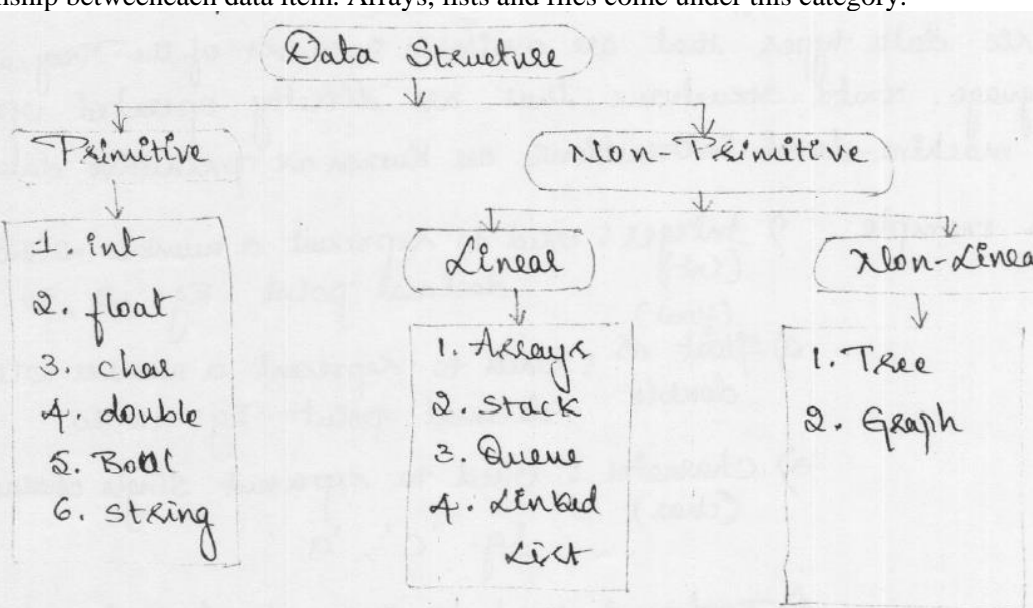


| Internal Assessment Test 1 – Sep-2020 | | | | | | | | | | |
|--|----------------------------------|-----------|---------|------------|-----------|----------|--------------------|-----|------|--|
| Sub: | Data Structures and Applications | | | | Sub Code: | 18CS32 | Branch: | CSE | | |
| Date: | 10/09/2020 | Duration: | 90 mins | Max Marks: | 50 | Sem/Sec: | 3 rd /C | | OBE | |
| <u>Answer any FIVE FULL Questions</u> | | | | | | | MARK S | CO | RB T | |
| <p>1. Define Data Structures. Explain the different types of data structures with examples?</p> <p>Data structure is a representation of logical relationship existing between individual elements of data. In other words, a data structure defines a way of organizing all data items that considers not only the elements stored but also their relationship to each other. The term data structure is used to describe the way data is stored.</p> <p>Data structures are divided into two types:</p> <ul style="list-style-type: none"> • Primitive data structures. • Non-primitive data structures. <p>Primitive Data Structures are the basic data structures that directly operate upon the machine instructions. They have different representations on different computers. Integers, floating point numbers, character constants, string constants and pointers come under this category.</p> <p>Non-primitive data structures are more complicated data structures and are derived from primitive data structures. They emphasize on grouping same or different data items with relationship between each data item. Arrays, lists and files come under this category.</p> <div style="text-align: center; margin-top: 10px;">  <pre> graph TD DS[Data Structure] --> P[Primitive] DS --> NP[Non-Primitive] P --> P1[1. int] P --> P2[2. float] P --> P3[3. char] P --> P4[4. double] P --> P5[5. Bool] P --> P6[6. string] NP --> L[Linear] NP --> NL[Non-Linear] L --> L1[1. Arrays] L --> L2[2. Stack] L --> L3[3. Queue] L --> L4[4. Linked List] NL --> NL1[1. Tree] NL --> NL2[2. Graph] </pre> </div> | | | | | | | [1+2+2] | CO1 | L1 | |
| <p>1.b What is the formula to calculate the location in the row major and column major? Suppose each student in a class of 25 is given 4 tests, assume the students are numbered from 1 to 25, and the test scores are assigned in the 25 X 4 matrix called SCORE. Suppose Base (SCORE)=200, w=4, and the programming language uses column major order to store this 2D array, then find the address of 3rd test of 12th student i.e SCORE(12, 3).?</p> <p>Answer:</p> <p>If we consider element in row r, column c in an array of M x N order, if B is the base address and w is the size of each element in the array, then the address of the elements can be given by</p> | | | | | | | [2+2+1] | CO1 | L4 | |

