Input/Output (I/O)

- I/O streams
- Interactive I/O
- cin >>
- cout <<
- get
- getline
Input and Output

• For computation to be useful, it must:
  – operate on data supplied to the program (input)
  – present results to user or other program (output)

• I/O operations act on streams
  – Infinite (conceptually) sequences of data elements
  – Standard I/O streams comprise ASCII characters
  – All I/O is in essence, done one character at a time
Standard I/O Streams

- Used for interactive I/O
- `cout`
  - standard output stream
  - typically associated with console window on primary display device (monitor)
- `cin`
  - standard input stream
  - input received from keyboard
  - typically “echoed” in console window
Standard I/O Libraries

• iostream
  – Stream data types: istream, ostream
  – Standard I/O stream objects: cin, cout
  – Manipulators: endl
#include <iostream>

```cpp
#include <iostream>

class Keyboard {
public:
  void execute() {
    // Execute input data
  }
}

class Screen {
public:
  void display() {
    // Display output data
  }
}

int main() {
  Keyboard keyboard;
  screen.display();
  return 0;
}
```

Interactive I/O

- Input data: Keyboard
- Executing program
- Output data: Screen
Example Output

cout << "Hello there world!";
Example Output

cout << "Hello there world!";

Hello there world!
Example Output

cout << "Hello there world!";

Hello there world!

cout << "Hello there world!"
   << "C++ is amusing.";
Example Output

cout << "Hello there world!";

Hello there world!

cout << "Hello there world!"
    << "C++ is amusing.";

Hello there world!C++ is amusing.
Example Output

cout << "Hello there world!";

Hello there world!

cout << "Hello there world!"
  << "C++ is amusing.";

Hello there world!C++ is amusing.

cout << "Hello there world!"  << endl
  << "C++ is amusing.";
Example Output

cout << "Hello there world!";

Hello there world!

cout << "Hello there world!"
    << "C++ is amusing.";

Hello there world! C++ is amusing.

cout << "Hello there world!"  << endl
    << "C++ is amusing.";

Hello there world!
C++ is amusing.
`endl` is an *output manipulator* generates a line feed when put to stream
Example Output (2)

cout << "Hello there!" << end << endl
  << "Bye";
Example Output

```cpp
cout << "Hello there!" << endl;
    << "Bye";
```

prints

Hello there!

Bye
Special Output Characters

\n new line
\t tab
\b backspace
\' single quote
\" double quote
\\ backslash
Example Output (3)

- `cout << "Hello \t and \n goodbye!" ;`

  tab  end of line

Hello  and
goodbye!
Interactive Input

- Write a prompt
  friendly, informative

- remember to put something to separate the prompt from the input
  - a space
  - :

- Read value(s)
  user types data at keyboard
Interactive Input: Example

```cpp
int num;
char response;

cout << "Enter a number: ":
cin >> num;

cout << "Enter Y or N: ":
cin >> response;
```

leaves a space
Contents of Output Window

Enter a number: 17<return>

Enter Y or N: Y<return>

the program will not process the input until the return key is struck

Memory

17  Y
num  response
Useful Experiment

• take this correctly working program
  – run it several times
  – experiment entering various types of invalid data
  – observe what happens
  – you can learn a lot this way
Enter a number, then Y or N

Enter NaN number, then Y or N

Enter 17

same line  \( == \) blank

Enter

2nd line

y

3rd line
Interactive Input: Contents of Output Window

Enter a number, then Y or N

- Enter 17

same line \[==\] blank

- Enter a blank line

2\textsuperscript{nd} line

- Enter Y

3\textsuperscript{rd} line

num $\leftarrow$ 17

response $\leftarrow$ ‘y’
Whitespace and >>

- Leading white space (blanks, TABs, and newlines) skipped over when reading with the >> operator

The newline character is created by hitting Enter or Return at the keyboard, or by using the manipulator endl or “\n” in a program.
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)
At keyboard you type:
A[space]B[space]C[Enter]

char first;
char middle;
char last;

NOTE: A file reading marker is left pointing to the newline character after the ‘C’ in the input stream.
At keyboard you type:
A[space]B[space]C[Enter]

char first;
char middle;
char last;

cin >> first;
cin >> middle;
cin >> last;

NOTE: A file reading marker is left pointing to the newline character after the ‘C’ in the input stream.
At keyboard you type:

```
[space]25[space]J[space]2[Enter]
```

```c
int age;
char initial;
float bill;
```

```c
age
initial
bill
```

```c
cin >> age;
```

```c
age
```

```c
cin >> initial;
```

```c
initial
```

```c
cin >> bill;
```

```c
bill
```

**NOTE:** A file reading marker is left pointing to the newline character after the 2 in the input stream.
At keyboard you type:

\[ \text{space}25\text{space}J\text{space}2\text{Enter} \]

```cpp
int age; char initial; float bill;
cin >> age; cin >> initial; cin >> bill;
```

NOTE: A file reading marker is left pointing to the newline character after the 2 in the input stream.
Another example using `>>`

<table>
<thead>
<tr>
<th>STATEMENTS</th>
<th>CONTENTS</th>
<th>MARKER POSITION</th>
</tr>
</thead>
<tbody>
<tr>
<td>int i ;</td>
<td></td>
<td></td>
</tr>
<tr>
<td>char ch ;</td>
<td></td>
<td></td>
</tr>
<tr>
<td>float x ;</td>
<td></td>
<td></td>
</tr>
<tr>
<td>cin &gt;&gt; i ;</td>
<td></td>
<td>25 A\n16.9\n</td>
</tr>
<tr>
<td>cin &gt;&gt; ch ;</td>
<td></td>
<td></td>
</tr>
<tr>
<td>cin &gt;&gt; x ;</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Another example using `>>`

NOTE: shows the location of the file reading marker

<table>
<thead>
<tr>
<th>STATEMENTS</th>
<th>CONTENTS</th>
<th>MARKER POSITION</th>
</tr>
</thead>
<tbody>
<tr>
<td>int i ;</td>
<td></td>
<td></td>
</tr>
<tr>
<td>char ch ;</td>
<td></td>
<td></td>
</tr>
<tr>
<td>float x ;</td>
<td>i</td>
<td>25 A\n16.9\n</td>
</tr>
<tr>
<td>cin &gt;&gt; i ;</td>
<td></td>
<td></td>
</tr>
<tr>
<td>cin &gt;&gt; ch ;</td>
<td>16.9\n</td>
<td></td>
</tr>
<tr>
<td>cin &gt;&gt; x ;</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Another example using `>>`

<table>
<thead>
<tr>
<th>STATEMENTS</th>
<th>CONTENTS</th>
<th>MARKER POSITION</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>int i;</code></td>
<td><code>25</code></td>
<td><code>25 A\n</code></td>
</tr>
<tr>
<td><code>char ch;</code></td>
<td><code>A</code></td>
<td></td>
</tr>
<tr>
<td><code>float x;</code></td>
<td><code>16.9</code></td>
<td><code>16.9\n</code></td>
</tr>
<tr>
<td><code>cin &gt;&gt; i;</code></td>
<td><code>i</code></td>
<td><code>A\n</code></td>
</tr>
<tr>
<td></td>
<td><code>ch</code></td>
<td><code>16.9\n</code></td>
</tr>
<tr>
<td></td>
<td><code>x</code></td>
<td><code>16.9\n</code></td>
</tr>
<tr>
<td><code>cin &gt;&gt; ch;</code></td>
<td><code>25</code></td>
<td><code>\n</code></td>
</tr>
<tr>
<td></td>
<td><code>‘A’</code></td>
<td><code>16.9\n</code></td>
</tr>
<tr>
<td><code>cin &gt;&gt; x;</code></td>
<td><code>25</code></td>
<td></td>
</tr>
<tr>
<td></td>
<td><code>‘A’</code></td>
<td></td>
</tr>
<tr>
<td></td>
<td><code>_</code></td>
<td></td>
</tr>
<tr>
<td></td>
<td><code>x</code></td>
<td><code>16.9\n</code></td>
</tr>
</tbody>
</table>

NOTE: The box shows the location of the file reading marker.
Another example using >>

NOTE: □ shows the location of the file reading marker

<table>
<thead>
<tr>
<th>STATEMENTS</th>
<th>CONTENTS</th>
<th>MARKER POSITION</th>
</tr>
</thead>
<tbody>
<tr>
<td>int i ;</td>
<td></td>
<td>25 A\n16.9\n</td>
</tr>
<tr>
<td>char ch ;</td>
<td></td>
<td>A\n16.9\n</td>
</tr>
<tr>
<td>float x ;</td>
<td></td>
<td>\n16.9\n</td>
</tr>
<tr>
<td>cin &gt;&gt; i ;</td>
<td></td>
<td>\n16.9\n</td>
</tr>
<tr>
<td>cin &gt;&gt; ch ;</td>
<td></td>
<td>\n16.9\n</td>
</tr>
<tr>
<td>cin &gt;&gt; x ;</td>
<td></td>
<td>\n16.9\n</td>
</tr>
</tbody>
</table>
What if user typed X instead of 17?

