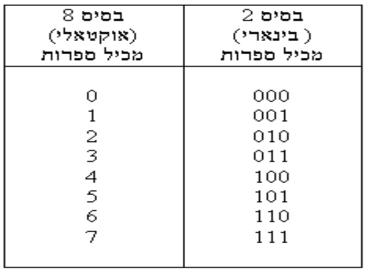
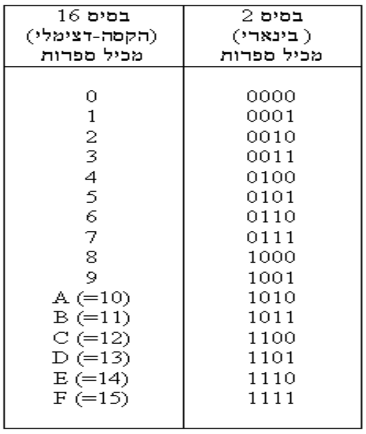
Bases:

בסיס 16 (הקסה-דצימלי) מכיל ספרות	בסיס 10 (עשרוני) מכיל ספרות	בסיס 8 (אוקטאלי) מכיל ספרות	בסיס 4 מכיל ספרות	בסיס 2 (בינארי) מכיל ספרות
0	0	0	0	0
1	1	1	1	1
2	2	2	2	
3	3	3	3	
4	4	4		
5	5	5		
6	6	6		
7	7	7		
8	8			
9	9			
A (=10)				
B (=11)				
C (=12)				
D (=13)				
E (=14)				
F (=15)				



בסיס 8 (אוקטאלי) מכיל ספרות	בסיס 2 (בינארי) מכיל ספרות
0	000
1	001
2	010
3	011
4	100
5	101
6	110
7	111



בסיס 16 (הקסה-דצימלי) מכיל ספרות	בסיס 2 (בינארי) מכיל ספרות
0	0000
1	0001
2	0010
3	0011
4	0100
5	0101
6	0110
7	0111
8	1000
9	1001
A (=10)	1010
B (=11)	1011
C (=12)	1100
D (=13)	1101
E (=14)	1110
F (=15)	1111

- מעבר מבסיס 2 (בינארי) לבסיס 10 (עשרוני): $0111_2 = 0*2^3 + 1*2^2 + 1*2^1 + 1*2^0 = 7_{10}$

- מעבר מבסיס 8 (אוקטאלי) לבסיס 10 (עשרוני): $146_8=1*8^2 + 4*8^{1+}6*8^0 = 102_{10}$

מעבר מבסיס 16 (הקסה-דצימלי) לבסיס 10(עשרוני):
$1F3A_{16} = 1*16^3 + 15*16^2 + 3*16^1 + 10*16^0 = 7994_{10}$

- **יהיה X מספר עשרוני. יהיה n הבסיס עליו רוצים להעביר את X.**

1. חלק את X ב n ורשום את השארית

2. חלק את המנה שהתקבלה שוב ב n ורשום את השארית החדשה

3. חזור על השלב 2 עד לקבלת מנה 0.

4. המספר בבסיס n הוא סדרת השאריות בסדר כתיבה הפוך

Programing computer for the test-

Getting input from user:

Example:

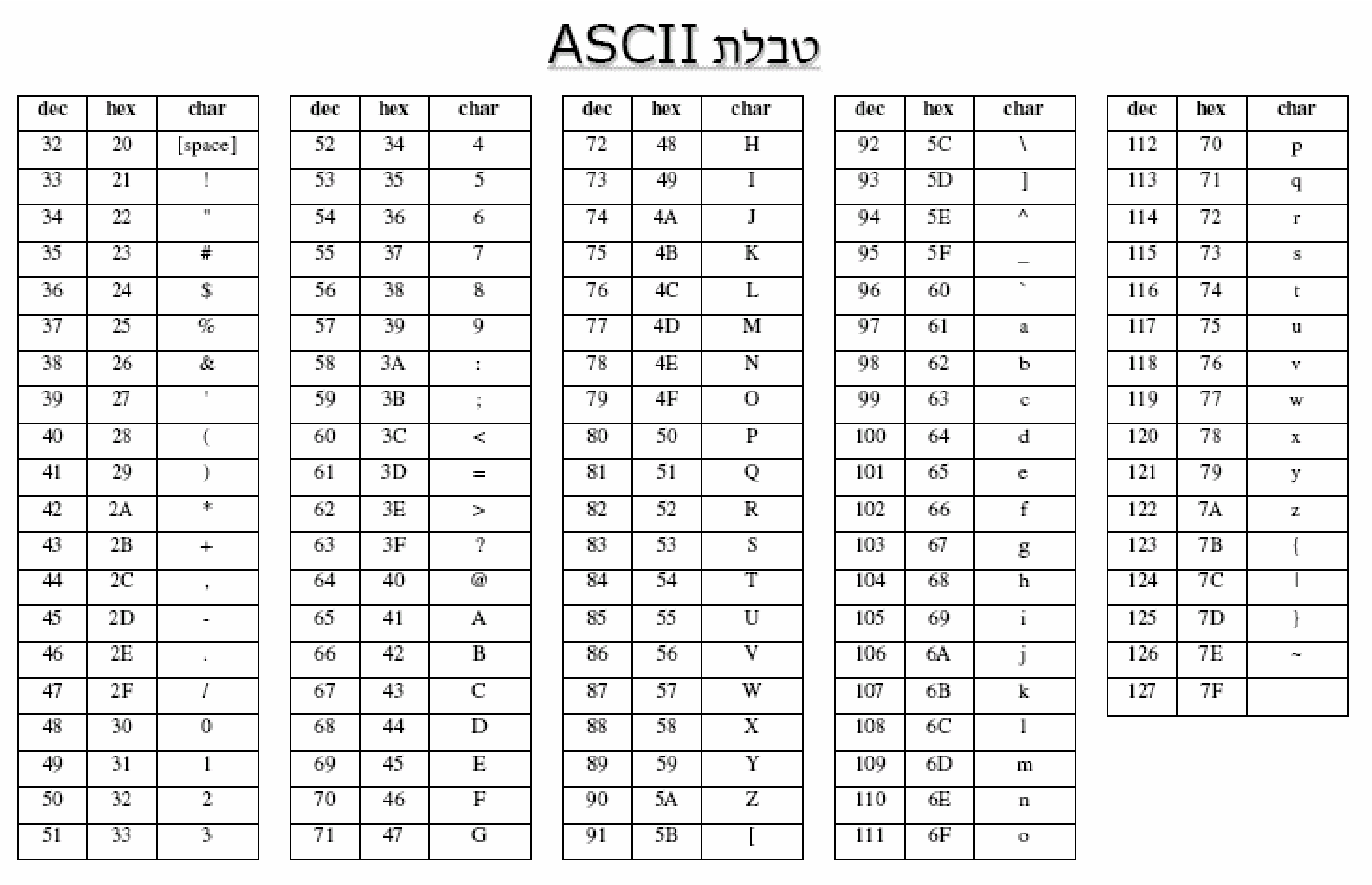
```
scanf("%d", &a);
printf("You entered: %d\n", a);
```

Some many rules :

- `%d` - int (same as %i)
- `%ld` - long int (same as %li)
- `%f` - float
- `%lf` - double[1]
- `%c` - char
- `%s` - string
- `%x` - hexadecima

Data Types:

Data Type Keyword	Size in bytes	Range
unsigned char	1 byte	0 to 255
char	1 byte	-128 to 127
unsigned short int	2 bytes	0 to 65535
short	2 bytes	-32768 to 32767
unsigned int	4 bytes	0 to 65535 (if 2 bytes)
int	4 bytes	-32768 to 32767 (if 2 bytes)
unsigned long	4 bytes	0 to 4294967295
long	4 bytes	-2147483648 to 2147483647
float	4 bytes	3.4e-38 to 3.4e+38
double	8 bytes	1.7e-308 to 1.7e+308
long double	10 bytes	3.4e-4932 to 1.1e+4932



