McGill

COMP 208
Computers in Engineering

Lecture 16

Jun Wang
School of Computer Science
McGill University

Fall 2007

McGill

COMP 208 – Computers in Engineering

Review

- Function prototype
  - function declaration must appear before any invocation of the function
    - declare all functions at the beginning of the file
    - create a header file, and in the source code include the header file
- type casting (coercion)

```
(type) expression
```

- global variables: variables defined outside all functions

Review

- Pointers: variables that hold address of other variables

```
type *name;
```

  – To get the address of a variable, we use the & operator

  – To get the value of the variable pointed by a pointer, we use the * operator.

```
int x = 3, y = 5;
int *ip = &x;
y = *ip; // y set to 3
*ip = 6; // x set to 6
```

Aha! Now we know!

That resolves an old issue we had!

In the `scanf` function, we used `&a` to read into the variable `a`

Now we know why!

Without the `&a`, the value of a would not have changed.

Pointers and Arrays

- Pointers are very closely linked to arrays in C
- There is a duality between an array, which is a block of memory cells, and a pointer to a memory location
- The array name is a pointer to the first of these cells

Pointers and Arrays

```
int a[10], x;
int *pa;

pa = &a[0];  /* pa gets address of a[0] */

x = *pa;  /* x gets contents of pa (a[0]) */
```

`&a[0]` is the same as `a`.

Pointer Arithmetic

- C allows us to add integer values to pointers
- Adding a value, i, to a pointer gives the address of the ith memory cell following
- If the pointer, `arr`, references an array `arr + i` is equivalent to `arr[i]` (but more efficient)

a[2] is equivalent to *(a+2)

A Sorting Algorithm

- Computers are frequently used to sort data stored in arrays
- We will soon look at several different ways this can be done
- For now we will look at a sorting algorithm that illustrates the use of a swap

Bubble Sort

We can compare pairs of values working backwards through the array.

When two values are out of order, swap them

When we are finished one pass, the smallest value is at the front of the array (it "bubbles" down)

We repeat this process until all the values are in order

A Note on Parameters

- We want to pass an array as a parameter to the sorting algorithm
- We want the contents of the array to be modified
- Do we have to pass the parameter as a pointer?
- NO! An array is a pointer.

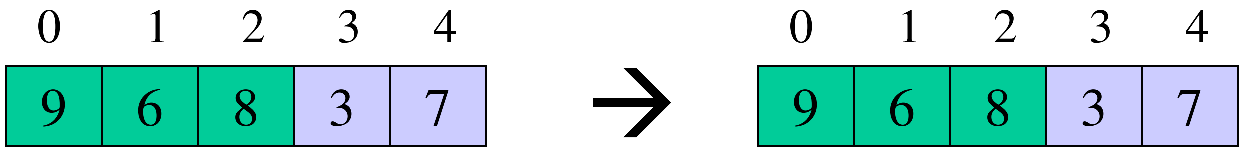
Bubble Sort

```
void bubble_sort(int arr[], int size)
{
    int i, j;
    for (i=0; i<size-1; i++)
    {
        for (j=size-1; j>i; --j)
        {
            if (arr[j] < arr[j-1])
                swap (&arr[j], &arr[j-1]);
        }
        //at this point:
        // 1. elements from 0 to i have been sorted
        // 2. elements 0 to i are smaller than the rest of the array: the
        // smallest i+1 numbers are in their correct positions
    }
}
```

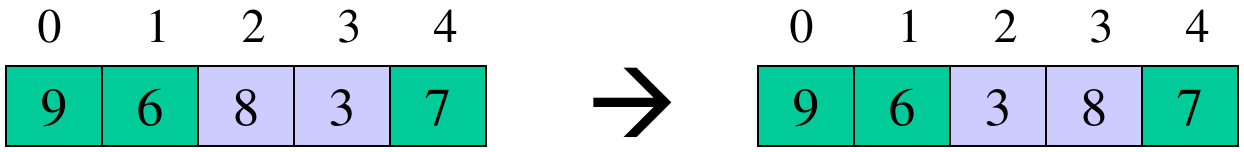
example

Iteration 1: i=0; j= 4 to 1

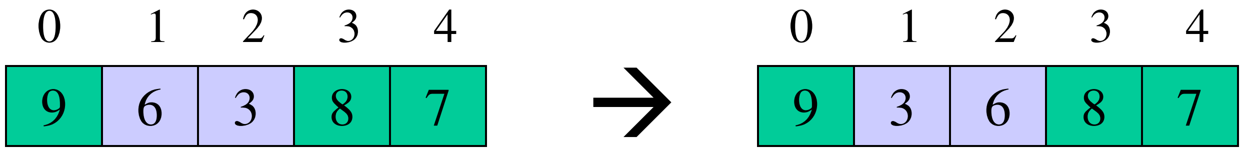
1. j=4, settle a[4] and a[3]



