Top 50 C Programming Interview Questions and Answers

Q1. You know what are Increment and Decrement operators correct? What I want to know is how increment and decrement operators can be constructed using basic mathematical operators?

Ans. (*First, in a line explain Increment and Decrement operators then answer the question*)

Increment and Decrement operators are used to increase or decrease the value of the variable by 1. We can achieve this functionality by using '+' and '-' operators. For eg:

```
int x=1;
int y= x++; // this can be divided into two steps (y=x; x=x+1;)
int z= --x; // this can be divided into two steps (x=x+1; z=x;)
```

(*Such example will show your understanding of post and pre, increment/decrement also*)

Q2. How will you print "Hello World" without using semicolon even once in your code?

Ans. (*Simply write the below code and explain it's working*)

```
int main(void)
{
if(printf("Hello World"))
{
}
}
```

printf() function will print "Hello World" to the console. It will return a non-zero value to *if* statement and there is no other code to execute.

(*Interesting fact*: printf returns the number of characters(including space) inside the inverted quotes. For the above example, it will return 11.)

Q3. Do you know how many places a variable can be declared in C? If you do, then mention the places and the type of variable corresponding to that place.

Ans. (*Name all the 3 places where the variables can be declared*)

Variables can be declared in 3 places:

1. Inside functions
2. In the definition of function parameters
3. Outside all functions.

These positions correspond to

1. Local variables
2. Formal parameters
3. Global variables

Q4. Explain Local variables, Formal parameters and Global variables.

Ans. Local Variables:

1. Variables that are declared inside a function are called local variables.

2. Local variables exist only while the block of code in which they are declared is executing, i.e., a local variable is created upon entry into its block and destroyed upon exit.

(*Interesting fact*: A variable declared within one block has no relationship to another variable with the same name declared within a different code block.)

Formal Parameters:

1. If a function is to use arguments, it must declare variables that will accept the values of the arguments. These variables are called the formal parameters of the function.
2. They behave like any other local variables inside the function.

Global Variables:

1. Global variables are declared outside any function.
2. Unlike local variables, global variables are known throughout the program and may be used by any piece of code.
3. Also, they will hold their value throughout the program's execution.

(*Interesting fact*: After defining a local variable, the system or the compiler won't be initializing any value to it. Global variables get initialized automatically by the compiler as and when defined based on datatype. int = 0, char = '', float = 0, double = 0 and pointer = Null)

Q5. Explain to me the concept of Null pointer. And if you can, shed some light on Dangling pointer as well.

Ans. If the pointer variable is not assigned any valid memory address then NULL can be used to initialize that pointer. (Conceptually, the NULL pointer doesn't point anywhere)

(*Interesting fact*: we have a void pointer as well. A pointer that points to some data location in storage, which doesn't have any specific type, so "void" pointer).

Dangling pointer: Pointer that doesn't point to a valid memory location. (*Don't stop here it will give the interviewer a chance to confuse you, provide a detailed answer*)

If you delete or deallocate a memory of an object without modifying the pointer pointing to it, in such cases dangling pointer arises.

```
int* dang = (int *)malloc(sizeof(int));
free(dang); // deallocated the memory used by the dang variable.
*dang = 100; // now the pointer is pointing to invalid memory locations. So, this becomes a danglir
```

Q6. In a C language, how many ways are there of passing arguments to a subroutine(function)?

Ans. In C language there are two ways that arguments can be passed to a subroutine.

1. Call by Value
2. Call by Reference

(Now explain these two points)

1. Call by Value: This method copies the *value* of an argument into the formal parameter of the subroutine. In this case, changes made to the parameter *do not affect the argument*. (This is how we pass the arguments usually)
2. Call by Reference: In this method, the *address of an argument is copied* into the parameter. Inside the subroutine, the address is used to access the actual argument used in the call. This means that changes made to the parameter *affect the argument*. (This is when pointers arrive into the picture)

(*You can stop here, or you can also provide a simple Swap function example for Call by Reference*)

```
#include<stdio.h>
void swap(int *x, int *y)
{
int temp;
temp=*x;  // store the value present at address of ‘x’
*x=*y;    // copy the value present at the address of ‘y’ to address of ‘x
*y=temp;  // copy the value present in temp to address of ‘y’
}
int main(void)
{
int i=10, j=20;
swap(&i, &j);
printf("i=%d and j=%d",i,j);
return 0;
}
```

Q7. Is there a way in C programming such that the local variables maintain their values between function calls?

