ECE 131 – Programming Fundamentals

Module 4, Lecture 1: Pointers, Arrays and Structures – Basic Arrays

Dr. Daryl O. Lee

University of New Mexico

Declaring Arrays

- In programming, an array is a collection of values, all of the same type, grouped together under one name.
- The array elements are indexed, and you gain access to a particular array element through its index.
- The syntax for declaring an array in C:

  $$\mathit{type}\ \mathit{identifier}[\mathit{expr}] < \, = \{\mathit{initialization\ string}\} >;$$

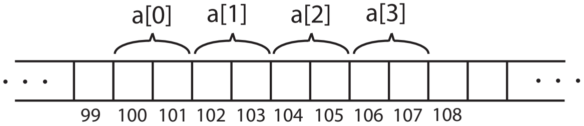
  where *type* is any valid C data type, either built-in or user-defined, *identifier* is the name of the array, which can be any valid C identifier, *expr* is a C expression that determines how many elements will be in the array, and an optional initialization string, enclosed in curly braces, can be used to store initial values in the array.

Ex: `int a[4] = {2, 4, 5, 1};`

Declaring Arrays

In C, array elements are assumed to be stored in computer memory in consecutive memory locations.

Ex: `int a[4];`



- In this example, we're assuming that an `int` is stored in two bytes, so the array occupies a total of 8 consecutive bytes. If each `int` required 4 bytes, then 16 consecutive bytes would be allocated as a result of the array declaration above.
- We'll make extensive use of the fact that array elements are stored in consecutive memory locations when we discuss pointers and pointer arithmetic in the next lecture.

THE UNIVERSITY *of* NEW MEXICO

Using Arrays

- There are no array operators in C, i.e., operators that take arrays as operands.

  Ex: The following will not work in C:

  ```
  int a[3]={1,2,3}, b[3]={3,2,1}, c[3];
  c = a + b;
  ```

- In C, you must write code to operate on each element of the array that you want to manipulate.
- This is not true for other programming languages. In particular, many programming languages that are oriented towards mathematics support operations that can be applied to arrays.

  Ex: Matlab and FORTRAN 90 support array operations.

Using Arrays

Ex. The following program will compute the average age from a set of ages supplied in an array:

```
#include <stdio.h>
main()
{
   int i, age[5]={18, 17, 23, 21, 17};
   float avg=0;
   for(i=0; i<5; i++)
      avg += age[i]; // operate on each array element
   printf("The average age is:  %2.1f\n", avg/5);
}
```

producing the result: The average age is: 19.2

- Notice that we operate on each array element by indexing into the array using `age[i]`.

Using Arrays

- Notice that arrays in C are zero-based — the first element in an n-element array is array_name[0], and the last element is array_name[$n-1$]. I.e., the valid array indices are $0, 1, \ldots, n-1$.
- In other programming languages, e.g., Matlab and FORTRAN, arrays are one-based, in which case the valid indices in an n-element array are $1, 2, \ldots, n$.
- We'll see that zero-based arrays make more sense if pointers exist in the language — Matlab and FORTRAN do not support pointers.

Using Arrays

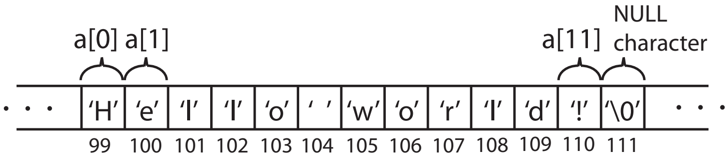
- Important: C compilers are not required to check if you are indexing off the end of an array. This is a common programming error in C, and is called a buffer overflow.
- Buffer overflows are one of the most common attack techniques used in malicious software.
- It may happen that the following compiles and runs without error:

```
#include <stdio.h>
main()
{
   int i, age[5]={18, 17, 23, 21, 17};
   float avg=0;
   for(i=0; i<10; i++)
      avg += age[i]; // operate on each array element
   printf("The average age is:  %2.1f\n", avg/5);
}
```

producing the result: The average age is: -205390432.0

Strings and Arrays

- In C, string constants are created by surrounding text with double quotes.
  Ex: `char a[] = "Hello world!";`
- The compiler will store the string in an array of `char`s, and will append a null character ('\0') to the end of this array.
  Ex:



- String-related functions use the null character to determine when the end of a string has been reached.

Strings and Arrays

Recall our first C program:

```
#include <stdio.h>
main()
{
  printf("Hello World!\n");
}
```

which printed “Hello World” to the standard output (our terminal screen).

This program can be rewritten as:

```
#include <stdio.h>
main() {
  char a[] = "Hello World!\n";
  int i;
  for (i=0; a[i] != ’\0’; i++)
    printf("%c", a[i]);
}
```

–or–

```
#include <stdio.h>
main() {
  char a[] = "Hello World!\n";
  printf("%s", a);
}
```

Sizeof operator

The 'sizeof' operator gives the size of its operand.
The operand may be either a type or a variable:

```
#include <stdio.h>
main() {
  char a = ’x’;
  int n = 3;
  float x = 3.14;
  printf("size of char = %d\n", sizeof(char));
  printf("size of n = %d\n", sizeof(n));
}
```

Using sizeof to get array length

Goal: let the program compute array length
Problem: no “length” operator for arrays.

```
#include <stdio.h>
int main() {
  int length;
  float x[] = {1.414, 3.14, 2.718, 6.02e23};
  length = sizeof(x)/sizeof(x[0]);
  printf("length of array x = %d\n", length);
  return 0;
}
```