C/C++ Programming

Parallel Arrays using C-strings

0	AL
1	AK
2	AZ
3	AR
4	CA
5	CO
6	CT
7	DE
8	DC
9	FL
10	GA

0	Alabama
1	Alaska
2	Arizona
3	Arkansas
4	California
5	Colorado
6	Connecticut
7	Delaware
8	District of Columbia
9	Florida
10	Georgia

Dan McElroy

Hello programmers. This presentation introduces the concept of working with parallel arrays where the data in two or more arrays are related to each other by their position in the arrays. This is a presentations for using C-strings. Another presentation is provided for using C++ strings. If you only want to study the C-language, you can skip the presentation on C++ strings. However, if you plan on using C++, I recommend at least studying the code for the C-strings version.

Parallel Arrays

0	AL
1	AK
2	AZ
3	AR
4	CA
5	CO
6	CT
7	DE
8	DC
9	FL
10	GA

0	Alabama
1	Alaska
2	Arizona
3	Arkansas
4	California
5	Colorado
6	Connecticut
7	Delaware
8	District of Columbia
9	Florida
10	Georgia

Sometimes it is convenient to work with two or more parallel arrays. The example below shows two arrays where the data in the two arrays is related to each other by their respective position in the arrays, called the index value.

Search One Array – Display the Other

char *abbrev[ ] = {…}

0	AL
1	AK
2	AZ
3	AR
4	CA
5	CO
6	CT
7	DE
8	DC
9	FL
10	GA

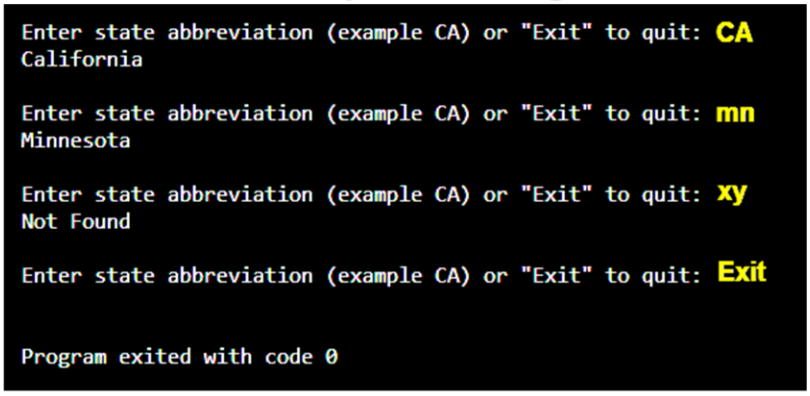
char *stateName[ ] = {…}

0	Alabama
1	Alaska
2	Arizona
3	Arkansas
4	California
5	Colorado
6	Connecticut
7	Delaware
8	District of Columbia
9	Florida
10	Georgia

In this example, the program asks the user to enter an abbreviation, search the stateAbbrev array for a match, remember the index position in the array where the match was found and use the same index to look up the name of the state in the stateNames array. If a match was not found, a "Not Found" message will be displayed.

For example, if the user enters the abbreviation CA, we would search the array that contains the abbreviation until we found "CA" in index position 4, then look up the full name of the state "California" in index 4 in the array that holds the full names of the states.

Sample Program Execution



Here is a sample execution. The program asks the user to enter a state abbreviation or the word "Exit".
The first entry is "CA" in capital letters. The program responds with "California".
The second entry is "mn" in small letters. The program responds with "Minnesota".
The third entry is "xy" which is not an abbreviation for a state. The program responds with "Not Found".
The fourth entry is "Exit" which ends the program.

Sample Program part 1, using C-strings

```
// ParallelArrays.c    C-string version of the program
// User enters state abbreviation, program finds the name of the state

#include <stdio.h>
#include <string.h>
using namespace std;

int main(int argc, char* argv[])
{
    char *stateAbbrev[] = {
        "AL","AK","AZ","AR","CA","CO","CT","DE","FL","GA","HI","ID","IL","IN",
        "IA","KS","KY","LA","ME","MD","MA","MI","MN","MS","MO","MT","NE","NV",
        "NH","NJ","NM","NY","NC","ND","OH","OK","OR","PA","RI","SC","SD","TN",
        "TX","UT","VT","VA","WA","WV","WI","WY"
    };

    char *stateNames[] = {
        "Alabama","Alaska","Arizona","Arkansas","California","Colorado",
        "Connecticut","Delaware","Florida","Georgia","Hawaii","Idaho",
        "Illinois","Indiana","Iowa","Kansas","Kentucky","Louisiana","Maine",
        "Maryland","Massachusetts","Michigan","Minnesota","Mississippi",
        "Missouri","Montana","Nebraska","Nevada","New Hampshire","New Jersey",
        "New Mexico","New York","North Carolina","North Dakota","Ohio",
        "Oklahoma","Oregon","Pennsylvania","Rhode Island","South Carolina",
        "South Dakota","Tennessee","Texas","Utah","Vermont","Virginia",
        "Washington","West Virginia","Wisconsin","Wyoming"
    };
```

If writing the program in C++, the #include statements need to be.
#include <iostream> // for console input/output.
#include <cstring> // to use the C-string library, and
using namespace std;

Here is part 1 of the program showing how the two arrays are each filled with string literals. These arrays contain character pointers to C-strings. The declaration of the arrays start with char *, the name of the array, open/close square brackets [ ] and the = assignment operator. char * indicates that the data type is a pointer to characters which works with both C and C++. Nothing is placed in the square brackets because the size of the arrays are going to be determined by their initialization data. The data for the

array is placed within a curly-brace pair { }. Each string literal is separated by a comma. The last element in the array does not need a comma after it, but most compilers won't complain if you put one there.

The declaration and initialization of each array is similar, except the second array contains strings of state names instead of abbreviations. When working with parallel arrays, it is extremely important that if one array gets updated, the other array must be updated in the same position. For example if Washington DC becomes a state, both arrays need updating in the same location. It can be easy to mess this up, so be careful.

Sample Program part 2, using C-strings

```
    char userRequest[10];
    int  i;     // search index
    int  arraySize = sizeof(stateAbbrev) / sizeof(char *);   // should be equal to 50

    // get first user entry
    printf ("Enter state abbreviation (example CA) or \"Exit\" to quit: ");
    scanf_s ("%s", userRequest, 10);  // read a line from the keyboard

    // NOTE: Visual Studio uses _stricmp most other compilers use stricmp or casecmp
    while ( _stricmp(userRequest, "Exit") != 0)  { // loop until "Exit" is entered, ignore case
        for (i=0; i<arraySize; i++)  {   // search the Abbrev array for a state abbreviation
            if ( _stricmp(userRequest, stateAbbrev[i]) == 0) { // match found
                printf("%s\n\n", stateNames[i]); // display name of state
                break;      // break out of loop before reaching end of the array
            }
        }
        if (i == arraySize)   // end of array reached without finding a match
            printf("Not Found\n\n");

        // get next user entry
        printf("Enter state abbreviation (example CA) or \"Exit\" to quit: ");
        scanf_s("%s", userRequest, 10);  // read a line from the keyboard
    }
    return 0;
}
```

Here is the executable code that does the actual work. First the variables are declared for the program. char userRequest[10]; is declaring an array of 10 characters that will be used when inputting from the keyboard. The integer i will be used to index through the arrays. arraySize is computed to be the number of elements in the stateNames array by taking the entire memory size of the array of character pointers and dividing it by the size of an individual character pointer. This should be equal to 50. The nice thing about having the program compute the number of elements is that if states get added to the arrays, then only the arrays themselves need updated and the program will automatically compute the new size.

The program starts executing code by asking the user to

enter a state abbreviation or the word "Exit". A C++ program would use cout instead of printf.

The program then inputs the user's selection as a string of multiple characters. A C-language program would use either scanf("%s", userRequest); or scanf_s("%s", userRequest, 10); with Visual Studio. Previously when using scanf or scanf_s we needed to use the & address-of operator in front of the name of an integer or double data-type variable that is going to receive the data from the keyboard. When scanf or scanf_s is inputting into an array, we don't need the & address-of operator because the name of an array is automatically the address of the array. A C++ program uses cin.getline(userRequest, 10); where userRequest is the name of the character array, size 10.

The program has two loops. The outer loop is a while loop is used to get the two-character state abbreviation from the user and then use the inner for loop to search the stateAbbrev array for a match. The outer for loop ends when the user types the word "Exit" instead of a state abbreviation. The construction of the outer while loop looks like a sentinel value loop in that the first input is done before the start of the loop.

The while loop and its test condition is shown here as
while(_stricmp(userRequest, "Exit") != 0) { .
The test condition says keep executing the loop as long as it evaluates to TRUE.

I am using _stricmp to compare two C-strings because I am using Microsoft Visual Studio. Your compiler may be using stricmp without the underscore, or maybe even casecmp. It would be really nice if every compiler used the same code for string compare.

The letter i in stricmp or the word case in casecmp says to ignore case when comparing the strings. This way it does not matter if the user types the word "exit" or "Exit" with capital or small letters. These functions return a negative integer if the first string is less than the second string, a zero if both strings are the same, or a positive value if the first string is greater than the second string. As long as the user's input is not the word "Exit", the result of the string compare will be non-zero and the code inside the while loop will execute.

Three things are inside the while loop.
1) a for loop is used to search the stateAbbrev array for a match with the user's input. If a match is found, the full name of the state is displayed from the stateName array and the break; statement exits the loop with the variable i set to the index in the array where the match was found. If a name is not found, then the for loop will end with i equal to 50 because the for loop's test statement is i<arraySize;

2) The second thing in the while loop is an if statement that checks to see if i is equal to arraySize. If it is, then the for loop made it all the way to the end of the array without

finding a match and executing the break statement. Then the "Not found" message is output.

3) The third thing in the while loop completes the construction of a sentinel value loop by getting the next piece of data, which in this case is asking for another state abbreviation and then going back to the top of the while statement. It is here that the test to see if the user typed "Exit". If so, the while loop ends and the return 0; statement ends the program.

The End

This is the end of the discussion on parallel arrays using C-strings. If C++ is your language of preference, then you should definitely study the similar presentation on C++ strings. Especially the parts about declaring an array of C++ strings and how the string compare ignore case works. Until next time, bye for now.