Test 1

1. Consider the following program:

```
#include<setjmp.h>
static jmp_buf  buf ;
main( )
{
   volatile  int b;
   b =3 ;
   if(setjmp(buf)!=0)  {
      printf("%d ", b) ;
      exit(0);
   }
   b=5;
   longjmp(buf , 1) ;
}
```

The output for this program is:

(a) 3
(b) 5
(c) 0
(d) None of the above

2. Consider the following program:

```
main( )
{
   struct node {
      int a ;
      int b ;
      int c ;
   } ;
   struct node  s= { 3, 5,6 } ;
   struct node *pt = &s ;
   printf("%d" ,  *(int*)pt);
}
```

The output for this program is:

(a) 3
(b) 5
(c) 6

(d) 7

3. Consider the following program:

```
void f(char *);
main( )
{
   f("123");
}
void f(char a[])
{
   if(a[1] =='\0') return ;
      f(a+1);
      f(a+1);
      printf("%c ", a[1]);
   }
```

The output for this program is:

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

4. Consider the following program:

```
/* tmp.c */
main(int argc , char*argv[] )
{
   if(argc ==1)
   printf("error");
   printf("%c " , *(argv[1] +1));
   printf("%c " , (*(argv+1))[2]);
   printf("%c " , argv[1][2]);
}
```

The output for this program is if input is given as $ tmp 3579

(a) 5 7 7
(b) 5 7 9
(c) 3 5 7
(d) 7 7 9

5. Consider the following program:

```
main( )
{
   unsigned char c;
   typedef struct name {
      long a;
      int b ;
      long c;
   }r ;
   r re = { 3,4,5};
   r *na= &re;
   printf("%d" ,*(int*) ((char*)na +  (unsigned int) &  ( (
   (struct name *)0 )->b) ) )   ;
}
```

The output for this program is:

(a) 3
(b) 4
(c) 5
(d) 6

6. Consider the following code segment:

```
int foo ( int x , int n)
{
   int val ;
   val =1;
   if (n>0) {
   if (n%2 == 1)  val = val *x;
      val = val * foo(x*x , n/2) ;
  }

 }
```

What function of x and n is compute by this code segment?

(a) x^n
(b) x*n
(c) n^x
(d) None of the above

7. Consider the following program:

```
void  foo(int a ,int b ,...);
main( )
{
```

```
    foo(1,5);
    foo(2,5,6);
}
void foo( int a,int b, ...)
{
    int j ;
    int *ptr = &b ;
    j=0 ;
    while(j<a) {
    printf("%d " , *ptr);
    ++j ;
    ++ptr ;
    }
}
```

The output for this program is:

(a) 5 5 6
(b) 1 5 2 5 6
(c) 1 2
(d) None of the above

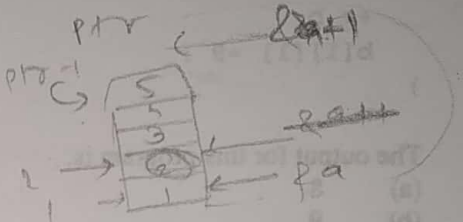
8. Consider the following program:

```
char i;
void try1();
void try2( char*);
main( )
{
    try1();
}
void try1( )
{
    static char *ptr = "abcde" ;
    i =*ptr ;
    printf("%c" , i);
    try2(++ptr) ;
}
void try2(char *t)
{
    static char *pt;
    pt = t + strlen(t)-1 ;
    if(i!=*pt--)
    if(t!=pt)
```

```
    try1();
}
```

The output for this program is:
(a) ab
(b) abcd
(c) abc
(d) abcde

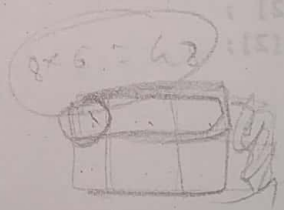
9. Consider the following program:



```
main( )
{
   int  a[5] = {1,2,3,4,5};
   int *ptr =  (int*)(&a+1);
   printf("%d %d" , *(a+1), *(ptr-1) );

}
```

The output for this program is:
(a) 2 2
(b) 2 1
(c) 2 5
(d) None of the above

10. Consider the following program:



```
main( )
{
   double a[2][3] ;
   printf("%d " ,  sizeof(a));
   printf("%d " ,  sizeof(a[1]));
   printf("%d " ,  sizeof(a[1][1]));
}
```