(Sometimes a question is not asked directly, such as above question could be asked simply as - *Explain use of static keyword in C language*)

Ans. (*First, explain **static** keyword*)

Variables declared as **static** are permanent variables within their own function or file. They are not known outside their function or file.

(*Now, explain only about **static** Local Variable, no need to explain about **static** Global Variables*)

static Local Variables: When you apply the **static** modifier to a local variable, the compiler creates permanent storage for it, much as it creates storage for a global variable.

The key difference between a **static** local variable and a global variable is that the **static** local variable remains known only to the block in which it is declared.

In simple terms a **static** local variable is a local variable that retains its value between function calls.

```
#include <stdio.h>
  int product();
    int main()
        {
      product();
       return 0;
        }
   int product()
        {
static int var = 1; // var initialized to 1
var=var*5; // updates value of var, this will be retained after recursive call as well
       if(var>1000)
            return;
    printf("%d, ", var);
        product();
        }
```

Q8. You can easily print numbers from 1 to 100 using a loop, can you do the same without using any type of loop?

Ans. (Use recursion instead loop)

```
#include<stdio.h>
void printNumbers(int);
      int main()
        {
         int n=1;
      printNumbers(n);
         return 0;
        }
void printNumbers(int n)
        {
        if(n<=100)
          {
    printf("%d     ", n);
      printNumbers(n+1);
        }
        }
```

Q9. In the lines #include <stdio.h>, #define MOD 10000007, what are #include and #define?

Ans. (*Interviewer is expecting an explanation of Preprocessor Directives*)

#include and #define are Preprocessor directives. Preprocessor directives are placed at the beginning of every C program. This is where library files are specified, another use is to declare constants. Preprocessors are programs that process our source code before compilation.

#include - It is used to include libraries, which would depend on what functions are to be used in the program.

#define - It defines an identifier and a character sequence that will be substituted for the identifier each time it is encountered in the source file. The identifier is referred to as a *macro name* and the replacement process as *macro replacement*.

(*Interesting fact*: Each preprocessing directive must be on its own line. For example, this will not work -> #include <stdio.h> #include<stdlib.h>)

Q10. Explain about Entry control and Exit control loops in C.

Ans.

1. Entry control: This loop is categorized in 2 part

1. while loop

2. for loop

2. Exit control: In this category, there is only one type of loop known as

1. do while loop

Q11. Memory layout of C consist of Text, Data, Stack and Heap areas, explain about stack and heap areas?

Ans. Stack Area: It is used to store local variables and is used for **passing arguments** to the functions along with the return address of the instruction which is to be executed after the function call is over. These values stay in the memory only till the termination of that function.

Heap Area: Heap is a segment where dynamic memory allocation usually takes place. In C language dynamic memory allocation is done by using malloc() and calloc() functions.

Q12. In which memory segment of the C program, static variables are stored?

Ans. (There are two cases here, explain both to the interviewer)

1. static variables that do not have an explicit initialization or are initialized to zero are stored in the uninitialized data segment(also known as the BSS segment).
2. static variables that are initialized are stored in the initialized data segment.

Q13. What functions are used for dynamic memory allocation in C language?

Ans. Standard C defines four dynamic allocation functions that all compilers will supply:

1. calloc()
2. malloc()
3. free()
4. realloc()

1. calloc() :

1. The **calloc()** function takes two arguments (n, sizeof n), and allocated (n*size) memory. That is, **calloc()** allocates enough memory for an array of *num* objects of size *size*. All bits in the allocated memory are initially set to zero.
2. The **calloc()** function returns a pointer to the first byte of the allocated region. If there is not enough memory to satisfy the request, a null pointer is returned.

```
#include<stdlib.h>
void *calloc(num, size);
```

2. malloc():

1. The **malloc()** function returns a pointer to the first byte of a region of memory of size *size* that has been allocated from the heap.
2. If there is insufficient memory in the heap to satisfy the request, **malloc()** returns a null pointer.

```
#include <stdlib.h>
void *malloc(size_t size);
```

3. free():

1. The **free()** function deallocates the memory assigned to a pointer by one of the dynamic allocation functions.
2. This makes the memory available for future allocation.

```
#include <stdlib.h>
void free(void *ptr);
```

4. realloc():

1. In simple English, **realloc()** is used to change the size of previously allocated memory pointed to by the pointer.
2. For C89, **realloc()** changes the size of the previously allocated memory pointed to by *ptr* to that specified by *size*.
3. For C99, the block of memory pointed to by *ptr* is freed, and a new block of specified *size* is allocated, and the previous content is copied.

```
#include <stdlib.h>
void *realloc(void *ptr, int size)
```

Q14. Can we convert a higher data type into lower data type in C? If yes, then how?

