Output

1. Output of the program | Dereference, Reference, Dereference, Reference….

Predict the output of below program

```
int main()
{
 char *ptr = "geeksforgeeks";
 printf("%c\n", *&*&*ptr);

 getchar();
 return 0;
}
```

Output: g

Explanation: The operator * is used for dereferencing and the operator & is used to get the address. These operators cancel effect of each other when used one after another. We can apply them alternatively any no. of times. For example *ptr gives us g, &*ptr gives address of g, *&*ptr again g, &*&*ptr address of g, and finally *&*&*ptr gives ‘g’

Now try below

```
int main()
{
 char *ptr = "geeksforgeeks";
 printf("%s\n", *&*&ptr);

 getchar();
 return 0;
}
```

2. Output of the Program | Use Macros Carefully!

Predict the output of the below program

```c
#define square(x) x*x
int main()
{
  int x;
  x = 36/square(6);
  printf("%d",x);

  getchar();
  return 0;
}
```

Output: 36

Explanation:
Preprocessor replaces square(6) by 6*6 and the expression becomes x = 36/6*6 and value of x is calculated as 36. If we want correct behavior from macro square(x), we should declare it as

#define square(x) ((x)*(x)) /* Note that the expression
(x*x) will also fail for square(6-2) */

3. Output of the Program | Pointer to a Constant or Constant Pointer?

Predict the output of the below program.

```c
int main()
{
    int x = 5;
    int * const ptr = &x;
    ++(*ptr);
    printf("%d", x);

    getchar();
    return 0;
}
```

Output: 6

Explananation:
See following declarations to know the difference between constant pointer and a pointer to a constant.

int * const ptr —> ptr is constant pointer. You can change the value at the location pointed by pointer p, but you can not change p to point to other location.

int const * ptr —> ptr is a pointer to a constant. You can change ptr to point other variable. But you cannot change the value pointed by ptr.

Therefore above program works well because we have a constant pointer and we are not changing ptr to point to any other location. We are only icrementing value pointed by ptr.

Try below program, you will get compiler error.

```
int main()
{
    int x = 5;
    int const * ptr = &x;
    ++(*ptr);
    printf("%d", x);

    getchar();
    return 0;
}
```

4. Output of C Programs | Set 1

Predict the output of below programs.

Question 1

```
#include<stdio.h>
int main()
{
   int n;
   for(n = 7; n!=0; n--)
     printf("n = %d", n--);
   getchar();
   return 0;
}
```

Output:Above program goes in infinite loop because n is never zero when loop condition (n != 0) is checked.

Question 2

```
#include<stdio.h>
int main()
{
   printf("%x", -1<<1);
   getchar();
   return 0;
}
```

Output is dependent on the compiler. For 32 bit compiler it would be fffffffe and for 16 bit it would be fffe.

Question 3

```
# include <stdio.h>
# define scanf  "%s Geeks For Geeks "
main()
{
   printf(scanf, scanf);
   getchar();
   return 0;
}
```

Output: %s Geeks For Geeks Geeks For Geeks
Explanation: After pre-processing phase of compilation, printf statement will become.

```
   printf("%s Geeks For Geeks ",  "%s Geeks For Geeks ");
```

Now you can easily guess why output is %s Geeks For Geeks Geeks For Geeks.

Question 4

```
#include <stdlib.h>
#include <stdio.h>
enum {false, true};
int main()
{
   int i = 1;
   do
   {
      printf("%d\n", i);
      i++;
      if (i < 15)
        continue;
   } while (false);

   getchar();
   return 0;
}
```

Output: 1
Explanation: The do wile loop checks condition after each iteration. So after continue statement, control transfers to the statement while(false). Since the condition is false 'i' is printed only once.

Now try below program.

```
#include <stdlib.h>
#include <stdio.h>
enum {false, true};
int main()
{
   int i = 1;
   do
   {
     printf("%d\n", i);
     i++;
     if (i < 15)
       break;
     } while (true);

     getchar();
     return 0;
}
```

Question 5

```
char *getString()
{
   char *str = "Nice test for strings";
   return str;
}

int main()
{
   printf("%s", getString());
   getchar();
   return 0;
}
```

Output: “Nice test for strings”
The above program works because string constants are stored in Data Section (not in Stack Section). So, when getString returns *str is not lost.

5. Output of C Programs | Set 2

Predict the output of below programs.

Question 1

```
char *getString()
{
    char str[] = "Will I be printed?";
    return str;
}
int main()
{
    printf("%s", getString());
    getchar();
}
```

Output: Some garbage value

The above program doesn’t work because array variables are stored in Stack Section. So, when getString returns values at str are deleted and str becomes dangling pointer.

Question 2

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

Output: 0 0 0 0

Explanation: Since i is a static variable and is stored in Data Section, all calls to main share same i.

Question 3

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

Output: 5 4 3 2 1

Explanation: Same as previous question. The only difference here is, sequence of calling main and printf is changed, therefore different output.

Question 4

```
int main()
{
    int x;
    printf("%d",scanf("%d",&x));
    /* Suppose that input value given
        for above scanf is 20 */
    return 1;
}
```

Output: 1

scanf returns the no. of inputs it has successfully read.

Question 5

```
# include <stdio.h>
int main()
{
   int i=0;
   for(i=0; i<20; i++)
   {
     switch(i)
     {
       case 0:
         i+=5;
       case 1:
         i+=2;
       case 5:
         i+=5;
       default:
         i+=4;
         break;
     }
     printf("%d  ", i);
   }

   getchar();
   return 0;
}
```

Output: 16 21
Explanation:
Initially i = 0. Since case 0 is true i becomes 5, and since there is no break statement till last statement of switch block, i becomes 16. Now in next iteration no case is true, so execution goes to default and i becomes 21.

In C, if one case is true switch block is executed until it finds break statement. If no break statement is present all cases are executed after the true case. If you want to know why switch is implemented like this, well this implementation is useful for situations like below.

```
switch (c)
{
   case 'a':
   case 'e':
   case 'i' :
   case 'o':
   case 'u':
     printf(" Vowel character");
     break;
   default :
     printf("Not a Vowel character");; break;
 }
```

6. Output of C Programs | Set 3

Predict the output of the below program.

Question 1

```
#include <stdio.h>
int main()
{
  printf("%d", main);
  getchar();
  return 0;
}
```

Output: Address of function main.
Explanation: Name of the function is actually a pointer variable to the function and prints the address of the function. Symbol table is implemented like this.

```
struct
{
   char *name;
   int (*funcptr)();
}
symtab[] = {
   "func", func,
   "anotherfunc", anotherfunc,
};
```

Question 2

```
#include <stdio.h>
int main()
{
   printf("\new_c_question\by");
   printf("\rgeeksforgeeks");

   getchar();
   return 0;
}
```

Output: geeksforgeeksl
Explanation: First printf prints “ew_c_questioy”. Second printf has \r in it so it goes back to start of the line and starts printing characters.

Now try to print following without using any of the escape characters.

```
new c questions by
geeksforgeeks
```

Question 3

```c
# include<stdio.h>
# include<stdlib.h>

void fun(int *a)
{
    a = (int*)malloc(sizeof(int));
}

int main()
{
    int *p;
    fun(p);
    *p = 6;
    printf("%d\n",*p);

    getchar();
    return(0);
}
```

It does not work. Try replacing “int *p;” with “int *p = NULL;” and it will try to dereference a null pointer.

This is because fun() makes a copy of the pointer, so when malloc() is called, it is setting the copied pointer to the memory location, not p. p is pointing to random memory before and after the call to fun(), and when you dereference it, it will crash.

If you want to add memory to a pointer from a function, you need to pass the address of the pointer (ie. double pointer).

Thanks to John Doe for providing the correct solution.

Question 4

```c
#include <stdio.h>
int main()
{
    int i;
    i = 1, 2, 3;
    printf("i = %d\n", i);

    getchar();
    return 0;
}
```

Output: 1
The above program prints 1. Associativity of comma operator is from left to right, but = operator has higher precedence than comma operator.
Therefore the statement i = 1, 2, 3 is treated as (i = 1), 2, 3 by the compiler.

Now it should be easy to tell output of below program.

```
#include <stdio.h>
int main()
{
    int i;
    i = (1, 2, 3);
    printf("i  = %d\n", i);

    getchar();
     return 0;
}
```

Question 5

```
#include <stdio.h>
int main()
{
    int first = 50, second = 60, third;
    third = first /* Will this comment work? */ + second;
    printf("%d /* And this? */ \n", third);

    getchar();
    return 0;
}
```

Output: 110 /* And this? */

Explanation: Compiler removes everything between "/*" and "*/" if they are not present inside double quotes ("").

7. Output of C Programs | Set 4

Predict the output of below programs

Question 1

```
#include<stdio.h>
int main()
{
    struct site
    {
        char name[] = "GeeksforGeeks";
        int no_of_pages = 200;
    };
    struct site *ptr;
    printf("%d",ptr->no_of_pages);
    printf("%s",ptr->name);
    getchar();
    return 0;
}
```

Output:

Compiler error

Explanation:
Note the difference between structure/union declaration and variable declaration. When you declare a structure, you actually declare a new data type suitable for your purpose. So you cannot initialize values as it is not a variable declaration but a data type declaration.

Reference:
http://www.lix.polytechnique.fr/~liberti/public/computing/prog/c/C/SYNTAX/struct.html

Question 2

```
int main()
{
    char a[2][3][3] = {'g','e','e','k','s','f','o',
                           'r','g','e','e','k','s'};
    printf("%s ", **a);
    getchar();
    return 0;
}
```

Output:
geeksforgeeks

Explanation:
We have created a 3D array that should have 2*3*3 (= 18) elements, but we are initializing only 13 of them. In C when we initialize less no of elements in an array all uninitialized elements become '\0' in case of char and 0 in case of integers.

Question 3

```
int main()
{
   char str[]= "geeks\nforgeeks";
   char *ptr1, *ptr2;

   ptr1 = &str[3];
   ptr2 = str + 5;
   printf("%c", ++*str - --*ptr1 + *ptr2 + 2);
   printf("%s", str);

   getchar();
   return 0;
}
```

Output:
heejs
forgeeks

Explanation:
Initially ptr1 points to 'k' and ptr2 points to '\n' in "geeks\nforgeeks". In print statement value at str is incremented by 1 and value at ptr1 is decremented by 1. So string becomes "heejs\nforgeeks" .

First print statement becomes
printf(“%c”, ‘h’ – ‘j’ + ‘n’ + 2)

‘h’ – ‘j’ + ‘\n’ + 2 = -2 + ‘\n’ + 2 = ‘\n’

First print statements newline character. and next print statement prints “heejs\nforgeeks”.

Question 4

```
#include <stdio.h>
int fun(int n)
{
    int i, j, sum = 0;
    for(i = 1;i<=n;i++)
        for(j=i;j<=i;j++)
            sum=sum+j;
    return(sum);
}

int main()
{
    printf("%d", fun(15));
    getchar();
    return 0;
}
```

Output: 120
Explanation: fun(n) calculates sum of first n integers or we can say it returns n(n+1)/2.

Question 5

```
#include <stdio.h>
int main()
{
    int c = 5, no = 1000;
    do {
        no /= c;
    } while(c--);

    printf ("%d\n", no);
    return 0;
}
```

Output: Exception – Divide by zero

Explanation: There is a bug in the above program. It goes inside the do-while loop for c = 0 also. Be careful when you are using do-while loop like this!!

8. Output of C Programs | Set 5

Predict the output of below programs

Question 1

```
int main()
{
    while(1){
        if(printf("%d",printf("%d")))
            break;
        else
            continue;
    }
    return 0;
}
```

Output:
Can't be predicted

Explanation:
The condition in while loop is 1 so at first shot it looks infinite loop. Then there are break and continue in the body of the while loop, so it may not be infinite loop. The break statement will be executed if the condition under if is met, otherwise continue will be executed. Since there's no other statements after continue in the while loop, continue doesn't serve any purpose. In fact it is extraneous. So let us see the if condition. If we look carefully, we will notice that the condition of the if will be met always, so break will be executed at the first iteration itself and we will come out of while loop. The reason why the condition of if will be met is printf function. Function printf always returns the no. of bytes it has output. For example, the return value of printf("geeks") will be 5 because printf will output 5 characters here. In our case, the inner printf will be executed first but this printf doesn't have argument for format specifier i.e. %d. It means this printf will print any arbitrary value. But notice that even for any arbirary value, the no. of bytes output by inner printf would be non-zero. And those no. of bytes will work as argument to outer printf. The outer printf will print that many no. of bytes and return non-zero value always. So the condition for if is also true always. Therefore, the while loop be executed only once. As a side note, even without outer printf also, the condition for if is always true.

