Data Structures in C

Prof. Georg Feil

Arrays and Strings

Summer 2018

Acknowledgement

- These lecture slides are partly based on slides by Professor Simon Hood
- Additional sources are cited separately

Reading Assignment (required)

- C for Programmers (supplementary textbook)
  - Chapter 6, sections 6.1 to 6.7
  - Chapter 8, sections 8.1, 8.2, 8.6, 8.7

Arrays

- The general idea of arrays in C is similar to Java
- Recall that an array is a container that can hold many values
  - Using an array size 1000 is much better than using 1000 separate variables
  - All the values must have the same data type, e.g. int, double…
- The array itself is a variable and has a data type
  - For example “array of int”
- Each value in the array is called an element
  - The elements have index numbers starting at 0 up to (size - 1)
- The size of an array is given when declaring the array and is fixed (can’t be changed later)

Arrays in C

- Declaring and filling an array with numbers in C looks like this

```
int values[30];
for (int index = 0; index < 30; index++) {
    values[index] = 999;
}
```

- Notice there's no 'new', just give the length in square brackets
- In C there's no easy way to retrieve the length of an array later like in Java
  - There is no array.length

Arrays in C

- To access the first element of our array, we type `values[0]`
- To access the last element of this array we should use `values[29]`
  - Here the index is the length of the array minus one
- If an element has already been set to something, we can use it in a calculation

```
    values[1] = values[0] * 2;
```

Array length

- Since a C array doesn’t “know” its length, it’s a good idea to keep track of the length using a separate variable like a constant

```
const int numValues = 30;
int values[numValues];
for (int index = 0; index < numValues; index++) {
    values[index] = 999;
}
```

- To access the last element of this array we would use `values[numValues-1]`

Array bounds checking

- C does not check if an array index is valid
  - Remember C is very efficient… that would just waste time!

- What happens if an array index is out of bounds?
  - Unpredictable…
  - You might get the wrong results, or a strange crash

- Try it with the short program on the previous slide
  - Print the last element of the array
  - Print one past the last element… or way past the end
  - Print elements *before* the array (!?)

Array Initialization

- Just like other local variables, C does not initialize arrays that are local variables
  - Contents could be random junk (whatever was in memory)
- Arrays that are global variables, or declared static, are always initialized (filled with zero)
- We can initialize arrays with specific values, e.g.

```
int myArray[] = {10, 20, 30, 40, 50};
```

- Notice that you can leave out the size if you want, the C compiler will calculate it for you
  - Empty square brackets go after the variable name, not data type!

Examples: Array Initialization

- double grades[] = {91.3, 50.1, 88.0, 69.9};

- bool flags[] = {true, true, false, true, false};

- char alphaBack[] = {'z', 'y', 'x', 'w', 'v', 'u', 't', 's', 'r', 'q', 'p', 'o', 'n', 'm', 'l', 'k', 'j', 'i', 'h', 'g', 'f', 'e', 'd', 'c', 'b', 'a'};

Array and Array Element Data Types

- Suppose we create this array:

  char letters[] = { ‘A’, ‘B’, ‘C’, ‘D’ };

- What is the data type of letters?
  - “array of char”
- What is the data type of letters[2]?
  - Its data type is char
  - This is one element of the array, one of the things in the container
- What is the value of letters[2]?
  - ‘C’

Exercise 1: Statistics Calculator

- Write a C program that inputs 5 integers from the user and stores them in an array
    - Use a constant so you can easily change the ‘5’ later
- After all the numbers have been entered your program should
    - Print out all the numbers
    - Calculate and display the average, maximum, and minimum values
- Here’s what a sample run might look like:

```
Please enter 5 integers: 1 2 -1 0 4
You entered 1, 2, -1, 0, 4
The average is 1.2
The highest number is 4
The lowest number is -1
```

Value Types vs. Reference Types

- C’s fundamental (primitive) types are value types
  - A variable contains its value (number, character, boolean)
  - Assigning one variable to another copies the value
  - Passing a value type as a parameter copies the value, and the called function cannot change the original value
- Arrays are reference types
  - A variable contains a reference to data in memory
  - Assigning one variable to another does not copy the data
  - If you pass an array as a parameter the array is not copied, and the function can change its contents
    - *The original array is changed!*

Example of passing an array parameter

```
#include <stdio.h>

double ave(int arr[], int len) {
    int total = 0;
    for (int index = 0; index < len; index++) {
        total += arr[index];   // Add up all the numbers
    }
    return (double)total/len;  // Calculate the average
}

int main(int argc, char** argv) {
    int numbers[] = {29,5,-7,101,-555}; // Initialize array
    int len = 5;

    // Call function to calculate the average and print it
    printf("The average is %f\n", ave(numbers, len));
}
```

Don't need to give a size, can pass any size array

Passing value types and reference types

What’s the difference?

```
// What will the output of the following program be?

double ave(int arr[], int len) {
    int total = 0;
    for (int index = 0; index < len; index++) {
        total += arr[index];   // Add up all the numbers
        arr[index] = 0;
    }
    double result = (double)total/len;  // Calculate the average
    len = 0;
    return result;
}

int main(int argc, char** argv) {
    int numbers[] = {29,5,-7,101,-555}; // Initialize array
    int len = 5;

    // Call function to calculate the average and print it
    printf("The average is %f\n", ave(numbers, len));

    for (int index = 0; index < len; index++) {
        printf("%d ", numbers[index]);   // Print out all the numbers
    }
    printf("\n len is %d", len);
}
```

Passing reference types with const

```
#include <stdio.h>

// To guarantee a parameter will never be changed use const
double ave(const int arr[], int len) {
    int total = 0;
    for (int index = 0; index < len; index++) {
        total += arr[index];   // Add up all the numbers
    }
    return (double)total/len;  // Calculate the average
}

int main(int argc, char** argv) {
    int numbers[] = {29,5,-7,101,-555}; // Initialize array

    // Call function to calculate the average and print it
    printf("The average is %f\n", ave(numbers, 5));

    for (int index = 0; index < len; index++) {
        printf("%d ", numbers[index]); // Print out all the numbers
    }
}
```

Exercise 2: Sorting and Median

- Extend your program from Exercise 1 so that it displays the median of the list of numbers
  - You can find how to calculate a median here
- Before finding the median you'll need to sort the numbers!
  - Write a function that sorts an array of integers (review the bubble sort algorithm)
  - The function should have two parameters
    - The integer array to sort
    - The length of the array
  - Call the function to sort your array

C Strings

“Null-terminated strings”

C Strings

- C does not have a “String” data type
- Instead, Strings are created by making an array of characters

```
char str[10];
```

- If we want, we can initialize the array at creation

```
char myString[] = "Today is the day!";
```

  - The compiler automatically calculates the required array size

String size

- Just like other arrays, strings have a fixed size and we can’t change the size except by creating a whole new string
- You should declare strings to be more than big enough for all possible uses of your program
- When initializing a string, supply a size if it might need room to grow

```
    char myString[100] = "Today is the day!";
```

- If you want the string to be empty at first, use this

```
    char bigString[256] = "";
```

String termination (null)

- Q: If we create a char array of size 100, but only store 17 chars in it, how does it know where the end of the string is?

```
    char myString[100] = "Today is the day!";
```

  - If we print myString using printf, it will only print 17 characters, not 100!
- Answer: Every C string is terminated by a null character
  - The null character is ASCII code zero
  - Written as '\0'
- When C sees the null character it knows it has reached the end of the string, even if there is a more room in the array
  - The null character is never printed by printf
  - If the null is missing this may cause errors or crashes!

String literals

- As in Java, string literals use double quotes "xxxxx" and character literals use single quotes 'x'
  - String literals have a null at the end (it’s “invisible”)
- When C sees the null character it knows it has reached the end of the string, even if there’s a more room in the array
  - The null character is never printed by printf
  - But the null does take up space in memory!
- Q: How many bytes does the string "hello" occupy in memory?
- Answer: 6 bytes, 5 for hello and 1 for the terminating null
  - When declaring strings be sure to leave room for the null… a string to hold the word “hello” must have size 6 or more

String literals

- When creating strings from individual characters you must remember to add the terminating null character

- Here’s one way to create a string from characters

```
char myString[100] = {'H','e','l','l','o','\0'};
```

- This is the same as

```
char myString[100] = "Hello";
```

String library functions

- C comes with many useful headers for the C library
  - We’ve been using <stdio.h> so far
- Another useful one is <string.h>
- We can access string manipulation functions by using `#include <string.h>`
  - strlen
  - strcmp
  - strcpy
  - strcat
  - etc.

