Chapter 1
Arrays

1] What would the output of the following program?

```
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));
}
```

2] For the following statements would arr[3] and ptr[3] fetch the same character?   <Yes   /   No>

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

3] For the statements in 9.2 does the compiler fetch the character arr[3] and ptr[3] in the same manner?

4] What would be the output 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));
}
```

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

6] What would be the output of the following programm, if the aray begins at address 65486 ?

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

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

8]   What would be the output of the following prograam, if the aray
      begins at address 65486 ?

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

9]   When are 'char a[]' and 'char *a' treated as same by the
      compiler ?

10]   Would the following program compile successfully ?

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

11]   What would be the output of the following program ?

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

12]   A pointer to a block of memory is effectively same as an array.
<True   /   False>

13]   What would be the output 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);
}
```

14]   What does the follwoing declaration mean:

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

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

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

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

17]   What would be the output of the following program ?

```
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);
}
```

****

***********************   ANSWERS   ***************************

****

1]   11 2
     1 1

2]   Yes

3]   No.   For arr[3] the compiler generates code to start at
     location arr, move past it, and fetch the character there. When
     it sees the expression ptr[3] it generates the code to start at

location stored in ptr, add three to the pointer, 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]   1200 10

5]   No.   Whenever mentioning the array name gives its base
      address it is said that the array has decayed into a pointer.
      This decaying doesn't take place in two situations:

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

6]   65486 65486

7]   No.   Even though both may give the same addresses as in (6)
      they mean two different things. 'arr' gives the address of the
      first 'int' whereas '&arr' gives the address of array of
      'ints'. Since these addresses happen to be same the results of
      the expressions are same.

8]   65488 65496

9]   When using them as formal parameters while defining a function.

10]   No, because we may assign a new string ot a pointer but not to
      an array.

11]   4

12]   True

13]   65480 65496

14]   'ptr' is a pointer to an array of 10 integers.

15]   It decays into a pointer to an array and not a pointer to a
      pointer.

16]   fun(int a[][MAXCOL])
      {
      }

```
        OR

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


17]     1
```

Chapter-2
Bitwise Operators

```
1]      What would be the output of the following program?

        main()
        {
                int i = 32,j = 0x20,k,l,m;
                k = j | j;
                l = i & j;
                m = k ^ l;
                printf("%d %d %d %d %d",i,j,k,l,m);
        }
```

A]   32 32 32 32 0
B]   0 0 0 0 0
C]   0 32 32 32 32
D]   32 32 32 32 32

```
2]      What would be the output of the following program?

        main()
        {
                unsigned int m = 32;
                printf("%x",~m);
        }
```

A]   ffff
B]   0000
C]   ffdf
D]   ddfd

```
3]      What would be the output of the following program?
```

```
main()
{
        unsigned int a = 0xffff;
        ~a;
        printf("%x",a);
}
```

A]   ffff
B]   0000
C]   00ff
D]   None of the above.

4]   Point out the error in the following program.

```
main()
{
        unsigned int a,b,c,d,e,f;
        a = b = c = d = e = f = 32;
        a <<= 2;
        b >>= 2;
        c ^= 2;
        d |= 2;
        e &= 2;
        f ~= 2;
        printf("\n %x %x %x %x %x %x",a,b,c,d,e,f);
}
```


5]   To which numbering system can the binary number
      1011011111000101 be easily converted to?

6]   Which bitwise operator is suitable for checking whether a
      particular bit is 'on' or 'off'?

7]   Which bitwise operator is suitable for turning of a particular
      bit in a number?

8]   Which bitwise operator is suitable for putting on a particular
      bit in a number?

9]   On left shifting, the bits from the left are rotated an brought
      to the right and accommodated where there is empty space on the
      right? < True / False >

10]   Left shifthing a number by 1 is always equibalent to multiplying
it by 2. < Yes / No >

11]   Left shifting an unsigned int or char by 1 is always equivalent
      to multiplying it by 2. < Yes / No >
```

12]    What would be the output of the following program?

```
main()
{
      unsigned char i = 0x80;
      printf("\n %d",i << 1);
}
```

A]   0
B]   256
C]   100
D]   None of the above.

13]   What is the following program doing?

```
main()
{
  unsigned int m[] = {0x01,0x02,0x04,0x08,0x10,0x20,0x40,0x80};
  unsinged char n,i;
  scanf("%d",&n);
  for(i = 0; i <= 7; i++)
  {
      if (n & m[i])
         printf("\n yes");
  }
}
```


14]   What does the follwing program do?

```
main()
{
      char *s;
      s = fun(128,2);
      printf("\n %s",s)
}

fun(unsigned int num,int base)
{
      static char buff[33];
      char *ptr;

      ptr = &buff[ sizeof(buff) - 1];
      *ptr = '\0';
      do
      {
         *--ptr = "0123456789abcdef"[num % base];
         num /= base;
      }while (num != 0);
    return ptr;
}
```

A]   It converts a number to given base.
B]   It gives a compilation error.
C]   None of the above.

```
15]    #define CHARSIZE 8
       #define MASK(y) ( 1 << % CHARSIZE)
       #define BITSLOT(y) (y / CHARSIZE)
       #define SET(x,y) (x[BITSLOT(y) |= MASK(y))
       #define TEST(x,y) (x[BITSLOT(y)] & MASK(y))
       #define NUMSLOTS(n) ((n + CHARSIZE - 1) / CHARSIZE)

       Given the above macros how would you
(a)    declare an array arr of 50 bits.
(b)    put the 20th bit on.
(c)    test whether the 40th bit is 'on' or 'off'.


16]    Consider the macros in problem 14.15 above. On similar lines
       can you define a macro which would clear a given bit in a bit
       array?


17]    What does the following progaram do?

       main()
       {
             unsigned int num;
             int c = 0;
             scanf("%u",&num);
             for( ; num; num>>= 1)
             {
                   if(num & 1)
                      c++;
             }
             printf("%d",c);
       }

A]     It counts the number of bits which are on in the number num.
B]     It sets all bits in the number num to 1.
C]     It sets all bits in the number num to 0.
D]     None of the above.


18]    What would be the output of the following program?

       main()
       {
             printf("\n %x", -1 >> 4);
       }

A]     ffff
B]     0fff
C]     0000
D]     fff0


19]    In the statement expression1 >> expression2 if expressin1 is a
       signed integer with its leftmost bit set to 1 then on
```

right-shifting it the result of the statement would vary from
computer to computer. < True / False >

20]   What does the following program do?

```
main()
{
      unsigned int num;
      int i;
      scanf("%u",&num);
      for( i = 0; i < 16; i++)
         printf("%d",(num << 1 & 1 << 15)? 1: 0);
}
```

A]   It prints all even bits from num.
B]   It prints all odd bits from num.
C]   It prints binary equivalent fo num.
D]   None of the above.

****

*****************   ANSWERS   *****************

****

1]   A

2]   C

3]   A

4]   Error is in f~= 2, since there is no operator as '~='.

5]   Hexadecimal, since each 4-digit binary represents one
      hexadecimal digit.

6]   The '&' operator.

7]   The '|' operator.

8]   False.

10]   No.

11]   Yes.

12]   B

13]   B

14]   A

15]   char arr[NUMSLOTS(50)];
       SET(arr,20);
       if (TEST(arr,40))

16]   #define CLEAR(x,y) (x[BITSLOT(y)] &= ~MASK(y))

17]   A

18]   A. On computers which don't support sign extension you may
       get B.

19]   True.

20]   C

Chapter-3
Command Line arguments

1]   What do the 'c' and 'v' in 'argc' and 'argv' stand for?

2]   According to ANSI specifications which is the correct way of
     decalrating main() when it receives command line arguments?

A]   main(int argc, char *argv[])
B]   main(argc,argv)
     int argc; char *argv[];
C]   main()
     {
          int argc; char *argv[];
     }
D]   None of the above.

3]   What would be the output of the following program?

```
/* sample.c*/
main(int argc,char **argv)
{
     argc = argc - (argc - 1);
     printf("%s",argv[argc - 1]);
}
```


4]   If different command line arguments are supplied at different
     times would the output of the following program change?
     < Yes / No>

```
main(int argc, char *argv[])
{
     printf("%d",argv[argc]);
}
```


5]   If the following program (myprog) is run form the command line
     as myprog 1 2 3

     What would be the output/

```
main(int argc, char *argv[])
{
     int i;
     for(i = 0; i < argc; i++)
      printf("%s",argv[i]);
}
```


6]   If the following program (myprog) is run form the command line
     as myprog 1 2 3

     What would be the output?

```
main(int argc,char *argv[])
{
     int i;
     i = argv[1] + argv[2] + argv[3];
     printf("%d",i);
}
```

A] 123
B] 6
C] Error.
D] "123"

7]   If the following program (myprog) is run form the command line
     as myprog 1 2 3

     What would be the output?

```
main(int argc,char *argv[])
{
      int i,j = 0;
      for(i = o; i < argc; i++)
        j = j + atoi(argv[i]);

      printf("%d",j);
}
```

A] 123
B] 6
C] Error.
D] "123"

8]   Would the following program give the same output at all times?
     < Yes / No>

```
main(int argc, char*argv[])
{
      strcpy(argv[0],"hello");
      strcpy(argv[1],"good morning");
      printf("%s %s",argv[0],argc[1]);
}
```


9]   If the following program (myprog) is run from the command line
     as

     myprog one two three

     What would be the output?

```
main(int argc, char *argv[])
{
      printf("%s",*++argv);
}
```


10] If the following program (myprog) is run from the command line
     as

     myprog one two three

     What would be the output?

```
main(int argc, char *argv[])
{
      printf("%c",*++argv);
}
```

11] The variables 'argc' and 'argv' are always local to main.
     <True / False>

12] The maximum combined length of the command line arguments
     including the spaces between adjacent arguments is

A]   128 characters.
B]   256 characters.
C]   67 hcaracters.
D]   It may vary from one operating system to another.

13] What will the following program output?

```
main(int argc, char *argv[],char *env[])
{
      int i;
      for(i = 1; i < argc ; i++)
         printf("%s",env[i]);
}
```

A]   List of all environment variables.
B]   List of all command line arguments.
C]   Error.
D]   NULL.

14] If the following program (myprog) is run from the command line
     as

     myprog "*.c"

     What would be the output?

```
main(int argc, char *argv[])
{
      int i;
      for(i = 1; i < argc; i++)
         printf("%s",argv[i]);
}
```

A]   *.c
B]   List of all .c files in the current directory
C]   "*.c"
D]   None of the above

15] If the following program (myprog) is run from the command line
     as
```

```
myprog *.c
```

What would be the output?

```
main(int argc, char *argv[])
{
      int i;
      for(i = 1; i < argc; i++)
         printf("%s",argv[i]);
}
```

A]   *.c
B]   List of all .c files in the current directory
C]   "*.c"
D]   None of the above

16] If we want that any wildcard characters in the command line
     arguments should be appropriately expanded, are we required to
     make any special provision? If yes, which?

17] Does there exist any way to make the command line arguments
     available to other functions without passing them as arguments
     to the function? < Yes / No>

18] If the following program (myprog) is run from the command line
     as

     myprog Jan Feb Mar

     What would be the output?

     #include "dos.h"
     main()
     {
        fun();
     }

     fun()
     {
           int i;
           for(i = 0; i <_argc; i++)
             printf("%s",_argv[i]);
     }

19] If the following program (myprog) is present in the directory
     c:\bc\tucs then what would be its output?

     main(int argc, char *argv[])
     {
           printf("%s",argv[0]);
     }
```

A]      MYPROG
B]      C:\BC\TUCS\MYPROG
C]      Error
D]      C:\BC\TUCS


20]     Which is an easy way to extract myprog form the output of
        program 13.19 above?


21]     Which of the following is true about argv?

A]      It is an array of character pointers.
B]      It is a pointer to an array of character pointers.
C]      It is an array of strings.
D]      None of the above.



22]     If the following program (myprog) is run from the command line
        as

        myprog monday tuesday wednesday thursday

        What would be the output?

```
main(int argc, char *argv[])
{
      while(--argc > 0)
          printf("%s",*++argv);
}
```

A]   myprog monday tuesday wednesday thursday
B]   monday tuesday wednesday thursday
C]   myprog tuesday thursday
D]   None of the above.

23]   If the following program (myprog) is run form the command line
        as

        myprog friday tuesday sunday

        What will be the output?

```
main(int argc, char *argv[])
{
      printf("%c",(*++argv)[0]);
}
```

A]   m
B]   f
C]   myprog
D]   friday

24] If the following program (myprog) is run form the command line
    as

    myprog friday tuesday sunday

    What will be the output?

    ```
    main(int argc, char *argv[])
    {
        printf("%c",**++argv);
    }
    ```

A]   m
B]   f
C]   myprog
D]   friday

25] If the following program (myprog) is run form the command line
    as

    myprog friday tuesday sunday

    What will be the output?

    ```
    main(int argc, char *argv[])
    {
        printf("%c",*++argv[1]);
    }
    ```

A]   r
B]   f
C]   m
D]   y

26] If the following program (myprog) is run form the command line
    as

    myprog friday tuesday sunday

    What will be the output?

    ```
    main(int sizeofargv, char *argv[])
    {
        while(sizeofargv)
            printf("%s",argv[--sizeofargv]);
    }
    ```

A]   myprog friday tuesday sunday
B]   myprog friday tuesday
C]   sunday tuesday friday myprog
D]   sunday tuesday friday

```
*********************************************************************
************************   ANSWERS   *********************************
*********************************************************************
```

1]   Count of arguments and vector(array) of arguments.

2]   A

3]   C:\SAMPLE.EXE

4]   No

5]   C:\MYPROG.EXE 1 2 3

6]   C

7]   B. When atoi() tries to convert argv[0] to a number it cannot
      do so (argv[0] being a file name) and hence returns a zero.

8]   No

9]   one

10]   P

11]   True

12]   D

13]   B

14]   A

15]   A

16]   Yes. Compile the program as

       tcc myprog wildargs.obj

          This compiles the file myprog.c and links it withe the
       wildcard expansion module WILDCARGs.OBJ, then run the resulting
       executable file MYPROG.EXE.
          If you want the wildcard expansion to be default so that you
       won't have to link your program explicitly with WILDCARGS.OBJ,
       you can midigy our standard C?.LIB library files to have
       WILDCARGS.OBJ linked automatically. To acheive htis we have to
       remove SET ARGV from the library and add WILDCRAGS. The
       commands will invoke the Turbo Librarian to modify all the
       standard library files (assuming the current directory contains
       the standard C libraries, and WILDCRAGS.OBJ):

       tlib cs -setargv +wildargs
       tlib cc -setragv +wildargs
       tlib cm -strargv +wildargs
       tlib cl -setargv +wildargs
       tlib ch -setargv +wildargs

17]   Yes. Using the predefined variables_arcg,_argv.

18]   C:\MYPROG.EXE Jan Feb Mar

19]   B

20]   #include "dir.h"
       main(int arc, char *argv[])
       {
             char drive[3],dir[50],name[8],ext[3];
             printf("\n %s",argv[0]);
             fnsplit(argv[0],drive,dir,name,ext);
             printf("\n %s \n %s \n %s \n %s",drive,dir,name,ext);
       }

21]   A

22]   B

23]   B

24]   B

```
25]   A

26]   C
```

Chapter-4
Complicated Declarations

```
1]    What do the following declarations signify?

A.    int *f();

B.    int (*pf)();

C.    char **argv;

D.    void (*f[10])(int,int);

E.    char far *scr;

F.    char far *arr[10];

G.    int (*a[5])(int far *p);

H.    char far *scr1, *scr2;

I.    int (*ftable[])(void) = {fadd,fsub,fmul,fdiv};

J.    int (*ptr)[30];

K.    int *ptr[30];

L.    void *cmp();

M.    void (*cmp)();

N.    char (*(*f())[])();

O.    char (*(*x[3])())[5];

P.    void (*f)(int, void(*)());

Q.    int **(*)(int **, int **(*)(int **, int **));
```

```
R.      void(*)(void(*)(int *, void **), int(*)(void **, int *));

S.      char far * far *ptr;

T.      char far * near *ptr;

U.      char far * huge *ptr;
```

2]   What would be the output of the following program?

```
main()
{
    char near * near *ptr1;
    char near * far *ptr2;
    char near * huge *ptr3;
    printf("%d %d %d",sizeof(ptr1),sizeof(ptr2),sizeof(ptr3));
}
```

3]   What would be the output of the following program?

```
main()
{
    char far * near *ptr1;
    char far * far *ptr2;
    char far * huge *ptr3;
    printf("%d %d %d",sizeof(ptr1),sizeof(ptr2),sizeof(ptr3));
}
```

4]   What would be the output of the following program?

```
main()
{
    char huge * near *ptr1;
    char huge * far *ptr2;
    char huge * huge *ptr3;
    printf("%d %d %d",sizeof(ptr1),sizeof(ptr2),sizeof(ptr3));
}
```

5]   What would be the output of the following program?

```
main()
{
    char huge * near * far *ptr1;
    char near * far * huge *ptr2;
    char far * huge * near *ptr3;
    printf("%d %d %d",sizeof(ptr1),sizeof(ptr2),sizeof(ptr3));
}
```

6]   What would be the output of the following program?

```
main()
{
  char huge * near * far *ptr1;
  char near * far * huge *ptr2;
  char far * huge * near *ptr3;
  printf("%d %d %d",sizeof(ptr1),sizeof(*ptr2),sizeof(**ptr3));
}
```

7]   What would be the output of the following program?

```
main()
{
 char huge * near * far *ptr1;
 char near * far * huge *ptr2;
 char far * huge * near *ptr3;
 printf("%d %d %d",sizeof(*ptr1),sizeof(**ptr2),sizeof(ptr3));
}
```

8]   What would be the output of the following program?

```
main()
{
  char huge * near * far *ptr1;
  char near * far * huge *ptr2;
  char far * huge * near *ptr3;
  printf("%d %d %d",sizeof(**ptr1),sizeof(ptr2),sizeof(*ptr3));
}
```

9]   Are the following two declarations same? < Yes / No>

```
char far * far *scr;
char far far **scr;
```

10]   How would you declare the following:

      –   An array of three pointers to chars.
      –   An array of three char pointers.
      –   A pointer to an array of three chars.
      –   A pointer to a function which receives an int pointer
      and returns a float pointer.
      –   A pointer to a function which receives nothing and
      returns nothing.

11]   Can you write a program which would implement the follwoing
      declaration.

```
void (*f)(int, void (*)());
```

```
*********************************************************************

*********************         ANSWERS        **************************

*********************************************************************


1]

A]    f is a funciton returning pointer to an int.

B]    pf is a pointer to function which returns an int.

C]    argv is a pointer to a char pointer.

D]    f is an array of 10 function pointers, where each function
      receives two ints and returns nothing.

E]    scr is a far pointer to a char. ( far pointer is a pointer  which
contains an address which lies outside the data segment).

F]    arr is an array of 10 character pointers.

G]    a is an array of 5 function pointers. Each of these functions
      receive a far pointer to an int and returns an int.

