Getting Started with Pointers

What is a Pointer?

- A pointer is a variable designed to hold a memory address
- Pointers are easy to understand.
- But two things make using them difficult to work with:
  1) The C/C++ syntax for pointers is somewhat inconsistent
  2) Surprise - if you make a little mistake, it can really mess up your memory big-time.
- In general, there are three steps to using a pointer:
  1) Declare a pointer variable
  2) Give it a value, e.g. the address of another variable
  3) "Dereference" the pointer to get to the data it points to (to get to the data at the address it is storing)
- Pointers provide direct and fast access to memory

Example: int pointer

```
int main(){
    int x;
    int *p;   // p is an integer pointer; its type is int *

    x = 5;
    p = &x;  // p is assigned the address of x
    // Both of these printf statements print the value of x:
    printf("x is %d ", x);
    printf("x is %d ", *p);  // the * operator is used to "dereference" a pointer

    *p = 6;  // This changes x to 6

    int y = 10;
    p = &y;
    *p = 20;      // What changes? x? y? neither? both?
}
```

Example: char pointer

```
int main(){
    char x;
    char *p;   // p is a char pointer; its type is char *

    x = 'e';
    p = &x;  // p is assigned the address of x
    // Both of these printf statements print the value of x:
    printf("x is %c ", x);
    printf("x is %c ", *p);  // the * operator is used to "dereference" a pointer

    *p = 'f';  // This changes x to 'f'

    char y = 'g';
    p = &y;
    *p = 'h';      // What changes? x? y? neither? both?
}
```

Function Parameters

- A function parameter may be a pointer. The function can change the variable whose address is passed to it.

```
int main(){
    int x = 5;
    int y = 5;
    foo(x, &y);
    printf("x is %d, y is %d", x, y);     // what prints?
}

void foo(int a, int *b){
    a = a + 2;
    *b = *b + 2;
}
```

Function Return Types

- A function may return pointer.

```
int main(){
    int x = 5, y = 10;
    int *p;
    p = biggest(&x, &y);
    if (*p == 0)
        printf("same");
    else
        printf("biggest: ", *p);
}

int * biggest(int *p1, int *p2){
    if (*p1 > *p2)
        return p1;
    else if (*p2 > *p1)
        return p2;
    else return 0;
}
```

null pointer

- With every C compiler, address 0 does not hold any valid data
- By convention, we assign 0 to a pointer if we want to indicate that it's not pointing to anything. Likewise, a method that returns a pointer can return 0 to indicate failure, or no data.
- C does not have a keyword called "null" (like Java does).
- However if you #include <stdio.h> or <stdlib.h> you have access to a macro called NULL which is defined to be 0:

```
#define  NULL  0
```

Then in your programs you can do things like this:

```
int *p = NULL;   // initialize the pointer to 0

if ( foo() == NULL )  // if foo() returns NULL
```

Traversing an Array

A pointer is commonly used to traverse an array.
Typically, we use "pointer arithmetic" to advance a pointer through an array

```
int data[] = { 18, 6, 3, 9, 1, 12 };
int size = 6;
```

```
// Using [] notation:
int i = 0;
for (i = 0; i < size; i++)
    printf("%d ", data[i]);
```

```
// Advancing a pointer along an array
// (This approach is commonly used)
int i = 0;
int *p;
for (p = data; i < size; p++, i++)
    printf("%d ", *p);
```

```
// Alternative way to use a pointer
int i = 0;
int *p;
for (p = data; i < size; i++)
    printf("%d ", *(p + i));
```

Traversing an Array, cont.

The array and the pointer that traverses it should have the same type.
This example is just like the last slide, except it uses doubles vs. ints.
To traverse an array of doubles, use a double * (pointer to double)

```
double data[] = { 18.0, 6.0, 3.0, 9.0, 1.0, 12.0 };
int size = 6;
```

```
// Using [] notation:
int i = 0;
for (i = 0; i < size; i++)
    printf("%f ", data[i]);
```

```
// Advancing a pointer along an array
// (This approach is commonly used)
int i = 0;
double *p;
for (p = data; i < size; p++, i++)
    printf("%f", *p);
```

```
// Alternative way to use a pointer
int i = 0;
double *p;
for (p = data; i < size; i++)
    printf("%f", *(p + i));
```

Pointer Arithmetic, sizeof

- We can add or subtract a number from a pointer (most typically, 1). Why? so that we can easily traverse an array

- If you add 1 to an int pointer, it will point to the next int in memory,
  If you add 1 to a char pointer, it will point to the next char in memory,
  If you add 1 to a double pointer, it will point to the next double,
  and so on

- sizeof reports the number of bytes in a data type, which can vary from machine to machine. Examples:

```
printf("The size of an int on this machine is %d bytes.", sizeof (int) );
printf("The size of a char on this machine is %d bytes.", sizeof (char) );
```

- Example:

```
double  data = { 1, 2, 3, 4, 5 };
double  *p = data;
. . .
p++;     // p = p + sizeof(double)
```

C Pointers, Java References

- C pointers are at a "lower level" than Java references. A C pointer is an actual virtual address.

- In C, you can work directly with addresses in ways not available in Java:
  1. With the & operator
  2. Using pointer arithmetic (p++, *(p + 5), p--)
  3. With direct addressing (dangerous but valid):

```
int *p;
p = 50002010;   // compiles & runs, even if the address is invalid
int x = *p;     // Either crashes or assigns to x whatever
                // data is at the address stored in p.
```

- In Java,
  - references are used only for objects, not primitives
  - reference values are not actual virtual addresses; they refer to a table of virtual addresses managed by the JVM. The JVM periodically rearranges objects in memory (changing their virtual addresses) to defragment it.
  - the only reference value that causes a crash is null (null pointer exception)