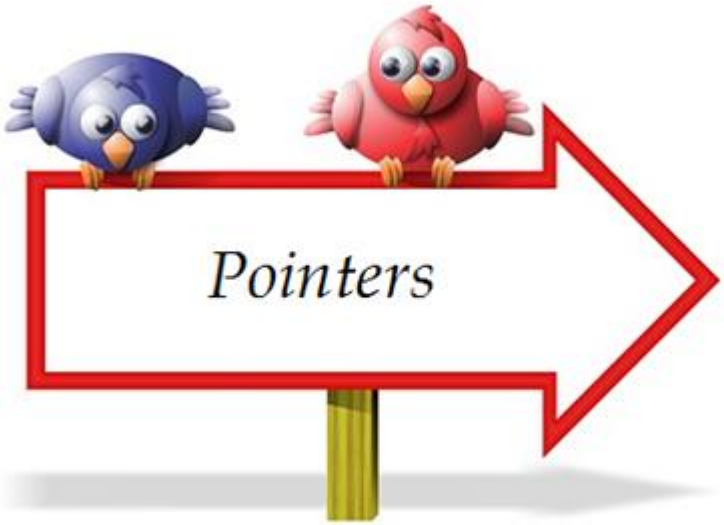
אוניברסיטת
אריאל
בשומרון

Pointers

Variable address

- During running, the compiler will assign memory to each variable – double (8 bytes), short (2 bytes), char (1 byte)
- Every memory cell owns a numerical value that refers to its exact location in the computer's memory
- The address is actually the address of the first byte (from all bytes) that the variable owns in the memory



Why Pointers?

אוניברסיטת
אריאל
בשומרון

- With pointers, you can create **dynamic data structures**.
- Instead of declaring your worst-case memory consumption up-front in an array, you instead **allocate** memory and use exact amount of required memory, with no waste.

Pointers Vs. Variables

- When you pass something to a function, that something is copied to the function's arguments.
- It is problematic especially when we work with the same variable in several functions or when we have huge variable size. In these cases, making copies is inefficient and memory consuming
- The original variables will not be changed in the caller function because their values are copied into the function's arguments.
- A function needs a way to refer to the original variables, not copies of their values. How can we refer to other variables in C? Using pointers.
- A pointer variable is a *special* variable in the sense that it is used to store an address of another variable. To differentiate it from other variables that do not store an address, we use * as a symbol in the declaration.
- For example, we live in a house and our house has an address, which helps other people to find our house. The same way the value of the variable is stored in a memory address, which helps the C program to find that value when it is needed.

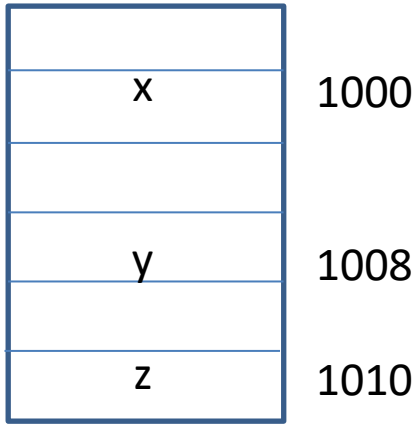
Pass by value (V) Vs. Reference (P)

אוניברסיטת
אריאל
בשומרון

- While pass by value is suitable in many cases, it has a couple of limitations.
  - ✓ Pass by value will make a copy of the argument into the function parameter. Therefore, if we do not want to use a vast number of arguments – it is not efficient as most of the CPU efforts spent on copying the variables
  - ✓ Pass by reference (like pointers) is much more flexible (dynamic allocation).
- When a parameter is passed by reference, the caller and the callee **use the same variable** for the parameter. If the callee modifies the parameter variable, the effect is visible to the caller's variable.
- This is important especially if we want to work in parallel – the same variable in several functions – better to work on the specific variable and not on its individual copies
- When a parameter is passed by value, the caller and callee have **two independent variables** with the same value. If the callee modifies the parameter variable, the effect is not visible to the caller.

Unary “&” operator

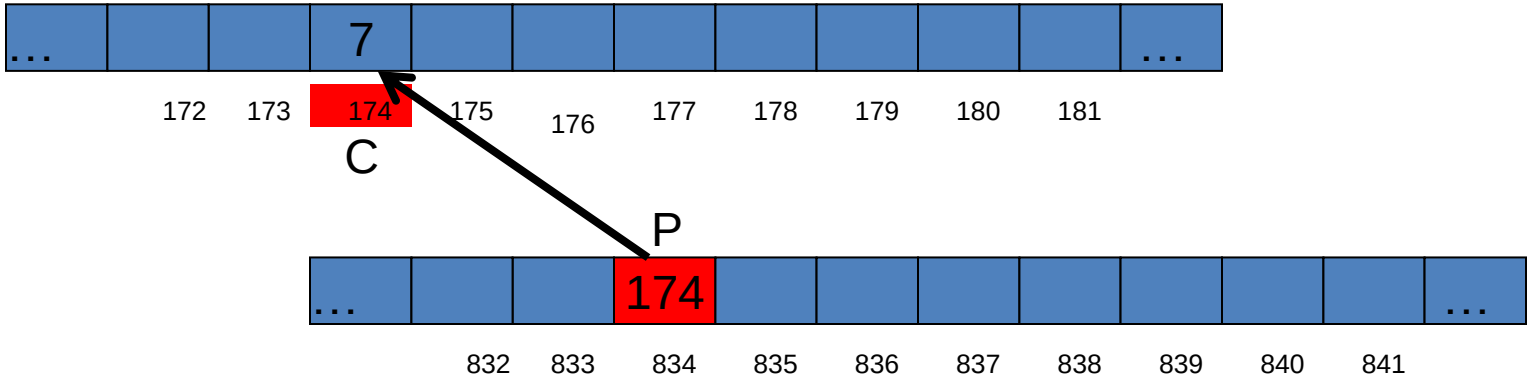
- If x is a variable, then &x is the address of the location of x that has been assigned in the memory
- The memory address includes value and type. The type is defined according to the type of the variable the pointer points to