H]    scr1 is a far pointer to a char, whereas scr2 is a near pointer
      to a char.

I]    ftable is an array of 4 function pointers which point to the
      functions fadd(), fsub() etc. Each of these functions accept
      nothing and return an int.

J]    ptr is a pointer to an array of 30 integers.

K]    ptr is an array of 30 pointers ot integers.

L]    cmp is a function returning pointer to void.

M]    cmp is a pointer to function which returns a void.

N]    f is a funciton returning pointer to array[] of pointer to
      function returning char.

O]    x is an array of 3 pointers to functions returning pointer to
      an array of 5 chars.
```

P]   f is a pointer to a funciton which returns nothing and receives
       as its parameter an integer and a pointer to a funciton which
       receives nothing and returns nothing.

Q]   f is a pointer to a function which returns a pointer to an int
       pointer and receives two arguments: a pointer to an int pointer
       and a function pointer which points to a function which
       receives two pointers to int pointers an returns a pointer to
       an int pointer.

R]   f is a pointer to a function which returns nothing and receives
       two arguments, both function pointers: the first function
       pointer points to a function which returns nothing but receives
       two arguments – an int pointer and a pointer to a void pointer;
       the second function pointer points to a function which returns
       an int pointer an receives a pointer to a void pointer and an
       int pointer.

S]   ptr is a far pointer to a far pointer to a char, or in easier
       words, ptr contains a far address of a far pointer to a char.

T]   ptr is a near pointer to a far pointer to a char, or in easier
       words, ptr contains a near address of a far pointer to a char.

U]   ptr is a huge pointer to a far pointer to a char, or in easier
       words, ptr contains a huge address of a far pointer to a char.

2]   2 4 4

3]   2 4 4

4]   2 4 4

5]   4 4 2

6]   4 4 4

7]   2 2 2

8]   4 4 4

9]   No.

10]   char *ptr[3];
       char *ptr[3];
       char (*ptr)[3];
       float *(*ptr)(int *);
       void(*ptr)();

```
11]    main()
       {
             void( *f)(int, void(*)());
             void fun(int, void(*)());
             void fun1();
             void(*p)();
             f = fun;
             p = fun1;
             (*f)(23,p);
       }

       void fun(int i, voud (*q)())
       {
             printf("Hello");
       }
       void fun1()
       {
             ;
       }
```

Chapter-5
Control Instructions

1]   What would be the output of the following program?

```
       main()
       {
             int i = 4;
             switch(i)
             {
                   default:
                   printf("\n a mouse is an elephant built by the
       japanese.");

                   case 1:
                   printf("\n Breeding rabbits is a hare raising
       experience.");

                   case 2:
                   printf("\n Friction is drag.");

                   case3:
```

```
              printf("\n If practice makes perfect, then
    nobody's perfect");
          }
      }
```

2]   Point out the error, if any, in the for loop:

```
    main()
    {
        int i = 1;
        for( ; ; )
        {
            printf("%d",i++);
            if (i > 10)
            break;
        }
    }
```

OPTIONS:
    (a)   The condition in the for loop is a must.
    (b)   The two semicolons should be dropped.
    (c)   The for loop should be replaced by a while loop.
    (d)   No error.

3]   Point out to the error, if any, in the while loop:

```
    main()
    {
        int i = 1;
        while()
    {
        printf("%d",i++);
            if (i > 10)
            break;
        }
    }
```

OPTIONS:
    (a)   The condition in the while loop is a must.
    (b)   Thereshould be a semicolons in the while loop.
    (c)   The while loop should be replaced by a for loop.
    (d)   No error.

4]   Point out the error, if any, in the while loop:

```
    main()
    {
        int i = 1;
        while(i <= 5)
        {
            printf("%d",i);
            if( i > 2)
            goto here;
        }
```

```
        }
        fun()
        {
            here:
                printf("\n If it works, Don't fix it.");
        }
```

5]   Point out the error, if any, in the following program:

```
main()
{
        int i = 4, j = 2;
        switch(i)
        {
                case 1:
                printf("\n To err is human, to forgive is
against company policy.");

                case j:
                printf("\n If you have nothing to do, don't do
it here.");
                break;
        }
}
```

6]   Point out the error, if any, in the following program:

```
main()
{
        int i = 1;
        switch(i)
        {
                case 1:
                printf("\n Radioactive cats have 18
half-lives.");
                break;

                case 1*2+4:
                printf("\n Bottle for rent-inquire within.");
                break;
        }
}
```

7]   Point out the error, if any, in the following program:

```
main()
{
        int i = 10;
        switch(a)
        {
        }
printf("Programmers never die. They just get lost in the
processing.");
        }
```

8] Point out the error, if any, in the following program:

```
main()
{
    int i = 1;
    switch(i)
    {
        printf("Hello");

        case 1:
        printf("\n Individualists unite.");
        break;

        case 2:
        printf("\n Money is the root of all wealth.");
        break;
    }
}
```

9] Rewrite the following set of statements using conditional operators:

```
int a = 1,b;
if(a > 10)
b = 20;
```

10] Point out the error, if any, in the following program:

```
main()
{
    int a = 10, b;
    a >= 5? b = 100: b = 200;
    printf("\n %d",b);
}
```

11] What would be the output of the following program?

```
main()
{
    char str[] = "part-time musicians are semiconductors";
    int a = 5;
    printf( a > 10 ? "%50s":"%s",str);
}
```

OPTIONS:
    (a)   Part-time musicians are semiconductors
    (b)   Part-time musicians are semiconductors
    (c)   Error
    (d)   None of the above

12]   What is more efficient a 'switch' statement or an 'if-else'
      chain?

13]   Can we use a 'switch' statement to switch on strings?

14]   We want to test whether a value lies in the range 2 to 4 or 5
      to 7. Can we do this using a 'switch'?

15]   The way 'break' is used to take the control out of 'switch' can
      'continue' be used to take the control to the beginning of the
      'switch?   <Yes / No>

*********************   ANSWERS   **********************

1]   A mouse is an elephant built by the Japanese
      Breeding rabbits is a hare raising experience

2]   D

3]   A

4]   'goto' cannot take control to a different function.

5]   Constant expression required in the second case, we cannot use
      j.

6]   No error. Constant exprerssions like 1*2+4 are acceptable in
      cases of a 'switch'.

7]   Though never required, there can exist a 'switch which has no
      cases.

8]   Though there is no error, irrespective of the value of i the
      first 'printf()' can never get executed. In other words, all
      statements in a switch have to beling to some case or the   other.

9]   int a = 1,b,dummy;
      a > 10 ? b = 20: dummy = 1;

```
        Note that the following would not have worked:
        a > 10 ? b = 20: ; ;
```

10]   1value required in function 'amin()'. The second assignment
       should be written in parentheses as follows:

```
        a >= 5 ? b = 100: (b = 200);
```

11]   A

12]   As far as eficiency is concerned there would hardly be any
       difference if at all. If the cases in a 'switch' are sparsely
       distributed the compiler may internally use the equivalent of
       an 'if-else' chain instead of a compact jump table. However,
       one should use 'switch' where one can. It is definitely a
       cleaner way to program and certainly is not any less efficient
       than the 'if-else' chain.

13]   No. The cases in a 'switch' must either have integer constants
       or constant experssion.

14]   Yes, though in a way which would not be very pratical if the
       ranges are bigger. The vay is ahown below:

```
        switch(a)
        {
               case 2:
               case 3:
               case 4:
                     /* statements */
               case 5:
               case 6:
               case 7:
                     /* some other statements */
               break;
        }
```

15]   No. 'continue' can work only with loops and not with 'switch'.

Facts About Arrays

FACTS ABOUT ARRAYS:

1]   An array is a collection of similar elements. It is also known
     as a subscripted variable.

2]   Before using an array its type and size must be declared. For
     example
     int arr[30];
     float a[60];
     char ch[25];

3]   The first element in the array is numbered as 0, so the last
     element is 1 less than the seze of the array.

4]   However big an array may be, its elements are always stored in
     contigouos locations.

5]   If we desire an array can be initilised at the same place where
it is declared. For example,

     int num[6] = {2,3,12,5,45,5};
     int n[] = {2,4,12,5,45,5};
     float press[] = {12.3,34.2,-23.4,-11.3};

6]   If the array is initialised where it is declared, mentioning the
dimension of the array is optional as in 2nd example above.

7]   Till the array elements are not given any specific values, they
are supposed to contain garbage values.

8]   In 'C' there is no check to see if the subscript used for an
array exceeds the size of the array. Data entered with a subscript
exceeding the array size will simply be placed in memory outside the
array; porbably on top of other data, or on the porgram itself. This
will lead to unpredictable results, to say the least, and there will be
no error message to warn you that you are going beyond the array size.
In some cases the computer may just hang. Thus, the following program
may turn out to be suicidal;:

```
/*   program  */
main()
{
      int num[40];
      for(i = 0;i<= 100;i++)
      num[i] = i;
}
```

     So do remember that, to see to it that we do not reach beyond the
array size is entirely the programmer's botheraation and not the
compiler's.

     While accessing array elements call by reference method is faster
as compared to accessing elements by subscripts. However for
convenience in programming we should observe the following :

     Array elements should be accessed using pointers, if the elements
are to be accessed in a fixed order, say form beginning to end, or form
end to beginning, or every alternate element or any such definite logic.

It would be easier to access the elements using a subscript if there is no fixed logic in accessing the elements. However, in this case also, accessing the elements by pointers would work faster than subscripts.

Chapter-6
Declarations and Initializations

WHAT WOULD BE THE OUTPUT OF THE FOLLOWING PROGRAMS:

```
1]    main()
      {
            char far *s1,*s2;
            printf("%d %d",sizeof(s1),sizeof(s2));
      }


2]    int x = 40;
      main()
      {
            int x = 20;
            printf("\n %d",x);
      }


3]    main()
      {
            int x = 40;
            {
                  int x = 20;
                  printf("\n %d",x);
            }
            printf("\n %d",x);
      }

4]    IS THE FOLLOWING STATEMENT RIGHT?
      extern int i;

5]    main()
      {
            extern int i;
            i = 20;
            printf("%d",sizeof(i));
      }

      (a)   2
      (b)   4
```

(c)   Would vary form compiler to compiler
        (d)   Error, i undefined

6]   Is it true that a global variable may have several
       declarations, but only one definition?   [Yes / No]

7]   Is it true that a function may have several declarations, but
       only one definition?   [Yes / No]

8]   In the following program where is the variable a getting
       defined and where is it getting declared?

       main()
       {
            extern int a;
            printf("%d",a);
       }
       int a = 20;

9]   main()
       {
            extern int a;
            printf("\n %d",a);
       }
       int a = 20;

       (a)   20
       (b)   0
       (c)   Garbage value
       (d)   Error

10]   What's the difference between a defintion and declaration of a
       variable?

11]   If the definition of an external variable occursin the source
file before its use in a particular function, then there is no need for
an extern declaration in the fuction.   [True / False]

12]   Suppose a program is divided into three files 'f1','f2' and 'f3'
and varable is defined in the file 'f1' but used in the files 'f2' and
'f3'. In such a case would we need the external declaration for the
variables in the files 'f2' and 'f3'.   [Yes / No]

13]   When we mention the prototype of a function are we defining the
function or declaring it?

14]   What's the difference between the following declarations?

       (a)   extern int fun();

```
(b)    int fun();
```

15]   Why does the following program report a redecalration error of
funtion display()?

```
main()
{
        display();
}
void display()
{
        printf("\n Cliffhanger");
}
```

16]   main()
```
{
        extern int fun(float);
        int a;
        a = fun(3.14);
        printf("%d",a);
}
int fun(aa)
float aa;
{
        return((int)aa);
}
```

(a)   3
(b)   3.14
(c)   0
(d)   Error

17]   Point out the error, if any, in the following program?

```
struct emp
{
        char name[20];
        int age;
}
/*    some more code may go here    */

fun(int aa)
{
        int bb;
        bb = aa * aa;
        return(bb);
}

main()
{
        int a;
        a = fun(200;
        printf("\n %d",a);
```

```
        }


18]    If you are to share the variables or fuctions across several
source files how would you ensure that all definitions and declarations
are consistent?


19]    How would you rectify the error in the following program/

       f(struct emp)
       /*    any other prototypes may go here     */

       struct emp
       {
              char name[20];
       int age;
       };

       main()
       {
              struct emp e = {"Soicher",34};
              f(e);
       }

       f(struct emp ee)
       {
              printf("\n %s %d",ee.name, ee.age);
       }


20]    Global variables are available to all functions. Does there exist
a mechanism by way of which K can make it available to some and not to
others.


21]    What do you mean by a translation unit?


22]    What would be the output of the following program?

       main()
       {
              in a[5] = {2,3};
              printf("\n %d %d %d",a[2],a[3],a[4]);
       }

       (a)    garbage values
       (b)    2 3 3
       (c)    3 2 2
       (d)    0 0 0


23]    main()
       {
              struct emp
              {
```

```
            char name[20];
            int age;
            float sal;
        };
    struct emp e = {"Tiger"};
    printf("\n %d %f",e.age,e.sal);
    }

    (a)    0 0.000000
    (b)    Garbage values
    (c)    Error
    (d)    None of the above
```

24]   some books suggest that the following definitions should be
proceded by the word static. Is it correct?

```
    int a[] = {2,3,4,12,32};
    struct emp e = {"sandy",23};
```

25]   Point out the error, if any, in the following program.

```
    main()
    {
        int(*p)() = fun;
        (*p)();
        }
        fun()
    {
        printf("\n loud and clear");
    }
```

26]   Print out the error, if any, in the following program.

```
    main()
    {
    union a
    {
        int i;
        char ch[2];
    };
    union a z = 512;
    printf("%d %d",z.ch[0],z.ch[1]);
    }
```

27]   What do you mean by scope of a variable? what are the different
types of scopes that a   variable can have?

28]   What are the different types of linkages?

```
*************************** ANSWERS ****************************


1]    4 2


2]    20
         Whenever there is a conflict between a local variable
      and a global variable, it is the local variable which
      gets a priority.


3]    20 40
         In case of a conflict between local variables, the one
      which is more local that gets the priority.


4]    Declaration


5]    D
         "extern int i" is a declaration and not a definition,
      hence the error.


6]    Yes


7]    Yes


8]    "extern in a" is the declaration whereas "int a = 20" is the
      definition.


9]    A


10]      In the definition of a variable space is reserved for
      the variable and some initial value is given to it, whereas a
      declaration only identifies the type of the variable for a
      function. Thus definition is the place where the variable is
      created or assigned storage is allocated.


11]   True


12]   Yes


13]   We are declaring it. When the function alongwith the statements
      belonging to it are mentioned we are defining the function.
```

14]   There is no difference except for the fact that the first one
       gives a hint that the function "fun()" is probably in another
       source file.

15]   Here "display()" is called before it is defined. In such cases
       the compiler assumes that the function "display()" is declared
       as
           int display();
       That is, an undeclared function is assumed to return an int and
       accept an unspecified number of arguements. Then when we define
       the function the compiler finds that it is returning void hence
the compiler reports the discrepancy.

16]   D
               The error occurs because we have mixed the ANSI
       prototype with K & R style of function definition.
               When we use ANSI prototype for a function and pass a float
to the function it is promoted to a double. When the fuction accepts
this double into a float a type mismatch occurs   hence the error.
       The remedy for this error could be to define the functiona as:

       int fun(float aa)
       {
           ..
           ..
           ..
       }

17]   Because of the missing semicolon at the end of the structure
       declaration (the intervening comment further obscures it) the
       compiler believes that fun() would return something of the type
       struct emp, whereas in actuality it is attempting to return an
       int. This causes a mismatch, hence an error results.

18]   The best arrangement is to place each definiton in a relevant
       .c file. Then, put an external decalration in a header file (
       .h file) and use "#include" to bring in the declaration
       wherever needed.

       The .c file which contains the definition should also include
       the header file, so that the compiler can check that compiler
       can check thta the definition matches the declaration.

19]   Declare the structure before the prototype of f().

20]   No.
               The only way this can be achieved is to define the
       variable locally in main() instead of defining it globally and
       then passing it to the functions which need it.

21] A translatin unit is a set of source files seen by the compiler
     and translated as a unit: generally one .c file, plus all
     header files mentioned in "#include" directives.

22] D.
          When an automatic array is partially initialised, the
     remaining array elements are initialised to 0.

23] A.
          When an automatic structure is partially initialised,
     the remaining elements of the structure are initialised to 0.

24] Pre-ANSI C compilers had such a requirement. Compilers which
conform to ANSI C standard do not have such a requirement.

25] Here we are initialising the function pointer p to the address of
the function "fun()". But during this initialisation the function has
not been defined. Hence an error.
     To eliminate this error add the prototype of the "fun()" before
declaration of p, as shown below:

     extern int fun();

     or simply

     int fun();

26] In a pre-ANSI compiler a union variable cannot be initialised.
     However, ANSI C permits initialisation of first member of the
     union.

27] Scope indicates the region over which the variable's
     declaration has an effect. The four kinds of scopes are: file,
     function, block and prototype.

28] There are three different types of linkages: external, internal
     and none. External linkage means global, non-static variables
     and functions, internal linkages means static variables and
     functions with file scope, and no linkage means local
     variables.

Chapter-7
Expressions

1]   What would be the output of the following program?

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

2]   What would be the output of the following program?

```
main(0
{
      int i = 3;
      i = i++;
printf("%d",i);
}
```

3]   The expression on the right hand side of && and || operators
      does not get evaluated if the left hand side determines the
      outcome?   <True /   False>

4]   What would be the output of the following program?

```
main()
{
      int i = 2;
      printf("\n %d %d",++i,++i);
}
```

OPTIONS:
      (a)   3 4
      (b)   4 3
      (c)   4 4
      (d)   Output may vary from compiler to compiler.

5]   What would be the output of the following program?

```
min(0
{
      int x = 10, y = 20,z = 5, i;
      i = x < y < z;
      printf("\n %d",i);
}
```

OPTIONS:
      (a)   1
      (b)   0
      (c)   Error
      (d)   None of the above.

6]   Are the following two statements true?

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

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

7]   Can you suggest any other way of writing the following
       expression such that 30 is used only once?

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

8]   How come that the C standard says that the expression

       j = i++ * i++;
       is undefined, whereas, the expression
       j = i++ && i++;
       is perfectly legal.

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

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

11]   In the following code in which order the funcitons would be
       called?

       a = f1(23,14) * f2(12/4) + f3();

OPTIONS:
       (a)   f1,f2,f3
       (b)   f3,f2,f1
       (c)   The order may vary from compiler to compiler.
       (d)   None of the above.

12]   In the following code in which order the functions would be
       called?

       a = (f1(23,14) * f2(12/4)) + f3();

OPTIONS:
       (a)   f1,f2,f3
       (b)   f3,f2,f1
       (c)   The order may vary from compiler to compiler.
       (d)   None of the above.

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

       main()
       {
            int i = -3, j = 2, k = 0, m;

```
        m = ++i && ++j || ++k;
        printf("\n %d %d %d %d",i,j,k,m);
    }
```

14]   What would be the output of the following program?


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


15]   What would be the output of the following program?


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


16]   What would be the output of the following program?


