B.TECH.

SECOND SEMESTER EXAMINATION 2009-10

COMPUTER CONCEPTS & PROGRAMMING IN C

Time : 3 Hours *Total Marks : 100*

Section-A

This question contains 10 questions of multiple choice. Attempt all parts of this section.

(10×2=20)

Q.1. (*a*) C is a:

(*i*) High Level Language

(*ii*) Low Level Language

(*iii*) High Level Language with some low level features.

(*iv*) Low Level Language with high level features

Ans. (*iii*) High Level Language with some low level features.

Q.1. (*b*) The purpose of the following program fragment:

$b = s + b;$

$s = b - s;$

$b = b - s;$

where *s, b* are two Integers is to:

(*i*) Transfer the content of *s* to *b*

(*ii*) Transfer the content of *b* to *s*

(*iii*) Negate the content of *s* and *b*

(*iv*) Swap the contents of *s* and *b*

Ans. (*iii*) Negate the content of *s* and *b*

Q.1. (*c*) Consider the function

Find (int *x*, int *y*)

[return ((*x* < *y*?0:(*x* − *y*));]

Let *a, b* be two non negative Integers. The call find (*a*, find(*a, b*)) can be used to find the:

(*i*) Maximum of *a, b*

(*ii*) Posltive difference of *a, b*

(*iii*) Sum of *a, b*

(*iv*) Minimum of *a, b*

Ans. (*iv*) Minimum of *a, b*

Q.1. (*d*) Consider the function

Find (int *x*, int *y*)

[return ((*x*<*y*?0:(*x*−*y*));]

let *a, b* be two non-negative integers. The call find (*a*, find (*a, b*)) can be used to find the:

(*i*) Maximum of *a, b*

(*ii*) Positive difference of *a, b*

(*iii*) Sum of *a, b*

(*iv*) Minimum of *a, b*

Ans. (*ii*) Positive difference of *a, b*

Q.1. (*e*) The maximum value of a signal integer is:

(*i*) $2^{16}-1$ (*ii*) $2^{15}-1$

(*iii*) 2^{16} (*iv*) 2^{15}

Ans. (*ii*) $2^{15}-1$

Q.1. (*f*) The value of an automatic variable that is declared but not initialized will be:

(*i*) 0

(*ii*) Unpredictable

(*iii*) − 1

(*iv*) None of these

Ans. (*ii*) Unpredictable

Q.1. (*g*) If *n*=3 then the output of the statement printf("%d %d", n++, ++n); will be:

(*i*) 3 5 (*ii*) 4 5

(*iii*) 4 4

(*iv*) is implementation dependent

Ans. (*iii*) 4 4

Q.1. (h) The following program fragment

```
for (i=3; i<15; i=i+3);
printf("%d", I);
```

results in

(i) A syntax error

(ii) An execution error

(iii) Printing of 12

(iv) Printing of 15

Ans. (ii) An execution error

Q.1. (i) 2s complement of $(5)_{10}$ will be:

(i) 5 (ii) 6

(iii) – 5 (iv) Not possible

Ans. (iii) – 5

Q.1. (j) Select the odd man out:

(i) Integer (ii) Structure

(iii) Union (iv) Array

Ans. (i) Integer

SECTION — B

Note: Attempt any three questions. All questions carry equal marks: (3×10=30)

Q.2. (a) (i) Differentiate between DOS, UNIX and Windows Operation System.

(ii) Explain the following UNIX command.

ls, chomd, sh, su, who

Ans. (i) DOS: DOS operating system is a command line interface i.e. based on various commands to perform the operations, it does not support the GUI to perform the operations. It is single user operating system.

UNIX: UNIX operating system is based on GUI and it is a multi user and multitasking operating system. It is required for operation which are not covered by GUI based program.

Windows: Windows operating system is based on GUI and it is single user and multitasking operating system. It has huges changes from first version to last version i.e. from 1985 to 2010.

(ii) ls: Used to show the list

chmod: Used to change the mod of file

sh: used to run or process the job through Bourneshell.

su: Used to change the login session is owner.

who: used to show the current path and user name.

(b) Explain the various data types in C giving suitable examples of each.

Ans. Data Types: Programming language C has a concepts of data types which are used to define a variable before its use. The data types determine the types of values and the ranga of values that can be stored in a variable.

Example: *If the count of items purchased has to be stored, the data type will be int (An integer). The reason is that there is no fractional value for the count of items.*

Similarly, there are other data types to store different types of data.