- In our example –
  - &x=1000, &y=1008, &z=1010

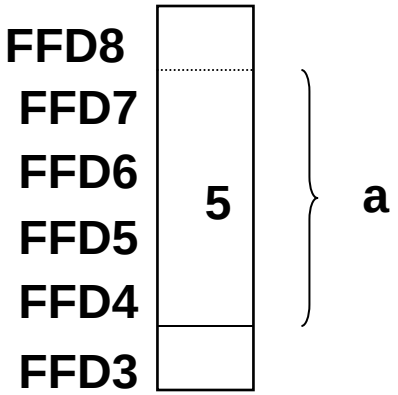
“*” operator – complementary to “&”

- Pointer contains the memory address of another variable.
- The operator * has two roles –
  - Declaration phase - we will add * on the left of the pointer
  - Running phase - access the variable that the pointer points to
- Example
  - int *<identifier>; // pointer to int variable
  - char *<identifier>; // pointer to char variable
- After declaring pointers, we will able to assign them variable addresses (P is a pointer to variable C - P=&C).



“&” and “*” - summary

- Every variable has an address and a value.



- C has two operators, **&** (address of) and * (indirection), used with pointers:
  - **&** - gives the address of a variable **(&a == FFD4).**
  - * - has 2 different meanings :
    - Upon declaration – ‘I am a pointer’ (example: **int *p;**).
    - After declaration – the variable content whose address is held by the pointer:

```
printf(“%d”, *ptr);    scanf(“%d”, &ptr);
```

“&” and “*” - summary

אוניברסיטת
אריאל
בשומרון

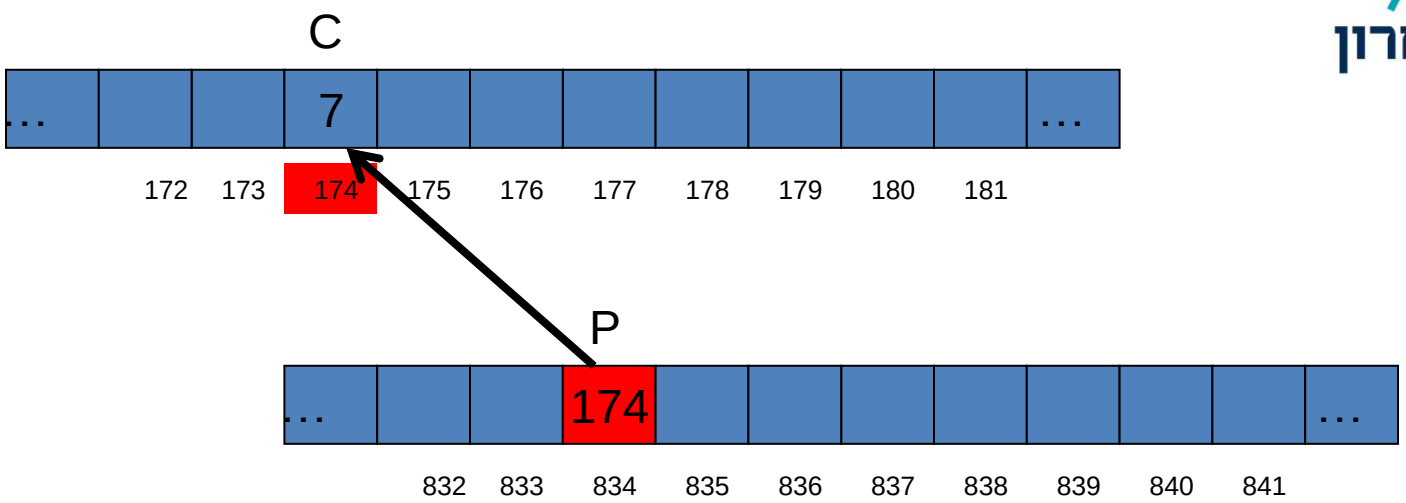
- Declaration
  - <type> *P1; <type> *P2;
  - P points to objects of type <type>

- Value content - by *Pointer
  - *P1 = x;

- Variable address - by &Variable

  &x - the variable’s address

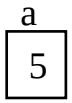
Using pointers



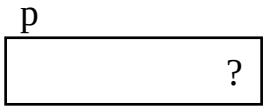

```
int C;

int *P;        //   Declare P as a pointer to int

P = &C;      // P point to the address of C

printf("%d", *P);     //   Prints out '7' – the content of the variable that P points to

*P = 5;      // puts 5 in C

printf("%d", C);      // Prints out '5'
```

More examples

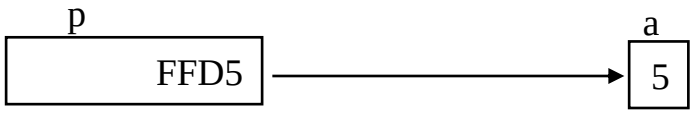
- **int a = 5;** **a** is an **int**, initialized as value 5. It has an address.



- **int *p;** **p** is a pointer to **int**, currently contains 'garbage'.



- **p = &a;** p holds the address of **a**. p points to a**.**



- ***p** is the content of the variable that p points to. In our case, ***p** is **a**.
- There are 2 ways to access **a** : via **a**, via **p**.

Assigning a Value to a Pointer

- Pointer type - a pointer can hold the address of a variable of the same type only as it was defined to point to.
- Pointer initialization - unless a value is assigned, a pointer will point to some garbage address by default.
- A null value is a special value that means the pointer is not pointing at anything. A pointer holding a null value is called a null pointer

```
int main(){

int i, *i_ptr1 = NULL, *i_ptr2 = NULL;

        double d, *d_ptr = NULL;

        i_ptr1 = &i; /* i_ptr1 holds the address of i */

        i_ptr2 = i_ptr1; /* now i_ptr2 also points to i */

        d_ptr = &d;  /* d_ptr holds the address of d */

Return 0;

}
```

Pointer arithmetic

אוניברסיטת
אריאל
בשומרון

- Pointers can be incremented and decremented.
- If p is a pointer to a particular type, p+1 points the **address of the next variable** of the same type.
- For example -

```
int * p ;

p = p +1;    means    p = p + sizeof(int);   // sizeof(int) =4

p=p + 4;     means     p + (4 * sizeof(int))
```

- p++, p+i, and p += i also make same sense.

Pointer arithmetic

אוניברסיטת
אריאל
בשומרון

```
ptr++;     ' Pointer moves to the next int position (as if it was an array)
++ptr;     ' Pointer moves to the next int position (as if it was an array)
++*ptr;    ' The value of ptr is incremented
++(*ptr);  ' The value of ptr is incremented
++*(ptr);  ' The value of ptr is incremented
*ptr++;    ' Pointer moves to the next int position (as if it was an array). But returns the old content
(*ptr)++;  ' The value of ptr is incremented
*(ptr)++;  ' Pointer moves to the next int position (as if it was an array). But returns the old content
*++ptr;    ' Pointer moves to the next int position, and then get's accessed, with your code, segfault
*(++ptr);  ' Pointer moves to the next int position, and then get's accessed, with your code, segfault
```

List of common errors

אוניברסיטת
אריאל
בשומרון

- It is impossible to define pointers to constants or expressions.

- It is also impossible to change a variable's address (because it is not for us to determine!), contrary to pointers' address.

- Therefore, the following are errors:
  - Pnt = &3; // constant
  - Pnt = &(k+5); // expression
  - Pnt = &(a==b); // expression
  - &a = &b; // changing address of a variable a
  - &a = 150; // changing address of a variable a

Pointers errors

- Omitting the pointer "*" character when declaring multiple pointers in same declaration
  - int* p1, p2;
- Using uninitialized pointers – will store an undefined random value – should be avoided

```
int main()
{
  int* p1; // p1 can point to any location in memory

  int n = *p1; // Error on debug builds

  printf("%d", n); // access violation on release builds
  return 0;
}
```

- Assigning a pointer to an uninitialized variable – will lead to serious logic errors in your code

Pointers errors – cont'd

אוניברסיטת
אריאל
בשומרון

- Incorrect syntax for incrementing dereferenced pointer values - if the intent is to increment a variable pointed to by a pointer, the following code fails to achieve that –

```
int main()
{
  int* p1; // create a pointer to an integer
  int m = 100;
  p1 = &m; // assign address of m to p1

  *p1++; // ERROR: we did not increment value of m

  printf("%d\n", *p1);
  printf("%d\n", m);

  return 0;
}
```

- Off by one errors when operating on C pointers

  Given a block of memory of SIZE objects pointed to by p, the last object in the block can be retrieved by using another pointer q and setting it to (p+SIZE-1) instead of (p+SIZE).

Pointers errors – cont'd

- Mismatching the type of pointer and type of underlying data - always use the appropriate pointer type for the data type. Consider the code below where a pointer to an integer is assigned to a short:

```
int main()
{
  int  num = 2147483647;
  int *pi = &num;
  short *ps = (short*)pi;
  printf("pi: %p  Value(16): %x  Value(10): %d\n", pi, *pi, *pi);
  printf("ps: %p  Value(16): %hx  Value(10): %hd\n", ps, (unsigned short)*ps, (unsigned
short)*ps);
}
```

- Comparing two pointers to determine object equality - in the code below, clearly the intent was to check if both strings are "Thunderbird". But, we ended up comparing the memory addresses (not the content) with the statement "str1 == str2". Here str1 and str2 are essentially pointers to different memory addresses which holds the same string.