```
    main()
    {
        int i = -3, j = 2, k = 0, m;
        m = ++i && ++j && ++k;
        printf("\n %d %d %d %d",i,j,k,m);
    }
```


****

*************************   ANSWERS   **********************

****

1]   0 0 1
      That's what some of the compilers would give. But some other
      compiler may give a different answer. The reason is, when a
      single expression causes the same object to be modified or to
      be modified and then inspected the behaviour is undefined.

2]   4.
      But basically the behaviour is undefined for the same reason as
      in (1) above.

3]   True.
      For example if 'a' is non-zero then 'b' will not be evaluated
      in the expression 'a || b'.

4]   D.
      The order of evaluation of the arguments to a function call is
      unspecified.

5]   A.

6]   No.

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

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

      -     at the end of full expression (expression which is not
      a sub-expression in a larger expression)
      -     at the &&, || and ?: operators
      -     at a fucntion call (after the evaluation of all
      arguments, just before the actual call)

      Since in the first expression 'i' is getting modified twice
      between two sequence points the expression is undefined. Also,
      the second expression is legal because a sequence point is
      occurring at && and 'i' is getting modified once before and
      once after this sequence point.

9]   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[i] = i++' is disallowed because one
      of the accesses of 'i' (the one in a[i]) has nothing to do
      with the value that ends up being stored in 'i'.
          In this case the compiler may not know whether the
      access should take place before or after the incremented value
      is stored. Since there's no good way to define it. the standard
      declares it as undefined. As against this the expression   'i =
i   + 1' is allowed because 'i' is accessed to determine 'i's' final
value.

10]   No.

Because here even though the value of p is accessed twice it is used to modify two different objects p and *p.

11]   C.
       Here the multiplication will happen before the addition, but in which order the functions would be called is undefined.

12]   C.
       Here the multiplication will happen before the addition, but in which order the functions would be called is undefined. In an arithmetic expression the parentheses tell the compiler which operators but do not force the compiler to evaluate everything within the parentheses first.

13]   -2 3 0 1

14]   -2 3 0 1

15]   -2 2 0 1

16]   -2 3 1 1

Chapter-8
Floating point Issues

1]   WHAT WOULD THE OUTPUT OF THE FOLLOWING PROGRAM BE?

```
main()
{
     float a = 0.7;
     if ( a < 0.7)
           printf("C");
     else
           printf("C++");
}
```

OPTIONS:
     (a)   C
     (b)   C++
     (c)   Error
     (d)   None of the above

2]   WHAT WOULD THE OUTPUT OF THE FOLLOWING PROGRAM BE?

```
main()
{
        float a = 0.7;
        if ( a < 0.7f)
                printf("C");
        else
                printf("C++");
}
```

OPTIONS:
       (a)   C
       (b)   C++
       (c)   Error
       (d)   None of the above

3]   WHAT WOULD THE OUTPUT OF THE FOLLOWING PROGRAM BE?

```
main()
{
        printf("%f",sqrt(36.0));
}
```

OPTIONS:
       (a)   6.0
       (b)   6
       (c)   6.000000
       (d)   Some absurd result

4]   Would this program give proper results?   <Yes / No>

```
main()
{
        printf("%f",log(36.0));
}
```


5]   Would the following 'printf()' print the same values for any
value of a?   <Yes / No>

```
main()
{
        float a;
        scanf(%f",&a);
        printf("%f",a + a + a);
        printf("%f",3 * a);
}
```


6]   We want to round off c, a float, to an int value. The correct way
to do so would be

       (a)   y = (int)(x + 0.5);
       (b)   y = int(x + 0.5);
       (c)   y = (int) x + 0.5;
       (d)   y = (int)((int)x + 0.5);
```

7]   Which error are you likely to get when you run the following
program?

```
        main()
        {
                struct emp
                {
                        char name[20];
                        float sal;
                }
                struct emp e[10];
                int i;
                for ( i = 0; i <= 9; i++)
                scanf("%s %f",e[i].name,&e[i].sal);
        }
```

OPTIONS:
        (a)   Suspicious pointer conversion
        (b)   Floating point formats not linked
        (c)   Cannot use 'scanf()' fpr structures
        (d)   Strings cannot be nested inside structures

8]   What causes the error in problem 4.7 above to occur and how would
you rectify the error   in the above program?

9]   Which are the three different types of real data types of real
data types available in C   and what are the format specifiers used
for them?

10]   By default any real number is treated as

        (a)   a float
        (b)   a double
        (c)   a long double
        (d)   Depends upon the memory model that you are using.

11]   What should you do to treat the constant 3.14 as a 'float'?

12]   What would be the output of the following program?

```
        main()
        {
                printf("%d %d %d",sizeof(3.14f),sizeof(3.14));
        }
```

OPTIONS:
        (a)   4 4 4
        (b)   4 Garbage value Garbage value
        (c)   4 8 10
        (d)   Error

14]   The binary equivalent of 5.375 is

      (a)   101.101110111
      (b)   101.011
      (c)   101011
      (d)   None of the above

15]   How 'floats' are stored in binary form?

16]   A float occupies 4 bytes. If the hexadecimal equivalent of each
of these bytes is A,B,C   and D, then when this float is stored in
memory these bytes get stored in the order

      (a)   ABCD
      (b)   DCBA
      (c)   0xABCD
      (d)   0xDCBA

17]   If the binary equibalent of 5.375 in normalised form in 0100 0000
1010 1100 0000 0000   0000 0000, what would be the output of the
following program?

```
main()
{
      float a = 5.375;
      char *p;
      int i;
      p = (char*) &a;
      for(i = 0;i <= 3; i++)
            printf("%02x",(unsigned char)p[i]);
}
```

OPTIONS:
      (a)   40 AC 00 00
      (b)   04 CA 00 00
      (c)   00 00 AC 40
      (d)   00 00 CA 04

****

*******************   ANSWERS   *********************

****

1]   A

2]   B

3]   D

4]   No, since we have not included the header file "math.h".

5]   No. For example, for 1.7 the two 'printf()s' would print
different values.

6]   A

7]   A

8]   What causes the 'floating point formats not linked' error to
      occur? When the compiler encounters a reference to the address
      of a float it sets a flag to have the linder link in the
      floationg point emulator. A floating point emulator is used to
      manipulate floating point numbers in runtime library functions
      like 'scanf()' and 'atof()'. There are some cases in which the
      reference to the folat is a bit obscure and the compiler does
      not detect the need for the emulator.

      These situations usually occur during the initial stages of
      program development. Normally, once the program is fully
      developed, the emulator will be used in such a fashion that the
      compiler can accurately determine when to link in the emulator.

      To force linking of the floating point emulator into an
      application just include the following function in your
      progaram:

      void LinkFloat(void)
      {
            float a = 0, *b = &a; /* cause emulator to be linked */
            a = *b;
      }

      There is no need to call this funciton form your program.

9]   float   4 bytes   %f
      double   8 bytes   %lf
      long double 10 bytes   %Lf

10]   B

11]   Use 3.14f

12]   Use 3.141

13]   C

14]   B

15]   Floating-point numbers are represented in IEEE format.
       The iEEE format for floating point storage uses a sign   bit,
a   mantissa and an exponent for representing the power of 2. The
       sign bitdenotes the sign of the number: a) represents   a
       positive value   and a 1 denotes a negative value. The
       mantissa is represented in binary after converting it   to
its   normalised from. The normalised form results in a manissa   whose
most significant digit is always 1. The IEEE format takes advantage
of this by not storing this bit at all. The   exponent is an integer
stored in unsigned binary format after   adding a positive integer
bias. This ensures that the stored exponent is always positive.. The
value of the bias is   127 for   floats and 1023 for doubles.

16]   B

17]   C

Chapter-9
Functions

1]   What would be the output of the following program?

```
main()
{
    int a,b;
    a = sumdig(123);
    b = sumdig(123);
    printf(" %d %d",a,b);
}

sumdig(int n)
{
    static int s = 0;
    int d;
    if( n != 0)
    {
        d = n % 10;
        n = (n - d) / 10;
        s = s + d;
        sumdig(n);
    }
```

```
              else
                     return(s);
        }


2]     What error would the following function give on compilation?

       f(int a, int b)
       {
              int a;
              a = 20;
              return a;
       }

OPTIONS:
       (a)    Missing parenthesis in return statement.
       (b)    The function should be defined as 'int f(int a,int b)'.
       (c)    Redeclaration of 'a'.
       (d)    None of the above.


3]     There is a mistake in the following code. Add a statement in it
       to remove  it.

       main()
       {
              int a;
              a = f(10,3.14);
              printf("%d",a);
       }

       f(int aa, float bb)
       {
              return ((float) aa + bb);
       }


4]     Point out the error in the following code.

       main()
       {
              int a = 10;
              void f();
              a = f();
              printf("\n %d",a);
       }

       void f()
       {
              printf("Hi");
       }


5]     Point out the error, if any, in the following function.

       main()
       {
```

```
            int b;
            b = f(20);
            printf(" %d",b);
      }

      int f(int a)
      {
            a>20? return(10): return(20);
      }
```

6]   A function cannot be defined inside another function.
      <True / False>

7]   Will the following functions work? <Yes / No>

```
      f1(int a,int b)
      {
            return(f2(20));
      }

      f2(int a)
      {
            return(a * a);
      }
```

8]   What are the following two notations of defining functions
      commonly known as:

```
      int f(int a,  float b)
      {
            /* some code */
      }

      int f(a,b)
      int a; float b;
      {
            /* some code */
      }
```

9]   In a function two return statements should never occur.
      <True / False>

10]   In a function two return statements should never occur
      successively. < True / False>

11]   In C all functions except 'main()' can be called recursively.
      < True / False>

12]   Usually recursion works slower than loops. < True / False>
```

13]   Is it true that too many recursive calls may result int stack
overflow?  <Yes  / No>


14]   How many times the following program would print 'Jamboree'?

```
mani()
{
      printf)"\n Jamboree");
      main();
}
```

OPTIONS:
      (a)   Infinite number of times
      (b)   32767 times
      (c)   65535 times
      (d)   Till the stack doesn't overflow.


```
**********************************************************************

**********************       ANSWERS        *********************

**********************************************************************
```

1]   6 12

2]   C

3]   Add the following function prototypw in amin():

      float f(int, float);

4]   In spite of defining the function 'f()' as returning 'void',
      the program is trying to colect the value returned by 'f()' in
      the variable 'a'.

5]   'return' statement cannot be used as shown withe the conditional
operators. Instead the following statement may be   used:

```
        return(a > 20 ? 10: 20);
```

5]   True.

7]   Yes.

8]   The first one is known as ANSI notation and the second is known
       as Kernighan and Ritchie or simply, K & R notation.

9]   False.

10]   True.

11]   False. Any function including 'main()' can be called recursively.

12]   True.

13]   Yes.

14]   D.

12]

Chapter-10
Inputs and Outputs

```
*********************************************************************

*********************      INPUT AND OUTPUT  *************************

*********************************************************************
```

1]   What would be the output of the following program ?

```
       main()
       {
            int a = 250;
            printf("%1d",a);
```

```
        }


2]    What would be the output of the following program ?

      main()
      {
            float a = 3.15529;
            printf("\n %6.2f",a);
            printf("\n %6.3f",a);
            printf("\n %5.4f",a);
            printf("\n %2.1f",a);
            printf("\n %0.0f",a);
      }


3]    In the following code

      #include <stdio.h>
      main()
      {
            FILE *fp;
            fp = fopen("trial","r");
      }
            'fp' points to
      (a)   The first character in the file.
      (b)   A structure which contains a 'char' pointer which
      points to the first character in the file.
      (c)   The name of the file.
      (d)   None of the above.


4]    Point out the error, if any, in the following program:

      main()
      {
            unsigned char;
            FILE *fp;
            fp = fopen((ch = getc(fp)) != EOF)
                  printf("%c",ch);

            fclose(fp);
      }


5]    Point out the error , if any, in the following program:

      #include "stdio.h"
      main()
      {
            unsigned char;
            FILE *fp;
            fp = fopen("trial","r");
            if (!fp)
            {
                  printf("Unable to open file.");
                  exit();
```

```
              }
              fclose(fp);
       }


6]     If a file contains the line " I am a boy\r\n" then on reading
       this line into the array 'str' using 'fgets()' what would 'str'
       contain ?

       (a)    "I am a boy\r\n\0"
       (b)    "I am a boy\r\0"
       (c)    "I am a boy\n\0"
       (d)    "I am a boy"


7]     Point ou the error if any in the following progarm :

       #include "stdio.h"
       main()
       {
            FILE *fp;

            fp = fopen("trial","r");
            fseek(fp,20,SEEK_SET);
            fclose(fp);
       }


8]     To print out a and b below, which printf() statement would you
       use ?

       float a = 3.14;
       double b = 3.14;

       (a)    printf("%f %f",a,b);
       (b)    printf("%Lf %f",a,b);
       (c)    printf("%Lf Lf",a,b);
       (d)    printf("%f %Lf",a,b);


9]     To scan a and b fiven below which 'scanf()' statement would you
       use ?

       float a;
       double b;

       (a)    scanf("%f %f",&a,&b);
       (b)    scanf("%Lf %Lf",&a,&b);
       (c)    scanf("%f %Lf",&a,&b);
       (d)    scanf("%f %lf",&a,&b);


10]    Point out the wrror in the following program.

       # include<stdio.h>
       main()
       {
```

```
       FILE *fp;
       char str[80];

       fp = fopen("trial","r");
       while(!feof(fp))
       {
              fgets(str,80,fp);
              puts(str);
       }
fclose(fp);
}
```

11] Point out the error in the following program.

```
#include"stdio.h"
main()
{
       char ch;
       int i;
       scanf("%c",&i);
       scanf("%d",&ch);
       printf("%c %d",ch,i);
}
```

12] What would be the output of the following program ?

```
main()
{
       printf("\n % % % %");
}
```

13] Point out the error, if any, in the following program?

```
#include"stdio.h"
main()
{
       FILE *fp;
       fp = fopen("c:\tc\trial","w");
       if(!fp)
              exit();
       fclose(fp);
}
```

14] Would the following code work?   <Yes, No>
    If yes, what would be the output?

```
main()
{
       int n = 5;
       printf("\nn = %*d",n,n);
}
```

15]   What is the * in the printf() of (14) indicative of?

16]   Can we specify variable field width in a scanf() format string?
       <Yes , No>

17]   To tackle a double in printf() we can use %f, Whereas in
       scanf() we should use %lf.   <Yes , No>

18]   Out of fgets() and gets() which function is safe to use?

19]   A file writtern in text mode canf be read back in binary mode.
       <True , False>

20]   We should not read after a write to a file without an
       intervening call to fflush(),fseek() or rewind().
<True / False>

****

************************** ANSWERS *****************************

****

1]   250

2]   3.16
       3.155
       3.1553
       3.2
       3

3]   B

4]   EOF has been defined as #define EOF -1 in the file "stdio.h"
       and an usingned char ranges from 0 to 255 hence when EOF is
       read from the file it cannot be accommodated in ch. Solution is
       to declare ch as an int.

5]   No error.

6]   C

7]   Instead of 20 use 20L since fseek() needs a long offset value.

8]   A. It is possible to print a souble using %f.

9]   D

10]   The last line from the file "trial" would be read twice. To
      avoid this use:
            while(fgets(str,80,fp) != NULL)
              puts(str);

11]   You would not get a chance to supply a character for the second
      scanf() statement. Solution is to precede the second scanf()
      with the following statement.
            fflush(stdin);
         This would flush out the enter hit for the previous scanf()
      to be flushed out from the input stream, i.e. keyboard.

12]   %%

13]   The path of the filename should have been written as
      "c:\\tc\\trial".

14]   Yes.
      n = 5

15]   It indicates that an 'int' value from the argument list will be
      used for fiedld width. In the argument list the width precedes
      the value to be printed. In this case the format specifier
      becomes %5d.

16]   No. A '*' in scanf() format string after a % sign is used for
      suppression of assignment. That is, the current input field is
      scanned but not stored.

17]   True.

18]   'fgets()', because unlike 'fgets()', gets() cannot be told the
      size of the buffer into which the string supplied would be

stored. As a result there is always a possibility of overflow of buffer.

19]   False.

20]   True.

Chapter11
Library Functions


```
*********************************************************************

*****************            LIBRARY FUNCTIONS            *****************

*********************************************************************
```


1]   What do the functions atoi(), atoa() and gcvt() do? Show how
      would you use them in a program.

2]   Does there exist any other function which can be used to
      convert an integer or a float to a string? If yes, show how you
      would use it.

3]   How would you use qsort() function to sort an array of
      structures?

4]   How would you use qsort() function to sort the names stored in
      an array of pointers to strings?

5]   How would you use bsearch() function to search a name stored in
      an array of pointers to strings?

6]   How would you use the function sin(), pow(), sqrt()?

7]   How would you use the function memcpy()?

8]   How would you use the function memset()?

9]   How would you use the function memmove()?

10]   How would you use the functions fseek(), fread(), fwrite() and
       ftell()?

11]   How would you obtain the current time and difference between
       two times?

12]   How would you use the function randomize() and random()?

13]   Would the following program always output 'Bangalore'?

```
main()
{
      char str1[] = "Bangalore - 440010";
      char str2[10];

      strncpy(str2,str1,8);
      printf("\n %s",str2);
}
```


14]   Can you shorten this code?

```
main()
{
      char str1[] = "Bangalore - 440010";
      char str2[10];

      str2[0] = '\0';
      strncat(str2,str1,8);
      printf("%s",str2);
}
```


15]   How would you implement a substr() function that extracts a
       substring from a given string?

16]   Given a sentence containing words separated by spaces how would
you construct an array of pointers to strings containing
       addresses of each word in the sentance?

17]   Write the comparison function qs_compare()for the following
code struct date
```
      {
            int d,m,y;
      };
      qs_compare(const void*,const void*);
```

```
main()
{

    struct date  dd[]={
                        {17,11,62},
                     {24,8,78},
                   {17,11,62},
                      {16,12,76},
                           {19,2,94}
              };

    inti,w;
    clrscr ();
    w=sizeof(struct date);
    qsort(dd,5,w,qs_compare);


    for(i=0;i<4;i++)
        printf("\n%d%d%d",dd[i].dd[i].m,dd[i].y);
}
```

18]   How should we sort a linked list?

19]   What's the difference between the functions
rand(),random(),srand() and randomise()?

20]   What's the difference between the funtion memmove() and
memcpy()?

21]   How would you print a string on the printer?

22]   Can you use the funtion fprintf()to display the output on the
screen?

****

*********************   ANSWERS   *********************

```
*********************************************************************


1]    atoi() converts a string to an integer.

      itoa() converts an integer to a string.

      gcvt() converts a floating-point number to a string.


