Programming in C

CHAPTER 6:

Arrays

In many of the applications it is required to store a set of similar data and manipulate them. Under such circumstances arrays are used. Arrays store elements of similar type which can be accessed by means of a number called index. Consider the following expression:

int arr[10];

this expression denotes an integer array by name 'arr' with ten elements. Each element of the array will be of type integer. Here 10 is the index or subscript denoting the size of the array.

Note: In C, indexing starts from 0, so an array arr[10] will have ten elements namely arr[0], arr[1], ... arr[9].

Just like other variables, arrays must also be declared before being used. The declaration includes the type of the element stored in the array (int, float or char), and the maximum number of elements that we will store in the array. The C compiler needs this to determine how much of memory space to reserve for the array.

The elements of an array are manipulated in the same manner as any other ordinary variable (i.e. they can be added, subtracted or compareed).

Program 6.1

```
#include <stdio.h>

main ()
{
  int arr[4];
  int index;
  arr[0] = 100;
  arr[1] = 200;
  arr[2] = 300;
  arr[3] = arr[0] + arr[1];
  for(index = 0; index < 4; ++index)
    printf("arr[%d] = %d\n", index, arr[index]);
}
```

Let us consider a more practical program like generating a Fibonacci series. A well-known series with a number of applications in the fields of mathematics and computer science between others.

Program 6.2

```
/* Program to generate first 10 Fibonacci numbers using arrays */
```

```
#include <stdio.h>

main ()
{
  int fib[10], ind;
  fib[0] = 0;
  fib[1] = 1;
  for (ind = 2; ind < 10; ++ind)
    fib[ind] = fib[ind-2] + fib[ind-1];
  for (ind = 0; ind < 10; ++ind)
    printf("%d\n", fib[ind]);
}
```

Initializing an array

Initial values can be assigned to the elements of an array at the time of declaration. The keyword static is used to declare variable data with a value which persists for the whole life of the running process. In C, this is usually used to store constant variables. This requires the usage of the keyword static.

static int arr[4] = {100,200,300,400};

You can partially initialize an array. The remaining elements of an array are set to zero. The following statement will initialize the first elements of an array,

static int arr[4] = {10, 20, 30};

Note: C does not provide any shorter method to initialize an array. All the elements must be explicitly initialized.

Program 6.3

```
/* Array initialization */

/* Program to calculate cube of a number */

#include <stdio.h>

main ()
{
  int cbe[20];
  int i;
  for (i = 1; i < 20; i++)
    cbe[i] = i * i * i;
  for(i = 1; i < 20; i++)
    printf("%d\t %d\n", i, cbe[i]);
}
```

Passing an array

You can pass an entire array from one function to another. C handles a passed array differently from variables. When a variable is passed, C makes a copy of the data and places it in a memory location associated with the receiving variable. This is known as pass by value. There are two copies of data existing, changing the variable in the called function does not change the original variable in the calling function.

When passing an array to a function, you actually pass the address of the array. This is known as pass by reference. So a copy of the data is not made, C assigns the same address area to the second array name that is used in the called function. Since you are passing the address of an array, you need not specify the size of the array. This kind of array passing saves memory, but changing the value of an element of the array in the called function, also changes the value of the corresponding element of the array in the calling function.

Strings

A string is an array of characters. The last element in a string is the NULL terminator ('\0'). So, a string is an array with one element more than the maximum number of actual characters.

Program 6.4

```
#include <stdio.h>

main ()
{
  char name[39+1];
  int index;
  printf("Enter your name :");
  scanf("%s", name);
  for(index = 0; index<40; index++)
    printf("Hello %s", name);
}
```

We can copy a string str1 into another str2 like this:

Program 6.5

```
#include <stdio.h>

main ()
{
  char str1[19+1], str2[19+1];
  int i;
  printf("Enter a string \n");
  scanf("%s", str1);
  i = 0;
  while( (str2[i]=str1[i]) != '\0' )
    i++;
  printf("str2 contains \t%s", str2);
}
```

One important point to remember when using character arrays or strings in a program is to add a '\0' or NULL terminator at the end of a string. If we want to print the line using printf(), it is necessary.

Program 6.6

```
#include <stdio.h>

main ()
{
  int n;
  char str[49+1];
    n = 0;
    while( (str[n++]=getchar( )) != '\n' )
      str[n] = '\0';
      /* append NULL terminator to the end of the string */
    printf("%d:\t%s", n, str);
}
```

C compilers include some special functions for working with strings. Built-in functions like strcmp(), strcpy(), strcat(), strrev() found in C standard library come in handy when manipulating strings.

