Advanced Programming
COEN 11

Lecture 3

1

Pointers

Address Operator

- A variable can be referenced using the **address** operator **&**

example

```
scanf("%f", &x);
```

- This statement specifies that the value read is to be stored at the address of x

Pointers

- A **pointer** is a variable that holds the *address* of a memory location
- ***p*** points to ***q***
  - Variable ***p*** holds the address of variable ***q***
  - Variable ***q*** is at location x in memory
    - ***p*** would have the value x (*q*'s address)

Declaring a pointer variable

- Pointer variables are declared using an asterisk (*):
  - Examples:

```
int     *ip;     // ip is a pointer to an integer
char    *cp;     // cp is a pointer to a char
```

Declaring a pointer variable

- When a pointer is defined, the type of the variable to which it will point must be specified
  - Example: a pointer defined to point to an integer cannot also point to a floating point variable.

Example

```
int         *ip;
double      *dp;
```

- Variable ip is declared to point to an int
- Variable dp is declared to point to a double
- Neither variable has been initialized
- Declaring a pointer creates a variable capable of holding an address

More about declaring pointers

- The * operator does not distribute
  - Example

```
int     *p, q;
```

  - *p* is declared to be a pointer to int.
  - *q* is declared to be an int.

Operators & and *

- The operator & in front of an ordinary variable produces the address of that variable.
- The operator * in front of a pointer produces the value pointed by the pointer.

Assigning values to a pointer

- the assignment operator (=) is defined for pointers
  - the right operand can be any expression that evaluates to the same type as the left
  - Example
    - pointer = address
    - pointed value = value

Example

```
int      i = 6, j;
int      *ip;

ip= &i;
j = *ip;
```

Practice!

- Give a memory snapshot after each set of assignment statements

```
int a=1, b=2, *ptr;
ptr = &b;

int a=1, b=2, *ptr=&b;
a = *ptr;
```

NULL pointer

- NULL is a void pointer
  - A symbolic constant defined in <stdio.h>
  - A pointer can be assigned or compared to the void pointer NULL
  - A pointer variable whose value is NULL is not pointing to anything that can be accessed.

Example

```
int       *ip= 0;
char      *s = 0;
double    *dp = NULL;
```

Pointer Assignments

- A pointer can point to only one location at a time
  - but several pointers can point to the same location.

Pointer Assignments

- Example

```
int x = -5, y = 8, *ptr1, *ptr2;

ptr1 = &x;
ptr2 = ptr1;
```

Pointer Arithmetic

- The following arithmetic operations are supported
  - +, -, ++, --, +=, -=
  - These operations determine pointer movements
  - Only integers may be used in these operations

Pointer Arithmetic

- Arithmetic is performed relative to the variable type being pointed to
- Example: *p*++;
  - when applied to pointers, ++ means increment pointer to point to next value in memory
    - if *p* is defined as int **p*, *p* will be incremented by 4 bytes (system dependent)
    - if *p* is defined as double **p*, *p* will be incremented by 8 bytes

Comparing Pointers

- You may compare pointers using relational operators
- Common comparisons are:
  - check for null pointer (*p* == NULL)
  - check if two pointers are pointing to the same object
    - Are these equivalent?
      - (*p* == *q*)
      - (**p* == **q*)
  - compare two pointers that are pointing to a common object such as an array.

Pointers and Arrays

- The name of an array is
  - the address of the first elements (i.e., a pointer to the first element)
  - a constant that always points to the first element of the array and its value cannot be changed.

Pointers and Arrays

- Array names and pointers may often be used interchangeably.
- Example

```
int num[4] = {1,2,3,4}, *p;
p = num;          //the same as p = &num[0];
printf ("%i", *p);
p++;
printf ("%i",  *p);
```

More Pointers and Arrays

- You can also index a pointer using array notation
- Example:

```
char string[ ] = "This is a string";
char *str;
int i;
str = string;
for (i = 0; str[i] != '\0'; i++) // look for end of the string
   printf ("%c", str[i]);
```

Pointer Arithmetic

- Can you traverse a character array using a short pointer?

Pointers and Functions

- An address passed as an argument to a function is received as a pointer
  - Changing the value pointed, affects the original value

switch Example

```
void switch2 (int *a, int *b)
{
   int temp;
   // Switch values pointed to by a and b.
   temp = *a;
   *a=*b;
   *b=temp;
   return;
}
```

Example

```
a ( )
{
   int  x[3] = {0, 1, 2};
   int  m = 8;
   int  n = 10;
   int o = 0;
   int  *p = x;

   o = b (m, &n, p);
}
```

```
int
b (int m, int *n, int *q)
{
   m++;
   *n += 100;
   *q = 10;
   return (m);
}
```

Pointers and Functions

- A function receiving an array as an argument can receive it in two formats
  - As an array, which can be also used as a pointer
  - As a pointer, which can be also used as an array
- Example:

```
void some_function (char *);
void some_function (char [ ]);
```

Pointers and Functions

```
void
some_function (char *str)
{
   printf ("%s\n", str);
   printf ("%c\n", *str);
   str++;
   printf ("%c\n", str[0]);
}
```

Pointers and Functions

```
void
some_function (char str[ ])
{
   printf ("%s\n", str);
   printf ("%c\n", *str);
   str++;
   printf ("%c\n", str[0]);
}
```

Pointers and 2D Arrays

- A two-dimensional array is stored in sequential memory locations, in row order.
- To use the indices to access values in an array received by a function, need to declare the argument as an array
  - Example

```
void some_function (char array[ ][NCOLS])
{
…
}
```

Pointers and 2D Arrays

Example

int s[2][3] = {{2,4,6}, {1,5,3}}, *sp=s;

Memory allocation:

s[0][0]	2	s[0][1]	4	s[0][2]	6
s[1][0]	1	s[1][1]	5	s[1][2]	3

A pointer reference to s[0][1] would be *(sp + ***1***)
A pointer reference to s[1][1] would be *(sp + ***4***)

row offset * number of columns + column offset

Common Pointer Problems

- Using un-initialized pointers

```
int *ip;
*ip= 100;
```

  - Pointer ip has not been initialized.
  - The value 100 will be assigned to some memory location.
- Errors are due to
  - Incorrect/unintended syntax
  - Out-of-range pointers

Precedence

- () []
- ++ -- + - ! (type) & * right to left
- * / %
- + -
- < <= > >=
- == !=
- &&
- ||
- ?: right to left
- = += -= *= /= %= right to left
- ,

Precedence

```
int     x[10] = {0, 5, 10, 15, 20, 25, 30, 35, 40, 45};
int     y = 5;
int     *p = x;

y = *p++;
printf ("%d, %d\n", y, *p);

y = *(p++);
printf ("%d, %d\n", y, *p);

y = (*p)++;
printf ("%d, %d\n", y, *p);
```

Precedence

```
int     x[10] = {0, 5, 10, 15, 20, 25, 30, 35, 40, 45};
int     y = 5;
int     *p = x;

y = ++*p;
printf ("%d, %d\n", y, *p);

y = *(++p);
printf ("%d, %d\n", y, *p);

y = ++(*p);
printf ("%d, %d\n", y, *p);
```

Pointers to Pointers

- Example

```
int **p;
int *q;
int x = 5;

q = &x;
p = &q;
printf ("%d, %d, %d\n", x, *q, **p);
```