      # include"stdlib.h"
      main()
      {
            char s[] = "12345";
            char buffer[15],string[20];
            int i;

            i = atoi(s);
            printf("\n %d",i);

            gcvt(20.141672,4,buffer);
            printf("\n %s",buffer);

            itoa(15,string,2);
            printf("\n %s",string);
      }


2]    The function sprintf() can be used for this purpose. This
      function also has the ability to format the numbers as they are
      converted to strings. The following program shows how to use
      this function.

      # include"stdio.h"
      main()
      {
            int a = 25;
            float b = 3.14;
            char str[40];

            sprintf(str,"a = %d b = %f",a,b);
            puts(str);
      }


3]    # include"string.h"
      # inlucde"stdlib.h"

      struct stud
      {
            int rollno;
            int marks;
            char name[30];
      };

            int sort_m(struct stud *,struct stud *);
```

```c
        int sort_name(struct stud *,struct stud *);
        int sort_marks(struct stud *,struct stud *);

main()
{
        statis struct stud ss[] = {
                                {15,96,"Akshay"};
                                {2,97,"Madhuri"};
                                {8,85,Aishvarya"};
                                {10,80,"Sushmita"};
                        };
        int x,w;

        clrscr();
        w = sizeof(struct stud);

        printf("\n In order of roll numbers:");
        qsort(ss,4,w,sort_m);

    for(x = 0; x < 4; x++)
    printf("\n %d %s %d",ss[x].rollno,ss[x].name,ss[x].marks);

        printf("\n\n In order of names:");
        qsort(ss,4,w,sort_name);

    for(x = 0; x < 4; x++)
    printf("\n %d %s %d",ss[x].rollno,ss[x].name,ss[x].marks);

        printf("\n\n In order of marks:");
        qsort(ss,4,w,sort_marks);

    for(x = 0; x < 4; x++)
    printf("\n %d %s %d",ss[x].rollno,ss[x].name,ss[x].marks);
}


int sort_m(struct stud *t1,struct stud *t2)
{
        return(t1->rollno - t2->rollno);
}

int sort_name(struct stud *t1,struct stud *t2)
{
        return(strcmp(t1->name - t2->name));
}

int sort_marks(struct stud *t1,struct stud *t2)
{
        return(t1->marks - t2->marks);
}


4]   # include"string.h"
     # include"stdlib.h"

     int sort_name(const void * ,const void * );
```

```
main()
{
        char *names[] = {
                        "Akshay";
                        "Madhuri";
                        "Aishvarya";
                        "Sushmita";
                        "Sudeepta";
                };

        int i;

        qsort(names,5,sizeof(char *), sort_name);
        for(i = 0; i < 5; i++)
           printf("\n %s",names[i]);
}

int sort_name(const void *t1, const void *t2)
{
  /* t1 and t2 are always pointers to objects being compared */

        char **t11, **t22;

        /* cast appropriately */
        t11 = (char **)t1;
        t22 = (char **)t2;
        return(strcmp(*t11,*t22));
}
```

5]   ```
include"string.h"
include"stdlib.h"

int sort_name(const void *, const void *);
int bs_compare(char **,char **);

main()
{
        char *names[] = {
                        "Akshay";
                        "Madhuri";
                        "Aishvarya";
                        "Sushmita";
                        "Sudeepta";
                };

        int i,wide,nel;
        char *s = "aMadhuri", **b;

        qsort(names,5,sizeof(char *),sort_name);
        clrscr();
        for(i = 0; i < 5; i++)
           printf("\n %s",names[i]);

        wide = sizeof(names[0]);
        nel = sizeof(names) / wide;
        b = bsearch(&s, names, nel, wide, bs_compare);
```

```c
            if(b == NULL)
               printf("Not found");
            else
               pritf("\n\n %s",*b);
        }


        int sort_name(const void *t1, const void *t2)
        {
          /* t1 and t2 are always pointers to objects being compared */

                char **t11, **t22;

                /* cast appropriately */
                t11 = (char **)t1;
                t22 = (char **)t2;
                return(strcmp(*t11,*t22));
        }

        int bs_compare(char **s1,char **s2)
        {
                return(strcmp(*s1, *s2));
        }


6]     # include"math.h"
       main()
       {
                int ang;
                float angrad,x,y,a,b;

                printf("\nEnter the angle in degrees");
                scanf("%d",&and);

                angrad = ang * 3.14/ 180;
                c = sin(angrad);
                a = pow(x,2);
                b = 1 - a;
                y = sqrt(b);

                printf("\n cosine of angle %d is: %d",ang,y);
       }


7]     # include"math.h"
       # include"alloc.h"

       main()
       {
                int area;
       char scr[]="Pray, not for lighter load, but for stronger back";
                char *dest;

                area = sizeof(src);
                dest = malloc(area);
                mamcpy(dest,src,area);
```

```
            printf("\n %s",src);
            printf("\n %s",dest);
      }


8]    # include"mem.h"
      main()
      {
            int area;
            char src[] = "Bon jour, Madam";

            area = sizeof(src);
            memset(src,"!",area - 7);

            printf("\n %s",src);
      }


9]    # include"mem.h"
      # include"alloc.h"

      main()
      {
            int area;
            char *dest;
            char src[] ="Life is the camera and you are the target"
                        "so keep smiling always";

            area = sizeof(Src);
            dest = malloc(Area);
            memmove(dest,src,area);

            printf("\n %s",dest);
            printf("\n %s",src);
      }


10]   # include"stdio.h"

      struct stud
      {
            int rollno;
            char name[10];
            float per;
      }e;

      FILE *fs;

      main()
      {
            long position = OL;
            int rollno;
            char ch;
            float temp;

            fs = fopen("stud.dat","rb+");
```

```c
            if(fs == NULL)
            {
                    puts("Unable to open file");
                    exit(1);
            }

            do
            {
              printf("\n Enter code no. to modify:");
              scanf("%d",&rollno);

              while(fread(&e,sizeof(e),1,fs) == 1)
              {
                    if(e.rollno == rollno)
                    {
                      printf("\n Enter the new record");
                      dcanf("%s %f",e.name, &temp);
                      e.per temp;
                      fseek(fs,position, SEEK_SET);
                      fwrite(&e,sizeof(e),1,fs);
                      break;
                    }
                position = ftell(fs);
              }

              puts("You want to modify records");
              ch = getche();
            }while(ch == 'Y');
      }


11]   # include"time.h"
      # include"dos.h"

      main()
      {
            time_t t1,t2;
            double diff,f;
            int i = 2;

            time(&t1);
            sleep(i);
            time(&t2);
            diff = difftime(t2,t1);

            printf("\n Program was active for %lf seconds",diff);
      }


12]   # include"dos.h"
      # inlcude"stdlib.h"

      main()
      {
            randomize();
```

```
            printf("\n Press any key to stop.");
            while!kbhit())
            {
                    sound(random(500));
                    delay(random(100));
                    nosound();
            }
        }


13]     No. Because after copying the source string into the target
        string strncpy() doesn't terminate the target string with a
        '\0'. A better way of copying would be:

        str[2] = '\0';
        strncat(str2,str1,8);

        strncat() always terminates the target string with a '\0'.


14]     Yes, using sprintf() as shown below:

        main()
        {
                char str1[] = "Bangalore-440010";
                char str2[10];

                sprintf(str2,"%.*s",8,str1);
                printf("%s",str2);
        }


15]     main()
        {
                char str1[] = "Bangalore";
                char str2[5];

        /* Extract 3 characters beginning with first character */
                sprintf(str2,str1,1,3);
                printf("%s",str2);
        }

        substr(char *t,char *s,int pos,int len)
        {
                t[0] = '\0';
                strncat(t,s + pos,len);
        }


16]     # include"stdio.h"
        # include"string.h"
        main()
        {
                char str[] = "This is a test";
                char *ptr;
                char *p;
                int i = 1,j;
```

```
                p = strtok(str," ");
                if(p != NULL)
                {
                        [tr[0] = p;
                        while(1)
                        {
                                p = strtok(NULL," ");
                                if(p == NULL)
                                        break;
                                else
                                {
                                        ptr[i] = p;
                                        i++;
                                }
                        }
                }

                for(j = 0; j < i; j++)
                        printf("\n %s",ptr[j]);
        }


17]     int qs_compare(const void *p1,const void *p2)
        {
                const struct date *sp1 = p1;
                const struct date *sp2 = p2;

                if(sp1->y < sp2->y)
                        return(-1);
                else if(sp1->y > sp2->y)
                        return(1);
                else if(sp1->m < sp2->m)
                        return(-1);
                else if(sp1->m > sp2->m)
                        return(1);
                else if(sp1->d < sp2->d)
                        return(-1);
                else if(sp1->d > sp2->d)
                        return(1);
                else
                        return(0);
        }
```

18]   Often it's easier to keep the list in order as we build it
        rather than sorting it later. Still if we want to sort the list
        then we can allocate a temporary array of pointers, fill it
        with pointers to the various nodes in the list, call qsort()
        and finally rebuild the list pointers based on the sorted   array.

19]   rand()   : returns a random number.
        random()   : retruns a random number in a specified range.
        srand()   : initialses a random number generator with a
                given seed value.

```
          randomize   : initialises a random number genenrator with a
                    random value based on time.


20]    Both the functions copy a block of bytes from source to
       destination. However, if source and destination overlaps the
       behaviour of memcpy() is undefined whereas memmove() carries
       out the copying correctly. memcpy() is more efficient, whereas
       memmove() is safer to use.


21]    # include"stdio.h"
       main()
       {
              char str[] = "There wouldn't have been COBOL without C";
              fprintf(stdprn,"\n %s \n",str);
       }


22]    Yes, by replacing the usual file pointer with stdout as shown
       below:

       fprintf(stdout,"%s %d %f",str,i,a);
```

Chapter-12
Memory Allocation

```
********************************************************************

*************          MEMORY ALLOCATION        *****************

********************************************************************


1]     What would be the output of the follwoing program?

       main()
       {
              char *s;
              char *fun();


              s =s fun();
              printf("%s",s);
       }

       char *fun()
       {
              char buffer[30];
```

```
            strcpy(buffer,"RAM - Rarely Adequate Memory");
            return(buffer);
    }
```

2]   What is the solution for the problem in program 17.1 above?

3]   Does there exist any other solution for the problem 17.1?

4]   How would you dynamically allocate a 1-D array of integers?

5]   How would you dynamically allocate a 2-D array of integers?

6]   How would you dynamically allocate a 2-D array of integers such
      that we are able to access any element using 2 subscripts, as
      in arr[i][j]?

7]   How would you dynamically allocate a 2-D array of integers such
      that we are able to access any element using 2 subscripts, as
      in arr[i][j]? Also the rows of the array should be stored in
      adjacent memory locations.

8]   How many bytes would be allocated by the following code?

```
#include "alloc.h"
#define MAXROW 3
#define MAXCOL 4

main()
{
    int(*p)[MAXCOL];
    p = (int(*)[MAXCOL]malloc(MAXROW * sizeof(*p));
}
```

9]   What would be the output of the following program?

```
#include"alloc.h"
#define MAXROW 3
#define MAXCOL 4

main()
{
    int(*p)[MAXCOL];
    p = (int(*)[MAXCOL])malloc(MAXROW * sizeof(*p));
    printf("%d %d",sizeof(p),sizeof(*p));
}
```

10]   In the following code p is a pointer to an array of MAXCOL
      elements. Also, malloc() allocates memory for MAXROW such

arrays. Would these arrays be stored in adjacent locations? How
would you access the elements of these array using p?

```
#define MAXROW 3
#define MAXCOL 4
main()
{
    int i,j;
    int (*p)[MAXCOL];
    p = (int(*)[MAXCOL])malloc(MAXROW * sizeof (*p));
}
```


11]   How many bytes would be allocated by the following code?

```
#include "alloc.h"
#define MAXROW 3
#define MAXCOL 4
main()
{
    int(*p)[MAXCOL][MAXROW];
    [ = (int(*)[MAXCOL])malloc(sizeof(*p));
}
```


12]   How would you dynamically allocate a 3-D array of integers?

14]   How many bytes of memory would the following code reserve?

```
#include "alloc.h"
main()
{
    int *p;
    p = (int *)malloc(256 * 256);
    if(p == NULL)
        printf("Allocation failed");
}
```


14]   How would you free the memory allocated by the follwoing
program?

```
#include"alloc.h"
#define MAXROW 3
#define MAXCOL 4

main()
{
    int **p,i;
    p = (int **)malloc(MAXROW * sizeof(int *));
    for(i = 0; i < MAXROW; i++)
        p[i] = (int *)malloc(MAXCOL * sizeof(int));
}
```

15]   How would you free the memory allocated by the following
      program?

```
#include"alloc.h"
#define MAXROW 3
#define MAXCOL 4

main()
{
      int **p,i,j;

      p = (int **)malloc(MAXROW * sizeof(int*));
      p[0] = (int *)malloc(MAXROW * MAXCOL * sizeof(int));

      for(i = 0; i < MAXROWl i++)
          p[i] = p[0] + MAXCOL;
}
```

16]   Would the following code work at all times?

```
main()
{
      char *ptr;
      gets(ptr);
      printf("%s",ptr);
}
```

17]   The following code is improper though it may work some times.
      How would you improve it?

```
main()
{
      char *s1 = "Cyber";
      char *s2 = "Punk";
      strcat(s1,s2);
      printf("%s",s1);
}
```

18]   What ould be the output of the second printf() in the following
      program?

```
#include "alloc.h"
main()
{
      int *p;
      p = (int*)malloc(20);
      printf("%u",p);    /* Suoopse this prints 1314 */
      free(p);
      printf("%u",p);
}
```

19]   Something is logically wrong with this program. Can you point
      it out?

```
#include "alloc.h"
main()
{
        struct ex
        {
          int i;
          float j;
          char *s;
          }
struct ex *p;

p = (struct*) malloc(sizeof(struct ex));
p -> (char *) malloc(20);
free(p);
}
```

20] To free() we only pass the pointer to the block of memory which
     we want to deallocate. Then how does free() know how many bytes
     it should deallocate?

21] What would be the output of the following program?

```
main()
{
        char *ptr;
        *ptr = (char*) maolloc(30);
        strcpy(ptr,"RAM- Rarely Adequate Memory");
        printf("\n %s",ptr);
        free(ptr);
}
```

****

****************   ANSWERS   *********************

****

1] The output is unpredictable since buffer is an auto array and
    would die when the control goes back to main(). Thus s would be
    pointing to an array which no longer exists.
```

```
2]    main()
      {
             char *s;
             char *fun();

             s = fun();
             printf("%s",s);
      }

      char *fun()
      {
             static char bufer[30];
             static(buffer,"RAM - Rarely Adequate Memory");
             return(bufer);
      }


3]    #include "alloc.h"
      main()
      {
          char *s;
          char *fun();

             s = fun();
             printf("%s",s);
             free(s);
      }

      char *fun()
      {
             char *ptr;
             ptr = (char *)malloc(30);
             strcpy(ptr,"RAM - Rarely Adequate Memory");
             return(ptr);
      }


4]    #include "alloc.h"
      #define MAX 10
      main()
      {
             int *p,i;
             p = (int *)malloc(MAX * sizeof(int));
             for(i = 0;i < MAX; i++)
             {
               p[i] = i;
               printf("%d",p[i]);
             }
      }


5]    #include "alloc.h"
      #define MAXROW 3
      #define MAXCOL 4
      main()
```

```
        {
              int *p,i,j;
              p = (int *)malloc(MAXROW * MAXCOL * sizeof(int));
              for(i = 0; i < MAXROW; i++)
              {
                 for(j = 0; j < MAXCOL; j++)
                 {
                    p[i * MAXCOL + j] = i;
                    plrintf("%d",p[i * MAXCOL + j]);
                 }
              printf("/n");
              }
        }


6]    #include "alloc.h"
      #define MAXROW 3
      #define MAXCOL 4
      main()
      {
              int **p,i,j;
              p = (int **)malloc(MAXROW * sizeof(int));
              for(i = 0; i < MAXROW; i++)
                p[i] = (int *)malloc(MAXCOL * sizeof(int));

      for(i = 0; i < MAXROW; i++)
              {
                 for(j = 0; j < MAXCOL; j++)
                 {
                    p[i][j] = i;
                    printf("%d",p[i][j]);
                 }
              printf("/n");
              }
        }


7]    #include "alloc.h"
      #define MAXROW 3
      #define MAXCOL 4
      main()
      {
              int **p,i,j;
              p = (int **)malloc(MAXROW * sizeof(int));
              p[0] = (int *)malloc(MAXROW * MAXCOL * sizeof(int));
              for(i = 0; i < MAXROW; i++)
                p[i] = p[0] + i * MAXCOL;

      for(i = 0; i < MAXROW; i++)
              {
                 for(j = 0; j < MAXCOL; j++)
                 {
                    p[i][j] = i;
                    printf("%d",p[i][j]);
                 }
              printf("/n");
              }
```

```
        }


8]    14 bytes


9]    2  8


10]   The arrays are stored in adjacent locations. You can confirm
      this by printing their addresses using the following loop.

      int i;
      for(i = 0; i < MAXROW; i++)
        lprintf("%d",p[i]);

      To access the array elements we can use the following set of
      loops.

      for(i = 0; i < MAXROW; i++)
          {
             for(j = 0; j < MAXCOL; j++)
                printf("%d",p[i][j]);
          }


11]   14 bytes


12]   #include "alloc.h"
      #define MAXY 3
      #define MAXY 4
      #define MAXZ 5
      main()
      {
          int ***p,i,j,k;
          p = (int ***)malloc(MAXX * sizeof(int**));
          for(i = 0; i< MAXX; i++)
          {
             p[i] = (int **)malloc(MAXY * sizeof(int *));
             for(j = 0; j < MAXY;  j++)
                 p[i][j] = (int *)malloc(MAXZ * sizeof(int));
          }

      for(k = 0; k < MAXZ; k++)
      {
         for(i = 0; i < MAXX; i++)
         {
            for(j = 0; j < MAXY; j++)
            {
               p[i][j][k] = i + j + k;
               lprintf("%d",p[i][j][k]);
            }
         printf("/n");
         }
      printf("/n");
      }
```

```
        }


13]    It would fail to allocate any memory because 256 * 256 id 65536
       which when passed to malloc() would become a negative number
       since the formal argument of malloc(), an unsigned int, can
       accomodate numbers only upto 65535.


14]    for(i = 0; i < MAXROW; i++)
          free(p[i]);
          free(p);


15]    free(p[0]);
       free(p);


16]    No. Since ptr is an uninitialised pointer it must be pointing
       at some unknown location in memory. The string that we type in
       would get stored at the location to which ptr is pointing
       thereby overwriting whatever is present at that location.


17]    main()
       {
            char s1[25] = "Cyber";
            char *s2 = "Punk";
            strcat(s1,s2);
            printf("%s",s1);
       }


18]    1314


19]    The memory chunk at which 's' is pointing has not been freed.
       Just freeing the memory pointed to by the struct pointer p is
       not enough. Ideally we should have used:

       free(p->s);
       free(p);

            In short when we allocate structures containing
       pointers to other dynamically allocated objects before freeing
       the structure we have to first free each pointer in the
       structure.


20]    In most implimentations of malloc() the number o fbytes
       allocted is stored adjacent to the allocated block. Hence it is
       simple for free() to know how many bytes ato deallocate.


21]    2
```

22]   No.

23]   Yes, using the realloc() function as shown below:

```
#include "alloc.h"
main()
{
      int *p;
      p = (int *)malloc(20);
      t = p;
      t = (int *)realloc(p,40);
      if(t == NULL)
      {
         printf("Cannot reallocate, leaves previous");
         printf("region unchanged.");
      }
      else
      {
         if(p == t)
            ; /* the array expanded at the same region */
         else
         {
            free(p); /* deallocate the original array */
            p = t; /* ste p to newly allocated region */
         }
      }
}
```


24]   Both. If the first strategy fails then it adopts the second. If
       the first is successful it returns the same pointer that you
       passed to it otherwise a different pointer for the newly
       allocated space.

25]   If realloc() expands allocated memory at the same place then
       there is no need of readjustment of other pointers. However. if
       it allocates a new region somewhere else the programmer has to
       readjust the other pointers.

26]   As against malloc(), calloc() needs two arguments, the number
       of elements to be allocted and othe size of each element. For
       example,

       p = (int *)calloc(10,sizeof(int));

       would allocate space for a 10 integer array. Additionally,
       calloc() would also set each of this element with a value 0.

27]   The same that we use with mallloc(), i.e. free().

28]   True.

29]   64 KB.

30]   Use the standard library functions farmalloc() and farfree().
      These functions are specific to DOS.

31]   Yes. Using interrupt 0x67, the details of which are beyond the
      scope of this book.

32]   While assigning the address returned by malloc() we should use
      ptr and not *ptr.

Chapter-13
Pointers


```
**********************************************************************

**********************          POINTERS          **********************

**********************************************************************
```


1]   Can you combine the following two statements into one?

      ```
      char *p;
      p = malloc(100);
      ```


2]   Can you split the following statement into two statements?

      ```
      char far *scr = (char far *) 0xB8000000L;
      ```


3]   Are the expressions '*ptr++' and '++*ptr' same?

4]   Can you write another expression which does the same job as
      '++*ptr' ?

5]   What would be the equivalent pointer expression for referring
      the same element as 'a[i] [j] [k] [l]'?

6]   What would be the ouptut of the following program?

      ```
      main()
      ```

```
    {
        int arr[] = { 12,13,14,15,16};
        printf("\n d%d%d"  sizeof(arr), sizeof(*arr),
    sizeof(arr[0])));
```

7]   What would be the output of the following program assuming that
      the array begins at location 1002?

```
main()
{
    int a[3][4] = {
                1,2,3,4,
                5,6,7,8,
                9,10,11,12
                 };
        printf("\n %u %u %u",a[0]+1, *(a[0]+1), *(*(a+0)+1));
```

8]   What would be the output of the followingt program assuming
      that the array begins at location 1002?

```
main()
{
    int a[2][3][4] = {
                     {
                         1,2,3,4,
                         5,6,7,8,
                         9,1,1,2
                     },
                     {
                         2,1,4,7,
                         6,7,8,9,
                         0,0,0,0
                     }
                 };
    printf('\n %u %u %u %d",a,*a,**a,***a);
}
```

9]   In the following program how would you proint 50 using 'p'?

```
main()
{
    int a[] = { 10,20,30,40,50};
    char *p;
    p = (char *)a;
};
```

10]   Where can one think of using pointers?

11]   In the following program add a statement in the function
      'fun()' such that address of a gets a stored in j.

```
main()
```

```
    {
        int *j;
        void fun(int**);

        fun(&j);
    }

    void fun(int**k)
    {
        int a = 10;
        /* add statement here */
    }
```

12] Would the following program give a compilation error or
     warning?   <Yes / No>

```
    min()
    {
        float i = 10, *j;
        void *k;
        k = &i;
        j = k;
        printf("\n %f",*j);
    }
```

13] Would the following program compile?

```
    main()
    {
        int a = 10, *j;
        void *k;
        j = k = &a;
        j++;
        k++;
        printf("\n %u %u",j,k);
    }
```

14] Would the following code compile successfully?

```
    main()
    {
        printf("%c",7["Sundaram"]);
    }
```

```
*********************************************************************

************************   ANSWERS   *****************************

*********************************************************************


1]    char *p = malloc(100);


2]    char far *scr;
      scr = (char far *) 0xB8000000L;


3]    No.   '*ptr++' increments the pointer and not the value
      pointed by it,whereas '++*ptr' increments the value being
      pointed by 'ptr'.


4]    (*ptr)++


5]    *(*(*(*(a + i) + j) + k) + l)


6]    10  2  2


7]    1004  2  2


8]    1002  1002  1002  1


9]    printf("\n %d",*((int *)p + 4));


10]   At lot of places, some of which are:

      (a)   Accreeing array or string elements.
      (b)   Dynamic memory allocation.
      (c)   Call by reference
      (d)   Implementing linked lists, trees, graphs and many other
      data structures.


11]   *k = &a;


12]   float(*arr[3])(int,int);
```

13]   No.   Here no typecasting is required while assigning the   value
to and form k because conversions are applied   automatically
when other pointer types are assigned to and from   void* .

14]   No.   An error would be reported in the statement 'k++' since
      arithmetic on void pointers is not permitted unless the void
      pointer is appropriately typecasted.

15]   Yes. It would print 'm' of 'Sundaram'.

Chapter-14
More about pointers


```
***********************************************************************

****************          MORE ABOUT POINTERS       ***********************

***********************************************************************
```


1]   Is the NULL pointer same as an uninitialiseed pointer?
      <Yes / No>

2]   In which header file is the NULL macro defined.

3]   Why is it that for large memeory models NULL has been difined
      as '0L' and for small memory models as just 0?

4]   What is a NULL pointer?

5]   What's the difference between a null pointer, a NULL macre, the
      ASCII NUL character and 'null string'?

6]   What would be the output of the following program?

      ```
      #include "stdio.h"
      main()
      {
            int a,b = 5;
            a = b + NULL;
            printf("%d",a);
      ```

```
        }


7]      Is the programming style adopted in 8.6 good?


8]      What would be the output of the following program?

        #include "stdio.h"
        main()
        {
             printf("%d %d",sizeof(NULL),sizeof(""));
        }


9]      How many bytes are occupied by near, far and huge pointers ?


10]     What does the error "Null Pointer Assignment" mean and what
        causes this error ?


11]     How do we debug a Null Pointer Assignment error ?


12]     Can anything else generate a Null Pointer Assignment error ?


13]     Are the three declarations char **apple, char *orange[], and
        char cherry[][] same ?  <Yes  /  No>


14]     Can two different near pointers contain two different address
        but refer to the same loction in memory?  <Yes   /  No>


15]     Can two different far pointers contain two different address
        but refer to the same location in memeory?   <Yes  /  No>


16]     Can two different huge pointers contain two different addresses
        but refer to the same location in memory?


17]     Would the following program give any warning on compilation ?

        #include"stdio.h"
        main()
        {
             int *p1,i = 25;
             void *p2;
             p1 = &i;
             p2 = &i;
             p1 = p2;
             p2 = p1;
        }
```

18] Would the following program give any warning on compilation ?

```
#include"stdio.h"
main()
{
      int *p1,i = 25;
      void *p2;
      p1 = &i;
      p2 = &i;
}
```

19] What warning would be generated on compiling the following program ?

```
main()
{
      char far *scr;
      scr = 0xB8000000;
      *scr = 'A';
}
```

20] How would you eliminate the warning generated on compiling the following program?

```
main()
{
      char far *scr;
      scr = 0xB8000000;
      *scr = 'A';
}
```

21] How would you obtain a 'far' address from the segment and offset addresses of a memory location ?

22] How would you obtain segment and offset addresses form a 'far' address of a memory location ?

23] In a large data model(compact, large, huge) all are 32 bits long, whereas in a small data model(tiny,small,medium) all pointers are 16 bits long.   < True /   False>

24] A 'near' pointer uses the contents of CS register (if the pointer is pointing to data) for the segement part, whereas the offset part is stored in the 16-bit 'near' pointer.
<True / False>

25] What would be the output of the following program ?

```
main()
```

```
        {
                char far *a = 0x00000120;
                char far *b = 0x00100020;
                char far *c = 0x00120000;

                if(a == b)
                printf("\n Hello");
                if(a == c)
                printf("Hi");
                if(b == c)
                printf("\n Hello Hi");
                if(a > b && a > c && b > c)
                printf("\n Bye");
        }


26]    main()
        {
                char huge *a = 0x00000120;
                char huge *b = 0x00100020;
                char huge *c = 0x00120000;

                if(a == b)
                printf("\n Hello");
                if(a == c)
                printf("Hi");
                if(b == c)
                printf("\n Hello Hi");
                if(a > b && a > c && b > c)
                printf("\n Bye");
        }
```

****

************************ ANSWERS   *****************************

****

1]   No

2]   In files "stdio.h" and "stddef.h"

3]   Because in small memeory models the pointer is two bytes long
     whereas in large memory models it is 4 bytes long.

4]   For each pointer type (like say a char pointer) C defines a
     special pointer balue which is guarenteed not to point to any
     object or function of that type. Usually, the null pointer
     constant used for representing a null pointer is the integer 0.

5]   A null pointer is a pointer which doesn't point anywhere.
     A NULL macro is used to represent the null pointer in source
     code. It has a value 0 associated with it.
     The ASCII NUL character has all its bits as 0 but doesn't have
     any relationaship with the null pointer.
     The null string is just another name for an empty string " ".

6]   5

7]   No.   Only in context of pointers should NULL and 0 be
     considered equivalent. NULL should not be used when other kind
     of 0 is required. Even though this may work it is a bad style
     of programming. ANSI C permits deinition of the NULL macro as
     ((void *)0), which ensures that the NULL will not work in non-
pointer contexts.

8]   2 1

9]   A 'near' pointer is 2 bytes long whereas a far and a 'huge'
     pointer are 4 bytes long.

10] The Null Poiner Assignment error is generated only in small and
     medium memory models. This error occurs in programs which
     attempt to change the bottom of tahe data segment.

         In Borland C and C++ compilers, Borland places four   zero
bytes at the bottom of the data segment, followed by the   Borland
copyright notice " Borland C++ – Copyright 1991 Borland   Intl.". In
the small and medium memory models, a null pointer   points to DS:
0000. Thus assigning a value to the memory   referenced by this
pointer will overwrite the first zero byte   in the data segment. At
program termination, the four zeros and   the copyright banner are
checked. If either has beenm modified,   then the Null Pointer
Assignment error is generated. Note that the pointer may not truly be
null, but may be a wild pointer   that references these key areas in
the data segment.

11] In the Integrated Development Environment set two watches on
     the key memory locations mentioned in 8.10. These watches, and
     what they should display in the watch window, are:
     *(char *)4,42MS   Borland C++ – Copyright 1991 Borland Intl."

```
(char *)0
```

Of course, the copyright banner will vary depennding on your version of the Borland C/C++ compiler.

Step through your program using F8 or F7 and monitor these values in the watch window. At the point where one of them changes, you have just executed a statement that used a pointer that has not been properly initialized.

The most common cause of this error is probably declaring a pointer and then using it before allocating memory for it. For example, compile the following program in the small memory model and execute it:

```
#include "dos.h"
#include "stdio.h"
#include "string.h"
main()
{
        char *ptr,*banner;
        banner = ( char *) MK_FP(_DS,4);
        printf("banner:%s\n",banner);
        strcpy(ptr,"the world cup saga");
        printf("&ptr = %Fp \n",(void far*) &ptr[0]);
        printf("banner:%s \n",banner);
}
```

One of the best debugging techniques for catching Null pointer assignment errors is to turn on all warinin compiler messages. If the above program is compiled with warnings turned off, no warnin g messages will be generated. However, if all warnings are turned on, both the strcpy() and printf() calls using the ptr bariable will generate warnings. YOu should be particularly suspicious of any warnings that a variable might be used before being initialzed, or of a suspicious pointer assignment.

Note that a Null Pointer Assignment error is not generated in all midels. In the compact, large and huge memory models, far pointers are used for data. Therefore, a null pointer will reference 0000:0000, or the base of system memory, and using it will ot cause a corruption of the key balues at the base of the data segment. Modifying the base of system memory usually causes a system crah, however. Athough it would be possible that a wild pointer would ovrewrote the key walues, it would not indicate a null pointer. In the tiny memory model, DS = CS = SS. Therefore, using a null pointyer will overwrite the beginning of the code segment.

12}   Yes, using a wild pointer that happens to regerence the base area of the data segment may cause the same error since this would change the zeros or the copyright banner. Since data corruption or stack corruption could cause an otherwise-valid pointer to be corrupted and point to the base of the data segment, any memory corruption could result in this error being generated. If the pointer used in the program statement which

corrupts the key values appears to have been properly   initialized,
place a watch on that pointer. Step through your   program again and
watch for its value (address) to change.

13]   No.

14]   No.

15]   Yes.

16]   No.

17]   No.

18]   Yes. Suspicious pointer conversion in function main.

19]   Non-portable pointer assignment in function main.

20]   Use the typecast scr = (char far *) 0xb8000000;

21]   # include "dos.h"
      main()
      {
            char *seg = (char *) 0xb000;
            char *off = (char *) 0x8000;
            char far *p;
            p = MK_FP(seg,off);
      }

22]   # include "dos.h"
      main()
      {
            char far *scr = (char *) 0xb8000000;
            char *seg,*off;
            seg = (char*) FP_SEG(scr);
            off = (char *) FP_OFF(scr);
      }

23]   True.

24]   True.

25]   Bye

Here a,b and c reger to same location in memory still
the first three ifs fail because while comparing the are
pointers using ==(and !=) the full 32-bit value is used and
since the 32-bit balues are different the ifs fail. The last if
however gets satisfied, because while comparing using > (and
>=, < , <=) only the offset value is used for comparsion. And
the offset values of a,b and c are such that the last condition
is satisfied.

26]   Hello
      Hi
      Hello Hi
      Unlike far pointers, huge pointer are 'normalized' to
      avoid the strange behaviour as in the 25th above. A normalized
      pointer is a 32-bit pointer which has as much of its value in
      the segment address as possible. Since a segment can start every
16 bytes, this means that the offset will only have a   value for 0
to F. Huge pointers are always kept normalized. As   a result for any
given memory address there is only one   possible huge address –
segment: offset pair for it.

Chapter-15
Sample Programs

WILL THE FOLLOWING PROGRAM WORK???

SUPPOSE THAT THE BASE ADDRESS OF i IS 2008 AND THAT OF a IS 7006. THE
ADDRESS OF ii IS 7602 AND THAT OF aa IS 9118.

```
    /*    PROGRAM     */
main()
{
     int i = 54;
     float a = 3.14;
     char *ii,*aa;

     ii = &i;
     aa = &a;
     printf("\n address cintained in ii = %u",ii);
     printf("\n address cintained in aa = %u",aa);
     printf)"\n value at the address contained in ii = %d",*ii);
     printf("\n value at the address contained in aa = %d",*aa);
}
```

ANSWER:
     Here ii and aa have been declared as char pointers. Still the
statements ii = &i and aa = &a will work. Once again the addresses 2008
and 7006 get stored in ii and aa which are printed through the first

two printf()s. However, the porgram flaters at the next two
printf()s.This is so since ii is a character pointer *ii gives the
value at the address 2008 and dot the one present at 2008 and
2009.Similarly, *aa gives the value at 7006 and not the one comtained
in 7006,7007,7008 and 7009. THIS IS BECAUSE A CHARACTER VARIBLE
OCCUPIES ONLY ONE MEMORY LOCATION WHEN WE ACCESS THEM, ONLY ONE MEMORY
LOCATION IS REFERRED TO.

    From this we come to know that iv we wish to access an integer
value stored in a variable using its address it is necrssary that the
address be stored in an integer pointer. Likewise if we wish to accrss
a float value stored in a bariable using its address it is necessary to
store the address in a float pointer.

PASSING ADDRESSES TO FUNCTIONS:

    There are 2 methods of doing so. They are
    1)   call by value(sending the values of the arguements).
    2)   call by reference(dending the addresses of the
    arguements.)

1)   In the first menthod i.e. "call by value", the 'value' of each
arguement in the calling function is copied into corresponding formal
arguenents of the called fuction. With this method changes made to the
formal arguenents in the called fuction will have no effect on the
values of the actual arguements in the calling function.

EXAMPLE:
THE FOLLOWING PROGRAM ILLUSTRATES THE ' CALL BY VALUE'.


```
     /*    PROGRAM    */
main()
{
     int a = 10;
     int b = 20;
swapv(a,b);
printf(" \n a = %d",a);
printf(" \n b = %d",b);
}

/* fuction subprogram    */
swapv(int x,int y)
{
     int t;
     t = x;
     x = y;
     y = t;
     printf("\n x = %d",x);
     printf("\n y = %d",y);
}
```

OUTPUT:
     x = 20
     y = 10

```
        a = 10
        b = 20
NOTE:        Here the values of a and b aremain unchanged even after
exchanging the values of x and y.


2)      In the second method i.e. "call by reference" the address of the
actual arguements in the calling function are copied into the formal
aruements of the called function. This means that using the formal
arguements in the called function we can make changes in the actual
arguements of the calling function. This method is faster than call by
value method.


EXAMPLE:
        THE FOLLOWING PROGRAM ILLUSTRATES THIS FACT:

        /* PROGRAM  */
main()
{
        int a = 10;
        int b = 20;
swapr(&a,&b);
printf(" \n a = %d",a);
printf(" \n b = %d",b);
}

        /* fucntion subprogram  */
swapr(int *x,int *y)
{
        int t;

        t = *x;
        *x = *y;
        *y = t;
printf("\n x = %d",x);
printf("\n y = %d",y);
}

OUTPUT:
        a = 20
        b = 10
        x = 20
        y = 10


SOLVED PROBLEMS:

WHAT WILL BE THE OUTPUT OF THE FOLLOWING PROGRAMS:

FIRST PROGRAM

1]      main()
        {
                int i = -5,j = -2;
                junk(i,&j);
                printf("\n i = %d  j = %d",i,j);
```

```
}

        /*     subprogram   */
        junk(int i, int *j)
        {
                i = i * i;
                *j = *j * *j;
        }

OUTPUT:
        i = -5    j = 4


EXPLANATION:      One doubt immediately that comes to the mind is that
" can we use same variables in different functions?". Yes, by all means,
without absolutely any conflict. Thus, the two sets of 'i' and 'j' are
two totally different sets of variables. While calling the fuction
'junk()' the value of i and the address of j are passed to it.
Naturally, in 'junk()' 'i' is declared as an ordinary int, whereas 'j'
is dexlared as a pointer to an 'int'.

        Even though the value of 'i' is changed to 25 in junk(), this
change will not be reflexted back in main(). As against this, since 'j'
s' address is being passed to 'junk()', any change in 'junk()' gets
reflected back in 'main()'.