The main data types are:

- Integer data tyeps.
- Floating point data types.
- Character data types.

Integer data tyeps: Integer are those numbers that are whole numbers i.e. they do not contain a decimal point and decimal digits.

For example 12820.

The integer data types are again categorised into three data types which are given below:

- Short integer (short)
- Integer (int)
- Long integer (long or long int)

Each of the integer types can be signed or unsigned. Signed integers can store negative values also. Unsigned integers can store only positive values.

Floating point data types: It is a number that can have a fractional part. It can contain a decimal point and decimal digit.

It is a two types: -

- Double precision floating point
- Single precision floating point

Depending upon type of values these two data types assigned. For example:

float salary

double total-profit

double temperature

Character Data Types: The 'char' data type in C can store a single character. That is, any character that can be printed on the computer screen can be stored using this data types.

Example: 'p', '1', '*', 'A'

A character data type is nothing but an integer of size 1 byte. Therefore, characters can also be signed and unsigned.

Data Types Supported in C: Depending on the range of values for a data type, the size (in number of bytes) also varies. The size of a data type is machine dependent. In some platforms 'int' may occupy two types whereas in some other it may occupy four bytes.

The Table lists the data types supported by C language and their respective range:

Table *Data Types Supported in C.*

Data Type	*Bytes*	*Bits*	*Range*
Short int	2	16	– 32,768 to + 32,767
Unsigned Short int (or) unsigned short	2	16	0 to 65, 535
singned int (or) int	4	32	– 2,147,483,648 to + 2,147,483,647
unsigned int	4	32	0 to 4,294,967,295
long int (or) long	4	32	– 2,147,483,648 to + 2,147,483,647
unsigned long int (or) unsigned long	4	32	0 to 4,294,967,295
char (or) signed char	1	8	– 128 to + 127
unsigned char	1	8	0 to 255
float	4	32	3.4 e – 38 to 3.4 e+37
double	8	64	1.7e – 308 to 1.7 e+308
long doube	10	80	3.4E – 4932 to 1.1E + 4932

Sample program illustrating each data type:

```
#include<stdio.h>
main( )
{
   int total;
   float money;
   char letter;
   double pi;
   total = 20;      /*assign integer value*/
   money= 12.21; /*assign float value*/
```

```
    letter = 'A';     /*assign character value */
    pi=22.01E6;    /*assgin a double value */
    printf("value of total = %d\n", total);
    printf("value of money=%f\n", money);
    printf("value of letter = %c\n", letter);
    printf("value of pi = %e\n", pi);
}
```

(c) Convert the following numbers as specified

(i) $(1011011101.01101)_2 = (?)_{10}$

(ii) $(3142.28)_{10} = (?)_2$

(iii) $(1101111010.01101)_2 = (?)_{16}$

(iv) 2s complement of 110100100

(v) $(AC01.2DB)_2 = (?)_8$

Ans. (i) $(1011011101.01101)_2 = (733.40625)_{10}$

(ii) $(3142.28)_{10} = (110001000110.010001)_2$

(iii) $(1101111010.01101)_2 = (DEA.68)_{16}$

(iv) 2s complement of 110100100 is 001011100

(d) Write a program in C to copy the content of a given file say "a.txt" to another file "b.txt".

Ans. Program to Copy one file into another file

```
#include<stdio.h>
main
{    char in_name[25], out_name[25];
     FILE *in_file, *out_file, *fopen( );
     int c;
     printf("\nPlease enter File to be copied:\n");
     scanf("%24s", in_name);
     printf("Output filename:\n");
     scanf("%24s", out_name);
     in_file = fopen(in_name, "r");
     if(in_file==NULL)
          printf("Cannot open %s for reading \n", in_name);
     else
     {
          out_file = fopen(out_name, "w");
          if(out_file==NULL)
               printf("Can't open %s for writing.\n", out_name)
          else
          {
               while(c = getc(in_file))!=EOF)
```

```
                putc(c, out_file);
                putc(c, out_file);
                printf("File has been  copied.\n")
                fclose(out_file);
            }
            fclose(in_file);
        }
    }
```

(e) Differentiate between call by value and call by reference. Make a program in C to show the usage of both.

