UNIT 7
ARRAY

LH – 6HRS

PRESENTED BY:

ER. SHARAT MAHARJAN

C PROGRAMMING

CONTENTS (LH – 6HRS)

7.1 Introduction,

7.2 Declaration, Initialization,

7.3 One Dimensional Array,

7.4 Multi Dimensional Array,

7.5 Sorting (Bubble, Selection),

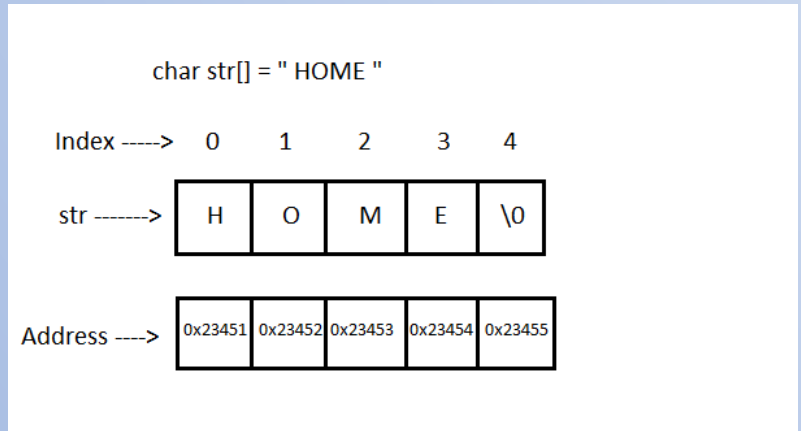
7.6 Searching (Sequential),

7.7 String Handling.

7.1 Introduction

- An array in C is a collection of similar data items stored at contiguous (consecutively) memory locations and elements can be accessed randomly using indices of an array.
- They can be used to store collection of primitive data types such as int, float, double, char, etc of any particular type.
- Using an array, we can store multiple values of type integer with a single identifier without having to declare multiple variables with a different identifier.
- Arrays are useful when we store related data items, such as grades received by the students, marks of students, matrix addition, subtraction etc.
- An array can be single dimensional or multidimensional.

- For example: To declare 30 integers, we simply write: int num[30]
  - This statement tells the compiler that num is an array of type int and can store 30 integers.
  - The individual elements of num are recognized by num[0], num[1],…,num[29].
  - The integer value within square bracket (i.e. []) is called subscript or index of the array.
  - Index of an array always starts from 0 and ends with one less than the size of the array.
- The most important property of an array is that its elements are stored in contiguous memory locations. For example:



Characteristics of Array:

- An array is always stored in consecutive memory location.
- It can store multiple values of similar type, which can be referred with single name.
- The pointer points to the first location of memory block, which is allocated to the array name.
- An array can either be an integer, character, or float data type that can be initialized only during the declaration.
- The particular element of an array can be modified separately without changing the other elements.
- All elements of an array can be distinguished with the help of index number.

7.2 Declaration, Initialization

1. Declaration of 1-D Array

- A list of items can be given one variable name using only one subscript (or dimension or index) and such a variable is called a single-subscripted variable or a one-dimensional array.
- The value of the single subscript or index from 0 to n-1 refers to the individual array elements; where n is the size of the array.
- E.g. the declaration int a[4]; is a 1-D array of integer data type with 4 elements: a[0], a[1], a[2] and a[3].
- Before using an array it must be declared.

- The general form of the single dimensional array is as follows:

Syntax: storage_class data_type array_name[size];

- **storage_class** refers to the storage class of the array. It may be auto, static, extern and register. It is optional.
- **data_type** is the data type of array. It may be int, float, char, etc.
- **array_name** is the name of the array. Any valid name of a variable can be provided.
- **size** of the array is the number of elements in the array and is mentioned within square bracket. The size must be an integer constant like 100 or a symbolic constant (if symbolic constant size is defined as #define SIZE 100, then array can be defined as int a[SIZE];)

Examples:

- int a[10];
- float x[20];
- char name[10];

2. Initializing 1-D Array

- Assigning specific values to the individual array elements at the time of array declaration is known as array initialization.
- Since an array has multiple elements, braces are used to denote the entire array and commas are used to separate the individual values assigned to the individual elements in the array.

Syntax: storage_class data_type array_name[size]={value1, value2,…, valueN};

- E.g. int a[5]={21, 13, 54, 5, 101};
- Here, a is an integer type array which has 5 elements. Their values are initialized to 21, 13, 54, 5 and 101 (i.e. a[0]=21, a[1]=13, a[2]=54, a[3]=5 and a[4]=101). These array elements are stored sequentially in separate memory locations.

- The following array can hold marks for five subjects.

```
int marks[5];                    // array of 5 integers
char color[3]={'R','E','D'};     //array of character
```

- The elements of an array can be initialized in two ways. In the first approach, the value of each element of the array is listed within two curly brackets { } and a comma (,) is used to separate one value from another.

Example:

```
marks[5]={45, 3, 65, 88, 60};
or
marks[]={45, 3, 65, 88, 60};
```

- In the second approach elements of the array can be initialized one at a time.

```
marks[0]=45;
marks[1]=3;
marks[2]=65;
marks[3]=88;
marks[4]=60;
```

7.3 One Dimensional Array

PRACTICAL IN LAB

PREPARED BY: ER. SHARAT MAHARJAN

7.4 Multi Dimensional Array

- They are the arrays with more than one dimension.
- 2-D (two dimensional) array requires two pairs of square brackets; a 3-D array requires three pairs of square brackets and so on.
- The two dimensional array is used to handle the tabular data of the similar type. C language allows arrays of more than two dimensions.
- Syntax: **storage_class data_type array_name[dim1][dim2]…[dimN];**
- Here, dim1, dim2,…,dimN are positive valued integer expressions that indicate the number of array elements associated with each subscript. Thus, total no. of elements=dim1*dim2*…*dimN.

Example:

- **int survey[3][5][12];**

Here, survey is a 3-D array that can contain 3*5*12=180 integer type data. This array survey may represent a survey data of rainfall during past **three years** (2018, 2019, 2020) from **months Jan. to Dec**. in **five cities.**

- Its individual elements are **from survey[0][0][0] to survey[2][4][11].**

Example:

```
int a[3][4];        //two-dimensional array with 3-rows and 4-columns
int b[4][5][3];     //three-dimensional array with dimensions 4, 5 and 3.
```

Two Dimensional Array

- A two-dimensional array is an array of one-dimensional arrays.
- Two dimensional arrays are useful to represent and manipulate tabular data (matrices).

Example: int a[3][4]; //An array of 3 elements(rows), in which every
//element is an array of 4 integers(columns)

Accessing Elements of 2D-array

- The element of i^{th} row and j^{th} column of the 2 dimensional array is accessed with a[i][j];

Initializing two dimensional arrays

- Two dimensional arrays can be initialized rows by rows by separating the each element by commas and each row by { }.

- Let us consider some cases of two dimensional array initialization which are described as follows:

  int a[4][2] = {{12,56},{13,33},{14,80},{15,78}};

  or even this would work with following

  int s[4][2] = {12,56,13,33,14,80,15,78};

- It is important to remember that while initializing a 2-D array, it is necessary to **mention the second (column) dimension**, whereas the **first dimension (row) is optional.**

- Thus the declarations,

  int a[2][3] = {12,34,23,45,56,45};//right

  int a[][3] = {12,34,23,45,56,45}; //right

  are perfectly acceptable and equivalents, whereas the following declaration does not work.

  int a[2][] = {12,34,23,45,56,45}; //wrong

  int a[][] = {12,34,23,45,56,45}; //wrong

- When all the elements are to be initialized to zero, the following short-cut method may be used:

  int marks[3][5] = {{0},{0},{0}};

2-D ARRAY PRACTICAL IN LAB

PREPARED BY: ER. SHARAT MAHARJAN

7.5 Sorting

- Sorting is the process of arranging items in some sequence (ascending or descending) by value or by alphabet or by any other weight.
- Various sorting technique exist but they are beyond the scope of this course (included in Data Structures and Algorithm).
- We will be using a simple sorting technique (selection sort and bubble sort) in the coming slide to sort numbers in ascending order.

1. Selection Sort

```
#include<stdio.h>
#include<conio.h>
int main(){
        int num[100],i,j,n,temp;
        printf("How many numbers you want to sort?");
        scanf("%d",&n);
        for(i=0;i<n;i++){
                scanf("%d",&num[i]);
        }
        printf("The numbers before sorting:\t");
        for(i=0;i<n;i++){
                printf("%d\t",num[i]);
        }
        for(i=0;i<n-1;i++){
                for(j=i+1;j<n;j++){
                        if(num[i]>num[j]){
                                temp=num[i];
                                num[i]=num[j];
                                num[j]=temp;
                        }
                }
        }
        printf("\nThe numbers after sorting:\t");
        for(i=0;i<n;i++){
                printf("%d\t",num[i]);
        }
return 0;
}
```

OUTPUT:

```
How many numbers you want to sort?5
50
30
10
40
20
The numbers before sorting:     50      30      10      40      20
The numbers after sorting:      10      20      30      40      50
--------------------------------
```

- The sort works by passing over each element in our array and comparing it to the first element in the array.
- If the first element is greater than the second element, the two are switched. If the first element is less than or equal to the second, nothing happens. When the sort has looked at every element, one ‘pass’ gets completed. After one pass, one number gets in the correct position. In our ascending order program, the smallest value will ‘bubble’ to the start of the array.
- As we don’t know if the rest of the array is sorted, so we have to take another ‘pass’ for the second element in the array. If the second element is greater than the third element, the two are switched. If the second element is less than or equal to the third element, nothing happens. When the sort has looked at every other element, second ‘pass’ gets completed.
- After second pass, the second number gets in the correct position. The process is continued for several passes to complete.
- The most passes it will require is equal to the number of elements in the array minus 1. Because when (n-1) elements are in their correct positions, the array is itself sorted. So if we have 6 elements in array, it takes 5 passes to complete the sort.

2. Bubble Sort

```
#include<stdio.h>
#include<conio.h>
int main(){
        int num[100],i,j,n,temp;
        printf("How many numbers you want to sort?");
        scanf("%d",&n);
        for(i=0;i<n;i++){
                scanf("%d",&num[i]);
        }
        printf("The numbers before sorting:\t");
        for(i=0;i<n;i++){
                printf("%d\t",num[i]);
        }
        for(i=0;i<n-1;i++){
                for(j=0;j<n-1-i;j++){
                        if(num[j]>num[j+1]){
                                temp=num[j];
                                num[j]=num[j+1];
                                num[j+1]=temp;
                        }
                }
        }
        printf("\nThe numbers after sorting:\t");
        for(i=0;i<n;i++){
                printf("%d\t",num[i]);
        }
        return 0;
}
```

OUTPUT:

```
How many numbers you want to sort?5
30
40
10
20
50
The numbers before sorting:     30      40      10      20      50
The numbers after sorting:      10      20      30      40      50
--------------------------------
```

7.6 Searching

- Searching is the process of finding an element within the list of elements (i.e. array).
- A table or a file is group of elements, each of which is called a record. With each record a key is associated, which is used to differentiate among different records.
- A search algorithm is an algorithm that accepts an argument or a search key and tries to find a record in a given table or list or array.

Sequential Search

- It is a search technique in which each item of the array is examined in turn and compared with the item being searched for until a match occurs. Simply this technique finds a record with a given key by sequentially comparing the key of the records one by one in the table. Scan down it until the desired key is found or end of the list is reached.