SECOND PROGRAM

2]     main()
{
        int *c;
        c = check(10,20);
printf(\n c = %u",c);
}

/*    subprogram   */
        check(int i, intj)
        {
                int *p,*q;
                p = &i;
                q = *j;
                if( i >= 45)
                return(p);
                else
                return(q);
        }

OUTPUT:
        error message:non portable pointer assignment in main.

EXPLANATION:      The reason for the error is simple. The integers bing
passed to 'check()' are collexted in 'i' and 'j', and then their
addresses are assigned to 'p' and 'q'. Then in the next statement the
conditional operators test the value of 'i' against 45, and return
return either tha address stored in 'p' or the address stored in'q'. It
appears that this address would be collected in 'c' in 'main()', and
```

then wouldbe printed out. And there lies the error. The function
'check()' is not capable of returning an integer pointer. All that it
can return is an ordinary integer. Thus just declaring 'c' as an
integer ponter is not sufficient. We must make the following
modifications is the program to make it work properly.

MODIGFIED PROGRAM:

```
main()
{
      int *c:
      int *check();
      c = check(10,20);
      printf("\n c = %u ",c);
}

/* subprogram  */
int *check(int i,int j)
{
      int *p,*q;
      p = &i;
      q = &j;
      if(i >= 45)
      return(p);
      else
      return(q);
}
```

4]   Assume that the value of p is stored at 5498.
```
main()
{
      float *jamboree();
      float p = 23.5,*q;
      q = &p;
      printf(" \n before call = %u",q);
      q = jamboree(&p);
      printf(" \n after call = %u",q);
}

/* subprogram  */
float *jamboree(float *r)
{
      r = r + 1;
      return(r);
}
```

OUTPUT:
      q before call = 5498
      q after call = 5502

EXPLANATION: In 'main()', q has been declared as a float pointer.
It means 'q' is a variable capable of holding the address of a float.
Through 'q = &p' the address of 'p', is stored in q and then pronted
out through the printf(). This is the value of 'q' before 'jamporee()'
is called. When 'jamboree()' is called the address of p is sent to it
and collected in 'r'. At this juncture 'r' contains 5498(when we ran

the program it was 5498; when you execute the program this may turn out
be some other address). When 'r' is incremented it would become 5502.
why a step of 4? This is because 'r' is a float pointer and in
incrementing it by 1 it would point to the next float which would be
present 4 bytes hence, since every float is 4 bytes long. The return
statement then returns this address 5502 back to 'main()'.

   Since a float pointer is being returned, a decalration float
'*jamboree()' is necessary in 'main()', which tells the compoler that
down the line there exists a function called 'jamboree()', which will
return a float pointer.

 ****************   POINTERS AND ARRAYS   *****************

         To be able to see what pointers have got to do with arrays,
let us first learn some pointer arithmetic. Consider the following
example:

```
     /*  program   */
    main()
    {
         int i = 3,*x;
         float j = 1.5,*y;
         char k = 'c',*z;

    printf("\n value of i = %d",i);
    printf("\n value of j = %f",j);
    printf("\n value of k = %c",k);

    x = &i;
    y = &j;
    z = &k;

    printf("\n\n original value in x = %u",x);
    printf("\n original value in y = %u",y);
    printf("\n original value in z = %u",z);

    x++;
    y++;
    z++;

    printf("\n new value in x = %u",x);
    printf("\n new value in y = %u",y);
    printf("\n new value in z = %u",z);
    }
```

SOLUTION: suppose 'i','j' and 'k' are stored in memeory at addresses
1002, 2004 and 5006, the output would be:
    value of i = 3
    value of j = 1.500000
    value of k = c

    original value in x = 1002
    original value in y = 2004
    original value in z = 5006

```
       new value in x = 1004
       new value in y = 2008
       new value in z = 5007
```

      Observe the last three lines of the outut. 1004 is original value
in x plus 2, 2008 is original value in y plus 4, and 5007 is original
balue in z plus 1.This so happens because every time a pointer is
incremented it points to the immediately next location of its type.
That is why, when the integer pointer x is incremented, it points to an
address two locations after the current location, since an int is
always 2 bytes long. Similarly 'y' points to an address 4 locations
after the current location and 'z' points 1 location after the current
location. This is a very important result and can be effectively used
while passing the entire array to a function.

      The way a pointer can be incremented, it can be decremented as
well, to earlier locations. Thus, the following operations can be
performed on a pointer:
1]   addition of a number to a pointer.

```
       int i = 4,*j,*k;
       j = &i;
       j = j + 1;
       j = j + 9;
       k = j + 3;
```

2]   subtraction of a number from a pointer.

```
       int i = 4, *j,*k;
       j = &i;
       j = j - 2;
       j = j - 5;
       k = j - 6;
```

CAUTION: Do not attempt the following operations on pointers... they
would never work out.
(a)   addition of two pointers.
(b)   Multiplying a pointer with a number.
(c)   Dividing a pointer with a number.

ADVANTAGE: While handling arrays, accessing array elements by pointers
is always faster than accessing them by subscripts.

/********************* SAMPLE PROGRAMS   *******************/

WHAT WILL BE THE OUTPUT OF THE FOLLOWING PROGRAMS:

```
1]    main()
      {
            int a[] = {10,20,30,40,50};
            int j;
            for(j = 0; j < 5; j++)
            {
                  printf("\n %d ",*a);
                  a++;
            }
```

```
        }
```

OUTPUT:
        ERROR MESSAGE: Lvalue required in function main

EXPLANATION: Whenever we mention the name of the array, we get its
base address. Therefore, first time through the loop, the 'printf()'
should print the value of this base address. There is no problem upto
this. The problem lies in the next statement, 'a++'. Since C doesn;t
perform bounds checking on an array, once declared is its base address.
And 'a++' attempts to change this base address, which C won't allow
because if it does so, it would be unable to remember the beginning of
the array. Anything which can change – in compiler's language – is
called Lvalue. Since vbalue of a cannot be changed through ++, it
flashes the reeor saying 'Lvalue required' so that ++ operator can
change it.

```
2]    main()
      {
            float a[] = { 13.24,1.5,1.5,5.4,3.5};
            float *j,*k;
            j = a;
            k = a + 4;
            j = j * 2;
            k = k/2;
            printf("\n %f %f",*j,*k);
      }
```

OUTPUT:
        Error message: illegal use of pointer in function main

EXPLANATION: 'j' and 'k' have been decalred as pointer variables,
which would contain the addresses of floats. In other words, 'j' and
'k' are float pointers. To begin with, the base address of the array a[]
is stored in 'j'. The next statement is perfectly acceptable; the
address of the 4th float form the base address is dtored in 'k'. The
next two statements aree erroneous. This is because the only operations
that can be performed on pointers are addition and subtraction.
Multiplication or division of a pointer is not allowed. Hence the error
message.

```
3]    main()
      {
            int n[25];
            n[0] = 100;
            n[24] = 200;
            printf("\n %d %d",*n,*(n + 24) + *(n + 0));
      }
```

OUTPUT:
        100 300

EXPLANATION: 'n[]' has been declared as an array capable of
holding 25 elements numbvered form 0 to 24. Then 100 and 200 are
assigned to 'n[0]' and 'n[24]' rexpectively. Then comes the most
important part– the 'printf()' statement. Whenever we mention the name

of the array, we get its base address(i.e. address of the '0th' element
of the array). Thus,'*n' would give the value at this base address,
which in this case is 100. This is then printed out. Take a look at the
next expression, '*(n + 24) + *(n + 0)'.
                    'n' gives the address of the zeroth element, 'n = 1'
gives the address of the next element of the array, and so on. Thus,
'*(n+24)' would give the value at this address, which is 200 in our
case. Similarly, '*(n + 0)' would give 100 and the addition of the two
would result into 300, which is outputted next.

4]   main()
      {
            int b[] = {10,20,30,40,50};
            int i,*k;
            k = &b[4] - 4;
            for(i = 0; i <= 4; i++)
            {
                  printf("%d",*k);
                  k++;
            }
      }

OUTPUT:
      10 20 30 40 50

EXPLANATION: The elements of the array are stored in contiguous
memory locations and each element is an integer, hence is occupying 2
locations.

            ARRAY ELEMENT   VALUE   MEMORY LOCATION

                  b[0]   10   4002
                  b[1]   20   4004
                  b[2]   30   4006
                  b[3]   40   4008
                  b[4]   50   4010

        The expression '&b[4]' gives the addressof 'b[4]'(4010 in the
above case). From this address if we subtract 4, we get 4002. Or did
you expect to get 4006? Remembre that by subtraction 4 form 4010 what
we mean is: get the address of an integer which is 4 integers to the
left of the integer whose address is 4010. Now, address of the integer
which is 4 integers to the left of the integer whose address is 4010,
i.e the address 4002. This address, 4002, is stored in k, which has
been declared as a variable capable of holding an integer's address.
First time through the for loop '*k' would result into 10, i.e. value
at the address in 'k'. 'k++' then increments k such that it contains
the address of the next integer, i.e. 4004, which is 20. Similarly, the
loop prints out the rest of the elements of the array.

5]   main()
      {
            int a[] = {2,4,6,8,10};
            int i;

```
              for(i = 0; i <= 4; i++)
              {
                      *(a + i) = a[i] + i[a];
                      printf("%d",*(i + a));
              }
      }
```

OUTPUT:
      4 8 12 16 20

EXPLANATION: Imbibe the following three facts and the
program becomes very simple to understand:

      (a)   Mentioning the name of the array gives the base address
      of the array.
      (b)   Array elements are stored in contiguous memory
      locations.
      (c)   On adding 1 to the address of an integer, we get the
      address of the next integer.

              With those facts clearly laid out, let us now try to
understand the program. Remember that internally C always accesses
array elements using pointers. Thus, when we say a[i] internally C
converts it to '*(a + i)', which means value of ith integer from the
base address. Now, if the expression 'a[i]' is same as '*(a + i)' then
'*(i + a)' must be same as 'i[a]'. But '*(a + i)' is same as
'*(i + a)'. Therefore 'a[i]' must be same as 'i[a]'.thus 'a[i]',
'*(a + i)', '*(i + a)' and 'i[a]' refer to the same element- 'i'th
element form the base address.

      Therefore the expression used in the for loop,   '*(a
+ i)' = 'a[i]' + 'i[a]' is nothing but 'a[i]' = 'a[i]' + 'a[i]'. thus
all that is done in the for loop is each array element is doubled and
then printedout through 'printf()'.

```
6]    main()
      {
              int a[5] = {2,4,6,8,10};
              int i,b = 5;
              for(i = 0; i < 5; i++)
              {
                      f(a[i],&b)
                      printf("\n %d %d",a[i],b);
              }
      }
      f(int x,int *y)
      {
              x = *(y) += 2;
      }
```

OUTPUT:
      2 7
      4 9
      6 11
      8 13
      10 15
```

EXPLANATION: After initialising the array when the control enters
the for loop, the function f() gets called with value of a[i] and the
address of b. In f() these are collected in variables x and y. then
comes the expression x = *(y) += 2. Here *(y) += 2 is evaluated forst
and then the result of this expression is assigned to x. The first time
through the for loop *(y) gives 5, to which 2 is added and the result
is stored at *(y). It means 7 is addigned to b. Finally, the = oerator
assigns 7 to x. However, on assigning a new value to x, the array
element a[0] in main remains unchanged. Thus, during every call to f(),
b's value keeps getting updated whereas there is no change in the
values of the array elements.

    array element a[0]   b

    value   2   5

    memory location   4008

            x   y

            2   4008

7]   main()
      {
            int a[5] = {2,3,4,5,6};
            int i;

      change(a);

Chapter-16
Strings

****

*************************   STRINGS   ****************************

****

1]   What would be the output of the following program ?

      main()
      {

```
        printf(5 + "Fascimile");
        }
```

OPTIONS:
        (a)   Error
        (b)   Fascimile
        (c)   mile
        (d)   None of the above

2]   What would be the output of the following program ?

```
        main()
        {
            char str1[] = "Hello";
            char str2[] = "Hello";
            if (str1 == str2)
                    printf("\n Equal");
            else
                    printf("\n Unequal");
        }
```

OPTIONS:
        (a)   Equal
        (b)   Unequal
        (c)   Error
        (d)   None of the above

3]   What would be the output of the following program ?

```
        main()
        {
            printf("%c","abcdefgh"[4]);
        }
```

OPTIONS:
        (a)   Error
        (b)   d
        (c)   e
        (d)   abcdefgh

4]   What would be the output of the following program ?

```
        main()
        {
            char str[7] = "Strings";
            printf("%s",str);
        }
```

OPTIONS:
        (a)   Error
        (b)   Strings
        (c)   Cannot predict
        (d)   None of the above
```

5]    How would you output '\n' on the screen ?


6]    What would be the output of the following program ?

```
main()
{
      char ch = 'A';
      printf("%d %d",sizeof(ch),sizeof('A'));
}
```

OPTIONS:
      (a)   1   1
      (b)   1   2
      (c)   2   2
      (d)   2   1

7]   What would be the output of the following program ?

```
main()
{
      printf("\n%d %d %d",sizeof('3'),sizeof("3"),sizeof(3));
}
```

OPTIONS:
      (a)   1   1   1
      (b)   2   2   2
      (c)   1   2   2
      (d)   1   2   1

8]   Is the following program correct ? < Yes / No>

```
main()
{
      char *str1 = "United";
      char *str2 = "Front";
      char *str3;
      str3 = strcat(str1,str2);
      printf("\n %s",str3);
}
```


9]   How would you improve the code in (8) above ?

10]   In the following code which function would get called, the user-
defined strcpy() or the one in the standard library ?

```
main()
{
      char tr1[] = "keep India Beautiful ...emigrate!");
      char str2[40];
      strcpy(str2,str1);
      printf("\n %s",str2);
```

```
        }

        strcpy(char *t,char *s)
        {
              while(*s)
              {
                    *t = *s;
                    t++;
                    s++;
              }
              *t = "\0";
        }
```

11]   Can you compact the code in strcpy() into one line ?

12]   Can you compact the code in strcpy() into one line ?

```
        main()
        {
              char *str[] = {"Frogs","Do","Not","Die.","They","Crock!"};
              printf("%d %d",sizeof(str),sizeof(str[0]));
        }
```


13]   How would you find the length of each string in the program
       (12) above ?

14]   What is the difference in the following declarations ?

```
        char *p = "Samuel";
        char a[] = "Samuel";
```


15]   While handling a string do we always have to process it
       character by character or there exists a method to process the
       entire string as one unit.

****

*********************   ANSWERS   ****************************

****

1]   C
```

2]   B

3]   C

4]   C.   Here 'str[]' has been declared a 7 character array and
      into it a 8 character string has been stored. This would result
      into overwriting of the byte beyond the seventh byte reserved
      for the array with a '\0'. There is always a possibility that
      something important gets overwritten which would be unsafe.

5]   printf("\\n");

6]   B

7]   B

8]   No, since what is present in memory beyond 'United' is not known
and we are attaching 'Front' at the end of 'United', thereby
overwriting something, which is an unsafe thing to do.

9]   main()
      {
            char str1[15] = "United";
            char *str2 = "Front";
            char *str3;
            str3 = strcat(str1,str2);
            printf("\n %s",str3);
      }

10]   User-defined tyrcpy()

11]   strcpy(char *t,char *s)
      {
            while(*t++ = *s++)
      }

12]   12 2

13]   main()
      {
            char *str[] = {"Frogs","Do","Not","Die","They","Croak!");
            int i;

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

```
                printf("%s %d",str[i],strlen(str[i]));
        }
```

14]   Here 'a' is an array big enough to hold the message and
        the '\0' following the message. Individual characters within
        the array can be changed but the address of the array would
        remain same.
               On the other hand, 'p' is a pointer, initialized to   point
to a string constant. The pointer 'p' may be nodified to   point to
another string, but if you attempt to modify the   string at which
'p' is pointing the result is undefined.

15]   A string can be processed only on a character by character basis.

Chapter-17
Structure Union and Enumerator

        ****

        *************   STRUCTURES,UNIONS AND ENUMERATIONS   ****************

        ****

1]   What is the similarity between, union and an enumeration ?

2]   Would the following declaration work ?

      typedef struct s
      {
            int a;
            float b;
      } s;

3]   Can a structure contain a pointer to itself ?

4]   Point out the error, if any, in the following code:

      typedef struct
      {
            int data;
            NODEPTR link;
      } *NODEPTR;

5]   How will you eliminate the problem in (4) above ?

6]   Point out the error, if any, in the following code:

```
void mocify(struct emp*);
struct emp
{
      char name[20];
      int age;
};

main()
{
      struct emp e = {"Sanjay",35};
      modify(&e);
      printf("\n %s %d",e.name, e.age);
}

void modify(struct emp *p)
{
      strupr(p->name);
p->age = p->age + 2;
}
```


7]   Would the folowing code work ?


```
#include<calloc.h>
struct emp
{
      int len;
      char name[1];
};

main()
{
      char newname[] = "Rahul";
      struct emp *p = (struct emp *)malloc(sizeof(struct emp)
            -1 + strlen(newname) + 1);
      p->len = strlen(newname);
      strcpy(p->name,newname);
      printf("\n %d %s",p->len,p->name);
}
```


8]   Can you suggest a better way to write the program in (7) above?

9]   How would you free the memory allocated in (8) above ?

10]   Can you rewrite the progaram in (8) such that while freeing the
       memory only one call to free() would suffice?

11] What would be the output of the following program ?

```
main()
{
    struct emp
    {
        char *n;
        int age;
    };
    struct emp e1 = {"Dravid",23};
    struct emp e2 = e1;
    strupr(e2.n);
    printf("\n %s",e1.n);
}
```

12] Point out the error, if any, in the following code :

```
main()
{
    struct emp
    {
        char n[20];
        int age;
    };
    struct emp e1 = {"Dravid",23};
    struct emp e2 = e1;
    if(e1 == e2)
        printf("\n the structures are equal.");
}
```

13] How would you check whether the contents of two structure variables are same or not ?

14] How are structure passing and returning implemented by the compiler ?

15] How can I read/ write structures from/to data files ?

16] If the following structure is written to a file using fwrite(), can fread() read it back successfully?

```
struct emp
{
    char *n;
    int age;
};
struct emp e = {"Sujay",15};
FIlE *fp;
fwrite(&e,sizeof(e),1,fp);
```

17] Would the following program always output the size of the
     structure as 7 bytes ?

```
struct ex
{
     char ch;
     int i;
     long int a;
};
```

18] What error does the following progarm give and what is the
     solution for it?

```
main()
{
     struct emp
     {
          char name[20];
          float sal;
     };
struct emp e[10];
int i;
for(i = 0;i <= 9; i++)
     scanf("%s %f",e[i].name,&e[1].sal);
}
```

19] How can I determine the byte offset of a field within a
     structure ?

20] The way mentioning the array name of function name without []
     or () yields their base addresses, what do you obtain on
     mentioning the structure name ?

