STORAGE CLASSES AND TYPES

Solutions

1. Consider the following C function, what is the output?

```
int f(int n)
{
   static int r = 0;
   if (n <= 0) return 1;
   if (n > 3)
   {
      r = n;
      return f(n-2)+2;
    }
   return f(n-1)+r;
}

int main()
{
   printf("%d", f(5));
}
```

(a) 5
(b) 7
(c) 9
(d) 18

Solution: Option (d)

2. Which of the following is not a storage class specifier in C?

(a) auto
(b) register
(c) static
(d) volatile

Solution: Option (d)

Explanation:
Volatile is not a storage class specifier. Volatile and const are type qualifiers.

3. Output of following program?

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

(a) 4 3 2 1
(b) 1 2 3 4
(c) 0 0 0 0
(d) Compiler Error

Solution: Option (c)

Explanation:
A static variable is shared among all calls of a function. All calls to main() in the given program share the same i. i becomes 0 before the printf() statement in all calls to main().

4.

```
#include <stdio.h>

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

(a) 4 3 2 1
(b) 1 2 3 4
(c) 4 4 4 4
(d) 0 0 0 0

Solution: Option (a)

Explanation:
Since i is static variable, it is shared among all calls to main(). So is reduced by 1 by every function call.

5.

```
int main()
{
   int x = 5;
   int * const ptr = &x;
   ++(*ptr);
   printf("%d", x);
   return 0;
}
```

(a) Compiler Error
(b) Runtime Error
(c) 6
(d) 5

Solution: Option (c)

Explanation:
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 cannot 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 incrementing value pointed by ptr.

6.

```
#include<stdio.h>

int main()
{
  typedef static int *i;
  int j;
  i a = &j;
  printf("%d", *a);
```

```
  return 0;
}
```

(a) Runtime Error
(b) 0
(c) Garbage Value
(d) Compiler Error

Solution: Option (d)

Explanation:
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.

7. Output?

```
#include<stdio.h>

int main()
{
  typedef int i;
  i a = 0;
  printf("%d", a);
  return 0;
}
```

(a) Compiler Error
(b) Runtime Error
(c) 0
(d) 1

Solution: Option (c)

Explanation:

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

8.

```
#include<stdio.h>

int main()
```

```
{
  typedef int *i;
  int j = 10;
  i *a = &j;
  printf("%d", **a);
  return 0;
}
```

(a) Compiler Error
(b) Garbage Value
(c) 10
(d) 0

Solution: Option (a)

Explanation:

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.

9. Output?

```
#include<stdio.h>

int fun()
{
  static int num = 16;
  return num--;
}

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

(a) Infinite loop
(b) 13 10 7 4 1
(c) 14 11 8 5 2
(d) 15 12 8 5 2

Solution: Option (c)

Explanation:

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.

10.

```
#include<stdio.h>

int main()
{
  int x = 10;
  static int y = x;
  if(x == y)
     printf("Equal");
  else if(x > y)
     printf("Greater");
  else
     printf("Less");
  return 0;
}
```

(a) Compiler Error
(b) Equal
(c) Greater
(d) Less

Solution: Option (a)

Explanation:
In C, static variables can only be initialized using constant literals. This is allowed in C++ though.

11. Consider the following C function

```
int f(int n)
{
   static int i = 1;
   if (n >= 5)
      return n;
```

```
  n = n+i;
  i++;
  return f(n);
}
```

The value returned by f(1) is (GATE CS 2004)

(a) 5 (b) 6
(c) 7 (d) 8

Solution: Option (c)

Explanation:
Since i is static, first line of f() is executed only once.

Execution of f(1)
  i = 1
  n = 2
  i = 2

Call f(2)
  i = 2
  n = 4
  i = 3

Call f(4)
  i = 3
  n = 7
  i = 4

Call f(7)
  since n >= 5 return n(7)

12. In C, static storage class cannot be used with:

(a) Global variable (b) Function parameter
(c) Function name (d) Local variable

Solution: Option (b)

Explanation:
Declaring a global variable as static limits its scope to the same file in which it is defined.

A static function is only accessible to the same file in which it is defined.

A local variable declared as static preserves the value of the variable between the function calls.

13. Output? (GATE CS 2012)

```
#include<stdio.h>

int a, b, c = 0;
void prtFun (void);
int main ()
{
   static int a = 1; /* line 1 */
   prtFun();
   a += 1;
   prtFun();
   printf ( "\n %d %d " , a, b) ;
}

void prtFun (void)
{
   static int a = 2; /* line 2 */
   int b = 1;
   a += ++b;
   printf (" \n %d %d " , a, b);
}
```

(a) 31
41
42

(b) 42
61
61

(c) 42
62
20

(d) 31
52
52

Solution: Option (c)

Explanation:
'a' and 'b' are global variable. prtFun() also has 'a' and 'b' as local variables. The local variables hide the globals (See Scope rules in C). When prtFun() is called first time, the local 'b' becomes

2 and local ‘a’ becomes 4.

