Module I-Arrays, Strings,

Dr.Ganga Holi, Prof, & Head, ISE Dept.

Global Academy of Technology

ept. of Information Science & Engineering

Contents

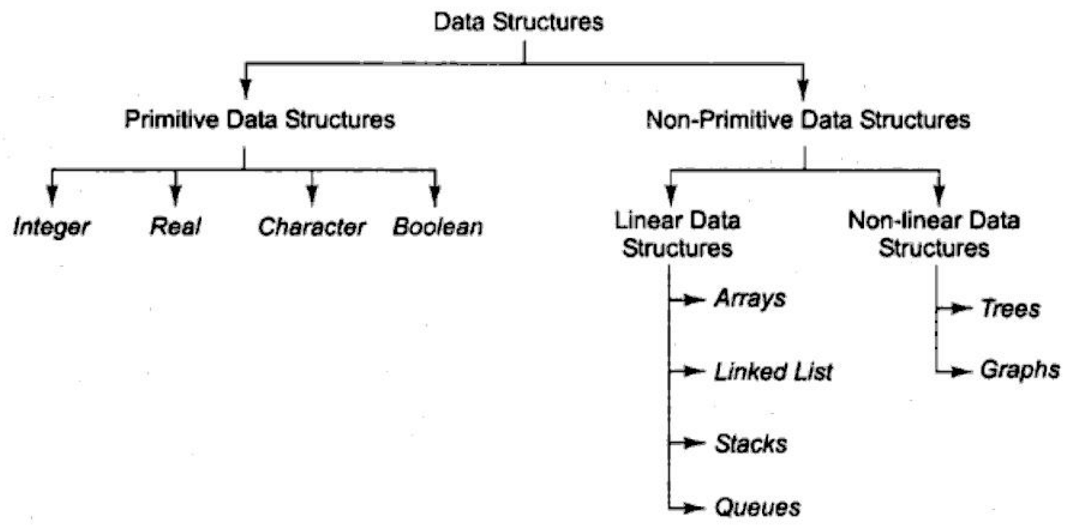
1. What is Data Structure?

Data can be organized in many different ways. The logical or mathematical model of a organization of data is called a Data Structure.

2. Explain the classification of data structures.

Data structures are generally classified into primitive and non-primitive data structures. Basic data types such as integer, real, character and boolean are known as primitive data structures. These data types consist of characters that cannot be divided, and hence they are also called as simple datatypes.

The simplest example of non-primitive data structure is the processing of complex numbers. Very few computers are capable of doing arithematic on complex numbers. Linked lists, stacks, queues, trees and graphs are examples of non-primitive data structures.



Based on the structures and arrangement of data, non-primitive data structures are further classified into linear and non-linear.

A data structure is said to be linear if its elements form a sequence or a linear list. In linear data structures, the data is arranged in a linear fashion although the way they are stored in memory need not be sequential. Arrays, linked list, stacks and queues are the examples of linear data structures.

Conversely, a data structure is said to be a non-linear if the data is not arranged in sequence. The insertion and deletion of data is therefore not possible in a linear fashion. Trees and graphs are examples of non-linear data structures.

3. What are the data structures operations?

The data appearing in our data structures are processed by means of certain operations. In fact, the particular data structure that one chooses for a given situation depends largely on the frequency with which specific operations are performed.

1. Traversing: Accessing each record exactly once so that certain items in the record maybe processed. (This accessing and processing is sometimes called "visiting" the record).
2. Searching: Finding the location of the record with a given key value, or finding the locations of all records which satisfy one or more conditions.
3. Inserting: Adding a new record to the structure.
4. Deleting: Removing a record from the structure.
The following two operations are used in special situations,
5. Sorting: Arranging the records in some logical order.
6. Merging: Combining the records of two different sorted files into a single sorted file.

4. Explain declaration, initialization of One dimensional and Two dimensional arrays.

One Dimensional Array

Declaration:

Syntax: data_type array_name[array_size];

where, data_type can be int, float or char.

array_name is the name of the array.

array_size indicates number of elements in the array.

For ex: int age[5];

Initialization:

Syntax: data_type array_name[array_size]={v1,v2,v3};

where, v1, v2, v3 are the values

For ex: int age[5]={2,4,34,25,18};

Two Dimensional Array

Declaration:

Syntax: data_type array_name[array_size][array_size];

where, first index shows the row number of the element and

second index shows the coulmn number of the element.

