**Syllabus( Programming through C Language) : Programming through C Language (Example programs may be used to explain each of the topics)introduction: History, Basic Structure, Algorithms, Structured programming constructs. C Programming elements: Character sets, Keywords, Constants, Variables, Data Types, Operators- Arithmetic, Relational, Logical and Assignment; Increment and Decrement and Conditional, Operator Precedence and Associations; Expressions. type casting. Comments, Functions, Storage Classes, Bit manipulation, Input and output. C Preproc;essor:File inclusion, Macro substitution. Statements: Assignment. Control statements- if, if~else, switch, break. continue, goto, Loops-while, do_while, for.Functions: argument passing, return statement, return values and their types, recursion. Arrays: String handling with arrays, String handling functions. Pointers: Definition and initialization, Pointer arithmetic. Pointers and arrays, String functions and manipulation, Dynamic storage allocation. User defined Data types: Enumerated data types, Structures. Structure arrays, Pointers to Functions and Structures, Unions. File access Opening Closing i/o operation.**

## What does static variable mean in C?

Static variable is available to a C application, throughout the life time. At the time of starting the program execution, static variables allocations takes place first. In a scenario where one variable is to be used by all the functions (which is accessed by main () function), then the variable need to be declared as static in a C program. The value of the variable is persisted between successive calls to functions. One more significant feature of static variable is that, the address of the variable can be passed to modules and functions which are not in the same C file.Static variables are the variables which retain their values between the function calls. They are initialized only once their scope is within the function in which they are defined.

## What is a pointer?

Ans: A **pointer** is a variable which contains the address in memory of another variable. We can have a **pointer** to any variable type. The unary or monadic operator & gives the ``address of a variable''. The indirection or dereference operator * gives the ``contents of an object pointed to by a **pointer**''.The pointed-to object may be part of a larger object, such as a field of a structure or an element in an array.

```
#include <stdio.h>
int main()
{
  int *ptr, q;
  q = 50;
  /* address of q is assigned to ptr */
  ptr = &q;
  /* display q's value using ptr variable */
  printf("%d", *ptr);

  return 0
}
```

**3. What are the uses of a pointer?**
Ans: Pointer is used in the following cases
i) It is used to access array elements
ii) It is used for dynamic memory allocation.
iii) It is used in Call by reference
iv) It is used in data structures like trees, graph, linked list etc.

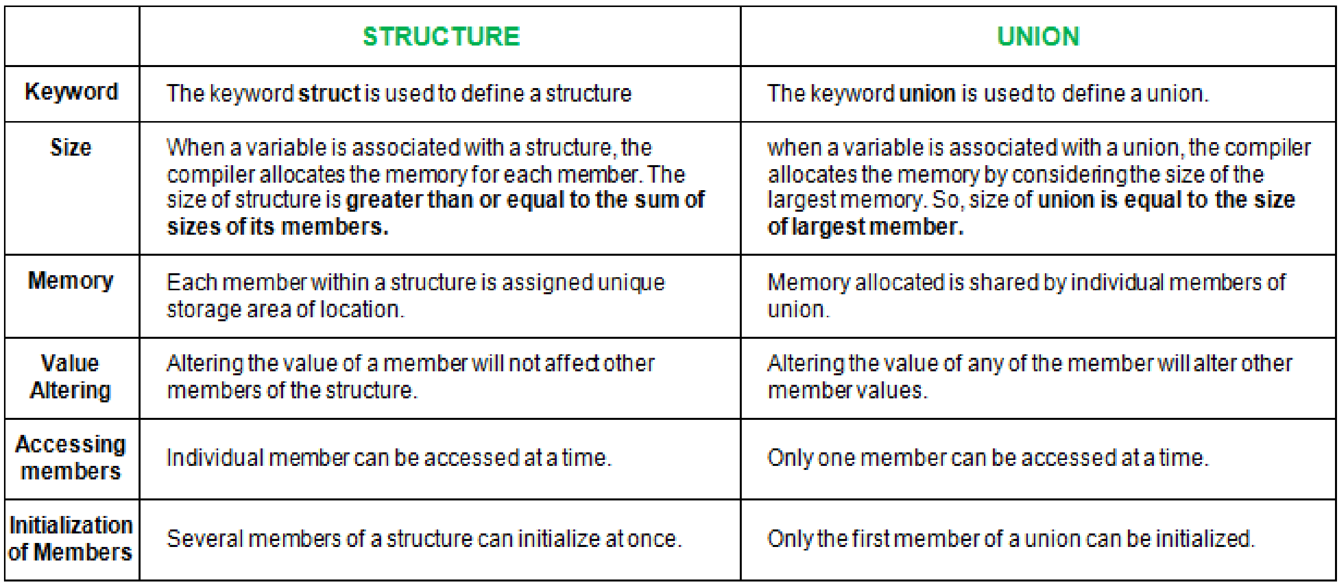
**4. What is a structure?**
Ans: Structure constitutes a super data type which represents several different data types in a single unit. A structure can be initialized if it is static or global.
**5. What is a union?**
Ans: Union is a collection of heterogeneous data type but it uses efficient memory utilization technique by allocating enough memory to hold the largest member. Here a single area of memory contains values of different types at different time. A union can never be initialized.

**6.What are the differences between structures and union?**

**Similarities between Structure and Union**

1. Both are user-defined data types used to store data of different types as a single unit.
2. Their members can be objects of any type, including other structures and unions or arrays. A member can also consist of a bit field.
3. Both structures and unions support only assignment = and sizeof operators. The two structures or unions in the assignment must have the same members and member types.
4. A structure or a union can be passed by value to functions and returned by value by functions. The argument must have the same type as the function parameter. A structure or union is passed by value just like a scalar variable as a corresponding parameter.
5. **'.'** operator is used for accessing members.

**Differences**



| | STRUCTURE | UNION |
|---|---|---|
| Keyword | The keyword **struct** is used to define a structure | The keyword **union** is used to define a union. |
| Size | When a variable is associated with a structure, the compiler allocates the memory for each member. The size of structure is **greater than or equal to the sum of sizes of its members.** | when a variable is associated with a union, the compiler allocates the memory by considering the size of the largest memory. So, size of **union is equal to the size of largest member.** |
| Memory | Each member within a structure is assigned unique storage area of location. | Memory allocated is shared by individual members of union. |
| Value Altering | Altering the value of a member will not affect other members of the structure. | Altering the value of any of the member will alter other member values. |
| Accessing members | Individual member can be accessed at a time. | Only one member can be accessed at a time. |
| Initialization of Members | Several members of a structure can initialize at once. | Only the first member of a union can be initialized. |

**7. What are the differences between structures and arrays?**

Ans: Structure is a collection of heterogeneous data type but array is a collection of homogeneous data types.

**Array**

1-It is a collection of data items of same data type.

2-It has declaration only

3-.There is no keyword.

4- array name represent the address of the starting element.

**Structure**

1-It is a collection of data items of different data type.

2- It has declaration and definition

3- keyword struct is used

4-Structure name is known as tag it is the short hand notation of the declaration.

**8.What are the differences between malloc () and calloc ()?**

Differences between malloc and calloc

| **malloc** | **calloc** |
|---|---|
| The name malloc stands for *memory allocation*. | The name calloc stands for *contiguous allocation*. |
| void *malloc(size_t n) returns a pointer to n bytes of uninitialized storage, or NULL if the request cannot be satisfied. If the space assigned by malloc() is overrun, the results are undefined. | void *calloc(size_t n, size_t size) returns a pointer to enough free space for an array of n objects of the specified size, or NULL if the request cannot be satisfied. The storage is initialized to zero. |
| malloc() takes one argument that is, *number of bytes*. | calloc() take two arguments those are: *number of blocks* and *size of each block*. |
| syntax of malloc():<br>void *malloc(size_t n);<br>Allocates n bytes of memory. If the allocation succeeds, a void pointer to the allocated memory is returned. Otherwise NULL is returned. | syntax of calloc():<br>void *calloc(size_t n, size_t size);<br>Allocates a contiguous block of memory large enough to hold n elements of size bytes each. The allocated region is initialized to zero. |
| malloc is faster than calloc. | calloc takes little longer than malloc because of the extra step of initializing the allocated memory by zero. However, in practice the difference in speed is very tiny and not recognizable. |

**9. What are macros? What are its advantages and disadvantages?**

Macro is a Pre-proceser. The main advantage of using the macro is the speed of the execution of the program (if macro is not used in the program many times or if the program is small). The main disadvantage of the macro is it increase the size of the program because the pre-processor will replace all the macro name in the program by it actual definition before the compilation of the program,

**Describe the advantages of using macro.**

A macro is a name given to a block of the code which can be substituted where the code snippet is to be used for more than once.

- The speed of the execution of the program is the major advantage of using a macro.

- It saves a lot of time that is spent by the compiler for invoking / calling the functions.

- It reduces the length of the program.

The **disadvantage** of the macro is the size of the program. The reason is, the pre-processor will replace all the macros in the program by its real definition prior to the compilation process of the program.

**What is is call by value and call by reference ? What ae the difference between pass by reference and pass by value?**

**Call by Value**

If data is passed by value, the data is copied from the variable used in for example main() to a variable used by the function. So if the data passed (that is stored in the function variable) is modified inside the function, the value is only changed in the variable used inside the function. Let’s take a look at a call by value example:

```
#include <stdio.h>

void call_by_value(int x) {
	printf("Inside call_by_value x = %d before adding 10.\n", x);
	x += 10;
	printf("Inside call_by_value x = %d after adding 10.\n", x);
}

int main() {
	int a=10;

	printf("a = %d before function call_by_value.\n", a);
	call_by_value(a);
	printf("a = %d after function call_by_value.\n", a);
	return 0;
}
```

The output of this call by value code example will look like this:

```
a = 10 before function call_by_value.
Inside call_by_value x = 10 before adding 10.
Inside call_by_value x = 20 after adding 10.
a = 10 after function call_by_value.
```

Ok, let’s take a look at what is happening in this call-by-value source code example. In the main() we create a integer that has the value of 10. We print some information at every stage, beginning by printing our variable a. Then function call_by_value is called and we input the variable a. This variable (a) is then copied to the function variable x. In the function we add 10 to x (and also call some print statements). Then when the next statement is called in main() the value of variable a is printed. We can see that the value of variable a isn’t changed by the call of the function call_by_value().

## **Call by Reference**

If data is passed by reference, a pointer to the data is copied instead of the actual variable as is done in a call by value. Because a pointer is copied, if the value at that pointers address is changed in the function, the value is also changed in main(). Let’s take a look at a code example:

```
#include <stdio.h>

void call_by_reference(int *y) {
        printf("Inside call_by_reference y = %d before adding 10.\n", *y);
        (*y) += 10;
        printf("Inside call_by_reference y = %d after adding 10.\n", *y);
}

int main() {
        int b=10;

        printf("b = %d before function call_by_reference.\n", b);
        call_by_reference(&b);
        printf("b = %d after function call_by_reference.\n", b);

        return 0;
}
```

The output of this call by reference source code example will look like this:

```
b = 10 before function call_by_reference.
Inside call_by_reference y = 10 before adding 10.
Inside call_by_reference y = 20 after adding 10.
b = 20 after function call_by_reference.
```

Let's explain what is happening in this source code example. We start with an integer b that has the value 10. The function call_by_reference() is called and the address of the variable b is passed to this function. Inside the function there is some before and after print statement done and there is 10 added to the value at the memory pointed by y. Therefore at the end of the function the value is 20. Then in main() we again print the variable b and as you can see the value is changed (as expected) to 20.

**Difference between *call by value* and *call by reference***

| call by value | call by reference |
|---|---|
| In *call by value*, a copy of actual arguments is passed to formal arguments of the called function and any change made to the formal arguments in the called function have no effect on the values of actual arguments in the calling function. | In *call by reference*, the location (address) of actual arguments is passed to formal arguments of the called function. This means by accessing the addresses of actual arguments we can alter them within from the called function. |
| In call by value, actual arguments will remain safe, they cannot be modified accidentally. | In *call by reference*, alteration to actual arguments is possible within from called function; therefore the code must handle arguments carefully else you get unexpected results. |

**What is static identifier?**
**Ans:** A file-scope variable that is declared static is visible only to functions within that file. A function-scope or block-scope variable that is declared as static is visible only within that scope. Furthermore, static variables only have a single instance. In the case of function- or block-scope variables, this means that the variable is not ―automatic‖ and thus retains its value across function invocations.

**Difference between arrays and linked list?**
**Ans:** An array is a repeated pattern of variables in contiguous storage. A linked list is a set of structures scattered through memory, held together by pointers in each element that point to the next element. With an array, we can (on most architectures) move from one element to the next by adding a fixed constant to the integer value of the pointer. With a linked list, there is a ―next‖ pointer in each structure which says what element comes next.