Enter a number -> 17<return>

- data type of value read does not match type of variable it needs to be stored in
- cin enters **fail state**
- subsequent input operations don’t succeed
- num and response may be garbage
- More LATER
The `get()` function can be used to read a single character.

It obtains the very next character from the input stream without skipping any leading whitespace characters.
At keyboard you type:

A[space]B[space]C[Enter]

NOTE: The file reading marker is left pointing to the space after the ‘B’ in the input stream.
At keyboard you type:
A[space]B[space]C[Enter]

char first;
char middle;
char last;

first middle last

`cin.get( first );`
`cin.get( middle );`
`cin.get( last );`

first middle last

NOTE: The file reading marker is left pointing to the space after the ‘B’ in the input stream.
The `ignore()` function is used to skip (read and discard) characters in the input stream. The call

```cpp
    cin.ignore ( howMany, whatChar ) ;
```

will skip over up to `howMany` characters or until `whatChar` has been read, whichever comes first.
An Example Using `cin.ignore()`

NOTE: shows the location of the file reading marker

<table>
<thead>
<tr>
<th>STATEMENTS</th>
<th>CONTENTS</th>
<th>MARKER POSITION</th>
</tr>
</thead>
<tbody>
<tr>
<td>int a ;</td>
<td></td>
<td>957 34 1235\n</td>
</tr>
<tr>
<td>int b ;</td>
<td></td>
<td>128 96\n</td>
</tr>
<tr>
<td>int c ;</td>
<td></td>
<td></td>
</tr>
<tr>
<td>cin &gt;&gt; a &gt;&gt; b ;</td>
<td></td>
<td>a b c</td>
</tr>
<tr>
<td>cin.ignore(100, '\n') ;</td>
<td></td>
<td>a b c</td>
</tr>
<tr>
<td>cin &gt;&gt; c ;</td>
<td></td>
<td>a b c</td>
</tr>
</tbody>
</table>
**An Example Using cin.ignore( )**

NOTE: shows the location of the file reading marker

<table>
<thead>
<tr>
<th>STATEMENTS</th>
<th>CONTENTS</th>
<th>MARKER POSITION</th>
</tr>
</thead>
<tbody>
<tr>
<td>int a ;</td>
<td></td>
<td></td>
</tr>
<tr>
<td>int b ;</td>
<td></td>
<td></td>
</tr>
<tr>
<td>int c ;</td>
<td></td>
<td></td>
</tr>
<tr>
<td>cin &gt;&gt; a &gt;&gt; b ;</td>
<td>957 34</td>
<td>1235\n</td>
</tr>
<tr>
<td></td>
<td></td>
<td>128 96\n</td>
</tr>
<tr>
<td>cin.ignore(100, ‘\n’) ;</td>
<td></td>
<td></td>
</tr>
<tr>
<td>cin &gt;&gt; c ;</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>1235\n</td>
</tr>
<tr>
<td></td>
<td></td>
<td>128 96\n</td>
</tr>
</tbody>
</table>
### An Example Using `cin.ignore()`

NOTE: □ shows the location of the file reading marker

<table>
<thead>
<tr>
<th>STATEMENTS</th>
<th>CONTENTS</th>
<th>MARKER POSITION</th>
</tr>
</thead>
<tbody>
<tr>
<td>int a;</td>
<td>a</td>
<td>957 34 1235\n</td>
</tr>
<tr>
<td>int b;</td>
<td>b</td>
<td>128 96\n</td>
</tr>
<tr>
<td>int c;</td>
<td>c</td>
<td>1235\n</td>
</tr>
<tr>
<td>cin &gt;&gt; a &gt;&gt; b;</td>
<td></td>
<td></td>
</tr>
<tr>
<td>cin.ignore(100, ‘\n’);</td>
<td></td>
<td></td>
</tr>
<tr>
<td>cin &gt;&gt; c;</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
An Example Using `cin.ignore()`

NOTE: □ shows the location of the file reading marker

<table>
<thead>
<tr>
<th>STATEMENTS</th>
<th>CONTENTS</th>
<th>MARKER POSITION</th>
</tr>
</thead>
</table>
| int a;           |          | 957 34 1235
| int b;           |          | 128 96
| int c;           | a b c    |                 |
| cin >> a >> b;   | 957 34   |                 |
| cin.ignore(100, \n'); | 957 34   | 1235\n |
| cin >> c;        | 957 34 128 | 96\n |
### Another Example Using `cin.ignore()`

NOTE: box shows the location of the file reading marker

<table>
<thead>
<tr>
<th>STATEMENTS</th>
<th>CONTENTS</th>
<th>MARKER POSITION</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>int i;</code></td>
<td></td>
<td></td>
</tr>
<tr>
<td><code>char ch;</code></td>
<td></td>
<td></td>
</tr>
<tr>
<td><code>cin &gt;&gt; ch;</code></td>
<td></td>
<td></td>
</tr>
<tr>
<td><code>cin.ignore(100, 'B');</code></td>
<td></td>
<td></td>
</tr>
<tr>
<td><code>cin &gt;&gt; i;</code></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
### Another Example Using cin.ignore( )

NOTE: ■ shows the location of the file reading marker

<table>
<thead>
<tr>
<th>STATEMENTS</th>
<th>CONTENTS</th>
<th>MARKER POSITION</th>
</tr>
</thead>
<tbody>
<tr>
<td>int i; char ch;</td>
<td></td>
<td>A 22 B 16 C 19\n</td>
</tr>
<tr>
<td>cin &gt;&gt; i;</td>
<td>i ch</td>
<td></td>
</tr>
</tbody>
</table>
Another Example Using `cin.ignore()`

NOTE: □ shows the location of the file reading marker

<table>
<thead>
<tr>
<th>STATEMENTS</th>
<th>CONTENTS</th>
<th>MARKER POSITION</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>int i;</code></td>
<td>![i]</td>
<td>![ch]</td>
</tr>
<tr>
<td><code>char ch;</code></td>
<td>![i]</td>
<td>![ch]</td>
</tr>
<tr>
<td><code>cin &gt;&gt; ch;</code></td>
<td>![i]</td>
<td>![‘A’]</td>
</tr>
<tr>
<td><code>cin.ignore(100, ‘B’);</code></td>
<td>![i]</td>
<td>![‘A’]</td>
</tr>
<tr>
<td><code>cin &gt;&gt; i;</code></td>
<td>![i]</td>
<td>![ch]</td>
</tr>
</tbody>
</table>
Another Example Using `cin.ignore()`

**NOTE:** ❑ shows the location of the file reading marker

<table>
<thead>
<tr>
<th>STATEMENTS</th>
<th>CONTENTS</th>
<th>MARKER POSITION</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>int i;</code></td>
<td></td>
<td></td>
</tr>
<tr>
<td><code>char ch;</code></td>
<td></td>
<td></td>
</tr>
<tr>
<td><code>cin &gt;&gt; ch;</code></td>
<td><code>i</code></td>
<td>❑22 B 16 C 19\n</td>
</tr>
<tr>
<td><code>cin.ignore(100, ‘B’);</code></td>
<td><code>i</code></td>
<td>❑22 B 16 C 19\n</td>
</tr>
<tr>
<td><code>cin &gt;&gt; i;</code></td>
<td><code>16</code></td>
<td>❑16 C 19\n</td>
</tr>
</tbody>
</table>
### Another Example Using `cin.ignore()`

**NOTE:** shows the location of the file reading marker

<table>
<thead>
<tr>
<th>STATEMENTS</th>
<th>CONTENTS</th>
<th>MARKER POSITION</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>int    i ;</code></td>
<td></td>
<td></td>
</tr>
<tr>
<td><code>char   ch ;</code></td>
<td></td>
<td></td>
</tr>
<tr>
<td><code>cin &gt;&gt; ch ;</code></td>
<td></td>
<td></td>
</tr>
<tr>
<td><code>cin.ignore(5, ‘C’) ;</code></td>
<td></td>
<td></td>
</tr>
<tr>
<td><code>cin &gt;&gt; i ;</code></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Another Example Using `cin.ignore()`

