Data Structures in C

Prof. Georg Feil

Working With Pointers

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 7, sections 7.1 to 7.9

Passing pointers as parameters to functions

- Recall that when passing fundamental (primitive) types like int or double to a function the value gets copied
  - These are value types, and passing them is “call by value”
- When passing arrays the array data is not copied so the called function can change the original array
  - Arrays are reference types, and passing them is “call by reference”
- *Pointers work like references* when passing them to functions
  - The data pointed to does not get copied, so the called function can change the data. We can consider this an output parameter.

Exercise 1: Call by value, call by reference

- Write a main function that calls both of these functions to increment a number.
  - Declare one int variable in the main function (init to zero)
  - Print the value of the variable before and after each call
  - Remember: To convert a variable to a pointer put & in front

```
void increment(int num) {
    num = num + 1;
}

void increment2(int* num) {
    *num = *num + 1;
}
```

Exercise 2: Function returning more than one value

- Write a function called powers that accepts a double parameter and returns two doubles using output parameters: the square and cube of the original number
- Your function declaration should look like this

```
void powers(double num, double* square, double* cube)
```

- Write a main function to test that the powers function works

Pointer Arithmetic

Pointer Arithmetic

- We can add to pointer variables to move forward in memory
- We can also subtract to move backward in memory
- Pointer arithmetic is a very powerful tool when working with C data structures
  - Allows algorithms to move freely around data in memory
- How pointer arithmetic works depends on the pointer data type… let's try it first with characters (bytes)

Example: Pointer arithmetic with char

- We can make a char pointer point to the start of a string like this:

```
char str[] = "Today is the day!";
char* pos = str;
```

- The simplest way to increment a pointer is using ++. We can use this to iterate through the string:

```
while (*pos != '\0') {
    printf("%c\n", *pos);
    pos++;
}
```

Try running the code on this page. Print out the pointer address to watch it change.

Example: Pointer arithmetic with char

- This loop can be written more concisely:

```
while (*pos != '\0') {
    printf("%c\n", *pos++);
}
```

- Here's how to make a pointer point to any character in the string (not just the first)

```
char* pos = str + 2;   // Points to 3rd character
char* last = str + (strlen(str) - 1);   // Last char
```

Exercise 3

- Write a program that inputs a string from the user then prints the characters in the string both forwards and backwards
  - Hint: Use two pointers

```
Please enter a string: Aardvark
A k
a r
r a
d v
v d
a r
r a
k A
```

Pointer arithmetic "step" size

- When we increment a char pointer the memory address increments by 1
- When we increment an int pointer the memory address increments by 4
- The size of the increment or decrement matches the size of the pointer's data type (the type pointed to)
  - This helps ensures that pointers don't end up misaligned
  - Programmers can forget about data sizes: Add one to get to the next data item in memory (no matter its size), or subtract one to move backward
  - Add a larger number to move ahead many data items in memory

Exercise 4: Pointer arithmetic with int

- We can make an int pointer point to the start of an int array like this:

```
int nums[] = {87, -5, 33, 2, -9};
int* pos = nums;
```

- Write a program to print out this array using the pointer
- Also print out the pointer addresses
- How would your code change for an array of doubles?

Exercise 5: strlen

- Write your own version of the strlen function
  - Use a different name, for example myStrlen
- It should return an int and accept one char pointer as a parameter
- Use pointers and pointer arithmetic, don't use [ ] syntax
  - See how concise (short) you can make the code
- Test your function and compare the results to the regular strlen function

Additional Pointers Exercise: strcmp

- Write your own version of the strcmp function
  - Use a different name, for example myStrcmp
- Declare your function like this:

```
int myStrcmp(const char* str1, const char* str2)
```

- Use pointers and pointer arithmetic, don't use [ ] syntax
  - Remember you must stop looping when you reach the null terminator of either string
- Test your function and compare the results to the regular strcmp function