Week 12 Assignment 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. Can we declare function inside structure of C Programming?
    a) Yes
    b) No
    c) Compiler dependant
    d) Its possible, but causes runtime errors.

Sol: (b) No. It's not allowed in C. C++ allows it.

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

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

int main()
{
   struct p p1[] = {1,21,69,42,64};
   struct p *ptr1 = p1;
   int x = (sizeof(p1) / 4);
   if ((x == sizeof(int) + 2*sizeof(char)))
      printf("True");
   else
      printf("False");
   return 0;
}
```

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

Solution: (a) True

Due to padding operations of structures the sizeof struct p1 is 24.
The reason is as follows:
The memory assignment of struct p is as follows:

int 1st byte	int 2nd byte	int 3rd byte	int 4th byte	char

To store the second element of p1 i.e. 21, 3 bytes are padded, which makes it 8 bytes.
While storing the 3rd element, the memory gets allocated for 8*2=16 bytes as shown below.

1st element				2nd element		
3rd element				Blank spaces		

Finally the memory structure of p1 will look like this

1st element				2nd element		
3rd element				4th element		
5th element				Blank spaces		

In the program, x = 24/4 = 6. And sizeof(int) +2* sizeof(char) is also 6. Therefore, the TRUE is printed.

4. Which of the following statements is true about the equality of two structure variables?
   a) Two structure variables are equal if all their members are equal
   b) Two structure variables are equal if they have the same address
   c) Two structure variables cannot be compared for equality
   d) None of the above

Answer: a) Two structure variables are equal if all their members are equal

Explanation: Two structure variables are considered equal if all their members have the same values.

5. What will be output?

```
#include<stdio.h>
int main()
{
        struct insti
        {
                int x = 2;
                char ins[] = "IIT";
        };
        struct insti s1;
        printf("%d",s1.ins);
        printf("%d", s1.x);
        return 0;
}
```

a) IIT

b) 2

c) IIT 2

d) Compilation error

Solution: (d) Error

When we declared members in structure, it just tells the compiler about their presence. There is no memory allocated for that members. So we can't initialize structure members.

6. What is the size of the following structure in bytes?

```
struct example {
   char c;
   int i;
   double d;
};
```

a) 7

b) 12

c) 16

d) 20

Answer: c) 16

Explanation: The size of a structure is the sum of the sizes of its members. In this case, the size of char is 1 byte, the size of int is 4 bytes, and the size of double is 8 bytes. Therefore, the total size of the structure is 1 + 4 + 8 = 13 bytes. However, due to memory alignment, the size is padded to the next multiple of the largest member, which is 8 bytes. So, the final size of the structure is 16 bytes.

7. 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().

8. 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);
}
```

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

9. What is the output of the following code?

```
#include <stdio.h>
int main() {
   char *str = "Hello, world!";
   char *ptr = str;
   printf("%s\n", ptr + 7);
   return 0;
}
```

a) Hello, world!
b) Hello!
c) world
d) world!

Solution: (d) world!

The program defines a pointer `str` to a string literal "Hello, world!" and assigns its address to another pointer `ptr`. The `printf()` function is used to print the string starting from the

character at index 7 (counting from 0) of the string pointed to by `ptr`. Since `ptr` points to the beginning of the string "Hello, world!", adding 7 to `ptr` gives a pointer to the character 'w', so the output is "world!".

10. Which of the following is a disadvantage of the Trapezoidal Rule?

a) It is computationally expensive

b) It is only accurate for certain types of functions

c) It can produce inaccurate results for functions with rapidly changing curvature

d) It requires a large number of subintervals to achieve high accuracy

Answer: c) It can produce inaccurate results for functions with rapidly changing curvature

Explanation: The Trapezoidal Rule can be accurate for functions that are relatively smooth and do not change curvature rapidly over the interval of integration. However, for functions that have rapidly changing curvature, such as those with sharp peaks or valleys, the trapezoids may not provide a good approximation of the area under the curve, leading to inaccurate results. In such cases, other numerical integration methods, such as Simpson's Rule, may be more appropriate.