnik zmiennej i, musi być zainicjowany
*wk=5;                // OK, można zmienić wartość zmiennej
wk=&k;                // blad, nie moze wskazywac innej zmiennej
```

- Stały wskaźnik stałej: kombinacja obydwu powyższych

```
const char* const s = "witam";
```

- Przykład zastosowania wskaźnika stałej:

```
#include <iostream>
using namespace std;

int suma(const int *t, int n)
{
   int s=0;
   for (int j=0; j<n; j++)
      s += *(t+j);
   return s;
}

int main() {
   const int rok[12]={31,28,31,30,31,30,31,31,30,31,30,31};
   int pierwszyKwartal[3]={31,28,31};
   cout << suma (rok,3) << endl;
   cout << suma(pierwszyKwartal,3) << endl;
   cin.get();
   return 0;
}
```

Zadania

1. Prawda czy fałsz? Przy każdym z poniższych stwierdzeń zaznacz P lub F.

a) Zarówno wskaźniki jak i referencje przechowują adres obszaru pamięci.
b) Referencja musi być inicjalizowana podczas deklaracji.
c) Operator `new` zwraca referencję.
d) Funkcja może zwracać wskaźnik ale nie może zwracać referencji.

2. Które z poniższych wierszy mają poprawną składnię? Jeśli nie, to dlaczego?

```
a) double a, *wa=0;
b) char *str, ch=&str;
c) int n, r=&n;
d) char ch, &rch=ch, *pch=&ch;
e) double z, &z1=z, &z2=z;
f) int k, &refK=k, &ref2K=refK;
g) char str[20], *s=str;
h) int punkty[100], *x=&punkty[0], y=&punkty[1];
i) const int N=20; char str[N], *tail=&str[N-1];
j) double *probka1=new double[200], *probka2=probka1+100;
```

3. Co zostanie wydrukowane w poniższych przykładach:

```
a) char ch1='*', ch2='+', *s=&ch1;
   ch2=*s;
   cout << ch1 << ch2;

b) int x=3, y=9, &r=y, *p;
   p=&y;
   *p=0;
   cout << x<<r;

c) double u=1.1, v=1.2, *max=&u;
   if (v>u) max=&v;
   cout << *max;
```

4. Dana jest funkcja:

```
int F(int &x, int &y)
{ x *= 2; y *=2; return x*y; }
```

Co zostanie wyświetlone w wyniku wykonania instrukcji:

```
int a=3;
cout << F(a,a) << ' ';
cout << a << endl;
```

A. `9 3`	B. `18 3`	C. `36 6`	D. `39 12`	E. `144 12`

5. Napisz funkcję `Odwroc()`, która pobiera jeden argument `x` typu `double` przekazany przez referencję. Jeśli `x` jest różne od zera funkcja przypisuje zmiennej `x` wartość `1/x` i zwraca `true`. W przeciwnym wypadku zwraca `false`. Napisz funkcję `main()` testującą opracowaną funkcję.

6. Napisz i przetestuj funkcję, która oblicza średnią i odchylenie standardowe. Prototyp funkcji ma postać:

```
void Probka(double x[], int n, double &srednia, double &stddev);
```

7. Dane są następujące deklaracje:

```
struct Punkt {
   int x;
   int y;
};

struct Wielobok {
int liczbaBokow;
Punkt wierzcholki[5];
};

Wielobok z;              // wielobok
Wielobok *wsk = &z;      // wskaźnik do wieloboku
Wielobok rysunek[10];    // zestaw wieloboków tworzących rysunek
```

Uzupełnij poniższe deklaracje wieloboku:

a) posługując się nazwą wieloboku `z`

```
int n=                   // liczba boków w wieloboku
Punkt w=                 // i-ty wierzchołek w wieloboku
Punkt *wsk_w=            // wskaźnik do i-tego wierzchołka w wieloboku
int x=                   // wsp. x i-tego wierzchołka w wieloboku
```

b) posługując się wskaźnikiem do wieloboku `wsk`

```
int n=                   // liczba boków w wieloboku
int x=                   // wsp. x i-tego wierzcholka w wieloboku
```

Uzupełnij poniższe deklaracje dla zestawu wieloboków `rysunek`

```
int n=                   // liczba boków w k-tym wieloboku
int x=                   // wsp. x i-tego wierzchołka w k-tym wieloboku
```

8. Dane są następujące deklaracje:

```
struct DUZALICZBA {
    char znak;
    int cyfry[100];
};
DUZALICZBA x, *w, a[5];
```

Które z poniższych instrukcji mają poprawną składnię?

a) `int x = x.cyfry[0];`
b) `int y = x.cyfry[0];`
c) `int cyfry = a[3].cyfry[3];`
d) `char znak = w->znak;`
e) `int *cyfr = x.cyfry;`
f) `char *s = &w->znak;`