CSC 180 lab 6: Arrays

Thurs Oct 18 or Mon Oct 22, 2001

An *array* is an "aggregate" data type in which all elements are of the same type. An array is an array of some type, and it has a specified size. For example, we can have an array of ten ints. This would be declared by saying `int whatever[10];`. As with other variables, you can use any variable name where I've written "whatever".

1. One of the central purposes of arrays is that you can select the array element with an expression, which can involve variables whose values are not determined until run-time. Valid indices range from 0 to one less than the dimension of the array, e.g. from 0 to 9 for our `int whatever[10];`. In C, it is the programmer's responsibility to ensure that only valid array indices are used.

Write a program which inputs and stores three numbers in an array of three ints, then in a loop accepts an index number (0, 1, or 2), and tells the user what that index's value is. The user can type -1 to exit the program.

2. Arrays can be initialized by a special syntax involving braces:

```
int myarray[5] = { 41, 52, 63, 74, 85 };
```

Also, in this case the array dimension can be omitted, as follows:

```
int myarray[] = { 41, 52, 63, 74, 85 };
```

When the array dimension is omitted, the compiler counts the number of initializers, and creates an array of exactly that size. The array dimension may only be omitted when there is an initializer present.

Since strings are arrays of char, there is an analogous but more specialized array initialization syntax for arrays of char:

```
char mystring[] = "Hello, world";
```

Write a program which contains a declaration such as the above, and then, as in question 1, repeatedly in a loop inputs an array index and outputs the character at that point. Output it two ways each time: once with a printf format of %c and once with a printf format of %d, showing the ASCII value for that character.

3. An array name mentioned in an expression context decays into a pointer to the zeroth element of that array. In fact, almost all of the time we are dealing with arrays, we are dealing with pointers into them.

Here is a declaration of an array of pointers-to-char which contains the names of the days of the week:

```
char *weeknames[] = {
    "Sunday", "Monday", "Tuesday", "Wednesday", "Thursday",
    "Friday", "Saturday"
};
```

In this array, element number *i* is the name of weekday number *i*. It is of type pointer-to-char; the actual array element is a pointer to the character beginning the appropriate string. The declaration declares an array of pointers-to-char (of unspecified size, to be filled in by counting the number of initializers (which is seven)).

Write a program which defines this array and outputs the name of an input day number. Remember that zero-terminated strings (such as the above) can be printed with the printf format "%s".

(You can probably copy and paste the above array definition from the web page, more or less...)

4. Following the previous program, modify your weekday-printing program from assignment one so that it prints the day name instead of the day number.

(over)

5. The following program fragment displays the digits of the number ‘x’ in base three, but does it in reverse order:

```
while (x > 0) {
    printf("%d", x % 3);
    x = x / 3;
}
```

For example, for x == 22, which is 211 in base 3, it will output “112”.

Use an array to store the digits of the number in base three so that you can then print them out in the correct order.

6. Since an array name in an expression context decays into a pointer to the zeroth element, when you pass an array to a function you are actually passing a pointer.

Write a function which takes an argument of type pointer-to-int and another argument of type int, and adds up the numbers in the array. The second argument is the dimension of the array. The function will return the sum as an int. Your function declaration will look like this:

```
int add(int *a, int size)
```

Note that you can still use the array subscripting syntax inside this function! For example, if size≥2, then a[1] is the element at index 1. (The array subscripting *always* works with pointers; if “a” is an array, then in the expression "a[1]", “a” decays into a pointer to its zeroth element.)

7. Write a main program which declares an array of size 10, fills it with the numbers 1 through 10, then calls the above function and prints the result (which should be 55).