

## 1. What is C language?

The C programming language is a standardized programming language developed in the early 1970s by Ken Thompson and Dennis Ritchie for use on the UNIX operating system. It has since spread to many other operating systems, and is one of the most widely used programming languages. C is prized for its efficiency, and is the most popular programming language for writing system software, though it is also used for writing applications.

## 2. What does static variable mean?

There are 3 main uses for the static.

1. If you declare within a function: It retains the value between function calls
2. If it is declared for a function name: By default function is extern..so it will be visible from other files if the function declaration is as static..it is invisible for the outer files
3. Static for global variables: By default we can use the global variables from outside files If it is static global..that variable is limited to within the file.

```
#include <stdio.h>
int t = 10;
main(){
int x = 0;
void funct1();
funct1();
printf("After first call \n");
funct1();
printf("After second call \n");
funct1();
printf("After third call \n");
}
void funct1()
{
    static int y = 0;
    int z = 10;
    printf("value of y %d z %d",y,z);
    y=y+10;
}
```

value of y 0 z 10 After first call  
value of y 10 z 10 After second call  
value of y 20 z 10 After third call

## 3. What are the different storage classes in C?

C has three types of storage: automatic, static and allocated. Variable having block scope and without static specifier have automatic storage duration.

Variables with block scope, and with static specifier have static scope. Global variables (i.e, file scope) with or without the the static specifier also have static scope. Memory obtained from calls to malloc(), alloc() or realloc() belongs to allocated storage class.

#### **4. What is hashing?**

To hash means to grind up, and that's essentially what hashing is all about. The heart of a hashing algorithm is a hash function that takes your nice, neat data and grinds it into some random-looking integer.

The idea behind hashing is that some data either has no inherent ordering (such as images) or is expensive to compare (such as images). If the data has no inherent ordering, you can't perform comparison searches.

#### **5. Can static variables be declared in a header file?**

You can't declare a static variable without defining it as well (this is because the storage class modifiers static and extern are mutually exclusive). A static variable can be defined in a header file, but this would cause each source file that included the header file to have its own private copy of the variable, which is probably not what was intended.

#### **6. Can a variable be both constant and volatile?**

Yes. The const modifier means that this code cannot change the value of the variable, but that does not mean that the value cannot be changed by means outside this code.

The function itself did not change the value of the timer, so it was declared const. However, the value was changed by hardware on the computer, so it was declared volatile. If a variable is both const and volatile, the two modifiers can appear in either order.

#### **7. Can include files be nested?**

Yes. Include files can be nested any number of times. As long as you use precautionary measures, you can avoid including the same file twice. In the past, nesting header files was seen as bad programming practice, because it complicates the dependency tracking function of the MAKE program and thus slows down compilation. Many of today's popular compilers make up for this difficulty by implementing a concept called precompiled headers, in which all headers and associated dependencies are stored in a precompiled state.

## **8. What is a null pointer?**

There are times when it's necessary to have a pointer that doesn't point to anything. The macro `NULL`, defined in `<stddef.h>`, has a value that's guaranteed to be different from any valid pointer. `NULL` is a literal zero, possibly cast to `void*` or `char*`.

Some people, notably C++ programmers, prefer to use `0` rather than `NULL`.

The null pointer is used in three ways:

- 1) To stop indirection in a recursive data structure.
- 2) As an error value.
- 3) As a sentinel value.

## **9. What is the output of `printf("%d")` ?**

When we write `printf("%d",x);` this means compiler will print the value of `x`. But as here, there is nothing after `%d` so compiler will show in output window garbage value.

## **10. What is the difference between `calloc()` and `malloc()` ?**

`calloc(...)` allocates a block of memory for an array of elements of a certain size. By default the block is initialized to 0. The total number of memory allocated will be  $(\text{number\_of\_elements} * \text{size})$ .

`malloc(...)` takes in only a single argument which is the memory required in bytes. `malloc(...)` allocated bytes of memory and not blocks of memory like `calloc(...)`.

`malloc(...)` allocates memory blocks and returns a void pointer to the allocated space, or `NULL` if there is insufficient memory available.

`calloc(...)` allocates an array in memory with elements initialized to 0 and returns a pointer to the allocated space. `calloc(...)` calls `malloc(...)` in order to use the C++ `_set_new_mode` function to set the new handler mode.

## **11. What is the difference between `printf()` and `sprintf()` ?**

`sprintf()` writes data to the character array whereas `printf(...)` writes data to the standard output device.

## **12. How to reduce a final size of executable?**

Size of the final executable can be reduced using dynamic linking for libraries.

### 13. Can you tell me how to check whether a linked list is circular?

Create two pointers, and set both to the start of the list. Update each as follows:

```
while (pointer1) {
    pointer1 = pointer1->next;
    pointer2 = pointer2->next;
    if (pointer2) pointer2=pointer2->next;
    if (pointer1 == pointer2) {
        print ("circular");
    }
}
```

If a list is circular, at some point pointer2 will wrap around and be either at the item just before pointer1, or the item before that. Either way, its either 1 or 2 jumps until they meet.

### 14. Advantages of a macro over a function?

Macro gets to see the Compilation environment, so it can expand `__TIME__` `__FILE__` `#defines`. It is expanded by the preprocessor.

For example, you can't do this without macros  
`#define PRINT(EXPR) printf( #EXPR "=%d\n", EXPR)`  
`PRINT( 5+6*7 )` // expands into `printf("5+6*7=%d", 5+6*7 );`  
You can define your mini language with macros:  
`#define strequal(A,B) (!strcmp(A,B))`

### 15. What is the difference between strings and character arrays?

A major difference is: string will have static storage duration, whereas as a character array will not, unless it is explicitly specified by using the static keyword.

Actually, a string is a character array with following properties:

- \* the multibyte character sequence, to which we generally call string, is used to initialize an array of static storage duration. The size of this array is just sufficient to contain these characters plus the terminating NUL character.
- \* it not specified what happens if this array, i.e., string, is modified.
- \* Two strings of same value[1] may share same memory area.

