Extraction Examples

```
int x;
cin >> x;
```

The user types…	Value of x is…	Left on the stream is…
78 94 42	78	94 42
901abh29ks	901	abh29ks
-15.4	-15	.4
jk	not changed!	jk

Extraction Examples

```
double x;
cin >> x;
```

The user types…	Value of x is…	Left on the stream is…
78.56 94.2 42.09	78.56	94.2 42.09
-901abh29ks	-901.0	abh29ks
67.84.29.19	67.84	.29.19
jk	not changed!	jk

Extraction Examples

```
char x;
cin >> x;
```

The user types…	Value of x is…	Left on the stream is…
78 94 42	‘7’	8 94 42
901abh29ks	‘9’	01abh29ks
901abh29ks	‘9’	01abh29ks
jk	‘j’	k

- On the third line, note that extraction always skips leading whitespace!

Extraction Examples

```
string x;
cin >> x;
```

The user types…	Value of x is…	Left on the stream is…
78 94 42	“78”	94 42
901.23ab%!@h29ks	“901.23ab%!@h29ks”	\n (return character)
The rain in Spain	“The”	rain in Spain
jk	“jk”	\n (return character)

- When reading to a string, extraction stops on whitespace

Program Input and Output

- A very common pattern for programs to follow:
  - Get input from some source
  - Process that input
  - Show the results

User Input Statement

- Looks very similar to a print statement

```
cin >> x;
cin >> myVariable;
```

- Extraction operator (`>>`) tells the computer to read from a *input stream* and store in a variable
  - LHS argument is the input stream to read from
    - `cin` gets characters typed into that black box on the screen
  - RHS argument is the variable to store in

User Input Using the `iostream` Library

- The *extraction* operator (`>>`) is a built-in operator
  - It retrieves characters from an *input stream* and stores their value in a variable
  - Like insertion, this requires using the `iostream` library
- The `iostream` library defines the type `istream` (input stream)
  - Input streams move characters from an output device (the keyboard, a file, etc.) to the program
- The `iostream` library also declares the variable `cin`
  - `cin` is of type `istream` (i.e. `istream cin`;)
  - `cin` reads characters typed into the black box on the screen

Stream Input

- A stream handles characters in *sequential order*
  - E.g. Characters output to the screen in order
- A program gets characters from an input stream
  - In the order they are typed by the user
  - The program can only get one character at a time
    - It can get remove it from the stream or not
- The cin iostream *only* sends characters when the user presses the return key
- Working at the level of individual characters is tedious and error-prone
  - The extraction operator (<<) provides a higher level of abstraction for you to work with

Chaining Insertion/Extraction

- You can chain together insertion/extraction expressions in the same statement

```
cout << x;
cout << 67;
cout << endl;
```

  - Does the same thing as:

```
cout << x << 67 << endl;
```

Chaining Insertion/Extraction

- This is possible because:
  - Every expression evaluates to a value
  - The insertion and extraction operators evaluate to the value of their LHS argument (the stream)
- For example:

$$\underbrace{\texttt{cout << x}}_{\texttt{cout}}\ \texttt{<< 67 << endl;} \quad \longleftarrow \quad \text{Prints x}$$

$$\underbrace{\texttt{cout << 67}}_{\texttt{cout}}\ \texttt{<< endl;} \quad \longleftarrow \quad \text{Prints 67}$$

$$\underbrace{\texttt{cout << endl;}}_{\texttt{cout;}} \quad \longleftarrow \quad \text{Prints a newline}$$

Chaining Insertion/Extraction

- Extraction is chained in the same way

```
cin >> x;
cin >> y;
```

  - Is the same as

```
cin >> x >> y;
```

- Common mistake:

```
cin >> x >> endl;
```

  - Attempts to read characters into the variable `endl`, which is not a variable
  - Results in an error

Extraction Rules

- User input is more complicated than output
  - You expect certain data…
  - …but have to deal with it if they type something else
  - (You don’t control what the user types)

- So what *algorithm* (set of steps) does the extraction operator use to turn individual characters into a proper value for the given variable?

Extraction Rules

```
int x, y;
char ch;
```

For the input:

```
5 28 36
```

What are the values of `x`, `y` and `ch` after:

```
a. cin >> x >> y >> ch;
b. cin >> x >> ch >> y;
```

Extraction Rules

```
int x, y;
double z;
```

For the input:

```
37 86.56 32
```

What are the values of `x`, `y` and `z` after:

```
c. cin >> x >> y >> z;
d. cin >> z >> x >> y;
```

Function Prototypes

- Like variables, functions must be declared before they are used
  - That's why we've been putting them at the top of the file, before `main`
  - This simultaneously *declares* and *defines* the function
  - In practice, we like to put `main` first, so we separate the function *declaration* from the function *definition*

Function Prototypes

- Functions are ***declared*** with a prototype
  - