Data Structure Using C - ***16MCA11***
Internal Assessment Test I – November 2016

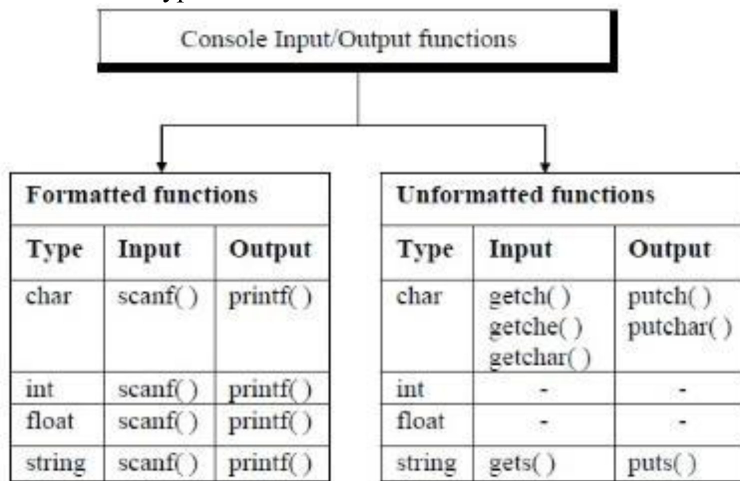
1. a) Explain different types of input and output functions in C **(5)**

DATA INPUT/OUTPUT FUNCTIONS

There are many library functions for input and output in C language.
For using these functions in a C-program there should be preprocessor statement #include<stdio.h>.

Input Function
• The input functions are used to read the data from the keyboard and store in memory-location.
• For ex:
scanf(), getchar(), getch(), getche(), gets()

Output Functions
• The output functions are used to receive the data from memory-locations and display on the monitor.
• For ex:
printf(), putchar(), putch(), puts()
Types of I/O Functions
• There are 2 types of I/O Functions as shown below:



Console Input/Output functions

Formatted functions

Type	Input	Output
char	scanf( )	printf( )
int	scanf( )	printf( )
float	scanf( )	printf( )
string	scanf( )	printf( )

Unformatted functions

Type	Input	Output
char	getch( ) getche( ) getchar( )	putch( ) putchar( )
int	-	-
float	-	-
string	gets( )	puts( )

UNFORMATTED I/O

getchar() and putchar()
• getchar() is used to
→ read a character from the keyboard and
→ store this character into a memory-location
• You have to press ENTER key after typing a character.
• The syntax is shown below:
char variable_name = getchar( );
• For ex: char z; z= getchar( );
• putchar() is used to display a character stored in the memory-location on the screen.

getch() and putch()
• getch() is used to read a character from the keyboard without echo(i.e. typed character will not be

visible on the screen). The character thus entered will be stored in the memory location.
• putch() is used to display a character stored in memory-location on the screen

getche()
• getche() is used to read a character from the keyboard with echo(i.e. typed character will be visible on the screen). The character thus entered will be stored in the memory-location.
gets() and puts()
• gets() is used to
→ read a string from the keyboard and
→ store the string in memory-locations
• puts() is used to display a string stored in memory-locations on the screen.

FORMATTED I/O
scanf()
• The scanf function does following tasks:
→ Scan a series of input fields one character at a time
→ Format each field according to a corresponding format-specifier passed in format-string (format means convert).
→ Store the formatted input at an address passed as an argument i.e. address-list
• Syntax is shown below:
n=scanf("format-string", address-list);
where format-string contains one or more format-specifiers
address-list is a list of variables. Each variable name must be preceded by &

• For ex:

```
n=scanf("%d %f %c", &x, &y, &z);
```

Format specifiers	Meaning
%d	an int argument in decimal
%ld	a long int argument in decimal
%c	a character
%s	a string
%f	a float or double argument
%e	same as %f, but use exponential notation
%o	an int argument in octal (base 8)
%x	an int argument in hexadecimal (base 16)

Printf

The printf function does following tasks:
→ Accept a series of arguments
→ Apply to each argument a format-specifier contained in the format-string
→ Output the formatted data to the screen
• The syntax is shown below:
n=printf("format-string", variable-list);
where format-string contains one or more format-specifiers
variable-list contains names of variables
• For ex:
n=printf("%d %f %c", x, y, z);

b) Explain the use of break statement in loop with an example (5)