**16. Write down the equivalent pointer expression for referring the same element `a[i][j][k][l]` ?**

```
a[i] == *(a+i)
a[i][j] == (*(a+i)+j)
a[i][j][k] == (*(a+i)+j)+k)
a[i][j][k][l] == (*(a+i)+j)+k)+l)
```

**17. Which bit wise operator is suitable for checking whether a particular bit is on or off?**

The bitwise AND operator. Here is an example:

```
enum {
KBit0 = 1,
KBit1,
...
KBit31,
};
if ( some_int & KBit24 )
printf ( "Bit number 24 is ON\n" );
else
printf ( "Bit number 24 is OFF\n" );
```

**18. Which bit wise operator is suitable for turning off a particular bit in a number?**

The bitwise AND operator, again. In the following code snippet, the bit number 24 is reset to zero.

```
some_int = some_int & ~KBit24;
```

**19. Which bit wise operator is suitable for putting on a particular bit in a number?**

The bitwise OR operator. In the following code snippet, the bit number 24 is turned ON:  
`some_int = some_int | KBit24;`

**20. Does there exist any other function which can be used to convert an integer or a float to a string?**

Some implementations provide a nonstandard function called `itoa()`, which converts an integer to string.

```
#include
char *itoa(int value, char *string, int radix);
DESCRIPTION
The itoa() function constructs a string representation of an integer.
PARAMETERS
value: Is the integer to be converted to string representation.
string: Points to the buffer that is to hold resulting string.
The resulting string may be as long as seventeen bytes.
radix: Is the base of the number; must be in the range 2 - 36.
A portable solution exists. One can use sprintf():
char s[SOME_CONST];
int i = 10;
float f = 10.20;
sprintf ( s, "%d %f\n", i, f );
```

## 21. Why does malloc(0) return valid memory address ? What's the use?

malloc(0) does not return a non-NULL under every implementation. An implementation is free to behave in a manner it finds suitable, if the allocation size requested is zero. The implementation may choose any of the following actions:

- \* A null pointer is returned.
- \* The behavior is same as if a space of non-zero size was requested. In this case, the usage of return value yields to undefined-behavior.

Notice, however, that if the implementation returns a non-NULL value for a request of a zero-length space, a pointer to object of ZERO length is returned! Think, how an object of zero size should be represented

For implementations that return non-NULL values, a typical usage is as follows:

```
void
func ( void )
{
int *p; /* p is a one-dimensional array, whose size will vary during the the lifetime of the
program */
size_t c;
p = malloc(0); /* initial allocation */
if (!p)
{
perror ("FAILURE" );
return;
}
/* ... */
while (1)
{
c = (size_t) ... ; /* Calculate allocation size */
p = realloc ( p, c * sizeof *p );
/* use p, or break from the loop */
```

```

/* ... */
}
return;
}

```

Notice that this program is not portable, since an implementation is free to return NULL for a malloc(0) request, as the C Standard does not support zero-sized objects.

## 22. Difference between const char\* p and char const\* p

In const char\* p, the character pointed by 'p' is constant, so u cant change the value of character pointed by p but u can make 'p' refer to some other location.

In char const\* p, the ptr 'p' is constant not the character referenced by it, so u cant make 'p' to reference to any other location but u can change the value of the char pointed by 'p'.

## 23. What is the result of using Option Explicit?

When writing your C program, you can include files in two ways. The first way is to surround the file you want to include with the angled brackets < and >. This method of inclusion tells the preprocessor to look for the file in the predefined default location. This predefined default location is often an INCLUDE environment variable that denotes the path to your include files.

For instance, given the INCLUDE variable

INCLUDE=C:\COMPILER\INCLUDE;S:\SOURCE\HEADERS; using the #include version of file inclusion, the compiler first checks the C:\COMPILER\INCLUDE directory for the specified file. If the file is not found there, the compiler then checks the S:\SOURCE\HEADERS directory. If the file is still not found, the preprocessor checks the current directory.

The second way to include files is to surround the file you want to include with double quotation marks. This method of inclusion tells the preprocessor to look for the file in the current directory first, then look for it in the predefined locations you have set up. Using the #include file version of file inclusion and applying it to the preceding example, the preprocessor first checks the current directory for the specified file. If the file is not found in the current directory, the C:\COMPILER\INCLUDE directory is searched. If the file is still not found, the preprocessor checks the S:\SOURCE\HEADERS directory.

The #include method of file inclusion is often used to include standard headers such as stdio.h or stdlib.h.

The #include file include nonstandard header files that you have created for use in your program. This is because these headers are often modified in the current directory, and you will want the preprocessor to use your newly modified version of the header rather than the older, unmodified version.

## **24. What is the benefit of using an enum rather than a #define constant?**

The use of an enumeration constant (enum) has many advantages over using the traditional symbolic constant style of #define. These advantages include a lower maintenance requirement, improved program readability, and better debugging capability.

- 1) The first advantage is that enumerated constants are generated automatically by the compiler. Conversely, symbolic constants must be manually assigned values by the programmer.
- 2) Another advantage of using the enumeration constant method is that your programs are more readable and thus can be understood better by others who might have to update your program later.
- 3) A third advantage to using enumeration constants is that some symbolic debuggers can print the value of an enumeration constant. Conversely, most symbolic debuggers cannot print the value of a symbolic constant. This can be an enormous help in debugging your program, because if your program is stopped at a line that uses an enum, you can simply inspect that constant and instantly know its value. On the other hand, because most debuggers cannot print #define values, you would most likely have to search for that value by manually looking it up in a header file.

## **25. What is the quickest sorting method to use?**

The answer depends on what you mean by quickest. For most sorting problems, it just doesn't matter how quick the sort is because it is done infrequently or other operations take significantly more time anyway. There are three sorting methods in this author's toolbox that are all very fast and that are useful in different situations. Those methods are quick sort, merge sort, and radix sort.

## **26. When should the volatile modifier be used?**

The volatile modifier is a directive to the compiler's optimizer that operations involving this variable should not be optimized in certain ways. There are two special cases in which use of the volatile modifier is desirable. The first case involves memory-mapped hardware (a device such as a graphics adaptor that appears to the computer's hardware as if it were part of the computer's memory), and the second involves shared memory (memory used by two or more programs running simultaneously).

## **27. When should the register modifier be used?**

The register modifier hints to the compiler that the variable will be heavily used and should be kept in the CPU's registers, if possible, so that it can be accessed faster.



## **28. How can you determine the size of an allocated portion of memory?**

You can't, really. `free()` can, but there's no way for your program to know the trick `free()` uses. Even if you disassemble the library and discover the trick, there's no guarantee the trick won't change with the next release of the compiler.

## **29. When does the compiler not implicitly generate the address of the first element of an array?**

Whenever an array name appears in an expression such as

- array as an operand of the size of operator
- array as an operand of `&` operator
- array as a string literal initializer for a character array

Then the compiler does not implicitly generate the address of the address of the first element of an array.

## **30. Why `n++` executes faster than `n+1` ?**

The expression `n++` requires a single machine instruction such as `INR` to carry out the increment operation whereas, `n+1` requires more instructions to carry out this operation.

## **31. Why doesn't the following statement work?**

```
char str[ ] = "Hello" ;  
strcat ( str, '!' ) ;
```

Answer: The string function `strcat()` concatenates strings and not a character. The basic difference between a string and a character is that a string is a collection of characters, represented by an array of characters whereas a character is a single character. To make the above statement work writes the statement as shown below:

```
strcat ( str, "!" ) ;
```

## **32. What is the benefit of using `#define` to declare a constant?**

Using the `#define` method of declaring a constant enables you to declare a constant in one place and use it throughout your program. This helps make your programs more maintainable, because you need to maintain only the `#define` statement and not several instances of individual constants throughout your program.