The output for this program is:
(a) 48 8 8
(b) 48 48 48
(c) 40 40 8
(d) 48 24 8

11. Consider the following program:

```
void foo(int [][3] );
main( )
```

```
{
   int a [3][3]= { { 1,2,3} , { 4,5,6},{7,8,9}};
   foo(a);
   printf("%d" , a[2][1]) ;
}
void foo( int b[][3])
{
   ++ b;
   b[1][1] =9 ;
}
```

The output for this program is:

(a) 8
(b) 9
(c) 7
(d) None of the above

12. Consider the following program:

```
main( )
{
   union  {
      struct{
         char c[2] ;
         char ch[2];
      }s;
      struct  {
         int i;
         int j ;
      }st;

   } u = { {12 ,1} , {15 , 1}} ;

   printf("%d %d", u.st.i , u.st.j) ;
}
```

The output for this program is:

(a) 15 12
(b) 268 271
(c) 12, 15
(d) None of the above

13. Consider the following program:

```
main( )
{
   struct {
      char a[11] ;
      int  i;
   }st= { "done" , 10 } ;
   printf("%s" , (&st)->a);
   printf("%d" , (&st)->i);
}
```

The output for this program is:

(a) done 10
(b) undefined undefined
(c) 0 0
(d) Not compile

14. Consider the following program:

```
main( )
{
   char *ptr;
   ptr = strtok("jan:feb,mar", ":");
   printf(ptr);
   do {
   ptr = strtok('\0', ":");
   if(ptr) printf(" %5s ", ptr);
   }  while(ptr);
}
```

The output for this program is:

(a) jan feb,mar
(b) jan feb mar
(c) jan:feb,mar
(d) None of the above

15. Consider the following program:

```
main( )
{
   int a, b,c, d;
   a=3;
   b=5;
   c=a,b;
   d=(a,b);
```

```
    printf("c=%d" ,c);
    printf("d=%d" ,d);
}
```

The output for this program is:

(a) c=3 d=3
(b) c=5 d=3
(c) c=3 d=5
(d) c=5 d=5

16. Consider the following program:

```
struct st {
    int a ;
    int b ;
} ;
void foo( struct st *);
main()
{
    struct st ab = { 128 , 768} ;
    struct st *pq =&ab;
    foo(pq);
}
void foo(struct st *p)
{
    char *pt;
    p->a =768;
    p->b =128;
    pt= (char*)p ;
    printf("%d" ,*++pt) ;
}
```

The output for this program is:

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

17. Consider the following program:

```
main( )
{
```

```
    int a[][3] = { 1,2,3 ,4,5,6};
    int (*ptr)[3] =a ;
    printf("%d %d "  ,(*ptr)[1], (*ptr)[2] );
    ++ptr;
    printf("%d %d"  ,(*ptr)[1], (*ptr)[2] );
}
```

The output for this program is:

(a) 2 3 5 6
(b) 2 3 4 5
(c) 4 5 0 0
(d) None of the above

18. Consider following function

```
int *f1(void)
{
int x =10 ;
return(&x);
}
int *f2(void)
{
   int*ptr ;
   *ptr =10;
   return ptr;
}
int *f3(void)
{
   int *ptr;
   ptr=(int*) malloc(sizeof(int));
   return ptr;
}
```

Which of the above three functions are likely to cause problem with pointers

(a) Only f3
(b) Only f1 and f3
(c) Only f1 and f2
(d) f1 , f2 ,f3

19. Consider the following statement:

s1 : An operator may require an lvalue operand , yet yield an rvalue
s2 : An operator may accept an rvalue operand , yet gives an lvalue

Which of the following is true about s1 and s2

(a) Both s1 and s2 are correct

(b) Only s1 is correct
(c) Only s2 is correct
(d) Both s1 and s2 are incorrect

20. Consider the following program:

```
main( )
{
   int i=3;
   int j;
   j = sizeof(++i+ ++i);
   printf("i=%d j=%d", i ,j);
}
```

