
NJIT
THE EDGE IN KNOWLEDGE

CS 280
Programming Language Concepts

Spring 2023

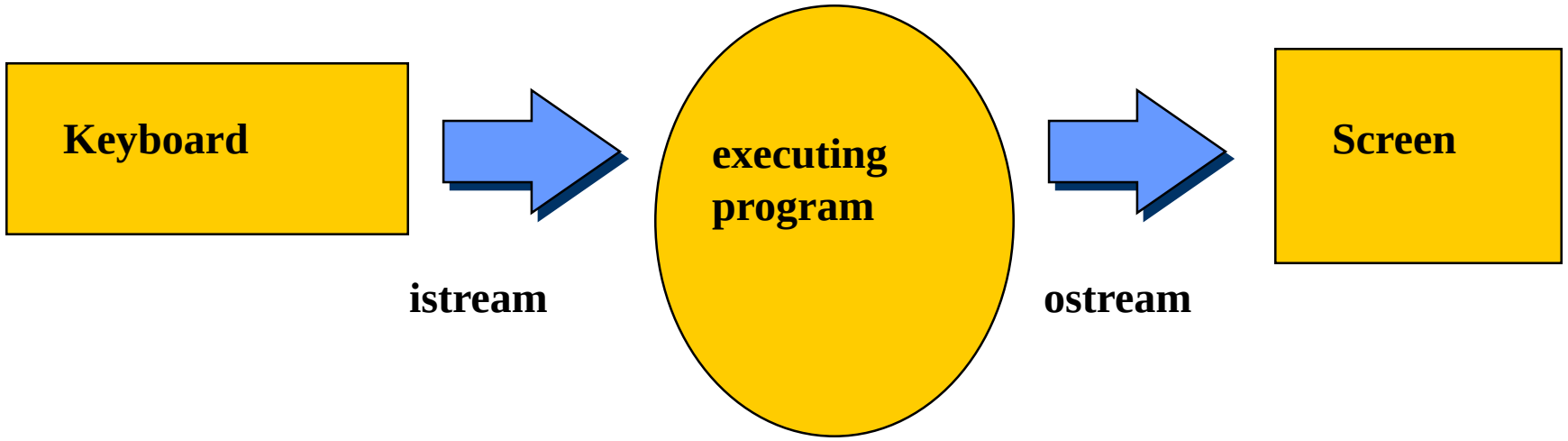
Streams and File I/O

Topics

- C++ Input/Output
- String Overview
- Files
- Examples
- Recitation Assignment 2

C++ Input/Output

- **No built-in I/O in C++**
- **A library provides input stream and output stream**