Pointers errors – cont'd

אוניברסיטת
אריאל
בשומרון

```c
int main()
{
  char* str1 = (char*)malloc(strlen("Thunderbird") + 1);
  strcpy(str1, "Thunderbird");

  char* str2 = (char*)malloc(strlen("Thunderbird") + 1);
  strcpy(str2, "Thunderbird");

  if (str1 == str2)
  {
    printf("Two strings are equal\n");
  }
  else
  {
    printf("Two strings are NOT equal\n");
  }
}
```

More can be seen here –

https://www.acodersjourney.com/top-20-c-pointer-mistakes/

Pointers comparison

אוניברסיטת
אריאל
בשומרון

- We can compare pointers if they are pointing to the same array (even if they point to different locations of the array).

- Pointers can’t be multiplied or divided.

- Any pointer may be compared to NULL (zero).

Pointers types - NULL pointer

אוניברסיטת
אריאל
בשומרון

- It is always a good practice to assign a NULL value to a pointer variable in case you do not have an exact address to be assigned. This is done at the time of variable declaration. A pointer that is assigned NULL is called a **null** pointer.
- Suppose we searched the array and didn't find anything – what should we return in case of failure?
  - NULL pointer is often used to indicate failure
  - NULL pointer is an 'empty' pointer that points to nothing.
  - Its address is 0.
  - If a pointer p is set to NULL, trying to access *p (a variable that p should point to) results in an error.

Pointers types - void * pointer

אוניברסיטת
אריאל
בשומרון

- void* p pointer is a pointer that has no associated data type with it.
- A void pointer can hold address of any type and can be typcasted to any type.
- int j;

  int* p= &j;

  void* q= p;

  p = (int*)q ;

Example: pointer.c

אוניברסיטת
אריאל
בשומרון

```
/* pointer.c
This program illustrates pointers (declaration, initialization and
indirection) */

#include <stdio.h>

void main(void)
{
        int   i = 1;
        char  c = 'a';
        int *Pi = NULL; /* pointer to int, initialized with NULL. */
        char *Pc = NULL;  /* pointer to char, initialized with NULL.*/

        printf("Address of i (&i): %p. Value of i (i): %d\n" , &i, i);
        printf("Address of c (&c): %p. Value of c (c): %c\n" ,&c, c);
        putchar('\n');
```

Example: pointer.c – cont'd

אוניברסיטת
אריאל
בשומרון

```
22          Pi = &i; /* Pi holds the address of i. i is the value of Pi */
23          Pc = &c; /* Pc holds the address of c. c is the value of Pc */
24          *Pi = 2; /* *Pi is actually i. i was assigned the value 2 */
25          *Pc ='b'; /* *Pc is actually c. c was assigned the value 'b'*/
26
27          ++i; /* i is now 3. */
28          ++c; /* c is now 'c' */
29          putchar('\n');
30          printf("Address of i (&i): %p. Value of i (i): %d\n",&i, i);
31          printf("Address of c (&c): %p. Value of c (c): %c\n",&c, c);
32          putchar('\n');
33          printf("Address of Pi (&Pi): %p. Value of Pi (pi):%p\n", &Pi, Pi);
34          printf("Address of Pc (&Pc): %p. Value of Pc (pc): %p\n", &Pc, Pc);
35          putchar('\n');
36
37           /* *Pi is actually i and *Pc is actually c */
38           printf("The value of Pi (*Pi) : %d\n", *Pi);
39           printf("The value of Pc (*Pc) : %c\n", *Pc);
40  }
```

אוניברסיטת
אריאל
בשומרון

Pointers--Arrays

Pointer-Array equivalence

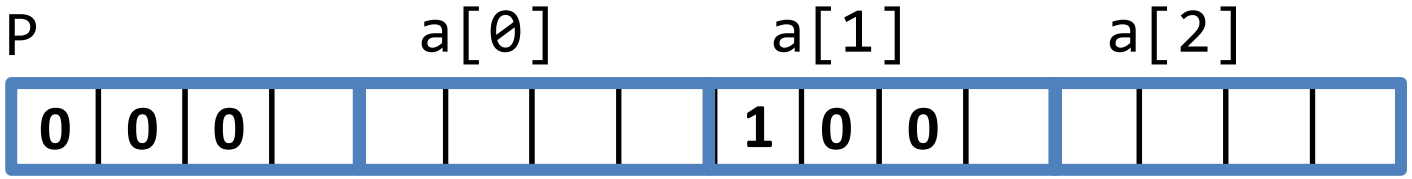
אוניברסיטת
אריאל
בשומרון

- Arrays are actually a kind of **constant** pointers!
- When an array is defined, a fixed amount of memory is allocated (the size of the array).
- The array variable is set to point the beginning of a memory segment (a[0]).
- A pointer is also set to point the beginning of a memory segment (a[0]).
- Contrary to variable/array – pointer is dynamically allocated
- Assume:

```
int S[10], *P;
P=S;   // Both P and S are now pointing to S[0]
```

Pointers & Arrays

```
int *p;
int a[3];
p = &a[0]; // same as p = a
*(p+1) = 1; // assignment to a[1]!
```