Initialisation:

Syntax: data_type array_name[array_size][array_size]={V1,V2,V3,.....Vn};

where, V1, V2, V3,......,Vn are the values

For ex: int matrix[2][3]={2,4,56,3,6};

5. Discuss the operations that can be performed on arrays.

(i)Traversal

Let **A** be a collection of data elements stored in the computer's memory. To print the contents of each element of **A** or count the elements of **A** with a given property, each element of **A** will have to be accessed or processed at least once. This is called **traversing.**

Algorithm for Array Traversal

Let **A** be a linear array with lower bound LB upper bound UB. The following algorithm traverses **A** applying an operation PROCESS to each element of **A**.

```
1. Initialization counter
        Set counter = LB
2. Repeat steps 3 and 4 while counter <= UB
3. Visit element
        Apply PROCESS to arr[counter]
4. Increase counter
        Set counter = counter + 1
5. Exit
```

(ii) Insertion

Insertion refers to the operation of adding another element to the array. If an element is to be inserted at the end of the array, then the task is easily done provided, there is enough space to accommodate additional elements in the array.

But if we need to insert in the middle of an array then half of the elements must be moved downwards to new locations to accommodate the new element and retain the order of other elements.

Algorithm for Insertion

Let A be a linear array--The function used is INSERT(A, N, K, ITEM). N is the number of items, K is the positive integer such that K<=N. The following algorithm inserts an element ITEM into the Kth position of array A.

```
1. Initialize Counter
```

```
        Set J := N
2. Repeat Steps 3 and 4 while J >= K
3. Move Jth element downward
        Set[J+1] := A[J]
4. Decrease counter
        Set J := J-1
        End of step 2 loop
5. Insert element
        Set A[K] := ITEM
6. Reset N
        Set N := N+1
7. Exit
```

(iii) Deletion

Deletion refers to the operation of removing an element from the array. Deleting the element from the end of an array is not a problem, but deleting from middle requires movement of each element upwards in order to fill up the void in the array.

Algorithm for Deletion

Let A be a linear array. The function used to delete from the array is DELETE(a, N, K, ITEM) where, N is the number of elements, K is the positive integer such that K<=N. The algorithm delets Kth element from the array.

```
1. Set ITEM := A[k]
2. Repeat for J - K to N-1
        [Move J+1 element upward]
                Set A[J] := A[J+1]
        End of loop
3. Reset the number N of elements in A
        Set N := N-1
4. Exit
```

6. Describe the method to calculate the actual memory address of the array location.

Address Calculation in One Dimension Array:

Array of an element of an array say “A[I]” is calculated using the following formula:

Address of A [I] = B + W * (I – LB)

Where,

B = Base address

W = Storage Size of one element stored in the array (in byte)

I = Subscript of element whose address is to be found

LB = Lower limit / Lower Bound of subscript, if not specified assume 0 (zero)

Example: Given the base address of an array B[1300…..1900] as 1020 and size of each element is 2 bytes in the memory. Find the address of B[1700].

Solution: The given values are: B = 1020, LB = 1300, W = 2, I = 1700

Address of A [I] = B + W * (I – LB)

= 1020 + 2 * (1700 – 1300)

= 1020 + 2 * 400

= 1020 + 800

= 1820 [Ans]

Address Calculation in Two Dimensional Array:

Address of A [I][j] = B + W *((i-LB)*(UB-LB+1)+ (j – LB))
Example:
Hint: Formula for calculating address for multi dimensional arrays
L_i=upper bound-lower bound+1, where L is the number of elements in Column.
For a given subscript $K_{i,}$ the effective index E_i of Li is the number of indices preceding K_i in the index set and E_i can be calculated from
E_i=K_i-lower bound

Then the address LOC(K1, K2, K3…...Kn) of an ordinary element of C array can be obtained from the formula
Base(C)+w[(((…(E_nL_{n-1} +E_{n-1})L_{n-2} + E_{n-2})L_{n-3} + +E_3)L_2 + E_2)L_1 +E_1)
or
from the formula Base(C)+ w[(…((E_1L_2 +E_2)L_3 + E_3)L_4 + ...+$E_{n-1)}$Ln+ E_n)
While storing the elements of a 2-D array in memory, these are allocated contiguous memory locations. Therefore, a 2-D array must be linearized so as to enable their storage. There are two alternatives to achieve linearization: Row-Major and Column-Major.
Address of an element of any array say "A[I][J]" is calculated in two forms as given:
(1) Row Major System (2) Column Major System
Row Major System: The address of a location in Row Major System is calculated using the following formula:
Address of A [I][J] = B + W * [N * (I – Lr) + (J – Lc)]
Column Major System: The address of a location in Column Major System is calculated using the following formula:
Address of A [I][J] Column Major Wise = B + W * [(I – Lr) + M * (J – Lc)]
Where, B = Base address
I = Row subscript of element whose address is to be found
J = Column subscript of element whose address is to be found
W = Storage Size of one element stored in the array (in byte)
Lr = Lower limit of row/start row index of matrix, if not given assume 0 (zero)
Lc = Lower limit of column/start column index of matrix, if not given assume 0 (zero)
M = Number of row of the given matrix
N = Number of column of the given matrix

7. What are Polynomials and Sparse matrices? Explain with example.

Polynomial is an expression consisting of variables and co-efficients which only employs operations of addition, subtraction and multiplication.
Ex: x^2-4x+7 is a single variable equation.
x^3+xyz^2+x^2y+z+10 is a 3 variable polynomial equation.
The largest exponent (power) of a polynomial is called its degree. Co-efficients that are zero are not displayed.

