CHAPTER 9
ARRAYS

1. O/p?

```
main()
{
 char a[]="Visual C++"
 char *b= "Visual C++"
 printf("\n %d %d", sizeof(a), sizeof(b));
 printf("\n %d %d", sizeof(*a), sizeof(*b));
}
```

Ans:

```
  11  2
  1    1
```

2. For the following statements would arr[3] and ptr[3] fetch the same character?

```
char arr[]="Surprised"
char *ptr="Surprised"
```

Ans:YES.

3. For the statements in Q2 does the compiler fetch the character arr[3] and ptr [3] in the same manner?

Ans: NO.

For arr[3] the compiler generates code to start at location arr, move three past it, and fetch the character there. When it sees the expression ptr[3] it generates to start at location stored in ptr, add three to the pointer, and and finally fetch the character pointed to.

In other words, arr[3] is three places past the start of the object named arr, whereas ptr[3] is three places past the object pointed to by ptr.

4. What would be the o/p of the following program, if the array begins at address 1200?

```
main()
{
 int  arr[]={2, 3, 4, 1, 6};
 printf("%d  %d", arr, sizeof(arr));
}
```

Ans: 1200 10

5. Does mentioning the array name gives the base address in all the contexts?

Ans: NO.
Whenever mentioning the array name gives its base address it is said that the array has decayed in to a pointer. This decaying does take place in two situations:

- When array name is used with sizeof operator
- When the array name is an operand of the & operator

6. What would be the O/p of the following program, if the array begind at address 65486?

```
main()
{
 int arr[]={12, 14, 15, 23, 45};
 printf("%u  %u",arr, &arr);
}
```

Ans: 65486 65486

7. Are the expressions arr and &arr same for an array of 10 integers?

Ans: NO.
Even though both may give the same address as in Q6 above they mean two different things. arr gives the address of the first integer, whereas &arr gives the address of array of ints. Since these address happen to be same the result of the expressions are same.

8. What be the output of the following program if the array begins at 65486?

```
main()
{
 int arr[]= {12, 14, 15, 23, 45}
 printf("%u  %u", arr+1, &arr+1);
}
```

Ans: 65488 65496

9.   Wher are char a[] and char *a treated as same by the compiler?

Ans: When using them as formal parameters while defining a function.

10. Would the following program compile successfully?

```
main()
{
 char a[]= "Sunstroke";
 char *p= "Coldwave";
 a="Coldwave";
 p="Sunstroke";
 printf("\n %s  %s", a, p);
}
```

Ans: NO. Because we may assign a new string to a pointer but not to an array.

11. O/p?

```
main()
{
  float a[]= { 12.4, 2.3, 4.5, 6.7};
  printf("\n %d", sizeof(a)/sizeof(a[0]));
}
```

Ans: 4

12. A pointer to a block of memory is effectively same as an array.

Ans: TRUE.

13. What would be the o/p of the following program if the array begins at 65472?

```
main()
{
 int a[3][4] = {
                    1, 2, 3, 4,
                    4, 3, 2, 1,
                    7, 8, 9, 0
                 };
 printf("\n %u  %u", a+1, %a+1);
}
```

Ans: 65480 65496

14. What does the following declaration mean

```
int (*ptr)[10];
```

Ans: ptr is a pointer to an array of 10 integers.

15. If we pass the mane of a 1-D array to a function it decays into a pointer to an int. If we pass the name of 2-D array of integers to a function what would it decay into?

Ans: It decays into a pointer to an array and not a pointer to a pointer.

16. How would define the function f() in the following program?

```
int  arr[MAXROW][MAXCOL};
fun(arr);
```

Ans:

```
 fun ( int a[][MAXCOL])
{
}
or
fun (int (*ptr)[MAXCOL])  /* ptr is pointer to an array*/
{
}
```

17. O/p?

```
main()
{
   int a[3][4] = {
                        1, 2, 3, 4,
                        4, 3, 2, 8,
                        7, 8, 9, 0
                   };
   int *ptr;
   ptr = &a[0][0];
   fun (&ptr);
}

fun (int **p)
{
  printf("\n%d",**p);
}
```

Ans: 1