Computer Programming Using C
COP 3275 - Summer 2017

Lecture 17: Pointers (Cont.)

By Ahmed E. Khaled

/* Programming */
Recap to the previous lecture:

- Each variable in a program occupies one or more bytes of memory, where the address of the first byte is said to be *the address of the variable*.

- Addresses can be stored in special variable type named *pointer variables*.

- When we store the address of a variable \( i \) in the pointer variable \( X \), we say that \( X \) “points to” \( i \).
The Address and Indirection Operators

- C provides a pair of operators for use with pointers.
  - To find the address of a variable, we use the \& (address) operator (e.g., \&x is the address of variable x in the memory).
  - To gain access to the object that a pointer points to, we use the * (indirection) operator. (e.g., *p is the object to which p currently points to).
# Example

```c
int i, *p;
p = &i;
i = 1;

printf("%d\n", i);  /* prints 1 */
printf("%d\n", *p);  /* prints 1 */

*p = 2;

printf("%d\n", i);  /* prints 2 */
printf("%d\n", *p);  /* prints 2 */
```
Pointer Assignment

• C allows the use of the assignment operator to copy pointers of the same type.

• Assume that the following declaration is in effect:

```c
int i, *p, *q;
p = &i;
q = p;
/* copy the content of p which is the address of i into q */
```

![Diagram showing the assignment of a pointer to another pointer]
If \( p \) and \( q \) both point to \( i \), we can change \( i \) by assigning a new value to either \(*p\) or \(*q\) (indirection operator):

\[ *p = 1; \]

\[ *q = 2; \]
q = p is not the same as \[ *q = *p \]

\[ p = &i; \quad q = &j; \quad i = 1; \]
\[ *q = *p; \]

/*Copies the value that p points to (value of i) into the location that q points to (variable j)*/

![Diagram](image-url)
Pointers as arguments

- What are pointers good for? Since pointers have several uses in C, we will point one application:

Passing a pointer to a variable when calling a function, it’s possible for the function to change the variable’s value instead of the *pass-by-value*. 
void decompose (float x, int *int_part, float *frac_part) {
    *int_part = (int) x;
    *frac_part = x - *int_part;
}

main(){
    int i;
    float j;
    decompose (3.14, &i, &j);
}

When decompose is called, the value of 3.14 is copied into variable x, a pointer to i is stored in int_part and a pointer to j is stored in frac_part.
When decompose returns, i and f will have values 3 and 0.14.
Find both max and min in an Array

```c
void max_min(int a[], int n, int *max, int *min);
```

- Example call of max_min:
  ```c
  int N=100, B[N], big, small;
  max_min(B, N, &big, &small);
  ```

When max_min is called, the max value is stored in variable big and the min is stored in variable small. Array B and variable N are passed by value, while big and small are passed by reference.
void max_min(int a[], int n, int *max, int *min) {
    int i;
    *max = *min = a[0];
    for (i = 1; i < n; i++) {
        if (a[i] > *max)
            *max = a[i];
        else if (a[i] < *min)
            *min = a[i];
    }
}

main(void) {
    int N=10, b[N], i, big, small;
    printf("Enter %d numbers: ", N);
    for (i = 0; i < N; i++)
        scanf("%d", &b[i]);

    max_min(b, N, &big, &small);
    printf("Largest: %d and Smallest: %d\n", big, small);
}
Using `const` to Protect Arguments

• When an argument is a pointer to a variable `x`, we normally assume that `x` will be modified:

```c
f(&x);
```

• It’s possible, though, that `f` merely needs to examine the value of `x`, **not change it**.

• The pointer might be efficiency: passing the value of a variable can **waste time and space** if the variable requires a **large amount** of storage.
We can use `const` to document that a function won’t change an object whose address is passed to the function.

`const` goes in the **parameter’s declaration**, just before the specification of its type:

```c
void f(const int *p)
{
    printf("value is: %d\n", *p); /* Correct */
    *p = 0;                       /* WRONG */
}
```

Attempting to modify `*p` is an error that the compiler will detect.
Pointers as Return Values

• For example, we may want a function to return the location of an answer instead of returning its value (discussed in strings)

```c
int *max(int *a, int *b){
    if (*a > *b) return a;
    else return b;
}
```

• A call of the `max` function requires two integer values to be passed as reference and store the result in a `pointer variable`:

```c
int *p, i, j;
p = max(&i, &j);
```

After the call, `p` points to either `i` or `j`. 
• Although `max` returns one of the pointers passed to it as an argument, that’s not the only possibility.

• Some functions return a pointer to one element of an array passed as an argument, or return a pointer to an external variable.

• Never return a pointer to a local variable:

```c
int *f(void){
    int i;
    return &i;
}
```

The variable `i` won’t exist after `f` returns.