Program to evaluate polynomial

How to represent polynomial and Evaluate polynomials?

```
// Evaluation of polynomial equation
#include<stdio.h>
typedef struct
{
        int coef,power;
}term;

term a[100];
int main()
{   int i, x, n,m, sum=0;
    i=0;
    printf("Enter n=" );
        scanf("%d", &n);
        printf(" Enter Polynomial Coef and power=");
        for(i=0;i<n;i++)
                scanf("%d %d", &a[i].coef, & a[i].power);
        printf("Enter x=" );
        scanf("%d", &x);

    for(i=0;i<n;i++)
    {   sum=sum+a[i].coef*pow(x,a[i].power);
        }
        printf("Sum=%d\n",sum);
}
```

Program for Addition of two polynomials

How to represent polynomial and add two polynomials?

```
#include<stdio.h>
typedef struct
{
        int coef,power;
}term;

term a[100];
int startA, startB, finishA, finishB, avail, count=0;
int compare(int a, int b)
{   if(a>b)
        return 1;
    else
    if(a<b)
        return -1;
    else
        return 0;
}
```

```
void attach(int coef, int power)
{          printf("avail=%d\n",avail);
           if(avail>=100)
           {
                     fprintf(stdout, "cannot add poly\n");
                     return ;
           }
           a[avail].coef=coef;
           a[avail].power=power;
           printf("added coef=%d  ",a[avail].coef);
           avail++; count++;
}

void display()
{   int i=0, k;
           printf(" Ist polynomial=\n");
           for(;i<=finishA;i++)
                     printf("%d  %d\n",a[i].coef,a[i].power);
           printf(" IIst polynomial=\n");
           for(;i<=finishB;i++)
                     printf("%d  %d\n",a[i].coef,a[i].power);
           printf(" \n \n Result polynomial=\n");
           k=i;
           for(k=i;k<avail;k++)
                     printf("%d  %d\n",a[k].coef,a[k].power);
}


void polyadd()
{
           int t,  coeff;
           avail=finishB+1;
           printf("startA=%d   startB=%d\n", startA, startB);
           while(startA<=finishA && startB<=finishB)
           {
                     t=compare(a[startA].power,a[startB].power);
                     printf("t=%d \n",t);
                     switch(t)
                     {
                               case -1 : attach(a[startB].coef, a[startB].power);
                                                     startB++; break;
                               case 0  : coeff=a[startA].coef+a[startB].coef;
                                       attach(coeff, a[startA].power);
                                                     startA++; startB++; break;
                               case 1  : attach(a[startA].coef, a[startA].power);
                                                     startA++; break;

                     }
           }
                     for(;startA<=finishA;startA++)
                               attach(a[startA].coef, a[startA].power);
```

```
                    for(;startB<=finishB;startB++)
                              attach(a[startB].coef, a[startB].power);
          }

int main()
{   int i, n,m;
   i=0; startA=0;
   printf("Enter n=" );
          scanf("%d", &n);
          printf("Enter m=" );
          scanf("%d", &m);
          printf(" Enter Ist Polynomial Coef and power=");
          for(i=0;i<n;i++)
                    scanf("%d %d", &a[i].coef, & a[i].power);
          finishA=i-1; startB=i;
          printf(" Enter Ist Polynomial Coef and power=");
          for(;i<n+m;i++)
                    scanf("%d %d", &a[i].coef, & a[i].power);
          finishB=i-1; avail=i;
          polyadd();
          display();
          return 0;
}
```

Operations on Sparse Matrix: (i) Create (ii) Transpose (iii) Add (iv) Multiply

Program to create Sparse matrix

```
#include<stdio.h>
#define MAX_TERMS 100
typedef struct
{
          int row;
          int col;
          int value;
}term;
term a[MAX_TERMS];
int main()
{
          int i,j;
          printf("Enter size of sparse matrix:");
          printf("Max rows=%d",a[0].row);
          printf("Max col=%d",a[0].col);
          printf("Total ele=%d \n",a[i].value);
          printf("Enter % elements \n",a[0].value);
          for(i=0;i<a[i].value;i++)
                    scanf("%d%d%d",a[i].row,a[i].col,a[i].value);
          printf("Output of the matrix \n);
          k=1;
```

```
        for(i=0;i<a[0].row;i++)
        {
                for(j=o;j<a[0].col;j++)
                {
                        if(i==a[k].row && j==a[k].col)
                        {
                                printf("%d\t",a[k].value);
                                k++;
                        }
                        else
                                printf("0 \t");
                }
        }
        printf("\n");
}
```

