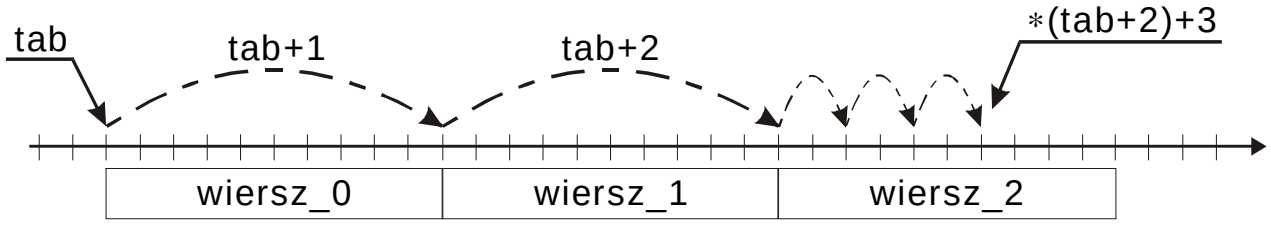
WSKAŹNIKI I TABLICE WIELOWYMIAROWE

1) Operacje na tablicach wielowymiarowych w zapisie indeksowym:

```
int  tab[ 3 ][ 5 ] ;
int  i, j ;
for( i=0 ; i<3 ; i++ )
   for( j=0 ; j<5 ; j++ )
      {
         printf( "TAB[%d, %d]= ", i, j );        // cout<<"TAB["<< i <<","<< j <<"]=";
         scanf( "%d" , &tab[ i ][ j ] );                          // cin >> tab[ i ] [ j ];
      }
```

2) Reprezentacja tablicy **int** tab[3][5] w pamięci komputera:



3) Operacje na tablicy dwuwymiarowej w zapisie wskaźnikowym:

```
int  tab[ 3 ][ 5 ] ;
int  i, j ;
for( i=0 ; i<3 ; i++ )
   for( j=0 ; j<5 ; j++ )
      {
         printf( " TAB[ %d , %d ]= ", i, j );
         scanf( "%d" , *(tab + i) + j  );          // scanf("%d", & *(*(tab + i) + j) );
      }                                            //          cin >> *(*(tab + i) + j) ;
```

4) Operacje na tablicy dwuwymiarowej bez wykorzystywania indeksów liczbowych:

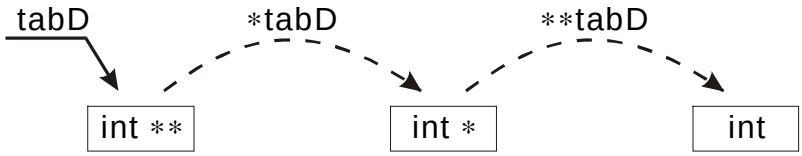
```
int  tab[ 3 ][ 5 ] ;

int  (*wsk_w) [ 5 ] ;      // wskaźnik na wiersz  tzn. na 5-cio elementową tablicę int
int*  wsk_k ;              // wskaźnik na kolumnę tzn. na liczbę int

for( wsk_w = tab ; wsk_w < tab + 3 ; wsk_w++ )
   for( wsk_k = *wsk_w ; wsk_k < *wsk_w + 5 ; wsk_k++ )
      {
         printf( " TAB[ %d , %d ]= ", wsk_w – tab,  wsk_k – *wsk_w );
         scanf( "%d" , wsk_k );
      }
```

5) Różnica pomiędzy „wskaźnikiem na tablicę” a „wskaźnikiem na wskaźnik”:

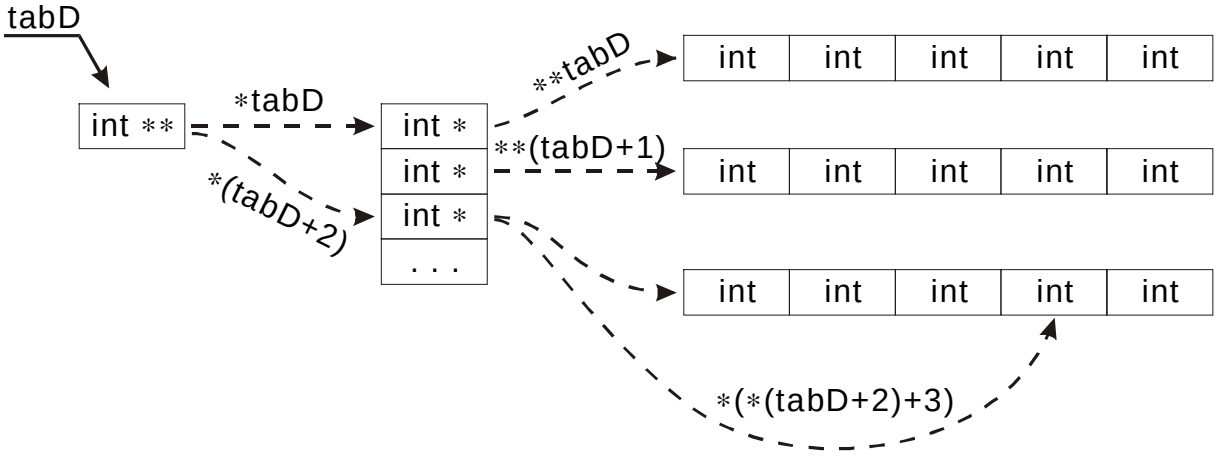
```
int  tab[ 3 ][ 5 ] ;
int   i, j ;
tab[ i ][ j ]  == *( *(tab + i) + j)        np. tab[ 0][ 0]  == *(*(tab+0)+0) == **tab
```