טבלת ASCII

dec	hex	char	dec	hex	char	dec	hex	char	dec	hex	char	dec	hex	char
32	20	[space]	52	34	4	72	48	H	92	5C	\	112	70	p
33	21	!	53	35	5	73	49	I	93	5D	]	113	71	q
34	22	"	54	36	6	74	4A	J	94	5E	^	114	72	r
35	23	#	55	37	7	75	4B	K	95	5F	_	115	73	s
36	24	$	56	38	8	76	4C	L	96	60	`	116	74	t
37	25	%	57	39	9	77	4D	M	97	61	a	117	75	u
38	26	&	58	3A	:	78	4E	N	98	62	b	118	76	v
39	27	'	59	3B	;	79	4F	O	99	63	c	119	77	w
40	28	(	60	3C	<	80	50	P	100	64	d	120	78	x
41	29	)	61	3D	=	81	51	Q	101	65	e	121	79	y
42	2A	*	62	3E	>	82	52	R	102	66	f	122	7A	z
43	2B	+	63	3F	?	83	53	S	103	67	g	123	7B	{
44	2C	,	64	40	@	84	54	T	104	68	h	124	7C	|
45	2D	-	65	41	A	85	55	U	105	69	i	125	7D	}
46	2E	.	66	42	B	86	56	V	106	6A	j	126	7E	~
47	2F	/	67	43	C	87	57	W	107	6B	k	127	7F	
48	30	0	68	44	D	88	58	X	108	6C	l			
49	31	1	69	45	E	89	59	Y	109	6D	m			
50	32	2	70	46	F	90	5A	Z	110	6E	n			
51	33	3	71	47	G	91	5B	[	111	6F	o			

Saved words:

auto
break
case
char
const
continue
default
do
double
else
enum
extern
float
for
goto
if
int
long
register
return
short
signed
sizeof
static
struct
switch
typedef
union
unsigned
void
volatile
while
_Packed

Const words:

Difference between #define and const:

1) #define is pre-processor directive while const is a keyword

2) #define is not scope controlled whereas const is scope controlled

Example

```
 static const int xn = 5;
#define XN 5
enum { xn = 5, xm=6 };
```

Conversions and castings:

There are some many ways:

- **double d = 12.5;**
  int a = 3, b = 4;
  a = b + d;
- **a= (Double) a;**

SizeOf:

Example:

```
 int arr[] = { 1, 2, 3, 4, 7, 98, 0, 12, 35, 99, 14 };
 printf("Number of elements:%lu ", sizeof(arr) / sizeof(arr[0]));


printf("%lu\n", sizeof(char));
printf("%lu\n", sizeof(int));
printf("%lu\n", sizeof(float));
printf("%lu", sizeof(double));
```

Operators:

types:

+	plus (unary and binary)
-	minus (unary and binary)
*	multiplication
/	division
%	remainder of division (modulus)

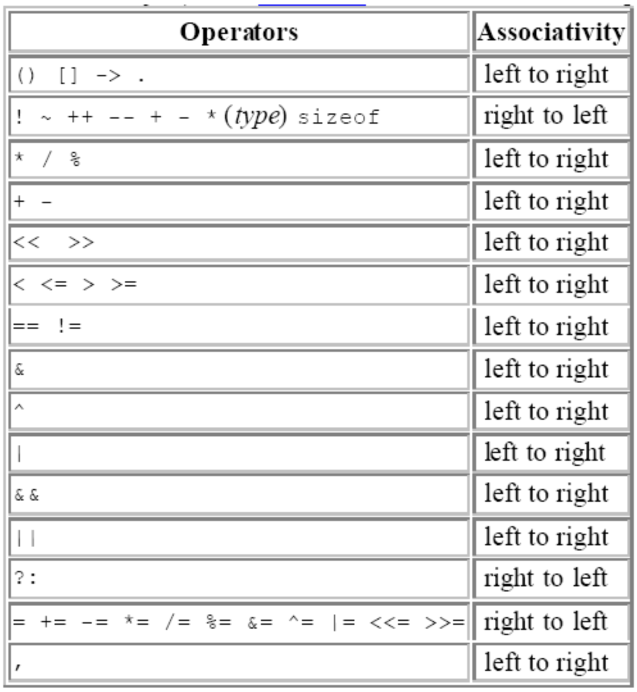
Relational operators:

- == Equal
- != Different (not equal)
- > Greater than
- < Less than
- >= Greater or equal than
- <= Less or equal than

Logical operators:

a	b	a || b	a && b	!a
T	T	T	T	F
T	F	T	F	F
F	T	T	F	T
F	F	F	F	T

Operation priority:



Operators	Associativity
() [] -> .	left to right
! ~ ++ -- + - * (*type*) sizeof	right to left
* / %	left to right
+ -	left to right
<< >>	left to right
< <= > >=	left to right
== !=	left to right
&	left to right
^	left to right
|	left to right
&&	left to right
||	left to right
?:	right to left
= += -= *= /= %= &= ^= |= <<= >>=	right to left
,	left to right