For instance, if your program used the value of pi (approximately 3.14159) several times, you might want to declare a constant for pi as follows: `#define PI 3.14159`

Using the `#define` method of declaring a constant is probably the most familiar way of declaring constants to traditional C programmers. Besides being the most common method of declaring constants, it also takes up the least memory.

Constants defined in this manner are simply placed directly into your source code, with no variable space allocated in memory. Unfortunately, this is one reason why most debuggers cannot inspect constants created using the `#define` method.

### **33. What is the purpose of `main( )` function ?**

The function `main( )` invokes other functions within it. It is the first function to be called when the program starts execution.

- It is the starting function
- It returns an int value to the environment that called the program
- Recursive call is allowed for `main( )` also.
- It is a user-defined function
- Program execution ends when the closing brace of the function `main( )` is reached.
- It has two arguments 1) argument count and 2) argument vector (represents strings passed).
- Any user-defined name can also be used as parameters for `main( )` instead of `argc` and `argv`

### **34. How can I search for data in a linked list?**

Unfortunately, the only way to search a linked list is with a linear search, because the only way a linked list's members can be accessed is sequentially. Sometimes it is quicker to take the data from a linked list and store it in a different data structure so that searches can be more efficient.

### **35. Why should we assign NULL to the elements (pointer) after freeing them?**

This is paranoia based on long experience. After a pointer has been freed, you can no longer use the pointed-to data. The pointer is said to dangle; it doesn't point at anything useful.

If you NULL out or zero out a pointer immediately after freeing it, your program can no longer get in trouble by using that pointer. True, you might go indirect on the null pointer instead, but that's something your debugger might be able to help you with immediately.

Also, there still might be copies of the pointer that refer to the memory that has been deallocated; that's the nature of C. Zeroing out pointers after freeing them won't solve all problems.

### 36. What is a null pointer assignment error? What are bus errors, memory faults, and core dumps?

These are all serious errors, symptoms of a wild pointer or subscript. Null pointer assignment is a message you might get when an MS-DOS program finishes executing. Some such programs can arrange for a small amount of memory to be available “where the NULL pointer points to (so to speak). If the program tries to write to that area, it will overwrite the data put there by the compiler.

When the program is done, code generated by the compiler examines that area. If that data has been changed, the compiler-generated code complains with null pointer assignment. This message carries only enough information to get you worried. There’s no way to tell, just from a null pointer assignment message, what part of your program is responsible for the error. Some debuggers, and some compilers, can give you more help in finding the problem.

Bus error: core dumped and Memory fault: core dumped are messages you might see from a program running under UNIX. They’re more programmer friendly. Both mean that a pointer or an array subscript was wildly out of bounds. You can get these messages on a read or on a write. They aren’t restricted to null pointer problems. The core dumped part of the message is telling you about a file, called core, that has just been written in your current directory. This is a dump of everything on the stack and in the heap at the time the program was running. With the help of a debugger, you can use the core dump to find where the bad pointer was used. That might not tell you why the pointer was bad, but it’s a step in the right direction. If you don’t have write permission in the current directory, you won’t get a core file, or the core dumped message

### 37. Predict the output or error(s) for the following programmes:

```
void main()
{
int const * p=5;
printf("%d",++(*p));
}
```

Answer: Compiler error: Cannot modify a constant value.

Explanation: p is a pointer to a "constant integer". But we tried to change the value of the "constant integer".

```
38. main()
{
char s[ ]="man";
int i;
for(i=0;s[ i ];i++)
printf("\n%c%c%c%c",s[ i ],*(s+i),*(i+s),i[s]);
}
```

Answer:

mmm

aaaa

nnnn

Explanation:  $s[i]$ ,  $*(i+s)$ ,  $*(s+i)$ ,  $i[s]$  are all different ways of expressing the same idea. Generally array name is the base address for that array. Here  $s$  is the base address.  $i$  is the index number/ displacement from the base address. So, indirecting it with  $*$  is same as  $s[i]$ .  $i[s]$  may be surprising. But in the case of C it is same as  $s[i]$ .

### 39. main()

```
{  
float me = 1.1;  
double you = 1.1;  
if(me==you)  
printf("I love U");  
else  
printf("I hate U");  
}
```

Answer: I hate U

Explanation: For floating point numbers (float, double, long double) the values cannot be predicted exactly. Depending on the number of bytes, the precision with of the value represented varies. Float takes 4 bytes and long double takes 10 bytes. So float stores 0.9 with less precision than long double.

Rule of Thumb: Never compare or at-least be cautious when using floating point numbers with relational operators ( $==$ ,  $>$ ,  $<$ ,  $<=$ ,  $>=$ ,  $!=$ ).

### 40. main()

```
{  
static int var = 5;  
printf("%d ",var--);  
  
if(var)  
main();  
}
```

Answer: 5 4 3 2 1

Explanation: When static storage class is given, it is initialized once. The change in the value of a static variable is retained even between the function calls. Main is also treated like any other ordinary function, which can be called recursively.

**41. main()**

```

{
int c[ ]={2.8,3.4,4,6.7,5};
int j,*p=c,*q=c;
for(j=0;j<5;j++) {
printf(" %d ",*c);
++q; }
for(j=0;j<5;j++){
printf(" %d ",*p);
++p; }
}

```

Answer: 2 2 2 2 2 3 4 6 5

Explanation: Initially pointer c is assigned to both p and q. In the first loop, since only q is incremented and not c, the value 2 will be printed 5 times. In second loop p itself is incremented. So the values 2 3 4 6 5 will be printed.

**42. main()**

```

{
extern int i;
i=20;
printf("%d",i);
}

```

Answer: Linker Error : Undefined symbol '\_i'

Explanation: extern storage class in the following declaration,  
extern int i;  
specifies to the compiler that the memory for i is allocated in some other program and that address will be given to the current program at the time of linking. But linker finds that no other variable of name i is available in any other program with memory space allocated for it. Hence a linker error has occurred .

**43. main()**

```

{
int i=-1,j=-1,k=0,l=2,m;
m=i++&& j++&& k++|| l++;
printf("%d %d %d %d %d",i,j,k,l,m);
}

```

Answer: 0 0 1 3 1

Explanation: Logical operations always give a result of 1 or 0. And also the logical AND (&&) operator has higher priority over the logical OR (||) operator. So the expression 'i++ && j++ && k++' is executed first. The result of this expression is 0 (-1 && -1 && 0 = 0).

Now the expression is  $0 \parallel 2$  which evaluates to 1 (because OR operator always gives 1 except for '0  $\parallel$  0' combination- for which it gives 0). So the value of m is 1. The values of other variables are also incremented by 1.