The output for this program is:
(a) i=4 j=2
(b) i=3 j=2
(c) i=3 j=4
(d) i=3 j=6

Answers With Detailed Explanations

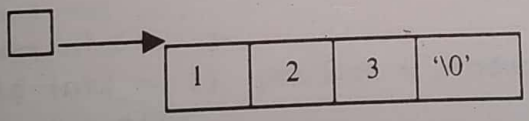
Test 1

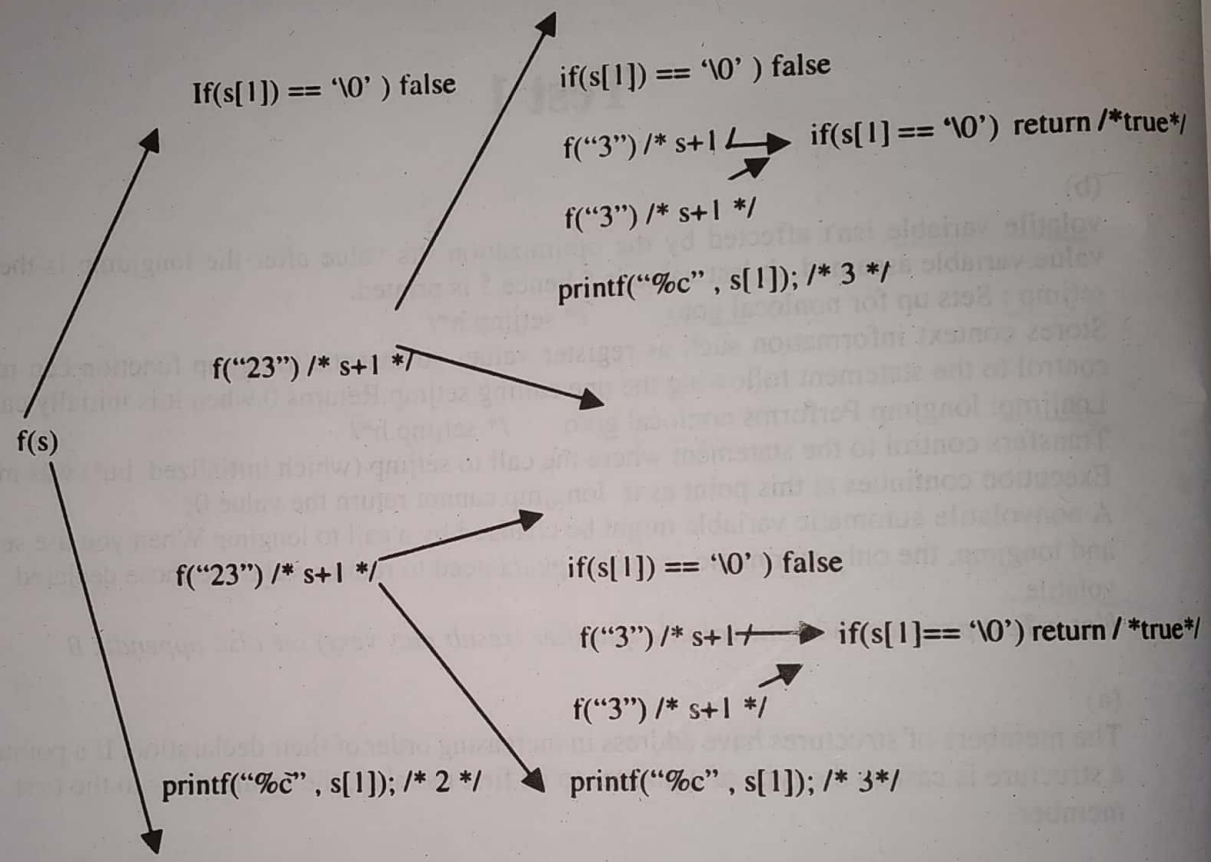
1. (b)
volatile variable isn't affected by the optimization. Its value after the longjump is the last value variable assumed. b last value is 5 hence 5 is printed.
setjmp : Sets up for nonlocal goto /* setjmp.h*/
Stores context information such as register values so that the lomgjmp function can return control to the statement following the one calling setjmp.Returns 0 when it is initially called.
Lonjjmp: longjmp Performs nonlocal goto /* setjmp.h*/
Transfers control to the statement where the call to setjmp (which initialized buf) was made. Execution continues at this point as if longjmp cannot return the value 0;
A nonvolatile automatic variable might be changed by a call to longjmp.When you use setjmp and longjmp, the only automatic variables guaranteed to remain valid are those declared volatile.
Note: Test program without volatile qualifier (result may very) see also appendix B

2. (a)
The members of structures have address in increasing order of their declaration. If a pointer to a structure is cast to the type of a pointer to its first member, the result refers to the first member.