Ans. (Typecasting is asked here)

Yes, we can convert higher data type into lower data type using explicit typecasting. **Typecasting** is a way to convert a variable from one data type to another data type.

But it is not a good practice to convert the higher data type to lower data type, because data will be truncated when the higher data type is converted into a lower data type, resulting in data loss.

If a float is converted to int, then we will lose values after decimal point (5.550 -> 5, a loss of .550).

Q15. Explain all the types of recursion.

Ans.

1. Linear Recursion

1. In linear recursion, a function calls exactly once to itself each time the function is invoked.
2. For ex: Find maximum in an Array.

```
int recursiveMax(int arr[],int n)
{
if(n==1)
return arr[0];
return max(recursiveMax(arr,n-1),arr[n-1]);// max is a function to calculate maximum
}
```

3. As you can see in the above code, recursiveMax() is called exactly ones each time the function is called.
4. Also keep in mind, though recursiveMax() call is in the last statement, it is not the last operation, last operation here is returning Maximum element.

2. Tail Recursion

1. Tail recursion is another form of linear recursion, where the function makes a recursive call as its very last operation.
2. For example, printing 1 to 100

```
void printNumbers(int n)
{
if(n<=100)
{
printf("%d  ", n);
printNumbers(n+1);
}
}
```

3. Here printNumbers() is the last operation, so it is a tail recursion.

3. Binary Recursion

1. In binary recursion a function makes two recursive calls to itself when invoked.
2. For example, Fibonacci series using recursion.

```
int recursiveFib(int n)
{
if(n<=1)
return n;
return recursiveFib(n-1) + recursiveFib(n-2);
}
```

4. Multiple Recursion

1. Multiple recursion can be treated as a generalized form of binary recursion.
2. When a function makes multiple recursive calls probably more than two, it is called multiple recursion.

(*Interesting fact*: Recursion can also be divided into Direct and Indirect Recursion based on the place where a recursive function is called.
Direct recursion – Function calls itself from within itself.
Indirect recursion – Two functions call one another mutually.)

Q16. Write a program to swap two numbers without using the third variable?

Ans.

```
#include<stdio.h>
int main()
{
int x=50, y=60;
x=x+y; // x=110 (50+60)
y=x-y; // y=110-60 =50
x=x-y; // x=110-50 =60
// so now x=60 and y=50.
printf("x=%d and y=%d",x,y);
return 0;
```

```
}
```

Q17. Write a program to find Fibonacci series without using recursion?

Ans.

```
#include<stdio.h>
   void main()
        {
int num1=0,num2=1,num3,i,number;
printf("Enter the number of elements: ");
     scanf("%d",&number);
printf("%d %d",num1,num2);
   for(i=2;i<number;i++)
          {
     num3=num1+num2;
   printf(" %d",num3);
       num1=num2;
       num2=num3;
          }
        }
```

Q18. Write a program to print Fibonacci series using recursion?

Ans.

```
#include<stdio.h>
  int fib(int);
   int main()
        {
    int num=0,i,n;
    scanf("%d",&n);
printf("Fibonacci series:
       ");
    for(i=1;i<=n;i++)
          {
   printf("%d ",fib(num));
           num++;
          }
        }
  int fib(int n);
        {
   if(n==0 || n==1)
        return n;
         else
return (fib(n-1))+fib(n-2));
        }
```

Q19. Write a C program to check whether a number is Prime or not.

Ans.

```
#include<stdio.h>
#include<math.h>
   int main()
        {
    int num,i;
    int flag=0;
printf("Enter any positive number:
```



```
        {
  if(num%i==0)
       {
     flag=1;
      break;
       }
      }
  if(flag==0)
       {
```

```
    printf("%d is Prime Number",num);
                }
              else
                {
printf("%d is not a Prime Number",num);
                }
             return 0;
               }
```

Q20. Write a program to check whether the number is prime or not using recursion.