**What are enumerations?**
**Ans:** They are a list of named integer-valued constants. Example:enum color { black , orange=4, yellow, green, blue, violet };This declaration defines the symbols "black", "orange", "yellow" etc. to have the values "1","4", "5", … etc. The difference between an enumeration and a macro
is that the enum actually declares a type, and therefore can be type checked.
**Describe about storage allocation and scope of global, extern, static, local and register variables?**
**Ans:**
Globals have application-scope. They‘re available in any compilation unit that includes an appropriate declaration (usually brought from a header file). They‘re stored wherever the linker

puts them, usually a place called the ―BSS segment.
Extern? This is essentially ―global.
**Static:** Stored the same place as global, typically, but only available to the compilation unit that contains them. If they are block-scope global, only available within that block and its subblocks.
**Local:** Stored on the stack, typically. Only available in that block and its subblocks.
(Although pointers to locals can be passed to functions invoked from within a scope where that local is valid.)
**Register:** See tirade above on ―local‖ vs. ―register.‖ The only difference is that the C compiler will not let you take the address of something you‗ve declared as ―register.
**What are register variables? What are the advantages of using register variables?**
**Ans:** If a variable is declared with a register storage class,it is known as register variable.The register variable is stored in the cpu register instead of main memory.Frequently used variables are declared as register variable as it‗s access time is faster.
**What is the use of typedef?**
**Ans:** The typedef help in easier modification when the programs are ported to another machine.
A descriptive new name given to the existing data type may be easier to understand the code.
**Difference between strdup and strcpy?**
**Ans:** Both copy a string. strcpy wants a buffer to copy into. strdup allocates a buffer using malloc(). Unlike strcpy(), strdup() is not specified by ANSI .
**What is storage class? What are the different storage classes in C?**
**Storage Classes in C**
Storage Classes are used to describe about the features of a variable/function. These features basically include the scope, visibility and life-time which help us to trace the existence of a particular variable during the runtime of a program.

C language uses 4 storage classes, namely:

**auto**: This is the default storage class for all the variables declared inside a function or a block. Hence, the keyword auto is rarely used while writing programs in C language. Auto variables can be only accessed within the block/function they have been declared and not outside them (which defines their scope). Of course, these can be accessed within nested blocks within the parent block/function in which the auto variable was declared. However, they can be accessed outside their scope as well using the concept of pointers given here by pointing to the very exact memory location where the variables resides. They are assigned a garbage value by default whenever they are declared.

**extern**: Extern storage class simply tells us that the variable is defined elsewhere and not within the same block where it is used. Basically, the value is assigned to it in a different block and this can be overwritten/changed in a different block as well. So an extern variable is nothing but a global variable initialized with a legal value where it is declared in order to be used elsewhere. It can be accessed within any function/block. Also, a normal global variable can be made extern as well by placing the 'extern' keyword before its declaration/definition in any function/block. This basically signifies that we are not initializing a new variable but instead we are using/accessing the global variable only. The main purpose of using extern variables is that they can be accessed between two different files which are part of a large program. For more information on how extern variables work, have a look at this link.

**static**: This storage class is used to declare static variables which are popularly used while writing programs in C language. Static variables have a property of preserving their value even after they are out of their scope! Hence, static variables preserve the value of their last use in their scope. So we can say that they are initialized only once and exist till the termination of the program. Thus, no new memory is allocated because they are not re-declared. Their scope is local to the function to which they were defined. Global static variables can be accessed anywhere in the program. By default, they are assigned the value 0 by the compiler.

**register**: This storage class declares register variables which have the same functionality as that of the auto variables. The only difference is that the compiler tries to store these variables in the register of the microprocessor if a free register is available. This makes the use of register variables to be much faster than that of the variables stored in the memory during the runtime of the program. If a free register is not available, these are then stored in the memory only. Usually few variables which are to be accessed very frequently in a program are declared with the register keyword which improves the running time of the program. An important and interesting point to be noted here is that we cannot obtain the address of a register variable using pointers.