21] What is main() returning in the following progarm ?

```
struct transaction
{
     int sno;
     char desc[30];
     char dc;
     float amount;
}
/*  here is the main progarm */

main(int argc, char *argv[])
{
     struct transaction t;
     scanf("%d %s %c %f",&t.sno,t.desc,&t.dc,&t.amount);
     printf("%d %s %c %f",t.sno,t.desc,t.dc,t.amount);
}
```

22] What would be the output of the following progarm ?

```
main()
{
    struct a
    {
        category : 5;
        scheme : 4;
    };
    printf("size = %d",sizeof(struct a));
}
```

23] What's the difference between a structure and a union ?

24] Is it necessary that size of all elements in a union should be same ?

25] Point out the error, if any, in the following code:

```
main()
{
    union a
    {
        int i;
        char ch[2];
    };
    union a z1 = {512};
    union a z2 = {0,2};
}
```

26] What is the difference between an enumeration and a set of proprocessor # defines ?

27] Since enumerations have integral type and enumeration constants are of type int can we freely intermix them with other integral types, without errors ?   < Yes / No>

28] Is there an easy way to print enumaeration values symbolically?

29] What is the use of bit fields in a structure declaration ?

30] Can we have an array of bit fields ? <Yes / No>

```
*********************************************************************

************************     ANSWERS     *****************************

*********************************************************************



1]    All of them let you define new data types.


2]    Yes


3]    Certainly. Such structures are known as self-refertial
      structures.


4]    A 'typedef' defines a new name for a type, and in simpler cases
      like the one shown below you can define a new structure type
      and a 'typedef' for it at the same time.

      typedef struct
      {
            char name[20];
            int age;
      }emp;

            However, in the structure defined in (4) there is an  error
because a typedef declaration cannot be used until it is    defined. In
the given code fragment the 'typedef' declaration     is not yet
defined at the point where the link field is    declared.


5]    To fix the code,first give the structure a name("struct node").
      Then declare the link field as a simple struct node * as shown
      below:

      typedef sturct node
      {
            int data;
            struct node *link;
      }*NODEPTR;

            Another way to eliminate the problem is to disentangle the
typedef declaration from the structure definition as shown below:

      struct node
      {
            int data;
            struct node *link;
      };
      typedef struct node *NODEPTR;
```

Yet another way to eliminate the problem is to precede the structure declaration with the typedef, in which case you could use the NODEPTR typedef when declaring the link field as shown below:

```
typedef struct node *NODEPTR;

struct node
{
    int data;
    NODEPTR next;
};
```

In this case, you declare a new typedef name involving struct node even though struct node has not been completely defined yet; this you're allowed to do. It's a matter of style which of the above solutions would you prefer.

6]   The 'stuct emp' is mintioned in the prototype of the function modify() before defining the structure. To solve the problem just put the prototype after the declaration of the structure or just add the statement 'struct emp' before the prototype.

7]   Yes. The program allocates space for the structure   with the size adjuct so that the name field can hold the   requested name (not just one character, as the structure declaration would suggest). I don't know whether it is legal or   portable. However, the code did work on all the compilers that   I have tried it with.

8]   The truly safe way to implement the program is to use a character pointer instead of an array as shown below:

```
#include <alloc.h>
struct emp
{
    int len;
    char *name;
};

main()
{
    char newname[] = "Rahul";
    struct emp *p = (struct emp *)malloc(sizeof(struct emp));
    p->len = strlen(newname);
    p->name = malloc(p->len + 12);
    strcpy( p->name,newname);
    printf("\n %d %s",p->len,p->name);
}
```

Obviously, the "convenience" of having the lenght and the string stored in the same block of memory has now been lost, and freeing instances of this structure will reqwuire two   calls to the function free().

```
9]     free(p->name);
       free(p);


10]    #include <alloc.h>
       struct emp
       {
             int len;
             char *name;
       };

       main()
       {
         char newname[] = "Rahul";
         char *buf = malloc(sizeof(struct emp) + (strlen(newname + 1);
         struct emp *p = (struct emp *)buf;
         p->len = strlen(newname);
         p->name = buf + sizeof(struct emp);
         strcpy(p->name, newname);
         printf("\n %d %s",p->len,p->name);
         free(p);
       }
```

11]   DRAVID
             When a sturcture is assigned, passed, or returned, the
       copying is done monolithically. This means that the copies of
       any pointer fields will point to the same place as the
       original. In other words, anything pointed to is not copied.
       Hence, on changing the name through 'e2.n' is automatically
       changed 'e1.n'.

12]   Sturctures can't be compared using the built-in '==' and
'!=' operations. This is because there is no single, good   way for a
compiler to implemenmt structure comparison. A simple   byte-by-
byte comparison could fail while comparing the bits   present in unused
paddings in the structure (such padding is   used to keep the
alignment of later fields correct). A   field-by-field
comparison might require unacceptable amounts of   repetitive code
for large sturctures. Also, any   compiler-generated comparison could
not expected to compare   pointer fields appropriately in all cases;
for example, it's   often appropriate to compare 'char *' fields
with 'strcmp()'   rather than with '=='.


```
13]    struct emp
       {
             char n[20];
             int age;
       };

       main()
       {
             struct emp e1 = {"Dravid",23};
```

```
        struct emp e2;
        scanf("%s %d",e2.name,&e2.age);
        if(structcmp(e1,e2) == 0)
            printf("the structures are equal");
        else
            printf("the structures are not equal.");
}

structcmp(struct emp x, struct emp y)
{
        if (strcmp(x.n,y.n) == 0)
            if(x.age == y.age)
                return(0);

        return(1);
}
```

In short, if you need to compare two sturctues, you'll have to write your own function to do so which carries out the comparison field by field.

14]   When structures are passed as arguements to functions, the entire structure is typically pushed on the stack. To avoid this overhead many programmers often prefer to pass pointers to structures instead of actual structures. Structures are often returned from functions in a location pointed to by an extra, compiler-supplied 'hidden' argument to the function.

15]   To write out a sturcture we can use 'fwrite()' as shown
        fwrite(&e,sizeof(e),1,fp);
        where e is a structure variable. A corresponding 'fread()' invocation can read the structure back form a file.
        On calling 'fwrite()' it writes out 'sizeof(e)' bytes from the address '&e'. Data files written as memory images with 'fwrite()', however, will not be portable, particularly if they contain floating-point fields or pointers. This is because memory layout of structures is machine and compiler dependent.   Different compilers may use different amounts of padding, and   the sizes and byte orders of fundamental types vary across   machines. Therefore, structures written as memory images   cannot necessarily be read back in by programs running on other   machines (or even compiled by other compilers), and this is an   important concern if the data files you're writing will ever be   interchanged between machines.

16]   No, since the structure contains a 'char' pointer while writing the structure to the disk using 'fwrite()' only the value stored in the pointer 'n' would get written (and not the string pointed by it). When this structure is read back the address would be read back but it is quite unlikely that the desired string would be present at this address in memory.

17]   No.   A compiler may leave holes in structures by padding the first 'char' in the sturcture with another byte

just to ensure that the integer that follows is stored at an
even location. Also there might be two extra bytes after the
integer to ensure that the long integer is stored at an address
which is a multiple of 4. This is done because many machines
access values in memory most efficiently when the values are
appropriately aligned. Some machines cannot perform unaligned
accesses at all and require that all data be appropriately
aligned.

Your compiler may provide an extension to give you
control over the packing of structures(i.e., whether they are
padded), perhaps with a #pragma, but there is no standard
menthod.

If you're worried about wasted space, you can minimise
the effects of padding by ordering the members of a structure
from largest to smallest. You can sometimes get more control
over size and alignment by using bitfields, although they have
their own drawbacks.

18]   Error: Floating point formats not linked. What causes this error
to occur? When the compiler encounters a reference to   the address
of a 'float', it sets a flag to have the linker   link in the
floating point emulator.

A floating point emulator is used to manupilate
floating point numbers in runtime library functions like
'scanf()' and 'atof()'. There are some cases in which the
reference to the 'float' is a bit obscure and the compiler does
not detect the need for the emulator.

These situations usually occur during the initial
stages of program development. Normally, once the progaram is
fully developed, the emulator will be used in such a fashion
that the compiler can accurately determine when to link in the
emulator.

To force linking of the floating point emulator into an
application, just include the follwoing function in your
program:

```
void LinkFloat(void)
{
        float a = 0, *b = &a; /* cause emulator to be linked */
        a = *b; /* suppress warning - bar not used */
}
```

There is no need to call this function from your
program.

20]   You can use the offset macro given below. How to use this
macro has also been shown in the program:

```
#define  offset(type,mem)((int)((char*)&((type*)0)-mem-(char*)(type*)0))
main()
{
      struct a
      {
            char name[15];
            int age;
```

```
            float sal;
        };

        int offsetofname, offsetofage, offsetofsal;
        offsetname = offset(struct a, name);
        printf("\n %d",offsetofname);

        offsetofage = offset(struct a,age);
        printf("\n %d",offsetofage);

        offsetofsal = offset(struct a, sal);
        printf("\n %d",offsetofsal);
        }
```

The output of the program will be
0   15   17

21]   The entire structure itself and not its base address.

22]   A missing semicolon at the end of the structure declarations is
       causing 'main()' to be declared as returning a structure. The
       connection is difficult to see because of the intervening
       comment.

23]   size = 2
            Sin ce we have used bit fields in the structure and the
       total number of bits is turning out to be more than 8(9 bits
       to be precise) the size of the structure is bing reported as 2
       bytes.

24]   A union is essentially a structure in which all of the
       fields overlay each other; you can use only one field at a time.
You can also write to one field and read from another, to   inspect a
type's bit patterns or interpret them differently.

25]   The ANSO C Standard allows an initializer for the first
       member of a union. There is no standard way of initializing any
       other member, hence the error in initilizing z2.
            Many proposals have been advanced to allow more
       flexible union initialization, but none has been adopted yet.
       If you still want to initialise different members of the union,
       with the members in different orders, so that you can declare
       and intitialize the one having the appropriate first member as
       shown below:

       unoin a
       {
            int i;
            char ch[2];
       };

       union b
```

```
        {
                char ch[2];
                int i;
        };

        main()
        {
                union a z1 = {512};
                union b z2 = {0,2};
        }
```

26]   There is hardly any difference between the two, except
        that a '#define' we have to explicitly define them. A
        disadvantage is that we have no control over the sizes of
        enumeration variables.

27]   Yes

28]   No.   You can write a small function (one per enumeration)
to map an enumeration contant to a string, either   by using a switch
statement or by searching an array.

29]   Bitfields are used ato save space in sturctures having
        several binary flags or other small fields. Note that the colon
        notation for specifying the size of a field in bits is valid
        only in structures (and in unions); you cannot use this
        mechanism to specify the size of arbitary variables.

30]   No

Chapter-18
Subtleties of typedef

****

*************   SUBTLETIES OF TYPEDEF   *****************

****

1]   Are tahe properties of i,j and x,y in the folowing program same?
< Yes / No >

```
typedef unsigned ling int uli;
uli i,j;
unsigned long in x,y;
```

2]   What is the type of compare in the following code segment?

```
typedef int (*ptrtofun)(char *,char *);
ptrtofun compare;
```

3]   What are the advantages of using typedef in a program?

4]   Is there any difference in the #define and the typedef in the
      follwoing code? If yes, what?

```
typedef char *string_t;
#define string_d char*
string_t s1,s2;
string_d s3,s4;
```

5]   typedefs have the advantage that obey scope rules, that is,
      they can be declared local to a function or a block whereas
      #defines always have aglobal effect. < True / False>

6]   Point out the error in the following decalrations and suggest
      atleast there solutions for it.

```
tyupedef struct
{
      int data;
      NODEPTR link;
} *NODEPTR;
```

7]   In the following code can be delcare a new typedef name emp
      even though struct employee has not been completely defined
      while using typedef? < Yes / No >

```
typedef struct employee *ptr;
struct employee
{
      char name[20];
      int age;
      ptr next;
};
```

8]   There is an error in the following declarations. Can you
      rectify it?

```
typedef struct
{
      int data1;
```

```
        BPTR link1;
}*APTR;

typedef struct
{
        int data2;
        APTR link2;
}*BPTR;
```

9]   What do the following decalrations mean?

```
typedef char *pc;
typedef pc fpc();
typedef fpc *pfpc;
typedef pfpc fpfpc();
typedef fpfpc *pfpfpc;
pfpfpc a[N];
```

10]   How would you define a[N] in 15.9 above without using typedef?

11]   Improve the following code using typedef.

```
struct node
{
        int data1; float data2;
        struct node *left;
        struct node *right;
};
struct node *ptr;
ptr = (struct node *) malloc(sizeof(struct node));
```

12]   Is the following declaration acceptable? < Yes / No>

```
typedef long no,*ptrtono;
no n;
ptrtono p;
```

13]   In the following code what is constant, p or the cahracter it
      is pointing to?

```
typedef char *charp;
const charp p;
```

14]   In the following code is p2 an integer or on integer pointer?

```
typeder int *ptr;
ptr p1,p2;
```

```
*********************************************************************

******************        ANSWERS          *********************

*********************************************************************


1]    Yes


2]    It is a pointer to function which receives two character
      pointers and an integer.


3]    There are three main reasons for using typedefs:

      (a)        It makes writing of complicated decalrations a lot
      easier. This helps in eliminationg a lot of clutter in
      the program.

      (b)        It helps in achieving portability in programs. That
      is, if we use typedefs for data types that are
      machine-dependent, only the typedefs need change when
      the program is moved to a new machine platform.

      (c)        It helps in providing a better doumentation for a
      program. For example, a node of a doubly linked list is
      better understood as ptrtolist rather than just a
      pointer to a complicated structure.


4]    In these declarations, s1,s2 and s3 are all treated as char *,
      but s4 is treated as a char, which is probably not the
      intention.


5]    True.


6]    A typedef decalration cannot be used until it is defined, and
      in our example it is not yet defined at the point where the
      link field is declared.

      We can fix this problem in three ways:

      (a)        Give the structure a name, say node and then decare
      the link as a simple struct node * as shown below:
```

```
typedef struct node
{
        int data;
        struct node *link;
}*NODEPTR;
```

(b)   Keep the typedef declaration separate from the
structure definition:

```
struct node
{
        int data;
        struct node *link;
};
typedeaf struct node *NODEPTR;
```

(c)   Predede the structure declaration with typedef, so
that you can use the NODEPTR typedef when declaring the
link field:

```
typedef struct node *NODEPTR;
struct node
{
        int data;
        NODEPTR link;
};
```

7]   Yes

8]   The problem with the code is the compiler doesn't know about
     BPTR when it is used in the first structure declaration. We are
     violating the rule that a typedef declaration cannot be used
     until it is defined, and in our example it is not yet defined
     at the point where the link1 field is declared.

     To avoid this problem we can defint the structures as shown
     below:

```
struct a
{
     int data1;
     struct b *link1;
};
struct b
{
     int data2;
     struct a link*2;
};

typedef struct a *APTR;
typedef struct b *BPTR;
```

          The compiler can accept the field declaration strcut b
     *ptr1 within struct a, even though it has not yet heard   of
```

struct b (which is "incomplete" at that point).   Occasionally, it
is necessary to precede this couplet with the   line

        struct b;

           This empty declaration masks the pair of structure
        declarations (if in an inner scape) from a different struct b
        in an outer scope. After decalring the two structures we can
        then declare the typedefs separately as shown above.

        typedef struct a *APTR;
        typedef struct b *BPTR;
        struct a
        {
              int data1;
              BPTR link1;
        };
        struct b
        {
              int data2;
              APTR link2;
        };

9]   pc   is a pointer to char.
       fpc   is function returning pointer to char.
       pfpc   is pointer to a function returning pointer to
       char.
       fpfpc   is a function returning pointer to a function
       returning pointer to char.
       pfpfpc   is a pointer to function returning pointer to a
                   function returning pointer to char.
       pfpfpc a[N] is an array of N pointers to functions
       returning pointers to functions returning
       pointers to characters.

10]   char *(*(*a[N])())();

11]   typedef struct node *treeptr
       typedef struct node
       {
              int data1;
              float data2;
              treeptr *left;
              treeptr *right;
       }treenode;
       treeptr ptr;
       ptr = (treeptr)malloc(sizeof(treenode));

12]   Yes.

13]   p is a constant.

14]   Integer pointer.

Chapter-19
Constant Phenomenon

```
********************************************************************
*************        THE CONST PHENOMENON        *****************
********************************************************************
```

1]   Point out the error in the following program.

```c
main()
{
    const int x;
    x = 128;
    printf("%d",x);
}
```

2]   What would be the output of the following program?

```c
main()
{
    int y = 128;
    const int x = y;
    printf("%d",x);
}
```

A]   128
B]   Garbage value
C]   Error
D]   0

3]   What would be the output of the following program?

```
main()
{
      const int x = get();
      pjrintf("%d",x);
}

get()
{
      return(20);
}
```

A]   20
B]   Garbage value
C]   Error
D]   0

4]   Point out the error, if any, in the following program :

```
#define MAX = 128
main()
{
      const int mac = 128;
      char array[max];
      char string[MAX];
      array[0] = string[0] = 'A';
      printf("%c %c",array[0],string[0]);
}
```


5]   Point out the error in the following program :

```
main()
{
      char mybuf[] = "Zanzibar";
      charyourbuf[] = "Zienzkewiz";
      cajr *const ptr = mybuf;
      *ptr = '';
      ptr = yourbuf;
}
```


6]   Point out the error in the following program :

```
main()
{
      char mybuf[] = "Zanzibar";
      char yourbuf[] = "Zienckewiz";
      const char *ptr = mybuf;
      *ptr = 'a';
      ptr = yourbuf;
}
```


7]   Point out the error in the following program :

```
main()
```

```
{
        char mybuf[] = "Zanzibar";
        const char *const ptr = "Hello";
        ptr = mybuf;
        *ptr = 'M';
}
```

8]   What does the following prototype indicate?

```
strcpy(cahr *target, const char *source);
```


9]   What does the following prototype indicate?

```
const char *change(char *,int)
```


10]   Point out the error in the following program:

```
main()
{
        const char *fun();
        char *ptr = fun();
}

const cahr *fun()
{
        return"Hello";
}
```


11]   Is there any error in the following program? <Yes / No>

```
main()
{
        const cahar *fun();
        char *ptr = fun();
}

const char *fun()
{
        return"Hello";
}
```


12]   Point out the error in the following program :

```
main()
{
        const char *fun();
        *fun() = 'A';
}

const char *fun()
{
        return"Hello";
```

```
    }

13]   What do you mean by const correctness?


14]   What would be the output of the following program?

      main()
      {
            const int x = 5;
            int *ptrx;
            ptrx = &x;
            *ptrx = 10;
            printf("%d",x);
      }

A]    5
B]    10
C]    Error
D     Garbage value


15]   What would be the output of the following program?

      main()
      {
            const int x = 5;
            const int *ptrx;
            ptrx = &x;
            *ptrx = 10;
            printf("%d",x);
      }

A]    5
B]    10
C]    Error
D     Garbage value


16]   Point out the error in thee following program:

      main()
      {
            const int k = 7;
            int *const q = &k;
            printf("%d",*q);
      }


17]   What is the difference int the following declarations?

      const char *s;
      char const *s;


18]   What is the difference int the following declarations?
```

```
        const char *const s;
        char const *const s;
```

19]   Is the following a properly written function? If not, why not?

```
        int fun(const int n)
        {
            int a;
            a = n * n;
            return a;
        }
```

****

*****************   ANSWERS   ********************

****

1]   Nowhere other than through initialization can a program assign
        a value to a const identifier. 'x' should have been initialised
        where it is declared.