3. (c)

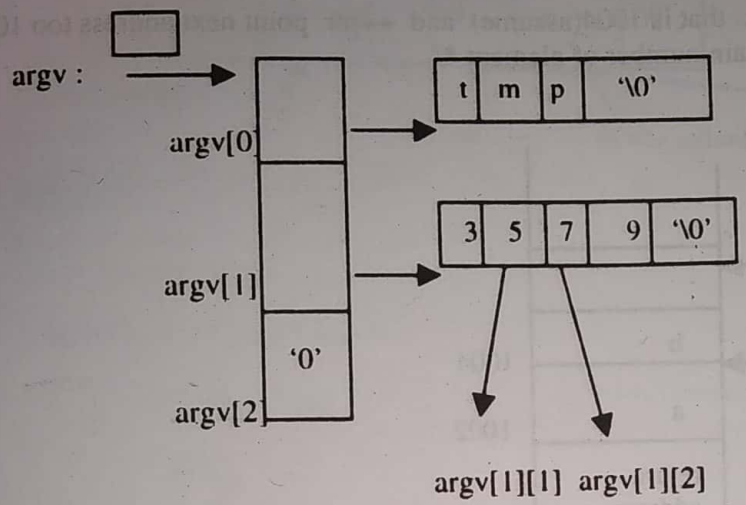
4. s:





4. (a)

argv[1][1]
Let x = argv[1]
argv[1][1] becomes x[1]
Now x[1] = (*(x+1))
Put x
argv[1][1] =(*(argv[1])+1))
Outer parenthesis are unnecessary, drop it
Second expression
argv[1] = (*(argv +1))
Now argv[1][2] = (*(argv+1))[2]



5. (b)
Null pointer is converted is supposed to guarantee that a simple offset(of a structure) is computed. cast to (char*) insure that offset so computed is a byte offset

6. (a)
Non recursive version of the program

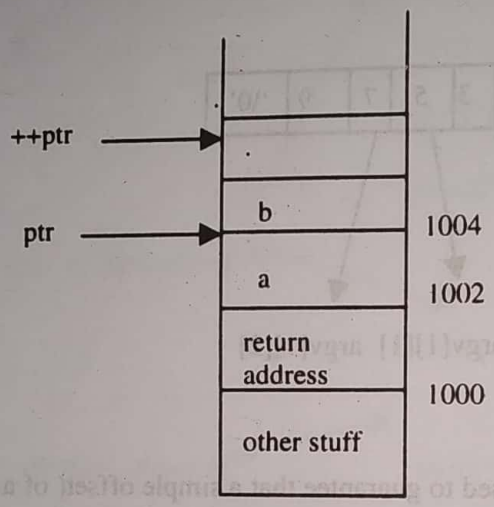
```
int  what ( int x , int  n)
{
   int val ;
   int product ;
   product =1;
   val =x;
   while(n>0)  {
   if (n%2 == 1)  product = product*val;
      n = n/2;
      val = val* val ;
   }
}
```

/* Code raise a number (x) to a large power (n) using binary doubling strategy */
Algorithm description
(while n>0) {
if next most significant binary digit of n(power) is one
then multiply accumulated product by current val ,
reduce n(power) sequence by a factor of two using integer division .

get next val by multiply current value of itself

}

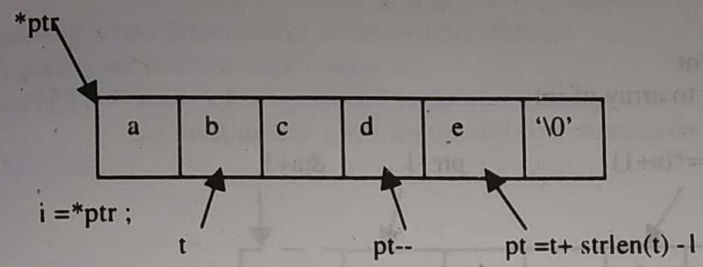
7. (a)
ptr contain addres of b that is 1004(assume) and ++ptr point next address too 1006 and so on /*while(j<a) a contain number of element */