**What is the difference between Strings and Arrays?**
**Ans:** String is a sequence of characters ending with NULL .it can be treated as a one dimensional
Array of characters terminated by a NULL character.
**What is a normalized pointer, how do we normalize a pointer?**
**Ans:** It is a 32bit pointer, which has as much of its value in the segment register as possible.
Since a segment can start every 16bytes so the offset will have a value from 0 to F. for normalization convert the address into 20bit address then use the 16bit for segment address and 4bit for the offset address. Given a pointer 500D: 9407,we convert it to a 20bitabsolute address
549D7,Which then normalized to 549D:0007.
**In C, why is the void pointer useful? When would you use it?**
**Ans:** The void pointer is useful because it is a generic pointer that any pointer can be cast into
And back again without loss of information.
**What is a NULL Pointer?**
NULL pointer is pointer which is not pointing to anything in the memory. NULL is defined as (void*)0. Uninitialised pointer is pointing to some memory location but the pointer values are not assigned.Malloc returns the memory address, but the contents of the pointer are junk values.

**How pointer variables are initialized?**
Pointer variables are initialized by one of the following ways.
I. Static memory allocation
II. Dynamic memory allocation
**static memory allocation:** Compiler allocates memory space for a declared variable. By using the address of operator, the reserved address is obtained and this address is assigned to a pointer variable. This way of assigning pointer value to a pointer variable at compilation time is known as static memory allocation.
**Dynamic memory allocation:** A dynamic memory allocation uses functions such as malloc() or calloc() to get memory dynamically. If these functions are used to get memory dynamically and the values returned by these function are assigned to pointer variables, such a way of allocating memory at run time is known as dynamic memory allocation.

```
#include<stdio.h>
#define MAX 5
int main()
```

```
{
   int p[] = {1,2,3,4,5};//Static initialization
   int *q = NULL;
   int i;

   for(i=0;i<MAX ;i++)
      printf("%d",p[i]);

   q = (int*)malloc(sizeof(int)*MAX );//Malloc initialization will be done with garbage values and calloc will
initialize with 0's
   for(i=0;i<MAX ;i++)
   scanf("%d",&q[i]);

      for(i=0;i<MAX ;i++)
      printf("%d ",q[i]);

   getch();
   return 0;
}
```

**What is the purpose of realloc?**
**Ans:** It increases or decreases the size of dynamically allocated array. The function realloc (ptr,n) uses two arguments. The first argument ptr is a pointer to a block of memory for which the size is to be altered. The second argument specifies the new size. The size may be increased or decreased. If sufficient space is not available to the old region the function may create a new region.
**What is pointer to a pointer?**
**Ans:** If a pointer variable points another pointer value. Such a situation is known as a pointer to a pointer.
Example:
int *p1,**p2,v=10;
P1=&v; p2=&p1;
Here p2 is a pointer to a pointer.

**What is an array of pointers?**
In general, an **array of pointers** can be used to point to an **array** of data items with each element of the **pointer array** pointing to an element of the data **array**. Data items can be accessed either directly in the data **array**, or indirectly by dereferencing the elements of the **pointer array**.
**Is it possible to have negative index in an array?**
**Ans:** Yes it is possible to index with negative value provided there are data stored in this location. Even if it is illegal to refer to the elements that are out of array bounds, the compiler will not produce error because C has no check on the bounds of an array.
**Why is it necessary to give the size of an array in an array declaration?**
**Ans:** When an array is declared, the compiler allocates a base address and reserves enough space In memory for all the elements of the array. The size is required to allocate the required space and hence size must be mentioned.
**What is a function?**
**Ans:** A large program is subdivided into a number of smaller programs or subprograms. Each

Subprogram specifies one or more actions to be performed for the larger program. Such sub programs are called functions.

**Difference between formal argument and actual argument?**

**Ans:** Formal arguments are the arguments available in the function definition. They are preceded
By their own data type. Actual arguments are available in the function call. These arguments are
Given as constants or variables or expressions to pass the values to the function.

**What is the difference between an enumeration and a set of pre-processor # defines?**

**Ans:** There is hardly any difference between the two, except that #defines has a global effect
(throughout the file) whereas an enumeration can have an effect local to the block if desired.
Some advantages of enumeration are that the numeric values are automatically assigned whereas
in #define we have to explicitly define them. A disadvantage is that we have no control over the
size of enumeration variables.

**what is the similarity between a Structure, Union and enumeration?**

**Ans:** All of them let the programmer to define new data type.

**Write a program which uses Command Line Arguments?**

**Ans:**

```
#include
void main(int argc,char *argv[])
{
int i;
clrscr();
for(i=0;i
printf(―\n%d‖,argv[i]);
}
```

**Difference between array and pointer?**

**Ans:**

**Array**

1- Array allocates space automatically
2- It cannot be resized
3- It cannot be reassigned
4- sizeof (arrayname) gives the number of bytes occupied by the array.