THE break STATEMENT

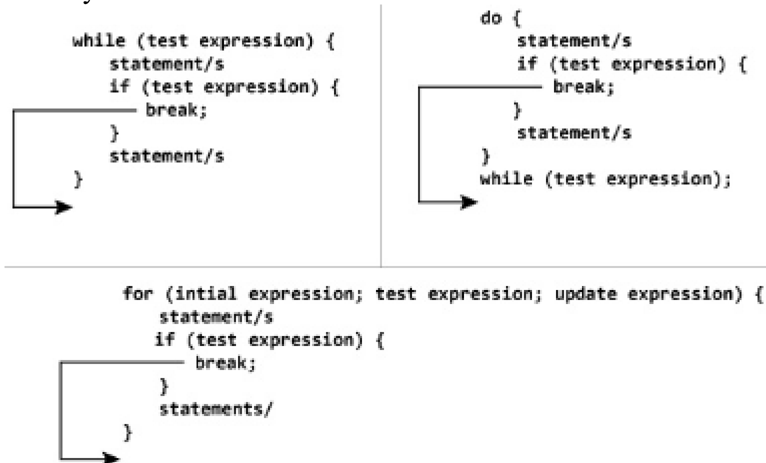
• The break statement is jump statement which can be used in switch statement and loops.

• The break statement works as shown below:

1) If break is executed in a switch block, the control comes out of the switch block and the statement following the switch block will be executed.

2) If break is executed in a loop, the control comes out of the loop and the statement following the loop will be executed.

• The syntax is shown below:




```
void main()
{
char grade; // local variable definition
printf("enter grade A to D: \n");
scanf("%c",&grade);
switch(grade)
{
case 'A': printf("Excellent! \n ");
break;
case 'B': printf("Well done \n ");
break;
case 'C': printf("You passed \n ");
break;
case 'D': printf("Better try again\n ");
break;
default: printf("Invalid grade \n ");
return;
}
printf("Your grade is %c", grade);
return 0;
}
```

Output :

enter grade A to D:

B

Well done

Your grade is B

2. a) Demonstrate call by reference through an example program (5)

When, argument is passed using pointer, address of the memory-location is passed instead of value.
• Example: Program to swap 2 number using call by reference.

```
#include<stdio.h>
void swap(int *a,int *b)
{ // pointer a and b points to address of num1 and num2 respectively
int temp;
temp=*a;
*a=*b;
*b=temp;
}
void main()
{
int num1=5,num2=10;
swap(&num1, &num2); //address of num1 & num2 is passed to swap function
printf("Number1 = %d \n",num1);
printf("Number2 = %d",num2);
}
```

Output:

Number1 = 10
Number2 = 5

Explanation

• The address of memory-location num1 and num2 are passed to function and the pointers *a and *b accept those values.
• So, the pointer a and b points to address of num1 and num2 respectively.
• When, the value of pointer is changed, the value in memory-location also changed correspondingly.
• Hence, change made to *a and *b was reflected in num1 and num2 in main function.
• This technique is known as call by reference.

b) Write a program that uses a function to sort an array of integers (5)

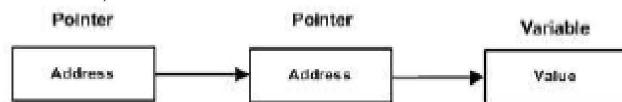
```
/** C PROGRAM TO SORT AN ARRAY USING FUNCTION **/
#include<stdio.h>
#include<conio.h>
#define MAX 100    // maximum no of elements of array
int sortArray(int);
int array[MAX];
int main()
{
  int i,size;
  printf("\n>>>> PROGRAM TO SORT ARRAY USING FUNCTION <<<<\n\n");
  printf("\n Enter the size of array: ");
  scanf("%d",&size);
  printf("\n Enter the %d elements of array: \n",size);
  for(i=0;i<size;i++)
  {
   printf("\n array[%d]=",i);
   scanf("%d", &array[i]);
  }
```

```
 sortArray(size);    //calling sortArray() function
  printf("\n The Sorted elements of array are:");
  for(i=0;i<size;i++)
  {
    printf(" %d",array[i]);
  }
  getch();
  return 0;
}

sortArray(n)   // function for sorting array elements
{
 int temp=0,i,j;   // temp var is temporary variable used for swapping
 for(i=0;i<n;i++)
 {
   for(j=i+1;j<n;j++)
   {
     if(array[i]>array[j])
     {
       temp=array[i];   //swapping for the array to be sorted
       array[i]= array[j];
       array[j]=temp;
     }
   }
 }
}
```

