Chapter 4

Program Input and the Software Design Process

Getting Data into Programs

- An advantage of programs is that they can be run many times with different data

- Programs must have a way of reading (inputting) the data

- An input stream is an endless sequence of characters coming into a program

- **To use stream I/O, the preprocessor directive
  
  \#include <iostream>

  is required.

  - It makes the declarations for cin (standard input) and cout (standard output)
cin and extraction operator (>>)

- *cout* is associated with the standard output device – monitor - and it uses the **INSERTION** operator `<<`

- *cin* is associated with the standard input device – keyboard - and it uses the **EXTRACTION** operator `>>`

- The extraction operator requires two operands.
  - The left hand side is a stream expression - *cin* for now.
  - The right hand side is the variable which will store the input data
  - The right hand side can only be a variable (*cin >> variable*)

  cin and extraction operator (>>)

- The **extraction operator** skips any leading **white space characters** (such as spaces, tabs and new-line characters)
  - For *char* variables, the input stops after 1 character is read
  - For other DataTypes, **input stops at a trailing whitespace character or the first character inappropriate** for the DataType being read(extracted)

- **SYNTAX TEMPLATE:**  
  ```
  cin >> Variable >> Variable ... ;
  ```
Reading Marker and Newline Character

- The *reading marker* keeps track of the point in the input stream where the computer should continue reading. It indicates the next character that should be read.

- Each input line has an *invisible end-of-line character* that tells the computer where one line ends and the next line begins.

- New line characters are inserted when you press “return” or “enter” on the keyboard.

- \n is used to represent the new line character, and it CAN BE stored in a char variable.

### Extraction Operator (>>) Examples

```cpp
int i, j; float x; char ch; string text;

Statement                  Input Stream  Contents
---                        ---------      -------
cin >> i >> j >> x;         10 5          i = 10, j = 5
                             waiting for x

cin >> i >> ch >> x          10\n                             i = 10, ch = ‘B’,
                             B\n                             x = 16.9
                             16.9\n                             (same for 10B16.9\n)
```

<table>
<thead>
<tr>
<th>Statement</th>
<th>Input Stream</th>
<th>Contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>cin &gt;&gt; i;</td>
<td>22 7 A Name 10.5\n</td>
<td>i contains 22</td>
</tr>
<tr>
<td>cin &gt;&gt; j;</td>
<td>22 7 A Name 10.5\n</td>
<td>j contains 7</td>
</tr>
<tr>
<td>cin &gt;&gt; ch;</td>
<td>22 7 A Name 10.5\n</td>
<td>ch contains A</td>
</tr>
<tr>
<td>cin &gt;&gt; text;</td>
<td>22 7 A Name 10.5\n</td>
<td>text contains Name</td>
</tr>
<tr>
<td>cin &gt;&gt; x;</td>
<td>22 7 A Name 10.5\n</td>
<td>x contains 10.5</td>
</tr>
</tbody>
</table>

*See Program Ch_04_01.cpp for examples*
Reading Characters with the \texttt{get} Function

- Use the \texttt{get} function to input the next character in the input stream only without skipping any characters (reads \texttt{chars} only)

- \textbf{SYNTAX TEMPLATE:} \texttt{get function call}
  
  \begin{verbatim}
  input_file_stream_name.get(char); \hspace{1em} (i.e. cin.get(CharVariable); )
  \end{verbatim}

- The function \texttt{get} is a \texttt{void} function - it does not return a value.

- \textbf{The argument to the \texttt{get} function must be a \texttt{char} variable}

  \begin{verbatim}
  for the input stream: A\hspace{0.6em}B\hspace{0.6em}C\hspace{0.6em}(\square = \text{blank})
  char c1,c2,c3,c4,c5;
  cin.get(c1); cin.get(c2); cin.get(c3);
  cin.get(c4); cin.get(c5);
  \end{verbatim}

  The char variables contain the following values:
  \begin{verbatim}
  c1 = 'A', c2 = '', c3 = 'B', c4 = '', c5 = 'C'
  \end{verbatim}

Reading An Entire Line of \texttt{string} Words

- Reading \texttt{string} variables - the extraction operator (\texttt{\gg}) skips any leading whitespace characters, then reads successive characters stopping at the first trailing whitespace character.

- The \texttt{getline} function reads an entire line until the character specifying the end of the line is encountered. By default, the ending character is the newline character.

- \texttt{getline} \texttt{does not skip white space characters}, and it continues until it reaches the end-of-line character specified - which is consumed (removed from the input stream) but not stored in the \texttt{string} variable.

- \textbf{SYNTAX TEMPLATE:} \texttt{getline function}

  \begin{verbatim}
  getline(Input_file_stream_name, string \_char);
  \end{verbatim}

  Example: \texttt{getline(cin, inputStr, '(':});
Reading An Entire Line of string Words

- **getline** requires two arguments (the third one is optional) –
  - the first one is an input stream variable (i.e. `cin`),
  - the second is the `string` variable that will hold the line read, and
  - the third is the character that terminates the line:

  ```cpp
  getline(cin, question,'\n');
  ```

  Since the newline character is the default line termination character, the third argument ‘\n’ above is not required

  See Programs Ch_04_02.cpp and Ch_04_03.cpp for examples

---

The **ignore** Function

- The **ignore** function is used to skip characters in the input stream
- **SYNTAX TEMPLATE:** `ignore function`
  ```cpp
  Input_file_stream_name.ignore(int expression, char);
  ```
  Example: `cin.ignore(int expression, char);`

- This line tells the computer to
  - skip the next `int expression` input characters OR
  - to skip characters until the `char` specified IS READ (which removes it from the input stream)- whichever comes first.