**Pointer**

1-Explicitly assigned to point to an allocated space.
2-It can be sized using realloc()
3-pointer can be reassigned.
4-sizeof (p) returns the number of bytes used to store the pointer variable p.

**what are C tokens?**

**Ans:** There are six classes of tokens: identifier, keywords, constants, string literals, operators and other separators.

**What are C identifiers?**

**Ans:** These are names given to various programming element such as variables, function,
arrays.It is a combination of letter, digit and underscore.It should begin with letter. Backspace is
not allowed.

**Difference between syntax vs logical error?**

**Ans:**

**Syntax Error**

1-These involves validation of syntax of language.
2-compiler prints diagnostic message.

**Logical Error**

1-logical error are caused by an incorrect algorithm or by a statement mistyped in such a way
that it doesn‘t violet syntax of language.
2-difficult to find.

**What is preincrement and post increment?**

- ++a ( preincrement ) *first increments* the value of a and then returns an lvalue referring to a, so if the value of a is used then it will be the incremented value.
- a++ ( post increment) *first returns an rvalue* whose value is a, that is, the old value, and then increments a *at an unspecified time* before the next full-expression (i.e., "before the semicolon").
- Postfix increment has higher precedence than prefix increment.

**What is a preprocessor, what are the advantages of preprocessor?**

**Ans:** A preprocessor processes the source code program before it passes through the compiler.A preprocessor is a language that takes as input a text file written using some programming language syntax and output another text file following the syntax of another programming language.
Advantages of preprocessor are that it makes-
1) the program easier to develop.
2) easier to read.
3) easier to modify.
4) C code more transportable between different machine architecture.
The facilities provided by a preprocessor are given below-
1) File inclusion
2) Substitution facility
3) Conditional compilation.

**What are the two forms of #include directive?**

**Ans:**
1.#include‖filename‖
2.#include
the first form is used to search the directory that contains the source file.If the search fails in the home directory it searches the implementation defined locations.In the second form ,the preprocessor searches the file only in the implementation defined locations.

**How would you use the functions randomize() and random()?**

**Ans:**
Randomize() initiates random number generation with a random value.
Random() generates random number between 0 and n-1;

**What do the functions atoi(), itoa() and gcvt() do?**

**Ans:**
atoi() is a macro that converts integer to character.
itoa() It converts an integer to string.
gcvt() It converts a floating point number to string.

**How would you use the functions fseek(), freed(), fwrite() and ftell()?**

**Ans:**
fseek(f,1,i) Move the pointer for file f a distance 1 byte from location i.
fread(s,i1,i2,f) Enter i2 dataitems,each of size i1 bytes,from file f to string s.
fwrite(s,i1,i2,f) send i2 data items,each of size i1 bytes from string s to file f.
ftell(f) Return the current pointer position within file f.
The data type returned for functions fread,fseek and fwrite is int and ftell is long int.

**What is a file?**

**Ans:** A file is a region of storage in hard disks or in auxiliary storage devices.It contains bytes of information .It is not a data type. Files are of two types
1-high level files (stream oriented files) :These files are accessed using library functions

2-low level files(system oriented files) :These files are accessed using system calls.

**what is a stream?**

**Ans:** A stream is a source of data or destination of data that may be associated with a disk or other
I/O device. The source stream provides data to a program and it is known as input stream. The destination stream eceives the output from the program and is known as output stream.

**What is FILE?**

**Ans:** FILE is a predefined data type. It is defined in stdio.h file.

**Difference between a array name and a pointer variable?**

**Ans:** A pointer variable is a variable where as an array name is a fixed address and is not a variable. A pointer variable must be initialized but an array name cannot be initialized. An array name being a constant value , ++ and — operators cannot be applied to it.

**Difference between an array of pointers and a pointer to an array?**

**Ans:**

**Array of pointers**

1- Declaration is: data_type *array_name[size];
2-Size represents the row size.
3- The space for columns may be dynamically

**Pointers to an array**

1-Declaration is data_type ( *array_name)[size];
2-Size represents the column size.

**What are the advantages of using array of pointers to string instead of an array of strings?**

**Ans:**

i) Efficient use of memory.
ii) Easier to exchange the strings by moving their pointers while sorting.