3. What is a pointer? Write a program that uses a function pointer as a function argument (10)

A variable which contains address of a pointer-variable is called pointer to a pointer.
A variable that is a pointer to a pointer must be declared as such. This is done by placing an additional asterisk in front of its name.
For example, following is the declaration to declare a pointer to a pointer of type int:
int **var;



A function, like a variable, has a type and an address location in the memory. It is therefore, possible to declare a pointer to a function, which can then be used as an argument in another function. A pointer to a function is declared as follows:

```
type (*fptr) ();
```

This tells the compiler that **fptr** is a pointer to a function, which returns *type* value. The parentheses around ***fptr** are necessary. Remember that a statement like

```
type *gptr();
```

would declare **gptr** as a function returning a pointer to *type*.

We can make a function pointer to point to a specific function by simply assigning the name of the function to the pointer. For example, the statements

```
double mul(int, int);
double (*p1)();
p1 = mul;
```

declare **p1** as a pointer to a function and **mul** as a function and then make **p1** to point to the function **mul**. To call the function **mul**, we may now use the pointer **p1** with the list of parameters. That is,

(*p1)(x,y) /* Function call */

is equivalent to

mul(x,y)

Note the parentheses around ***p1**.

Program

```
#include  <math.h>
#define  PI  3.1415926
double y(double);
double cos(double);
double table (double(*f)(), double, double, double);

main()
{   printf("Table of y(x) = 2*x*x-x+1\n\n");
    table(y, 0.0, 2.0, 0.5);
    printf("\nTable of cos(x)\n\n");
    table(cos, 0.0, PI, 0.5);
}
double table(double(*f)(),double min, double max, double step)
{   double a, value;
    for(a = min; a <= max; a += step)
    {
       value = (*f)(a);
       printf("%5.2f  %10.4f\n", a, value);
    }
}
double y(double x)
{
   return(2*x*x-x+1);
}
```

4. Define a structure data type. Write a C program to create student structure(USN, Name, Mark1, Mark2) and do the following
   a) Read one student info
   b) Calculate total and average of Marks
   c) Print the student info with total and average

Structure is a collection of elements of different data type.

• The syntax is shown below:

```
struct structure_name
{
data_type member1;
data_type member2;
data_type member3;
};
```

• The variables that are used to store the data are called members of the structure.

• We can create the structure for a person as shown below:

```
struct person
{
char name[50];
int cit_no;
float salary;
};
```
This declaration above creates the derived data type struct person.

```
struct marks
{
char usn[10];
char name[25];
int mark1;
int mark2;
};
main()
{
int tot ;
float avg;
struct marks student;
printf("Input Values\n");
scanf("%s %s %d %d", student.usn, student.name, &student.mark1, &student.mark2);
tot = student.mark1 + student.mark2;
avg = tot/2.0;
printf("\nOutput Values\n");
scanf("Usn :  %s\nName : %s\nMark1 : %d\n Mark2 : %d\n", student.usn, student.name, student.mark1,
student.mark2);
printf("TOTAL   : %d \n", tot);
printf("AVERAGE : %f \n", avg);
}
```

5. a) What is an abstract data type? Write the ADT for an array (5)

a) In computer science, an **abstract data type** (ADT) is a mathematical model for **data types** where a **data type** is defined by its behavior (semantics) from the point of view of a user of the **data**, specifically in terms of possible values, possible operations on **data** of this **type**, and the behavior of these operations.

```
abstract typedef <<eltype, ub>> ARRTYPE(ub, eltype);
condition type(ub) == int;

abstract eltype extract(a,i)          /*   written a[i]     */
ARRTYPE(ub, eltype) a;
int i;
precondition  0 <= i < ub;
postcondition extract == a_i
```

```
abstract store(a, i, elt)                    /* written a[i] = elt */
ARRTYPE (ub, eltype) a;
int i;
eltype elt;
precondition  0 <= i < ub;
postcondition a[i] == elt;
```

b) What is string in C ? Write a C program for comparing two string without using library function (5)

Strings are actually one-dimensional array of characters terminated by a **null**character '\0'. Thus a null-terminated string contains the characters that comprise the string followed by a **null**.