Question 2

```
int main()
{
    unsigned int i=10;
    while(i-- >= 0)
        printf("%u ",i);
    return 0;
}
```

Output:
9 8 7 6 5 4 3 2 1 0 4294967295 4294967294 …… (on a machine where int is 4 bytes long)

9 8 7 6 5 4 3 2 1 0 65535 65534 …. (on a machine where int is 2 bytes long)

Explanation:
Let us examine the condition of while loop. It is obvious that as far as the condition of while loop is met, printf will be executed. There are two operators in the condition of while loop: post-decrement operator and comparison operator. From operator precedence, we know that unary operator post-decrement has higher priority than comparison operator. But due to post-decrement property, the value of i will be decremented only after it had been used for comparison. So at the first iteration, the condition is true because 10>=0 and then i is decremented. Therefore 9 will be printed. Similarly the loop continues and the value of i keeps on decrementing. Let us see what what happen when condition of while loop becomes 0 >= 0. At this time, condition is met and i is decremented. Since i is unsigned integer, the roll-over happens and i takes the value of the highest +ve value an unsigned int can take. So i is never negative. Therefore, it becomes infinite while loop.

As a side note, if i was signed int, the while loop would have been terminated after printing the highest negative value.

Question 3

```
int main()
{
    int x,y=2,z,a;
    if ( x = y%2)
         z =2;
    a=2;
    printf("%d %d ",z,x);
    return 0;
}
```

Output:
< some garbage value of z > 0

Explanation:
This question has some stuff for operator precedence. If the condition of if is met, then z will be initialized to 2 otherwise z will contain garbage value. But the condition of if has two operators: assignment operator and modulus operator. The precedence of modulus is higher than assignment. So y%2 is zero and it'll be assigned to x. So the value of x becomes zero which is also the effective condition for if. And therefore, condition of if is false.

Question 4

```
int main()
{
    int a[10];
    printf("%d",*a+1-*a+3);
    return 0;
}
```

Output: 4

Explanation:
From operator precedence, de-reference operator has higher priority than addition/subtraction operator. So de-reference will be applied first. Here, a is an array which is not initialized. If we use a, then it will point to the first element of the array. Therefore *a will be the first element of the array. Suppose first element of array is x, then the argument inside printf becomes as follows. It’s effective value is 4.

x + 1 – x + 3 = 4

Question 5

```
#define prod(a,b) a*b
int main()
{
    int x=3,y=4;
    printf("%d",prod(x+2,y-1));
    return 0;
}
```

Output:
10

Explanation:
This program deals with macros, their side effects and operator precedence. Here prod is a macro which multiplies its two arguments a and b. Let us take a closer look.

prod(a, b) = a*b
prod(x+2, y-1) = x+2*y-1 = 3+2*4-1 = 3+8-1=10

If the programmer really wanted to multiply x+2 and y-1, he should have put parenthesis around a and b as follows.

prod(a,b) = (a)*(b)

This type of mistake in macro definition is called – macro side-effects.

9. Output of C Programs | Set 6

Predict the output of below programs

Question 1

```
int main()
{
    unsigned int i=65000;
    while ( i++ != 0 );
    printf("%d",i);
    return 0;
}
```

Output:
1

Explanation:
It should be noticed that there's a semi-colon in the body of while loop. So even though, nothing is done as part of while body, the control will come out of while only if while condition isn't met. In other words, as soon as i inside the condition becomes 0, the condition will become false and while loop would be over. But also notice the post-increment operator in the condition of while. So first i will be compared with 0 and i will be incremented no matter whether condition is met or not. Since i is initialized to 65000, it will keep on incrementing till it reaches highest positive value. After that roll over happens, and the value of i becomes zero. The condition is not met, but i would be incremented i.e. to 1. Then printf will print 1.

Question 2

```
int main()
{
    int i=0;
    while ( +(+i--) != 0)
        i-=i++;
    printf("%d",i);
    return 0;
}
```

Output:
-1

Explanation:
Let us first take the condition of while loop. There are several operator there. Unary + operator doesn't do anything. So the simplified condition becomes (i–) != 0. So i will be compared with 0 and then decremented no matter whether condition is true or false. Since i is initialized to 0, the condition of while will be false at the first iteration itself but i will be decremented to -1. The body of while loop will not be executed. And printf will print -1.

So it wasn't that scary as it seemed to be!

Question 3

```
int main()
{
    float f=5,g=10;
    enum{i=10,j=20,k=50};
    printf("%d\n",++k);
    printf("%f\n",f<<2);
    printf("%lf\n",f%g);
    printf("%lf\n",fmod(f,g));
    return 0;
}
```

Output:

Program will not compile and give 3 errors

Explanation:

Here, i, j and k are inside the enum and therefore, they are like constants. In other words, if want to us 10 anywhere in the program , we can use i instead. In the first printf, the value of i is being modified which is not allowed because it's enum constant. In the second printf, left-shift operator is being applied on a float which is also not allowed. Similarly, in the third printf, modulus operator is being applied on float f and g which is also not allowed.

Question 4

```
int main()
{
    int i=10;
    void pascal f(int,int,int);
    f(i++, i++, i++);
    printf(" %d",i);
    return 0;
}
void pascal f(integer :i,integer:j,integer :k)
{
  write(i,j,k);
}
```

Output:

Program will give compile-time error

Explanation:

Compiler specific question. Not all compilers support this.

Otherwise, pascal enforces left to right processing of arguments. So even though, the argument processing order can be changed by the use of pascal, we can't use Pascal language routines such as write inside C program.

Question 5

```
void pascal f(int i,int j,int k)
{
  printf("%d %d %d",i, j, k);
}

void cdecl f(int i,int j,int k)
{
  printf("%d %d %d",i, j, k);
}

main()
{
    int i=10;
    f(i++,i++,i++);
    printf(" %d\n",i);
    i=10;
    f(i++,i++,i++);
    printf(" %d",i);
}
```

Output:
Compiler specific question. Not all the compilers allow this.

Explanation:
This question deals with the argument passing mechanism. If we call a function, the order in which the arguments of the function are processed is not governed by C Standard. So one compiler can process the arguments from left to right while the other compiler can process them right to left. Usually, the programs are not affected with this because the arguments of the programs are different. For example if we call funtion fun as fun(i,j), then no matter in which order the arguments are processed, the value of i and j will be consistent.

But in this case, we are passing the arguments to function f using the same variable. So the order in which arguments are processed by the function will determine the value of those arguments. cdecl enforces right to left processing of arguments while pascal enforces left to right processing of arguments.

So the value of i, j and k inside the first function f will be 10, 11 and 12 respectively while the value of i, j and k inside the second function f will be 12, 11 and 10 respectively.

10. Output of C Programs | Set 7

Predict the output of below programs

Question 1

```
int main()
{
    int i = 0;
    while (i <= 4)
    {
       printf("%d", i);
       if (i > 3)
        goto inside_foo;
       i++;
    }
    getchar();
    return 0;
}

void foo()
{
   inside_foo:
     printf("PP");
}
```

Output: Compiler error: Label “inside_foo” used but not defined.

Explanation: Scope of a label is within a function. We cannot goto a label from other function.

Question 2

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

  getchar();
  return 0;
}
```

Output: 50

Preprocessor doesn’t give any error if we redefine a preprocessor directive. It may give warning though. Preprocessor takes the most recent value before use of and put it in place of a.

Now try following

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

  getchar();
  return 0;
}
```

Question 3

```
int main()
{
     char str[] = "geeksforgeeks";
     char *s1 = str, *s2 = str;
     int i;

     for(i = 0; i < 7; i++)
     {
        printf(" %c ", *str);
        ++s1;
     }

     for(i = 0; i < 6; i++)
     {
        printf(" %c ", *s2);
        ++s2;
      }

      getchar();
      return 0;
}
```

Output

g g g g g g g g e e k s f

Explanation
Both s1 and s2 are initialized to str. In first loop str is being printed and s1 is being incremented, so first loop will print only g. In second loop s2 is incremented and s2 is printed so second loop will print "g e e k s f "

Question 4

```
int main()
{
    char str[] = "geeksforgeeks";
    int i;
    for(i=0; str[i]; i++)
        printf("\n%c%c%c%c", str[i], *(str+i), *(i+str), i[str]);

    getchar();
    return 0;
}
```

Output:

```
gggg
eeee
eeee
kkkk
ssss
ffff
oooo
rrrr
```

```
gggg
eeee
eeee
kkkk
ssss
```

Explanaition:
Following are different ways of indexing both array and string.

arr[i]
*(arr + i)
*(i + arr)
i[arr]

So all of them print same character.

Question 5

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

    getchar();
    return 0;
}
```

Output: Compiler dependent. I got output as “1 4″

Explanation:
Output of the above program depends on compiler. sizeof(*p) gives size of character. If characters are stored as 1 byte then sizeof(*p) gives 1.
sizeof(p) gives the size of pointer variable. If pointer variables are stored as 4 bytes then it gives 4.

11. Output of C programs | Set 8

Predict the output of below C programs.
Question 1:

```
#include<stdio.h>
int main()
{
    int x = 5, p = 10;
    printf("%*d", x, p);

    getchar();
    return 0;
}
```

Output:

```
   10
```

Explanation:
Please see standard printf function definition

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

format: String that contains the text to be written to stdout. It can optionally contain embedded format tags that are substituted by the values specified in subsequent argument(s) and formatted as requested. The number of arguments following the format parameters should at least be as much as the number of format tags. The format tags follow this prototype:

```
      %[flags][width][.precision][length]specifier
```

You can see details of all above parts here
http://www.cplusplus.com/reference/clibrary/cstdio/printf/.

The main thing to note is below the line about precision
*** (star)**: The precision is not specified in the format string, but as an additional integer value argument preceding the argument that has to be formatted.

So, in the above example 5 is precision and 10 is the actual value to be printed. (Thanks to Ajay Mishra for providing the solution)

Question 2

```
int main()
{
  char arr[]  = "geeksforgeeks";
  char *ptr  = arr;

  while(*ptr != '\0')
      ++*ptr++;
  printf("%s %s", arr, ptr);

  getchar();
  return 0;
}
```

Output: hffltgpshfflt

Explanation:
The crust of this question lies in expression ++*ptr++.

If one knows the precedence and associativity of the operators then there is nothing much left. Below is the precedence of operators.

```
Postfixx ++              left-to-right
Prefix  ++               right-to-left
Dereference *            right-to-left
```

Therefore the expression ++*ptr++ has following effect
Value of *ptr is incremented
Value of ptr is incremented

Question 3

```
int main()
{
  signed char i=0;
  for(; i >= 0; i++);
  printf("%d\n", i);

  getchar();
  return 0;
}
```

Output: -128

Explanation:
Here, the first thing to be noticed is the semi-colon in the end of for loop. So basically, there’s no body for the for loop. And printf will print the value of i when control comes out of the for loop. Also, notice that the first statement i.e. initializer in the for loop is not present. But i has been initialized at the declaration time. Since i is a signed character, it can take value from -128 to 127. So in the for loop, i will keep incrementing and the condition is met till roll over happens. At the roll over, i will become -128 after 127, in that case condition is not met and control comes out of the for loop.

Question 4

```
#include <stdio.h>
void fun(const char **p) { }
int main(int argc, char **argv)
{
   fun(argv);
   getchar();
   return 0;
}
```

Output: Compiler error.

Explanation:
Parameter passed to fun() and parameter expected in definition are of incompatible types. fun() expects const char** while passed parameter is of type char **.

Now try following program, it will work.

```
void fun(const char **p) { }
int main(int argc, char **argv)
{
   const char **temp;
   fun(temp);
   getchar();
   return 0;
}
```

12. Output of C Programs | Set 9

Predict the output of the below programs.

Question 1

```
int main()
{
 int c=5;
 printf("%d\n%d\n%d", c, c <<= 2, c >>= 2);
 getchar();
}
```

Output: Compiler dependent
Evaluation order of parameters is not defined by C standard and is dependent on compiler implementation. It is never safe to depend on the order of parameter evaluation. For example, a function call like above may very well behave differently from one compiler to another.

References:
http://gcc.gnu.org/onlinedocs/gcc/Non_002dbugs.html

Question 2

```
int main()
{
    char arr[] = {1, 2, 3};
    char *p = arr;
    if(&p == &arr)
     printf("Same");
    else
     printf("Not same");
    getchar();
}
```

Output: Not Same

&arr is an alias for &arr[0] and returns the address of the first element in array, but &p returns the address of pointer p.

Now try below program

```
int main()
{
    char arr[] = {1, 2, 3};
    char *p = arr;
    if(p == &arr)
     printf("Same");
    else
     printf("Not same");
    getchar();
}
```

Question 3

```
int main()
{
    char arr[] = {1, 2, 3};
    char *p = arr;
    printf(" %d ", sizeof(p));
    printf(" %d ", sizeof(arr));
    getchar();
}
```

Output 4 3

sizeof(arr) returns the amount of memory used by all elements in array
and sizeof(p) returns the amount of memory used by the pointer variable itself.

Question 4

```
int x = 0;
int f()
{
   return x;
}

int g()
{
   int x = 1;
   return f();
}

int main()
{
  printf("%d", g());
  printf("\n");
  getchar();
}
```

Output: 0
In C, variables are always statically (or lexically) scoped. Binding of x inside f() to global variable x is defined at compile time and not dependent on who is calling it. Hence, output for the above program will be 0.

On a side note, Perl supports both dynamic ans static scoping. Perl's keyword "my" defines a statically scoped local variable, while the keyword "local" defines dynamically scoped local variable. So in Perl, similar (see below) program will print 1.

