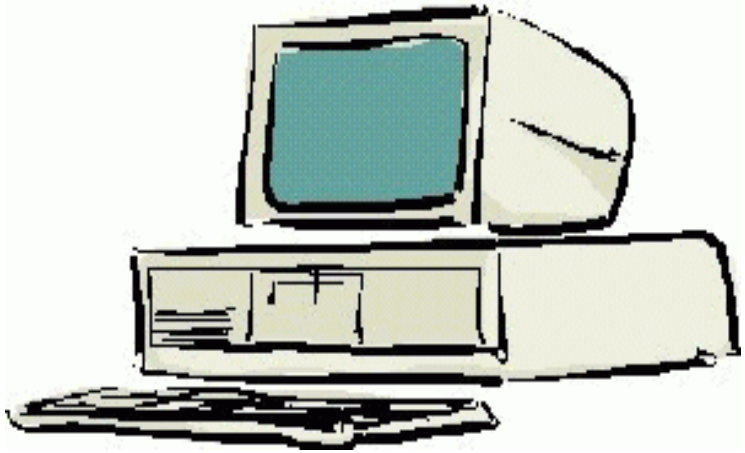
Programming in C



Session 6

Seema Sirpal
Delhi University Computer Centre

Arrays

Arrays are a data structure that is used to store a group of objects of the same type sequentially in memory.

All the elements of an array must be the same data type, for example float, char, int .

The elements of an array are stored sequentially in memory. This allows convenient and powerful manipulation of array elements using pointers.

Defining Arrays

An array is defined with this syntax.

datatype arrayName[size];

Examples:

```
int ID[30];
        /* Could be used to store the ID numbers of students in a class */
float temperatures[31];
        /* Could be used to store the daily temperatures in a month */
char name[20];
        /* Could be used to store a character string.
        Character strings in C are terminated by the null character, '\0'.
        This will be discussed later in the this lesson. */
unsigned short int[52];
        /* Holds 52 unsigned short integer values */
```

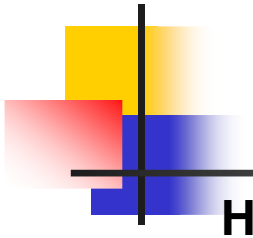
Using Arrays

Arrays in C are zero based. Suppose an array named myExample contains N elements. This array is indexed from 0 to (N-1). The first element of myExample is at index 0 and is accessed as myExample[0]. The second element is at index 1 and is accessed as myExample[1]. The last element is at index (N-1) and is accessed as myExample[N-1]. As a concrete example, suppose N equals 5 and that myExample will store integer data.

```
int myExample[5];     /* Defines myExample to be of length 5 and to
contain integer data */

myExample[0]     /* First element of myExample */
myExample[1]     /* Second element of myExample */
myExample[2]     /* Third element of myExample */
myExample[3]     /* Fourth element of myExample */
myExample[4]     /* Fifth and final element of myExample */
```

Sample Program



Here is a sample program that calculates and stores the squares of the first one hundred positive integers.

```
#include <stdio.h>
int main()
{
    int square[100];
    int i;    /* loop index */;
    int k;    /* the integer */
    /* Calculate the squares */
    for (i = 0; i < 100; i++) {
        k = i + 1;    /* i runs from 0 to 99 */
                      /* k runs from 1 to 100 */
        square[i] = k*k;
        printf("The square of %d is %d\n",k,square[i]);
    }
    return 0;
}
```

Practice Problem

A Fibonacci sequence is a numerical sequence such that after the second element all numbers are equal to the sum of the previous two elements.

1, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89, ...

Write a program to calculate and print the first 20 elements of this sequence.

Solution

```
#include <stdio.h>
int main()
{
    int i;
    int fib[20];
    fib[0] = 1;
    fib[1] = 1;
    for (i = 2; i < 20; i++)
    {
        fib[i] = fib[i-1] + fib[i-2];
    }
    for (i = 0; i < 20; i++)
    {
        printf("The %d element is %d\n", i+1, fib[i]);
    }
return 0;
}
```

Multidimensional Arrays

The C language also allows multidimensional arrays. They are defined as follows.

```
float temperatures[12][31];
                    /* Used to store temperature data for a year */

float altitude[100][100];
              /* Used to store the altitudes of 10000 grid points
                   of a 100 by 100 mile square */
```

Multidimensional Arrays

A common way to access the elements of a multidimensional arrays is with nested for loops.

```
#define MAXI 50
#define MAXJ 75

int i;
int j;
float values[MAXI][MAXJ];

for (i = 0; i < MAXI; i++) {
    for (j = 0; j < MAXJ; j++) {
        values[i][j] = whatever;
    }
}
```

Important Note about Array Dimensions

The C language performs no error checking on array bounds.

If you define an array with 50 elements and you attempt to access element 50 (the 51st element), or any out of bounds index, the compiler issues no warnings.

It is the programmer's task alone to check that all attempts to access or write to arrays are done only at valid array indexes.

Writing or reading past the end of arrays is a common programming bug and can be hard to isolate.

Important Note about Array Dimensions

What will happen if a program accesses past the end of an array? Suppose a program has the following code.

```
int val;

int buffer[10];

val = buffer[10];
   /* Bug, remember that the indexes of buffer run from 0 to 9. */
```

What value will be in val? Whatever happens to be in memory at the location right after the end of the array. This value could be anything. Worse yet, the program may continue to run with the incorrect value and no warnings are issued.

Important Note about Array Dimensions

What will happen if a program writes past the end of an array? Suppose a program has the following code.

```
int buffer[10];

buffer[593] = 99;
```

The value of 99 will be written at the memory location, buffer + 593. "buffer" is a pointer to the beginning of the array. buffer + 593 is pointer arithmetic for the address equal to the starting address of the array plus the size of 593 integers.

The overwriting of the value at this memory location will change the value of whatever variable is stored there. Some other variable may have its value changed unintentionally. If the program writes unintentionally to memory locations that are not valid, the program may crash.

Important Note about Array Dimensions

The most common cause of writing/reading to invalid array indexes are errors in loop limits.

```
int i;
float b[10];

for (i < 0 ; i <= 10; i++) {
    b[i] = 3.14 * i * i;
}
```

This loop should use "<" rather than "<="