Row major address = $B + w * (N * (r-1) + (c-1))$
 Column major address = $B + w * (M * (c-1) + (r-1))$

Consider the 25 x 4 matrix array SCORE. Suppose Base (SCORE) = 200 and there are w = 4 words per memory cell. Furthermore, suppose the programming language stores two-dimensional arrays using row-major order. Then the address of SCORE [12,3], the third test of the twelfth student, follows:

$$\text{LOC}(\text{SCORE}[12, 3]) = 200 + 4[4(12 - 1) + (3 - 1)] = 200 + 4[46] = 384$$

Column major

$$\text{LOC}(\text{Score}[12,3]) = 200 + 4(25(3-1) + (12-1)) = 444$$

2 Write the Knuth Morris Pratt pattern matching algorithm and apply the same to search the pattern 'aaaaab' in the text 'aaaaaaaaab'. Demonstrate steps also.

[4+3+3]

CO1

L3

3a. Explain Selection Sort and its working process.

[1.5+1.5+2]

CO1

L1

C function to perform selection sort

```
#include <stdio.h>

int main()
{
    int array[100], n, c, d, position, swap;

    printf("Enter number of elements\n");
    scanf("%d", &n);

    printf("Enter %d integers\n", n);

    for (c = 0; c < n; c++)
        scanf("%d", &array[c]);

    for (c = 0; c < (n - 1); c++)
    {
        position = c;

        for (d = c + 1; d < n; d++)
        {
            if (array[position] > array[d])
                position = d;
        }
        if (position != c)
        {
            swap = array[c];
            array[c] = array[position];
            array[position] = swap;
        }
    }
    printf("Sorted list in ascending order:\n");

    for (c = 0; c < n; c++)
        printf("%d\n", array[c]);

    return 0;
}
```

| | | | |
|--|--|--|--|
| | | | |
|--|--|--|--|

4. What is dynamic memory allocation? Explain different functions associated with dynamic memory allocation and deallocation with syntax and example. Code a C program to illustrate the same for allocating memory to store n integers and find the sum using dynamic memory allocation.

5+2.5+2.5

CO4

L3

Answer:

Dynamic Memory Allocation can be defined as a procedure in which the size of a data structure (like Array) is changed during the runtime.

C provides some functions to achieve these tasks. There are 4 library functions provided by C defined under **<stdlib.h>** header file to facilitate dynamic memory allocation in C programming. They are:

1. malloc()
2. calloc()
3. free()
4. realloc()

Dynamic memory allocation

Dynamic memory allocation is an aspect of allocating and freeing memory according to your needs. Dynamic memory is managed and served with pointers that point to the newly allocated space of memory in an area which we call the **heap**. Now you can create and destroy an array of elements at runtime without any problems.

To sum up, the automatic memory management uses the stack, and the dynamic memory allocation uses the heap.

The **<stdlib.h>** library has functions responsible for dynamic memory management.

| Function | Purpose |
|----------|---|
| malloc | Allocates the memory of requested size and returns the pointer to the first byte of |

allocated space.

calloc

Allocates the space for elements of an array. Initializes the elements to zero and returns a pointer to the memory.

realloc

It is used to modify the size of previously allocated memory space.

Free

Frees or empties the previously allocated memory space.

```
// Program to calculate the sum of n numbers entered by the user
#include<stdio.h>
#include<stdlib.h>

int main()
{
    int n, i, *ptr, sum = 0;

    printf("Enter number of elements: ");
    scanf("%d", &n);

    ptr = (int*) malloc(n * sizeof(int));

    // if memory cannot be allocated
    if(ptr == NULL)
    {
        printf("Error! memory not allocated.");
        exit(0);
    }

    printf("Enter elements: ");
    for(i = 0; i < n; ++i)
    {
        scanf("%d", ptr + i);
        sum += *(ptr + i);
    }

    printf("Sum = %d", sum);

    // deallocating the memory
    free(ptr);

    return 0;
}
```

5.a Consider two polynomials,

$$A(x) = 9x^6 + 3x^2 + 5 \text{ and } B(x) = 12x^3 + 10x^2 + 1$$

Show diagrammatically how these two polynomials can be stored in a 1-D array.