```
$x = 0;
sub f
{
   return $x;
}
sub g
{
   local $x = 1; return f();
}
print g()."\n";
```

Reference:
http://en.wikipedia.org/wiki/Scope_%28programming%29

Please write comments if you find any of the above answers/explanations incorrect.

13. Output of C Programs | Set 10

Predict the output of the below programs.

Difficulty Level: Rookie

Question 1

```
#include<stdio.h>
int main()
{
   typedef int i;
   i a = 0;
   printf("%d", a);
   getchar();
   return 0;
}
```

Output: 0
There is no problem with the program. It simply creates a user defined type *i* and creates a variable *a* of type *i*.

Question 2

```
#include<stdio.h>
int main()
{
  typedef int *i;
  int j = 10;
  i *a = &j;
  printf("%d", **a);
  getchar();
  return 0;
}
```

Output: Compiler Error -> Initialization with incompatible pointer type.
The line *typedef int *i* makes *i* as type int *. So, the declaration of *a* means *a* is pointer to a pointer. The Error message may be different on different compilers.

Question 3

```
#include<stdio.h>
int main()
{
  typedef static int *i;
  int j;
  i a = &j;
  printf("%d", *a);
  getchar();
  return 0;
}
```

Output: Compiler Error -> Multiple Storage classes for a.
In C, typedef is considered as a storage class. The Error message may be different on

different compilers.

Please write comments if you find any of the answers/explanations incorrect, or you want to share more information about the topics discussed above.

References:
http://www.itee.uq.edu.au/~comp2303/Leslie_C_ref/C/SYNTAX/typedef.html
http://publib.boulder.ibm.com/infocenter/macxhelp/v6v81/index.jsp?topic=/com.ibm.vacpp6m.doc/language/ref/clrc03sc03.htm
http://msdn.microsoft.com/en-us/library/4x7sfztk.aspx

14. Output of C Programs | Set 11

Asked by Shobhit

```
#include<stdio.h>
int fun(int n, int *fg)
{
   int t, f;
   if(n <= 1)
   {
     *fg = 1;
      return 1;
   }
   t = fun(n-1, fg);
   f = t + *fg;
   *fg = t;
   return f;
}
int main( )
{
  int x = 15;
  printf ( "%d\n", fun (5, &x));
  getchar();
  return 0;
}
```

In the above program, there will be recursive calls till n is **not** smaller than or equal to 1.

```
fun(5, &x)
       \
         \
       fun(4, fg)
           \
             \
            fun(3, fg)
```

```
                 \
                   \
                   fun(2, fg)
                      \
                        \
                          fun(1, fg)
```

fun(1, fg) does not further call fun() because n is 1 now, and it goes inside the if part. It changes value at address fg to 1, and returns 1.

Inside fun(2, fg)

```
 t = fun(n-1, fg); --> t = 1
/* After fun(1, fg) is called, fun(2, fg) does following */
 f = t + *fg;      -->  f = 1 + 1 (changed by fun(1, fg)) = 2
 *fg = t;           --> *fg = 1
return f (or return 2)
```

Inside fun(3, fg)

```
 t = fun(2, fg); --> t = 2
/* After fun(2, fg) is called, fun(3, fg) does following */
 f = t + *fg;       -->  f = 2 + 1 = 3
 *fg = t;            --> *fg = 2
return f (or return 3)
```

Inside fun(4, fg)

```
 t = fun(3, fg);   --> t = 3
/* After fun(3, fg) is called, fun(4, fg) does following */
 f = t + *fg;       -->  f = 3 + 2 = 5
 *fg = t;            --> *fg = 3
 return f (or return 5)
```

Inside fun(5, fg)

```
 t = fun(4, fg);   -->  t = 5
/* After fun(4, fg) is called, fun(5, fg) does following */
 f = t + *fg;       -->  f = 5 + 3 = 8
 *fg = t;            --> *fg = 5
 return f (or return 8 )
```

Finally, value returned by fun(5, &x) is printed, so 8 is printed on the screen

15. Output of C Programs | Set 12

Predict the output of below programs.

Question 1

```
int fun(char *str1)
{
  char *str2 = str1;
  while(*++str1);
  return (str1-str2);
}

int main()
{
  char *str = "geeksforgeeks";
  printf("%d", fun(str));
  getchar();
  return 0;
}
```

Output: 13
Inside fun(), pointer str2 is initialized as str1 and str1 is moved till '\0' is reached (note **;** after while loop). So str1 will be incremented by 13 (assuming that char takes 1 byte).

Question 2

```
void fun(int *p)
{
  static int q = 10;
  p = &q;
}

int main()
{
  int r = 20;
  int *p = &r;
  fun(p);
  printf("%d", *p);
  getchar();
  return 0;
}
```

Output: 20
Inside fun(), q is a copy of the pointer p. So if we change q to point something else then p remains unaffected.

Question 3

```
void fun(int **p)
{
  static int q = 10;
  *p = &q;
}

int main()
{
  int r = 20;
  int *p = &r;
  fun(&p);
  printf("%d", *p);
  getchar();
  return 0;
}
```

Output 10

Note that we are passing address of p to fun(). p in fun() is actually a pointer to p in main() and we are changing value at p in fun(). So p of main is changed to point q of fun(). To understand it better, let us rename p in fun() to p_ref or ptr_to_p

```
void fun(int **ptr_to_p)
{
  static int q = 10;
  *ptr_to_p = &q;  /*Now p of main is pointing to q*/
}
```

Also, note that the program won't cause any problem because q is a static variable. Static variables exist in memory even after functions return. For an auto variable, we might have seen some weird output because auto variable may not exist in memory after functions return.

Please write comments if you find any of the answers/explanations incorrect, or you want to share more information about the topics discussed above.

16. Output of C Programs | Set 13

Difficulty Level: Rookie

Question 1
Predict the output of below program.

```
int main()
{
  char arr[] = "geeksforgeeks";
  printf("%d", sizeof(arr));
  getchar();
  return 0;
}
```

Output: 14
The string “geeksforgeeks” has 13 characters, but the size is 14 because compiler includes a single ‘\0′ (string terminator) when char array size is not explicitly mentioned.

Question 2
In below program, what would you put in place of “?” to print “geeks”. Obviously, something other than “geeks”.

```
int main()
{
  char arr[] = "geeksforgeeks";
  printf("%s", ?);
  getchar();
  return 0;
}
```

Answer: (arr+8)
The printf statement prints everything starting from arr+8 until it finds ‘\0’

Question 3
Predict the output of below program.

```
int main()
{
  int x, y = 5, z = 5;
  x = y==z;
  printf("%d", x);

  getchar();
  return 0;
}
```

The crux of the question lies in the statement x = y==z. The operator == is executed before = because precedence of comparison operators (<=, >= and ==) is higher than assignment operator =.
The result of a comparison operator is either 0 or 1 based on the comparison result. Since y is equal to z, value of the expression y == z becomes 1 and the value is assigned to x via the assignment operator.

Question 4
Predict the output of below program.

```
int main()
{
  printf(" \"GEEKS %% FOR %% GEEKS\"");
  getchar();
  return 0;
}
```

Output: “GEEKS % FOR % GEEKS”

Backslash (\) works as escape character for double quote (“). For explanation of %%, please see this.

Please write comments if you find any of the answers/explanations incorrect, or you want to share more information about the topics discussed above

17. Result of sizeof operator

Asked by Kapil

Predict the output of below program.

```
#include <stdio.h>
#define TOTAL_ELEMENTS (sizeof(array) / sizeof(array[0]))
int array[] = {1, 2, 3, 4, 5, 6, 7};

int main()
{
 int i;

 for(i = -1; i <= (TOTAL_ELEMENTS-2); i++)
   printf("%d\n", array[i+1]);

 getchar();
 return 0;
}
```

Output: Nothing is printed as loop condition is not true for the first time itself.

The result of sizeof for an array operand is number of elements in the array multiplied by size of an element in bytes. So value of the expression TOTAL_ELEMENTS in the above program is 7.
The data type of the sizeof result is unsigned int or unsigned long depending upon the compiler implementation. Therefore, in the loop condition i <= (TOTAL_ELEMENTS-2), an int is compared to an unsigned value. int is implicitly converted to unsigned (true for both unsigned int and unsigned long). So -1 (unsigned equivalent 4294967295 if integers are stored using 32 bit) is compared to TOTAL_ELEMENTS - 2 and the condition is not true for the first time itself.

Please write comments if you find anything incorrect in the above post.

18. Output of C Programs | Set 14

Predict the output of below C programs.

Question 1

```
#include<stdio.h>
int main()
{
   int a;
   char *x;
   x = (char *) &a;
   a = 512;
   x[0] = 1;
   x[1] = 2;
   printf("%d\n",a);

   getchar();
   return 0;
}
```

Answer: The output is dependent on endianness of a machine. Output is 513 in a little endian machine and 258 in a big endian machine.

Let integers are stored using 16 bits. In a little endian machine, when we do x[0] = 1 and x[1] = 2, the number a is changed to 00000001 00000010 which is representation of 513 in a little endian machine. The output would be same for 32 bit numbers also.

In a big endian machine, when we do x[0] = 1 and x[1] = 2, the number is changed to 00000001 00000010 which is representation of 258 in a big endian machine.

Question 2

```
int main()
{
  int f = 0, g = 1;
  int i;
  for(i = 0; i < 15; i++)
  {
    printf("%d \n", f);
    f = f + g;
    g = f - g;
  }
  getchar();
  return 0;
}
```

Answer: The function prints first 15 Fibonacci Numbers.

Question 3

Explain functionality of following function.

```
int func(int i)
{
  if(i%2) return (i++);
  else return func(func(i-1));
}
```

Answer: If n is odd then returns n, else returns (n-1). So if n is 12 then we get and if n is 11 then we get 11.

Please write comments if you find any of the answers/explanations incorrect, or you want to share more information about the topics discussed above.

19. Output of C Programs | Set 15

Predict the output of below C programs.

Question 1

```
#include<stdio.h>
int main(void)
{
  int a = 1;
  int b = 0;
  b = ++a + ++a;
  printf("%d %d",a,b);
  getchar();
  return 0;
}
```

Output: Undefined Behavior
See http://en.wikipedia.org/wiki/C_syntax#Undefined_behavior)

Question 2

```
#include<stdio.h>

int main()
{
  int a[] = {1, 2, 3, 4, 5, 6};
  int *ptr = (int*)(&a+1);
  printf("%d ", *(ptr-1) );
  getchar();
  return 0;
}
```

Output: 6
&a is address of the whole array *a[]*. If we add 1 to *&a*, we get “base address of a[] + sizeof(a)”. And this value is typecasted to int *. So *ptr – 1* points to last element of a[]

Please write comments if you find any of the answers/explanations incorrect, or you want to share more information about the topics discussed above.

20. Output of C Programs | Set 16

Predict the output of below C programs.

Question 1

```
#include <stdio.h>

char* fun()
{
  return "awake";
}
int main()
{
  printf("%s",fun()+ printf("I see you"));
  getchar();
  return 0;
}
```

Output: Some string starting with “I see you”
Explanation: (Thanks to Venki for suggesting this solution)
The function fun() returns pointer to char. Apart from printing string “I see you”, printf() function returns number of characters it printed(i.e. 9). The expression [fun()+ printf(“I see you”)] can be boiled down to [“awake” + 9] which is nothing but base address of string literal “awake” displaced by 9 characters. Hence, the expression [“awake” + 9] returns junk data when printed via %s specifier till it finds ‘\0′.

Question 2

```
#include <stdio.h>

int main()
{
  unsigned i ;
  for( i = 0 ; i < 4 ; ++i )
  {
    fprintf( stdout , "i = %d\n" , ("11213141") ) ;
  }

  getchar();
  return 0 ;
}
```

Output: Prints different output on different machines.
Explanation: (Thanks to Venki for suggesting this solution)

The format specifier is %d, converts the base address of string “11213141” as an integer. The base address of string depends on memory allocation by the compiler. The for loop prints same address four times. Try to use C++ streams, you will see power of type system.

Please write comments if you find any of the answers/explanations incorrect, or you want to share more information about the topics discussed above

21. Output of C++ Program | Set 1

Predict the output of below C++ programs.

Question 1

