

POINTERS / ADDRESSES OF THE VARIABLES

Pointer → is a variable (or constant) that contains the address of another variable or address of any area in computer memory,
(e.g. it may be the address of a data or address of the program code)

The general form of the pointer definition:

type_of_data * pointerName ;

The most frequently used are „defined pointers, which contains information about:

- the **address** of the variable in the computer RAM memory
- the **type** of the data, stored in this variable

Examples of definitions:

```
int      * pointer;           // Pointer to an integer variable
double   * ptr_number;       // Pointer to the real (double) variable
char      * ptr_char;         // Pointer to a single character
char      * text;             // Pointer to the beginning of the string
                                (the first sign of the text string)
```

We can also use the "undefined" (anonymous) pointers. Such pointer provides information only about the physical “address” → the beginning of the region (indicated without specifying the type of data)

Example definition:

void * pointerName ;

This is a pointer to any kind of data (the sequence of bytes).

There are two operators related to pointers / addresses:

- address-of operator (reference) **&** which returns an address (position in RAM memory) of the variable, or object
- indirection (dereference) operator ***** which returns the value of variable (or object) pointed by the address

```
int number ;
int *pointer ;
pointer = &number;           // assignment of the physical address
                                // of existing variable number

*pointer = 10;               // assignment of the value 10 to memory area
                                // indicated by the pointer
                                // ( here, it is equivalent of: number = 10 )
```

Pointer Arithmetic

The list of operation which can be performed on pointers:

- assignment (=)

ptr = pointer_to_variable_or_any_memory_area ;

(in case of non-compliance types, the explicit type conversion is necessary)

- comparison operators (==, !=, <, >, <=, >=):

ptr_1 == ptr_2 // checking if variables contain the same addresses

ptr_1 < ptr_2 // check if variable **ptr_1** contains prior / smaller address
// than the address contained in variable/pointer **ptr_2**

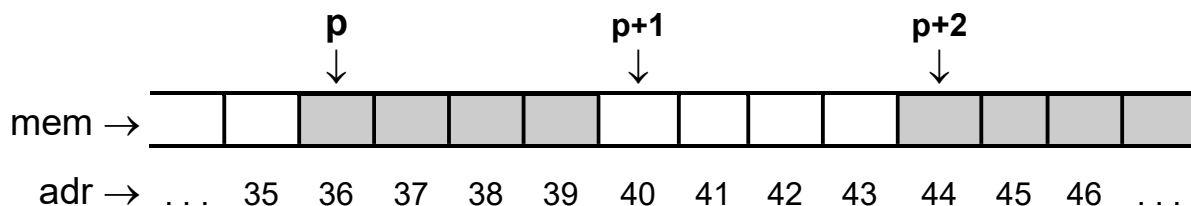
- operations of incrementing or decrementing the pointer (+, -, ++, --, +=, -=)
by an integer number (applied only for “defined” pointers)

→ incrementing (or decrementing) the pointer by integer value **N**
results in calculation of the address shifted by:

$$N * \text{sizeof}(\text{type_of_pointed_area})$$

bytes in the direction of increasing (decreasing) addresses.

np. **int *p;**



- subtracting of the two pointers (of the same type) → results in calculating the „distance” between two variables (areas in the memory) → number of objects
$$\text{distance} = (N * \text{sizeof}(\text{type_of_pointed_area}))$$

Examples of pointer variables:

int * ptr_number; // pointer to an integer number

int tab_A[10]; // 10-element array of integers

(identifier **tab_A** is the constant equal to the address
of the first element in this array
e.g. **tab_A == &(tab_A[0])**)

int * tab_B[10]; // 10-element array of pointers

int * (tab_C[10]); // the same as above

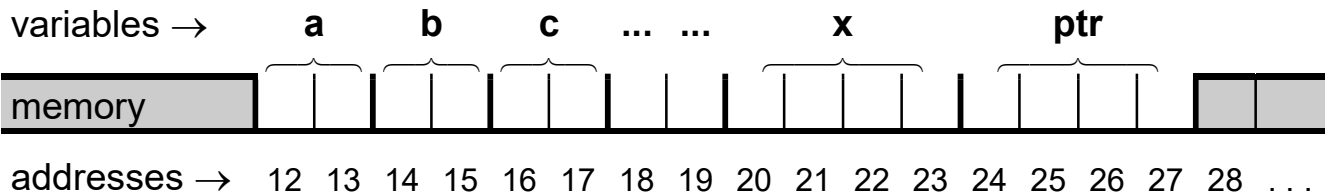
(int *) tab_D[10]; // as above

int (*tab_E)[10]; // pointer to 10-element array of integers

EXAMPLE-1: Access to variables using pointers

```
#include <stdio.h>
short a;      // the smaller representation of integer data: short ≡ short int
short b;      // by default it takes 16 bits (2 bytes)
short c;
float x;
short * ptr;
```

