

CSC 221: Computer Programming I

Fall 2001

- file input
 - ifstream
 - open, >>, close
 - end-of-file
- file output
 - ofstream
 - open, <<, close
- input with whitespace
 - get, getline

arrays recap

important points:

- arrays provide capability for storing related values in a single entity
- can access individual values using an index
- can traverse through the indices, systematically access all values in the array
- can pass the entire array as a single parameter
- for efficiency reasons, arrays are treated as pointers (references) to memory

arrays make it possible to store and access large amounts of data

- requiring the user to enter lots of data by hand is tedious
each execution of the program requires re-entering the data
- *better solution*: store the data in a separate file, read directly from the file
requires only one entry by the user, can re-read as many times as desired

file input

to read from a file, must declare an *input file stream*

- similar to the standard input stream, only input comes from a file
- in particular, can read values using >>
- defined in the <fstream> library

```
#include <fstream>           // loads definition of the input
                             // file stream class (ifstream)
. . .

ifstream myin;              // declares input file stream
myin.open("nums.dat");      // opens the input file stream
                             // using the file "nums.dat"

int numbers[100];
for (int i = 0; i < 100; i++) { // reads numbers from the input
    myin >> numbers[i];        // file stream & stores in array
}

myin.close();               // closes the input file stream
```

averaging grades

suppose we wanted to store grades in a file

to compute average, don't need to store the grades in an array – just read and process

```
OPEN INPUT FILE STREAM;
INITIALIZE GRADE COUNTER, SUM;

READ FIRST GRADE;
while (GRADE IS NOT SENTINEL){
    ADD GRADE TO SUM;
    INCREMENT GRADE COUNTER;

    READ NEXT GRADE;
}

COMPUTE & DISPLAY AVERAGE
```

```
// avg.cpp
//
// Averages the numbers from a file.
//
#include <iostream>
#include <fstream>
using namespace std;

int main()
{
    ifstream myin;
    myin.open("grades.dat");

    int numGrades = 0, gradeSum = 0;

    int grade;
    myin >> grade;
    while (grade != -1) {
        gradeSum += grade;
        numGrades++;

        myin >> grade;
    }

    if (numGrades > 0) {
        double avg = (double)gradeSum/numGrades;
        cout << "Your average is " << avg << endl;
    }
    else {
        cout << "There are no grades!" << endl;
    }

    myin.close();

    return 0;
}
```

storing grades

but suppose we wanted to allow the user to selectively display or manipulate the grades

e.g., want to allow the user to specify a cutoff grade, then determine the # above the cutoff → must store in an array

pseudo-code:

```
OPEN INPUT FILE STREAM;  
READ GRADES & STORE IN ARRAY;  
PROMPT USER FOR CUTOFF;  
TRAVERSE ARRAY & COUNT;
```

note: when you declare an array, must specify a maximum size

in this case, don't know how many grades there will be

as a result, must guess a maximum size and hope it's big enough

cutoff display

```
#include <iostream>  
#include <fstream>  
using namespace std;  
  
const int MAX_GRADES = 100;  
  
void ReadGrades(int grades[], int & numGrades);  
int CountAbove(const int grades[],  
               int numGrades, int cutoff);  
  
int main()  
{  
    int grades[MAX_GRADES], numGrades;  
    ReadGrades(grades, numGrades);  
  
    int cutoff;  
    cout << "Enter the desired grade cutoff: ";  
    cin >> cutoff;  
  
    cout << "There are "  
         << CountAbove(grades, numGrades, cutoff)  
         << " grades above " << cutoff << endl;  
  
    return 0;  
}  
  
////////////////////////////////////
```

```
void ReadGrades(int grades[], int & numGrades)  
// Results: reads grades and stores in array  
//          numGrades is set to the # of grades  
{  
    ifstream myin;  
    myin.open("grades.dat");  
  
    numGrades = 0;  
  
    int grade;  
    myin >> grade;  
    while (grade != -1) {  
        grades[numGrades] = grade;  
        numGrades++;  
  
        myin >> grade;  
    }  
    myin.close();  
}  
  
int CountAbove(const int grades[],  
               int numGrades, int cutoff)  
// Assumes: grades contains numGrades grades  
// Returns: number of grades >= cutoff  
{  
    int numAbove = 0;  
  
    for(int i = 0; i < numGrades; i++) {  
        if (grades[i] >= cutoff) {  
            numAbove++;  
        }  
    }  
  
    return numAbove;  
}
```