When prtFun() is called second time, same instance of local static ‘a’ is used and a new instance of ‘b’ is created because ‘a’ is static and ‘b’ is non-static. So ‘b’ becomes 2 again and ‘a’ becomes 6.

main() also has its own local static variable named ‘a’ that hides the global ‘a’ in main. The printf() statement in main() accesses the local ‘a’ and prints its value. The same printf() statement accesses the global ‘b’ as there is no local variable named ‘b’ in main. Also, the defaut value of static and global int variables is 0. That is why the printf statement in main() prints 0 as value of b.

14. What output will be generated by the given code segment if:

Line 1 is replaced by “auto int a = 1;”
Line 2 is replaced by “register int a = 2;” (GATE CS 2012)

(a) 31
41
42

(b) 42
61
61

(c) 42
62
20

(d) 42
42
20

Solution: Option (d)

Explanation:
If we replace line 1 by “auto int a = 1;” and line 2 by “register int a = 2;”, then ‘a’ becomes non-static in prtFun(). The output of first prtFun() remains same. But, the output of second prtFun() call is changed as a new instance of ‘a’ is created in second call. So “4 2″ is printed again. Finally, the printf() in main will print “2 0″. Making ‘a’ a register variable won’t change anything in output.

15. Output?

#include<stdio.h>

```
int main()
{
  register int i = 10;
  int *ptr = &i;
  printf("%d", *ptr);
  return 0;
}
```

(a) Prints 10 on all compilers
(b) May generate compiler Error
(c) Prints 0 on all compilers
(d) May generate runtime Error

Solution: Option (b)

Explanation :

The register variable may or may not be assigned a register in the computers CPU. But asking for the address of it will generate compilation errors.

16.

```
#include<stdio.h>

int main()
{
  extern int i;
  printf("%d ", i);
   {
      int i = 10;
      printf("%d ", i);
    }
}
```

(a) 0 10
(b) Compiler Error
(c) 0 0
(d) 10 10

Solution: Option (b)

17. Output?

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

```
int main(void)
{
   int i = 10;
   const int *ptr = &i;
   *ptr = 100;
   printf("i = %d\n", i);
   return 0;
  }
```

(a)) i = 100
(b) i = 10
(c) Compiler Error
(d) Runtime Error

Solution: Option (c)

Explanation:
Note that ptr is a pointer to a constant. So value pointed cannot be changed using the pointer ptr.

18. Output of following program

```
#include <stdio.h>

int fun(int n)
{
   static int s = 0;
   s = s + n;
   return (s);
}

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

(a) 0
(b) 100
(c) 110
(d) 55

Solution: Option (d)

Explanation:
Since s is static, different values of i are added to it one by one.

So final value of s is
s = i + (i-1) + (i-2) + … 3 + 2 + 1.

The value of s is i*(i+1)/2. For i = 10, s is 55.

19.

```
#include <stdio.h>

char *fun()
{
   static char arr[1024];
   return arr;
}

int main()
{
   char *str = "ravindrababus";
   strcpy(fun(), str);
   str = fun();
   strcpy(str, "gatessquiz");
   printf("%s", fun());
   return 0;
}
```

(a) ravindrababus
(b) gatesquiz
(c) ravindrababus gatesquiz
(d) Compiler Error

Solution: Option (b)

Explanation:
Note that arr[] is static in fun() so no problems of returning address, arr[] will stay there even after the fun() returns and all calls to fun() share the same arr[].

```
strcpy(fun(), str);  // Copies " ravindrababus " to arr[]

str = fun();    // Assigns address of arr to str

strcpy(str, "gatesquiz");  // copies gatesquiz to str which is address of arr[]

printf("%s", fun());   // prints "gatesquiz"
```

20.

```
#include <stdio.h>

int main()
{
   int i = 1024;
   for (; i; i >>= 1)
      printf("GatesQuiz");
   return 0;
}
```

How many times will GatesQuiz be printed in the above program?

(a) 10
(b) 11
(c) Infinite

(d) The program will show compile-time error

Solution: Option (b)

Explanation:
In for loop, mentioning expression is optional. >>= is a composite operator. It shifts the binary representation of the value by 1 to the right and assigns the resulting value to the same variable. The for loop is executed until value of variable i doesn't drop to 0.