```
// Assume that integers take 4 bytes.
#include<iostream>

using namespace std;

class Test
{
  static int i;
  int j;
};

int Test::i;

int main()
{
    cout << sizeof(Test);
    return 0;
}
```

Output: 4 (size of integer)
static data members do not contribute in size of an object. So ‘i’ is not considered in size of Test. Also, all functions (static and non-static both) do not contribute in size.

Question 2

```
#include<iostream>

using namespace std;
class Base1 {
 public:
     Base1()
     { cout << " Base1's constructor called" << endl;  }
};

class Base2 {
 public:
     Base2()
     { cout << "Base2's constructor called" << endl;  }
};

class Derived: public Base1, public Base2 {
   public:
     Derived()
     {  cout << "Derived's constructor called" << endl;  }
};

int main()
{
   Derived d;
   return 0;
}
```

Ouput:

```
Base1's constructor called
Base2's constructor called
Derived's constructor called
```

In case of Multiple Inheritance, constructors of base classes are always called in derivation order from left to right and Destructors are called in reverse order.

22. Output of C Program | Set 17

Predict the output of following C programs.

Question 1

```
#include<stdio.h>

#define R 10
#define C 20

int main()
{
   int (*p)[R][C];
   printf("%d",  sizeof(*p));
   getchar();
   return 0;
}
```

Output: 10*20*sizeof(int) which is "800" for compilers with integer size as 4 bytes.
The pointer p is de-referenced, hence it yields type of the object. In the present case, it is an array of array of integers. So, it prints R*C*sizeof(int).
Thanks to Venki for suggesting this solution.

Question 2

```
#include<stdio.h>
#define f(g,g2) g##g2
int main()
{
   int var12 = 100;
   printf("%d", f(var,12));
   getchar();
   return 0;
}
```

Output: 100
The operator ## is called "Token-Pasting" or "Merge" Operator. It merges two tokens into one token. So, after preprocessing, the main function becomes as follows, and prints 100.

```
int main()
{
   int var12 = 100;
   printf("%d", var12);
   getchar();
   return 0;
}
```

Question 3

```
#include<stdio.h>
int main()
{
   unsigned int x = -1;
   int y = ~0;
   if(x == y)
      printf("same");
   else
      printf("not same");
   printf("\n x is %u, y is %u", x, y);
   getchar();
   return 0;
}
```

Output: “same x is MAXUINT, y is MAXUINT” Where MAXUINT is the maximum possible value for an unsigned integer.
-1 and ~0 essentially have same bit pattern, hence x and y must be same. In the comparison, y is promoted to unsigned and compared against x. The result is “same”. However, when interpreted as signed and unsigned their numerical values will differ. x is MAXUNIT and y is -1. Since we have %u for y also, the output will be MAXUNIT and MAXUNIT.
Thanks to Venki for explanation.

Please write comments if you find any of the answers/explanations incorrect, or you want to share more information about the topics discussed above

23. Output of C++ Program | Set 2

Predict the output of below C++ programs.

Question 1

```
#include<iostream>
using namespace std;

class A {
 public:
    A(int ii = 0) : i(ii) {}
    void show() { cout << "i = " << i << endl;}
 private:
    int i;
};

class B {
 public:
    B(int xx) : x(xx) {}
    operator A() const { return A(x); }
 private:
    int x;
};

void g(A a)
{  a.show(); }

int main() {
  B b(10);
  g(b);
  g(20);
  getchar();
  return 0;
}
```

Output:

```
i = 10
i = 20
```

Since there is a Conversion constructor in class A, integer value can be assigned to objects of class A and function call g(20) works. Also, there is a conversion operator overloaded in class B, so we can call g() with objects of class B.

Question 2

```
#include<iostream>
using namespace std;

class base {
    int arr[10];
};

class b1: public base { };

class b2: public base { };

class derived: public b1, public b2 {};

int main(void)
{
  cout<<sizeof(derived);
  getchar();
  return 0;
}
```

Output: If integer takes 4 bytes, then 80.

Since *b1* and *b2* both inherit from class *base*, two copies of class *base* are there in class *derived*. This kind of inheritance without virtual causes wastage of space and ambiguities. virtual base classes are used to save space and avoid ambiguities in such cases. For example, following program prints 48. 8 extra bytes are for bookkeeping information stored by the compiler (See this for details)

```
#include<iostream>
using namespace std;

class base {
  int arr[10];
};

class b1: virtual public base { };

class b2: virtual public base { };

class derived: public b1, public b2 {};

int main(void)
{
  cout<<sizeof(derived);
  getchar();
  return 0;
}
```

Please write comments if you find any of the answers/explanations incorrect, or you want to share more information about the topics discussed above

24. Output of C Program | Set 18

Predict the output of following C programs.

Question 1

```
#include<stdio.h>
int fun()
{
  static int num = 40;
  return num--;
}

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

Output: 38 35 32 29 26 23 20 17 14 11 8 5 2

Since *num* is static in fun(), the old value of *num* is preserved for subsequent functions calls. Also, since the statement *return num–* is postfix, it returns the old value of *num*, and updates the value for next function call.

Question 2

```
#include<stdio.h>
int main()
{
  char *s[] = { "knowledge","is","power"};
  char **p;
  p = s;
  printf("%s ", ++*p);
  printf("%s ", *p++);
  printf("%s ", ++*p);

  getchar();
  return 0;
}
```

Output: nowledge nowledge s

Let us consider the expression ++*p* in first printf(). Since precedence of prefix ++ and * is same, associativity comes into picture. *p* is evaluated first because both prefix ++ and * are right to left associative. When we increment *p* by 1, it starts pointing to second character of *"knowledge"*. Therefore, the first printf statement prints *"nowledge"*.

Let us consider the expression *p*++ in second printf() . Since precedence of postfix ++ is higher than *, *p*++ is evaluated first. And since it's a psotfix ++, old value of *p* is used in this expression. Therefore, second printf statement prints *"nowledge"*.

In third printf statement, the new value of *p* (updated by second printf) is used, and third printf() prints *"s"*.

Please write comments if you find any of the answers/explanations incorrect, or you want to share more information about the topics discussed above

25. Output of C Program | Set 19

Predict the outputs of following program.

Difficulty Level: Rookie

Question 1

```
#include <stdio.h>
int main()
{
  int a = 10, b = 20, c = 30;
  if (c > b > a)
  {
    printf("TRUE");
  }
  else
  {
    printf("FALSE");
  }
  getchar();
  return 0;
}
```

Output: *FALSE*
Let us consider the condition inside the if statement. Since there are two greater than (>) operators in expression "c > b > a", associativity of > is considered. Associativity of > is left to right. So, expression c > b > a is evaluated as ((c > b) > a) which is false.

Please write comments if you find any of the answers/explanations incorrect, or you want to share more information about the topics discussed above

26. Output of C++ Program | Set 3

Predict the output of below C++ programs.

Question 1

```
#include<iostream>

using namespace std;
class P {
public:
   void print()
   { cout <<" Inside P::"; }
};

class Q : public P {
public:
   void print()
   { cout <<" Inside Q"; }
};

class R: public Q {
};

int main(void)
{
  R r;

  r.print();
  return 0;
}
```

Output:
Inside Q

The print function is not defined in class R. So it is looked up in the inheritance hierarchy. *print()* is present in both classes *P* and *Q*, which of them should be called? The idea is, if there is multilevel inheritance, then function is linearly searched up in the inheritance heirarchy until a matching function is found.

Question 2

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

using namespace std;

class Base
{
public:
  Base()
  {
    fun(); //note: fun() is virtual
  }
  virtual void fun()
  {
    cout<<"\nBase Function";
  }
};

class Derived: public Base
{
public:
  Derived(){}
  virtual void fun()
  {
    cout<<"\nDerived Function";
  }
};

int main()
{
  Base* pBase = new Derived();
  delete pBase;
  return 0;
}
```

Output:

Base Function

See following excerpt from C++ standard for explanation.

When a virtual function is called directly or indirectly from a constructor (including from the mem-initializer for a data member) or from a destructor, and the object to which the call applies is the object under construction or destruction, the function called is the one defined in the constructor or destructor's own class or in one of its bases, but not a function overriding it in a class derived from the constructor or destructor's class, or overriding it in one of the other base classes of the most derived object.

Because of this difference in behavior, it is recommended that object's virtual function is not invoked while it is being constructed or destroyed. See this for more details.

Please write comments if you find any of the answers/explanations incorrect, or you want to share more information about the topics discussed above.

27. Output of C Program | Set 20

Predict the outputs of following C programs.

Question 1

```
int main()
{
  int x = 10;
  static int y = x;

  if(x == y)
     printf("Equal");
  else if(x > y)
     printf("Greater");
  else
     printf("Less");

  getchar();
  return 0;
}
```

Output: Compiler Error
In C, static variables can only be initialized using constant literals. See this GFact for details.

Question 2

```
#include <stdio.h>

int main()
{
  int i;

  for (i = 1; i != 10; i += 2)
  {
    printf(" GeeksforGeeks ");
  }

  getchar();
  return 0;
}
```

Output: Infinite times GeeksforGeeks

The loop termination condition never becomes true and the loop prints GeeksforGeeks infinite times. In general, if a *for* or *while* statement uses a loop counter, then it is safer to use a relational operator (such as <) to terminate the loop than using an inequality operator (operator !=). See this for details.

Question 3

```
#include<stdio.h>
struct st
{
    int x;
    struct st next;
};

int main()
{
    struct st temp;
    temp.x = 10;
    temp.next = temp;
    printf("%d", temp.next.x);

    getchar();
    return 0;
}
```

Output: Compiler Error
A C structure cannot contain a member of its own type because if this is allowed then it becomes impossible for compiler to know size of such struct. Although a pointer of same type can be a member because pointers of all types are of same size and compiler can calculate size of struct.

Please write comments if you find any of the answers/explanations incorrect, or you want to share more information about the topics discussed above

28. Output of C++ Program | Set 4

Difficulty Level: Rookie

Predict the output of below C++ programs.

Question 1

```
#include<iostream>
using namespace std;

int x = 10;
void fun()
{
    int x = 2;
    {
        int x = 1;
        cout << ::x << endl;
    }
}

int main()
{
    fun();
    return 0;
}
```

Output: 10

If Scope Resolution Operator is placed before a variable name then the global variable is referenced. So if we remove the following line from the above program then it will fail in compilation.

```
int x = 10;
```

Question 2

```
#include<iostream>
using namespace std;
class Point {
private:
    int x;
    int y;
public:
    Point(int i, int j);  // Constructor
};

Point::Point(int i = 0, int j = 0)  {
    x = i;
    y = j;
    cout << "Constructor called";
}

int main()
{
   Point t1, *t2;
   return 0;
}
```

Output: Constructor called.

If we take a closer look at the statement "Point t1, *t2;:" then we can see that only one object is constructed here. t2 is just a pointer variable, not an object.

Question 3

```
#include<iostream>
using namespace std;

class Point {
private:
    int x;
    int y;
public:
    Point(int i = 0, int j = 0);    // Normal Constructor
    Point(const Point &t); // Copy Constructor
};

Point::Point(int i, int j)  {
    x = i;
    y = j;
    cout << "Normal Constructor called\n";
}

Point::Point(const Point &t) {
   y = t.y;
   cout << "Copy Constructor called\n";
}

int main()
{
   Point *t1, *t2;
   t1 = new Point(10, 15);
   t2 = new Point(*t1);
   Point t3 = *t1;
   Point t4;
   t4 = t3;
   return 0;
}
```

Output:

```
Normal Constructor called
Copy Constructor called
Copy Constructor called
Normal Constructor called
```

See following comments for explanation:

```
Point *t1, *t2;   // No constructor call
t1 = new Point(10, 15);  // Normal constructor call
t2 = new Point(*t1);   // Copy constructor call
Point t3 = *t1;  // Copy Constructor call
Point t4;   // Normal Constructor call
t4 = t3;   // Assignment operator call
```

Please write comments if you find any of the answers/explanations incorrect, or you want to share more information about the topics discussed above

29. Output of C++ Program | Set 5

Difficulty Level: Rookie

Predict the output of below C++ programs.

Question 1

```
#include<iostream>
using namespace std;

class Test {
    int value;
public:
    Test(int v);
};

Test::Test(int v) {
    value = v;
}

int main() {
    Test t[100];
    return 0;
}
```

Output: Compiler error

The class Test has one user defined constructor “Test(int v)” that expects one argument. It doesn’t have a constructor without any argument as the compiler doesn’t create the default constructor if user defines a constructor (See this). Following modified program works without any error.