Pointers & Arrays

אוניברסיטת אריאל בשומרון

- Arrays can be treated as address of the first member.
- int *p;

  int a[4];

  p = a; // p is pointer to a (array cell) - same as p = &a[0];

```
// Points to 0th element of the arr.
p = arr;

// Points to the whole array arr.
ptr = &arr;
```

  p[1] = 102; // same as *(p+1)=102;

  *(a+1) = 102; // same as prev. line

  p++; // p == a+1 == &a[1]

  a = p; // illegal (as opposed to p=a), as a is array and p is pointer

  a++; // illegal – array address can NOT be changed

Pointers and Arrays

אוניברסיטת
אריאל
בשומרון

- The address of the element with index 0 will be held by ptr;

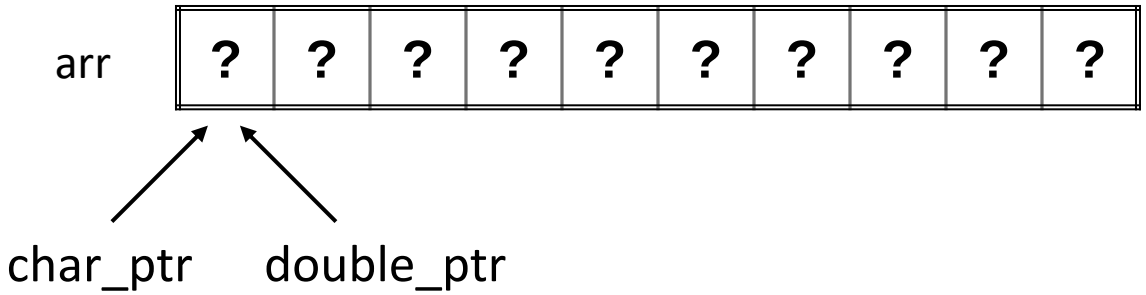
  In other words, ptr will point to the beginning of the array.

```
int a[100];

int *ptr = a;

printf("%p %p", &a, ptr); /*the same address*/
```

- In order to scan arrays with pointers we must know :
  - Memory layout of arrays.
  - Pointers arithmetic.

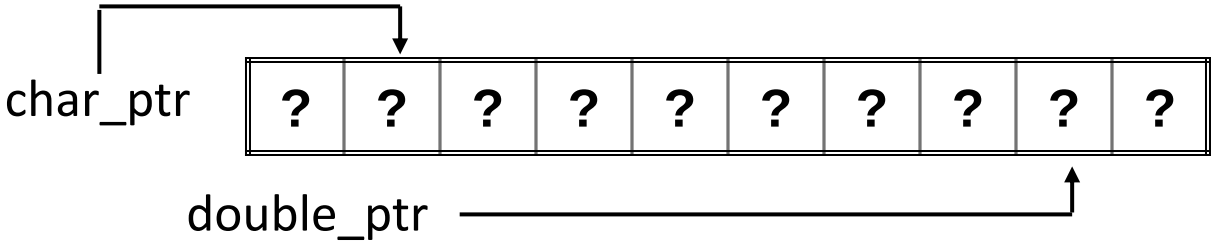
Pointers and Arrays – Arithmetic

```
char arr[10];
char *char_ptr = arr; /* will scan arr successfully */
double *double_ptr = arr; /* will not work  well … */
```




```
++char_ptr;    // adds 1 to char_ptr
++double_ptr; // adds 8 to double_ptr
```

- When we add/subtract a number to/from a pointer, the computer adds/subtracts that number multiplied by sizeof the pointers type.



Pointers and Arrays – cont'd

אוניברסיטת
אריאל
בשומרון

- When scanning arrays with indexes we should write:

```
float arr[5];

arr[2] = 3.3;
```

- The compiler sees :

```
float arr[5];

*( arr[0] +  2 * sizeof(float) ) = 3.3;
```

- It is the programmer's responsibility to keep the pointer within the borders of the array, just like when scanning arrays with indexes.

Pointers and Arrays – cont'd

אוניברסיטת
אריאל
בשומרון

- Two pointers to objects of the same type, in the same array, can be subtracted. The result is the number of indexes (not the number of bytes !) between them.

- If, for instance, p1 points to the beginning of a string and p2 points to the null terminator of this string, then p2 - p1 gives the string's length.

Pointer Vs. Array arithmetic - Summary

אוניברסיטת
אריאל
בשומרון

- Arrays are kind of pointers !!
- Unlike pointers, the value of an array address cannot be changed.

  We can write :

```
int a[10];
int * p;
p = a;
```

- But the array's address is constant !!
- If p and q are pointers to elements in an array, q-p yields the number of elements between p and q.