>> Operator

>> is called the input or extraction operator

>> is a binary operator

>> is left associative

```
int age, weight;
```

Expression	**Has value**
`cin >> age`	**`cin`**

Statement

```
cin >> age >> weight;
```

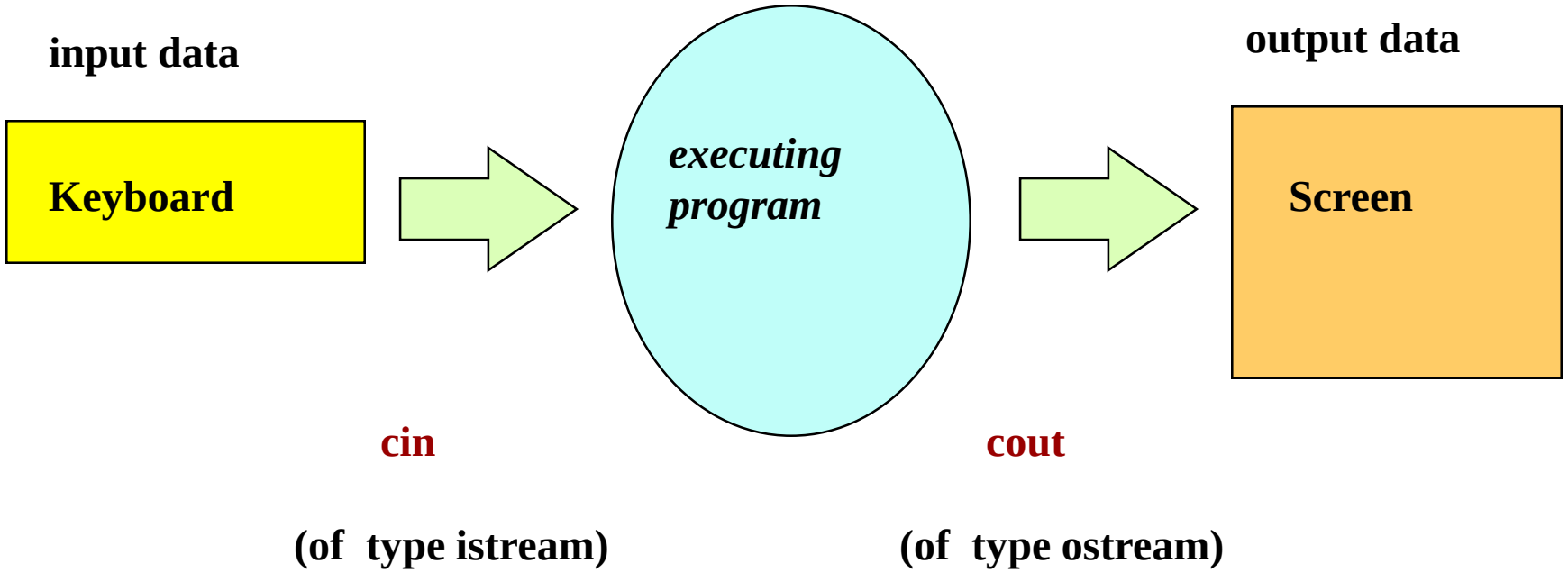
- Reading an integer

Extraction Operator (>>)

- "skips over" (actually reads but does not store anywhere) leading **white space characters** as it reads your data from the input stream (either keyboard or disk file).
  - Returns **`0`** when EOF encountered
    - Otherwise, returns reference to object
    - **`cin >> grade`**
  - State bits set if errors occur
- End-of-file
  - Indicates end of input
    - *ctrl-z* on IBM-PCs/Windows
    - *ctrl-d* on UNIX and Macs
  - **`cin.eof()`**
    - Returns **`1`** (**`true`**) if EOF has occurred

Keyboard and Screen I/O

#include <iostream>



Reading Data Using get() Function

- The `get()` function can be used to read a single character.
- `get()` obtains the very next character from the input stream without skipping any leading whitespace characters
  - `cin.get()`

    Returns one character from stream (even whitespace)
    Returns **`EOF`** if end-of-file encountered
- The **ignore()** function is used to skip (read and discard) characters in the input stream. The call
  - `cin.ignore(howMany, whatChar);`
  - will skip over up to **howMany** characters or until **whatChar** has been read, whichever comes first

- The **put()** function can be used to display a single character on the output stream.

Reading Data Using get() Function

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

int main()
{
    int character; // use int, because char cannot represent EOF

    // prompt user to enter line of text
    cout << "Before input, cin.eof() is " << cin.eof() << endl
         << "Enter a sentence followed by end-of-file:" << endl;

    // use get to read each character; use put to display it
    while ( ( character = cin.get() ) != EOF )
        cout.put( character );

    // display end-of-file character
    // use int, because char cannot represent EOF
    cout << "\nEOF in this system is: "<< character << endl;
    cout << "After input, cin.eof() is " << cin.eof() << endl;
    return 0;
}
```

Reading Data Using get() Function

- You can test a character if it is a decimal digit, alphabetic, alphanumeric, or a space by including <cctype> header and using the functions:
  - `isdigit()`
  - `isalpha()`
  - `isalnu()`
  - `isspace()`

```
Before input, cin.eof() is 0
Enter a sentence followed by end-of-file:
testing the eof on this system.
testing the eof on this system.
^Z

EOF in this system is: -1
After input, cin.eof() is 1
```

Strings Overview: Testing Digits Example