```
#include<iostream>
using namespace std;

class Test {
    int value;
public:
    Test(int v = 0);
};

Test::Test(int v) {
    value = v;
}

int main() {
    Test t[100];
    return 0;
}
```

Question 2

```
#include<iostream>
using namespace std;
int &fun() {
  static int a = 10;
  return a;
}

int main() {
  int &y = fun();
  y = y +30;
  cout<<fun();
  return 0;
}
```

Output: 40

The program works fine because ‘a’ is static. Since ‘a’ is static, memory location of it remains valid even after fun() returns. So a reference to static variable can be returned.

Question 3

```
#include<iostream>
using namespace std;

class Test
{
public:
  Test();
};

Test::Test()  {
    cout<<"Constructor Called \n";
}

int main()
{
    cout<<"Start \n";
    Test t1();
    cout<<"End \n";
    return 0;
}
```

Output:

```
Start
End
```

Note that the line “Test t1();” is not a constructor call. Compiler considers this line as declaration of function t1 that doesn’t recieve any parameter and returns object of type Test.

Please write comments if you find any of the answers/explanations incorrect, or you want to share more information about the topics discussed above

30. Output of C++ Program | Set 6

Predict the output of below C++ programs.

Question 1

```
#include<iostream>

using namespace std;

class Test {
    int value;
public:
    Test (int v = 0) {value = v;}
    int getValue() { return value; }
};

int main() {
    const Test t;
    cout << t.getValue();
    return 0;
}
```

Output: Compiler Error.

A const object cannot call a non-const function. The above code can be fixed by either making getValue() const or making t non-const. Following is modified program with getValue() as const, it works fine and prints 0.

```
#include<iostream>

using namespace std;

class Test {
    int value;
public:
    Test (int v = 0) { value = v; }
    int getValue() const { return value; }
};

int main() {
    const Test t;
    cout << t.getValue();
    return 0;
}
```

Question 2

```cpp
#include<iostream>

using namespace std;

class Test {
    int &t;
public:
    Test (int &x) { t = x; }
    int getT() { return t; }
};

int main()
{
    int x = 20;
    Test t1(x);
    cout << t1.getT() << " ";
    x = 30;
    cout << t1.getT() << endl;
    return 0;
}
```

Output: Compiler Error
Since t is a reference in Test, it must be initialized using Initializer List. Following is the modified program. It works and prints “20 30″.

```cpp
#include<iostream>

using namespace std;

class Test {
    int &t;
public:
    Test (int &x):t(x) {  }
    int getT() { return t; }
};

int main() {
    int x = 20;
    Test t1(x);
    cout << t1.getT() << " ";
    x = 30;
    cout << t1.getT() << endl;
    return 0;
}
```

Please write comments if you find any of the answers/explanations incorrect, or you want to share more information about the topics discussed above

31. Output of C++ Program | Set 7

Predict the output of following C++ programs.

Question 1

```
class Test1 {
    int y;
};

class Test2 {
    int x;
    Test1 t1;
public:
    operator Test1() { return t1; }
    operator int() { return x; }
};

void fun ( int x)  { };
void fun ( Test1 t ) { };

int main() {
    Test2 t;
    fun(t);
    return 0;
}
```

Output: Compiler Error
There are two conversion operators defined in the Test2 class. So Test2 objects can automatically be converted to both int and Test1. Therefore, the function call fun(t) is ambiguous as there are two functions void fun(int) and void fun(Test1), compiler has no way to decide which function to call. In general, conversion operators must be overloaded carefully as they may lead to ambiguity.

Question 2

```
#include <iostream>
using namespace std;

class X {
private:
  static const int a = 76;
public:
  static int getA() { return a; }
};

int main() {
  cout <<X::getA()<<endl;
  return 0;
}
```

Output: The program compiles and prints 76
Generally, it is not allowed to initialize data members in C++ class declaration, but static const integral members are treated differently and can be initialized with declaration.

Please write comments if you find any of the answers/explanations incorrect, or you want to share more information about the topics discussed above.

32. Output of C++ Program | Set 8

Predict the output of following C++ programs.

Question 1

```
#include<iostream>
using namespace std;

class Test1
{
    int x;
public:
    void show() {  }
};

class Test2
{
    int x;
public:
    virtual void show() {  }
};

int main(void)
{
    cout<<sizeof(Test1)<<endl;
    cout<<sizeof(Test2)<<endl;
    return 0;
}
```

Output:

```
4
8
```

There is only one difference between Test1 and Test2. show() is non-virtual in Test1, but virtual in Test2. When we make a function virtual, compiler adds an extra pointer vptr to objects of the class. Compiler does this to achieve run time polymorphism (See chapter 15 of Thinking in C++ book for more details). The extra pointer vptr adds to the size of objects, that is why we get 8 as size of Test2.

Question 2

```
#include<iostream>
using namespace std;
class P
{
public:
    virtual void show() = 0;
};

class Q : public P {
   int x;
};

int main(void)
{
    Q q;
    return 0;
}
```

Output: Compiler Error
We get the error because we can’t create objects of abstract classes. P is an abstract class as it has a pure virtual method. Class Q also becomes abstract because it is derived from P and it doesn’t implement show().

Please write comments if you find any of the answers/explanations incorrect, or you want to share more information about the topics discussed above.

33. Output of C Program | Set 21

Predict the output of following C programs.

Question 1

```
#include<stdio.h>
#define fun (x) (x)*10

int main()
{
    int t = fun(5);
    int i;
    for(i = 0; i < t; i++)
       printf("GeesforGeeks\n");

    return 0;
}
```

Output: Compiler Error
There is an extra space in macro declaration which causes fun to be raplaced by (x). If we remove the extra space then program works fine and prints “GeeksforGeeks” 50 times. Following is the working program.

```
#include<stdio.h>
#define fun(x) (x)*10

int main()
{
    int t = fun(5);
    int i;
    for(i = 0; i < t; i++)
       printf("GeesforGeeks\n");

    return 0;
}
```

Be careful when dealing with macros. Extra spaces may lead to problems.

Question 2

```
#include<stdio.h>
int main()
{
    int i = 20,j;
    i = (printf("Hello"), printf(" All Geeks "));
    printf("%d", i);

    return 0;
}
```

Output: Hello All Geeks 11
The printf() function returns the number of characters it has successfully printed. The comma operator evaluates it operands from left to right and returns the value returned by the rightmost expression (See this for more details). First *printf("Hello")* executes and prints *"Hello"*, the *printf(" All Geeks ")* executes and prints *" All Geeks "*. This printf statement returns 11 which is assigned to i.

Please write comments if you find any of the answers/explanations incorrect, or you want to share more information about the topics discussed above

34. Output of C Program | Set 22

Predict the output of following C programs.

Question 1

```c
#include<stdio.h>

int main()
{
    enum channel {star, sony, zee};
    enum symbol {hash, star};

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

    return 0;
}
```

Output:

```
compiler error: redeclaration of enumerator 'star'
```

In the above program, enumartion constant *'star'* appears two times in main() which causes the error. An enumaration constant must be unique within the scope in which it is defined. The following program works fine and prints 0 1 2 as the enumaration constants automatically get the values starting from 0.

```c
#include<stdio.h>

int main()
{
    enum channel {star, sony, zee};

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

    return 0;
}
```

Output:

```
0 1 2
```

Question 2

```
#include<stdio.h>

int main()
{
    int i, j;
    int p = 0, q = 2;

    for(i = 0, j = 0; i < p, j < q; i++, j++)
    {
      printf("GeeksforGeeks\n");
    }

    return 0;
}
```

Output:

```
GeeksforGeeks
GeeksforGeeks
```

Following is the main expression to consider in the above program.

```
i < p, j < q
```

When two expressions are separated by comma operator, the first expression (i < p) is executed first. Result of the first expression is ignored. Then the second expression (j < q) is executed and the result of this second expression is the final result of the complete expression (i < p, j < q). The value of expression 'j < q' is true for two iterations, so we get "GeeksforGeeks" two times on the screen. See this for more details.

Please write comments if you find any of the answers/explanations incorrect, or you want to share more information about the topics discussed above

35. Output of C++ Program | Set 9

Predict the output of following C++ programs.

Question 1

```
template <class S, class T> class Pair
{
private:
    S x;
    T y;
/* ... */
};

template <class S> class Element
{
private:
    S x;
/* ... */
};

int main ()
{
    Pair <Element<int>, Element<char>> p;
    return 0;
}
```

Output:

```
Compiler Error: '>>' should be '> >' within a nested template argument list
```

When we use nested templates in our program, we must put a space between two closing angular brackets, otherwise it conflicts with operator >>. For example, following program compiles fine.

```
template <class S, class T> class Pair
{
private:
    S x;
    T y;
/* ... */
};

template <class S> class Element
{
private:
    S x;
/* ... */
};

int main ()
{
    Pair <Element<int>, Element<char> > p;    // note the space between
    return 0;
}
```

Question 2

```
#include<iostream>
using namespace std;

class Test
{
private:
    static int count;
public:
    static Test& fun();
};

int Test::count = 0;

Test& Test::fun()
{
    Test::count++;
    cout<<Test::count<<" ";
    return *this;
}

int main()
{
    Test t;
    t.fun().fun().fun().fun();
    return 0;
}
```

Output:

```
Compiler Error: 'this' is unavailable for static member functions
```

this pointer is not available to static member methods in C++, as static methods can be called using class name also. Similarly in Java, static member methods cannot access this and super (super is for base or parent class).
If we make fun() non-static in the above program, then the program works fine.

```
#include<iostream>
using namespace std;

class Test
{
private:
    static int count;
public:
    Test& fun(); // fun() is non-static now
};

int Test::count = 0;

Test& Test::fun()
{
    Test::count++;
    cout<<Test::count<<" ";
    return *this;
}

int main()
{
    Test t;
    t.fun().fun().fun().fun();
    return 0;
}
```

Output:

```
Output:
1 2 3 4
```

Please write comments if you find any of the answers/explanations incorrect, or want to share more information about the topics discussed above.

36. Output of Java Program | Set 1

Difficulty Level: Rookie

Predict the output of following Java Programs.

Program 1

```
// filename Main.java
class Test {
    protected int x, y;
}

class Main {
    public static void main(String args[]) {
        Test t = new Test();
        System.out.println(t.x + " " + t.y);
    }
}
```

Output
0 0
In Java, a protected member is accessible in all classes of same package and in inherited classes of other packages. Since Test and Main are in same package, no access related problem in the above program. Also, the default constructors initialize integral variables as 0 in Java (See this GFact for more details). That is why we get output as 0 0.

Program 2

```
// filename Test.java
class Test {
    public static void main(String[] args) {
        for(int i = 0; 1; i++) {
            System.out.println("Hello");
            break;
        }
    }
}
```

Output: Compiler Error
There is an error in condition check expression of for loop. Java differs from C++(or C) here. C++ considers all non-zero values as true and 0 as false. Unlike C++, an integer value expression cannot be placed where a boolean is expected in Java. Following is the corrected program.

```
// filename Test.java
class Test {
    public static void main(String[] args) {
        for(int i = 0; true; i++) {
            System.out.println("Hello");
            break;
        }
    }
}
// Output: Hello
```

Program 3

```
// filename Main.java
class Main {
    public static void main(String args[]) {
        System.out.println(fun());
    }
    int fun() {
        return 20;
    }
}
```

Output: Compiler Error
Like C++, in Java, non-static methods cannot be called in a static method. If we make fun() static, then the program compiles fine without any compiler error. Following is the corrected program.

```
// filename Main.java
class Main {
    public static void main(String args[]) {
        System.out.println(fun());
    }
    static int fun() {
        return 20;
    }
}
// Output: 20
```

Program 4

```
// filename Test.java
class Test {
   public static void main(String args[]) {
       System.out.println(fun());
   }
   static int fun() {
       static int x= 0;
       return ++x;
   }
}
```

Output: Compiler Error
Unlike C++, static local variables are not allowed in Java. See this GFact for details. We can have class static members to count number of function calls and other purposes that C++ local static variables serve. Following is the corrected program.