אוניברסיטת
אריאל
בשומרון

Pointers--Functions

Pointers and Functions

- How can we change the value of a parameter sent to a function ?
- How can a function return a lot of information to the calling function ?… After all, there is only one return value…
- The solution is – sending the address of the variable causes the variable to be sent by reference rather than by value. The function will receive a pointer to the variable and will be able to change it.

```
int num, *ptr = &num;
```

```
func(&num);
func(ptr);
```

The same

Pointers and Functions – cont'd

אוניברסיטת
אריאל
בשומרון

- If we pass the variable i to a function, we actually pass the value of i.
- If we pass the pointer to i, we actually pass a address of i.
- The pointer is passed by value, of course, there is no other way BUT the value that is passed is the address of i, so the pointer that the function received as a parameter actually points to i.
- The function can change i via the pointer that it received as a parameter.
- When we passed an array to a function we actually passed the address of the array (where it begins) - a pointer.

Example: arrange_arr.c

אוניברסיטת
אריאל
בשומרון

- Write a program that sorts an array of characters so that the characters will be in increasing order. Pls use pointers only.

Solution: arrange_arr.c

```
/* This program sorts an array of characters so that the characters
will be in increasing order. It uses pointers only, no indexes.*/

#include  <stdio.h>

void print_arr(const char *arr, int len);
void rearrange_arr(char *arr, int len);
void swap(char x[],  char y[]);
 void main(void)
{
        char arr[] = {'p', 'o', 'p', 'y', 'f', 'l'};
        printf("Before rearranging the elements of the array :\n");
        print_arr(arr, length);
        rearrange_arr(arr, length);
        printf("After rearranging the elements of the array :\n");
        print_arr(arr, length);
}
```

Solution : arrange_arr.c – cont'd

אוניברסיטת
אריאל
בשומרון

```
/* This function rearranges the array. It uses bubble sort. (The highest value 'climbs' up to the top) */
void rearrange_arr(char arr[], int len)
{
        char *p1 = NULL,*p2 = NULL;    /* pointers for scanning arr */
        char *p_end = &arr[len -1];    /* pointer to the last element of arr */

        p1 = p2 = arr;  /* the 2 pointers point to the first element of arr */
        p1++;  /* one pointer points to one element forward than the other */

        while(p1 <= p_end) /* the following lines perform a bubble sort on the array */

        {
                p2 = arr;
                while(p2 < p1)
                {
                        if(*p1 < *p2)
                                swap(p1, p2);
                        ++p2;
                }
                        ++p1;
        }
}
```

Solution : arrange_arr.c – cont'd

```
/* this function prints the array */
void print_arr(const char *arr, int len)
{
        char *p_start = arr; /*pointer to the first element of arr*/
        char *p_end = &arr[len-1]; /* pointer to the last element of arr */

        /* loop until p_start points to the last element of arr(inclusive) */
        while(p_start <= p_end)
        {
                printf("%c", *p_start); /*print the char that p_start points to*/
                ++p_start;  /*p_start points to the next element*/
        }
        printf("\n\n");
}

/* this function swaps the values of two pointers */
void  swap(char x[],  char y[])
{
        char tmp;

        tmp = *x;
        *x = *y;
        *y = tmp;
}
```

Example: char-scan.c

- Write a program to scan for a certain character in a string, using pointers.

Summary

אוניברסיטת
אריאל
בשומרון

- A pointer is a variable that holds the address of another variable.
- A pointer can be assigned to the address of a variable of the same type as it was defined to point to.
- If the pointer is not pointing to a variable, make it point to NULL.
- Addition/subtraction of a whole number to/from a pointer is allowed. The number added/subtracted is first multiplied (by the computer) by the size of what the pointer was defined to point to.

Summary – cont'd

אוניברסיטת
אריאל
בשומרון

- Subtraction (but not addition) of pointers gives the number of indexes between them.

- Pointers to the same type may be compared. Any pointer may be compared with NULL.

- The name of an array is actually a pointer to its element #0. Arrays are scanned faster using pointers than using indexes.

- Pointers enable us to pass variables to functions by reference.

TA exercises

- Write a program that:

  Requests and reads a string.

  Requests and reads a character and informs whether the character occurs in the string.

  Requests and reads another character and informs whether the character occurs in the string.

  If both characters occur in the string, print a message to inform which one of them precedes the other, and the distance between the first occurrence of each of them.

  Use library function **strchr( ).**

- Write a function that copies the first n elements of one array to another one, using pointers. Write **main( )** to call the function.

TA exercises – cont'd

אוניברסיטת
אריאל
בשומרון

- Write a version of **strcat()** that works with pointers instead of array's indexes.
- Write a version of **strcmp(char *str1, char *str2)** that works with pointers, which compares (lexicographically) two strings and returns int value as follows:

<0	If str1 < str2
=0	If str1 ==str2
>0	If str1 >str2

- Write a function **chs(int *)** that changes the sign of its argument. (Note that return type is void).

- Write a function: **int index(char *src, char *str)** that finds the first occurrence of the string **str** in the string **src**. The function returns the position in **src** where **str** was found. If not found it returns some negative number.