#### 44. main()

```
{  
char *p;  
printf("%d %d ",sizeof(*p),sizeof(p));  
}
```

Answer: 1 2

Explanation: The sizeof() operator gives the number of bytes taken by its operand. P is a character pointer, which needs one byte for storing its value (a character). Hence sizeof(\*p) gives a value of 1. Since it needs two bytes to store the address of the character pointer sizeof(p) gives 2.

#### 45. main()

```
{  
int i=3;  
switch(i)  
{  
default:printf("zero");  
case 1: printf("one");  
break;  
case 2:printf("two");  
break;  
case 3: printf("three");  
break;  
}  
}
```

Answer : Three

Explanation: The default case can be placed anywhere inside the loop. It is executed only when all other cases doesn't match.

#### 46. main()

```
{  
printf("%x",-1<<4);  
  
}
```

Answer: fff0

Explanation: -1 is internally represented as all 1's. When left shifted four times the least significant 4 bits are filled with 0's. The %x format specifier specifies that the integer value be printed as a hexadecimal value.

#### 47. main()

```
{
char string[]="Hello World";
display(string);
}
void display(char *string)
{
printf("%s",string);
}
```

Answer: Compiler Error: Type mismatch in redeclaration of function display

Explanation: In third line, when the function display is encountered, the compiler doesn't know anything about the function display. It assumes the arguments and return types to be integers, (which is the default type). When it sees the actual function display, the arguments and type contradicts with what it has assumed previously. Hence a compile time error occurs.

#### 48. main()

```
{
int c=- -2;
printf("c=%d",c);
}
```

Answer: c=2;

Explanation: Here unary minus (or negation) operator is used twice. Same maths rules applies, ie. minus \* minus= plus.

Note: However you cannot give like --2. Because -- operator can only be applied to variables as a decrement operator (eg., i--). 2 is a constant and not a variable.

#### 49. #define int char

```
main()
{
int i=65;
printf("sizeof(i)=%d",sizeof(i));
}
```

Answer: sizeof(i)=1

Explanation: Since the #define replaces the string int by the macro char

```
50. main()
{
int i=10;
i=!i>14;
Printf ("i=%d",i);
}
```

Answer: i=0

Explanation: In the expression !i>14 , NOT (!) operator has more precedence than '>' symbol. ! is a unary logical operator. !i (!10) is 0 (not of true is false). 0>14 is false (zero).

### 51. #include<stdio.h>

```
main()
{
char s[]={ 'a','b','c','\n','c','\0' };
char *p,*str,*str1;
p=&s[3];
str=p;
str1=s;
printf("%d",++*p + ++*str1-32);
}
```

Answer: 77

Explanation: p is pointing to character '\n'. str1 is pointing to character 'a' ++\*p. "p is pointing to '\n' and that is incremented by one." the ASCII value of '\n' is 10, which is then incremented to 11. The value of ++\*p is 11. ++\*str1, str1 is pointing to 'a' that is incremented by 1 and it becomes 'b'. ASCII value of 'b' is 98.

Now performing (11 + 98 – 32), we get 77("M"); So we get the output 77 :: "M" (Ascii is 77).

### 52. #include<stdio.h>

```
main()
{
int a[2][2][2] = { {10,2,3,4}, {5,6,7,8} };
int *p,*q;
p=&a[2][2][2];
*q=***a;
printf("%d----%d",*p,*q);
}
```



Answer: SomeGarbageValue---1

Explanation: `p=&a[2][2][2]` you declare only two 2D arrays, but you are trying to access the third 2D(which you are not declared) it will print garbage values.

`*q=***a` starting address of `a` is assigned integer pointer. Now `q` is pointing to starting address of `a`. If you print `*q`, it will print first element of 3D array.

### 53. `#include<stdio.h>`

```
main()
{
struct xx
{
int x=3;
char name[]="hello";
};
struct xx *s;
printf("%d",s->x);
printf("%s",s->name);
}
```

Answer: Compiler Error

Explanation: You should not initialize variables in declaration

### 54. `#include<stdio.h>`

```
main()

{
struct xx
{
int x;
struct yy
{
char s;
struct xx *p;
};
struct yy *q;
};
}
```

Answer: Compiler Error

Explanation: The structure yy is nested within structure xx. Hence, the elements of yy are to be accessed through the instance of structure xx, which needs an instance of yy to be known. If the instance is created after defining the structure the compiler will not know about the instance relative to xx. Hence for nested structure yy you have to declare member.

#### 55. main()

```
{
printf("\nab");
printf("\bsi");
printf("\rha");
}
```

Answer: hai

Explanation:

\n - newline  
\b - backspace  
\r - linefeed

#### 56. main()

```
{
int i=5;
printf("%d%d%d%d%d%d",i++,i--,++i,--i,i);
}
```

Answer: 45545

Explanation: The arguments in a function call are pushed into the stack from left to right. The evaluation is by popping out from the stack. And the evaluation is from right to left, hence the result.

#### 57. #define square(x) x\*x

```
main()
{
int i;
i = 64/square(4);
printf("%d",i);
}
```

Answer: 64

Explanation: the macro call square(4) will be substituted by 4\*4 so the expression becomes i = 64/4\*4. Since / and \* has equal priority the expression will be evaluated as (64/4)\*4 i.e. 16\*4 = 64

**58. main()**

```
{
char *p="hai friends",*p1;
p1=p;
while(*p!='\0') ++*p++;
printf("%s %s",p,p1);
}
```

Answer: ibj!gsjfoet

Explanation: ++\*p++ will be parse in the given order

\_ \*p that is value at the location currently pointed by p will be taken  
\_ ++\*p the retrieved value will be incremented  
\_ when; is encountered the location will be incremented that is p++ will be executed Hence, in the while loop initial value pointed by p is 'h', which is changed to 'i' by executing ++\*p and pointer moves to point, 'a' which is similarly changed to 'b' and so on. Similarly blank space is converted to '!'. Thus, we obtain value in p becomes "ibj!gsjfoet" and since p reaches '\0' and p1 points to p thus p1 does not print anything.

**59. #include <stdio.h>**

```
#define a 10
main()
{
#define a 50
printf("%d",a);
}
```

Answer: 50

Explanation: The preprocessor directives can be redefined anywhere in the program. So the most recently assigned value will be taken.

**60. #define clrscr() 100**

```
main()
{
clrscr();
printf("%d\n",clrscr());
}
```

Answer: 100

Explanation: Preprocessor executes as a separate pass before the execution of the compiler. So textual replacement of clrscr() to 100 occurs. The input program to compiler looks like this :

```
main()
{
100;
printf("%d\n",100);
}
```

Note: 100; is an executable statement but with no action. So it doesn't give any problem

### 61. main()

```
{
41printf("%p",main);
}8
```

Answer: Some address will be printed.

Explanation: Function names are just addresses (just like array names are addresses). main() is also a function. So the address of function main will be printed. %p in printf specifies that the argument is an address. They are printed as hexadecimal numbers.

### 62. main()

```
{
clrscr();
}
clrscr();
```

Answer: No output/error

Explanation: The first clrscr() occurs inside a function. So it becomes a function call. In the second clrscr(); is a function declaration (because it is not inside any function).

### 63. enum colors {BLACK,BLUE,GREEN}

```
main()
{
printf("%d..%d..%d",BLACK,BLUE,GREEN);
}
```

```
return(1);  
}
```

Answer: 0..1..2

Explanation: enum assigns numbers starting from 0, if not explicitly defined.

#### **64. void main()**

```
{  
char far *farther,*farthest;  
printf("%d..%d",sizeof(farther),sizeof(farthest));  
}
```

Answer: 4..2

Explanation: The second pointer is of char type and not a far pointer

#### **65. main()**

```
{  
int i=400,j=300;  
printf("%d..%d");  
}
```

Answer: 400..300

Explanation: printf takes the values of the first two assignments of the program. Any number of printf's may be given. All of them take only the first two values. If more number of assignments given in the program, then printf will take garbage values.

#### **66. main()**

```
{  
char *p;  
p="Hello";  
printf("%c\n",*p);  
}
```

Answer: H

Explanation: \* is a dereference operator & is a reference operator. They can be applied any number of times provided it is meaningful. Here p points to the first character in the string "Hello". \*p dereferences it and so its value is H. Again & references it to an address and \* dereferences it to the value H.

#### **67. main()**

```
{
int i=1;
while (i<=5)
{
printf("%d",i);
if (i>2)
goto here;
i++;
}
}
fun()
{
here:
printf("PP");
}
```

Answer: Compiler error: Undefined label 'here' in function main

Explanation: Labels have functions scope, in other words the scope of the labels is limited to functions. The label 'here' is available in function fun() Hence it is not visible in function main.

#### **68. main()**

```
{
static char names[5][20]={"pascal","ada","cobol","fortran","perl"};
int i;
char *t;
t=names[3];
names[3]=names[4];
names[4]=t;
for (i=0;i<=4;i++)
printf("%s",names[i]);
}
```

Answer: Compiler error: Lvalue required in function main

Explanation: Array names are pointer constants. So it cannot be modified.

**69. void main()**

```
{
int i=5;
printf("%d",i++ + ++i);
}
```

Answer: Output Cannot be predicted exactly.

Explanation: Side effects are involved in the evaluation of i

**70. void main()**

```
{
int i=5;
printf("%d",i+++++i);
}
```

Answer: Compiler Error

Explanation: The expression i+++++i is parsed as i ++ ++ + i which is an illegal combination of operators.

**71. #include<stdio.h>**

```
main()
{
int i=1,j=2;
switch(i)
{
case 1: printf("GOOD");
break;
case j: printf("BAD");

break;

}

}
```

Answer: Compiler Error: Constant expression required in function main.

Explanation: The case statement can have only constant expressions (this implies that we cannot use variable names directly so an error).

Note: Enumerated types can be used in case statements.

## 72. **main()**

```
{
int i;
printf("%d",scanf("%d",&i)); // value 10 is given as input here

}
```

Answer: 1

Explanation: Scanf returns number of items successfully read and not 1/0. Here 10 is given as input which should have been scanned successfully. So number of items read is 1.

## 73. **#define f(g,g2) g##g2**

```
main()
{
int var12=100;
printf("%d",f(var,12));
}
```

Answer: 100

## 74. **main()**

```
{
int i=0;
for(;i++;printf("%d",i)) ;
printf("%d",i);
}
```

Answer: 1



Explanation: before entering into the for loop the checking condition is "evaluated". Here it evaluates to 0 (false) and comes out of the loop, and i is incremented (note the semicolon after the for loop).

#### 75. #include<stdio.h>

```
main()
{
char s[]={'a','b','c','\n','c','\0'};
char *p,*str,*str1;
p=&s[3];
str=p;
str1=s;
printf("%d",++*p + ++*str1-32);
}
```

Answer: M

Explanation: p is pointing to character '\n'.str1 is pointing to character 'a' ++\*p "p is pointing to '\n' and that is incremented by one." the ASCII value of '\n' is 10. then it is incremented to 11. the value of ++\*p is 11. ++\*str1 "str1 is pointing to 'a' that is incremented by 1 and it becomes 'b'. ASCII value of 'b' is 98. Both 11 and 98 is added and result is subtracted from 32. i.e. (11+98-32)=77("M");

#### 76. #include<stdio.h>

```
main()
{
struct xx
{
int x=3;
char name[]="hello";
};
struct xx *s=malloc(sizeof(struct xx));
printf("%d",s->x);
printf("%s",s->name);
}
```

Answer: Compiler Error

Explanation: Initialization should not be done for structure members inside the structure declaration

**77. #include<stdio.h>**

```
main()
{
struct xx
{
int x;
struct yy
{
char s;
struct xx *p;
};
struct yy *q;
};
}
```

Answer: Compiler Error

Explanation: in the end of nested structure yy a member have to be declared.

**78. main()**

```
{
extern int i;
i=20;
printf("%d",sizeof(i));
}
```

Answer: Linker error: undefined symbol '\_i'.

Explanation: extern declaration specifies that the variable i is defined somewhere else. The compiler passes the external variable to be resolved by the linker. So compiler doesn't find an error. During linking the linker searches for the definition of i. Since it is not found the linker flags an error.

**79. main()**

```
{
printf("%d", out);
}
```

```
}  
int out=100;
```

Answer: Compiler error: undefined symbol out in function main.

Explanation: The rule is that a variable is available for use from the point of declaration. Even though a is a global variable, it is not available for main. Hence an error.

### 80. main()

```
{  
extern out;  
printf("%d", out);  
}  
int out=100;
```

Answer: 100

Explanation: This is the correct way of writing the previous program.

### 81. main()

```
{  
show();  
}  
void show()  
{  
printf("I'm the greatest");  
}
```

Answer: Compiler error: Type mismatch in redeclaration of show.

Explanation: When the compiler sees the function show it doesn't know anything about it. So the default return type (ie, int) is assumed. But when compiler sees the actual definition of show mismatch occurs since it is declared as void. Hence the error.

The solutions are as follows:

1. declare void show() in main() .
2. define show() before main().
3. declare extern void show() before the use of show().

### 82. main( )

```
{
int a[2][3][2] = {{ {2,4},{7,8},{3,4}},{{2,2},{2,3},{3,4}}};
printf(“%u %u %u %d \n”,a,*a,**a,***a);
printf(“%u %u %u %d \n”,a+1,*a+1,**a+1,***a+1);
}
```

Answer:

100, 100, 100, 2

114, 104, 102, 3

Explanation: The given array is a 3-D one. It can also be viewed as a 1-D array.

