Introduction to Computing

MCS1101B

Lecture 6

By
Soumadip Biswas
Associate Professor, IEM

Array

- Many applications require multiple data items that have common characteristics
    - In mathematics, we often express such groups of data items in indexed form:
        - x1, x2, x3, …, xn
- Array is a data structure which can represent a collection of data items which have the same data type (float/int/char/…)

Example:

```
int n, A[100], i;
printf(“How many numbers to read? “);
scanf(“%d”, &n);

for (i = 0; i < n; ++i)
        scanf(“%d”, &A[i]);
for (i = 0; i < n; ++i)
        printf(“%d”, &A[i]);
```

Array(contd.)

- Declaration
  - <type> <name>[<no_of_elements>]
  - int a[100];
  - Float b[20];
- Initialization
  - int a[5] = {2,4,5,2,6};
  - int b[4] = {1,3,5}
- Accessing an element of array
  - a[2] →5
  - b[0] →1
  - b[3] → ?
  - a[5] →?

- Assignment of value later on in the program
  - It is same as a normal variable
  - b[3] = 3.14;
  - a[2] = 1000;
- A single variable has a name
- An array variable has a <name>
  - It’s a collection of single variables
  - Variables are accessed using <index>
  - Therefore, <name>[<index>] is a specific variable in an array

Array (contd)

- Some Basic Examples
  - Print all elements of an array
  - Scan elements into an array
  - Copy elements of on array into another
  - Sum of all elements in an array

- Some more examples
  - Find minimum of a set of 10 numbers
  - Write the code in a way so that the code works for a set of any given number (i.e. not only 10)

Array (contd)

Write the code in a way so that the code works for a set of any given number (i.e. not only 10)

- Recall **const** qualifier
    - const int size = 10;
- Another way …
    - #define SIZE 10
    - This is called a preprocessor/macro

Searching for an Element (key) in an Array

- You have an array full of integer elements
    - Can be hard coded
    - Can be user input
    - Can be redirected (using <) from some file <we learn this today>
    - Can be read from file <we will see how later on>
- You take an integer (*key*) user input from user
- Search through the array to check if the *key* exists in the array
    - Go through the array one element at a time in using a loop
    - Check is the element matches the *key* or not
- Print appropriate message to show the result of the exercise
- This is called a linear search

Functions (recall)

Passing of variables

- Variables values are copied when then are passed (by calling) to a function
- The actual variables are not passed
- So a change made to a variable within a function will not reflect in the variable at the end of the caller

- But scanf, which is a function, is able to change the values of a local variable
    - How does it do it?
- Recall the AddressOf (&) operator
    - scanf (“%d”, **&a**);
    - it sends (copies) the memory address of a variable
    - scanf makes change to that memory location
    - thereby changing the value of the variable

Pointers

- Pointers are a special variables that can store memory locations
- Declaration of a pointer variable
    - <type> *<name>;
    - Variable value can be accessed using <name>
- Access the value at the stored address
    - *<name>
    - It will treat the value at the stored location as the declared <type>

- int a=10; int *ptr; //this is an integer type pointer
- printf (“%d”, a); ⇒ 10
- printf (“%d”, ptr); ⇒ <some garbage value>
- printf (“%p”, ptr); ⇒ <the same garbage value in the form of an memory address>
- printf (“%p”, &ptr); ⇒ the address of the variable ptr
- printf (“%p”, &a); ⇒ memory address of the variable a
- ptr = &a; //stores the address of a on ptr
- printf (“%p”, ptr); ⇒ the address of the variable a
- printf (“%d”, *ptr); ⇒ value of the integer at the location of the variable a
- printf (“%p”, &ptr); ⇒ the address of the variable ptr; remains the same

Array and Pointers

- Array elements are accessed using indexes
  - int arr[10];
    - Allocates a memory block equal to the size of 10 integers in total
    - Elements accessed as arr[0], arr[1], etc.
  - The **arr** is the address of the entire memory block; it is of type int* (read as *integer pointer*)
  - Therefore It can also be accessed similar to pointers variables
  - So *arr is arr[0]
    - How do you access the rest? → you use pointer arithmatic
  - Adding 1 to a pointer variable means increasing the value of the pointer by the size of the type of that pointer
    - adding 1 to an int* **variable** means adding sizeof(int) to the value of the **variable**
  - So, arr[1] == *(arr+1), arr[2] == *(arr+2), etc., i.e., arr[i] = *(arr+i)
  - Also, arr+i = &arr[i]

Functions and Pointers

- Since variables passed to the functions are basically a copy
- Pointers to the variables are used instead of a variable to pass the **reference** to a variable - only when required
  - Addresses of the variable is copied
  - Changes made by function are done to the memory address
  - So when function exits, it only forgets the memory location and not the changes made ot that location

So, Let’s recall Swap

```
void swap (int a, int b)
{
        int tmp;
        tmp = a;
        a = b;
        b = tmp;
}
```

```
void swap (int *a, int *b)
{
        int tmp;
        tmp = *a;
        *a = *b;
        *b = tmp;
}
```

Functions Calling Functions

- int f1() {...}

-
  ```
  int f2()
  {...
          f1();
  …}
  ```
- int f3() {... f2(); …}

- int f4() {... f3(); …}
- int f5() {... f2(); …}

- int f6() {... f7(); …}
- int f7() {... f6(); …}

- int f8() {... f8(); …}

- These are basically never ending calls to one another
  →can this happen?

Recursion

- A function calling itself
    - Directly call made to self
    - Indirectly call made to self via another function
    - Indirectly call made to self via a sequence of function calls
- This is known as recursion
    - Both in mathematics and in programming

- Example (math)
    - $f(n) = n*f(n-1), f(0)=1$
    - $f(n) = f(n-1) + f(n-2), f(0)=0, f(1)=1$
      → what function is this?
    - $f(x) = x * g(x), g(x) = 2 + f(x-1)$

      $\Rightarrow f(x) = 2 * x + 2 * f(x-1)$

Recursion (contd.)

- Requires careful coding
- Needs to make sure that your program terminates
- You need to first define the base cases (exit condition) for your function
- Then you write the logic of the rest of the function
- For breaking the call sequence of a recursive function
    - a **return** statement is generally used with some if condition
    - You can also use if-else
- Exercise:
    - Implement the factorial function using recursion
    - Implement the gcd function using recursion

In The Next Class…

- You will learn about array and pointers
- You will learn about structures
- You will learn about files