Ans.

```
          #include<stdio.h>
       int isPrime(int,int);
            int main()
                {
             int n,i=2;
printf("Enter any positive number:
             ");
            scanf("%d",&n);
          if(isPrime(n,i)==1)
                {
             printf("Yes");
                }
              else
                {
             printf("No");
                }
               }
    int isPrime(int n, int i)
                {
              if(n<2)
               return 0;
             if(n==2)
               return 1;
            if(n%i==0)
               return 0;
           if(i*i > n)
               return 1;
      return isPrime(n,i+1);
                }
```

Q21. Write a program to check whether a number is Palindrome number or not in C programming.

Ans.

```
          #include<stdio.h>
              int main()
                {
           int num,rem,sum=0,temp;
        printf("enter the number=");
             scanf("%d",&num);
                temp=num;
              while(num>0)
                 {
                rem=num%10;
              sum=(sum*10)+rem;
                num=num/10;
                  }
             if(temp==sum)
         printf("Palindrome number ");
                 else
         printf("Not palindrome");
              return 0;
                }
```

Q22. Write a program to print factorial (without recursion) in C programming.

Ans.

```
          #include<stdio.h>
              int main()
                {
```

```
int i,fact=1,num;
printf("Enter a number: ");
scanf(" %d ",&numb);
for(i=1;i<=numb;i++)
{
fact=fact*i;
}
printf("Factorial of %d is: %d",num,fact);
return 0;
}
```

Q23. Write a program to print factorial (using recursion) in C programming.

Ans.

```
#include <stdio.h>
long int factorial(int n);
int main()
{
int n;
printf("Enter a positive integer: ");
scanf("%d", &n);
printf("Factorial of %d = %ld", n, factorial(n));
return 0;
}
long int factorial(int n)
{
if (n >= 1)
return n*factorial(n-1);
else
return 1;
}
```

Q24. Write a C program to check whether a number is Armstrong or not.

Ans.

```
#include <stdio.h>
#include <math.h>
int main()
{
int number, tempNum, rem, result = 0, n = 0 ;
printf("Enter an integer: ");
scanf("%d", &number);
tempNum = number;
while (tempNum != 0)
{
tempNum /= 10;
++n;
}
tempNum = number;
while (tempNum != 0)
{
rem = tempNum%10;
result += pow(rem, n);
tempNum /= 10;
}
if(result == number)
printf("%d is an Armstrong number.", number);
else
printf("%d is not an Armstrong number.", number);
return 0;
}
```

Q25. Write a C program to reverse an Integer.

Ans.

```
#include <stdio.h>
int main()
{
```

```
int n, reverseNum = 0, rem;
printf("Enter an integer: ");
scanf("%d", &n);
while(n != 0)
{
rem = n%10;
reverseNum = reverseNum*10 + rem;
n /= 10;
}
printf("Reversed Number = %d", reverseNum);
return 0;
}
```

Q26. Write a C program to find the sum of two integers without using '+' operator.

Ans.

```
#include<stdio.h>
int main()
{
int x=10;
int y=20;
int sum = -(-x-y); // -(-x-y)= x+y, simple mathematics trick
printf("%d",sum);
return 0;
}
```

Q27. Write a C program to print “Hello World” with the double-quotes.

Ans.

```
#include<stdio.h>
int main()
{
printf(""Hello World"");// use escape sequence
return 0;
}
```

Q28. Write a C program to divide an integer by 2 without using ‘/’ operator.

Ans.

```
#include<stdio.h>
int main()
{
int n=56;
printf("%d", n>>1);
}
```

(Interesting fact: Right shift operator is used to divide a number by powers of 2. N>>1 == N/2, N>>2 == N/4, N>>3 == N/8. Same is the case with Left shift operator, it is used for multiplication.)

Q29. Write a C program to reverse a String in C with the help of recursion.

Ans.

```
#include<stdio.h>
void reverseStr(char[]);
void reverseStr(char str[])
{
if(str[0]!='')
{
reverseStr(str+1);
printf("%c",str[0]);
}
}
int main()
{
char str[] ="PrepBytes";
reverseStr(str);
return 0;
```

```
}
```

Q30. Can the size of an array be declared at runtime in C?

