ASSIGNMENT-12 SOLUTION

1. Which of the following are themselves a collection of different data types?
   a) String
   b) Array
   c) Character
   d) Structure

Solution: (d) Structure

Structure is a user defined data type available in C that allows combining data items of different kinds.

2. What is the output?

```
#include<stdio.h>
int main()
{
struct xyz{ int a;};
struct xyz obj1={1};
struct xyz obj2 = obj1;
printf("%d", obj2.a);
obj2.a = 100;
printf("%d", obj1.a);
printf("%d", obj2.a);
return 0;
}
```

a) 11100
b) 11
c) 11001
d) 11000

Solution: (a) 11100

3. Which of the following comment about the usage structures in true?
   a) Storage class can be assigned to individual member
   b) Individual members can be initialized within a structure type declaration
   c) The scope of the member name is confined to the particular structure, within which it is defined
   d) None

Solution: (c) The scope of the member name is confined to the particular structure, within which it is defined

4. Assume sizeof an integer and a pointer is 4 byte. Output of the following program?

```
#include <stdio.h>
#define R 5
#define C 4

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

a) 80
b) 4
c) 20
d) 18

Solution: (a) 80
Output is 5*4*sizeof(int) which is “80″ for compilers with integer size as 4 bytes.When a pointer is de-referenced using *, it yields type of the object being pointed. In the present case, it is an array of array of integers. So, it prints R*C*sizeof(int).

5. What will be output?

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

int main(void)
{
   int i;
   int *ptr = (int *) malloc(5 * sizeof(int));

   for (i=0; i<5; i++)
      *(ptr + i) = i;

   printf("%d ", *ptr++);
   printf("%d ", (*ptr)++);
   printf("%d ", *ptr);
   printf("%d ", *++ptr);
   printf("%d ", ++*ptr);
   return 0;
}
```

a) Compiler error
b) 0 1 2 3 4
c) 1 2 3 4 5
d) 0 1 2 2 3

Solution: (d) 0 1 2 2 3

The important things to remember for handling such questions are--

Prefix ++ and * operators have same precedence and right to left associativity. Postfix ++ has higher precedence than the above two mentioned operators and associativity is from left to right.
We can apply the above two rules to guess all

*ptr++ is treated as *(ptr++)
*++ptr is treated as *(++ptr)

++*ptr is treated as ++(*ptr)

6. What will be output?

```
#include <stdio.h>
int fun(int arr[]) {
  arr = arr+1;
  printf("%d ", arr[0]);
}
int main(void) {
  int arr[3] = {5, 10, 15};
  fun(arr);
  printf("%d ", arr[0]);
  printf("%d ", arr[1]);
  return 0;
}
```

a) 5 10 10
b) 10 5 15
c) 10 5 10
d) 10 15 5

Solution: (c) 10 5 10

In C, array parameters are treated as pointers So the variable *arr* represents an array in main(), but a pointer in fun().

7. What is the output of the following C code? Assume that the address of x is 2000 (in decimal) and an integer requires four bytes of memory

```
#include <stdio.h>
int main()
{
  unsigned int x[4][3] = {{1, 2, 3}, {4, 5, 6}, {7, 8, 9}, {10, 11, 12}};
  printf("%u, %u, %u", x+3, *(x+3), *(x+2)+3);
  return 0;
}
```

a) 2036 2036 2036
b) 2012 4 2204
c) 2036 10 10
d) 2012 4 6

Solution: (a) 2036 2036 2036

x = 2000

Since x is considered as a pointer to an array of 3 integers and an integer takes 4 bytes, value of x + 3 = 2000 + 3*3*4 = 2036

The expression, *(x + 3) also prints same address as x is 2D array.
The expression *(x + 2) + 3 = 2000 + 2*3*4 + 3*4 = 2036

8. In which condition “Live long and Prosper” will be printed?

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

  int main()
  {
     int *ptr;
     ptr = (int *)malloc(sizeof(int)*10);
     if (ptr == NULL)
```

```
        printf("Live long and Prosper\n");
    return 0;
}
```

a) if the memory has been allocated to the pointer “ptr” successfully
b) if the memory could not be allocated to the pointer “ptr”
c) it will never print
d) None of the above

Solution: (b) if the memory could not be allocated to the pointer “ptr”

The malloc() returns NULL when the memory is not allocated.

9. The program will allocate ……...... bytes to ptr. Assume sizeof(int)=4.

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

  int main()
  {
      int *ptr;
      ptr = (int*)malloc(sizeof(int)*4);
      ptr = realloc(ptr,sizeof(int)*2);
      return 0;
  }
```

a) 2
b) 4
c) 8
d) None of the above

Solution: (c) 8

We can also use the realloc() to change memory block size.

10. This question has been deleted and reevaluated.

11. What does fp point to in the program?

```
#include<stdio.h>
int main()
{
    FILE *fp;
    fp=fopen("hello", "r");
    return 0;
}
```

a) The first character in the file
b) A structure which contains a char pointer which points to the first character of a file.
c) The name of the file.
d) The last character in the file

Solution: (b) The fp is a structure which contains a char pointer which points to the first character of a file.

12. What is the output of the following C program?

```
#include <stdio.h>
int main()
{
int *p,a=10;
p=&10;
```

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

a) 10
b) a
c) address of a
d) compilation error

Solution: (d) A pointer variable can be assigned as the address of any constant. Thus, the compiler will show error as “[Error] lvalue required as unary '&' operand”.

13. What is the output of the following C program?

```
#include <stdio.h>
struct p
{
   int x;
   char y;
};

int main()
{
   struct p p1[] = {1, 90, 62, 33, 3, 34};
   struct p *ptr1 = p1;
   int x = (sizeof(p1) / 3);
   if (x == sizeof(int) + sizeof(char))
      printf("True");
   else
      printf("False");
   return 0;
}
```

a) True
b) False
c) No output
d) Compilation error

Solution: (b) Size of the structure is the maximum size of the variable inside structure. Thus, the size of each element of structure p is 4 bytes (in gcc compiler, it can vary based on compiler). Thus, sizeof(p1) is 6*4=24. x will be 24/3=8. In the next step, sizeof(int)+sizeof(char) is 5 which is not equal to x. Hence, false will be printed.

14. What will be the output?

```
#include <stdio.h>
int main()
{
   int ary[4] = {1, 2, 3, 4};
   int *p;
   p = ary + 3;
   *p = 5;
   printf("%d\n", ary[3]);
   return 0;
}
```

a) 4
b) 3

c) 5
d) None of the above

Solution: (c) Array elements are stored in contiguous memory locations. Thus, increasing the address by 3, the pointer p is pointing to the variable ary[3]. Hence, storing 5 in that location will replace ary[3] with 5.
.

15. Calling a function f with a an array variable a[3] where a is an array, is equivalent to
a) f(a[3])
b) f(*(a + 3))
c) f(3[a])
d) all of the mentioned

Solution: (d) all the methods are correct.