Control flow:

if else statment:

```
if (test expression) {
    // statements to be executed if the test expression is true
}
else {
    // statements to be executed if the test expression is false
}
```

Switch case statment:

```
switch (variable or an integer expression)
{
     case constant:
     //C Statements
     ;
     case constant:
     //C Statements
     ;
     default:
     //C Statements
     ;
}
```

Loops - for:

```
for (initial value; condition; incrementation or decrementation )
{
  statements;
}
```

Loops- while :

```
while (condition) {
             statements;
}
```

Loops-Do while:

```
do {
  statements
} while (expression);
```

Break statement:

to exit the statement:

```
while (num > 0) {
  if (num == 3)
    break;
  printf("%d\n", num);
  num--;}
```

Continue statement:

To continue the statement from the beginning.

```
while (nb > 0) {
  nb--;
  if (nb == 5)
    continue;
 printf("%d\n", nb);}
```

Functions:

Decleration:

Return_type Function_name (parameters);

Definition :

Return value;

Parameters :

Function(int x, char 'c');

Example:

```
#include<stdio.h>
// function prototype, also called function declaration
float square ( float x );
// main function, program starts from here

int main( )
{
   float m, n ;
   printf ( "\nEnter some number for finding square \n");
   scanf ( "%f", &m ) ;
   // function call
   n = square ( m ) ;
   printf ( "\nSquare of the given number %f is %f",m,n );
}

float square ( float x )   // function definition
{
   float p ;
   p = x * x ;
   return ( p ) ;
}
```

Global :

- Scope: from the line it is defined up to the end of the file.
- Lifetime: during program execution.
- Initialization: if the global variable is not initialized, its default initialization is 0.

```
/* global variable declaration */
int g = 20;

int main () {

  /* local variable declaration */
  int g = 10;

  printf ("value of g = %d\n",  g);

  return 0;
}
```

Static :

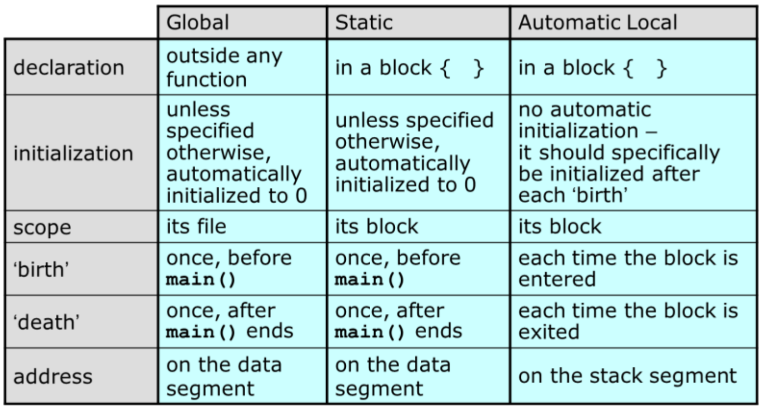
- Scope: from the line it is defined up to the end of its block.
- Lifetime: during program execution.
- Initialization: if the static variable is not initialized, its default initialization is 0.

```
#include<stdio.h>
int fun()
{
  static int count = 0;
  count++;
  return count;
}

int main()
{
  printf("%d ", fun());
  printf("%d ", fun());
  return 0;
}
```

Output:

```
1 2
```



	Global	Static	Automatic Local
declaration	outside any function	in a block { }	in a block { }
initialization	unless specified otherwise, automatically initialized to 0	unless specified otherwise, automatically initialized to 0	no automatic initialization – it should specifically be initialized after each 'birth'
scope	its file	its block	its block
'birth'	once, before **main()**	once, before **main()**	each time the block is entered
'death'	once, after **main()** ends	once, after **main()** ends	each time the block is exited
address	on the data segment	on the data segment	on the stack segment

Array

syntax-

int ages[10];

ages[3] = 23;

int matrix[4][3] = {1, 2, 3, 4, 5, 6, 7, 8, 9};

array as a parameter :

int arr[4]={1,2,3,4}, size=3;

initialize(arr, size, arr[0]);