The following declaration and initialization create a string consisting of the word "Hello". To hold the null character at the end of the array, the size of the character array containing the string is one more than the number of characters in the word "Hello."

```
char greeting[6] = {'H', 'e', 'l', 'l', 'o', '\0'};
```

If you follow the rule of array initialization then you can write the above statement as follows −

```
char greeting[] = "Hello";
```

C program to compare two strings without library function

```
#include<stdio.h>
main()
{
        char string1[15],string2[15];
        int i,temp = 0;

        printf("Enter the string1 value: ");
        scanf("%s",string1);

        printf(" Enter the String2 value: ");
        scanf("%s", string2);

        for(i=0; string1[i]!='\0'; i++)
        {
        if(string1[i] == string2[i])
        temp = 1;
        else
        temp = 0;
        }

        if(temp == 1)
        printf("Both strings are same.");

        else
        printf("Both string not same.");

}
```

6. Define Stack. Explain the implementation of push and pop operations. List the applications of stack.

Definition :

Stack is a LIFO(Last in First out data structure). It has only one end from which both insertion and deletion can be done. This is called the top.

Implementation of stack:

```
#include<stdio.h>
#include<conio.h>
#include<stdlib.h>

#define size 5
struct stack {
  int s[size];
  int top;
} st;

int stfull() {
  if (st.top >= size - 1)
    return 1;
  else
    return 0;
}

void push(int item) {
  st.top++;
  st.s[st.top] = item;
}

int stempty() {
  if (st.top == -1)
    return 1;
  else
    return 0;
}

int pop() {
  int item;
  item = st.s[st.top];
  st.top--;
  return (item);
}

void display() {
  int i;
  if (stempty())
    printf("\nStack Is Empty!");
  else {
    for (i = st.top; i >= 0; i--)
```

```
      printf("\n%d", st.s[i]);
  }
}

int main() {
  int item, choice;
  char ans;
  st.top = -1;

  printf("\n\tImplementation Of Stack");
  do {
    printf("\nMain Menu");
    printf("\n1.Push \n2.Pop \n3.Display \n4.exit");
    printf("\nEnter Your Choice");
    scanf("%d", &choice);
    switch (choice) {
    case 1:
      printf("\nEnter The item to be pushed");
      scanf("%d", &item);
      if (stfull())
        printf("\nStack is Full!");
      else
        push(item);
      break;
    case 2:
      if (stempty())
        printf("\nEmpty stack!Underflow !!");
      else {
        item = pop();
        printf("\nThe popped element is %d", item);
      }
      break;
    case 3:
      display();
      break;
    case 4:
      exit(0);
    }
    printf("\nDo You want To Continue?");
    ans = getche();
  } while (ans == 'Y' || ans == 'y');

return 0;
}
```

Application of stacks

1. Conversion of expressions: The compiler converts the infix expressions into postfix expressions using stack.

2. Evaluation of expression: An arithmetic expression represented in the form of either postfix or prefix can be easily evaluated using stack.
3. Recursion: A function which calls itself is called recursive function.
4. Other applications: To find whether the string is a palindrome, to check whether a given expressionis valid or not.

7. a) Write a C Program to convert infix expression to postfix expression (5)

Program

```
#define SIZE 50            /* Size of Stack */
#include <ctype.h>
char s[SIZE];
int top=-1;       /* Global declarations */

push(char elem)
{                       /* Function for PUSH operation */
   s[++top]=elem;
}

char pop()
{                      /* Function for POP operation */
   return(s[top--]);
}

int pr(char elem)
{                 /* Function for precedence */
   switch(elem)
   {
   case '#': return 0;
   case '(': return 1;
   case '+':
   case '-': return 2;
   case '*':
   case '/': return 3;
   }
}

main()
{                        /* Main Program */
   char infx[50],pofx[50],ch,elem;
   int i=0,k=0;
   printf("\n\nRead the Infix Expression ? ");
   scanf("%s",infx);
   push('#');
   while( (ch=infx[i++]) != '\0')
   {
      if( ch == '(') push(ch);
      else
         if(isalnum(ch)) pofx[k++]=ch;
         else
```

```
            if( ch == ')')
            {
                while( s[top] != '(')
                    pofx[k++]=pop();
                elem=pop(); /* Remove ( */
            }
            else
            {       /* Operator */
                while( pr(s[top]) >= pr(ch) )
                    pofx[k++]=pop();
                push(ch);
            }
    }
    while( s[top] != '#')     /* Pop from stack till empty */
        pofx[k++]=pop();
    pofx[k]='\0';          /* Make pofx as valid string */
    printf("\n\nGiven Infix Expn: %s  Postfix Expn: %s\n",infx,pofx);
    }
```

b) Convert the infix expression **A+(B*C-(D/E-F)*G)*H** by using the above algorithm (5)

