Lecture 13.1

Topics

1. Derived Data Types – Arrays
2. Local Arrays – Initialization and Access
3. Array Applications

1. Derived Data Type – Arrays

Problems:

Consider the situation where there are `100` numbers of type **`int`** involved in some operations. With many or all of these `100` numbers are needed at once, one may choose to declare 100 variables of **`int`**. This, however, is not an attractive option.

Another problem is to sort (or rearrange) values of some sequence of numbers in certain order. The numbers can be integers or floating-point and the order can either be ascending or descending.

Solution Tools:

A tool to be used in the above problems is to use an array to handle these numbers (which are of the same type, such as `int`).

1.1 Definition

In general, an array is a contiguous block of memory that has zero, one two or more elements of the same type.

In **`C`**, more specifically, an array MUST have ONE or more elements.

Array type is a derived data type, which is created using some combination of a valid type, an operator, and an integer value.

Again in C, consider a general local array being declared as follows,

```
arrayType arrayName[SIZE];
```

where

- `arrayName` is the name of the array,
- `SIZE` is the length of the array as an integer constant, and
- `arrayType` is the type of the element of the array.
- `[]` is called the array operator (in the declaration)

In the above declaration, `SIZE` is a **constant integer,** and that the array declaration as above will have **compiled-time memory allocation**.

Examples

```
int iA1[5];
int iA2[3] = {1, 2, 3};
int iA3[] = {1, 2, 3, 4};
double dA1[3];
```

1.2 Initialization through Storage

The initialization of an array depends on how the array was stored. There are three kinds of storage in C in which a variable’s value can be stored under.

1.2.1 Automatic (Local) Storage

- An **automatic or local storage** will be given to variables and arrays that were defined inside functions (including the formal arguments).
- These variables and arrays will be available only within the functions and they will no longer exist as soon as the functions are done with their execution.

Example

```
int main() {
  int iA1[3];
  int iA2[2] = {1, 2};

  return 0;
}
```

1.2.2 External (Global) Storage

- External (global) storage is given to variables and arrays that were defined outside of any functions.
- Global variables and arrays are available everywhere after they are defined.
- They last through out the program execution.
- Global variables and arrays are initialized with **`zero(s)`** unless other specific values were given with the declarations.

Example

```
int common;
const int MAXSIZE = 5;
int iExtArray[3] = {1, 2, 3};

int main() {
  int iA1[3];
  int iA2[2] = {1, 2};

  return 0;
}
```

1.2.3 ***`Static`*** *Storage*

- A declaration that was modified with the **`static`** keyword will force the system to provide a **static storage**.
- This static storage reserves a common block of memory for a **`static`** variable and its value **will be shared by any invocation** that would require access to this static member.
- **Static** variables are defined inside a function and will have the values initialized to **zero(s)** unless other values were given.

Example

```
int test() {
  static int iCommonValue;
  static int iCommonArray[2] = {1, 2};

  return 0;
}
```

2. Local Arrays – Initialization & Access

- Arrays are declared and should be defined (i.e., assigned with known values) before using.
- The declaration asks the system to allocation one contiguous block of memory that is the size of the array size times the size of the array type. Thus, the total number of bytes for some array is found as follows,

```
SIZE * sizeof(arrayType)
```

With this block of memory, values can be stored in the array and then an array is completely defined.

Let us look at the initialization again.

2.1 Initialization

For local arrays, the initialization can be done in several ways. They can be left undefined, completely specified, or forcing zero completion.

For examples,

```
int iA1[3]; /* No initialization: values are unknown */

int iA2[4] = {1, 2, 3, 4}; /* Complete initialization
                              for all four members */

int iA3[] = {5, 6, 7}; /* Complete initialization for
                          an array of three members */

int iA4[5] = {8, 9}; /* Partially initialized for the
                        first two members; the remaining
                        three members will have value 0 */

int iA5[10] = {0}; /* Partially initialize the first
                      member with 0 and fill in the
                      remaining members with 0’s. That
                      means all values are 0 */
```

2.2 Accessing Arrays

- Array members can be accessed by the array name and a valid index.
- The index should be in the range from `0` to `(SIZE - 1).` Any index that is outside of this range may produce an unknown value and may also alter data belonging to other objects or programs – This will generate index related error (at run-time)

  C has no index or boundary checking. The user must keep track of this range of indexes and make sure the individual index is valid.
- Array members can be either **`lvalue`** or **`rvalue`**. That means one can either retrieve an array member value or assign new value to an array member.

For examples,

```
iA1[0] = 100;
iA2[0] = iA1[0];
iA4[3] = iA3[1] < iA2[2];
iA5[0] = iA1[0] + iA2[0] + iA3[2] - iA4[1];
```

Note that

- Here, the square brackets `[]` are called the **index operator (**or also, **subscription operator)**.
- It has very high precedence and thus its evaluation is almost immediate in many expressions.

3. Array Applications

There are several important applications that would involve arrays such as,

- Storing large amount of data
- Searching for some value
- Sorting some blocks of data

And several tasks will be critical or unavoidable such as,

- Getting largest value
- Getting the smallest value
- Swapping 2 values
- Rearranging a block of values based on given criteria

We will look at or touch on these tasks and applications through arrays and functions.