Transpose of a sparse Matrix

Write an algorithm to transpose sparse matrix.

```
#include<stdio.h>
typedef struct
{
        int row, col, value;
}term;
term a[20],b[20];
int main()
{   int i, n, j,current=0;;
   printf("Enter size of Matrix row & col and Total elemnts =\n" );
        scanf("%d%d%d", &a[0].row, &a[0].col,&a[0].value);
        printf(" Enter sparse matrix elements\n");
        for(i=1;i<=a[0].value;i++)
        {       printf(" row  col and value=");
                scanf("%d%d%d", &a[i].row, &a[i].col, &a[i].value);
        }
        printf(" Transpose of sparse matrix\n\n");
        current=1; n=a[0].value;
        b[0].col=a[0].row;
        b[0].row=a[0].col;
        b[0].value=n;
        if(n>0)
        {
                for(i=0;i<a[0].col;i++)
                {
                        for(j=1;j<=n;j++)
                        {
                                if(a[j].col==i)
                                {
                                        b[current].row=a[j].col;
                                        b[current].col=a[j].row;
                                        b[current].value=a[j].value;
                                        current++;
                                }

                        }
                }
        }
        printf(" \n Input  matrix\n\n");
        for(i=0;i<=n;i++)
                printf("%d  %d  %d\n",a[i].row,a[i].col,a[i].value);

        printf(" \n Transpose of matrix\n\n");
        for(i=0;i<=n;i++)
                printf("%d  %d  %d\n",b[i].row,b[i].col,b[i].value);
        return 0;
}
```

8. What are strings? Explain the operations on strings.

String : Strings are one-dimensional array characters. The length of a string is determined by a terminating null character.

Initialization : Strings can be initialized in the following way :

char string_name[string_size];

Example : char greeting[6]={'H','E','L','L','O'};

```
#include<stdio.h>
int main()
{
        char greeting[6]={'H','E','L','L','O'};
        printf("Greeting message : %s\n",greeting);
        return 0;
}
```

Operations on string :

1. Strcpy(dest,src);

This operation copies string source to destination string.

```
int main()
{
        char src[50],dest[50];
        printf("Enter source");
        gets(src);
        printf("Enter destination string");
        gets(dest);
        strcpy(dest,src);
        printf("Copied string is %s\n",dest);
        return 0;
}
```

Strcat(dest,char) :

This operation concatenates string string vhar into the end of the string dest.

```
int main()
{
        char src[20],dest[20];
        strcpy(src,"This is source");
        strcpy(dest,"This is destination");
        strcat(dest,src);
        printf("Final destination string is %s\n",dest);
        return (0);
}
```

Strlen(str) : This operation returns thr length of the entered string str.

```
int main()
{
        char str[50];
int len;
printf("Enter string");
gets(str);
len=strlen(str);
printf("Length of %s is %d\n",str,len);
return (0);
}
```

Strcmp(s1,s2) :

This returns 0 if s1 and s2 are same, less than 0 if s1<s2, greater than 0 if s1>s2.

```
int main()
{
        char s1[50],s2[50];
        int ret;
printf("Enter string s1");
gets(s1);
printf("Enter string s2");
gets(s2);
ret=strcmp(s1,s2);
if(ret<0)
{
                printf("s1<s2");
}
else if(ret>0)
{
                printf("s1>s2");
}
else
{
                printf("s1=s2");
}
return (0);
}
```

Strchr(s1,s2) :
This returns a pointer to the first occurrence of characterch in the string s1.

```
int main()
{
        const char s1='Tea & Biscuits';
        const char s2='&';
        char ret;
        ret=strchr(s1,ch);
        printf("String after %cis %s",ch,ret);
return (0);
}
```

Strstr(s1,s2) :
Returns a pointer to the first occurrence of string s2 in string s1.

```
int main()
{
        const char s1[50]="Global Academy of Technology";
        const char s2[50]="Academy";
        char ret;
ret=strstr(s1,s2);
printf("The substring is %s\n",ret);
return (0);
}
```

Char strncpy(dest,src)
This operation copies only 'n' characters from src string intp destination string and returns dest string.
Char strncat(dest,char) :
This concatenates dest and only n characters from src, returning in dest.
Strcmp(s1,s2,n)
This operation compares first 'n' characters of the string entered.