Sequential Search Program

```
#include<stdio.h>
#include<conio.h>
#define MAX 5
int main(){
        int num[MAX],i,key,flag=0;
        for(i=0;i<MAX;i++){
                printf("Enter the %d th element of the array :",i+1);
                scanf("%d",&num[i]);
        }
        printf("Enter the key element :");
        scanf("%d",&key);
        for(i=0;i<MAX;i++){
                if(num[i]==key){
                        flag=1;
                        break;
                }
                else{
                flag=0;
                }
        }
        if(flag==1){
                printf("\n%d was found at position %d",key,i+1);
        }
        else{
                printf("\n%d was not found", key);
        }
        return 0;
}
```

OUTPUT 1

```
Enter the 1 th element of the array :10
Enter the 2 th element of the array :20
Enter the 3 th element of the array :30
Enter the 4 th element of the array :40
Enter the 5 th element of the array :50
Enter the key element :20

20 was found at position 2
```

OUTPUT 2

```
Enter the 1 th element of the array :10
Enter the 2 th element of the array :20
Enter the 3 th element of the array :30
Enter the 4 th element of the array :40
Enter the 5 th element of the array :50
Enter the key element :22

22 was not found
--------------------------------
```

7.7 String Handling

- Strings are **array of characters** i.e. they are characters arranged one after another in memory. Thus, a **character array is called string**. Each character within the string is stored within one element of the array successively.
- A string is always terminated by a null character (i.e. slash zero \0).
- Operations performed on string include:
  - Reading and writing strings
  - Copying one string to another
  - Combining strings together
  - Comparing strings for equality
  - Extracting a portion of a string

Declaring String Variables

A string variable is declared as an array of characters.

Syntax: char string_name[size];

- The size determines the number of characters in the string_name.

  E.g. char name[20];
  char city[15];

- When the compiler assigns a character string to a character array, it automatically supplies a null character (‘\0’) at the end of the string. Thus, **the size should be equal to the maximum number of characters in the string plus one**.

Initializing String Variables

Strings are initialized in either of the following two forms:

- char name[4]={‘R’,’A’,’M’,’\0’};
- char name[]={‘R’,’A’,’M’,’\0’};
- char city[9]={‘N’,’E’,’W’,’ ‘,’Y’,’O’,’R’,’K’,’\0’};
- char name[4]=“RAM”;
- char name[]=“RAM”;
- char city[9]=“NEW YORK”;

Reading Strings from Terminal

- The input function scanf can be used with %s format specification to read in a string of characters.

  E.g. char name[20];

  scanf("%s", name);

  No ampersand(&) is required before variable name.

- Some versions of scanf support the following conversion specification for strings:
  - %[characters]
  - %[^characters]
- The specification %[characters] means that only the characters specified within the brackets are allowed in the input of string. If the input string contains any other characters, the reading of string will be terminated at the first encounter of such a character.
- The specification %[^characters] means that the characters specified after the caret(^) are not allowed in the string and reading will be terminated.

String Manipulation using Library Functions

- The <string.h> header file defines some string handling functions such as strcpy(), strcat(), strrev(), strcmp(), strlen() etc are used to manipulate the string data.
- These functions are used for:
  - Manipulating string data
  - Determining the length of strings
  - Copying a string
  - Concatenation of two strings
  - Changing to upper and lower case
- We call these functions from string.h header file.

1. **Copying string**

The string function strcpy() is used to copy one string into another string. There is another function strncpy() is used to copy n characters to strings.

2. **Finding the Length of String**

The function strlen() returns an integer which denotes the length of the string passed into the function. The length of the string is defined as the number of characters present in the string, excluding the null character.

Syntax: integer_variable=strlen(input_string);

3. **Concatenating two Strings**

The string function strcat() is used to concatenate two strings.

4. **Reversing the String**

The function strrev() is used to reserve the given string.

5. Converting a String to Uppercase

The string function strupr() is used to convert the given string into uppercase.

6. Converting a String to Lowercase

The string function strlwr() is used to convert the given string into lowercase.

7. String Comparison

The function strcmp(s1,s2) compares the string s1 to the string s2. The function returns 0, less than 0 or greater than 0 if s1 is equal to (identical), less than (alphabetically less than) or alphabetically greater than s2, respectively.

THANK YOU FOR YOUR ATTENTION