Dlaczego **tab** jest typu **int (*)[5]** a nie typu **int **** ?

int **tabD → To jest „wskaźnik na wskaźnik na zmienną” (*adres adresu obiektu*):



int **tabD → lub wskaźnik na „tablicę wskaźników” na tablice:




```
// przykładowy program tworzący dynamiczna strukturę danych j.w.
int** tabD = new int* [4];
                                        // lub równoważnie:
for( int i=0; i<4; i++ )                //    for( int i=0; i<4; i++ )
   *(tabD+i) = new int [5];             //       tabD[ i ] = new int [5];


// zapis liczby 111 do wybranego elementu tablicy tabD
*(*(tabD+2) + 3) = 111 ;                //   tabD[ 2 ][ 3 ] = 111 ;


// zamiana miejscami wierszy o indeksach 1 i 3
int* wsk_pom ;
wsk_pom = *(tabD +1) ;                  // wsk_pom = tabD[0];
*(tabD + 1) = *(tabD + 3) ;             // tabD[0] = tabD[3] ;
*(tabD + 3) = wsk_pom ;                 // tabD[3] = wsk_pom ;
```

WSKAZANIA NA FUNKCJE

→ nazwa funkcji jest stałą równą adresowi kodu funkcji w pamięci komputera (analogicznie jak nazwa tablicy jest stałą równą adresowi tablicy),

```
#include <conio.h>
. . .
clrscr ;      // podanie samej nazwy funkcji jest równoważne podaniu adresu
              // i nie powoduje żadnej akcji (podobnie jak polecenie  10 ;  )
clrscr() ;    // nazwa funkcji z nawiasami () jest traktowana jako „wywolanie
              // funkcji" tzn. polecenie wykonania fragmentu kodu umieszczo-
              // nego pod podanym adresem
```

→ możliwość pośredniego dostępu do funkcji (poprzez zmienną zawierającą adres / wskazanie na funkcję). Ogólna postać definicji wskaźnika funkcji:

typ_zwracany_przez_funkcję (∗**nazwa_zmiennej**) (**parametry_funkcji**);

```
void  clrscr( void );                          // prototyp funkcji clrscr()
void  (∗nowy_clrscr) ( void );                 // definicja zmiennej wskazującej na funkcję
. . .
nowy_clrscr = clrscr;                          // przypisanie zmiennej nowy_clrscr adresu clrscr
. . .
clrscr();          // bezposrednie wywołaniefunkcji clrscr()
nowy_clrscr();     // wywolanie funkcji wskazywanej przez zmienną
```

→ to daje możliwość napisania **funkcji, których parametrami są inne funkcje** ! (wywoływanie funkcji, których adresy zostaną podane dopiero w trakcie wykonywania programu).

```
// np. uniwersalna funkcja licząca sume N elementów dowolnego ciągu
double Suma_Ciagu(   double (∗Element)( int ) ,  int ilosc )
{
   double s = 0;
   for( int i = 0; i < ilosc; i++ )
      s += Element( i );
}

double Nty_Element ( int n )                   // 1, 1/2, 1/3, 1/4, 1/5, ...
{  return( 1.0/(n+1) );  }

. . .
printf( "Suma elementów = %lf", Suma_Ciagu(  Nty_Element  , 100 );
. . .
```

funkcja **qsort** **<stdlib.h>**

→ implementacja algorytmu sortowania szybkiego (ang. quick sort)
pozwalająca sortować tablice obiektów dowolnego typu
według dowolnego zadanego kryterium (funkcji definiującej relację porządku)

prototyp funkcji:

```
void qsort(
            void    *base,                  // adres poczatku sortowanego obszaru
            size_t  nelem,                  // ilosc sortowanych elementów
            size_t  width,                  // rozmiar pojedynczego elementu
            int (*fcmp)( void *,  void *)   // adres funkcji porównująca
          );
```

Przykład sortowania tablicy liczb całkowitych

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

int liczby_rosnaco( const void* , const void* );
void wyswietl( int [ ], int  );

void main( void )
{
   int tab[10] = { 12, –1, 3, 0, 10, 1, 2, 6, 4, 9 };
   wyswietl( tab, 10 );
   qsort( tab, 10, sizeof(int), liczby_rosnaco );
   wyswietl( tab, 10 );
}

int liczby_rosnaco( const void* wsk_1, const void* wsk_2)
{
   int *wsk_liczby_1, *wsk_liczby_2;
   wsk_liczby_1 = (int*)wsk_1;
   wsk_liczby_2 = (int*)wsk_2;
   return( *wsk_liczby_1 – *wsk_liczby_2 );
}

void wyswietl( int tab[ ], int ilosc )
{
   int i;
   for( i = 0;  i<ilosc;  i++ )
      printf( "tab[%d]=%d\n" , i , tab[i] );
}
```

Przykład sortowania tablicy tekstów

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

int teksty_rosnaco( const void *wsk_1, const void *wsk_2)
{
   return(  strcmp(  (char *) wsk_1,  (char *) wsk_2)  );
}

void main( void )
{
   char tab_tekstow[5][10] = { "Opel", "Audi", "Ford", "Trabant", "Fiat" };
   qsort(  tab_tekstow,  5,  sizeof( tab_tekstow[0] ) ,  teksty_rosnaco );
}
```

Przykład sortowania bazy danych (tablicy struktur)

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

struct student
   {
      char  nazwisko[31];
      char  imie[16];
      int     wiek;
      char  plec;
      float  stypendium;
   } ;

int wedlug_nazwisk( const void *wsk_1, const void *wsk_2)
{
   struct student *osoba_1 = (struct student *) wsk_1;
   struct student *osoba_2 = (struct student *) wsk_2;
   return(  strcmp( osoba_1->nazwisko, osoba_2->nazwisko );
}


void main( void )
{
   #define MAX_IL  100
   struct student  baza[ MAX_IL ];
     . . .
   qsort(  baza,  MAX_IL,  sizeof( struct student ), wedlug_nazwisk );
}
```

PRZYKŁADY RÓŻNYCH KOMBINACJI WSKAŹNIKÓW

Przykładowe elementy:

```
float  LICZBA;                  // liczba rzeczywista float
int  TAB_INT [ 5 ];             // 5-cio elementowa tablica liczb int
double FUNKCJA ( int x )        // funkcja z parametrem int zwracająca
  {                             // wartość double
    return x+0.1;
  }
```

Wskaźniki na w/w elementy:

```
float* wsk_liczby ;             // wskaźnik na liczbę float
wsk_liczby = & LICZBA ;

int ( *wsk_tab ) [ 5 ] ;        // wskaźnik na 5-cio elementowa tablicę
wsk_tab = & TAB_INT ;

double ( *wsk_fun ) ( int ) ;   // wskaźnik na funkcję
wsk_fun = FUNKCJA ;
```

Tablice elementów:

```
float tab_liczb [ 10 ] ;        // tablica liczb float

int tab_tab [ 10 ] [ 5 ] ;      // tablica tablic

// – – – – – – – – – –          // nie ma tablicy funkcji !
```

Tablice wskaźników na elementy:

```
float* tab_wsk_liczb [ 10 ] ;           // tablica wskaźników na liczby
tab_wsk_liczb [ 2 ] = & LICZBA ;

int (* tab_wsk_tab [ 10 ] ) [ 5 ] ;    // tablica wskaźników na tablice
tab_wsk_tab [ 2 ] = & TAB_INT ;

double (* tab_wsk_fun [ 10 ] ) ( int ) ;  // tablica wskaźników na funkcje
tab_wsk_fun [ 2 ] = FUNKCJA ;
```

Funkcje zwracające elementy:

```
float FUNKCJA_E1 ( void )       // funkcja zwracająca liczbę float
  { return 0.1; }

// – – – – – – – – – –          // nie ma funkcji zwracającej tablicę !

// – – – – – – – – – –          // nie ma funkcji zwracającej funkcję !
```

Funkcje zwracające wskaźniki elementów:

```
float* FUNKCJA_W1( void )                  // funkcja (void) zwracająca
  {                                        // wskaźnik na liczbę float
    float* wsk_liczby ;
    wsk_liczby = &LICZBA ;
    return wsk_liczby ;
  }

int (* FUNKCJA_W2( void ) ) [ 5 ]          // funkcja (void) zwracająca
  {                                        // wskaźnik na tablicę
    int (*wsk_tab)[ 5 ] ;                  // pięciu liczb int
    wsk_tab = &TAB_INT ;
    return wsk_tab ;
  }

double (* FUNKCJA_W3( void ) ) ( int )     // funkcja (void) zwracająca
  {                                        // wskaźnik na funkcję
    double (*wsk_fun)( int );              //  double (int)
    wsk_fun = FUNKCJA ;
    return wsk_fun ;
  }
```

Tylko dla koneserów ☺

Tablica wskaźników na funkcje **double** (**int**)

```
double (* tab_wsk_fun[ 10 ] ) ( int ) ;
```

Wskaźnik tablicy wskaźników na funkcje **double** (**int**)

```
double (* (*wsk_tab_wsk_fun) [ 10 ] ) ( int ) ;
```

Funkcja (**void**) zwracająca wskaźnik tablicy wskaźników na funkcje **double**(**int**)

```
double (* (* fun_wsk_tab_wsk_fun( void ) ) [ 10 ] ) ( int )
  {
    return wsk_tab_wsk_fun ;
  }
```