COMS 1003: Introduction to Computer Programming in C

Pointers 1 & 2

November 1st and 3rd 2005
Announcements

- Good constructive feedback on the midterm comments
- No class November 8th. Go vote.
- Read Chapter 5 in K&R
  - go back and re-read the chapter on structs and unions, now paying heed to the information on pointers
Outline

• Recall the properties of variables
  - Recall the outline of main memory

• Learn how to:
  - declare pointers
  - dereference pointers
  - obtain addresses,
  - do pointer arithmetic
  - use the sizeof operator
Variables

• Represent an object that you can store data into
  - a cell of memory (or group of cells)
• Have a name (address)
• Have a type
• Hold (contain) some value
Mental Model: Main Memory

- Array of cells with address and value
- 32-bit machine, 1 word = 4 bytes
- Primitive types occupy some amount of memory
- Use 'sizeof' operator in C to determine how many bytes a type takes up
  - sizeof(int) == 4, sizeof(char) == 1
Intro to Pointers

• A variable that holds the address of another variable
  – The value of a pointer variable is the address of another memory cell (or group of cells)

• this is the proper, appropriate, and legal semantics of pointers
  – but you can store arbitrary integer values into pointers. Dangerous!
Pointer Operators

• Declaration operator: *
  
  - int *y_ptr = NULL;

• 'Addressof' operator: &
  
  - int x = 456;
    y_ptr = &x;

• 'Dereference' operator: *
  
  - printf("val of addr in y_ptr = %d\n",*y_ptr);
Reasons to Use Pointers

- Can refer to a large data structure in a compact way
- Facilitate data sharing between functions
- Can dynamically allocate memory
- Can be used to indicate logical relationships between data structures
sizeof Operator

- Can be used to determine amount of storage necessary for both types and named variables

```c
int num_horses = 100;
printf("%d",sizeof(int));
printf("%d",sizeof num_horses);
```
Pointer Arithmetic

- Pointer values are treated slightly differently from regular variables
- Assignment works as expected, assigning and comparing to zero
- Addition, Subtraction (pointer and integer)
- Subtracting or Comparing two pointers in the same array
Pointer Math Operations

- All semantically legal operations listed on previous slide
- No other operations are legal
  - cannot add two pointers
  - no mul, div, shift, mask, add/sub float, double
Generic Pointer Type

• Many times, you may not know what specific pointer type you are going to point to, need flexibility

• The generic pointer type is void*
  - void *foo = NULL; // foo can point to any type

• The NULL pointer points to nothing
  - consistent default initialization value
Pointers to Functions

• Functions are just code blocks stored in memory, so it is entirely legal to have a variable that holds the location of a function

```c
int say_hello()
{
    printf("hello\n");
    return 0;
}
```
Passing Arguments to Functions

• C has *call by value* semantics for function parameters

• What if you need to modify an argument's value as well as return another value?

```c
int set_zero(int var) {
    var = 0;
    return SUCCESS;
}
```

```c
int set_zero(int *var) {
    *var = 0;
    return SUCCESS;
}
```
Passing Arguments to Functions

- If a function takes a pointer argument, then you need to pass in a pointer or an address.

```c
int set_zero(int *var)
{
    *var = 0;
    return SUCCESS;
}
int main()
{
    int x = 500;
    int result = set_zero(&x);
}
```
String Processing With Pointers