Improvement 1: end-of-file

a sentinel value is really not necessary for a file

- the physical end-of-file marks the end of input

input via `>>` evaluates to a Boolean value

- if the input succeeds (the expected type of input is read in), the expression evaluates to true
- if the input fails (the wrong type of input is read in or end-of-file has been reached), the expression evaluates to false

thus, can have a while loop driven by an input statement

```
void ReadGrades(int grades[], int & numGrades)
// Results: reads grades and stores in array
//          numGrades is set to the # of grades
{
    ifstream myin;
    myin.open("grades.dat");

    numGrades = 0;

    int grade;
    while (myin >> grade) {
        grades[numGrades] = grade;
        numGrades++;
    }

    myin.close();
}
```

Improvement 2: guarding against OOB

what if you guessed wrong when setting the size of the array?

- suppose the user enters 101 grades
- what will happen?

must stop the loop before indexing out-of-bounds

```
void ReadGrades(int grades[], int & numGrades)
// Results: reads grades and stores in array
//          numGrades is set to the # of grades
{
    ifstream myin;
    myin.open("grades.dat");

    numGrades = 0;

    int grade;
    while (numGrades < MAX_GRADES && myin >> grade) {
        grades[numGrades] = grade;
        numGrades++;
    }
    myin.close();
}
```

Improvement 3: generalizing file names

we can generalize the program to read from an arbitrary file

- can prompt the user for the file name, read into a string
- note that the constructor for an ifstream requires a C-style string as argument (char array), not a C++ string object
- fortunately, there is a string member function that automatically converts a C++ string to a C-style string

```
void ReadGrades(int grades[], int & numGrades)
// Results: reads grades and stores in array
//          numGrades is set to the # of grades
{
    string filename;
    cout << "Enter the grades file name: ";
    cin >> filename;

    ifstream myin;
    myin.open( filename.c_str() );

    numGrades = 0;

    int grade;
    while (numGrades < MAX_GRADES && myin >> grade) {
        grades[numGrades] = grade;
        numGrades++;
    }
    myin.close();
}
```

Improvement 4: guarding against file errors

we can test the ifstream to be sure that the specified file was opened correctly

- the ifstream has a Boolean value associated with it
- if the file exists (and not past end-of-file), the ifstream evaluates to true
- otherwise, the ifstream evaluates to false
- must clear the input file stream between attempts to open (in order to reset true/false value)

```
void ReadGrades(int grades[], int & numGrades)
// Results: reads grades and stores in array
//          numGrades is set to the # of grades
{
    string filename;
    cout << "Enter the grades file name: ";
    cin >> filename;

    ifstream myin;
    myin.open( filename.c_str() );

    while (!myin) {
        cout << "File not found. Try again: ";
        cin >> filename;
        myin.clear();
        myin.open( filename.c_str() );
    }

    numGrades = 0;

    int grade;
    while (numGrades < MAX_GRADES && myin >> grade) {
        grades[numGrades] = grade;
        numGrades++;
    }

    myin.close();
}
```

in-class exercise

suppose you have been hired by the Weather Channel

- temperatures are automatically read at 5 minute intervals
- temperature readings are stored in a file (in order, with midnight first)
- at any given point in the day, you want to be able to determine the low and high temperatures so far

programming tools?

pseudo-code?

pseudo-code: no array version

must read in all temps

- keeping track of lowest and highest values seen so far
- when done reading all temps, then have lowest & highest

could structure the code lots of ways

- how should low & high be initialized?

```
PROMPT FOR & READ FILE NAME;

if (READ FIRST TEMP OK) {
  SET LOW TO TEMP;
  SET HIGH TO TEMP;

  while (READ ANOTHER OK){
    if (HIGHER THAN HIGH) {
      UPDATE HIGH;
    }
    else if (LOWER THAN LOW){
      UPDATE LOW;
    }
  }
  DISPLAY HIGH & LOW;
}
else {
  DISPLAY ERROR MESSAGE;
}
```

