Streams and File I/O

Streams

- A flow of characters
- Input stream
  - Flow into program
    - Can come from keyboard
    - Can come from file
- Output stream
  - Flow out of program
    - Can go to screen
    - Can go to file

Streams Usage

- We’ve used streams already
  - cin
    - Input stream object connected to keyboard
  - cout
    - Output stream object connected to screen
- Can define other streams
  - To or from files
  - Used similarly as cin, cout

Streams Usage Like cin, cout

- Consider:
  - Given program defines stream inStream that comes from some file:

    int theNumber;
    inStream >> theNumber;

    - Reads value from stream, assigned to *theNumber*
  - Program defines stream outStream that goes to some file

    outStream << “theNumber is “ << theNumber;

    - Writes value to stream, which goes to file

Files

- We’ll use text files
- Reading from file
  - When program takes input
- Writing to file
  - When program sends output
- Start at beginning of file to end
  - Other methods available
  - We’ll discuss this simple text file access here

File Connection

- Must first connect *file* to *stream object*
- For input:
  - File → ifstream object
- For output:
  - File → ofstream object
- Classes ifstream and ofstream
  - Defined in library <fstream>
  - Named in std namespace

File I/O Libraries

- To allow both file input and output in your program:

```
#include <fstream>
using namespace std;
        OR
#include <fstream>
using std::ifstream;
using std::ofstream;
```

Alternative Syntax for File Opens

- Can specify filename at declaration
  - Passed as argument to constructor
- ifstream inStream;
  inStream.open(“infile.txt”);

  EQUIVALENT TO:

  ifstream inStream(“infile.txt”);

Declaring Streams

- Stream must be declared like any other class variable:

  ```
  ifstream inStream;
  ofstream outStream;
  ```

- Must then ‘connect’ to file:

  ```
  inStream.open(“infile.txt”);
  ```

  - Called ‘opening the file’
  - Uses member function *open*
  - Can specify complete pathname

Streams Usage

- Once declared → use normally!

```
int oneNumber, anotherNumber;
inStream >> oneNumber >> anotherNumber;
```

- Output stream similar:

```
ofstream outStream;
outStream.open(“outfile.txt”);
outStream  << “oneNumber = “ << oneNumber
           << “ anotherNumber = “
           << anotherNumber;
```

  - Sends items to output file

Closing Files

- Files should be closed
  - When program completed getting input or sending output
  - Disconnects stream from file
  - In action:

    ```
    inStream.close();
    outStream.close();
    ```

    - Note no arguments
- Files automatically close when program ends

File Flush

- Output often ‘buffered’
  - Temporarily stored before written to file
  - Written in ‘groups’
- Occasionally might need to force writing: outStream.flush();
  - Member function *flush*, for all output streams
  - All buffered output is physically written
- Closing file automatically calls flush()

File Example:

Appending to a File

- Standard open operation begins with empty file
  - Even if file exists → contents lost
- Open for append:

```
ofstream outStream;
outStream.open(“important.txt”, ios::app);
```

  - If file doesn’t exist → creates it
  - If file exists → appends to end
  - 2nd argument is class *ios* defined constant
    - In <iostream> library, std namespace

Checking File Open Success

- File opens could fail
  - If input file doesn’t exist
  - No write permissions to output file
  - Unexpected results
- Member function fail()
  - Place call to fail() to check stream operation success

```
inStream.open(“stuff.txt”);
if (inStream.fail())
{
        cout << “File open failed.\n”;
        exit(1);
}
```

Functions for manipulation of file pointers

- Member functions work same:
  - seekg() : moves get pointer to a specified location.
  - seekp() : moves put pointer to a specified location.
  - tellg() : gives the current position of the get pointer.
  - tellp() : gives the current position of the put pointer.

Ex:

```
Infile.seekg(10);      int i=infile.tellg();
Outfile.seekp(10);     int i=outfile.tellg();
```

- Infile.seekg(offset,refposition);
- outfile.seekp(offset,refposition);

Ios::beg : start of the file

Ios::cur : current position of the pointer

Ios::end : end of the file

EX:

Infile.seekg(m,ios::cur);

outfile.seekp(m,ios::cur);

Random Access to Files

- Sequential Access
  - Most commonly used
- Random Access
  - Rapid access to records
  - Perhaps very large database
  - Access ‘randomly’ to any part of file
  - Use fstream objects
    - input and output

Checking End of File

- Use loop to process file until end
  - Typical approach
- Two ways to test for end of file
  - Member function eof()

```
inStream.get(next);
while (!inStream.eof())
{
        cout << next;
        inStream.get(next);
}
```

    - Reads each character until file ends
    - eof() member function returns bool

End of File Check with Read

- Second method
  - read operation returns bool value! (inStream >> next)
    - Expression returns true if read successful
    - Returns false if attempt to read beyond end of file
  - In action:

```
double next, sum = 0;
while (inStream >> next)
        sum = sum + next;
cout << “the sum is “ << sum << endl;
```

Sequential input and output operations

- put() and get() functions for characters.
- Write() and read() functions for various data types.

Char ch;

outfile.put(ch);

 // ch variable content will be written into the outfile stream.

Infile.put(ch);

// character which is pointed by the infile stream is place into

The ch variable

File1 Example: