Introduction to Computer Science and Programming in C

Session 16: October 28, 2008

Columbia University

Announcements

- Homework 3 is out. Due November 6th before class
- Everybody check your homework 2 submission files. If something is wrong with your tar file, go to office hours and get help submitting.

Review

- Pointer: variable that stores memory address
- Declare using:

```
int * x_ptr; /* a pointer called x_ptr to an int*/
```

- Pointer operations:
  - * <pointer> – the thing <pointer> points to
  - & <variable> – the address of <variable>

Today

- Pointers and Arrays
  - (correction on argv)
- Memory Management

Some vocabulary

- * operator is also known as **dereference**
- a pointer **references** a variable in memory

Pointers and Arrays

- C blurs the distinction between pointers and arrays
- When we declare an array
  `char A[10];`
  what is A?
  - A can be treated as a pointer to the first element of A

Pointers and Arrays

- In other words, the following two lines are equivalent:
  - `char * array_ptr = &A[0];`
  - `char * array_ptr = A;`
- This also means the following:
  - `A[0] == *array_ptr`
  - `A[1] == *(array_ptr+1)`

Pointers and Arrays

- When we want a function to be able to modify the value of a variable, we pass it **by reference**

```
sscanf(price, "$%f", &dollars);
```

- Because arrays are basically pointers, this happens *automatically* when we pass arrays to functions.

- For example:

```
strcpy(stringA, stringB);
```

Pointer Arithmetic

- What if **A** was an array of ints?
  `A[1] == *(array_ptr+1) ??`
- Yes. C automatically keeps pointer arithmetic in terms of the size of the variable type being pointed to.
- Be careful to keep track of what C does for you and what it does not.

*argv[]

- `int main(int argc, char *argv[])`
- Last class we were unsure if *argv[] is a pointer to an array or an array of pointers:
- If it was a pointer to an array, it would just be an array.
- So (char *) argv[]
  (*argv)[1] points to the first character of the first word

Memory Management

- We discussed before that C does not like to initialize arrays with variable sizes.
- To get around this, you can use stdlib.h's **malloc()** command.
- malloc() stands for memory allocation.
- malloc(N) returns a pointer to an allocated block of memory of N bytes.

malloc()

- Typical usage:

```
int N = 40000;
char *giantString = malloc(N*sizeof(char));
```

- Returns a null pointer if malloc fails.
- When we are done with the memory, we can free it with:

```
free(giantString);
```

Management

- With malloc() and free(), we are able to use arbitrary amounts of memory and able to clear memory to save space.
- This is one aspect of C that makes some people consider C too powerful.
- Many other languages have automated memory management.

Memory Leaks

- ```
  int N = 40000;
  char *giantString = malloc(N*sizeof(char));
  strcpy(giantString, argv[1]);
  giantString = malloc(N*sizeof(char));
  ```
- Now a huge block of memory is allocated but the program has no way of finding it.
- If this code runs a lot, the amount of memory the program is using will keep growing.
  ```