NOTE: shows the location of the file reading marker

<table>
<thead>
<tr>
<th>STATEMENTS</th>
<th>CONTENTS</th>
<th>MARKER POSITION</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>int i;</code></td>
<td></td>
<td><code>A 22 B 16 C 19\n</code></td>
</tr>
<tr>
<td><code>char ch;</code></td>
<td></td>
<td></td>
</tr>
<tr>
<td><code>cin &gt;&gt; ch;</code></td>
<td><code>i char</code></td>
<td><code>22 B 16 C 19\n</code></td>
</tr>
<tr>
<td><code>cin.ignore(5, 'C')</code></td>
<td><code>i char</code></td>
<td></td>
</tr>
<tr>
<td><code>cin &gt;&gt; i;</code></td>
<td><code>i char</code></td>
<td></td>
</tr>
</tbody>
</table>
Another Example Using cin.ignore( )

NOTE: shows the location of the file reading marker

<table>
<thead>
<tr>
<th>STATEMENTS</th>
<th>CONTENTS</th>
<th>MARKER POSITION</th>
</tr>
</thead>
<tbody>
<tr>
<td>int i;</td>
<td></td>
<td>A 22 B 16 C 19\n</td>
</tr>
<tr>
<td>char ch;</td>
<td></td>
<td></td>
</tr>
<tr>
<td>cin &gt;&gt; ch;</td>
<td>i ch</td>
<td>22 B 16 C 19\n</td>
</tr>
<tr>
<td>cin.ignore(5, ‘C’) ;</td>
<td>i ‘A’ ch</td>
<td>16 C 19\n</td>
</tr>
<tr>
<td>cin &gt;&gt; i;</td>
<td>i ch</td>
<td></td>
</tr>
</tbody>
</table>
### Another Example Using `cin.ignore()`

NOTE: [ ] shows the location of the file reading marker

<table>
<thead>
<tr>
<th>STATEMENTS</th>
<th>CONTENTS</th>
<th>MARKER POSITION</th>
</tr>
</thead>
<tbody>
<tr>
<td>int i; char ch;</td>
<td>i ch</td>
<td>A 22 B 16 C 19\n</td>
</tr>
<tr>
<td>cin &gt;&gt; ch;</td>
<td>i 'A' ch</td>
<td>22 B 16 C 19\n</td>
</tr>
<tr>
<td>cin.ignore(5, 'C');</td>
<td>i 'A' ch</td>
<td>16 C 19\n</td>
</tr>
<tr>
<td>cin &gt;&gt; i;</td>
<td>16 'A' ch</td>
<td>C 19\n</td>
</tr>
</tbody>
</table>
String Input in C++

Input of a string is possible using the extraction operator $\gg$.

EXAMPLE

```cpp
string message;
cin >> message;
cout << message;
```

HOWEVER . . .
Extraction operator $\gg$

When using the extraction operator ($\gg$) to read input characters into a string variable:

- the $\gg$ operator skips any leading whitespace characters such as blanks and newlines
- it then reads successive characters into the string, and stops at the first trailing whitespace character (which is not consumed, but remains waiting in the input stream)
#include <string>
...
string firstName;
string lastName;
cin >> firstName >> lastName;

Suppose input stream looks like this:

Joe Hernandez 23

WHAT ARE THE STRING VALUES?
Results Using >>

```cpp
string firstName;
string lastName;
cin >> firstName >> lastName;
```

RESULT

firstName      lastName

"Joe"          "Hernandez"
getline( ) Function

- getline ( <stream>, <stringvar>);
- leading whitespace NOT skipped
- reading stops when newline(<eoln>) encountered
  - read pointer left just after <eoln>
  - <eoln> not put into string

getline (cin, message );
String Input Using getline

```cpp
string firstName;
string lastName;
getline(cin, firstName);
getline(cin, lastName);
```

Suppose input stream looks like this:

```
Joe Hernandez 23
```

WHAT ARE THE STRING VALUES?
Results Using `getline`

```cpp
string firstName;
string lastName;
getline(cin, firstName);
getline(cin, lastName);
```

```
"Joe Hernandez 23"
```

firstName

lastName
You have (hopefully) now learned the basics!

• You can write a program in C++ to:
  – Take in some input (cin, etc.)
  – Compute something interesting (using variables and operators)
  – Output the results (cout, etc.)

• Try writing your own program:
  – Compute the volumes of various figures
  – Encrypt or decrypt text or data
  – Compute the cost per area of a pizza
  – Some other problem you need solved…