temp extremes (no array)

```
// temp.cpp
////////////////////////////////////

#include <iostream>
#include <fstream>
#include <string>
using namespace std;

int main()
{
    string tempFile;
    cout << "Enter the temperature file name: ";
    cin >> tempFile;

    ifstream ifstr;
    ifstr.open( tempFile.c_str() );

    while (!ifstr) {
        cout << "File not found. Try again: ";
        cin >> tempFile;
        ifstr.clear();
        ifstr.open( tempFile.c_str() );
    }
}
```

```
int temp;
if (ifstr >> temp) {
    int lowTemp = temp;
    int highTemp = temp;

    while (ifstr >> temp) {
        if (temp > highTemp) {
            highTemp = temp;
        }
        else if (temp < lowTemp) {
            lowTemp = temp;
        }
    }

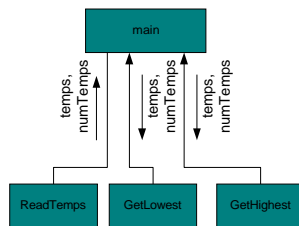
    cout << "Low temperature: " << lowTemp
        << endl;
    cout << "High temperature: " << highTemp
        << endl;
}
else {
    cout << "There were no temps!" << endl;
}

ifstr.close();
return 0;
}
```

pseudo-code: array version

can use a more top-down approach if we instead read in and store the temperature in an array

- not necessarily as efficient, but easier to understand and code
- tasks are separated, can be coded and tested independently



```
PROMPT FOR & READ FILE NAME;
```

```
READ TEMPS & STORE IN ARRAY;
```

```
if (ANY TEMPS WERE READ) {
    GET & DISPLAY LOW TEMP;
    GET & DISPLAY HIGH TEMP;
}
else {
    DISPLAY ERROR MESSAGE;
}
```

temp extremes (array version)

```
// temp.cpp
////////////////////////////////////
#include <iostream>
#include <fstream>
#include <string>
using namespace std;

const int MAX_TEMPS = 24*60;

void ReadTemps(int temps[], int & numTemps);
int GetLowest(const int temps[], int numTemps);
int GetHighest(const int temps[], int numTemps);

int main()
{
    int temps[MAX_TEMPS], numTemps;
    ReadGrades(temps, numTemps);

    cout << "The lowest temperature was: "
         << GetLowest(temps, numTemps) << endl;
    cout << "The highest temperature was: "
         << GetHighest(temps, numTemps) << endl;

    return 0;
}

////////////////////////////////////

void ReadTemps(int temps[], int & numTemps)
// Results: reads temps and stores in array
// numTemps is set to the # of temps
{
    string tempFile;
    cout << "Enter the temperature file name: ";
    cin >> tempFile;

    ifstream ifstr;
    ifstr.open( tempFile.c_str() );

    while (!ifstr) {
        cout << "File not found. Try again: ";
        cin >> tempFile;
        ifstr.clear();
        ifstr.open( tempFile.c_str() );
    }

    numTemps = 0;

    int temp;
    while (numTemps < MAX_TEMPS && ifstr >> temp){
        temps[numTemps] = temp;
        numTemps++;
    }

    ifstr.close();
}
```

array version (cont.)

once temperatures have
been stored in an array,

GetLowest and GetHighest
functions are:

- straightforward
- similar
- independent

```
int GetLowest(const int temps[], int numTemps)
// Assumes: temps contains numTemps temperatures
// numTemps > 0
// Returns: the lowest temp in the array
{
    int lowest = temps[0];
    for (int i = 1; i < numTemps; i++) {
        if (temps[i] < lowest) {
            lowest = temps[i];
        }
    }
    return lowest;
}

int GetHighest(const int temps[], int numTemps)
// Assumes: temps contains numTemps temperatures
// numTemps > 0
// Returns: the highest temp in the array
{
    int highest = temps[0];
    for (int i = 1; i < numTemps; i++) {
        if (temps[i] > highest) {
            highest = temps[i];
        }
    }
    return highest;
}
```

file output

similarly, it is possible to direct program output to a file

- must declare an output file stream (ofstream)
- open using a file name (same as with ifstream)
- write using <<
- close when done (same as with ifstream)

```
#include <fstream>                // loads definition of the output
                                   // file stream class (ofstream)
. . .

ofstream myout;                   // declares output file stream
myout.open("nums.out");           // opens the output file stream
                                   // using the file "nums.out"

for (int i = 0; i < 100; i++) {    // writes numbers to output file,
    myout << i << endl;           // one per line
}

myout.close();                   // closes the output file stream
```

updating the grade file

suppose we wanted to scale up the grades in the file

- can read the grades from a file
- write the updated grades back to that same file

```
PROMPT FOR & READ FILE NAME;

OPEN INPUT FILE STREAM;
READ GRADES & STORE IN ARRAY;
CLOSE INPUT FILE STREAM;

PROMPT FOR BONUS AMOUNT;
SCALE UP GRADES;

OPEN OUTPUT FILE STREAM;
WRITE UPDATED GRADES;
CLOSE OUTPUT FILE STREAM;
```