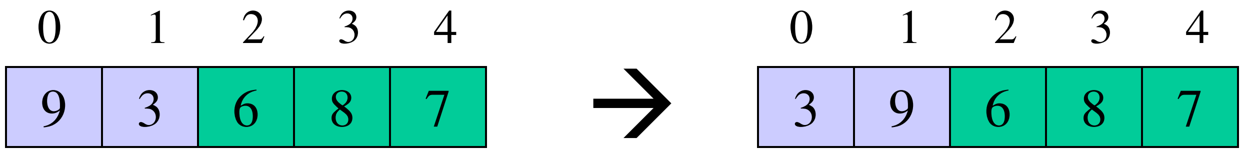
2. j=3, settle a[3] and a[2]



3. j=2, settle a[2] and a[1]



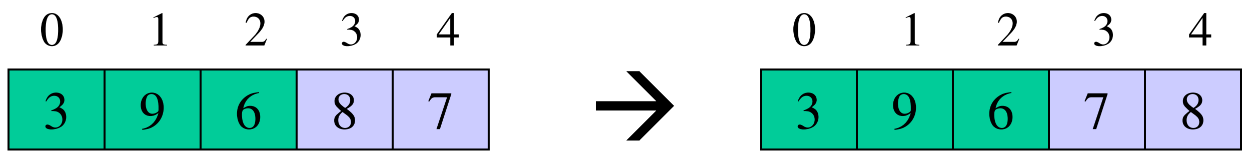
4. j=1, settle a[1] and a[0]



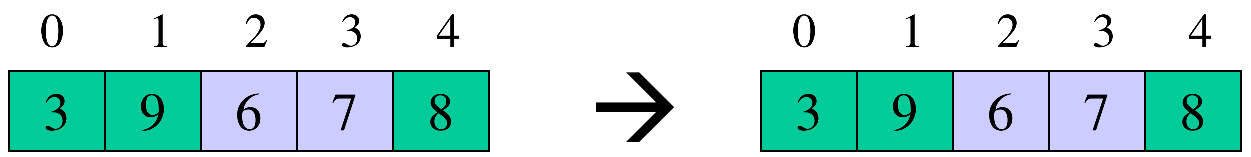
example

Iteration 2: i=1; j= 4 to 2

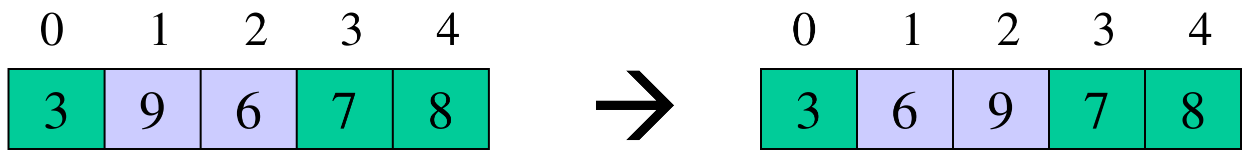
1. Settle a[4] and a[3]



2. Settle a[3] and a[2]



3. Settle a[2] and a[1]

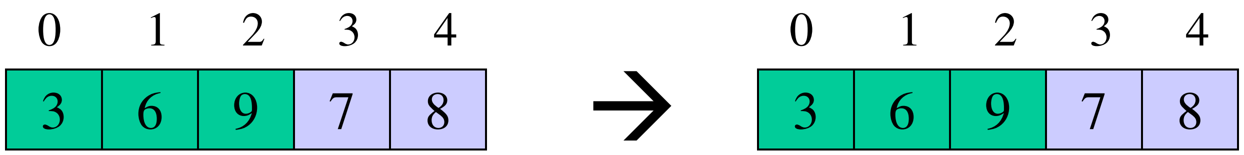


In the 2nd iteration, the 1st element is already in its correct position, and is not affected.