```
class Test {
   private static int x;
   public static void main(String args[]) {
       System.out.println(fun());
   }
   static int fun() {
       return ++x;
   }
}
// Output: 1
```

Please write comments if you find any of the answers/explanations incorrect, or want to share more information about the topics discussed above.

37. Output of Java Program | Set 2

Predict the output of following Java programs.

Question 1

```
package main;

class Base {
    public void Print() {
        System.out.println("Base");
    }
}

class Derived extends Base {
    public void Print() {
        System.out.println("Derived");
    }
}

class Main{
    public static void DoPrint( Base o ) {
        o.Print();
    }
    public static void main(String[] args) {
        Base x = new Base();
        Base y = new Derived();
        Derived z = new Derived();
        DoPrint(x);
        DoPrint(y);
        DoPrint(z);
    }
}
```

Output:

```
Base
Derived
Derived
```

Predicting the first line of output is easy. We create an object of type Base and call DoPrint(). DoPrint calls the print function and we get the first line.

DoPrint(y) causes second line of output. Like C++, assigning a derived class reference to a base class reference is allowed in Java. Therefore, the expression Base y = new Derived() is a valid statement in Java. In DoPrint(), o starts referring to the same object as referred by y. Also, unlike C++, functions are virtual by default in Java. So, when we call o.print(), the print() method of Derived class is called due to run time polymorphism present by default in Java.

DoPrint(z) causes third line of output, we pass a reference of Derived type and the print() method of Derived class is called again. The point to note here is: unlike C++, object slicing doesn't happen in Java. Because non-primitive types are always assigned by reference.

Question 2

```
package main;

// filename Main.java
class Point {
    protected int x, y;

    public Point(int _x, int _y) {
        x = _x;
        y = _y;
    }
}

public class Main {
    public static void main(String args[]) {
        Point p = new Point();
        System.out.println("x = " + p.x + ", y = " + p.y);
    }
}
```

Output:
Compiler Error
In the above program, there are no access permission issues because the Test and Main are in same package and protected members of a class can be accessed in other classes of same package. The problem with the code is: there is not default constructor in Point. Like C++, if we write our own parametrized constructor then Java compiler doesn't create the default constructor. So there are following two changes to Point class that can fix the above program.
1) Remove the parametrized constructor.
2) Add a constructor without any parameter.
Java doesn't support default arguments, so that is not an option.

Please write comments if you find any of the answers/explanations incorrect, or want to share more information about the topics discussed above.

38. Output of C Program | Set 23

Predict the output of following C Program.

```
#include <stdio.h>
#define R 4
#define C 4

void modifyMatrix(int mat[][C])
{
   mat++;
   mat[1][1] = 100;
   mat++;
   mat[1][1] = 200;
}

void printMatrix(int mat[][C])
{
    int i, j;
    for (i = 0; i < R; i++)
    {
        for (j = 0; j < C; j++)
            printf("%3d ", mat[i][j]);
        printf("\n");
    }
}

int main()
{
    int mat[R][C] = { {1, 2, 3, 4},
        {5, 6, 7, 8},
        {9, 10, 11, 12},
        {13, 14, 15, 16}
    };
    printf("Original Matrix \n");
    printMatrix(mat);

    modifyMatrix(mat);

    printf("Matrix after modification \n");
    printMatrix(mat);

    return 0;
}
```

Output: The program compiles fine and produces following output:

```
Original Matrix
  1   2   3   4
  5   6   7   8
  9  10  11  12
 13  14  15  16
Matrix after modification
  1   2   3   4
  5   6   7   8
  9 100  11  12
 13 200  15  16
```

At first look, the line *“mat++;”* in *modifyMatrix()* seems invalid. But this is a valid C line as array parameters are always pointers (see this and this for details). In *modifyMatrix()*, *mat* is just a pointer that points to block of size *C*sizeof(int)*. So following function prototype is same as *"void modifyMatrix(int mat[][C])"*

```
void modifyMatrix(int (*mat)[C]);
```

When we do *mat++, mat* starts pointing to next row, and *mat[1][1]* starts referring to value 10. *mat[1][1]* (value 10) is changed to 100 by the statement *"mat[1][1] = 100;"*. *mat* is again incremented and *mat[1][1]* (now value 14) is changed to 200 by next couple of statements in *modifyMatrix()*.

The line *"mat[1][1] = 100;"* is valid as pointer arithmetic and array indexing are equivalent in C.

On a side note, we can't do *mat++* in *main()* as *mat* is 2 D array in *main()*, not a pointer.

Please write comments if you find above answer/explanation incorrect, or you want to share more information about the topic discussed above

39. Output of C Program | Set 24

Predict the output of following C programs:

Difficulty Level: Rookie

Question 1

```
#include<stdio.h>
int main()
{
    int arr[] = {10, 20, 30, 40, 50, 60};
    int *ptr1 = arr;
    int *ptr2 = arr + 5;
    printf ("ptr2 - ptr1 = %d\n", ptr2 - ptr1);
    printf ("(char*)ptr2 - (char*) ptr1 = %d",  (char*)ptr2 - (char*)p
    getchar();
    return 0;
}
```

Output:

```
ptr2 - ptr1 = 5
(char*)ptr2 - (char*) ptr1 = 20
```

In C, array name gives address of the first element in the array. So when we do ptr1 = arr, ptr1 starts pointing to address of first element of arr. Since array elements are accessed using pointer arithmetic, arr + 5 is a valid expression and gives the address of

6th element. Predicting value ptr2 – ptr1 is easy, it gives 5 as there are 5 inetegers between these two addresses. When we do (char *)ptr2, ptr2 is typecasted to char pointer. In expression "(int*)ptr2 – (int*)ptr1", pointer arithmetic happens considering character poitners. Since size of a character is one byte, we get 5*sizeof(int) (which is 20) as difference of two pointers.
As an excercise, predict the output of following program.

```
#include<stdio.h>
int main()
{
    char arr[] = "geeksforgeeks";
    char *ptr1 = arr;
    char *ptr2 = ptr1 + 3;
    printf ("ptr2 - ptr1 = %d\n", ptr2 - ptr1);
    printf ("(int*)ptr2 - (int*) ptr1 = %d",  (int*)ptr2 - (int*)ptr1)
    getchar();
    return 0;
}
```

Question 2

```
#include<stdio.h>

int main()
{
  char arr[] = "geeks\0 for geeks";
  char *str = "geeks\0 for geeks";
  printf ("arr = %s, sizeof(arr) = %d \n", arr, sizeof(arr));
  printf ("str = %s, sizeof(str) = %d", str, sizeof(str));
  getchar();
  return 0;
}
```

Output:

```
arr = geeks, sizeof(arr) = 17

str = geeks, sizeof(str) = 4
```

Let us first talk about first output *"arr = geeks"*. When %s is used to print a string, printf starts from the first character at given address and keeps printing characters until it sees a string termination character, so we get *"arr = geeks"* as there is a \0 after geeks in arr[].
Now let us talk about output *"sizeof(arr) = 17"*. When a character array is initialized with a double quoted string and array size is not specified, compiler automatically allocates one extra space for string terminator '\0' (See this Gfact), that is why size of arr is 17.
Explanation for printing "str = geeks" is same as printing "arr = geeks". Talking about value of sizeof(str), str is just a pointer (not array), so we get size of a pointer as output.

Please write comments if you find above answer/explanation incorrect, or you want to share more information about the topic discussed above

40. Output of C program | Set 25

Predict the output of following C program.

```
int main(void)
{
    struct str
    {
        int i: 1;
        int j: 2;
        int k: 3;
        int l: 4;
    };

    struct str s;

    s.i = 1;
    s.j = 2;
    s.k = 5;
    s.l = 10;

    printf(" i: %d \n j: %d \n k: %d \n l: %d \n", s.i, s.j, s.k, s.l)

    getchar();
    return 0;
}
```

The above code is non-portable and output is compiler dependent. We get following output using GCC compiler for intel 32 bit machine.

```
[narendra@ubuntu]$ ./structure
i: -1
j: -2
k: -3
l: -6
```

Let us take a closer look at declaration of structure.

```
struct str
{
    int i: 1;
    int j: 2;
    int k: 3;
    int l: 4;
};
```

In the structure declaration, for structure member ‘i’, we used width of bit field as 1, width of ‘j’ as 2, and so on. At first, it looks we can store values in range [0-1] for ‘i’, range [0-3] for ‘j’, and so on. But in the above declaration, type of bit fields is integer (**signed**).

That’s why out of available bits, 1 bit is used to store sign information. So for ‘i’, the values we can store are 0 or -1 (for a machine that uses two’s complement to store signed integers). For variable ‘k’, number of bits is 3. Out of these 3 bits, 2 bits are used to store data and 1 bit is used to store sign.

Let use declare structure members as “unsigned int” and check output.

```
int main(void)
{
    struct str
    {
        unsigned int i: 1;
        unsigned int j: 2;
        unsigned int k: 3;
        unsigned int l: 4;
    };
    struct str s;

    s.i = 1;
    s.j = 2;
    s.k = 5;
    s.l = 10;

    printf(" i: %d \n j: %d \n k: %d \n l: %d \n", s.i, s.j, s.k, s.l)

    getchar();
    return 0;
}
```

output:

```
[narendra@ubuntu]$ ./structure
i: 1
j: 2
k: 5
l: 10
```

This article is compiled by “Narendra Kangralkar“ and reviewed by GeeksforGeeks team. Please write comments if you find anything incorrect, or you want to share more information about the topic discussed above.

41. Output of C Program | Set 26

Predict the output of following C programs.

Question 1

```c
#include <stdio.h>

int main()
{
  int arr[] = {};
  printf("%d", sizeof(arr));
  return 0;
}
```

Output: 0

C (or C++) allows arrays of size 0. When an array is declared with empty initialization list, size of the array becomes 0.

Question 2

```c
#include<stdio.h>

int main()
{
  int i, j;
  int arr[4][4] = { {1, 2, 3, 4},
                    {5, 6, 7, 8},
                    {9, 10, 11, 12},
                    {13, 14, 15, 16} };
  for(i = 0; i < 4; i++)
    for(j = 0; j < 4; j++)
      printf("%d ", j[i[arr]] );

  printf("\n");

  for(i = 0; i < 4; i++)
    for(j = 0; j < 4; j++)
      printf("%d ", i[j[arr]] );

  return 0;
}
```

Output:

```
1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16
1 5 9 13 2 6 10 14 3 7 11 15 4 8 12 16
```

Array elements are accessed using pointer arithmetic. So the meaning of arr[i][j] and j[i[arr]] is same. They both mean (arr + 4*i + j). Similarly, the meaning of arr[j][i] and i[j[arr]] is same.

Question 3

```
#include<stdio.h>
int main()
{
    int a[2][3] = {2,1,3,2,3,4};
    printf("Using pointer notations:\n");
    printf("%d %d %d\n", *(*(a+0)+0), *(*(a+0)+1), *(*(a+0)+2));
    printf("Using mixed notations:\n");
    printf("%d %d %d\n", *(a[1]+0), *(a[1]+1), *(a[1]+2));
    return 0;
}
```

Output:

```
Using pointer notations:
2 1 3
Using mixed notations:
2 3 4
```

Please write comments if you find anything incorrect, or you want to share more information about the topic discussed above.

42. Output of C++ Program | Set 10

Predict the output of following C++ programs.

Question 1

```
#include<iostream>
#include<string.h>
using namespace std;

class String
{
    char *p;
    int len;
public:
    String(const char *a);
};

String::String(const char *a)
{
    int length = strlen(a);
    p = new char[length +1];
    strcpy(p, a);
    cout << "Constructor Called " << endl;
}

int main()
{
    String s1("Geeks");
    const char *name = "forGeeks";
    s1 = name;
    return 0;
}
```

Output:

```
Constructor called
Constructor called
```

The first line of output is printed by statement “String s1(“Geeks”);” and the second line is printed by statement “s1 = name;”. The reason for the second call is, a single parameter constructor also works as a conversion operator (See this and this for deatils).

Question 2

```
#include<iostream>

using namespace std;

class A
{
    public:
    virtual void fun() {cout << "A" << endl ;}
};
class B: public A
{
    public:
    virtual void fun() {cout << "B" << endl;}
};
class C: public B
{
    public:
    virtual void fun() {cout << "C" << endl;}
};

int main()
{
    A *a = new C;
    A *b = new B;
    a->fun();
    b->fun();
    return 0;
}
```

Output:

```
C
B
```

A base class pointer can point to objects of children classes. A base class pointer can also point to objects of grandchildren classes. Therefor, the line “A *a = new C;” is valid. The line “a->fun();” prints “C” because the object pointed is of class C and fun() is declared virtual in both A and B (See this for details). The second line of output is printed by statement “b->fun();”.

Please write comments if you find any of the answers/explanations incorrect, or want to share more information about the topics discussed above.

43. Output of C++ Program | Set 11

Predict the output of following C++ programs.

Question 1