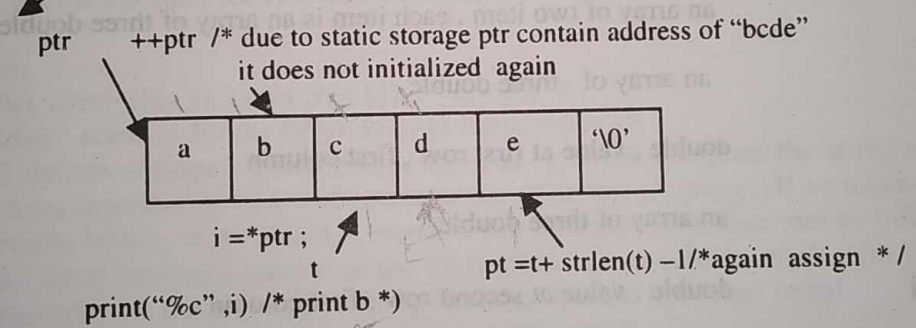
8. (c)

```
in try1( )
{
   .
   .

   printf("%c" , i);   /* print  a **/
   try2(++ptr)

}
```

figure show variable and pointer point after call for try2




```
try2( )
{

  if(i!=*pt--)  / * a!=e * true */

        if(t!=pt)  / true

          try1();
```

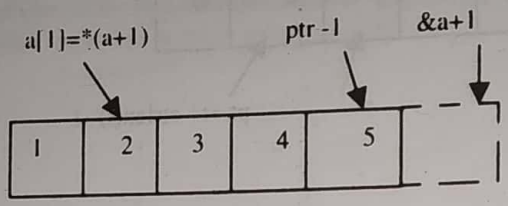



```
next call for try2
 try2()
 {

   if(t!=pt) /*true /
    try1();/*similarly */

 }
```

Note : ansi C support initialization of static variable within function

```
we may also initialize as
 void try2(char *t)
 {
      static char *pt= t + strlen(t)-1 ; / result vary(ab)/ ;
   ....
 }
```

9. (c)
type of a is array of int
type of &a is pointer to array of int



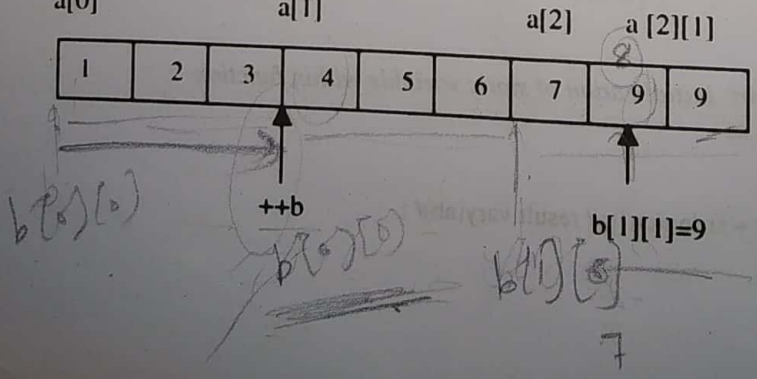
Taking a pointer to the element one beyond the end of an array is sure to work.

10. (d)
The sizeof operator produces different sizes. Compiler treats a and a[1] as array of different sizes. The size of a is 48 bytes (6 times the size of a double and the size of a[1] is 24 (3 time the size of a double).See table1

Expression	Meaning
a	an array of two item , each item is an array of three double
*a	an array of three double
**a	double , value at first row , first column
a[1]	an array of three double
*a[1]	double , value at second row , first column

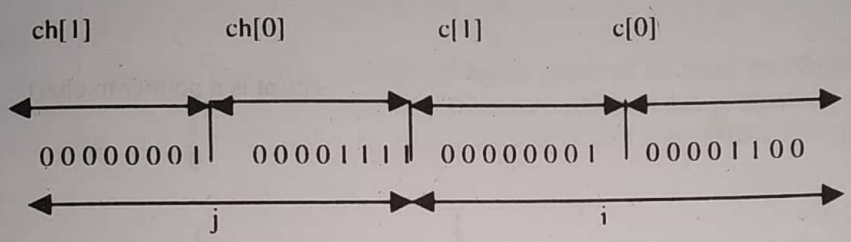
Table 1

11. (b)



For passing a two dimensional array to a function
The function declaration must match
foo(int [][3]) /* array of array decay to pointer to array)
in multidimensional array only first dimension may be missing */
foo((*a) [3]);

12 (b)