Stack	**Input**	**Output**
Empty	A+(B*C-(D/E-F)*G)*H	-
Empty	+(B*C-(D/E-F)*G)*H	A
+	(B*C-(D/E-F)*G)*H	A
+(	B*C-(D/E-F)*G)*H	A
+(	*C-(D/E-F)*G)*H	AB
+(*	C-(D/E-F)*G)*H	AB
+(*	-(D/E-F)*G)*H	ABC
+(-	(D/E-F)*G)*H	ABC*
+(-(	D/E-F)*G)*H	ABC*
+(-(	/E-F)*G)*H	ABC*D
+(-(/	E-F)*G)*H	ABC*D
+(-(/	-F)*G)*H	ABC*DE
+(-(-	F)*G)*H	ABC*DE/
+(-(-	F)*G)*H	ABC*DE/
+(-(-	)*G)*H	ABC*DE/F
+(-	*G)*H	ABC*DE/F-
+(-*	G)*H	ABC*DE/F-
+(-*	)*H	ABC*DE/F-G
+	*H	ABC*DE/F-G*-
+*	H	ABC*DE/F-G*-

+*	End	ABC*DE/F-G*-H
Empty	End	ABC*DE/F-G*-H*+

8. What is recursion and compare it with iteration process. Demonstrate recursion by solving tower of Hanoi problem using C

Definition :

- A function is recursive if a statement in the body of the function calls itself.
- Recursion is a name given for expressing anything in terms of itself.
- Recursion is the process of defining something in terms of itself.

Comparison

Recursion v/s Iteration

Iteration	Recursion
1. Process of executing the statements repeatedly	1. Defines in terms of itself
2. Start from base case and incremented in steps till the solution	2. Starts from (n-1)th case recursively and terminates at base case and backtracks to get the solution
3. The value of n, the no. of steps is not fixed	3. N is fixed in the beginning
4. More efficient interms of memory utilization and speed of execution	4. Not as efficient as iterative method, (more space required and less speed)
5. No compactness in programs	5. Compactness in writing programs
6. Any function can be solved using iterative method	6. All functions can't be solved by this method
7. Programmers work is more and takes more code to solve the problem	7. Increases programmers efficiency and makes it easy to understand code

8. It is counter controlled and the body of the loop terminates when the termination condition is failed. Each time the control enters into the loop the counter will be updated.
int fact(int n){
int i, prod = 1;
for(i=0;i<n;i++)
prod=prod*i;
return prod;
}
	8. It is terminated when a base condition is reached. Each time the function is called a simpler version of the original problem is obtained till the base condition is reached.
	int fact(int n){
	if(n==0) return1;
	return n*fact(n-1);
	}

Recursion Program for Tower of Hanoi

- In the game of Towers of Hanoi, there are three towers labeled 1, 2, and 3. The game starts with n disks on tower A.
- For simplicity, let n is 3. The disks are numbered from 1 to 3, and without loss of generality we may assume that the diameter of each disk is the same as its number. That is, disk 1 has diameter 1 (in some unit of measure), disk 2 has diameter 2, and disk 3 has diameter 3.
- All three disks start on tower A in the order 1, 2, 3.
- The objective of the game is to move all the disks in tower 1 to entire tower 3 using tower 2.
- That is, at no time can a larger disk be placed on a smaller disk.

The rules to be followed in moving the disks from tower 1 to tower 3 using tower 2 are as follows:

- Only one disk can be moved at a time.
- Only the top disc on any tower can be moved to any other tower.
- A larger disk cannot be placed on a smaller disk.

Algorithm

- To move n disks from tower A to tower C, using B as auxiliary.
- If n==1, move the single disk from A to C and stop.
- Move the top(n-1) disks from A to B, using C as auxiliary.
- Move the remaining disk from A to C.
- Move the (n-1) disks from B to C, using A as auxiliary.
- Given ‘n’ disks the tower of Hanoi problem takes $2^n - 1$ steps.

```
/*Program 4c Tower of Hanoi
#include<stdio.h>
int count=0;
tower(int n,char src,char temp,char dest)
{
	if(n==1)
	{
		printf("move disc %d from %c to %c\n",n,src,dest);
		return;
	}
	tower(n-1,src,dest,temp);
	printf("move disc %d from %c to %c\n",n,src,dest);
	tower(n-1,temp,src,dest);
}

int main()
{
	int n;
	printf("enter the number of disc\n");
	scanf("%d",&n);
	tower(n,'a','b','c');
}
```