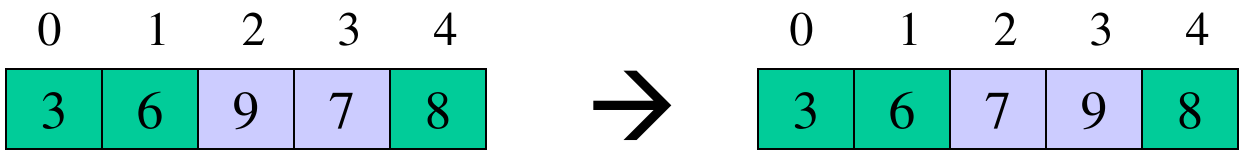
example

Iteration 3: i=2; j= 4 to 3

1. Settle a[4] and a[3]

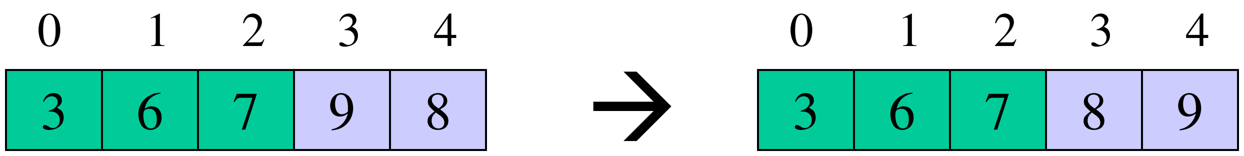


2. Settle a[3] and a[2]



Iteration 4: i=3; j= 4 to 4

1. Settle a[4] and a[3]



File Input

- To read from a file, you first declare a file pointer of type FILE

```
FILE* name;
```

- `FILE` is uppercase because it is an implementation dependent macro
- Once declared, you use `fopen` to associate the name with an actual file

```
FILE* datafile = fopen("test.data", "r");
```

- The header file is `stdio.h`.

File Specifications

```
FILE* datafile = fopen("test.data", "r");
```

- The “r” specifies that the file is to be read from
- To write to a file, we use “w”
- To append to a file, we use “a”

Reading and Writing

- In place of `scanf`, we use `fscanf` and specify the file to read from

```
fscanf (datafile, "%f%d", &value,
    &count);
```

- To write to a file we use fprintf instead of printf and also specify the file

```
FILE* results;
results = fopen("test.result", "w");
fprintf (results, "The average of %d
    values is %f.\n", totalcount, ave);
```

Closing a File

- As in Fortran, when we are finished reading from or writing to a file, the file should be closed:

```
fclose(results);
```

Writing to a File

When we open a file, the filename is assigned a nonzero value if the operation is successful and a value of zero if it fails.

```
FILE* myfile;

myfile = fopen(“name”, “r”);
if (myfile == NULL) {
    printf(“Could not open file.\n”);
    exit(1);
}
```

or

```
FILE* myfile;

if ((myfile = fopen(“name”, “r”)) == NULL) {
    printf(“Could not open file.\n”);
    exit(1);
}
```

Random Numbers

- Many applications use random numbers
- Before we go on with pointers, lets have a quick look at how to generate a sequence of numbers that looks random
- It really isn’t random, hence **pseudo**random numbers

Pseudo Random Numbers

Make sure to include the needed libraries

```
#include <stdlib.h>
#include <time.h>
```

The first step is to seed the pseudo-random number generator.

For testing, we can always reproduce the same sequence if we start with the same seed.

For "production" we might choose an arbitrary seed

```
srand((unsigned int) time(NULL));
```

Pseudo-Random Sequence

Once the random number generator has been seeded, the next number can be generated with

```
rand()
```

This generates a number in the range from 0 to `RAND_MAX` (which is often 32767 but may vary with different implementations)

Restricting the Range

To generate a random real number between 0 and 1, you could use

```
(double) rand() / RAND_MAX
```

To get a number in the range from x0 to x1, you could generate a number between 0 and 1 as above and then scale it as follows

```
num * (x1-x0) + x0
```

McGill

Searching and Sorting

Nathan Friedman

Fall, 2007

Where’s Waldo?

- A common use for computers is to search for the whereabouts of a specific item in a list
  - When you use a banking machine, the machine will search for your account info
  - When you log in to WebCT, the program will search for your personal info

- The most straightforward approach is just to start looking at the beginning and go on from there

Linear Search

```
int linear_search(int val, int arr[], int size)
{
    int i;

    for(i = 0; i < size; ++i) {
        if(arr[i] == val)
            return i;
    }

    return -1; // not found
}
```

Finding Max

The algorithm to find the location of the largest value in an array has a similar structure.

To find the smallest, just invert the comparison

```
int find_biggest(int arr[], int size)
{
    int index_of_big = 0, i;

    for(i = 0; i < size; ++i)
        if(arr[i] > arr[index_of_big])
            index_of_big = i;

    return index_of_big;
}
```

Searching Sorted Lists

- Is that the way we would look up a name in the Montreal telephone directory?
- I hope not!

Binary Search

To search a **sorted** array, we could check the middle element

The value we are looking for might be there

If not we can determine whether it is in the first or second half of the array and search that smaller array

Iterative Binary Search

```
int binary_search(int val, int arr[], int size)
{
   int left = 0, right = size-1, middle;
   do {
      middle = (left + right) / 2;
      if(arr[middle] < val)
         left = middle + 1;
      else if(arr[middle] > val)
         right = middle - 1;
      else
         return middle;
   } while(left <= right);
   return -1;
}
```