// example organization of memory



```
int main( )      // Illustration of accessing the variables,
{               // by using different addresses combinations
               // and pointer arithmetic

    // Printing the addresses of variables
    printf(" \n Address of variable A = %u ", (unsigned) &a );      // 4203212
    printf(" \n Address of variable B = %u ", (unsigned) &b );      // 4203214
    printf(" \n Address of variable C = %u ", (unsigned) &c );      // 4203216
    printf(" \n Address of variable X = %u ", (unsigned) &x );      // 4203220
    printf(" \n Address of variable PTR = %u ", (unsigned) &ptr );  // 4203224

    a = b = c = 0;
    printf( " \n A=%d, B=%d, C=%d ", a, b, c );

    b=10;
    printf( " \n A=%d, B=%d, C=%d ", a, b, c );

    ptr = &b;
    *ptr = 20;
    printf( " \n A=%d, B=%d, C=%d ", a, b, c );

    ptr = &a;
    *(ptr +1) = 30;
    printf( " \n A=%d, B=%d, C=%d ", a, b, c );

    *(&a + 1) = 40;
    printf( " \n A=%d, B=%d, C=%d ", a, b, c );

    *(&c - 1) = 50;
    printf( " \n A=%d, B=%d, C=%d ", a, b, c );

    * ( (short*)&x -3) = 60;
    printf( " \n A=%d, B=%d, C=%d ", a, b, c );

    *((short *)&x -1) -1) = 70;
    printf( " \n A=%d, B=%d, C=%d ", a, b, c );

    *((short *)&ptr -5) = 80;
    printf( " \n A=%d, B=%d, C=%d ", a, b, c );

    *((short*)&ptr -2) -1)= 90;
    printf( " \n A=%d, B=%d, C=%d ", a, b, c );

    getchar();
    return 0;
}
```

EXAMPLE-2: Accessing arrays with indexes and/or pointers

```
#include <stdio.h>
#define ARR_SIZE 10
```

// Common part of all examples on this page

```
int main(void)
```

```
{
```

```
    int arr[ ARR_SIZE ];
```

```
        // ← the “loop”, which is executing below listed operations:
```

```
        // ← reading a number into an array
```

```
    ...    // ← multiplying an array element by 2
```

```
        // ← display an element of the array
```

```
}
```

a) `int i;` *// accessing the elements of an array using the **index** operator*
`for(i = 0; i < ARR_SIZE; i++)`
`{`
 `scanf("%d", &arr[i]);`
 `arr[i] = 2 * arr[i];` *// arr[i] *= 2;*
 `printf("Array[%d] = %d \n", i+1 , arr[i]);`
`}`

b) `int i;` *// access by **index** and **dereference** operator*
`for(i = 0; i < ARR_SIZE; i++)`
`{`
 `scanf("%d", arr+i);` *// &*(arr+i) == arr+i*
 `*(arr+i) = 2 * *(arr+i);` *// *(arr+i) *= 2;*
 `printf("Array[%d] = %d \n", i+1 , *(arr+i));`
`}`

c) `int counter, *ptr;` *// access using a **pointer** and **dereference** operator, (+counter)*
`for(counter=0, ptr=arr; counter < ARR_SIZE; counter++, ptr++)`
`{`
 `scanf("%d", ptr);`
 `*ptr = 2* *ptr;` *// *ptr *= 2;*
 `printf("Array[%d] = %d \n", counter+1 , *ptr);`
`}`

d) `int *ptr;` *// access using only **pointers** (no additional counter)*
`for(ptr=arr; ptr < arr + ARR_SIZE; ptr++)`
`{` *// ptr < &arr[ARR_SIZE] ← pointer to „end of array”*
 `scanf("%d", ptr);`
 `*ptr *= 2;`
 `printf("Array [%d] = %d \n", ptr-arr+1 , *ptr);`
`}`