```cpp
#include<iostream>
using namespace std;

class Point
{
private:
    int x;
    int y;
public:
    Point(const Point&p) { x = p.x; y = p.y; }
    void setX(int i) {x = i;}
    void setY(int j) {y = j;}
    int getX() {return x;}
    int getY() {return y;}
    void print() { cout << "x = " << getX() << ", y = " << getY(); }
};


int main()
{
    Point p1;
    p1.setX(10);
    p1.setY(20);
    Point p2 = p1;
    p2.print();
    return 0;
}
```

Output: Compiler Error in first line of main(), i.e., “Point p1;”

Since there is a user defined constructor, compiler doesn’t create the default constructor (See this GFact). If we remove the copy constructor from class Point, the program works fine and prints the output as “x = 10, y = 20″

Question 2

```cpp
#include<iostream>
using namespace std;

int main()
{
    int *ptr = new int(5);
    cout << *ptr;
    return 0;
}
```

Output: 5

The new operator can also initialize primitive data types. In the above program, the value at address ‘ptr ‘ is initialized as 5 using the new operator.

Question 3

```
#include <iostream>
using namespace std;

class Fraction
{
private:
    int den;
    int num;
public:
   void print() { cout << num << "/" << den; }
   Fraction() { num = 1; den = 1; }
   int &Den() { return den; }
   int &Num() { return num; }
};

int main()
{
   Fraction f1;
   f1.Num() = 7;
   f1.Den() = 9;
   f1.print();
   return 0;
}
```

Output: 7/9
The methods Num() and Den() return references to num and den respectively. Since references are returned, the returned values can be uses as an lvalue, and the private members den and num are modified. The program compiles and runs fine, but this kind of class design is strongly discouraged (See this). Returning reference to private variable allows users of the class to change private data directly which defeats the purpose of encapsulation.

Please write comments if you find any of the answers/explanations incorrect, or you want to share more information about the topics discussed above.

44. Output of C++ Program | Set 12

Predict the output of following C++ programs.

Question 1

```
#include <iostream>
using namespace std;

int fun(int a, int b  = 1, int c =2)
{
    return (a + b + c);
}

int main()
{
    cout << fun(12, ,2);
    return 0;
}
```

Output: Compiler Error in function call fun(12, ,2)
With default arguments, we cannot skip an argument in the middle. Once an argument is skipped, all the following arguments must be skipped. The calls fun(12) and fun(12, 2) are valid.

Question 2

```
#include<iostream>
using namespace std;

/* local variable is same as a member's name */
class Test
{
private:
    int x;
public:
    void setX (int x) { Test::x = x; }
    void print() { cout << "x = " << x << endl; }
};

int main()
{
    Test obj;
    int x = 40;
    obj.setX(x);
    obj.print();
    return 0;
}
```

Output:

```
x = 40
```

Scope resolution operator can always be used to access a class member when it is made hidden by local variables. So the line “Test::x = x” is same as “this->x = x”

Question 3

```
#include<iostream>
using namespace std;

class Test
{
private:
    int x;
    static int count;
public:
    Test(int i = 0) : x(i) {}
    Test(const Test& rhs) : x(rhs.x) { ++count;  }
    static int getCount() { return count; }
};

int Test::count = 0;

Test fun()
{
    return Test();
}

int main()
{
    Test a = fun();
    cout<< Test::getCount();
    return 0;
}
```

Output: Compiler Dependent
The line "Test a = fun()" may or may not call copy constructor. So output may be 0 or 1. If copy elision happens in your compiler, the copy constructor will not be called. If copy elision doesn't happen, copy constructor will be called. The gcc compiler produced the output as 0.

Please write comments if you find any of the answers/explanations incorrect, or you want to share more information about the topics discussed above.

45. Output of C Program | Set 27

Predict the output of following C programs.

Question 1

```
#include <stdio.h>

int main(void)
{
    int i;
    int power_of_ten[5] = {
                            00001,
                            00010,
                            00100,
                            01000,
                            10000,
                        };

    for (i = 0; i < 5; ++i)
        printf("%d ", power_of_ten[i]);
    printf("\n");

    return 0;
}
```

In the above example, we have created an array of 5 elements, whose elements are power of ten (i.e. 1, 10, 100, 1000 and 10,000) and we are printing these element by using a simple for loop. So we are expecting output of above program is " 1 10 100 1000 and 10000", but above program doesn't show this output, instead it shows

```
"1 8 64 512 10000"
```

Let us discuss above program in more detail. Inside array we declared elements which starts with "0", and this is octal representation of decimal number (See this GFact for details).
That's why all these numbers are in octal representation. "010" is octal representation of decimal "8", "0100" is octal representation of decimal "64" and so on.
Last element "10000" which doesn't start with "0", that's why it is in decimal format and it is printed as it is.

Question 2

```
#include <stdio.h>

int main(void)
{

http://geeksforgeeks.org

     printf("Hello, World !!!\n");

     return 0;
}
```

At first sight, it looks like that the above program will give compilation error, but it will work fine (but will give compilation warning). Observe the above program carefully, in this program "http:" will be treated as goto label and two forward slashes will act as single line comment. That's why program will work work correctly.

Question 3

```
#include <stdio.h>

#define ARRAY_SIZE(arr) (sizeof(arr) / sizeof(arr[0]))

int main(void)
{
    int i;
    int arr[] = {1, 2, 3, 4, 5};

    for (i = -1; i < ARRAY_SIZE(arr) - 1; ++i)
        printf("%d ", arr[i + 1]);

    printf("\n");

    return 0;
}
```

In above program, “ARRAY_SIZE” macro substitutes the expression to get the total number of elements present in array, and we are printing these element by using a for loop. But program doesn’t print anything. Let us see what is wrong with code.
Value returned by the substituted expression is in “unsigned int” format, and inside for loop we are comparing unsigned 5 with signed -1. In this comparison -1 is promoted to unsigned integer. -1 in unsigned format is represented as all it’s bit are set to 1 (i.e. 0xffffffff), which is too big number

After substituting value of macro, for loop look like this.

```
for (i = -1; 0xffffffff < 5; ++i)
```

In above loop indeed 0xffffffff is not less than 5, that's why for loop condition fails, and program exits from loop without printing anything.

This article is compiled by **Narendra Kangralkar**. Please write comments if you find anything incorrect, or you want to share more information about the topic discussed above.

46. Output of C Program | Set 28

Predict the output of following C program.

Question 1

```
#include <stdio.h>

int main()
{
    char a = 30;
    char b = 40;
    char c = 10;
    char d = (a * b) / c;
    printf ("%d ", d);

    return 0;
}
```

At first look, the expression (a*b)/c seems to cause arithmetic overflow because signed characters can have values only from -128 to 127 (in most of the C compilers), and the value of subexpression '(a*b)' is 1200. For example, the following code snippet prints -80 on a 32 bit little endian machine.

```
    char d = 1200;
    printf ("%d ", d);
```

Arithmetic overflow doesn't happen in the original program and the output of the program is 120. In C, *char* and *short* are converted to *int* for arithmetic calculations. So in the expression '(a*b)/c', a, b and c are promoted to *int* and no overflow happens.

Question 2

```
#include<stdio.h>
int main()
{
    int a, b = 10;
    a = -b--;
    printf("a = %d, b = %d", a, b);
    return 0;
}
```

Output:

```
a = -10, b = 9
```

The statement 'a = -b–;' compiles fine. Unary minus and unary decrement have save precedence and right to left associativity. Therefore '-b–' is treated as -(b–) which is valid. So -10 will be assigned to 'a', and 'b' will become 9.
Try the following program as an exercise.

```
#include<stdio.h>
int main()
{
    int a, b = 10;
    a = b---;
    printf("a = %d, b = %d", a, b);
    return 0;
}
```

Please write comments if you find anything incorrect, or you want to share more information about the topic discussed above.

47. Output of C++ Program | Set 13

Predict the output of following C++ program.

```
#include<iostream>
using namespace std;

class A
{
    // data members of A
public:
    A ()           { cout << "\n A's constructor"; /* Initialize data 
    A (const A &a) { cout << "\n A's Copy constructor";  /* copy data 
    A& operator= (const A &a) // Assignemt Operator
    {
        // Handle self-assignment:
        if(this == &a) return *this;

        // Copy data members
        cout << "\n A's Assignment Operator";  return *this;
    }
};

class B
{
    A a;
    // Other members of B
public:
    B(A &a) { this->a = a; cout << "\n B's constructor"; }
};

int main()
{
    A a1;
    B b(a1);
    return 0;
}
```

Output:

```
A's constructor
A's constructor
A's Assignment Operator
B's constructor
```

The first line of output is printed by the statement “A a1;” in main().

The second line is printed when B’s member ‘a’ is initialized. This is important.
The third line is printed by the statement “this->a = a;” in B’s constructor.
The fourth line is printed by cout statement in B’s constructor.

If we take a look a closer look at the above code, the constructor of class B is not efficient as member ‘a’ is first constructed with default constructor, and then the values from the parameter are copied using assignment operator. It may be a concern when class A is big, which generally is the case with many practical classes. See the following optimized code.

```
#include<iostream>
using namespace std;

class A
{
    // data members of A
public:
    A()              { cout << "\n A's constructor"; /* Initialize data m
    A(const A &a) { cout << "\n A's Copy constructor"; /* Copy data me
    A& operator= (const A &a) // Assignemt Operator
    {
        // Handle self-assignment:
        if(this == &a) return *this;

        // Copy data members
        cout << "\n A's Assignment Operator";  return *this;
    }
};

class B
{
    A a;
    // Other members of B
public:
    B(A &a):a(a) {  cout << "\n B's constructor"; }
};

int main()
{
    A a;
    B b(a);
    return 0;
}
```

Output:

```
A's constructor
A's Copy constructor
B's constructor
```

The constructor of class B now uses initializer list to initialize its member ‘a’. When Initializer list is used, the member ‘a’ of class B is initialized directly from the parameter. So a call to A’s constructor is reduced.

In general, it is a good idea to use Initializer List to initialize all members of a class, because it saves one extra assignment of members. See point 6 of this post for more details.

Please write comments if you find anything incorrect, or you want to share more information about the topic discussed above

48. Output of C++ Program | Set 14

Predict the output of following C++ program.

Difficulty Level: Rookie

Question 1

```
#include <iostream>
using namespace std;

class A
{
    int id;
public:
    A (int i) { id = i; }
    void print () { cout << id << endl; }
};

int main()
{
    A a[2];
    a[0].print();
    a[1].print();
    return 0;
}
```

There is a compilation error in line “A a[2]”. There is no default constructor in class A. When we write our own parameterzied constructor or copy constructor, compiler doesn’t create the default constructor (See this Gfact). We can fix the error, either by creating a default constructor in class A, or by using the following syntax to initialize array member using parameterzied constructor.

```
// Initialize a[0] with value 10 and a[1] with 20
A a[2] = { A(10),  A(20) }
```

Question 2

```
#include <iostream>
using namespace std;

class A
{
    int id;
    static int count;
public:
    A()
    {
        count++;
        id = count;
        cout << "constructor called " << id << endl;
    }
    ~A()
    {
        cout << "destructor called " << id << endl;
    }
};

int A::count = 0;

int main()
{
    A a[2];
    return 0;
}
```

Output:

```
constructor called 1
constructor called 2
destructor called 2
destructor called 1
```

In the above program, object a[0] is created first, but the object a[1] is destroyed first. Objects are always destroyed in reverse order of their creation. The reason for reverse order is, an object created later may use the previously created object. For example, consider the following code snippet.

