Pointers

CSE 130: Introduction to Programming in C
Spring 2005

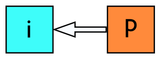
Pointer Basics

- A pointer is a variable that contains a memory address
  - pointer variables (like memory addresses) are integers
- Pointers are used to access memory and manipulate addresses
  - Ex. scanf() takes a pointer to a variable

Pointer Addressing

- & is the (unary) address operator
- If v is a variable, then &v is the memory address at which v is stored
- Ex:

```
int i, *p;
p = &i;
```



Pointer Dereferencing

- The dereference (indirection) operator * is used to access the value stored in a pointed-to memory location
- NOTE: this value is NOT the value stored in the pointer!
  - The pointer holds a memory address
  - * returns the value stored at that address

Pointer Examples

```
int a = 7;
int *p = &a; /* p points to a */
printf("*p = %d\n", *p); /* prints 7 */
*p = 3; /* a is now 3 */
printf("a = %d\n", a); /* prints 3 */
p = 0; /* only legal integer assignment */
```

More Examples

```
double x, y, *p;
p = &x;
y = *p; /* equivalent to y = *&x or y = x */

int a, *p = &a; /* &a is p's initial value */
int *p = &a, a; /* a must be declared first! */
```

Printing Pointers

- The %u format prints an address as an unsigned decimal integer
- The %p format prints an address in a system-specified format (ex. hexadecimal)

int i = 77, *p = &i;

printf("i's address: %u or %p", p, p);

  - prints "i's address: 234880252 or dfffcfc"

Pointers to `void`

- Pointers may not be assigned to one another unless they are of the same type (i.e., both int*), or one of them is a pointer to void
- Pointer to void is a "generic" pointer type

int *p; void *v;

p = v; /* legal */

v = p; /* legal */

v = 1; /* illegal */

Call-by-Reference

- Normally, variables are passed by value — their values are copied to the function parameters
  - Variables not changed in the calling ftn
- Using pointers, we can call by reference
  - Variables ARE changed in the calling ftn

An Example

```
void swap (int *p, int *q)
{
 int tmp;
 tmp = *p;
 *p = *q;
 *q = tmp;
}
```

Pointers and Arrays

- Pointers and arrays are closely related
- An array expression is a pointer to the first element of the array
- Thus, given an array A, pa = *A[0] means that pa and A have the same value
  - This can also be written as pa = A;
- BUT: A = pa is not legal (A is not a variable)

Pointer Arithmetic

- If pa is a pointer to an array A:
  - A[i] can be written as *(pa + i)
- Where pointers are concerned, (pa + 1) points to the next object, regardless of the variable type or size

An Example

- int A[] = {1, 3, 5, 7, 9};
- int *pa = A; /* pa points to A[0] */
- printf("%d", *pa); /* prints 1 */
- printf("%d", *(pa + 3)); /* prints 7 */
- pa++; /* pa now points to A[1] */
- printf("%d", *pa); /* prints 3 */

The Structure Pointer Operator (->)

- Minus sign followed by greater-than sign
- Usage:

  pointer_to_structure -> member_name
- Equivalent to:

  (*pointer_to_structure).member_name
  - The parentheses are necessary here!
    - . and -> have equal precedence

Structure Pointers

```
struct student temp, *p = &temp;
temp.grade = 'A';
p -> lastName = "Bushker";
p -> studentID = 590017;
printf("%c\n", (*p).grade);
```