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 tablice

// – – – – – – – – – – –        // nie ma funkcji zwracającej funkcje
```

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 ;
  }
```

DYNAMICZNE PRZYDZIELANIE PAMIECI

Pamięć komputera, dostępna dla programu, dzieli się na cztery obszary:

- **kod programu**,
- dane **statyczne** (np. stałe i zmienne globalne programu),
- dane **automatyczne** (zmienne lokalne funkcji - tworzone i usuwane automatycznie przez kompilator) → tzw. STOS (*ang. stack*)
- dane **dynamiczne** (zmienne, które można tworzyć i usuwać w dowolnym momencie pracy programu) → w pamięci wolnej komputera → tzw. STERTA (*ang. heap*)

Zmienne dynamiczne → są to zmienne tworzone przez programistę w pamięci wolnej komputera (na stercie)

→ dostęp do takiej zmiennej możliwy jest jedynie poprzez jej adres w pamięci (przechowywany w zmiennej wskaźnikowej).

W języku „**C**" do dynamicznego przydzielania pamięci (tworzenia zmiennych dynamicznych) służyły specjalne funkcje z biblioteki **< alloc.h >**

```
void *malloc( size_t  rozmiar );                  // przydział bloku o zadanej wielkosci
void *calloc( size_t  il_elementow,  size_t rozmiar);   // przydział tablicy
void  free( void *wskaznik);                      // zwolnienie wskazywanego obszaru
unsigned  coreleft( void );                       // sprawdzenie wolnego miejsca na stercie
```

np.
```
void main( void )
{
   int *wsk;          // zmienna wskaźnikowa do zapamiętania adresu liczby int
   • • •
   wsk = (int*) malloc( sizeof(int) );     // przydzielenie pamięci na liczbę int
   if( wsk == NULL )
      {  printf( "Błąd przydziału pamięci" );  return;
   • • •
   *wsk = 10;                  // przykładowe operacje na dynamicznej liczbie int
   *wsk *= 2;
   printf( "%d", *wsk );
   scanf( "%d", wsk );
   • • •
   free( wsk );                // zwolnienie pamięci przed zakończeniem programu
}
```

Przykład operacji na dynamicznej tablicy o dowolnej ilości elementów:

```
int rozmiar_tablicy;
double *tablica_liczb;
printf( "Ile liczb chcesz wprowadzić: " );
scanf( "%d", &rozmiar_tablicy );
if( tablica_liczb = (double*) calloc( rozmiar_tablicy, sizeof(double) ) )
  {
    for( int i = 0; i < rozmiar_tablicy, i++ );
      *( tablica_liczb+i ) = 100;          // tablica_liczb[ i ] = 100;
    • • •
    free( tablica_liczb );
  }
```

W języku „**C**++" do dynamicznego przydzielania pamięci wygodniej jest wykorzystywać operatory **new** i **delete** :

```
<wskaźnik_na_obiekt> = new  <typ_obiektu> [parametry_inicjacyjne] ;
delete  <wskaźnik_na_obiekt> ;
```

np.

```
int*  wsk ;                  // wskaźnik na zmienną typu całkowitego
wsk = new int ;              // utworzenie nowego obiektu (nowej zmiennej int)
if( wsk != NULL )
  {
    *wsk = 10 ;              // przypisanie wartości  (poprzez wskaźnik)
    printf( "%d" , *wsk );   // wydrukowanie zawartości zmiennej dynam.
    • • •
    delete wsk ;
  }
```

Porównanie utworzenia zwykłej tablicy i tablicy dynamicznej:

```
const  ROZMIAR_TABLICY = 100;
double zwykła_tablica[ ROZMIAR_TABLICY ];

int rozmiar_tablicy;
cout << "Ile liczb chcesz wprowadzić: " ;
cin >> rozmiar_tablicy ;
double *tablica_dynamiczna;
tablica_dynamiczna = new double[ rozmiar_tablicy ];
• • •
delete [ ] tablica_dynamiczna;
```