CHAP 3
EXPRESSIONS.

1. O/p?

```
main()
{
 static int a[20];
int x=0;
a[x]=x++;
printf("\n %d  %d  %d", a[0],a[1],x);
}
```

ans: 0 0 1
This is what some compliers give others may give a different answer. The reason is that the same statement causes the same object to be modified or to be modified and then inspected, the behavior is undefined.

2. O/p?

```
Main()
{
  int x=3;
  x=x++;
  printf("%d",x);
}
```

ans: 4 but basically the behavior is undefined because of the same reason as above.

3. the expressions on the right hand side of the && and || operator does not get evaluated if the left hand side determines the output.

Ans: True. It is called short circuited mode of evaluating the logical expressions.

4. O/p?

```
Main()
{
int x=2;
printf("%d  %d",++x,++x);
}
```

a 3 4
b 4 3
c 4 4
d o/p may vary from complier to complier

ans:d the order of evaluation of the arguments to a function call is unspecified.

5. O/p?

```
Main()
{
  int x=10,y=20,z=5,a
  a=x<y<z;
  printf("%d",a);
}
```

a 1
b 0
c error
d none of above.

Ans:a

6.Are the following statements same?

```
 a<=20?b=30:c=30;
 (a<=20)?b:c=30;
```

ans:No.

7.can you suggest the other way of writing the following expression such that 30 is used only once

```
a<=20?b=30:c=30;
```

ans: *((a<=20)?&b:&c)=30;

8. How come the c standards says that the expression

```
x=y++ * y++;
```

is undefined, whereas the expression

```
x=y++ && y++;
```

is perfectly legal.

Ans: According to C standards an object's stored value can be modified only once (by evaluation of the expression) between two sequence points. A sequence point occurs:

8 At the end of full expression (expression which is not a sub-expression in a larger expression).
9 At the && , || , and ? : operator.
10 At a function call (after the evaluation of all arguments, just before the actual call)

Since the first expression y is getting modified twice between two sequence points the expression is undefined. The second expression is legal because a sequence point is occurring at && and y is getting modified once after this sequence point.

9.If a[x]=x++ is undefined, then by the same reason x=x+1 should also be undefined. But it is not so. Why?

Ans:The standard says that if an object is to get modified within an expression then all accesses to it within the same expression must be for computing the value to be stored in the object. The expression a[x]=x++ is disallowed because one of the accesses of x (the one in a[x]) has nothing to do with the value that ends up being stored in x. In this case the complier may not know whether the access should take place before or after the incremented value is stored. Since there is no good way to define it , the standard declares it to be undefined. As against this the expression x=x+1 is allowed because x is accessed to determine x’s final value.

10.Would the expression *p++ = c be disallowed by the complier.

Ans:No. Because even the value of p is accessed twice it is used to modify two different objects p and *p.

11.In the following code in which order the functions would be called.
A=f1(23,65) * f2(12/4) + f3();
a. f1,f2,f3.
b. f3,f2,f1.
c. The order may vary from complier to complier.
d. None of above.

Ans:C. Here the multiplication will happen before the addition, but in which order the function would be called is undefined.

12. In the following code in which order the functions would be called
a=(f1(23,65) * f2(12/4)) + f3();
a. f1,f2,f3.
b. f3,f2,f1.
c. The order may vary from complier to complier.
d. None of above.

Ans:C. Here the multiplication will happen before the addition, but in which order the function would be called is undefined. In an arithmetic expression the parentheses tell the complier which operands go with which operator but do not force the complier to evaluate everything within the parenthesis first.

13. What would be the output of the following program?

```
Main()
{
 int x=-3,y=2,z=0,m;
 m=++x && ++y || ++z;
 printf("\n %d  %d  %d  %d",x,y,z,m);
}
```

ans:-2 3 0 1

14 What would be the output of the following program?

```
Main()
{
 int x=-3,y=2,z=0,m;
 m=++y && ++x || ++z;
 printf("\n %d  %d  %d  %d",x,y,z,m);.
}
```

ans:-2 3 0 1

15. O/p?

```
main()
{
 int x-3,y=2,z=0,m;
m=++x || ++y && ++z;
printf("\n %d %d %d %d",x,y,z,m)
}
```

ans: -2 2 0 1

16. O/p?

```
main()
{
 int x-3,y=2,z=0,m;
m=++x && ++y && ++z;
printf("\n %d %d %d %d",x,y,z,m)
}
```

ans:-2 3 1 1