Ans. Pass by value or call by value: In this you can use the concept of actual and formal parameters. Since the value of actual and formal parameter is stored in different memory location so the changes in formal parameter do not change the value of actual parameters. For example,

Simple program of formal and actual argument

```
#include<stdio.h>
#include<conio.h>
int compute(int, int);
main( )
{
    int number1 = 100, number2 = 500;
    printf("The value before calling the function");
    printf("number1 = %d and number2=%d \n", number1, number2);
    comput(number1, number2);
    printf("The value after calling the function");
    printf("number1 = %d and number2=%d\n", number1, number2);
    }
    int compute(int num1, int num2)
    {
            num1=num1 + 100;
            num2=num2 + 100;
    }
```

Program output: The value before calling the function number1 = 100 and number2 = 500.

The value after calling the function number = 100 and number2 = 500.

Pass by reference or call by reference: In this concept instead of sending the values of actual parameter you send the address of the actual parameter to the functions. The changes that are made to formal parameter also change the value of actual parameters. For example,

Write a function to swap two variable values

```
#include<stdio.h>
#include<conio.h>
int main( )
```

```
{
        int a;
        int b;
        printf("Enter the value of a and b :");
        printf("%d%d", &a, &b);
        swap(&a, &b)
        printf("The value of a and b after swap is %d and %d", a, b);
                getch( );
}
void swap(int *px, int *py)
}
        int temp;
        temp = *px;
        *px = *py;
        *py = temp;
}
```

The output is

Enter the value of a and b : 100 20

The value of a and b after swap is 20 and 100.

Difference between pass by value and reference

Pass by Value	*Pass by Reference*
1. It consumes more memory space because formal parameter also occupies memory space.	1. It consumes less memory space because irrespecvtive of the data type of the actual arguments, each pointer occupies only 4 types.
2. It takes more time to execute because the values are copied.	2. It takes less time because no values are copied.

SECTION — C

Note: Attemt any two parts from each question. All questions are compulsory:

(5×10=50)

Q.3. (a) Wite a program in C to find whether the given number is prime or not.

Ans.

```
main( )
{
int a, c=0; i, n;
printf("enter the number to be checked");
scanf("%d", &n);
for(i = 1; i<=n; i++)
```

```
    {
        a = n%i;
        if(a = 0)
            {
                c=c+1;
            }
    }
if(c=2)
    {printf("the given number is prime");}
else
    printf("the given number is not prime");
}
```

(b) Write a program in C to find out second largest element of a given list of integers.

Ans.

```
void main( )
{
int a[12], max, min1, min2, max2, i, temp;
printf("enter value");
for(i=0; i<11; i++)
scanf(%d, &a[i]);
max = a[0]'
min1=a[0];
for(i=1; i<11; i++)
{
if(max<a[i])
max = a[i];
if(min1>a[i]);
min1=a[i]
}
for(i=1; i<11; i++)
{
if((max2>a[i])&&(max2!=max)){
max2=a[i];
}
if((min2<a[i])&&(min2!=min1))
{
min2=a[i];}
    print("small is %d and large is %d", min2, max2);
    }
```

(c) Write a program in C to short the given list of names.

Ans.

```
#include<stdio.h>
#include<string.h>
void main(void)
{
int i, k, n;
char*name[50], *temp= "";
printf("\nEnter number of persons:");
scanf("%d", &n);
for(i=0; i<n; i++) gets(name[i]);
for(k=0; k<(n-1); k++)
{
for(i=0; i<(n-k-1); i++)
{
if(strcmp(name[i], name[i+1]>0))
{
strcpy(temp., name[i]);
strcpy(name[i], name[i+1]);
strcpy(name[i+1], temp);
}
}
}
printf("\nSorted List...\n");
for(i=0; i<n; ++i)
puts(name[i]);
}
```

Q.4. (a) Define the storage class In C.

Ans. There are five types of storage classes in C. These are used to store variables. These are as follows:

I. Extern
II. Static
III. Register
IV. Auto
V. typedef

I. Auto: This is the default storage class. Auto can only be used with in functions, i.e. only for local variables, not for globals.

II. Register: The variables declared using the register storage class may stored in CPU registers instead of RAM. Since it doesn't have a memory location, the '&' operator for getting the address of the variable cannot be applied (in C). This storage class cannot be used for global scope data.

III. Static: This is the default storage class for global variables. In case of local variable, it is initialized at compile time and retains its value between the calls. By default the static variables will be initialized to zero, incase of pointer variable initizlized to NULL.

IV. Extern: Defines the global variables that is visible to all objects modules. This type of variables cannot be initialized, since it is pointing to a storage location., where it is previously defined.

