What are different storage class specifiers in C?
Ans: auto, register, static, extern

What is scope of a variable? How are variables scoped in C?
Ans: Scope of a variable is the part of the program where the variable may directly be accessible. In C, all identifiers are lexically (or statically) scoped.

How will you print "Hello World" without semicolon?
Ans:

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

When should we use pointers in a C program?
1. To get address of a variable
2. *For achieving pass by reference in C:* Pointers allow different functions to share and modify their local variables.
3. *To pass large structures* so that complete copy of the structure can be avoided.
C
4. *To implement "linked" data structures* like linked lists and binary trees.

What is NULL pointer?
Ans: NULL is used to indicate that the pointer doesn't point to a valid location. Ideally, we should initialize pointers as NULL if we don't know their value at the time of declaration. Also, we should make a pointer NULL when memory pointed by it is deallocated in the middle of a program.

What is Dangling pointer?
Ans: Dangling Pointer is a pointer that doesn't point to a valid memory location. Dangling pointers arise when an object is deleted or deallocated, without modifying the value of the pointer, so that the pointer still points to the memory location of the deallocated memory. Following are examples.

```
// EXAMPLE 1
int *ptr = (int *)malloc(sizeof(int));
.............
.............
free(ptr);

// ptr is a dangling pointer now and operations like following are invalid
*ptr = 10;  // or printf("%d", *ptr);
// EXAMPLE 2
int *ptr = NULL
{
   int x  = 10;
   ptr = &x;
}
// x goes out of scope and memory allocated to x is free now.
// So ptr is a dangling pointer now.
```

What is memory leak? Why it should be avoided

Ans: Memory leak occurs when programmers create a memory in heap and forget to delete it. Memory leaks are particularly serious issues for programs like daemons and servers which by definition never terminate.

```
/* Function with memory leak */
#include <stdlib.h>

void f()
{
   int *ptr = (int *) malloc(sizeof(int));

   /* Do some work */

   return; /* Return without freeing ptr*/
```

What are local static variables? What is their use?

Ans:A local static variable is a variable whose lifetime doesn't end with a function call where it is declared. It extends for the lifetime of complete program. All calls to the function share the same copy of local static variables. Static variables can be used to count the number of times a function is called. Also, static variables get the default value as 0. For example, the following program prints "0 1"

```
#include <stdio.h>
void fun()
{
    // static variables get the default value as 0.
    static int x;
    printf("%d ", x);
    x = x + 1;
}

int main()
{
    fun();
    fun();
    return 0;
}
// Output: 0 1
```

What are static functions? What is their use?

Ans: In C, functions are global by default. The "static" keyword before a function name makes it static. Unlike global functions in C, access to static functions is restricted to the file where they are declared.

Therefore, when we want to restrict access to functions, we make them static. Another reason for making functions static can be reuse of the same function name in other files.

What are main characteristics of C language?
C is a procedural language. The main features of C language include low-level access to memory, simple set of keywords, and clean style. These features make it suitable for system programming like operating system or compiler development.

What is difference between i++ and ++i?
1) The expression 'i++' returns the old value and then increments i. The expression ++i increments the value and returns new value.
2) Precedence of postfix ++ is higher than that of prefix ++.
3) Associativity of postfix ++ is left to right and associativity of prefix ++ is right to left.
4) In C++, ++i can be used as l-value, but i++ cannot be. In C, they both cannot be used as l-value.

Difference between ++*p, *p++ and *++p

Predict the output of following C programs.

```
// PROGRAM 1
#include <stdio.h>
int main(void)
{
    int arr[] = {10, 20};
    int *p = arr;
    ++*p;
    printf("arr[0] = %d, arr[1] = %d, *p = %d", arr[0], arr[1], *p);
    return 0;
}
// PROGRAM 2
#include <stdio.h>
int main(void)
{
    int arr[] = {10, 20};
    int *p = arr;
    *p++;
    printf("arr[0] = %d, arr[1] = %d, *p = %d", arr[0], arr[1], *p);
    return 0;
}
// PROGRAM 3
#include <stdio.h>
int main(void)
{
    int arr[] = {10, 20};
    int *p = arr;
    *++p;
    printf("arr[0] = %d, arr[1] = %d, *p = %d", arr[0], arr[1], *p);
    return 0;
}
```

The output of above programs and all such programs can be easily guessed by remembering following simple rules about postfix ++, prefix ++ and * (dereference) operators
1) Precedence of prefix ++ and * is same. Associativity of both is right to left.
2) Precedence of postfix ++ is higher than both * and prefix ++. Associativity of postfix ++ is left to right.

The expression **++*p** has two operators of same precedence, so compiler looks for assoiativity. Associativity of operators is right to left. Therefore the expression is treated as ***++(*p)***. Therefore the output of first program is "*arr[0] = 11, arr[1] = 20, *p = 11*".

The expression ***p++** is treated as ***(p++)*** as the precedence of postfix ++ is higher than *. Therefore the output of second program is "*arr[0] = 10, arr[1] = 20, *p = 20*".

The expression ***++p** has two operators of same precedence, so compiler looks for assoiativity. Associativity of operators is right to left. Therefore the expression is treated as ***(++p)***. Therefore the output of second program is "*arr[0] = 10, arr[1] = 20, *p = 20*".

What is l-value?

l-value or location value refers to an expression that can be used on left side of assignment operator. For example in expression "a = 3", a is l-value and 3 is r-value.
l-values are of two types:
"nonmodifiable l-value" represent a l-value that can not be modified. const variables are "nonmodifiable l-value".
"modifiable l-value" represent a l-value that can be modified.

How to write your own sizeof operator?

```
#define my_sizeof(type) (char *)(&type+1)-(char*)(&type)
```

How will you print numbers from 1 to 100 without using loop?

```
#include<stdio.h>

void printNos(unsigned int n)

{

if(n > 0)

{

        printNos(n-1);

        printf("%d ", n);
}

}

int main()

{

   int n=100;

   printNos(100);

   return 0;

}
```

What is volatile keyword?

The volatile keyword is intended to prevent the compiler from applying any optimizations on objects that can change in ways that cannot be determined by the compiler.

Objects declared as volatile are omitted from optimization because their values can be changed by code outside the scope of current code at any time.

Can a variable be both const and volatile?

yes, the const means that the variable cannot be assigned a new value. The value can be changed by other code or pointer. For example the following program works fine.

```
int main(void)
{
    const volatile int local = 10;
    int *ptr = (int*) &local;
    printf("Initial value of local : %d \n", local);
    *ptr = 100;
    printf("Modified value of local: %d \n", local);
    return 0;
}
```

Practice the following quiz:

http://www.geeksforgeeks.org/quiz-corner-gq/#C%20Programming%20Mock%20Tests