Q9. What do you mean by pattern matching? Where do we use this concept?
Pattern matching : A code to check if the given string is present in another string is pattern matching.
For example : "Global is present in Global Academy of Technology" . If the string is present then its location is printed.
Assume that we have two strings, string and pat, where pat is a pattern to be searched. If pattern is present in the string, then it returns its position else returns -1. it examines each character of the string until it finds the pattern or it reaches the end of the string.
Example :

```
#include<stdio.h>
#include<string.h>
int match(char[],char[])
int main()
{
        char a[100],b[100];
        int position;
        printf("Enter    text\n");
gets(a);
printf("Enter a string to find");
gets(b);
position=match(a,b)
if(position!=-1)
{
        printf("Found at location %d\n",position+1);
}
else
{
        printf("Not found");
}
return (0);
}
int match(char text[],char pattern[])
{
        int c,d,e,txt_length,pattern_length,position=-1;
        txt_length=strlen(txt);
        pattern_length=strlen(pattern);
        if(pattern_length>txt_length)
{
                return -1;
        }
        for(c=0;c<=txt_length-pattern_length;c++)
        {
                position=e=c;
                for(d=0;d<pattern_length;d++)
                        {
                                if(pattern[d]==text[e])
                                {
                                        e++;
                                }
                                else
                                {
                                        break;
                                }
                        }
                        if(d==pattern_length)
                        {
                                return position;
                        }
                }
        return -1;
```

}

Q10.Define user defined data types : Structures and Unions.

Structures : A structure is a collection of data items of different data types.

```
        struct structure_name
        {
            member 1;
            member 2;
            member 3;
            member n;
        };
```

Ex : struct person

```
    {
        int age;
        float salary;
        char name[10];
    }p;
```

We can address data members of the structure using dot (.) notation.

```
    person.age=40 in first case and
    p.age =40 in second case.
```

Another way of defining structure is as follows :

```
typedef struct
{
char name[10];
    int age;
    float salary;
}human;
```

This says that 'Human' is the name of the type defined by the structure definition and we may follow this definition with declarations of variables such as : human person 1,person2;

We may use

```
    person1.age=40;person2.salary=10000;
```

ex : To compare two structures we might call a function by passing a parameter

```
    if(humanequal(person1,person2))
         printf("Two persons are same\n");
    else
        printf("Two persons are not same");
int humanequal(human person1,human person2)
{
    if(strcmp(person1.name,person2.name))
    return 0;
    if(person1.age==person2.age)
    return 0;
    if(person1.salary==person2.salary)
    return 0;
return 1;
}
```

We can also embed a structure within a structure. For example, associated with our human structure, we may wish to include the date of his or her birth.

```
typedef struct{
        int month;
        int day;
        int year;
}date;
typedef struct
{
        int age;
        float salary;
        char name[50];
        date dob;
}human;
```

We can assign data to a structure variable as follows :

```
human person1, person 2;
person1.age=40;
person1.dob.month=10;
person1.dob.year=1997;
```

Unions : A union declaration is similar to a structure, but the fields of a union must share their memory space. This means that only one field of the union is "active" at any given time.

```
        typedef struct{
        enum tagfield{female,male}gender;
        union{
                int children;
                int beard;
                }u;
        }gendertype;
        typedef struct{
                char name[10];
                int age;
                float salary;
                date dob;
                gendertype genderinfo;
        }human;
        human person1;
        person1.genderinfo.gender=male;
        person1.genderinfo.u.children=4;
```

Self Referential Structures :

A self referential structure is one in which one or more of its components is a pointer to itself. Self referential structures usually require dynamic storage management routines (malloc,free) to explicitly obtain and release memory cinsider as an example.

```
typedef stuct{
        char data;
        struct list *link;
}list;
```

There the structure will have two components, data and link, Data is a single character, while link is a pointer to a list structure.

```
list l1, l2, l3;
        l1.link=l2.link;
```

Q11. What are pointers? How do we use pointers to different data types?

A pointer is a variable which contains the address in memory of another variable.

Declaration :

```
        data_type *variable_name;
```

Ex : int *a;

Ex :

```
#include<stdio.h>
int main()
{
        int var = 20;
        int *ip;
        ip=&var;
printf("Address of variable is %x\n", &var);
        printf("Address stored in ip variable is :%x",ip);
        printf("value of *ip variable : %d", *ip);
        return 0;
}
```

Q12. Why do we need dynamic memory allocation techniques? Explain the functions available for allocating memory dynamically.

Dynamic memory allocation : It refers to performing manual management for dynamic memory allocation using standard library functions such as malloc, realoc, calloc and free.

The size of array initially declared can be sometimes insufficient or sometimes more than required but dynamic memory allocation allows a program to obtain more memory space, while running or to release space when not required.

Functions:

There are four standard library functions under "stdlib.h" for dynamic memory allocation.

malloc() – It allocates requested size of bytes and returns a pointer first byte of allocated space.

Syntax :

ptr=(data_type *)malloc(bysize);

Ex : (int*)malloc(100*sizeof(int));

calloc() – Allocates spaces for array elements, initializes to zero and then returns a pointer to memory.

Syntax : ptr=(data_type*)calloc(n,element_size);

Ex : ptr=(float*)calloc(25,sizeof(float));

realloc() – Changes the size of the previously allocated space according to the requirement.

Syntax : ptr=realloc(ptr,newsize);
free() – It deallocates the previously allocated space.
Syntax – free(ptr);

Q13.What is sparse matrix? How to represent a sparse matrix?