V. Typedef: A typedef declaration lets you define your own identifiers that can be used in place of type specifiers such as int, float, and double. A typedef declaration does not reserve storage. The names you define using typedef are not new data types, but synonyms for the data types or combinations of data types they represent.

(b) What are the merits and demerits of static and dynamic memory allocation techniques?

Ans. Static Merits: The merit is that you don't need to take care about deallocation.

Static Demerits: Static memory allocation utilizes stack which is a limited resource, especially in such environments as kernel mode.

In the most cases it's quicker but it also depends in the environment (in stack growth is involved).

You can't share the resource allocated on stack in multi-threaded environment.

Dynamic Merits: You can share dynamically allocated memory between threads, does not consume stack space.

You can allocate different types of memory (usually).

Dynamic Demerits: Allocation is slower than static.

You need to take care about deallocation.

(c) What are the different bit operators used in C? Givde an example of each.

Ans. The different bit operators supported by C are as follows:

&	Bitwise AND
|	Bitwise inclusive OR
^	Bitwise EXOR
~	Ones complement
<<	Left shift
>>	Right shift

Bitwise logical operators: Logical operations have two results, it is either true or false. In binary notation 1 denotes true and 0 a false. An int stored in memory can be seen as a string of true and false values.

The bitwise logical operators work ou integer operands, which are evaluated bit by bit to an integer result.

Bitwise AND: The logical AND evaluates to true (1), only if both operands are 1.

0 & 0 0

0 & 1 0

1 & 0 0

1 & 1 1

```
/*Program to demonstrate logical AND operator */
#include
main( )
    int a = 25;
    int b = 77;
    int c;
    c = a & b;
    printf("Value of c is %d\n", c);
}
```

The program results in the following output:

Value of c is 9.

The AND operation that occurs is shown below:

0000000000011101	25
0000000001001101	77
0000000000001001	9

The bitwise AND operation is used for masking operations. This operator can be used to set specific bits of a number to 0. Also you can use bitwise AND to test whether an integer is odd or even. When you do a bitwise AND of an integer with 1, the result will be true if the rightmost bit of the integer is 1. This is the case with an odd integer, whereas an even integer will have 0 as the rightmost bit.

Bitwise inclusive OR: In this again, the binary representation of the two operands involved are compared bit by bit. Each bit that is a 1 in the first operand or a 1 in the second operand will produce a 1 in the corresponding bit of the result.

0 | 0 0

0 | 1 1

1 | 0 1

1 | 1 1

The bitwise inclusive OR operation is used when you want to set some specified bits of a number to 1.

Bitwise Exclusive OR: The bitwise OR or XOR gives 1 if either bit is 1 but not both.

0 ^ 0 0

0 ^ 1 1

1 ^ 0 1

1 ^ 1 0

One important point to remember is that any value which is XORed with itself results in 0. This is used by assembly language programmers to test for equality of two values.

One's complement operator: The one's complement operator in C is represented as tilde ~. This converts a value into its one's complement, in other words, all the zeros become ones and the ones become zeros. This is a unary oeprator that operates on an integer constant or expression.

For example

~12

~(Total - 2)

Here, Total must be an integer.

One's complement is used to make a program more portable. For example, if we want to set the rightmost bit of an integer to zero on both 16 bit and 32 bit machines, it is wiser to AND it with a one's complement.

Total $ = ~ 1;

This will result in a one's complement of 1 with as many leftmost 1 bits as required to fill the size of an integer. It will be 15 leftmost bits on a 16-bit integer machine, and 31 on a 32-bit integer machine.

Shift Operators: Left shift and right shift operations are analogous to multiplication and division by 10. When we divide a number by 10, we shift the digits once to the right by retaining the decimal point. When we multiply by ten we shift the number left and add 0 on the right.

Left shift operator <<: When a left shift operation is performed on an operand, the bits of the operand are shifted left. Bits that are shifted out of the high order bit of the data item are lost and 0s are added through the low order bit of the operand.

If Salary is a variable with the octal value of 6, then a left shift operation results in the octal value 14.

Program:

```
/* Program to demonstrate left shift operator*/
#include
main( )
{
    unsigned int salary = 06;
    salary <<=1;
    printf("Value of salary is %o\n", salary);
}
```

Right shift operator >>: In a right shift operation, the bit on the right which is the low-order bit is lost and depending on the type of machine, either a 1 or 0 will be shifted into the leftmost bit. The shift operation is also sometimes referred to as rotating left and right.