2]   A

3]   a

4]   The dimension of an array must always be positive non-zero
        integer which max is not.

5]   'ptr' pointer is constant. In ptr = yourbuf the program is
        trying to modify it, hence an error.

6]   'ptr' can be modified but not the object that it is pointing
        to. Hence '*ptr = 'A'' gives an error.
```

7]    'ptr' is a constant pointer to a constant object. We can
      neither modify 'ptr' nor the object it is pointing to.


8]    We can modify the pointers source as well as target. However,
      the object to which source is pointing cannot be nodified.


9]    The function change() receives a char pointer and an int and
      returns a pointer to a constant char.


10]   No.


12]   fun() retruns a pointer to a const character which cannot be
      modified.


13]   A program is 'const correct' if it nevr changes (a more common
      term is mutates) a constant object.


14]   B


15]   C


16]   Warning: Suspicious pointer conversion. 'k' can be pointed to
      only by a pointer to a const; q is a const pointer.


17]   There is no difference.


18]   There is no difference.


19]   Since fun() would be called by value there is no need to
      declare n as a const since passing by value already prevents
      fun() from modifying n.


# *Chapter-20*
# *C preprocessors*


********************************************************************

*********************  THE C PREPROCESSOR  *************************

```
**************************************************************************
```

1]   If the file to be included doesn't exist, the preprocessor
      flashes an error message. < True / False>

2]   The preprocessor can trap simple errors like missing
      declarations, nested comments or mismatck of braces.
      <True / False>

3]   What would be the output of the following program?

```
# define SQR(x) (x * x)
main()
{
     int a, b = 3;
     a = SQR(b + 2);
     printf("\n %d",a);
}
```

OPTIONS:
      (a)   25
      (b)   11
      (c)   Error
      (d)   Some garbage value

4]   How would you define the SQR macro in 6.3 above such that it
      gives the result of a as 25.

5]   What would be the output of the following program?

```
# define CUBE(x) (x * x * x)
main()
{
     int a, b = 3;
     a = CUBE(b++);
     printf("\n %d %d",a,b);
}
```


6]   Indicate what would the SWAP macro be expanded to on
      preprocessing. Would the code compile?

```
# define SWAP(a,b,c) (ct; t = a; a = b; b = t;)
main()
{
     int x = 10,y = 20;
     SWAP(x,y,int);
     printf("%d %d",x,y);
}
```

7]   How would you modify the SWAP macro in 6.6 above such that it
      is able to exchange two integers.

8]   What is the limitation of the SWAP macro of 6.7 above?

9]   In which line of the following program an error would be
      reported?

      (1)   #define CIRCUM(R) (3.14 * R * R);
      (2)   main()
      (3)   {
      (4)   float r = 1.0, c;
      (5)   c = CIRCUM(r);
      (6)   printf("\n %f",c);
      (7)   if(CIRCUM(r) == 6.28)
      (8)   printf("\n Gobbledygook");
      (9)   }

10]   What is the type of the variable 'b' in the following declaration?

      # define FLOATPTR float*
      FLOATPTR a,b;

11]   Is it necessary that the header files should have a '.h'
extension?

12]   What do the header files usually contain?

13]   Would it result into an error if a header file is included twice?
< Yes / No>

14]   How can a header file ensure that it doesn't get included more
than once?

15]   On inclusion, where are the header files searched for?

16]   Would the following typedef work?

      typedef #include I;

17]   Would the following code compile correctly?

      main()
      {
            #ifdef NOTE
                  /* unterminated comment */
                  int a;

```
                a = 10;
        #else
                int a;
                a = 20;
        #endif

        printf("%d",a);
    }
```

18]   What would be the output of the following program?

```
#define MESS Junk
main()
{
    printf("MESS");
}
```

19]   Would the following program print the message infinite number of times? <Yes / No>

```
#define INFINITELOOP while(1)
main()
{
    INFINITELOOP
        printf("\n Grey haired");
}
```

20]   What would be the output of the following program?

```
#define MAX(a,b) ( a>b? a:b)
main()
{
    int x;
x = MAX(3 + 2,2 + 7);
printf("%d",x);
}
```

21]   What would be the output of the following program?

```
#define PRINT(int) printf("%d",int)
main()
{
    int x = 2, y = 3, z = 4;
    PRINT(x);
    PRINT(y);
    PRINT(z);
}
```

22]   What would be the output of the following program?

```
#define PRINT(int) printf(*int = %d*,int)
main()
```

```
        {
                int x = 2, y = 3, z = 4;
                PRINT(x);
                PRINT(y);
                PRINT(z);
        }
```

23]   How would you modify the macro of 6.22 such that it outputs:

       x = 2 y = 3 z = 4

24]   Would the following program compile successfully? < Yes / No)

```
       main()
       {
                printf("Tips" "Traps);
       }
```


25]   Define the macro DEBUG such that the following program outputs:

```
       DEBUG: x = 4
       DEBUG: y = 3.140000
       DEBUG: ch = A

       main()
       {
                int x = 4;
                float a = 3.14;
                char ch = 'A';

       DEBUG(x,%d);
       DEBUG(a,%f);
       DEBUG(ch,%c);
       }
```


****

********************   ANSWERS
****

****
```
```

```
1]    True


2]    False


3]    B.
      Because, on preprocessing the expression becomes
      'a = a(3 + 2 * 2 + 3).


4]    #define SQR(x) ((x) * (x))


5]    27 6.
      Though some compilers may give this as the answer, according to
      the ANSI C standard the expression b++ * b++ * b++ is
      undefined. Refer Chapter 3 for more details on this.


6]    (int t; t = a, a = b, b= t);
      This code won't compile  since declaration of t cannot occur
      within parentheses.


7]    #define SWAP(a,b,c) ct; t = a, a = b, b = t;


8]    It cannot swap pointers. For example, the following code would
      not compile.

      #define SWAP(a,b,c) ct; t = a, a = b, b = t;
      main()
      {
            float x = 10,y = 20;
            float *p,*q;
            p = &x; q = &y;
            SWAP(p,q,float*);
            printf("%f %f",x,y);
      }


9]    Line number 7, whereas the culprit is really the semicolon in
      line number 1. On expansion line 7 becomes
      ' if ((3.14 * 1.0 * 1.0); == 6.28). Hence the error.


10]   'float' and not a pointer to a float, since on expansion the
      declaration becomes:

      float *a,b;
```

11]   No.   However, traditionally they have been given a .h
       extension to identify them as someting different than the .c
       program files.

12]   Preprocessor directives like #define, structure, union and enum
       declarations, typedef declarations, global variable
       declarations and external function declarations. YOu should not
       write the actual code   (i.e. function bodies) or global
       variable definition (that is difining or initialising
       instances) in header files. The # include directives should be
       used to pull in header files, not other .c files.

13]   Yes, unless the header file has taken care to ensure that if
       already included it doesn't get included again.

14]   All declarations must be written in the manner shown below.
       Assume that the name of the header file in FUNCS.H.

       /*   funcs.h   */
       #ifdef_FUNCS
            #define_FUNCS
            /* all declarations would go here */
       #endif
       Now if we include this file twice as shown below, it would get
       included only once.

       #include "goto.c"
       #include "goto.c"
       main()
       {
            /* some code */
       }

15]   If included using <> the files get searched in the predefined
       (can be changed) include path. If included withe the " " syntax
       in addition to the predefined include path the files also get
       searched in the current directory (usually the directory form
       which you inboked the compiler).

16]   No.   Because 'typedef' goes to work after preprocessing.

17]   No.   Even though the #ifdef fails in this case (NOTE being
       undefined) and the if block doesn't go for compilation errors
       in it are not permitted.

18]   MESS

19]   Yes

20]   9

21]   2 3 4

22]   int = 2 int = 3 int = 4

23]   #defien PRINT(int) printf(#int "= %d", int)
       main()
       {
             int x = 2, y = 3, z = 4;
             PRINT(x);
             PRINT(y);
             PRINT(z);
       }

       The rule is if the parameter name is preceded by a # in the
       macro expansion, the combination(of # and parameter) will be
       expanded into quoted string with the parameter replaced by the
       actual argument. This can be combined with string concatenation
       to print the output desired in our program. On expansion the
       macro becomes:

       printf("x" "= %d",x);

       The two strings get concatenated, so the effect is

       printf("x = %d",x);

24]   Yes. The output would be TipsTraps. In fact this result has
       been used in 6.23 above.

25]   #define DEBUG(var,fmt) printf("DEBUG:" #var "="#fmt"\n",var)

26]   multiply
       Here two operations are being carried out expansion and
       stringizinf. Xstr() macro expands its argument, and then str()
       stringizes it.

27]   #define PRINT(var1,var2,var3) printf("\n" #var1 "= %d" #var2 "=
       %d" #var3 "= %d",var1,var2,var3)

Chapter-21
Variable Number of arguments

```
*********************************************************************

***************** VARIABLE NUMBER OF ARGUEMENTS *********************

*********************************************************************
```

1]   Which header file should you include if you are to develop a
     function which can accept variable number of arguments?

Options:
     (a) vararg.h
     (b) stdlib.h
     (c) stdio.h
     (d) stdarg.h

2]   What would be the output of the following program?

```
# include<stdio.h>
# include<stdarg.h>
main()
{
     fun("Nothing specific", 1,4,7,11,0);
}

fun(char *msg, ...)
{
     int tot = 0;
     va_list ptr;
     int num;

     va_start(ptr, msg);
     num = va_org(ptr,int);
     num = va_arg(ptr,int);
     printf("\n%d",num);
}
```


3]   Is it necessary that in a function which accepts variable
     argument list there should at least be one fixed argument?
     < Yes   /   No >

4]   Can the fixed arguments passed to the functio which accepts
     variable argument list occur at the end? < Yes   /   No >

5]   Point out the error, if any, in the follwoing program.

```
# include<stdarg.h>
main()
{
     varfun(3,7,-11,0);
}

varfun(intn, ...)
{
     va_list ptr;
     int num;

     num = va_arg(ptr,int);
     printf("\n %d",num);
}
```

6]   Point out the error, if any, in the follwoing program.

```
# include<stdarg.h>
main()
{
     varfun(3,7.5,-11.2,0.66);
}

varfun(int n, ... )
{
     float * ptr;
     int num;

     va_start(ptr,n);
     num = va_arg(ptr,int);
     printf("\n %d",num);
}
```

7]   Point out the error, if any, in the follwoing program.

```
# include<stdarg.h>
main()
{
     fun(1,4,7,11,0);
}

fun( ... )
{
     va_list ptr;
     int num;

     va_start(ptr,int);
     num = va_arg(ptr,int);
     printf("\n %d",num);
}
```

8]   The macro va_start is used to initialise a pointer to the
      beginning of the list of fixed arguments. < True / False >

9]   The macro va_arg is used to extract an argument from the
      variable argumetn list and advance the pointer to the next
      argument. < True / False >

10]   Point out the error, if any, in the following program.

```
# include"stdarg.h"
main()
{
      display(4,'A','a','b','c');
}

display(int num, ... )
{
      char c; int j;
      va_list ptr;
      va_start(ptr,num);

      for(j = 1; j <= num; j++)
      {
            c = va_arg(ptr,char);
            printf("%c",c);
      }
}
```


11]   Mention any variable argument list function that you have used
      and its prototype.

12]   Point out the error, if any, in the follwoing program.

```
# include"stdarg.h"
main()
{
      display(4,12.5,13.5,14.5,44.3);
}

display(int num, ... )
{
      float c; int j;
      va_list ptr;

      va_start(ptr, num);

      for(j = 1; j <= num; j++)
      {
            c = va_arg(ptr,float);
            printf("\n %f",c);
      }
}
```

13]   Point out the error, if any, in the follwoing program.

```
# include"stdarg.h"
main()
{
      display("Hello",4,2,12.5,13.5,14.5,44);
}

display(char *s,int num1, int num2, ... )
{
      float c; int j;
      va_list ptr;

      va_start(ptr, num);

      c = va_arg(ptr,double);
      printf("\n %f",c);
}
```

14]   Can we pass a variable argumetn list to a function at run-time?
      < Yes   /   No >

15]   While defining a variable argument list function can we drop
      the ellipsis. (...)   < Yes   /   No >

16]   Can we write a function that takes a variable argumetn list and
      passes the list to another function (which takes a variable
      number of argumetns) < Yes   /   No>

17]   Point out the error, if any, in the foll0wing program.

```
# include "stdarg.h"
main()
{
      display("Hello",4,12,13,14,44);
}

display(char *s, ... )
{
      show(s, ... )
}

show(char *t, ... )
{
      va_list ptr;
      int a;
      va_start(ptr,t);
      a = va_arg(ptr,int);
      printf("%f",a);
}
```

18]   How would you rewrite program 18.17 such that show() is able to
       handle variable argument list?

19]   If I use the following printf() to proint a long int why I am
       not warned about the type mismatch?

       printf("%d",num);

20]   Can you write a function that works similar to 'printf()'?

21]   How can a called function determine the number of arguments
       that have been passed to it?

22]   Can there be at least some solution to determine the number of
       arguments passed to a variable argument list function?

23]   Point out the error in the following program.

       # include "stdarg.h"
       main()
       {
             int (*p1)();
             int (*p2)();
             int fun1(),fun();

             p1 = fun1();
             p2 = fun2();
             display("Bye",p1,p2);
       }

       display(char *s, ...)
       {
             int (*pp1)();
             va_list ptr;

             va_start(ptr,s);
             pp1 = va_arg(ptr,int(*)());
             (*pp1)();
             pp2 = va_arg(ptr,int(*)());
             (*pp2)();
       }

       fun1()
       {
             printf("Hello");
       }

       fun2()
       {
             printf("Hi");
       }

24]   How would you rectify the error in the program 18.23 above?

```
********************************************************************
***********************      ANSWERS      **************************
********************************************************************
```

1]   D

2]   4

3]   Yes

4]   No

5]   Since 'ptr' has not been set at the beginning of the variable
      argument list using va_start it may be pointing anywhere and
      hence a call to va_arg would fetch a wrong integer.

6]   Irrespective of the type of argument passed to a function
      receiving variable argument list, 'ptr' must be of the type
      va_list.

7]   There is no fixed argument i n the definition of 'fun()'.

8]   False

9]   True

10]   While extracting a 'char' argument using va_arg we should have
      used   c = va_arg(ptr,int).

11]   printf(const char *format, ...)

12]   While extracting a float argrment using va_arg we should have
      used   c = va_arg(ptr,double).

13]   While setting the 'ptr' to the beginning of variable argument
      list we should have used   va_start(ptr,num2).

14]   No. Every actual argument list must be completely known at
      compile time. In that sense it is not truely a variable
      argument list.

15]   Yes

16]   Yes

17]   The call to 'show()' is improper. This is not the way to pass
      variable argument list to a function.

18]   # include"stdarg.h"
      main()
      {
            display("Hello",4,12,13,14,44);
      }

      display(char *s, ...)
      {
            va_list ptr;
            va_start(ptr,s);
            show(s,ptr);
      }

      show(char *,va_list ptr1)
      {
            int a,n,i;
            a = va_arg(ptr1,int);
            for(i = 0; i < a; i++)
            {
                  n = va_arg(ptr1,int);
                  printf("\n %d",n);
            }
      }

19]   When a function accepts a variable number of arguments, its
       prototype cannot provide any information about the number of
       arguments and type of those variable arguments. Hence the
       compiler cannot warn about the mismatches. The programmer must
       make sure that arguments match or must manually insert explicit
       typecasts.

20]   ```c
       # include<stdio.h>
       # include<stdarg.h>

       main()
       {
            void myprintf(char *, ...)
            char *convert(unsigned int,int);
            int i = 65;
            char str[] = "Intranets are here to stay ..";
            myprintf("\n Message = %s, %d %x",str,i,i);
       }

       void myprintf(char *fmt, ...)
       {
            char *p;
            int i;
            unsigned j;
            char *s;
            va_list argp;

            va_start(argp,fmt);

            p = fmt;
            for(p = fmt; *p != '\0\; p++)
            {
               if(*p != '%')
               {
                  putchar (*p);
                  continue;
               }
                  p++;
            switch(*p)
            {
               case 'c':

                   i = va_arg(argp,int);
                   putchar(i);
                   break;

               case 'd':

                   i = va_arg(argp,int);
                   if(i < 0)
                   {
                      i = -i;
                      purchar('-');
                   }
                   puts(cionvert(i,10));
                   break;
       ```

```
                case 'o':

                    u = va_arg(argp,unsigned int);
                    puts(convert(u,8));
                    break;

                case 's':

                    s = va_arg(argp,char *);
                    puts(s);
                    break;

                case 'u':

                    u = va_arg(argp,unsigned int);
                    puts(convert(u,10));
                    break;

                case 'x':

                    u = va_arg(argp,unsigned int);
                    puts(convert(u,16));
                    break;

                case '%':

                    purchar('%');
                    break;
            }
        }
    va_end(argp);
    }

    char *convert(unsigned int num, int base)
    {
        static char buff[33];
        char *ptr;

        ptr = &buff[sizeof(buff) - 1];
        *ptr = '\0\;
        do
        {
            *--ptr = "0123456789abcdef"[num % base];
            num /= base;
        }while(num != 0);

        returnptr;
    }
```

21]   It cannot. Any function that takes a variable number of
       arguments must be able to determine from the arguments
       themselves how many of them there are. The printf() function,
       for example, does this by looking for format specifiers (%d
       etc.) in the format string. This is the reason why such

functions fail badly if the format string does not match the argument list.

22]   When the arguments passed are all of same type we can think of passing a sentinel value like -1 or 0 or a NULL pointer at the end of the variable argument list. Another way could be to explicitly pass the count of number of variable arguments.

23]   'va_arg' cannot extract the function pointer from the variable argument list in the manner tried here.

24]   Use 'typedef' as shown below:

```
# include"stdarg.h"
main()
{
      int(*p1)();
      int(*p2)();
      int fun1(),fun2();

      p1 = fun1;
      p2 = fun2;
      display("Bye",p1,p2);
}

display(char *s, ... )
{
      int (*pp1)(),(*pp2)();
      va_list ptr;
      typedef int(*funcptr)();
      va_start(ptr,s);
      pp1 = va_arg(ptr,funcptr);
      (*pp1)();
      pp2 = va_arg(ptr,funcptr);
      (*pp2)();
}

fun1()
{
      printf("\n Hello");
}

fun2()
{
      printf("\n Hi");
}
```