**Are the expressions *ptr ++ and ++ *ptr same?**

**Ans:** No,*ptr ++ increments pointer and not the value pointed by it. Whereas ++ *ptr increments the value being pointed to by ptr.

**Explain bitwise logical operator with example in c.**

**Ans:**

The following table lists the Bitwise operators supported by C. Assume variable 'A' holds 60 and variable 'B' holds 13, then –

| Operator | Description | Example |
|---|---|---|
| & | Binary AND Operator copies a bit to the result if it exists in both operands. | (A & B) = 12 i.e., 0000 1100 |
| \| | Binary OR Operator copies a bit if it exists in either operand. | (A \| B) = 61 i.e., 0011 1101 |
| ^ | Binary XOR Operator copies the bit if it is set in one operand but not both. | (A ^ B) = 49 i.e., 0011 0001 |

| | | |
|---|---|---|
| ~ | Binary Ones Complement Operator is unary and has the effect of 'flipping' bits. | (~A ) = -61 i.e., 1100 0011 in 2's complement form. |
| << | Binary Left Shift Operator. The left operands value is moved left by the number of bits specified by the right operand. | A << 2 = 240 i.e., 1111 0000 |
| >> | Binary Right Shift Operator. The left operands value is moved right by the number of bits specified by the right operand. | A >> 2 = 15 i.e., 0000 1111 |

## Example

Try the following example to understand all the bitwise operators available in C –

```
#include <stdio.h>


main() {


   unsigned int a = 60;  /* 60 = 0011 1100 */

   unsigned int b = 13;  /* 13 = 0000 1101 */

   int c = 0;


   c = a & b;       /* 12 = 0000 1100 */

   printf("Line 1 - Value of c is %d\n", c );


   c = a | b;       /* 61 = 0011 1101 */

   printf("Line 2 - Value of c is %d\n", c );


   c = a ^ b;       /* 49 = 0011 0001 */

   printf("Line 3 - Value of c is %d\n", c );
   c = ~a;          /*-61 = 1100 0011 */

   printf("Line 4 - Value of c is %d\n", c );



   c = a << 2;     /* 240 = 1111 0000 */

   printf("Line 5 - Value of c is %d\n", c );



   c = a >> 2;     /* 15 = 0000 1111 */

   printf("Line 6 - Value of c is %d\n", c );

}
```

When you compile and execute the above program, it produces the following result −

```
Line 1 - Value of c is 12
Line 2 - Value of c is 61
Line 3 - Value of c is 49
Line 4 - Value of c is -61
Line 5 - Value of c is 240
Line 6 - Value of c is 15
```

**What is Recursion ? Guve example.**

**Ans:**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. The C programming language supports recursion, i.e., a function to call itself. But while using recursion, programmers need to be careful to define an exit condition from the function, otherwise it will go into an infinite loop.

Recursive functions are very useful to solve many mathematical problems, such as calculating the factorial of a number, generating Fibonacci series, etc.

```
#include <stdio.h>

int factorial(unsigned int i) {

   if(i <= 1) {
      return 1;
```

```
  }
  return i * factorial(i - 1);
}

int  main() {
  int i = 15;
  printf("Factorial of %d is %d\n", i, factorial(i));
  return 0;
}
```

**What are the difference between macro ad function ?**

## Difference between macro and function

| No | Macro | Function |
|---|---|---|
| 1 | Macro is **Preprocessed** | Function is **Compiled** |
| 2 | **No Type Checking** | **Type Checking** is Done |
| 3 | **Code** Length **Increases** | **Code** Length remains **Same** |
| 4 | Use of macro can lead to **side effect** | No **side Effect** |
| 5 | Speed of Execution is **Faster** | Speed of Execution is **Slower** |
| 6 | Before Compilation macro name is replaced by macro value | During function call , Transfer of Control takes place |
| 7 | Useful where small code appears many time | Useful where large code appears many time |
| 8 | Generally Macros do not extend beyond one line | Function can be of any number of lines |
| 9 | Macro does not Check **Compile Errors** | Function Checks **Compile Errors** |
| 10 | **example of Macro:** `#include<stdio.h>` `#define NUMBER 10` `int main(){` | **example of Function:** `#include<stdio.h>` `int number()` `{     return 10;}` `int main(){` |

| | | |
|---|---|---|
| | `printf("%d", NUMBER);`<br><br>`return 0;}` | `printf("%d", number());`<br><br>`return 0;}` |