2	4	7	8	3	4	2	2	2	3	3	4	100	102	104	106	108	110	112
												114	116	118	120	122		

thus, for the first printf statement a, \*a, \*\*a give address of first element. since the indirection \*\*\*a gives the value. Hence, the first line of the output.

for the second printf a+1 increases in the third dimension thus points to value at 114, \*a+1 increments in second dimension thus points to 104, \*\*a +1 increments the first dimension thus points to 102 and \*\*\*a+1 first gets the value at first location and then increments it by 1. Hence, the output.

### 83. main( )

```
{
int a[ ] = {10,20,30,40,50},j,*p;
for(j=0; j<5; j++)
{
printf(“%d” ,*a);
a++;
}
p = a;
for(j=0; j<5; j++)
{
printf(“%d ” ,*p);
p++;
}
}
```

Answer: Compiler error: lvalue required.

Explanation: Error is in line with statement `a++`. The operand must be an lvalue and may be of any of scalar type for the any operator, array name only when subscripted is an lvalue. Simply array name is a non modifiable lvalue.

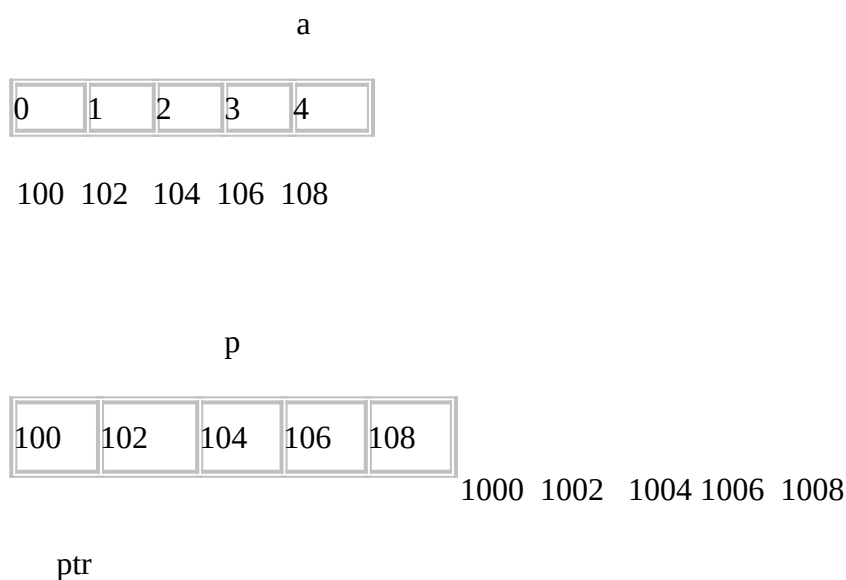
#### 84. `main( )`

```
{
static int a[ ] = {0,1,2,3,4};
int *p[ ] = {a,a+1,a+2,a+3,a+4};
int **ptr = p;
ptr++;
printf("\n %d %d %d", ptr-p, *ptr-a, **ptr);
*ptr++;
printf("\n %d %d %d", ptr-p, *ptr-a, **ptr);
*++ptr;
printf("\n %d %d %d", ptr-p, *ptr-a, **ptr);
++*ptr;
printf("\n %d %d %d", ptr-p, *ptr-a, **ptr);
}
```

Answer:

111  
222  
333  
344

Explanation: Let us consider the array and the two pointers with some address



1000

2000

After execution of the instruction `ptr++` value in `ptr` becomes 1002, if scaling factor for integer is 2 bytes. Now `ptr - p` is value in `ptr` - starting location of array `p`,  $(1002 - 1000) / (\text{scaling factor}) = 1$ ,

`*ptr - a` = value at address pointed by `ptr` - starting value of array `a`, 1002 has a value 102 so the value is  $(102 - 100) / (\text{scaling factor}) = 1$ , `**ptr` is the value stored in the location pointed by the pointer of `ptr` = value pointed by value pointed by 1002 = value pointed by 102 = 1. Hence the output of the first `printf` is 1, 1, 1.

After execution of `*ptr++` increments value of the value in `ptr` by scaling factor, so it becomes 1004. Hence, the outputs for the second `printf` are `ptr - p` = 2, `*ptr - a` = 2, `**ptr` = 2.

After execution of `***ptr` increments value of the value in `ptr` by scaling factor, so it becomes 1004. Hence, the outputs for the third `printf` are `ptr - p` = 3, `*ptr - a` = 3, `**ptr` = 3.

After execution of `****ptr` value in `ptr` remains the same, the value pointed by the value is incremented by the scaling factor. So the value in array `p` at location 1006 changes from 106 to 108. Hence, the outputs for the fourth `printf` are `ptr - p` =  $1006 - 1000 = 3$ , `*ptr - a` =  $108 - 100 = 4$ , `**ptr` = 4.

### 85. What is dangling pointer in c?

If any pointer is pointing the memory address of any variable but after some variable has deleted from that memory location while pointer is still pointing such memory location. Such pointer is known as dangling pointer and this problem is known as dangling pointer problem.

### 86. What are merits and demerits of array in c?

Merits:

- (a) We can easily access each element of array.
- (b) Not necessity to declare too many variables.
- (c) Array elements are stored in continuous memory location.

Demerits:

- (a) Wastage of memory space. We cannot change size of array at the run time.
- (b) It can store only similar type of data

### 87. Where are the auto variables stored?

Auto variables are stored in main memory and their default value is a garbage value.

### **88. Why Preincrement operator is faster than Postincrement?**

Evaluation of any expression is from left to right. Preincrement is faster because it doesn't need to save the current value for next instruction whereas Postincrement needs to save current value to be incremented after execution of current instruction.

### **89. Difference between arrays and linked list?**

Major differences between arrays and linked lists are: (i) In array consecutive elements are stored in consecutive memory locations whereas in linked list it is not so. (ii) In array address of next element is consecutive and whereas in linked list it is specified in the address part of each node. (iii) Linked List makes better use of memory than arrays. (iv) Insertion or deletion of an element in array is difficult than insertion or deletion in linked list

### **90. What is the use of typedef?**

- (i) It increases the portability.
- (ii) It simplifies the complex declaration and improves readability of the program.

### **91. What are library Functions?**

Library Functions are predefined functions and stored in .lib files.

### **92. What is a structure?**

Structure is a collection of heterogeneous (i.e. related data items which can be of different types) held together to a single unit. The data items enclosed within a structure are called its members which may be of data type int, float, char, array etc.

### **93. What is a pointer?**

Pointer is a variable that contains address of another variable in the memory. Pointers are quite useful in creation of linked data structures (such as linked list, trees, graphs), managing object allocated memory dynamically, optimize the program to execute faster and use less memory.

### **94. What are the techniques you use for debugging?**

- (i) Using compiler's features
- (ii) Read The Fine Manual
- (iii) printf( ) debugging
- (iv) Code grinding
- (v) Assertion

#### **95. What are macros? What are its advantages and disadvantages?**

Macro is a Pre-processor. Major advantage of using the macro is to increase the speed of the execution of the program.