```
#include <iostream>
#include <string>
#include <cctype>
//isdigit example
using namespace std;
int main(){
   string str = "hds24w7ds", num = "";
   //instead use getline to read a line from keyboard
   int value, i = 0;
   while (i < str.length()){
     if (isdigit(str[i]))
      num += str[i]; i++;
   }
   cout << "The digit characters form the number: "<< num;
   return 0;
}
```

getline() Function

- Because the extraction operator stops reading at the first trailing whitespace, >> cannot be used to input a string with blanks in it.
- Use the `getline` function with 2 arguments to overcome this obstacle.
  - `getline(inFileStream, str)`
- First argument is an input stream variable, and second argument is a string variable
- **Example**

```
string    message;
getline(cin,  message);
```

getline() Function

- `getline` reads successive characters(including blanks) into the string, and stops when it reaches the newline character '\n'
- The newline is consumed by `getline`, but is not stored into the string variable
- Notice the difference between the newline marker on Windows and Linux.
  - On Windows it is the combination of **carriage return (ASCII 0x0d or \r)** and a newline(\n), also referred to as CR/LF.
  - On Linux it is just a newline (\n)

Strings Overview

- **`string`** features
  - Not necessarily **`null`** terminated
  - **`length`** member function: **`s1.length()`**
  - Use **`[]`** to access individual characters: **`s1[0]`**
    - **`0`** to **`length-1`**
  - **`string`** not a pointer
  - Many member functions take start position and length
    - If length argument too large, max chosen
  - Stream extraction
    - **`cin >> stringObject;`**
    - **`getline( cin, s)`**
      - Delimited by newline

Strings Overview

- Assignment
  - `s2 = s1;`
    - Makes a separate copy
  - `s2.assign(s1);`
    - Same as `s2 = s1;`
  - `myString.assign(s, start, N);`
    - Copies `N` characters from `s`, beginning at index `start`
  - Individual characters
    - `s2[0] = s3[2];`
- Comparing strings
  - Overloaded operators
    - `==`, `!=`, `<`, `>`, `<=` and `>=`
    - Return `bool`

Strings Overview

- Range checking
  - **`s3.at( index );`**
    - Returns character at **`index`**
    - Can throw **`out_of_range`** exception
  - **`[]`** has no range checking
- Concatenation
  - **`s3.append( "pet" );`**
  - **`s3 += "pet";`**
    - Both add **`"pet"`** to end of **`s3`**
  - **`s3.append( s1, start, N );`**
    - Appends **`N`** characters from **`s1`**, beginning at index **`start`**

Strings Overview

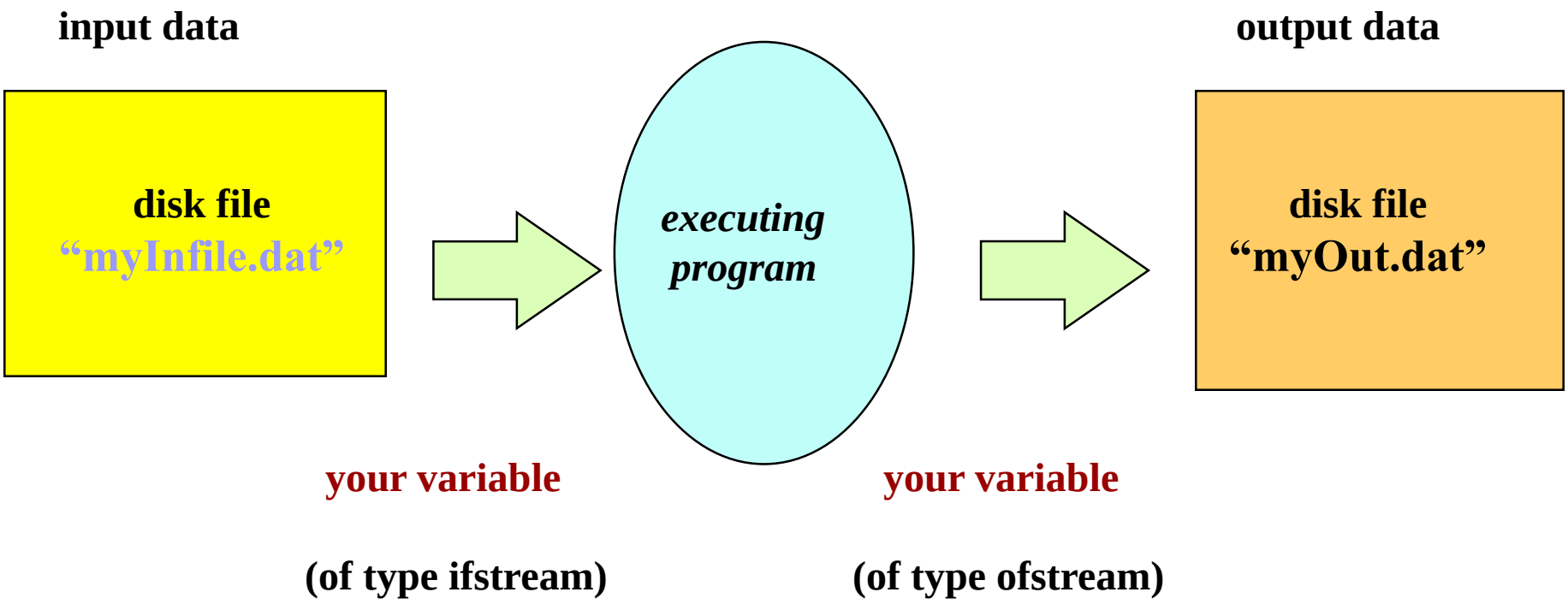
- Substring
  - **myStr = "Concepts of Programming Languages"**
  - **str = myStr.substr(12, 7);**
    - **returns the string "Program"**
  - **str = myStr.substr(12);**
    - **returns the string "Programming Languages"**
  - **Str = myStr.substr(10, 0);**
    - **returns null string ""**
- Find: Searches a string or the first occurrence of a particular substring:
  - If found, index returned. If not found, **string::npos** returned
    - Public static constant in class string
  - **s1.find(s2): s2 could be a literal string, or a string expression**
  - **s1.rfind(s2):** Searches s1 string right-to-left

Strings Overview

- Conversion functions to C-Style char * Strings
  - **`string`**s not necessarily null-terminated
  - **`s1.copy( ptr, N, index )`**
    - Copies **`N`** characters into the array **`ptr`**
    - Starts at location **`index`**
    - Need to null terminate
  - **`s1.c_str()`**
    - Returns **`const char * (constant char pointer)`**
    - Null terminated

Disk Files for I/O

#include <fstream>



Disk I/O

To use disk I/O

- **Access #include <fstream>**
- **Choose valid identifiers for your filestreams and declare them**
- **Open the files and associate them with disk names**
- **Use your filestream identifiers in your I/O statements (using >> and << , manipulators, get, ignore)**
- **Close the files**

Disk I/O Statements