Pointers to pointers

- Pointer is a variable type, it also has an address:
- int main()

```
{
  int n = 17;
  int *p = &n;
  int **p2 = &p;
  printf("the address of p2 is %p \n",&p2);
  printf("the address of p is %p \n",p2);
  printf("the address of n is %p \n",*p2);
  printf("the value of n is %d \n",**p2);
  return 0;
}
```

אוניברסיטת
אריאל
בשומרון

Size_t

Size_t – biggest unsigned type

אוניברסיטת
אריאל
בשומרון

- An unsigned data type, defined in stddef.h
- When you use a size_t object, you have to make sure that in all the contexts it is used, including arithmetic, you want non-negative values. For example, it will be good for length.
- Make sure by "for (size_t i=ARR_SIZE-1; i-- > 0;)"
- It is guaranteed to be big enough to contain the size of the biggest object the host system can handle. Basically, the maximum permissible size is dependent on the compiler; if the compiler is 32 bit then it is simply an unsigned int but if the compiler is 64 bit then it would be a typedef for unsigned long long.
- It's a type which is used to represent the size of objects in bytes and is therefore used as the return type by the sizeof keyword operator.

Example – size_t in functions

אוניברסיטת
אריאל
בשומרון

```
int sum (int arr[])
{
        size_t i;
        int sum = 0;

        for (i=0; i<sizeof(arr)/sizeof(arr[0]); ++i)
        {
                sum += arr[i];
        }
        return sum;
}
```

Const

“Const” Usage

אוניברסיטת
אריאל
בשומרון

The const declaration should be used in the definition of a function’s arguments, to indicate it would not change them:

```
size_t strlen(const char []);
```

Why use? (This is not a recommendation but a **must**)

- clearer code
- avoids errors
- part of the interfaces you define!
- can be used with const objects

C's "const"

- C's "const" is a qualifier that can be applied to the declaration of any variable to specify that its value will not be changed.
- Examples -

```
const double E = 2.71828;
E= 3.14;                    // compile error!
```

```
const int arr[] = {1,2};
arr[1] = 1;                 // compile error!
```

Const and Pointer's Syntax

אוניברסיטת
אריאל
בשומרון

- Const protects his left side, unless there is nothing to his left and only then it protects his right side (examples next slide).

```
const int arr[] = {1,2};
int* arr_ptr = (int*)arr;

arr [0] = 3;          // error
```

Const and Pointer's Syntax

אוניברסיטת
אריאל
בשומרון

```
// Cannot change the int that is pointed by p

const int * p = arr;
int const * p = arr;

// Cannot change the address stored in p
int * const p = arr;

// Cannot change the address stored in p and the int pointed by p using p
int const * const p = arr;
```

Const and Pointer's Syntax

אוניברסיטת
אריאל
בשומרון

```
// Both, cannot change the int pointed by p

const int * p = arr;
int const * p = arr;
```

Very Useful

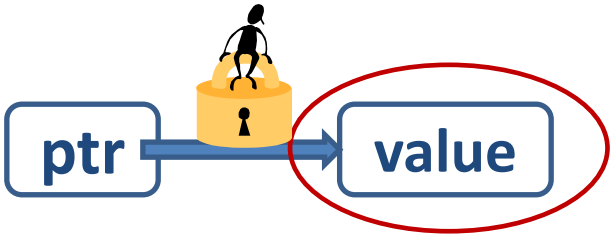
```
// Cannot change the address stored in p
int * const p = arr;

// Cannot change the address stored in p
// and the int pointed by p using p
int const * const p = arr;
```

Never saw it being used

C's "const"

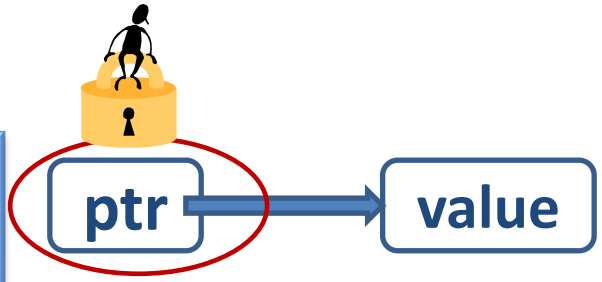
- Do not confuse what the "const" declaration "protects"!
- A pointer to a **const variable**:

```
int arr[] = {1,2,3};
int const * p = arr;
p[1] = 1;          // illegal!
*(p+1) = 1;        // illegal!
p = NULL;          // legal
```



- A **const pointer** to a variable:

```
int arr[] = {1,2,3};
int * const const_p = arr;
const_p[1] = 0;    // legal!
const_p = NULL;    // illegal!
```