Major disadvantage of the macros are:

- (i) No type checking is performed in macro. This may cause error.
- (ii) A macro call may cause unexpected results.

#### **96. What is difference between Structure and Unions?**

- (i) In structure every member has its own memory whereas in union its members share the same member space.
- (ii) In structure, it is possible to initialize all the members at the same time which is not possible in case of union.
- (iii) A structure requires more space than union (for the same type of members).
- (iv) In union different interpretations of the same memory space are possible which is not so in case of structures.

#### **97. What are the advantages of using Unions?**

- (i) Efficient use of memory as it does not demand memory space for its all members rather it requires memory space for its largest member only.
- (ii) Same memory space can be interpreted differently for different members of the union.

#### **98. What is the difference between ordinary variable and pointer in C?**

An ordinary variable is like a container it can hold any value and we can change the value of ordinary variable at a time throughout the program. A pointer is a variable that stores the address of another Variable.

#### **99. What are segment and offset addresses?**

When paging technique is performed, the page will break into segments and its sequence is said to be segments and its width can be said as offset. In short, segment is a physical address and offset is logical address.



### **100. When should a type cast be used?**

There are two situations in which to use a type cast. The first use is to change the type of an operand to an arithmetic operation so that the operation will be performed properly. The second case is to cast pointer types to and from void \* in order to interface with functions that expect or return void pointers. For example, the following line type casts the return value of the call to malloc() to be a pointer to a foo structure.

```
struct foo *p = (struct foo *) malloc(sizeof(struct foo));
```

### **101. What is the difference between %d and %\*d in c language?**

%d give the original value of the variable and %\*d give the address of the variable.

```
eg:-int a=10,b=20;
printf("%d%d",a,b);
printf("%*d%*d",a,b);
```

Result is 10 20 1775 1775 .Here 1775 is the starting address of the memory allocation for the integer.a and b having same address because of contagious memory allocation.

### **102. How does a C program come to know about command line arguments?**

When we execute our C program, operating system loads the program into memory. In case of DOS, it first loads 256 bytes into memory, called program segment prefix. This contains file tables,environment segment, and command line information. When we compile the C program the compiler inserts additional code that parses the command, assigning it to the argv array, making the arguments easily accessible within our C program.

### **103. How are pointer variables initialized?**

Pointer variable are initialized by one of the following two ways

- Static memory allocation
- Dynamic memory allocation

### **104. What is modular programming?**

If a program is large, it is subdivided into a number of smaller programs that are called modules or subprograms. If a complex problem is solved using more modules, this approach is known as modular programming

**105. Where does global, static, local, register variables and C Program instructions get stored?**

Global , static, local : In main memory  
Register variable: In registers  
C program : In main memory.

**106. Where are the auto variables stored?**

Auto variables are stored in main memory and their default value is a garbage value.

**107. What is an lvalue?**

An lvalue is an expression to which a value can be assigned. The lvalue expression is located on the left side of an assignment statement, whereas an rvalue is located on the right side of an assignment statement. Each assignment statement must have an lvalue and an rvalue. The lvalue expression must reference a storable variable in memory. It cannot be a constant

**108. What is an argument? Differentiate between formal arguments and actual arguments?**

An argument is an entity used to pass the data from calling function to the called function. Formal arguments are the arguments available in the function definition. They are preceded by their own data types. Actual arguments are available in the function call.

**109. When is a switch statement better than multiple if statements?**

A switch statement is generally best to use when you have more than two conditional expressions based on a single variable of numeric type.

**110. Differentiate between a linker and linkage?**

A linker converts an object code into an executable code by linking together the necessary build in functions. The form and place of declaration where the variable is declared in a program determine the linkage of variable.

### 111. Define Operator, Operand, and Expression in 'C'?

Operators are symbols which take one or more operands or expressions and perform arithmetic or logical computations.

Operands are variables or expressions which are used in operators to evaluate the expression. Combination of operands and operators form an expression.

### 112. What will be the result of the following code?

```
#define TRUE 0 // some code
while(TRUE)
{
    // some code
}
```

Answer: This will not go into the loop as TRUE is defined as 0.

### 113. What will be printed as the result of the operation below:

```
main()
{
    int a=0;
    if(a==0)
        printf("Cisco Systemsn");
        printf("Cisco Systemsn");
}
```

Answer: Two lines with "Cisco Systems" will be printed.

### 114. Do you know pragma directives in c?

Pragma is implementation specific directive i.e each pragma directive has different implementation rule and use. If compiler does not recognize particular pragma it simply ignore that pragma statement without showing any error or warning message and execute the whole program assuming this pragma statement is not present.

### 115. Predict the output or error

```
main()
{
    clrscr();
}
```

```
}  
clrscr();
```

Ans: No output/error

Explanation: The first `clrscr()` occurs inside a function. So it becomes a function call. In the second `clrscr();` is a function declaration (because it is not inside any function).

### 116. Predict the output or error

```
enum colors {BLACK,BLUE,GREEN}  
main()  
{  
  
printf("%d..%d..%d",BLACK,BLUE,GREEN);  
  
return(1);  
}
```

Answer: 0..1..2

Explanation: enum assigns numbers starting from 0, if not explicitly defined.

### 117. Predict the output or error

```
main()  
{  
int i;  
printf("%d",scanf("%d",&i)); // value 10 is given as input here  
}
```

Answer: 1

Explanation: `scanf` returns number of items successfully read and not 1/0. Here 10 is given as input which should have been scanned successfully. So number of items read is 1.

### 118. what will be the position of the file marker?

- a: `fseek(ptr,0,SEEK_SET);`
- b: `fseek(ptr,0,SEEK_CUR);`

Ans: a: The `SEEK_SET` sets the file position marker to the starting of the file.

b: The `SEEK_CUR` sets the file position marker to the current position of the file.

### 119. Predict the output or error

```
main()
```

```

{
    main();
}

```

Ans: Runtime error : Stack overflow.

Explanation: main function calls itself again and again. Each time the function is called its return address is stored in the call stack. Since there is no condition to terminate the function call, the call stack overflows at runtime. So it terminates the program and results in an error.

### 120. Predict the output or error

```

main()
{
    int i=5,j=6,z;
    printf("%d",i+++j);
}

```

Answer:11

Explanation:the expression i+++j is treated as (i++ + j)

### 121. Predict the output or error

```

main()
{
    int k=1;
    printf("%d==1 is \"%s\",k,k==1?"TRUE":"FALSE");
}

```

Ans: 1==1 is TRUE

Explanation: When two strings are placed together (or separated by white-space) they are concatenated (this is called as "stringization" operation). So the string is as if it is given as "%d==1 is %s". The conditional operator( ?: ) evaluates to "TRUE".

### 122. What is use of void data type?

Void is an empty data type normally used as a return type in C/C++, C#, Java functions/methods to declare that no value will be return by the function.

The another used of void is to declare the pointer in C/C++ where It is not sure what data type is addressed by the pointer.