```
#include <fstream>

ifstream   myInfile;          // Declarations
ofstream  myOutfile;

myInfile.open("myIn.dat"); // Open files
myOutfile.open("myOut.dat");

myInfile.close();             // Close files
myOutfile.close();
```

Opening a File

Opening a file

- **Associates the C++ identifier for your file with the physical(disk) name for the file**
  - **If the input file does not exist on disk, open is not successful**
  - **If the output file does not exist on disk, a new file with that name is created**
  - **If the output file already exists, it is erased**
- **Places a file reading marker at the very beginning of the file, pointing to the first character in the file**

Stream Fail State

- **When a stream enters the fail state,**
  - **Further I/O operations using that stream have no effect at all.**
  - **The computer does not automatically halt the program or give any error message.**

- **Possible reasons for entering fail state include**
  - **Invalid input data (often the wrong type).**
  - **Opening an input file that doesn't exist**
  - **"End of file" is reached**
  - **Opening an output file on a disk that is already full or is write-protected.**

Stream Fail State

- You need to check for errors!
- **Stream methods for checking errors**
  - `good()` is true if there are no errors.
  - `eof()` is true if the end of file was reached.
  - `fail()` is true if there was a logical error or a read/write error on the stream.
  - `bad()` is true if there is a read/write error on the stream.

Run Time File Name Entry

```
#include <string>
// Contains conversion function c_str

ifstream  inFile;
string    fileName;

cout << "Enter input file name: " << endl; // Prompt
cin    >>  fileName;

// Convert string fileName to a C string type
inFile.open(fileName.c_str());
```

Example 1: Reading from a File

```
#include <iostream>
#include <fstream>
#include <cstdlib>  // exit prototype
#include <iomanip>
using namespace std;
int main(){
  ifstream inClientFile( "clients.dat", ios::in );
  // exit program if unable to create file
  if ( !inClientFile ) {  // overloaded ! Operator
     cerr << "File could not be opened" << endl;
     exit( 1 );
   }
   cout << left << setw( 10 ) << "Account" << setw( 13 )
        << "Name" << "Balance" << endl << fixed << showpoint;
   int account;
   char name[ 30 ];
   double balance;
```

Example 1

```
  // display each record in file
  while ( inClientFile >> account >> name >> balance ){
        cout << left << setw( 10 ) << account << setw( 13 )
        << name << setw( 8 ) << setprecision( 2 ) << right
        << balance << endl;
   } // end while

   return 0;  // ifstream destructor closes file
} // end main
```

Example 2: Writing to a File

```
#include <iostream>
#include <fstream>
#include <cstdlib>  // exit prototype
using namespace std;

int main() {
   ofstream outClientFile( "clients.dat", ios::out );

   // exit program if unable to create file
   if ( !outClientFile ) {  // overloaded ! operator
      cerr << "File could not be opened" << endl;
      exit( 1 );
   }
```

Example 2

```
   cout << "Enter the account, name, and balance." << endl
        << "Enter end-of-file to end input.\n? ";
   int account;
   char name[30];
   double balance;

   while ( cin >> account >> name >> balance ) {
     outClientFile << account << ' ' << name
        << ' ' << balance << endl;
     cout << "? ";
     } // end while
   return 0;  // ofstream destructor closes file
} // end main
```

Recitation Assignment 2
Counting Lines

- Write a C++ program that acts like a simple counting tool for collecting information from textual files of documents prepared for a simple scripting language interpreter. An input file for the simple scripting interpreter includes two types of data, commented lines and scripting code lines. A commented line is recognized by either the two characters “##” or “//” at the beginning of the line, which would be skipped by the scripting interpreter. All other non-commented lines are considered as part of the scripting code, including any blank lines with whitespace.

Recitation Assignment 2
Counting Lines

- The program for the simple counting tool should read lines from a file until the end of file. The program should prompt the user for the file name to read from. The program should open the file for reading, and if the file cannot be opened, it should print the message "File cannot be opened ", followed by the filename, and exit.
- After reading the contents of the input file, the program should print out the total number of lines, the number of commented lines, the maximum length of commented lines, the maximum length of non-commented line, the commented line of maximum length delimited by double quotes, and the non-commented line of maximum length delimited by double quotes.

Recitation Assignment 2

Notes:

- **The example assumes that the file name is entered from the keyboard.**
- There are 13 lines in this input file.
- The screen pointer is at a new line after displaying the results.
- You have to apply the same format in order to have exact match.

Line number	File contents
1	run prog1
2	//
3	delete
4	### execute next command
5	delete myfile
6	
7	
8	copy file1 file2
9	
10	// welcome to cs 280
11	print myfile
12	
13	// terminate script
End of File marker →	

Entered file name →

```
Enter the name of a file to read from:
infile2.txt

Total Number of Lines: 13
Number of commented lines: 4
Maximum Length of commented lines: 24
Maximum Length of non-commented lines: 17
Commented line of maximum length: "### execute next command"
Non-commented line of maximum length: "run        prog1 "
```

Screen pointer →


NJIT
THE EDGE IN KNOWLEDGE