  Examples:
  ```cpp
  cin.ignore(20, '\n');
  cin.ignore(5*intVar,':');
  ```
The ignore Function

• The **ignore** function is used to skip characters in the input stream

• **SYNTAX TEMPLATE:**  
  ```
  Input_file_stream_name.ignore(int expression);
  ```
  Example: `cin.ignore(int expression);`
  • This line tells the computer to skip the next `int expression` input characters
  • Examples:
    ```
    cin.ignore(20);
    cin.ignore(5*intVar);
    ```
  • The following ignore function call will skip 10 characters (integer value of the new line character). It will not skip all characters up to the new line character
    • `cin.ignore(‘\n’);` // skips 10 characters

Input/Output

• **Interactive I/O** - occurs when a program prompts the user for information, and the user enters the information. Typically using `cout` and `cin` (or `cin.get` or `getline`)
  ```
  string name;  int age;
  cout << "Enter your first name: ";
  cin >> name;
  cout << "Enter your age: ";
  cin >> age;
  cout << name << "is " << age << " years old\n";
  ```

• **Non-interactive I/O** - occurs when a program does not prompt for information and it reads the necessary information from files
File Input/Output (I/O)

- Using files makes it easier to enter a large amount of data
- Writing to a file preserves the output from a program

**Using files requires 4 actions:**

1. Use the `#include <fstream>` preprocessor directive
2. Declare the input file stream variables (`ifstream InFile;`)
3. Declare the output file stream variables (`ofstream OutFile;`)
4. Prepare each file using the `open` function (`InFile.open(file_name)`)
5. Specify the name of the file stream in each input or output statement (`InFile >> some_int;`)

**#include <fstream> defines 2 DataTypes:**

- The `ifstream` DataType represents the stream of characters from an input file
- The `ofstream` DataType represents the stream of characters going to an output file

File Input/Output (I/O) (continued)

- All operations used with the standard streams `cin/cout` also apply to stream variables of the `ifstream/ofstream` DataTypes

**Declaring file stream variables:**

- `ifstream InFile;` // `ifstream` used to declare input files only
- `ofstream OutFile;` // `ofstream` used to declare output files only
- Note: You cannot simultaneously write to and read from the same file!

**Opening files** - Need to associate a filename with the declared file stream variables. This association is done with the `open` function

- `InFile.open(“Input.dat”);`
- `OutFile.open(“Output.dat”);`
- Must open a file before using any input or output statements that refer to that file
File Input/Output (I/O) (continued)

- **In C++ the file name used with the `open` function MUST BE A LITERAL STRING - NOT A string VARIABLE.**
- **It is usually more desirable to use a string variable with the `open` function. Use the string Data Type value returning function `c_str()` to return the literal equivalent of a string variable.**
- Example showing string conversion to a literal value to be used with the open function
  ```cpp
  ifstream InFile;
  string filename;
  cout << "enter the name of the input file: ";
  cin >> filename;
  InFile.open(filename.c_str());
  ```
  See Programs Ch_04_04.cpp and Ch_04_05.cpp for examples

File Input/Output (I/O) (continued)

- **Instead of using `cin` or `cout`, use the declared file stream variables to read/write from/to a file**
  ```cpp
  // read values for test and name from the input file associated with
  // the `ifstream` variable `InFile`
  InFile >> test >> name;
  // Print the value of test to the output file associated with the
  // `ofstream` variable `OutFile`
  OutFile << "Test name is: " << test;
  ```
- **There is a close function associated with `ifstream` and `ofstream`**
  - All open files need to be closed to break the connection between the stream variable and the disk file.
  - Closing a file open for writing empties the buffer into the file
  ```cpp
  InFile.close(); OutFile.close(); // examples
  ```
More File I/O

• More than one file stream can be open for input and/or output
• What if a mistake is made and cin or cout is used instead of the declared file stream variables?
  • cout just prints the information to the screen - easily discovered
  • cin is a little worse. The computer waits for an input with no prompt telling the user what is happening.
• Input Failure - Invalid data is the most common reason. Invalid data is mismatched data with DataType declarations of variables (trying to read a float and input a string, trying to read an int and input a float, etc)
  • When an Input stream enters the fail state mode
    • all future I/O operations using that input stream are null
    • does not halt the program or give any error message
  • May result in improper/incomplete reading of the input stream

Invalid Data Input

• Suppose we want to read values for a float and two integer variables.
  • First consider an integer, float and integer pattern:
    int i,j; float x;
    cin >> i >> x >> j; input stream: 12 1.5 15
    then i contains 12, x contains 1.5 and j contains 15 as we want
  • for an input stream of: 121.5 15 (no space between 12 and 1.5)
    then i contains 121, x contains .5 and j contains 15
  • There is no fail state mode here, just incorrect data entry
  • what about an input stream of 12.5 15 16.1 for an integer, integer, float input pattern
    cin >> i >> j >> x;
    Fail state mode is entered, i contains 12, j is not read in since the first character it sees is a decimal point, and x is not read due to the fail state mode. (x and j keep their previous values) Also, all future cin statements are ignored.

See Program Ch_04_06.cpp for example