Fundamentals of Computer Programming

C Programming
5. Arrays and Strings

Contents

- Single-Dimension Arrays
- Generating a Pointer to an Array
- Strings
- Two-Dimensional Arrays
- Multi-Dimensional Arrays
- Indexing Pointers
- Array Initialization
- Variable-Length Arrays

What Are Arrays?

- An *array* is a collection of variables of the *same type* that are referred to through a *common name*.
- A specific element in an array is accessed by an *index*.
- In C, all arrays consist of *contiguous memory locations*.
- Arrays can have from one to several dimensions.
- The most common array is the *string*, which is simply an array of characters terminated by a *null*.

Single-Dimension Arrays

- *type varName[size];*
- *type* declares the base type of the array, which is the type of *each element* in the array, and *size* defines *how many elements* the array will hold.
- For example, to declare a 100-element array called **balance** of type **double**:
  - **double balance[100];**
- An element is accessed by indexing the array name:
  - **balance[3] = 12.23;**
- In C, all arrays have **0** as the index of their *first element*.

Single-Dimension Arrays

```
#include <stdio.h>

int main(void)
{
  int x[100]; /* this declares a 100-integer array */
  int t;

  /* load x with values 0 through 99 */
  for(t=0; t<100; ++t) x[t] = t;

  /* display contents of x */
  for(t=0; t<100; ++t) printf(''%d ", x[t]);

  return 0;
}
```

Generating a Pointer to an Array

- You can generate a pointer to *the first element* of an array by simply specifying *the array name*, without any index.
  - `int *p;`
  - `int sample[10];`
  - `p = sample; // p = &sample[0];`

Strings

- In C, a *string* is *a* null-terminated *character array*.
- When declaring a character array that will hold a string, you need to declare it to be *one character longer than the largest string* that it will hold.
- For example, to declare an array str that can hold a *10-character* string, you would write:
  - `char str[11];`
- A *string constant* is a list of characters enclosed *in double quotes*. For example: **“hello there”**
- You do not need to add the **null** to the end of string constants manually.

[2] pp. 99-100

String Functions

Name	Function
strcpy(*s1*, *s2*)	Copies *s2* into *s1*
strcat(*s1*, *s2*)	Concatenates *s2* onto the end of *s1*
strlen(*s1*)	Returns the length of *s1*
strcmp(*s1*, *s2*)	Returns 0 if *s1* and *s2* are the same; less than 0 if *s1*<*s2*; greater than 0 if *s1*>*s2*
strchr(*s1*, *ch*)	Returns a pointer to the first occurrence of *ch* in *s1*
strstr(*s1*, *s2*)	Returns a pointer to the first occurrence of *s2* in *s1*

[2] pp. 99-100

Two-Dimensional Arrays

- C supports multidimensional arrays.
- A two -dimensional array is, essentially, an array of one-dimensional arrays.
- To declare a two-dimensional integer array **d** of size 10x20, you would write:

```
int d[10][20];
```

- Similarly, to access point **1,2** of array **d**, you would use:

```
d[1][2];
```

Two-Dimensional Array Example

```
#include <stdio.h>

int main(void)
{
  int t, i, num[3][4];

  for(t=0; t<3; ++t)
    for(i=0; i<4; ++i)
      num[t][i] = (t*4)+i+1;

  /* now print them out */
  for(t=0; t<3; ++t) {
    for(i=0; i<4; ++i)
      printf(''%3d ", num[t] [i]);
    printf("\n");
  }

  return 0;
}
```

Arrays of Strings

- To create an array of strings, use *a two-dimensional character array*.
- The size of *the left dimension* determines *the number of strings*, and the size of *the right dimension* specifies *the maximum length* of each string.
- The following declares *an array of 30 strings*, each with a maximum length of 79 characters:

```
char strArray[30][80];
```

- It is easy to access *an individual string*: You simply specify only *the left index*:

```
gets(strArray[3]);
```

Multidimensional Arrays

- Ref: p. 107

Indexing Pointers

- Pointers and arrays are closely related.
- An *array name* without an index is *a pointer to the first element* in the array.
- A pointer can be indexed as if it were declared to be an array:

```
int *p, i[10];
p = i;
p[5] = 100;  /* assign using index */
*(p+5) = 100; /* assign using pointer arithmetic */
```

Array Initializations

- *Character arrays* that hold strings allow a shorthand initialization that takes the form:

```
char arrayName[size] = “string”;
char str[6] = “Hello”;
char str[6] = {‘H’, ‘e’, ‘l’, ‘l’, ‘o’, ‘\0’};
```

- Multidimensional arrays are initialized the same as single-dimension ones.

```
int sqrs[3][2] = { 1, 1, 2, 4, 3, 9};
int sqrs[3][2] = { {1, 1}, {2, 4}, {3, 9}};
```


[2] p. 106

Unsized Array Initializations

- If, in an array initialization statement, *the size* of the array is *not specified*, the compiler automatically creates *an array big enough* to hold all the initializers present:

```
char e1[] = “Read error\n”;
char e2[] = “Write error\n”;
char e3[] = “Cannot open file\n”;
```

- For multidimensional arrays, you must specify all but the leftmost dimension.

```
int sqrs[][2] = { {1, 1}, {2, 4}, {3, 9}};
```


[2] pp. 112-113