Ans. No, in array declaration, the size must be known at compile time. You can't specify a size that's known only at runtime. The compiler needs to store memory for the array, and to do that compiler requires Array's size at compile. For example:

```
int size;
int arr[size]; // this is wrong as size is not initialized yet.
// The correct code is
int size;
scanf("%d", &size);
int arr[size];
```

Q31. Write a C program to convert String into Integer without using library function.

Ans.

```
#include <stdio.h>
#include <string.h>
int main()
{
char str[10]; // any string greater than 10, will be outside integer range.
int num = 0, i, j, len;
printf("Enter a number String: ");
gets(str);
len = strlen(str);
for(i=0; i<len; i++){
num = num * 10 + ( str[i] - '0' );
}
printf("%d", num);
return 0;
}
```

Q32. Write a C program to convert an integer into a string without using library function.

Ans.

```
#include <stdio.h>
#include <string.h>
int main()
{
int num,result,rem,len=0,n,i;
char str[10];
printf("Enter a number: ");
scanf("%d",&num);
n=num;
while(n!=0)
{
len++;
n=n/10;
}
for(i=0;i<len;i++)
{
rem=num%10;
num=num/10;
str[len-(i+1)]=rem+'0';
}
str[len]='';
printf("%s",str);
return 0;
}
```

Q33. What is the difference between Macros and Functions?

Ans.

Macros	Functions
Macro is Pre-processed	Function is compiled
No type checking is done in Micro	Type checking is done in function
Use of Micro can lead to side effect at	Functions do not lead to any side effect

later stages	in any case
Speed of execution using Micro is faster	Speed of execution using function is slower

Q34. How will you find the total number of elements in the array if the size of the array is not provided?

Ans. We will use "sizeof" operator to calculate number of elements in the array.

```
int main()
    {
int arr[]={1,2,3,4,5,6,7,8,9,10};
printf("Number of elements: %d",sizeof(arr)/sizeof(arr[0]); // sizeof(arr) = 40 , sizeof(arr[0])=4
return 0;
    }

O/p = 10
```

Q35. Do all character arrays end with "nul"() character in C?

Ans. All character arrays are not null terminated in C. Strings are special character arrays which are null terminated.

1. char arr[3] = {'a','b','c'}; is not a null terminated character array.
2. char arr[] = "PrepBytes"; is a null terminated string. These are known as String constants.

Q36. Describe how arrays can be passed to a user-defined function.

Ans. We cannot pass an entire array in the form of arguments. Instead, while calling a function we pass the address of the first element of the array(name of the array) and in formal arguments, a pointer is used to point to this address.

```
int main()
    {
int arr[5]={1,2,3,4,5};
    .....
    func(arr);
    .....
    }
void func(int arr[]) or void func(int *arr)
```

Q37. What are self-referential structures? Are they possible in C?

Ans. Self-referential structures are those structures that have one or more pointers which point to the same type of structure, as their member.

```
struct node{
    int data1;
struct node* link;
    };
```

Yes, self-referential structures are possible in C.

Q38. What are the different ways to access structure members in C?

Ans. They are two operators using which we can access structure members

1. (.) dot operator
2. (->) arrow operator

Syntax: (structure name) .(member name) , (structure name) ->(member name)
Difference between them is in the case of pointers.

```
struct node
    {
    int i;
    int j;
    }
struct node a,*p;
    p=&a;
```

Here, 'a' is structure of type node, and 'p' is a pointer to a structure of type node.
Now accessing elements with the help of 'a' is straight forward- a.i and a.j
But using 'p' is tricky. (*p.i) if we right like this, then precedence and associativity will arrive in the picture. Compiler will read (*p.i) as (*(p.i)) which is wrong, so you need to manage parenthesis as well, i.e. right correctly ((*p).i), so to reduce programming errors (->) operator is provided in case of pointers. Using arrow operators we can simply write

(p->i) and (p->j) to access members of structure node using pointers.

Q39. What is the main advantage of using Structures in C?

Ans. Structures can store heterogeneous type of data under a single name. Unlike arrays, we can store int, float, char any type of data inside structures and then access and manipulate this data.

Q40. What are the advantages and disadvantages of storing data in the heap area?