Matrix:

```
Enter value for disp[0][0]:1
Enter value for disp[0][1]:2
Enter value for disp[0][2]:3
Enter value for disp[1][0]:4
Enter value for disp[1][1]:5
Enter value for disp[1][2]:6
Two Dimensional array elements:
1 2 3
4 5 6
```

```
//variables for ex3
        int mat1[MATRIX_SIZE][MATRIX_SIZE] = { {1, 2, 3, 4}, { 5, 6, 7, 8 }, {
9, 10, 11, 12 }, { 13, 14, 15, 16 } };
        int mat2[MATRIX_SIZE][MATRIX_SIZE] = { { 1, 2, 3, 4 },{ 5, 6, 7, 8 },{
9, 10, 11, 12 },{ 13, 14, 15, 16 } };
        int mat3[MATRIX_SIZE][MATRIX_SIZE] = { { 1, 2, 3, 4 },{ 5, 6, 7, 8 },{
9, 10, 11, 12 },{ 13, 14, 15, 16 } };
        int mat4[MATRIX_SIZE][MATRIX_SIZE] = { { 1, 2, 3, 4 },{ 5, 6, 7, 8 },{
9, 10, 11, 12 },{ 13, 14, 15, 16 } };
        int opt1[MATRIX_SIZE][MATRIX_SIZE] = { { 13, 9, 5, 1 },{ 14, 10, 6, 2
},{ 15, 11, 7, 3 },{ 16, 12, 8, 4 } };
        int opt2[MATRIX_SIZE][MATRIX_SIZE] = { { 4, 8, 12, 16 },{ 3, 7, 11, 15
},{ 2, 6, 10, 14 },{ 1, 5, 9, 13 } };
        int opt3[MATRIX_SIZE][MATRIX_SIZE] = { { 13, 14, 15, 16 },{ 9, 10, 11,
12 },{ 5, 6, 7, 8 },{ 1, 2, 3, 4 } };
        int opt4[MATRIX_SIZE][MATRIX_SIZE] = { { 4, 3, 2, 1 },{ 8, 7, 6, 5 },{
12, 11, 10, 9 },{ 16, 15, 14, 13 } };

        //variables for ex4
        char test1_a[M][M] = { "abcdefg", "gfeg", "abcd" };
        char test1_b[M][M] = { "cdefgba", "geff", "abcde" };
        char test2_a[M][M] = { "aba", "aabbaab", "zxw" };
        char test2_b[M][M] = { "baa", "abababa", "zxx" };
        char test3_a[M][M] = { "abcdefg", "zxyz" };
        char test3_b[M][M] = { "gfedcba", "xyzz" };
        char test4_a[M][M] = { "abcdefg", "a" };
        char test4_b[M][M] = { "gfedcba", "a" };
```

Call BY Adress

Passing an array as argument: its address is passed by value (a local parameter stores the address). But,
the array’s elements are accessible by
using the operator [].

Call by value

Passing arguments by value:
the arguments are copied to the local parameters of the function.

strings

the lenth name need to be :9 and the plus one for the '\0'

syntax:

char name[13] = “Im a donkey”;

char name[] = “Im a donkey”;

char name[] = {‘I’,’m’,’ ‘,’a’,’ ‘,’d’,’o’,’n’,’k’,’e’,’y’,’\0’};

char name[13];

scanf(“%s”, name);   //Im a donkey

gets(name);   //Im a donkey

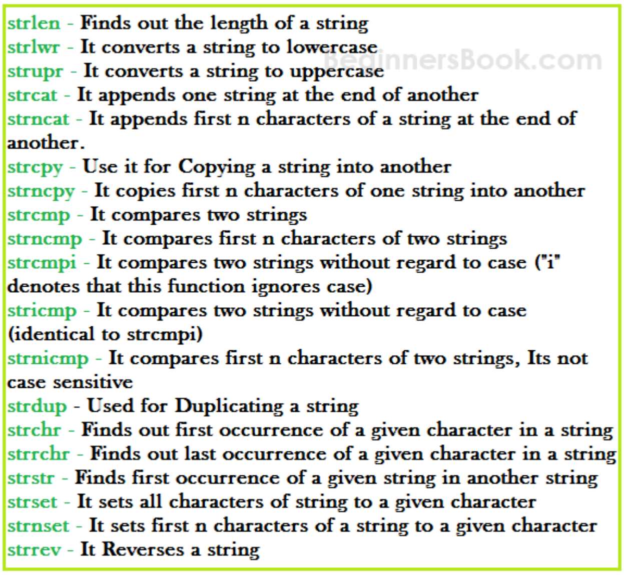
string library:

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

string functions:

String functions	Description
strcat ( )	Concatenates str2 at the end of str1
strncat ( )	Appends a portion of string to another
strcpy ( )	Copies str2 into str1
strncpy ( )	Copies given number of characters of one string to another

strlen ( )	Gives the length of str1
strcmp ( )	Returns 0 if str1 is same as str2. Returns <0 if strl < str2. Returns >0 if str1 > str2
strcmpi ( )	Same as strcmp() function. But, this function negotiates case. “A” and “a” are treated as same.
strchr ( )	Returns pointer to first occurrence of char in str1
strrchr ( )	last occurrence of given character in a string is found
strstr ( )	Returns pointer to first occurrence of str2 in str1
strrstr ( )	Returns pointer to last occurrence of str2 in str1
strdup ( )	Duplicates the string
strlwr ( )	Converts string to lowercase
strupr ( )	Converts string to uppercase
strrev ( )	Reverses the given string
strset ( )	Sets all character in a string to given character
strnset ( )	It sets the portion of characters in a string to given character
strtok ( )	Tokenizing given string using delimiter



strlen - Finds out the length of a string
strlwr - It converts a string to lowercase
strupr - It converts a string to uppercase
strcat - It appends one string at the end of another
strncat - It appends first n characters of a string at the end of another.
strcpy - Use it for Copying a string into another
strncpy - It copies first n characters of one string into another
strcmp - It compares two strings
strncmp - It compares first n characters of two strings
strcmpi - It compares two strings without regard to case ("i" denotes that this function ignores case)
stricmp - It compares two strings without regard to case (identical to strcmpi)
strnicmp - It compares first n characters of two strings, Its not case sensitive
strdup - Used for Duplicating a string
strchr - Finds out first occurrence of a given character in a string
strrchr - Finds out last occurrence of a given character in a string
strstr - Finds first occurrence of a given string in another string
strset - It sets all characters of string to a given character
strnset - It sets first n characters of a string to a given character
strrev - It Reverses a string

Pointers

Definition :

A pointer is a variable that contains the address of a variable.

Pointer size is 4 bytes.

Operators :

Indirection Operator * has 2 different meanings:

- Upon declaration – "I am a pointer".
- After declaration – access the variable whose address is held by the pointer.

Address Operator & provides the address of a variable

Errors:

- Cannot convert from int to int *
- Cannot convert from int * to float *
- points to garbage - trying to assign a value to memory not allocated

Calling :

- Passing argument **by value**: the argument is copied to the local parameter of the function.
- Passing argument **by address**: only the address of the argument is passed (a local parameter stores the address). So the operator * enables access to the value of the argument.

	By value	By address (of array)

Declaration	func(type name)	func(type name[])
		func(type *name)

Scope	In function	In function
Lifetime	In function	Address: lifetime only in function Array’s elements: original lifetime
Argument changeable	No	Address: no Array’s elements: yes

Example :

```
void main()
{
    int x=3, y=5;
    swap_by_value(x, y);
    swap_by_address(&x, &y);
}
```

```
void swap_by_value(int num1, int num2)
{
    int temp = num1;
    num1 = num2;
    num2 = temp;
}
```

```
void swap_by_address(int *num1, int *num2)
{
    int temp = *num1;
    *num1 = *num2;
    *num2 = temp;
}
```

Pointers and arrays ;

int array[5];

int *parr;

parr = array; // same as parr = &array[0];

Pointers and another operators :

Example :

```
int *p1 = NULL, *p2 = NULL;
if (p1 == p2) …
if (p1 < p2) …
if (p1 != NULL) …
```

the following experssions have the same mining :

```
int a[100];
int *p = a;
```

a,&a[0],p,&p[0]	address of the 1st element
***a,a[0],*p,p[0]**	value of the 1st element
a+1,&a[1],p+1,&p[1]	address of the 2nd element
***(a+1),a[1],*(p+1),p[1]**	value of the 2nd element
a+i,&a[i],p+i,&p[i]	address of the i-th element
***(a+i),a[i],*(p+i),p[i]**	value of the i-th element

Array of pointers :

Array of pointers is an array whose elements are pointer types.

```
int *arr[5];        // array of 5 int pointers
```

dynamic memory allocation:

- **The problem:**
  Array definition: its size must be known at compilation time.
  The array, when used, may be either too small – not enough room for all the elements, or too big – so it's a waste of memory.
- **The solution:**
  Use Dynamic Memory Allocation (DMA):

create the array at run-time, after determining the required number of elements.

- Dynamic memory allocation enables the programmer to:
  - Request exactly the required amount of memory.
  - Release the allocated memory when it is no longer needed.

Allocation Functions:

malloc(number of requested bytes);

calloc(number of elements, size of each element);

free(first_address);

realloc(start pointer ,size of each element);

we need to convert the answer to pointer type,example:

pointer = **(int *)** malloc(size*sizeof(int));

Pointer To Pointer

```
char **ppChar = NULL,  buffer[30];
int num;
printf("how many names?");
scanf("%d", &num);
ppChar = (char **) calloc(num, sizeof(char *));
```

Const and pointers

1. Pointer to read only data

const char *cp="Constant string";

2. Creating a read only pointer

char * const cp="A string";

3. Read only pointer to read only data

const char * const cp= "A constant string";

pointer to functions:

- Definition of pointer to function:

```
int (*pfunc)(int, int); //pfunc is a pointer to function
pfunc = sum;
int n = pfunc(3,5);
```

return_type (*pointer_func_name)(args)

```
int main()  {
/* function is a pointer to a function with signature Relation f(int, int) */
   Relation  (*function)(int, int) ;

   if (getchar() == '1') {
         function = bigger;
   } else  {
        function = bigger2;
   }
  int a = -5 , b = 3 ;
  Relation   relation =  function(a,b);
   switch (relation) {
      case Left: printf ("%d\n",a);
            break;
      case Eq: printf ("%d\n",a);
            break;
      case Right: printf ("%d\n",b);
            break:
```

```
typedef Relation (*CmpFunction)(void*, void*);
void sort(void **array, int n, CmpFunction compare){
   int i, j;
  void*  tmp;
   assert(array !=NULL && compare != NULL);
   for(i=0; i<n; i++) {
      for(j=i+1; j<n; j++) {
         if(compare(arr[i], arr[j])==Left) {
            tmp = array[i];
            array[i] = array[j];
            array[j] = tmp;
```

structures :

definition:

We can define multiple data types, but for some students we have to define multiple parallel arrays - for each type

How to import new structure ?

Way 1:

```
struct Student{

   char name[20];

   int id;

}
```

Way 2:

```
typedef struct{

  char name[20];

  int id;

} Student;
```

Initialize:

From the main :

```
struct Student std = {“arie”, 222};
```

reference:

- Reference a field via the variable std1 using operator . :

  std1.name
  std1.id

- Reference a field via the pointer pStudent:

  (*pStudent).name
  (*pStudent).id

- Another, more aesthetic way for referencing a field via a pointer is using the (shorthand for (*pointer).) operator -> (arrow):

  pStudent->name
  pStudent->id

Nested structer :

A member of a structure may be itself a structure.

For example, a student could be a member of a class.

Manage project :

## include:

- **#include “data.h”**
  - Import file named : "data.h"
  - We can write the full path
- **#include <stdio.h>**
  - Include library named : stdio

If def:

```
     #define one 0
+--- #ifdef one
|    printf("one is defined ");     // Everything in here is included.
| +- #ifndef one
| |  printf("one is not defined "); // Everything in here is excluded.
| |  :
| +- #endif
|    :                              // Everything in here is included
again.
+--- #endif
```

Recurtions :

Definition:

Recursion is the process of repeating items in a self-similar way. In programming languages, if a program allows you to call a function inside the same function, then it is called a recursive call of the function.

Example :

```
void func(int i)
{
   if(i>0)
  {
    func(i-1);
       printf(“%d”, i);
 }
}
void main()
{
    func(3);
}
```

Algoritem:

1. Base – the base problem that we cant dismantle
2. Steps – all the steps that we need to do until we comes to the base.

	Recursive function	Iterative function
Length	Shorter	Longer
Writing style	Clearer	Less Clear
Execution Speed	Slower	Faster
Memory resources	More	Less
Writing time	Shorter	Longer (significantly)

Arguments to main :

- The parameters have conventional names and their order is fixed:

  main(int argc, char *argv[ ])

- argc -
  - An integer specifying how many arguments are passed to the program via the command line.
  - The program name is considered as an argument, hence argc is at least 1.
- argv -
  - array of pointers to the arguments list.
  - argv[0] is the program name.
  - argv[argc-1] is the last argument.

main gets a string and 2 numbers and prints the string before the sum of the numbers (the execution file is called “print”).