ARRAYS AND STRINGS IN CPP

Single-Dimension Arrays

The general form for declaring a single-dimension array is

type var_name[size] ;

Like other variables, arrays must be explicitly declared so that the compiler may allocate space for them in memory. Here, *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**,

use this statement:

double balance[100];

An element is accessed by indexing the array name. This is done by placing the index of the element within square brackets after the name of the array.

For example,

balance[3] = 12.23;

assigns element number 3 in **balance** the value 12.23.

Two-Dimensional Arrays

C/C++ supports multidimensional arrays. The simplest form of the multidimensional array is the two-dimensional array. A two-dimensional array is, essentially, an array of one-dimensional arrays.

To declare a two-dimensional integer array **d** of size 10,20, you would write int d[10][20];

Pay careful attention to the declaration. Some other computer languages use commas to separate the array dimensions; C/C++, in contrast, places each dimension in its own set of brackets.Similarly, to access point 1,2 of array **d**, you would use d[1][2]

The following example loads a two-dimensional array with the numbers 1 through 12 and prints them row by row.

```
#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;
}
```

In this example, **num[0][0]** has the value 1, **num[0][1]** the value 2, **num[0][2]** the value 3, and so on. The value of **num[2][3]** will be 12.

Arrays of Strings

It is not uncommon in programming to use an array of strings. For example, the input processor to a database may verify user commands against an array of valid commands. To create an array of null-terminated strings, use a two-dimensional character array. The size of the left index determines the number of strings and the size of the right index specifies the maximum length of each string. The following code declares an array of 30 strings, each with a maximum length of 79 characters, plus the null terminator. char str_array[30][80]; It is easy to access an individual string: You simply specify only the left index.

For example, the following statement calls **gets( )** with the third string in **str_array**.

```
gets(str_array[2]);
```

The preceding statement is functionally equivalent to

```
gets(&str_array[2][0]);
```

but the first of the two forms is much more common in professionally writtenC/C++ code. To better understand how string arrays work, study the following short program, which uses a string array as the basis for a very simple text editor:

```
/* A very simple text editor. */
#include <stdio.h>
#define MAX 100
#define LEN 80
char text[MAX][LEN];
int main(void)
{
register int t, i, j;
printf("Enter an empty line to quit.\n");
for(t=0; t<MAX; t++) {
printf("%d: ", t);
gets(text[t]);
if(!*text[t]) break; /* quit on blank line */
}
for(i=0; i<t; i++) {
for(j=0; text[i][j]; j++) putchar(text[i][j]);
putchar('\n');
}
return 0;
}
```

This program inputs lines of text until a blank line is entered. Then it redisplays each line one character at a time.

Multidimensional Arrays

C/C++ allows arrays of more than two dimensions. The exact limit, if any, is determined by your compiler. The general form of a multidimensional array declaration is

type name[*Size1*][*Size2*][*Size3*]. . .[*SizeN*];

Arrays of more than three dimensions are not often used because of the amount of memory they require. For example, a four-dimensional character array with dimensions 10,6,9,4 requires 10 * 6 * 9 * 4 or 2,160 bytes. If the array held 2-byte integers, 4,320 bytes would be needed. If the array held **double**s (assuming 8 bytes per **double**), 17,280 bytes would be required. The storage required increases exponentially with the number of dimensions. For example, if a fifth dimension of size 10 was added to the preceding array, then 172, 800 bytes would be required.

In multidimensional arrays, it takes the computer time to compute each index. This means that accessing an element in a multidimensional array can be slower than accessing an element in a single-dimension array. When passing multidimensional arrays into functions, you must declare all but the leftmost dimension.

For example, if you declare array **m** as int m[4][3][6][5]; a function, **func1( )**, that receives **m**, would look like this:

```
void func1(int d[][3][6][5])
{
.
.
.
}
```

Source : http://elearningatria.files.wordpress.com/2013/10/cse-iii-object-oriented-programming-with-c-10cs36-notes.pdf