note: storing the grades in an array is necessary

a file can't be opened for input & output at the same time

updating the grade file

```
#include <iostream>
#include <fstream>
#include <string>
using namespace std;

const int MAX_GRADES = 100;

void ReadGrades(int grades[], int & numGrades,
                string & fname);
void ScaleUp(int grades[], int numGrades);
void WriteGrades(const int grades[],
                 int numGrades, string fname);

int main()
{
    int grades[MAX_GRADES], numGrades;
    ReadGrades(grades, numGrades);

    ScaleUp(grades, numGrades);

    WriteGrades(grades, numGrades, fileName);

    return 0;
}

////////////////////////////////////////////////////////////////
```

```
void ReadGrades(int grades[], int & numGrades,
                string & fname)
// Results: reads grades and stores in array
//          numGrades is set to the # of grades
//          fname is the name of the input file
{
    // AS BEFORE, EXCEPT FOR THE FILE NAME
}

void ScaleUp(int grades[], int numGrades)
// Assumes: grades contains numGrades grades
// Results: a bonus is added to each grade
{
    int bonus;
    cout << "How many bonus points are there? ";
    cin >> bonus;

    for(int i = 0; i < numGrades; i++) {
        grades[i] += bonus;
        if (grades[i] > 100) {
            grades[i] = 100;
        }
    }
}

void WriteGrades(const int grades[],
                 int numGrades, string fname)
// Assumes: grades contains numGrades grades
// Results: writes updated grades back to fname
{
    ofstream ofstr;
    ofstr.open(fname.c_str());

    for (int i = 0; i < numGrades; i++) {
        ofstr << grades[i] << endl;
    }
    ofstr.close();
}
```

copy cat?

would the following
code suffice to copy
the contents of a file?

i.e., does copy.txt
look exactly like the
input file?

```
// copy.cpp
////////////////////////////////////////////////////////////////

#include <iostream>
#include <fstream>
#include <string>
using namespace std;

int main()
{
    string infile;
    cout << "Enter the input file name: ";
    cin >> infile;

    ifstream ifstr;
    ifstr.open(infile.c_str());

    ofstream ofstr;
    ofstr.open("copy.txt");

    char ch;
    while (ifstr >> ch) {
        ofstr << ch;
    }

    ifstr.close();
    ofstr.close();

    return 0;
}
```

real copy cat

recall that >> ignores all whitespace

- thus, the previous program will copy all non-whitespace chars, but spacing will be lost

the `get` member function (from `<iostream>`) reads a character, including whitespace

- applied to an input stream
- one argument: a char

```
// copy.cpp
////////////////////////////////////

#include <iostream>
#include <fstream>
#include <string>
using namespace std;

int main()
{
    string infile;
    cout << "Enter the input file name: ";
    cin >> infile;

    ifstream ifstr;
    ifstr.open(infile.c_str());

    ofstream ofstr;
    ofstr.open("copy.txt");

    char ch;
    while (ifstr.get(ch)) {
        ofstr << ch;
    }

    ifstr.close();
    ofstr.close();

    return 0;
}
```

faster copy cat

the `getline` function (from `<iostream>`) reads an entire line of text, including whitespace

- two arguments: input stream and a string

note inconsistency:

- `get` is a member function
- `getline` is a stand-alone function

```
// copy.cpp
////////////////////////////////////

#include <iostream>
#include <fstream>
#include <string>
using namespace std;

int main()
{
    string infile;
    cout << "Enter the input file name: ";
    cin >> infile;

    ifstream ifstr;
    ifstr.open(infile.c_str());

    ofstream ofstr;
    ofstr.open("copy.txt");

    string line;
    while (getline(ifstr, line)) {
        ofstr << line << endl;
    }

    ifstr.close();
    ofstr.close();

    return 0;
}
```

file I/O summary

can read input from files

- declare an input file stream (`ifstream`), as defined in `<fstream>`
- open the input file stream using a file name (C-style string)
- read data using `>>`, same as with `cin`
- close the input file stream when done

can write output to files

- declare an output file stream (`ofstream`), as defined in `<fstream>`
- open the output file stream using a file name (C-style string)
- write data using `<<`, same as with `cout`
- close the output file stream when done

alternatives to `>>` exist that don't ignore whitespace

- `get` reads a character (including whitespace)
- `getline` reads an entire line of text (including whitespace)