Sparse Matrix : If a matrix contains more zero entities then suc a matrix is called a sparse matrix.

Ex: $$\begin{bmatrix} 0&5&0&4&0&0 \\ 2&0&0&0&1&0 \\ 0&5&0&0&1&0 \\ 5&0&0&0&0&0 \\ 0&0&0&0&1&0 \\ 0&0&0&0&0&1 \end{bmatrix}$$

The concept of sparse matrix is used in scientific or engineering applications.
Representation :
Array representation : 2D array having n-rows and 3-coloumns,where n is number of non-zero elements.
a[i][0] – represents row value
a[i][1] – represents column value
a[i][2] – represents matrix value

1	2	5
1	4	4
2	1	2
2	5	1
3	2	5
3	5	1
4	5	1
5	6	1

Array of structures representation :

Triple values represent row, column, value.

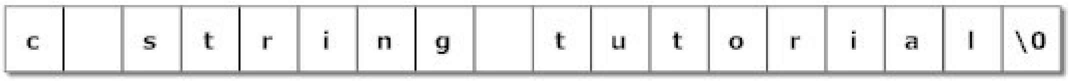
```
typedef struct
{
        int row;int col;int value;
}term;
term a[MAX_TERMS];
```

14. **Define string. Explain string handling functions: strcat, strcpy, strcmp, strncmp, strncat, strchr, strrchr, strtok, strstr, strspn, strcspn, strbrk**

In C programming, array of character are called strings. A string is terminated by null character /0. For example:

"c string tutorial"

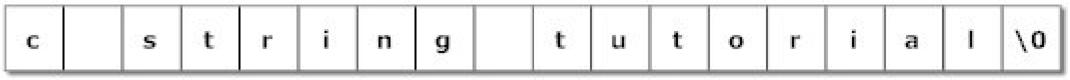
Here, "c string tutorial" is a string. When, compiler encounters strings, it appends null character at the end of string.



Declaration of strings

Strings are declared in C in similar manner as arrays. Only difference is that, strings are of char type.

char s[5];



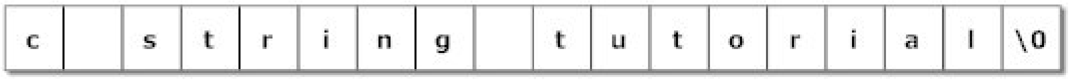
Strings can also be declared using pointer.

char *p

Initialization of strings

In C, string can be initialized in different number of ways.

```
char c[]="abcd";        OR,    char c[5]="abcd";      OR,    char c[]={'a','b','c','d','\0'};
   OR;
char c[5]={'a','b','c','d','\0'};
```



String can also be initialized using pointers

char *c="abcd";

Reading words from user.

```
char c[20];
scanf("%s",c);
```

String variable *c* can only take a word. It is beacause when white space is encountered, the scanf() function terminates.

Write a C program to illustrate how to read string from terminal.

```
#include <stdio.h>
int main()
{
   char name[20];
```

```
    printf("Enter name: ");
    scanf("%s",name);
    printf("Your name is %s.",name);
    return 0;

}
```

Output

```
Enter name: Dennis Ritchie
Your name is Dennis.
```

Here, program will ignore Ritchie because, scanf() function takes only string before the white space.

Reading a line of text

C program to read line of text manually.

```
#include <stdio.h>
int main()
{
    char name[30],ch;
    int i=0;
    printf("Enter name: ");
    while(ch!='\n')    // terminates if user hit enter
    {
        ch=getchar();
        name[i]=ch;
        i++;
    }
    name[i]='\0';       // inserting null character at end
    printf("Name: %s",name);
    return 0;
}
```

This process to take string is tedious. There are predefined functions gets() and puts in C language to read and display string respectively.

```
int main()
{
    char name[30];
    printf("Enter name: ");
    gets(name);     //Function to read string from user.
    printf("Name: ");
    puts(name);    //Function to display string.
    return 0;
}
```

Both, the above program has same output below:

Output:

Enter name: Tom Hanks
Name: Tom Hanks

String handling functions

C compiler provides a library function called string.h. The functions supported by the header file

string.h are:

int strlen(char* str)	length of the char array
char* strcat(char* dest, char* src)	concatenate dest and src strings return result in dest
char* strcat(char* dest, char* src, int n)	concatenate dest and n characters from src strings return result in dest
char* strcpy(char* dest, char* src)	copy src into dest ; return dest
char* strncpy(char* dest, char* src, int n)	copy n characters from src string into dest ; return dest
char* strcmp(char* str1, char* str2)	compares two strings . returns <0 if str1<str2, returns 0 if str1=str2, returns >0 if str1>str2
char* strcmpi(char* str1, char* str2) or char* stricmp(char* str1, char* str2)	compares two strings . returns <0 if str1<str2, returns 0 if str1=str2, returns >0 if str1>str2 **it ignores case.**
char* strcmp(char* str1, char* str2)	compare first n characters of two strings . returns <0 if str1<str2, returns 0 if str1=str2, returns >0 if str1>str2
char* strchr(char*s, int c)	returns pointer to the first occurrence of c in s; return NULL if not present.
char* strrchr(char*s, int c)	returns pointer to the last occurrence of c in s; return NULL if not present.
char* strstr(char* s, char* pat)	Return pointer to start of pat in s
char* strtok(char* s, char delimiters)	Return a token from s, token is surrounded by delimiters
char* strspn(char* s, char* spanset)	Scan s for characters in spanset , return length of span
char* strcspn(char* s, char* spanset)	Scan s for characters not in spanset , return length of span
strlwr()	converts from upper case to lower case
struper()	converts from upper case to lower case
strrev()	reverses a string
strset(char* s, char c)	It sets all characters in string to character c
strnset(char* s, char c)	It sets first n characters in string to character c

15. Write an algorithm for pattern/string matching.

Simple string matching algorithm using brute force method is given below.

```
Match(text, pat)
{
        L1=strlen(text);
L2=strlen(pat);
for(i=0;i<(L1-L2);i++)
{         j=0;
        while( j! =L2 && text[i+j] ==pat[j])
                  {         j++;
                  }
                  If(j==L2)
                  {         flag=1; break;
                  }
         }
         If(flag==1)
                  printf("pattern found");
         else
                  printf("Not found");
}
```

16. Write a program to search an element using linear and binary search.

Linear Search algorithm/function

```
Search(a, key)
{
         For(i=0;i<n;i++)
         {        if(a[i]==key)
                  {         flag=1; break;
                  }
         }
         If(flag==1)  printf(" Found");
         else      printf("not found");
}
```

Binary search

```
Binsearch(a, key)
{         low=0; high=n-1;
         mid=(low+high)/2
          while(low<high && a[mid]!=key)
         {
                  If(key<a[mid])
                           high=mid-1;
                  else      low=mid+1;
                  mid=(low+high)/2;
         }
         if(a[mid]==key)
                  printf(" found");
         else
```

```
                printf(“Not found”);
}
```

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

I. Creating an Array of N Integer Elements
II. Display of Array Elements with Suitable Headings
III. Inserting an Element (ELEM) at a given valid Position (POS)
IV. Deleting an Element at a given valid Position(POS)
V. Exit.

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

```
*/ #include<stdio.h>
#include<conio.h>
#define SIZE 15
void create(int array[], int n)
{
        int i;
        printf(" Enter n elemnts\n");
        for(i=1;i<=n;i++)
        {
                scanf("%d",&array[i]);
        }
        return ;
}

void display(int array[], int n)
{   int i;
        printf(" Elemnets of the array are\n");
        for(i=1;i<=n;i++)
        {
                printf("%d\t",array[i]);
        }
        printf("\n");
}

int insertAtPos(int array[], int n, int position, int value)
{       int i;
        for (i = n ; i >= position; i--)
                array[i+1] = array[i];
        array[position] = value;
        n++;    return n;
}

int deleteFromPos(int array[], int n, int position)
{       int i, v;
   v=array[position];
        for (i = position; i < n; i++)
                array[i] = array[i+1];
        n--;  return n;
}
int main()
{
        int array[SIZE], n, choice, flag=0;
        int position, value; int count=0;
        char answer;
        while(1)
        {
                printf("1. Create\n");
                printf("2. Display\n");
                printf("3. Inserting Elemnt  at given valid position\n");
                printf("4. Deleting an Element at a given valid Position(POS)\n");
                printf("5. Exit\n");
                printf("Enter choice =");
                scanf("%d", &choice);
                switch(choice)
                {
                        case 1 :    if(flag==0)
                                     {
                                        flag=1;
                                        printf("Enter no. of elements=");
                                        scanf("%d", &n);
                                        // if n is > SIZE. Reduce  n=SIZE
                                        if(n>SIZE)  n=SIZE;
                                                create(array,n);
                                      }
                                     else
                                     {
                                        printf("Array is already created.cannot create again ");
                                     }
                                    break;
                        case 2  :   display(array, n);
                                        break;
                        case 3  :   printf("Enter the location you wish to insert an element=\n");
                                    scanf("%d", &position);
                                    if(position>=SIZE || position>n+1)
                                    {      printf("IT is not valid Position");
                                        break;
                                     }
                                     printf("Enter the value to insert=\n");
                                     scanf("%d", &value);
                                     n=insertAtPos(array, n, position, value);
                                     break;
                        case 4  :    printf("Enter the location you wish to delete an element=\n");
                                     scanf("%d", &position);
                                     if(position>n)
                                     {    printf("Postion is beyond the array element\n"); break;
                                        }
                                        if(n==0)
                                        {
```

```
                                        printf("Array is empty\n"); break;
                                    }
                                    n=deleteFromPos(array, n, position);
                                    break;
                         case 5  :        return 0;
                         default :         printf(" Please enter correct choice");
                                           break;
                    }
               }
               return 0;
  }
```