String length

```
unsigned int strlen(const char s[])
```

- String length (strlen) returns the length of the string up to but not including the terminating null '\0'
  - It does not return the size of the char array!
  - It works by scanning through the string counting characters
- For the following string, strlen will return 5

```
char myString[100] = "Hello";

int len = strlen(myString);
```

String length (cont'd)

- One way to use strlen is to loop through a string using a for loop

```
char str[200] = "This is a string";
for (int index = 0; index < strlen(str); index++) {
    printf("%c\n", str[index]);
}
```

- To avoid calculating the string length many times do this

```
int len = strlen(str);
for (int index = 0; index < len; index++) {
    printf("%c\n", str[index]);
}
```

String compare

```
int strcmp(const char s1[], const char s2[]);
```

- Just like in Java, you can’t use comparison operators like `==` and `!=` to compare strings
- Use strcmp() to compare strings, it will return…
  - -1 if s1 is less than s2 (alphabetically)
  - 0 if s1 equals s2
  - 1 if s1 is greater than s2 (alphabetically)
- Can use strcmp to check equality, or alphabetical order
- This is like the String.compareTo method in Java

How to compare strings

- Do this:

```
if (strcmp(str1, str2) == 0) {
    // Strings are the same
}
```

- Don't do this:

```
if (str1 == str2) {
    // This is wrong (but no compile error)
}
```

String copy

```
char* strcpy(char dest[], const char src[])
```

- String copy (strcpy) copies the entire contents of src into dest
- This means that whatever was in dest before is destroyed in the process
  - Afterward, doing strcmp(dest, src) will return 0
- Remember you can’t make a copy of an array just by assigning variables, and strings are arrays!
- You must ensure that the destination has enough room
  - If not, bad/unpredictable things will happen!

Example: Using strcpy to set a string

- If you have an existing string variable, you can’t assign a string to it

```
char str[100];
…
str = "hey";          // This is a syntax error!
```

- You must copy the strings

```
char str[100];
…
strcpy(str, "hey");   // This is correct
```

Example: Using strcpy to copy a string

- To make a copy of an existing string variable you can’t do it using ‘=’ (assignment operator)
- Use strcpy:

```
char str1[100] = "This is a sentence to be copied";
char str2[100];


strcpy(str2, str1);  // Correct way to copy
```

String concatenate

```
char* strcat(char dest[], const char src[])
```

- String concatenate (strcat) appends the contents of src onto the end of dest
- Be careful to ensure that the dest string is large enough
  - strcat doesn’t know the available size of ‘dest’
  - The destination must have enough room for both strings combined, plus one terminating null character

Safer string functions

- With strcpy, strcat etc. it’s quite easy to accidentally write or read beyond the char array bounds
  - This is known as buffer overflow and can cause nasty crashes, undefined behavior, and security holes
- The C library provides safe versions of string functions that let you specify maximum sizes to stay within array bounds
  - strncmp
  - strncpy
  - strncat
  - strnlen
  - etc.
- Use these for “production” quality code

Example of using strncpy

- Here’s how a C function might use strncpy to safely make a copy of a string it was passed
    - If a very long string is passed it won’t overflow the string ‘cpy’
    - Note that strncpy doesn’t guarantee a null terminator will be added for very long strings, so we always add one just in case

```
void processString(char str[])
{
    const int size = 100;
    char cpy[size];  // Will hold a copy of str
    strncpy(cpy, str, size); // Copy at most 100 chars
    cpy[size-1] = '\0'; // Ensure it's null terminated
    printf("The copied string is %s\n", cpy);
}
```

Exercise 3: Display every other character in a string

- Write a program that inputs a string from the user and displays every other character in the string
  - For example if you enter “ThisIsABigString” your program should output “TiIAiSrn”

- Hints
  - Create a char array of length 100 to store the string
  - To read a string with scanf use %s, but *don't* put ‘&’ in front of the string variable
  - Use the length of the string as your loop end point, and increment by 2
  - Test your program thoroughly, watch out for off-by-one errors!

Exercise 4: Create a string one character at a time

- Modify your program from Exercise 3 so that it creates a new string containing the result
  - Don’t print the result one character at a time
  - If you enter “ThisIsABigString” your program should create a new string containing “TiIAiSrn”
    - Don’t forget to add the terminating null
  - Copy the new string to the original string using a library function, overwriting the original
  - Finally, print the original string (which has been updated)

Safer user input

- When reading strings with scanf and %s there’s a danger of exceeding the char array (string) bounds… buffer overflow!

```
    scanf("%s", str);    // This is unsafe
```

- To prevent buffer overflow specify a field width before the ‘s’ that’s one less than the string size (to leave room for the terminating null)

```
    char str[100];
    scanf("%99s", str);  // This is safe
```

- Scanf with %s will stop reading at spaces, you can’t input a sentence or first & last name with space in between… to read a whole line use

```
    scanf("%99[^\n]", str);  // Read a whole line safely
```

or

```
    fgets(str, 100, stdin);  // This is good too
```