Q.5. (a) Simulate calculator using switch statement.

Ans. Program of simple calculator using switch cases

```
#include <stdio.h>
main( )
{
    int invalid_operator = 0;
    char operator;
    float numb1, numb2, result;
    printf("Enter two numbers and an operator in the format\n");
    printf("number1 operator number2\n");
    scanf("%f%c%f", &numb1, &operator, &numb2);
```

```
        switch (operator)
        {
            case '*' : result = numb1 * numb2; break;
            case '/' : result = numb1 / numb2; break;
            case '+' : result = numb1 + numb2; break;
            case '-' : result = numb1 - numb2; break;
            default : invalid_operator = 1;
        }
        switch(invalid_operator)
        {
            case1 : printf("invalid operator.\n"); break;
            default : printf("%f %c %f is %f\n", numb1, operator, numb2, result);
        }
}
```

(b) Implement Stack using Linked List.

Ans. The different function of stack can be represented in C using linked list are shown below:

```
int create(struct stack *s)
{
    s -> top=NULL;
}
int Empty(struct stack *s)
{
    if(s->top = = NULL)
        return 1;
    else
        return 0;
}
    void push (struct stackNode **top, int info)
{
    struct stackNode *new;
    new = (struct stackNode *)malloc(sizeof (struct stackNode));
    if(new != NULL) {
        new->data = info;
        new ->next = *top;
        *top = new;
    }
    else
        printf("%d not inserted. No memory available.\n", info);
```

```
    }
    int pop(struct stackNode **top)
    {
        struct stackNode *temp;
        int value;
        temp = *top;
        value = (*top)->data
        *top = (*top)->next;
        free(temp);
        return value;
}
```

(c) Write a program in C to multiply two matrices. Take the size and element of matrice through keyboard.

Ans.

```
main( )
{
    int a[5], [5],, b[5][5], i, j, k
    static int c[s] [s];
    printf("Entr the first metrix element : \n");
    for (i=0; i<3; i++)
        for(j=0; u<3; j++)
            scan("%d", &a[i][j]);
    printf("\nEnter the second matrix elements : \n");
    for(i=0; i<3; i++)
        for(j=0; j<3; j++)
            scanf("%d", &b[i][j]);
    for (i=0; i<3; i++)
        for(j=0; j<3; j++)
            for(k=0; k<3; k++)
                c[i][j] = c[i][j] + a[i][k] * b[k][j];
    printf("\nThe multiplication of two matrix elements are :\n");
    for(i=0; i<3; i++)
    {
        for(j=0; j<3; j++)
        printf("%4d", c[i][j]);
        printf("\n");
    }
    getch( );
}
```

Q.6. (a) Show the usage of break and continue statement by taking an example.

Ans. The Break Statement: We have already met break in the discussion of the switch statement. It is used to exit from a loop or a switch, control poassing to the first statement beyond the loop or a switch.

With loops, break can be used to force an early exit from the loop, or to implement a loop with a test to exit in the middle of the loop body. A break within a loop should always be protected within an if statement which provides the test to control the exit condition.

The Continue Statement: This is similar to break but is encountered less frequently. It only works within loops where its effect is to force an immediate jump to the loop control statement.

- In a while loop, jump to the test statement.
- In a do while loop, jump to the test statement.
- In a for loop, jump to the test, and perform the iteration.

Like a break, continue should be protected by an if statement.

```
main( )
{
    int i, sum = 0;
    for(i=1; i<=20; i++)
    {
        if(i == 10)
            continue;
            sum = sum + i;
            if(i == 15)
                break;
        printf("%d", sum);
    }
    getch
}
```

(b) Write an algorithm to sort the given list of integers.

Ans. Algorithm: Let us assume that array 'Item' contains 'Count' elements:

```
Step #  Steps
        Begin
Step 1: Pass = 0
Step 2: Smallest Item Position = Pass
Step 3: Comparison = Pass + 1
Step 4: If Item (Comparison) < Item(Smallest Position) then
            Smallest Position = Comparison
Step 5: Comparison = Comparison + 1
Step 6: If Comparison < Count then goto Step 4
Step 7: Temp = Item[Pass]                          Interchange
        Item[Pass] = Item (smallest Item Position)  elements
```

Item[Smallst Item Position] = Temp

Step 8: Pass = Pass + 1

Step 9: If Pass < Count then goto Step 2

END.