(a) The given polynomials are,

$$A(x) = 9x^6 + 3x^2 + 5$$
$$B(x) = 12x^3 + 10x^2 + 1$$

array

| | | | | | | |
|---|---|---|---|---|---|---|
| 5 | 0 | 3 | 0 | 0 | 0 | 9 |
| 0 | 1 | 2 | 3 | 4 | 5 | 6 |

array

| | | | |
|---|---|----|----|
| 1 | 0 | 10 | 12 |
| 0 | 1 | 2 | 3 |

\therefore In C,

$$\text{int } A(x) [7] = \{5, 0, 3, 0, 0, 0, 9\}$$
$$\text{int } B(x) [4] = \{1, 0, 10, 12\}$$

2.5 * 2

CO3

L3

5b What is a pointer? Explain pointer declaration and initialization with syntax and example

5

Answer:

Pointers in C language is a variable that stores/points the address of another variable. A Pointer in C is used to allocate memory dynamically i.e. at run time. The pointer variable might be belonging to any of the data type such as int, float, char, double, short etc.

- Pointer Syntax : data_type *var_name; Example : int *p; char *p;
- Where, * is used to denote that "p" is pointer variable and not a normal variable.

```
#include <stdio.h>
int main()
{
    int *ptr, q;
    q = 50;
    /* address of q is assigned to ptr */
    ptr = &q;
    /* display q's value using ptr variable */
    printf("%d", *ptr);
    return 0;
}
```

6. Define stack. Write a C program demonstrating the various stack operations, including cases for overflow and underflow of stacks

A stack is a container of objects that are inserted and removed according to the last-in first-out (LIFO) principle. In the pushdown stacks only two operations are allowed: **push** the item into the stack, and **pop** the item out of the stack. A stack is a limited access data structure - elements can be added and removed from the stack only at the top. **push** adds an item to the top of the stack, **pop** removes the item from the top. A helpful analogy is to think of a stack of books; you can remove only the top book, also you can add a new book on the top.

```
#include<stdio.h>

int stack[100],choice,n,top,x,i;
void push(void);
void pop(void);
void display(void);
int main()
{
    //clrscr();
    top=-1;
    printf("\n Enter the size of STACK[MAX=100]:");
    scanf("%d",&n);
    printf("\n\t STACK OPERATIONS USING ARRAY");
    printf("\n\t-----");
    printf("\n\t 1.PUSH\n\t 2.POP\n\t 3.DISPLAY\n\t 4.EXIT");
    do
    {
        printf("\n Enter the Choice:");
        scanf("%d",&choice);
        switch(choice)
        {
            case 1:
            {
                push();
                break;
            }
            case 2:
            {
                pop();
                break;
            }
            case 3:
            {
                display();
                break;
            }
            case 4:
            {
                printf("\n\t EXIT POINT ");
                break;
            }
            default:
            {
                printf ("\n\t Please Enter a Valid Choice(1/2/3/4)");
            }
        }
    }
    while(choice!=4);
    return 0;
}
void push()
{
    if(top>=n-1)
    {
        printf("\n\tSTACK is over flow");
    }
}
```

```
else
{
    printf(" Enter a value to be pushed:");
    scanf("%d",&x);
    top++;
    stack[top]=x;
}
}
void pop()
{
    if(top<=-1)
    {
        printf("\n\t Stack is under flow");
    }
    else
    {
        printf("\n\t The popped elements is %d",stack[top]);
        top--;
    }
}
void display()
{
    if(top>=0)
    {
        printf("\n The elements in STACK \n");
        for(i=top; i>=0; i--)
            printf("\n%d",stack[i]);
        printf("\n Press Next Choice");
    }
    else
    {
        printf("\n The STACK is empty");
    }
}
}
```

7. Using nested structures, write a C program to store and display employee information for the following field Ename, Empid, DOB (Date, Month, Year) and salary (Basic, DA, HRA).

Answer:

```
7. struct employee
{
    int Empid;
    char Ename [30];
    struct DOB dob
    {
        int Date;
        char Month [10];
        int Year;
    } DOB;
    struct salary
    {
        float Basic;
        float DA;
        float HRA;
    } salary;
} emp;
```

```
void read()
{
    printf (" Enter Employee ID ");
    scanf ("%d", &emp.Empid);
    printf (" Enter Employee name ");
    scanf ("%s", emp.Ename);
    printf (" Enter date, month and year
    birth ");
    scanf ("%d", &emp.DOB.Date);
    scanf ("%s", emp.DOB.Month);
    scanf ("%d", &emp.DOB.Year);
    printf (" Enter the salary as Basic, DA
    HRA ");
    scanf ("%f", &emp.salary.Basic);
    scanf ("%f", &emp.salary.DA);
    scanf ("%f", &emp.salary.HRA);
}
```



```

void print()
{
    printf("Employee ID : %d", emp.Empid);
    printf("Employee Name : %s", emp.Ena);
    printf("DOB : %d %s %d",
           emp.DOB.date, emp.DOB.
           emp.DOB.year);
    printf("Salary : \n");
    printf("Basic : %f", emp.salary.Bas);
    printf("DA : %f", emp.salary.DA);
    printf("HRA : %f", emp.salary.HRA);
}

```

8.a

4

CO1

L2

Explain Sparse Matrix representation with example.

Sparse matrix: Sparse matrix with a very few non-zero elements and more zero elements.

| | | | | | | | | | | | |
|---|---|---|---|---|-----|---|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 4 | Row | 4 | 0 | 1 | 1 | 2 | 3 |
| 8 | 0 | 0 | 2 | 0 | col | 4 | 4 | 0 | 3 | 1 | 2 |
| 0 | 5 | 0 | 0 | 0 | val | 5 | 4 | 8 | 2 | 5 | 6 |
| 0 | 0 | 6 | 0 | 0 | | | | | | | |

The sparse matrix can be stored in the form of triples

Row: It consists of the row indices of non-zero elements

column: It consists of the column indices of non-zero elements

val: It consists of all the non-zero elements.

The first column which has entries 4, 4, 5 represent the no. of rows, no. of columns and total no. of non-zero elements respectively

8.b

6

CO2

L3

Design, Develop and Implement a menu driven Program in C for the following array operations.

- a. Creating an array of N Integer Elements**
- b. Display of array Elements with Suitable Headings**
- c. Inserting an Element (ELEM) at a given valid Position (POS)**
- d. Deleting an Element at a given valid Position (POS)**
- e. Exit.**

Support the program with functions for each of the above operations.

```
/* Function definition to create an array of n elements. */
int create(int n, int a[])
{
    int i;

    printf("\nEnter the number of elements: ");
    scanf("%d", &n);

    printf("\nEnter the elements:\n");
    for( i = 0; i < n ; i++ )
    {
        scanf("%d", &a[i]);
    }

    return n;
}

/* Function definition to display array elements. */
void display(int n, int a[])
{
    int i;

    if(n==0)
    {
        printf("\nNo elements to display!!!");
    }
    else
    {
        printf("\nThe elements are:\n");
        for( i = 0; i < n ; i++ )
        {
            printf("%d\t", a[i]);
        }
    }
}
```

```

/* Function definition to delete an element at a given valid position. */
int delete(int n, int a[])
{
    int pos, i;

    printf("\nEnter the position from which you want to delete: ");
    scanf("%d", &pos);

    if( pos<1 || pos>n )
    {
        printf("Invalid Position!!!");
    }
    else
    {
        printf("\nThe deleted element is: %d", a[pos-1]);

        for( i = pos-1 ; i < n-1 ; i++ )
        {
            a[i] = a[i+1]; /* Copy the element in i+1th location to ith location. */
        }

        n--; /* Decrement the count of elements in the array. */
    }

    return n;
}

int main()
{
    int ch, n, a[20];

    while(1) /*Infinite Loop.*/
    {
        printf("\nARRAY OPERATIONS");
        printf("\n-----");
        printf("\n1: CREATE");
        printf("\n2: DISPLAY");
        printf("\n3: INSERT AN ELEMENT AT GIVEN POS");
        printf("\n4: DELETE AN ELEMENT FROM GIVEN POS");
        printf("\n5: EXIT");
        printf("\n-----");

        printf("\nEnter your choice: ");
        scanf("%d", &ch);

        switch(ch)
        {
            case 1: n = create(n,a);
                    break;

            case 2: display(n, a);
                    break;

            case 3: n = insert(n, a);
                    break;

            case 4: n = delete(n, a);
                    break;

            case 5: return ( 0 );
        }
    }
}

```

```

/* Function definition to insert an element at a given valid position. */
int insert(int n, int a[])
{
    int ele, pos, i;

    printf("\nEnter the element to be inserted: ");
    scanf("%d", &ele);

    printf("\nEnter the position at which you want to insert: ");
    scanf("%d", &pos);

    if( pos<1 || pos>n )
    {
        printf("Invalid Position!!!");
    }
    else
    {
        for( i = n-1 ; i >= pos-1 ; i-- )
        {
            a[i+1] = a[i]; /* Copy the element in ith location to i+1th location. */
        }

        a[pos-1] = ele; /* Insert the element. */
        n++; /* Increment the count of elements in the array. */
    }

    return n;
}

```