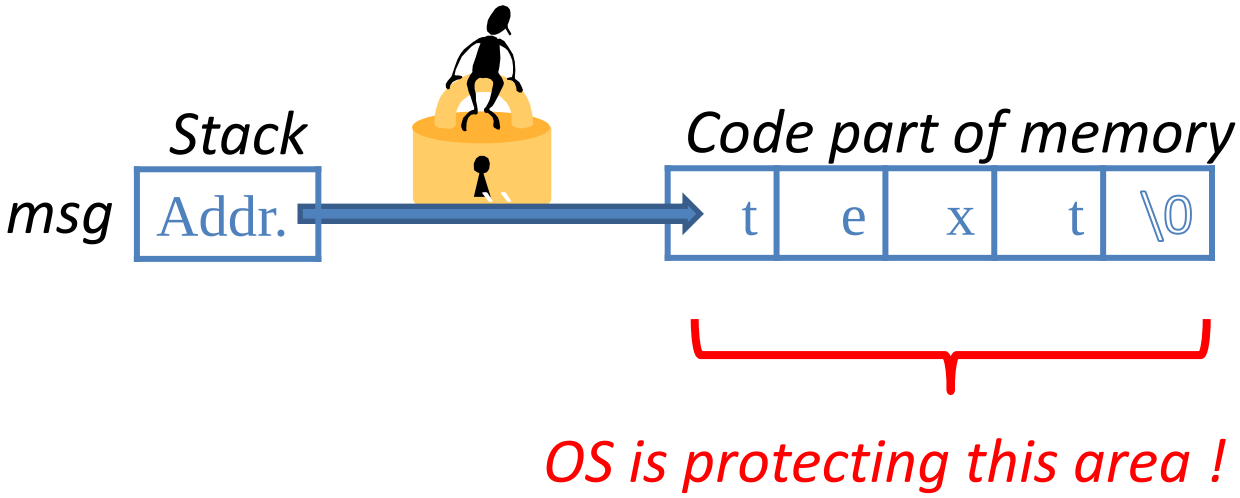
אוניברסיטת
אריאל
בשומרון

String literals

String literals ("")

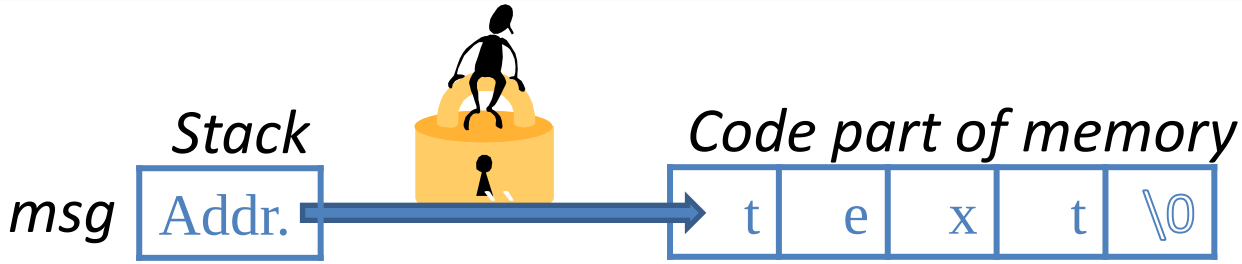
- When working with pointers, C string literals ("") are written in the read-only code segment (part) of the memory. Thus, you can't change them!

```
const char* msg = "text";
msg[0] = 't';          // error
```

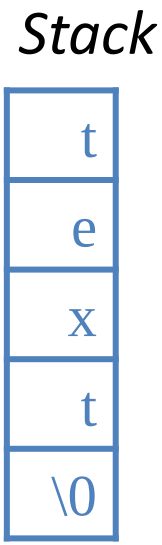


Difference between initializing a pointer and an array with string literals

```
char* msg = "text";
// msg is a pointer that points to a
memory that is in the code part
```




```
char msg2[] = "text";
// msg2 is an array of chars that
are on the stack
```



Difference between initializing a pointer and an array with string literals

אוניברסיטת
אריאל
בשומרון

```
char* msg = "text";
msg[0]= 'n'; // Seg fault (trying to change a read-
only memory slot)

char msg2[] = "text";
msg2[0]= 'n';       // ok - changing what is written
                       in the stack part of the memory
```

Strings literals - example

אוניברסיטת
אריאל
בשומרון

```
char txt1[] = "text";
char* txt2 = "text";

int i = strlen(txt1); // i = 4, same for strlen(txt2)

txt1[0] = ’n’; // now txt1="next“
*txt2 = ’n’; // illegal!
txt2 = txt1; // legal. now txt2 points to the same string.
txt1 = txt2; // illegal!

if (! (strcmp(txt2,"next")) //This condition is now true
{
  ...
```

C Strings Manipulation

אוניברסיטת
אריאל
בשומרון

- To manipulate a single character - use the functions defined in *ctype.h*.
- Please see *https://www.tutorialspoint.com/c_standard_library/ctype_h.htm* for more details

```
#include <ctype.h>
char c = 'A';
isalpha(c); isupper(c); islower(c);   …
```

- Manipulation of Strings is done by including the *string.h* header file

```
// copy a string
char* strcpy(char * dest, const char* src);
// append a string
char* strcat(char * dest, const char* src);
```

C Strings Manipulation

אוניברסיטת
אריאל
בשומרון

```
// compare two strings.
// when str1 < str2 lexicographically return < 0
// when str1 > str2 lexicographically return > 0
// when identical return 0
int strcmp(const char * str1, const char* str2);

// return strings length, not including the \0!!
size_t strlen(const char * str);

// Other functions:
strncpy(),strncat(),strncmp()   …
```

C Strings Functions

אוניברסיטת
אריאל
בשומרון

- An “array” version of strcpy():

```
void strcmp(char dest, const char src)
{
   size_t i = 0;
   while ((dest[i] = src[i])!= '\0')) i++;
}
```

C Strings Functions

אוניברסיטת
אריאל
בשומרון

- A “pointers” version of strcpy():

```
void strcmp(char * dest, const char* src)
{
   while ((*dest = *src)!= '\0'))
   {
      dest++;
      src++;
   }
}
```