18. Write a program to sort the numbers using selection and bubble sort algorithms.

Ans:1) SELECTION SORT:

```
#include <stdio.h>
 int main()
{
  int array[100], n, i,j, position, swap;
  printf("Enter number of elements\n");
  scanf("%d", &n);
  printf("Enter %d integers\n", n);
  for ( i = 0 ; i< n ; i++ )
        scanf("%d", &array[i]);

  for ( i = 0 ; i < ( n - 1 ) ; i++ )
  {
            position = i;
            for ( j = i + 1 ; j < n ; j++ )
            {
                        if ( array[position] > array[j] )
                          position = j;
            }
           if ( position != i )
           {
                        swap = array[i];
                        array[i] = array[position];
                        array[position] = swap;
            }
     }
    printf("Sorted list in ascending order:\n");
    for ( i = 0 ; i< n ; i++ )
           printf("%d\n", array[i]);
  return 0;
}
```

2)BUBBLE SORT:

```
 #include <stdio.h>
int main()
{
    int data[100],i,n,step,temp;
    printf("Enter the number of elements to be sorted: ");
    scanf("%d",&n);
    for(i=0;i<n;++i)
    {
       printf("%d. Enter element: ",i+1);
       scanf("%d",&data[i]);
    }

    for(step=0;step<n-1;++step)
    {        for(i=0;i<n-step-1;++i)
             {
                        if(data[i]>data[i+1])   /* To sort in descending order, change > to < in this line. */
                        {
                                   temp=data[i];
                                   data[i]=data[i+1];
                                   data[i+1]=temp;
                      }
             }
     }
     printf("In ascending order: ");
     for(i=0;i<n;++i)
                 printf("%d  ",data[i]);
     return 0;
}
```

Questions on Module I

1. What is Data structure?
2. Explain the classification of data structure
3. What are Data structure operations?
4. Explain declaration, initialization of one dimensional and 2 dimensional Arrays.
5. Discuss the operations that can be performed on the arrays.
6. Describe the representation of multidimensional arrays.
7. Describe the method to calculate the actual memory address of the array location of 1D, 2D.
8. Describe the method to calculate the actual memory address of the array location for 2D, 4D arrays
9. What strings? Explain the operations on strings.
10. What do you mean by pattern matching? Where do we use this concept?
11. Discuss user defined data types: structures and unions.
12. Explain structure within structures.
13. What are pointers? How do we use pointers to different data types?
14. Why do we need dynamic memory allocation techniques? Explain the functions available for allocating memory dynamically.
15. What is sparse Matrix? How to represent Sparse matrix?
16. What is polynomial? How to represent polynomial and add two polynomials?
17. Write an algorithm to transpose sparse matrix.
18. Define string. Explain string handling functions: strcat, strcpy, strcmp, strncmp, strncat, strchr, strrchr, strtok, strstr, strspn, strcspn, strbrk
19. Write an algorithm for pattern/string matching.
20. Develop structure to represent the planets in the solar system. Each planet has fields for planet's name, its distance from the sun(in miles), and the number of moons it has. Place items in each the fields for the planets: Earth and Venus.
21. Write a program to search an element using linear and binary search.
22. Write a program to sort the numbers using selection and bubble sort algorithms.
23. Write a program/algorithm to transpose sparse matrix.
24. Let A be an NxN square matrix array. Write an algorithm/function/program to
    a. Find the number nonzero elements in A
    b. Find the sum of the elements above diagonal (hint I<J)
    c. Find the product of the diagonal elements
25. Suppose multidimensional arrays A & B are declared using A(-2,;2,2;22) and B(1:8,-5:5.-10:5)
    a. Find the length of each dimension and the number of elements in A and B

b. Consider the element B[3,3,3] in B. Find the effective indices E1, E2,E3 and the address of the element assuming Base(B)=400 and there are w=4 words per memory location(size of the data).

Hint: Formula for calculating address
L_i=upper bound-lower bound+1, where L is the number of elements in Column.
For a given subscript K_i, the effective index E_i of Li is the number of indices preceding K_i in the index set and E_i can be calculated from
$E_i=K_i$-lower bound

Then the address LOC(K1, K2, K3…...Kn) of an ordinary element of C array can be obtained from the formula

Base(C)+w[(((…(E_nL_{n-1} +E_{n-1})L_{n-2} + E_{n-2})L_{n-3} + +E_3)L_2 + E_2)L_1 +E_1)

or from the formula
Base(C)+ w[(…((E_1L_2 +E_2)L_3 + E_3)L_4 + ….+E_{n-1})Ln+ E_n)