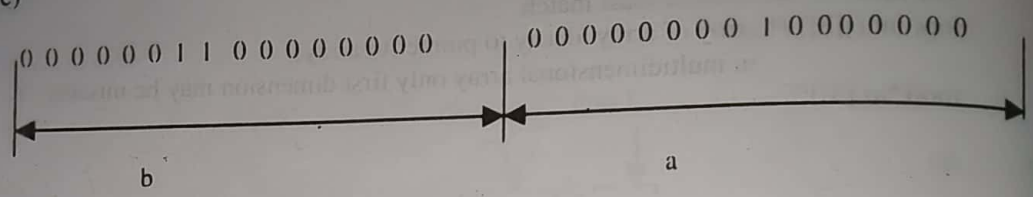
u.st.i = 00000001 00001100 = $(268)_{10}$
u.st.j = 00000001 00001111 = $(271)_{10}$

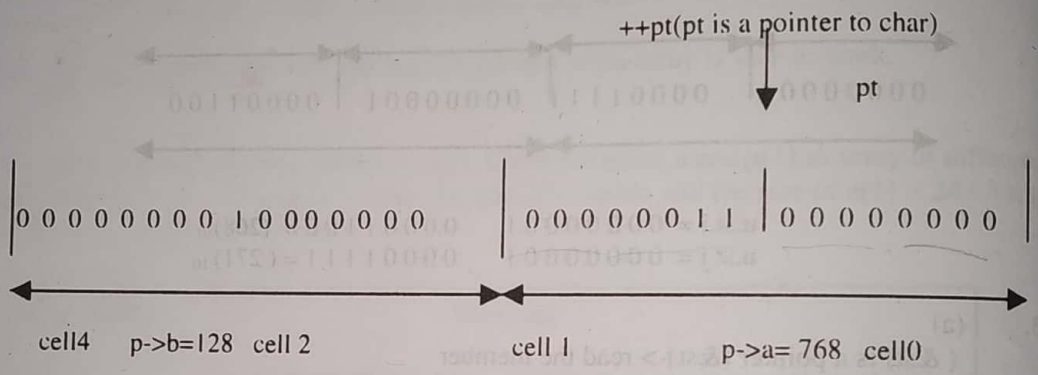
13. (a)
(&st) is a pointer (&st)-> read the member

14. (a)
char strtok(char s1, const char s2); /* string.h */
strtok scans s1 for the first token not contained in s2
s2 defines separator characters(in this case :). strtok interprets the string s1 as a series of tokens separated by spans of characters in s2(in this program :).If no tokens are found in s1, returns NULL. If the token is found, a null character is written into s1 following the token, and strtok returns a pointer to the token. Subsequent calls to strtok with NULL as the first argument use the previous s1 string, starting after the last token found.
"jan:feb,mar" /* only two token jan and feb,mar */
"jan:feb:mar" /for this string three token jan feb mar */

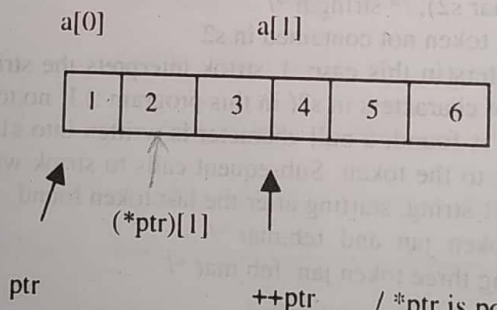
15. (c)
The comma separates the elements of a function argument list. The comma is also used as an operator in comma expressions. Mixing the two uses of comma is legal, but you must use parentheses to distinguish them. the left operand E1 is evaluated as a void expression, then E2 is evaluated to give the result and type of the comma expression. By recursion, the expression
E1, E2, ..., En
results in the left-to-right evaluation of each Ei, with the value and type of En giving the result of the whole expression.
c=a,b; / *yields c=a* /
d=(a,b); /* d =b */

16. (c)





17 (a)



18 (c)

f1 and f2 return address of local variable ,when function exit local variable disappeared

19. (a)

& is one such operator

```
int n;
&n =p      /* is not allowed */
*n  is lvalue
&n is  rvalue
```

operator	operand	result
&	n is lvalue	&n rvalue

20.

```
int a[2]
int *p = a;
p =3 ;          /* ok */
*(p+1) = 5;     /* ok */
```

operator	operand	result
*	(p+1) is rvalue	*(p+1) lvalue

20. (b)
sizeof operator gives the number of bytes required to store an object of the type of its operand . The operands is either an expression, which is not evaluated ((++i + ++ i) is not evaluated so i remain 3 and j is *sizeof* int that is 2)or a parenthesized type name.