Program 6.7

```
#include <stdio.h>

main ()
{
  char name1[39+1];
  char name2[39+1];
  int index;
    printf("Enter a name :");
    scanf("%s", name1);
    printf("Enter a second name :");
    scanf("%s", name2);
    if(strcmp(name1,name2) == 0)
      printf("The names are the same \n");
    else
      printf("The names are not the same\n");
}
```

String arrays

String arrays are multidimensional arrays, i.e. an array of strings, or an array of arrays of characters. Consider the following expression:

char names[10][39+1];

The above variable is an array of strings with ten names and upto 39 characters in each (plus one space for the string delimitator '\0')

C allows arrays of any dimension to be defined, the first index of the array refers to the row number, and the second index number refers the column number. Two dimensional arrays are initialized in the same manner as a one dimensional array.

Example:
static int matrix[4][5] = {
{10, 12, 18, 33, 12},
{11, 45, 66, 76, 32},
{12, 34, 43, 56, 67},
{35, 45, 65, 67, 32}
};

Searching and Sorting

Searching and sorting through arrays is one of the most labor-intensive tasks. There are two different approaches to searching through arrays: linear or sequential search, and binary search.
In a linear search, each element of the array is checked until a match is found. Below is an example of a function that searches an array for a specific item, and returns its location if the item is found or returns -1 if it was not found.

```
int search (int aray[], int size, int item)
{
  int cnt = 0;
    while (cnt < size)
    {
    if (item ==  array[cnt])
      return (cnt);
    cnt++;
    }
    return (-1);
}
```

In a binary search, the data in the array is ordered and then the middle of the contracted array is tested until the required match is found. Next is an example of a binary search:

```
Assume that the value passed to this function
are:
    int array = [50,8,20,3,40];
    int item = 40;
    int size = 5;


int BinSearch (int array[], int size, int item)
{
  int low = 0, count = 0;
  int high = size - 1;
  int middle;
    while (low <= high)
    {
      count++;
      middle = (high + low) / 2;
      if (item == array[middle])
        return middle;
      else if (item < array[middle])
        high = middle - 1;
      else
        low = middle + 1;
    }
  return -1;
}
```

```
What is the value of count when searching for:
a)      50              R: 3
b)      8               R: 2
c)      20              R: 1
d)      3               R: 2
e)      40              R: 3
```

Sorting data

There are three approaches to sorting arrays: selection sort, insertion sort, and bubble sort. As you will notice, whereas searching involves a single for loop and visiting each array location, sorting involves nested for loops, and n-1 passes through the array.

In a **selection sort**, we start with the first position in the array, find the smallest value in a first pass through the array, and swap the value in the first position in the array with the value in the position of the newly found smallest value; then we take each subsequent position in the array, find the smallest value in the remaining array, and swap position. Here is an example of a selection sort:

```
void selectionSort (int a[], int size)
{
  int temp, pass, k, min, minIndex;
    for (pass = 0; pass < size - 1; pass++)
    {
      min = a[pass];
      minIndex = pass;
      for (k = pass + 1; k < size; k++)
      {
        if (a[k] < min)
        {
          min = a[k];
          minIndex = k;
        }
      }
      temp = a[pass];
      a[pass] = a[minIndex];
      a[minIndex] = temp;
    }
    return;
}
```

In an **insertion sort**, you start with one item, take a new item and sort the two items relative to each other, then take a new item and sort the three items relative to each other (swapping the new item with consecutive values until it is no longer lower, and thus inserting it in that position), and so on. It is like sorting a deck of cards with your hands. Below is an example:

```
void insertionSort (int a[], int size)
{
  int temp, pass, k;
    for (pass = 1; pass <= size - 1; pass++)
    {
      for ( k = pass; k >= 1; k--)
      {
        if (a[k] < a[k-1])
        {
          temp = a[k];
          a[k] = a[k-1];
          a[k-1] = temp;
        }
      }
    }
    return;
}
```

In a **bubble sort**, you swap neighbors; the larger items drop down while the smaller ones bubble up, in n-1 passes through the array.Here's an example:

```
void bubbleSort (int a[], int size)
{
int temp, k, pass;
  for (pass = 1; pass <= size - 1; pass++)
  {
    for (k = 0; k < size - pass ; k++)
    {
      if (a[k] > a[k+1])
      {
        temp = a[k];
        a[k] = a[k+1];
        a[k+1] = temp;
      }
    }
  }
  return;
}
```

Review questions

Assignments

Quiz

Submit assignment

[Your opinion is important to us. If you have a comment, correction or question pertaining to this chapter please send it tocomments@peoi.org .]