Ans. Storing data on a heap is comparatively slower than storing data on slack. The main advantage is the flexibility provided by the heap area. Memory in the heap area can be allocated and deallocated in any particular order.

Q41. What is a Storage class in C?

Ans. Storage class in C decides the part of storage to allocate memory for a variable, it also determines the scope of a variable. All variables defined in a C program get some physical location in memory where the variable's value is stored. Along with the lifetime of a variable, storage class also determines variable's storage location (memory or registers), the scope (visibility level) of the variable, and the initial value of the variable. There are four storage classes in C those are automatic, register, static and external.

Q42. Describe Storage class Specifiers.

Ans. There are four storage class specifiers in C

1. auto
2. register
3. extern
4. static

These specifiers tell the compiler how to store the subsequent variable. The general form of a variable declaration that uses a storage class is:

Storage_class_specifier data_type variable_name;

At most one storage class specifier may be given in a declaration.

(Detailed explanation in the next question)

Q43. Describe types of Storage classes.

Ans.

1. Automatic storage class:
   1. A variable defined within a function or a block with (auto) specifier belongs to the automatic storage class.
   2. All variables defined within a function or block by default belong to automatic storage class if no storage class is mentioned.
2. Register storage class:
   1. The (register) specifier declares a variable of the register storage class.
   2. Variables belonging to the register storage class are local to the block which they are defined in, and get destroyed on exit from the block,
   3. A (register) declaration is equivalent to an (auto) declaration, but hints that the declared variable will be accessed frequently; therefore, they are placed in CPU registers, not in memory.
3. Static storage class:
   1. The (static) specifier gives the declared variable static storage class.
   2. Static variables can be used within function or file. Unlike global variables, static variables are not visible outside their function or file, but they maintain their values between calls.
4. Extern Storage class:
   1. The (extern) specifier gives the declared variable external storage class.
   2. A declaration declares the name and type of a variable or function. A definition causes storage to be allocated for the variable or the body of the function to be defined.
   3. When externspecifier is used with a variable declaration then no storage is allocated to that variable and it is assumed that the variable has already been defined elsewhere in the program. When we use extern specifier the variable cannot be initialized because with extern specifier variable is declared, not defined.

Q44. If storage classes are responsible for scope and memory storage of variable, then why don't we normally write them with each variable in the code?

Ans. Default rules that are used if no storage class specifier is specified are:

1. Variables declared inside the function are taken to be (auto).
2. Functions declared within a function are taken to be (extern).

3. Variables and functions declared outside a function are taken to be (static), with external linkage.

Q45. Suppose a global variable and local variable have the same name. Is it possible to access a global variable from a block where a local variable is defined in C?

Ans. No, it is not possible in C. It is always the most local variable that gets preference.

Q46. What is a sequential access file?

Ans.

1. When writing programs that will store and retrieve data in a file, it is possible to designate that file into different forms.
2. A sequential access file is such that data are saved in sequential order: one data is placed into the file after another.
3. To access a particular data within the sequential access file, data has to be read one data at a time, until the right one is reached.

Q47. What is translation unit?

Ans.

1. A translation unit is a set of files seen by the compiler.
2. It includes the source code under consideration and files that are included such as header files and other disk files contain C code.

Q48. Find the maximum & minimum of two numbers in a single line without using any condition & loop.

Ans.

```
#include <stdio.h>
#include<stdlib.h>
    int main()
        {
    int a=15,b=30;
printf ("max = %d, min = %d", ((a+b) + abs(a-b))/2, ((a+b) - abs(a-b)) /2);
      return 0;
        }

O/p: max=30, min=15
```

Q49. What is FILE pointer in C?

Ans.

1. File pointer is a pointer which is used to handle and keep track on the files being accessed. A new data type called “FILE” is used to declare file pointer. This data type is defined in stdio.h file. File pointer is declared as FILE *fp. Where ‘fp’ is a file pointer.
2. fopen() function is used to open a file that returns a FILE pointer. Once file is opened, file pointer can be used to perform I/O operations on the file. fclose() function is used to close the file.

Q50. What is the difference between an uninitialized pointer and a null pointer?

Ans. An uninitialized pointer is a pointer which points unknown memory location while the null pointer is pointer which points a null value or base address of the segment.

CONTACT US


+91-8800 2588 17


contact@prepbytes.com