```
A a;
B b(a);
```

In the above code, the object ‘b’ (which is created after ‘a’), may use some members of ‘a’ internally. So destruction of ‘a’ before ‘b’ may create problems. Therefore, object ‘b’ must be destroyed before ‘a’.

Question 3

```
#include <iostream>
using namespace std;

class A
{
   int aid;
public:
   A(int x)
   { aid = x; }
   void print()
   { cout << "A::aid = " <<aid; }
};

class B
{
    int bid;
public:
    static A a;
    B (int i) { bid = i; }
};

int main()
{
  B b(10);
  b.a.print();
  return 0;
}
```

Compiler Error: undefined reference to `B::a’
The class B has a static member ‘a’. Since member ‘a’ is static, it must be defined outside the class. Class A doesn’t have Default constructor, so we must pass a value in definition also. Adding a line “A B::a(10);” will make the program work.

The following program works fine and produces the output as “A::aid = 10″

```
#include <iostream>
using namespace std;

class A
{
   int aid;
public:
   A(int x)
   { aid = x; }
   void print()
   { cout << "A::aid = " <<aid; }
};

class B
{
    int bid;
public:
    static A a;
    B (int i) { bid = i; }
};

A B::a(10);

int main()
{
  B b(10);
  b.a.print();
  return 0;
}
```

Please write comments if you find anything incorrect, or you want to share more information about the topic discussed above

49. Output of C++ Program | Set 15

Predict the output of following C++ programs.

Question 1

```
#include <iostream>
using namespace std;

class A
{
public:
    void print() { cout << "A::print()"; }
};

class B : private A
{
public:
    void print() { cout << "B::print()"; }
};

class C : public B
{
public:
    void print() { A::print(); }
};

int main()
{
    C b;
    b.print();
}
```

Output: Compiler Error: ‘A’ is not an accessible base of ‘C’
There is multilevel inheritance in the above code. Note the access specifier in “class B : private A”. Since private access specifier is used, all members of ‘A’ become private in ‘B’. Class ‘C’ is a inherited class of ‘B’. An inherited class can not access private data members of the parent class, but print() of ‘C’ tries to access private member, that is why we get the error.

Question 2

```
#include<iostream>
using namespace std;

class base
{
public:
    virtual void show()  { cout<<" In Base \n"; }
};

class derived: public base
{
    int x;
public:
    void show() { cout<<"In derived \n"; }
    derived()   { x = 10; }
    int getX() const { return x;}
};

int main()
{
    derived d;
    base *bp = &d;
    bp->show();
    cout << bp->getX();
    return 0;
}
```

Output: Compiler Error: ‘class base’ has no member named ‘getX’
In the above program, there is pointer ‘bp’ of type ‘base’ which points to an object of type derived. The call of show() through ‘bp’ is fine because ‘show()’ is present in base class. In fact, it calls the derived class ‘show()’ because ‘show()’ is virtual in base class. But the call to ‘getX()’ is invalid, because getX() is not present in base class. When a base class pointer points to a derived class object, it can access only those methods of derived class which are present in base class and are virtual.

Question 3

```
#include<iostream>
using namespace std;

class Test
{
    int value;
public:
    Test(int v = 0) { value = v; }
    int getValue()  { return value; }
};

int main()
{
    const Test t;
    cout << t.getValue();
    return 0;
}
```

Output: Compiler Error

In the above program, object ‘t’ is declared as a const object. A const object can only call const functions. To fix the error, we must make getValue() a const function.

Please write comments if you find anything incorrect, or you want to share more information about the topic discussed above

50. Output of C++ Program | Set 16

Predict the output of following C++ programs.

Question 1

```
#include<iostream>
using namespace std;

class Base
{
public:
    int fun()      { cout << "Base::fun() called"; }
    int fun(int i) { cout << "Base::fun(int i) called"; }
};

class Derived: public Base
{
public:
    int fun(char x)   { cout << "Derived::fun(char ) called"; }
};

int main()
{
    Derived d;
    d.fun();
    return 0;
}
```

Output: Compiler Error.
In the above program, fun() of base class is not accessible in the derived class. If a derived class creates a member method with name same as one of the methods in base class, then all the base class methods with this name become hidden in derived class (See this for more details)

Question 2

```
#include<iostream>
using namespace std;
class Base
{
   protected:
      int x;
   public:
      Base (int i){ x = i;}
};

class Derived : public Base
{
   public:
      Derived (int i):x(i) { }
      void print() { cout << x ; }
};

int main()
{
    Derived d(10);
    d.print();
}
```

Output: Compiler Error

In the above program, x is protected, so it is accessible in derived class. Derived class constructor tries to use initializer list to directly initialize x, which is not allowed even if x is accessible. The members of base class can only be initialized through a constructor call of base class. Following is the corrected program.

```
#include<iostream>
using namespace std;
class Base {
   protected:
      int x;
   public:
      Base (int i){ x = i;}
};

class Derived : public Base {
   public:
      Derived (int i):Base(i) { }
      void print() { cout << x; }
};

int main()
{
    Derived d(10);
    d.print();
}
```

Output:

```
10
```

Please write comments if you find anything incorrect, or you want to share more information about the topic discussed above

51. Output of C++ Program | Set 17

Predict the output of following C++ programs.

Question 1

```
#include <iostream>
using namespace std;

class A
{
    public:
    A& operator=(const A&a)
    {
        cout << "A's assignment operator called" << endl;
        return *this;
    }
};

class B
{
    A a[2];
};

int main()
{
    B b1, b2;
    b1 = b2;
    return 0;
}
```

Output:

```
A's assignment operator called
A's assignment operator called
```

The class B doesn’t have user defined assignment operator. If we don’t write our own assignment operator, compiler creates a default assignment operator. The default assignment operator one by one copies all members of right side object to left side object. The class B has 2 members of class A. They both are copied in statement “b1 = b2″, that is why there are two assignment operator calls.

Question 2

```
#include<stdlib.h>
#include<iostream>

using namespace std;

class Test {
public:
    void* operator new(size_t size);
    void operator delete(void*);
    Test() { cout<<"\n Constructor called"; }
    ~Test() { cout<<"\n Destructor called"; }
};

void* Test::operator new(size_t size)
{
    cout<<"\n new called";
    void *storage = malloc(size);
    return storage;
}

void Test::operator delete(void *p )
{
    cout<<"\n delete called";
    free(p);
}

int main()
{
    Test *m = new Test();
    delete m;
    return 0;
}
```

```
new called
Constructor called
Destructor called
delete called
```

Let us see what happens when below statement is executed.

```
    Test *x = new Test;
```

When we use new keyword to dynamically allocate memory, two things happen: memory allocation and constructor call. The memory allocation happens with the help of operator new. In the above program, there is a user defined operator new, so first user defined operator new is called, then constructor is called.
The process of destruction is opposite. First, destructor is called, then memory is deallocated.

Please write comments if you find anything incorrect, or you want to share more information about the topic discussed above

52. Output of C++ Program | Set 18

Predict the output of following C++ programs.

Question 1

```cpp
#include <iostream>
using namespace std;

template <int N>
class A {
   int arr[N];
public:
   virtual void fun() { cout << "A::fun()"; }
};

class B : public A<2> {
public:
   void fun() { cout << "B::fun()"; }
};

class C : public B { };

int main() {
   A<2> *a = new C;
   a->fun();
   return 0;
}
```

Output:

```
B::fun()
```

In general, the purpose of using templates in C++ is to avoid code redundancy. We create generic classes (or functions) that can be used for any datatype as long as logic is identical. Datatype becomes a parameter and an instance of class/function is created at compile time when a data type is passed. C++ Templates also allow nontype (a parameter that represents a value, not a datatype) things as parameters.
In the above program, there is a generic class A which takes a nontype parameter N. The class B inherits from an instance of generic class A. The value of N for this instance of A is 2. The class B overrides fun() of class A<2>. The class C inherits from B. In main(), there is a pointer ‘a’ of type A<2> that points to an instance of C. When ‘a->fun()’ is called, the function of class B is executed because fun() is virtual and virtual functions are called according to the actual object, not according to pointer. In class C, there is no function ‘fun()’, so it is looked up in the hierarchy and found in class B.

Question 2

```
#include <iostream>
using namespace std;

template <int i>
int fun()
{
   i =20;
}

int main() {
   fun<4>();
   return 0;
}
```

Output:

```
Compiler Error
```

The value of nontype parameters must be constant as they are used at compile time to create instance of classes/functions. In the above program, templated fun() receives a nontype parameter and tries to modify it which is not possible. Therefore, compiler error.

Please write comments if you find anything incorrect, or you want to share more information about the topic discussed above

53. Output of Java Program | Set 3

Predict the output of following Java Programs:

```
// filename: Test.java
class Test {
    int x  = 10;
    public static void main(String[] args) {
         Test t = new Test();
         System.out.println(t.x);
    }
}
```

The program works fine and prints 10. Unlike C++, in Java, members can initialized with declaration of class. This initialization works well when the initialization value is available and the initialization can be put on one line (See this for more details). For example, the following program also works fine.

```
// filename: Test.java
class Test {
    int y = 2;
    int x  = y+2;
    public static void main(String[] args) {
         Test m = new Test();
         System.out.println("x = " + m.x + ", y = " + m.y);
    }
}
```

Output of the above program is “x = 4, y = 2″. y is initialized first, then x is initialized as y + 2. So the value of x becomes 4.

What happen when a member is initialized in class declaration and constructor both? Consider the following program.

```
// filename: Test.java
public class Test
{
    int x = 2;
    Test(int i) { x = i; }
    public static void main(String[] args) {
         Test t = new Test(5);
         System.out.println("x = " + t.x);
    }
}
```

Output of the above program is “x = 5″. The initialization with class declaration in Java is like initialization using Initializer List in C++. So, in the above program, the value assigned inside the constructor overwrites the previous value of x which is 2, and x becomes 5.

As an exercise, predict the output of following program.

```
// filename: Test2.java
class Test1 {
    Test1(int x) {
        System.out.println("Constructor called " + x);
    }
}

// This class contains an instance of Test1
class Test2 {
    Test1 t1 = new Test1(10);

    Test2(int i) { t1 = new Test1(i); }

    public static void main(String[] args) {
         Test2 t2 = new Test2(5);
    }
}
```

Please write comments if you find any of the answers/explanations incorrect, or want to share more information about the topics discussed above.

54. Output of Java Program | Set 4

Predict the output of following Java Programs:

Question 1

```
// file name: Main.java

class Base {
    protected void foo() {}
}
class Derived extends Base {
    void foo() {}
}
public class Main {
    public static void main(String args[]) {
        Derived d = new Derived();
        d.foo();
    }
}
```

Output: Compiler Error
foo() is protected in Base and default in Derived. Default access is more restrictive. When a derived class overrides a base class function, more restrictive access can't be given to the overridden function. If we make foo() public, then the program works fine without any error. The behavior in C++ is different. C++ allows to give more restrictive access to derived class methods.

Question 2

```
// file name: Main.java

class Complex {
    private double re, im;
    public String toString() {
        return "(" + re + " + " + im + "i)";
    }
    Complex(Complex c) {
        re = c.re;
        im = c.im;
    }
}

public class Main {
    public static void main(String[] args) {
        Complex c1 = new Complex();
        Complex c2 = new Complex(c1);
        System.out.println(c2);
    }
}
```

Output: Compiler Error in line "Complex c1 = new Complex();"
In Java, if we write our own copy constructor or parameterized constructor, then compiler doesn't create the default constructor. This behavior is same as C++.

Please write comments if you find anything incorrect, or you want to share more information about the topic discussed above.

55. Output of C++ Program | Set 19

Predict the output of following C++ programs.

Question 1

```
#include <iostream>
#include <string.h>
using namespace std;

int main()
{
    cout << sizeof("GeeksforGeeks") << endl;
    cout << strlen("GeeksforGeeks");
    return 0;
}
```

Output:

```
14
13
```

Sizeof operator returns the size of string including null character so output is 14. While strlen() function returns the exact length of string excluding null character so output is 13.

Question 2:

```
#include <iostream>
using std::cout;
class Test
{
public:
    Test();
    ~Test();
};
Test::Test()
{
    cout << "Constructor is executed\n";
}
Test::~Test()
{
    cout << "Destructor is executed\n";
}
int main()
{
    delete new Test();
    return 0;
}
```

Output:

```
Constructor is executed
Destructor is executed
```

The first statement inside the main () function looks strange, but it is perfectly valid. It is possible to create an object without giving its handle to any pointer in C++. This statement will create an object of class Test without any pointer pointing to it. This can be also done in languages like Java & C#.
For example consider following statement:

```
new  student();  // valid both in Java & C#
```

The above statement will create an object of student class without any reference pointing to it.

Question 3:

```
#include <iostream>
using std::cout;
class main
{
public:
    main()  {cout << "ctor is called\n";}
    ~main() {cout << "dtor is called\n";}
};
int main()
{
    main m;    // LINE 11
}
```

Output:

```
Compiler error:
11 8 [Error] expected ';' before 'm'
```

The above program looks syntactically correct but it fails in compilation. The reason class name. Class name is main so it is necessary to tell the compiler that main is the name of class. Generally struct or class keyword is not required to write to create an object of the class or struct. But when the name of class is main it becomes necessary to write struct or class when creating object of class or struct. Remember main is not a reserved word.

Following is a correct version of the above program:

```
#include <iostream>
using std::cout;
class main
{
public:
   main()  { cout << "ctor is called\n";}
   ~main() { cout << "dtor is called\n";}
};
int main()
{
   class main m;
}
```

Now predict the output of following program:

```
#include <iostream>
using std::cout;
class main
{
public:
   main()  { cout << "ctor is called\n"; }
   ~main() { cout << "dtor is called\n"; }
};
main m;    // Global object
int main()
{
}
```

The above program compiles and runs fine because object is global. Global object‘s constructor executes before main() function and it’s destructor executes when main() terminates.

Conclusion: When the class/struct name is main and whenever the local object is created it is mandatory to write class or struct when the object of class / and struct is created. Because C++ program execution begins from main () function. But this rule is not applied to global objects. Again, main isn’t a keyword but treat it as if it were.

This article is contributed **Meet Pravasi**. Please write comments if you find anything incorrect, or you want to share more information about the topic discussed above