### 123. four type of scope in c:

Block scope.  
Function scope.  
File scope.  
Program scope.

**124. Tell any five properties of auto variables?**

auto variables are defined inside a function. A variable declared inside the function without storage class name is, by default, an auto variable. These functions are declared on the stack. The stack provides temporary storage.

**125. What is automatic type promotion in c?**

In c if two operands are of different data type in a binary operation then before performing any operation compiler will automatically convert the operand of lower data type to higher data type .This phenomenon is known as automatic type conversion. For example:

```
int a=10,c;  
float b=5.5f;  
c=a+b;
```

Here a int variable while b is float variable. So before performing addition operation value of the variable a (Lower data type) will automatically convert into float constant (higher data type) then it will perform addition operation.

**126. What are differences between sizeof operator and strlen function?**

sizeof is keyword of c which can find size of a string constant including null character but strlen is function which has been defined string.h and can find number of characters in a string excluding null character.

**127. What is command line argument?**

Getting the arguments from command prompt in c is known as command line arguments. In c main function has three arguments.

They are:

- Argument counter
- Argument vector
- Environment vector

**128. void main(){**

```
int x=5,y=10,z=15,val;
```

```

    val=sum(x,(y=0,z=0,y),z);
    clrscr();
    printf("%d",val);
    getch();
}
sum(int x,int y,int z){
    return x+y+z;
}

```

Answer:20

Explanation: In the above program comma after Y=0 &Z=0 are behaving as operator.

### 129. what is nested structure?

A structure is a collection of one or more variables, possibly of different data types, grouped together under a single name for convenient handling. Structures can contain other structures as members; in other words, structures can nest.

### 130. What is slack byte in structure?

To store any type of data in structure there is minimum fixed byte which must be reserved by memory. This minimum byte is known as word boundary. Word boundary depends upon machine. TURBO C is based on 8086 microprocessor which has two byte word boundary. So any data type reserves at least two byte space.

### 131.What is prototype of printf function?

Prototype of printf function is:

```
int printf( const char *format ,...)
```

### 132.What is difference between declaration and definition?

During declaration we just specify the type and no memory is allocated to the variable. But during the definition an initial value is assigned and memory is allocated to the variable.

### 133. What is function recursion?

When a function of body calls the same function then it is called as 'recursive function.'

Example:

```

Recursion()
{
    printf("Recursion !");
    Recursion();
}

```

#### **134. What is self referential structure ?**

A self-referential structure is one of the data structures which refer to the pointer to (points) to another structure of the same type.

#### **135. What is far pointer?**

The pointer which can point or access whole the residence memory of RAM i.e. which can access all 16 segments is known as far pointer.

#### **136. What is pascal and cdecl keyword in c language?**

There are two types of parameters passing conventions in c:

1. pascal: In this style function name should (not necessary ) in the uppercase .First parameter of function call is passed to the first parameter of function definition and so on.
2. cdecl: In this style function name can be both in the upper case or lower case. First parameter of function call is passed to the last parameter of function definition. It is default parameter passing convention.

#### **137. What is use of #pragma inline directive in c language?**

#pragma inline only tells the compiler that source code of program contain inline assembly language code .In c we can write assembly language program with help of asm keyword.

#### **138. What is the meaning of multilevel pointers in c?**

A pointer is pointer to another pointer which can be pointer to others pointers and so on is known as multilevel pointers. We can have any level of pointers.

#### **139. What is huge pointer in c?**



The pointer which can point or access whole the residence memory of RAM i.e. which can access all the 16 segments is known as huge pointer.

**140. Is it possible to rename any function in c?**

Yes, we can rename any function using typedef keyword. It is useful when function declaration is too complex and we have to give any simple name or if we have to create more numbers of function of the same type.

**141. Do you know, what is the meaning and use of static keyword in c?**

Keyword static is used for declaring static variables in c. This modifier is used with all data types like int, float, double, array, pointer, structure, function etc.

**142. What is difference between .com program and .exe program?**

Both .com and .exe program are executable program but .com program execute faster than .exe program. All drivers are .com program. .com file has higher preference than .exe For example:

**143. Difference between TSR and TSO program**

TSO means terminate but stay outside. It is that program, which release the main memory after the execution of the program. Example ms paint, notepad, turbo c compilers etc.

TSR means terminate but stay residence .It is those program, which after the execution of the program does not release the RAM (main memory).e.g. antivirus.

**144. Describe turbo c compiler?**

Turbo c is an IDE of c programming language created by Borland. Turbo C 3.0 is based on MS DOS operation system. It is one of the most popular c compilers. It uses 8086 microprocessor which is 16 bit microprocessor. It has 20 address buses and 16 data bus. Its word length is two byte.

**145. Out of fgets() and gets() which function is safe to use and why?**

fgets() is safer than gets(), because we can specify a maximum input length. Neither one is completely safe, because the compiler can't prove that programmer won't overflow the buffer he pass to fgets ().

#### **146. Difference between strdup and strcpy?**

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 .

#### **147. Differentiate between a for loop and a while loop? What are it uses?**

For executing a set of statements fixed number of times we use for loop while when the number of

iterations to be performed is not known in advance we use while loop.

#### **148. What is storage class? What are the different storage classes in C?**

Storage class is an attribute that changes the behavior of a variable. It controls the lifetime, scope and linkage. The storage classes in c are auto, register, and extern, static, typedef.

#### **149. What are the uses of a pointer?**

- (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.

#### **150.In header files whether functions are declared or defined?**

Functions are declared within header file. That is function prototypes exist in a header file,not function bodies. They are defined in library (lib).

#### **151. Difference between pass by reference and pass by value?**

Pass by reference passes a pointer to the value. This allows the callee to modify the variable directly.Pass by value gives a copy of the value to the callee. This allows the callee to modify the value without modifying the variable. (In other words, the callee simply cannot modify the variable, since it lacks a reference to it.)

**152. What are enumerations?**

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.

**153. Are pointers integer?**

No, pointers are not integers. A pointer is an address. It is a positive number.

**154. What is 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.

**155. What is 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.

**156. What modular programming?**

If a program is large, it is subdivided into a number of smaller programs that are called modules or subprograms. If a complex problem is solved using more modules, this approach is known as modular programming

**157. What is a function?**

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.

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

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.

**159. what are C tokens?**

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

**160. What are C identifiers?**

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.

**161. Difference between syntax vs logical error?**

Syntax Error

These involve validation of syntax of language.  
compiler prints diagnostic message.

Logical Error

logical error are caused by an incorrect algorithm or by a statement mistyped in such a way that it doesn't violate syntax of language.  
difficult to find.

**162. What are the facilities provided by preprocessor?**

file inclusion

substitution facility

conditional compilation

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

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

**164. What is FILE?**

FILE is a predefined data type. It is defined in `stdio.h` file.

**165. What is a file?**

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.