Birla Institute of Technology & Science, Pilani
Computer Programming (CSF111)
Second Semester 2014-2015
Lab Week 7

Objectives

1. Arrays (random access lists) : One Dimensional and Multidimensional
2. Searching: Linear Search and Binary Search
3. Sorting: Selection and Bubble Sort
4. Exercises

Now that you are aware with the conditional (*if - else*) and iterative (*for - while - do while loops*) constructs, today you will understand the concept of Arrays.

Arrays

Think about writing a program to find the average of 3 integers. You can declare three integers, get there values from the user (using *scanf ()*), get the average, and print it. Now think about finding the average of 1000 integers. Will you declare 1000 integer variables in your program and will continue the same way as above? You can, but ***is it feasible****??*

Now think on the same problem again. You actually need 1000 integer variables, but as a programmer we need some way out to declare, access, and process these many large numbers of variables with ease. C (and in fact many other programming languages) provides a way out for it, **i.e. ARRAYS**.

An array in C Programming Language can be defined as number of memory locations, each of which can store the same data type and which can be referenced through the same variable name. In other words, it is a collective name given to a group of similar quantities. These similar quantities could be percentage marks of 100 students, number of chairs in home, or salaries of 300 employees or ages of 25 students. Thus an array is a collection of similar elements. These similar elements could be all integers or all floats or all characters etc. Usually, the array of characters is called a "string", where as an array of integers or floats are called simply an array. [*We will specially deal with strings in another session as it requires special attention.*]All elements of any given array must be of the same type, i.e. we can't have an array of 10 numbers, of which 5 are int's and 5 are float's.

Declaration of One Dimensional Array

One Dimensional Arrays (Linear Array) must be declared before they can be used in the program. Standard array declaration is as:

```
type variable_name[lengthofarray];
```

Here type specifies the variable type of the element which is going to be stored in the array. For example:

```
double height[10];
float width[20];
int min[9];
```

In C Language, arrays start at position 0. So, **`int min [9]`** :

- declare 9 variables of type integer,
- All the 9 variables occupy adjacent locations in memory.
- Any item in the array can be accessed through its index, and it can be accessed any where from with in the program. Also index starts from 0. This means the first variable in min[0], second is min[1], third is min[2], ... and ninth is min[8]. Notice that now the elements of arrays can be processed by controlling the index with in a loop. In fact, to understand this last point, see the following example.

Example 1: Program to input the marks of 10 students and print it. Observe closely the two programs (ver1.c and ver2.c) for the same and understand clearly how the array variables can be controlled and accessed with in loops.

```
//     ver1.c
int main (void)
{
 int marks[10];
 scanf ("%d",&marks[0]);
 scanf ("%d",&marks[1]);
 scanf ("%d",&marks[2]);
 .
 .
 scanf ("%d",&marks[9];
 printf ("Marks of student 1 are \n",marks[0]);
 printf ("Marks of student 2 are \n",marks[1]);
 printf ("Marks of student 3 are \n",marks[2]);
 .
 .
 printf ("Marks of student 9 are \n",marks[8]);
}
```

```
//     ver2.c
int main (void)
{
 int marks[10];
 int i;
 for (i=0;i<10;i++)
  scanf ("%d",&marks[i]);
 for (i=0;i<10;i++)
  printf ("Marks of student %d are \n",(i+1), marks[i]);
 }
```

Example 2: Now that you have understood how to control the array variables, let us modify the same program (ver2.c) to find the average of marks of 100 students.

```
//     ver3.c
int main (void)
{
 int marks[100], sum = 0, i;
 float average;
 for (i=0;i<100;i++)
  scanf ("%d",&marks[i]);
 for (i=0;i<100;i++)
  sum += marks[i];
 average = sum / 100;
printf ("Average = %f",average);
 }
```

- Observe that the program (ver3.c) will not produce the correct result. Though it is syntactically correct, to produce the correct result, explicit type casting is to be done somewhere. Find it and correct it.
- Also observe that giving 100 inputs again and again is very difficult. This is especially true when you are writing large programs and you want to test it many times. So to overcome the difficulty of giving 100 inputs again and again, you can store all the 100 integers in a file (suppose input.txt) and while executing your program use indirection operator in unix, i.e.

  $ gcc ver3.c

  $ ./a.out < input.txt

  If this point is not clear, ask your lab instructor.

Task 1: Complete the following program that computes some statistics on a list of values:

```
#include<stdio.h>
int main(void)
{
 int i,n,a[25];
 float avg;
 printf("Enter the number of elements: \n");
 scanf("%d",&n);
 printf("Enter %d values: \n", n);
 for(i=0;i<n;i++)
 {
  scanf("%d",&a[i]);
 }

 /*Find the largest and second largest and third largest element*/

 /* Print the largest value, second largest value and third largest */
 return(0);
}
```

Task 2: Write a C program to copy elements of an array A into another array B so that following sequences should be followed:
1. All even elements of A from left to right are copied into C from left to right.
2. All odd elements of A from left to right are copied into C from right to left.

Declaration of a Two Dimensional Array

Standard Two Dimensional Array declaration is as:
type variable_name[NoOfRows][NoOfColumns];

Here type specifies the variable type of the element which is going to be stored in the array. For example if you want to stores marks of 5 students for 10 subjects each then define matrix (Two Dimensional Array) of 5 rows and 10 columns. Like below
double marks[5][10];
This matrix is composed of 5 One Dimensional Arrays of 10 elements each stored in consecutive memory locations. Row Number and Column Number will start from indices 0. i.e in this example row numbers are 0,1,2,3,4 and Column Numbers are 0,1,2,3,4,5,6,7,8,9.

Example 3: A program to input marks of 5 students in 10 subjects each and display the marks.

```
//     2DArray
int main (void)
{
 int marks[5][10];
 int I,j;
 for (i=0;i<5;i++)
printf("Enter the marks of student %d",i+1);
  for(j=0;j<10;j++)
     scanf ("%d",&marks[i][j]);
for (i=0;i<5;i++)
printf("Marks of student %d",i+1);
  for(j=0;j<10;j++)
     printf(" Subject %d %d",j+1,marks[i][j]);

}
```

Task 3: Write a program to input marks of 5 students in 10 subjects each and display what are the average marks of each student in 10 subjects and display what is the average of each subject.

Sorting: The aim of sorting algorithms is to put unordered information in an ordered form.

Selection Sort Algorithm:

1. The list is divided into two sublists, sorted and unsorted, which are divided by an imaginary wall like any index can partition the complete list in two parts.
2. We find the smallest element from the unsorted sublist and swap it with the element at the beginning of the unsorted data.
3. After each selection and swapping, the imaginary wall / partition index between the two sublists move one element ahead, increasing the number of sorted elements and decreasing the number of unsorted ones.
4. Each time we move one element from the unsorted sublist to the sorted sublist, we say that we have completed a sort pass.
5. A list of n elements requires n-1 passes to completely rearrange the data.

Task 4: Write a program to sort the elements of Array using Selection Sort.

Exercises:

1. Write a program which insert an element in already existing sorted array so that array will remain in sorted order after insertion.
2. Write a program to delete duplicates from an array.
3. Write a program to perform addition, subtraction and multiplication operations on two matrices.
4. Write a program to find transpose and determinant of a matrix.
5. Write a program to sort an array using Bubble Sort.

6. Modify binary search for a 2D array as follows: first find the row in which the element can be there (using linear search on the first column); and then find the element in that row (using binary search). Assume here that matrix is strictly sorted, i.e. elements are entered in increasing order row wise such that the first element of each row is greater than the last element of the previous row.