(c) Discuss the usage of macro in C.

Ans. Macro expansion is a preprocessor statement, which is used to make program more readable and fast executing. Each preprocessor statement is defined with the word define. The word define begins with a # symbol and is not terminated by a semi-colon. It is normally within at the beginning of the source program. For example,

```
#define TRUE 1
#define FALSE 0
#define AND   &&
#define OR   |
#define EQUALS ==
```

All preprocessor statemens are handled by the compiler (or preprocessor) before the program is actually compiled. All preprocessor statements (i.e. starting with #) are processed first and the symbols, which occur in the C program are replaced by their values. Once this substitution has taken place by the preprocessor, the program is then compiled.

In general, preprocessor constants are written in UPPERCASE.

Substitution of Symbolic Constants Using # Define

Let us consider some examples of using three symbolic constants. Consider the following program, which defines a constant called TAX_RATE.

Example:

```
#include<stdio.h>
#define TAX_RATE 0.20
main( )
{
    float income;
    float tax;
    income = 140500.0;
    tax = income * TAX_RATE;
    printf("The tax on %.2f is %.2f\n", income, tax);
    getch( );
}
```

The preprocessor first replaces all symbolic constants before the program is compiled, so after preprocessing the program and before its compiled, it now looks like

```
#include<stdio.h>
#define TAX_RATE 0.20
main( )
{
```

```
    float income;
    float tax;
    income = 140500.0;
    tax = income * 0.20;
    printf("The tax on %.2f is %.2f\n", income, tax);
    getch( );
}
```

Program output: The tax on 140500.00 is 28100.00.

Q.7. (a) What is the use of header file? Discuss.

Ans. The programming language C provides the concepts of library or header files to store the different types of files in which each file contain pre defined functions with their basic definition. The pre-defined functions are categories in the various categories according to their behaviour and each categories of function are defined within the some files are called header files. The following are the some of the header files:

ALLOC.H	ASSERT.H	BIOS.H	CONIO.H	CTYPE.H
DIR.H	DIRECT.H	DIRENT.H	DOS.H	ERRNO.H
FCNTL.H	FLOAT.H	GENERIC.H	GRAPHICS.H	IO.H
MALLOC.H	MATH.H	MEM.H	PROCESS.H	SEARCH.H
SETJMP.H	SHARE.H	STDARG.H	STDDEF.H	STDIO.H
STDLIB.H	STRING.H	TIME.H		

Different Header Files: The following are the header files with their meanings:

alloc.h : Declares memory management functions (allocation, deallocation, etc.).

assert.h : Defines the assert debugging macro.

bios.h : Declares various functions used in calling IBM-PC ROM BIOS routines.

conio.h : Declares various functions used in calling the DOS console I/O routines.

ctype.h : Contains information used by the character classification and character conversion macros.

dir.h : Contains structures, macros, and functions for working with directories and path names.

direct.h : Defines structures, macros, and functions for working with directories and path names.

dirent.h : Declares functions and structures for POSIX directory operations.

dos.h : Defines various constants and gives declarations needed for DOS and 8086-specific calls.

errno.h : Defines constant mnemonics for the error codes.

fcntl.h : Defines symbolic constants used in connection with the library routine open.

float.h : Contains parameters for floating-point routines.

generic.h : Contains macros for generic class declarations.

graphics.h : Declares prototypes for the graphics functions.

io.h : Contains structures and declarations for low-level input/output routines.

(b) Write a program to show the usage of structure in C.

Ans. A structure is a set of interrelated data. A structure in C is a set of primitive data types, which are grouped together to form new data types. A structure is a mechanism provided by the language to create complex data types.

Program to display a date using structure

```
#include<stdio.h>
struct dat {                                    /*global definition of type date */
        int day;
        int month;
        int year;
};
main( )
{
        struct date today;
        today.day = 16;
        today.month = 03;
        today.year = 2008;
        printf("Todays date is %d/%d/%d.\n", \today.day, today. month, today.year);
}
```

Program output:

Todays date is 16/3/2008.

(c) Write a program in C to find the factorial of a given number by recursion.

Ans.

```
#include<stdio.h>
long factorial(long);
void main( )
{
        long int number;
        printf("\nEnter an integer value:");
        scanf("%ld", &number);
        printf("\n The factorial of %ld is %ld\n", number, factorial(number));
}
long factorial (long x)
{
        if(x==1)
                return 1;
        else
                return x*factorial (x-1);
}
```