Data Structure Using C - Arrays

Dr. Nachiket Tapas

Introduction

- An array is a structured collecction of components, all of same type, that is given a single name.
- Each component (array element) is accessed by an index that indicates the component’s position within the collection.

Declaration

- Like any other variable, an array must be defined before it can be used to store information.
- Like other variable declarations, an array declaration specifies a variable type and a name. But it includes another feature i.e. size.

DataType ArrayName [Constant Integer Expression];

Array Elements

- The items in an array are called elements.
- All elements in an array are of the same type; only the values vary
- Like:

```
int array1[4] = { 10, 5, 678, -400 };
```

Accessing Array Elements

- To access an individual array component, we write the array name, followed by an expression enclosed in square brackets.
- The expression specifies which component to access.

ArrayName [IndexExpression]

Like

array1[2]
array1[i] where i = 3

Initializing array in Declaration

- To initialize an array, you have to specify a list of initial values for the array elements, separate them with commas and enclose the list within breaces.

```
int array1[5] = { 23, 10, 16, 37, 12 };
```

- We don’t need to use the array size when we initialize all the array elements, since the compiler can figure it out by counting the initializing variables.

```
int array1[] = { 23, 10, 16, 37 };
```

What if?

- What happens if you do use an explicit array size, but it doesn’t agree with the number of components?
- if there are too few components/items, the missing element will be set to zero.
- if there are too many, an error is signaled.

Lack of Aggregate Array Operations

- C does not allow aggregate operations on arrays. Meaning:

```
int x[50], y[50];
x = y; //This will generate an error.
```

- Instead, you need to do it element by element.

```
for ( i = 0; i < 50; i++ )
        x[i] = y[i];
```

- Similarly, comparison to two array is not possible as follows:

```
if ( x == y)
```

Other things which are not possible.

- Aggregate input:

  scanf(“%d”, x);

  Exception?

- Aggregate arithmetic operations:

  x = x + y;

- Return an entire array from a function:

  return x;

Example

```
void main() {
        double sales[6], average, total = 0;
        for ( int i = 0; i < 6; i++ )
                scanf(“%lf”, &sales[i]);
        for ( int i = 0; i < 6; i++ )
                total += sales[i];
        average = total / 6;
        printf(“Average = %lf”, average);
}
```

Multidimensional Arrays

- A two dimensional array is used to represent items in a table with rows and columns, provided each item in the table is of same data type.
- Each component is accessed by a pair of indexes that represent the component's position in each dimension.

- Two Dimensional Array:
- The array is defined with two size specifiers, each enclosed in brackets.

  DataType ArrayName [Constant Integer Expression]
  [Constant Integer Expression]
- Example

  double array2[3][4];

Accessing Multidimensional Array

- Array elements in two dimensional arrays required two indexes

array2[1][2]

Example

```
void main() {
        float array2[3][3] = {{ 12.2, 11.0, 9.6 },
                              { 23.9, -50.6, 2.3 },
                              { 2.2, 3.3, 4.4 }};

        for ( int row = 0; row < 3; row++ )
                for ( int col = 0; col < 3; col++ )
                        printf(“%f”, array2[row][col]);
}
```

Matrix Multiplication

```
matrixMultiply(A, B):
Assume dimension of A is (m x n), dimension of B is (p x q)
Begin
   if n is not same as p, then exit
   otherwise define C matrix as (m x q)
   for i in range 0 to m - 1, do
      for j in range 0 to q - 1, do
         for k in range 0 to p - 1, do
            C[i, j] = C[i, j] + (A[i, k] * A[k, j])
         done
      done
   done
End
```

Program

```
#include<stdio.h>
void readMatrix(int a[][3]) {
        for(int i=0; i<3; i++)
                for(int j=0; j<3; j++)
                        scanf("%d", &a[i][j]);
}
void printMatrix(int a[][3]) {
        for(int i=0; i<3; i++) {
                for(int j=0; j<3; j++)
                        printf("%d\t", a[i][j]);
                printf("\n");
        }
}
```

Contd.

```
void multiplyMatrix(int a[][3], int b[][3], int c[][3]) {
        for(int i=0; i<3; i++)
                for(int j=0; j<3; j++) {
                        c[i][j] = 0;
                        for(int k=0; k<3; k++)
                                c[i][j] += a[i][k] * b[k][j];
                }
}
```

Contd.

```
void main() {
        int mat1[][3] = {{1, 2, 3}, {1, 2, 3}, {1, 2, 3}};
        int mat2[][3] = {{1, 2, 3}, {1, 2, 3}, {1, 2, 3}};
        int mat3[][3] = {{}, {}, {}};
        printf("\nEnter Matrix A:\n")
        readMatrix(mat1);
        printf("\nMatrix A:\n")
        printMatrix(mat1);
        printf("\nEnter Matrix B:\n")
        readMatrix(mat2);
        printf("\nMatrix B:\n")
        printMatrix(mat2);
        multiplyMatrix(mat1, mat2, mat3);
        printf("\nMatrix C:\n")
        printMatrix(mat3);
}
```

String: What is it?

- In C programming, a string is a sequence of characters terminated with a null character \0. For example:

```
char c[] = "c string";
```

- When the compiler encounters a sequence of characters enclosed in the double quotation marks, it appends a null character \0 at the end by default.

Declaration and Initialization

- Declaration

```
char s[5];
```

- Initialization

```
char c[] = "abcd";

char c[50] = "abcd";

char c[] = {'a', 'b', 'c', 'd', '\0'};

char c[5] = {'a', 'b', 'c', 'd', '\0'};
```

Assigning Values

- Arrays and strings are second-class citizens in C; they do not support the assignment operator once it is declared. For example,

```
char c[100];
c = "C programming";  // Error! array type is not assignable.
```

Reading Input

- scanf()
  - You can use the scanf() function to read a string.
  - The scanf() function reads the sequence of characters until it encounters whitespace (space, newline, tab, etc.).

- gets()
  - The gets() function can also be to take input from the user.
  - However, it is removed from the C standard. It's because gets() allows you to input any length of characters. Hence, there might be a buffer overflow.

- fgets()
  - fgets(variable, sizeof(variable), stdin);

Print functions

- printf()
  - You can use the printf() function to print a string.

- puts()
  - The puts() function can also be used to print a string.

printf vs puts

- puts() can be preferred for printing a string because it is generally less expensive (implementation of puts() is generally simpler than printf()).
- If the string has formatting characters like ‘%s’, then printf() would give unexpected results.

Thank You!!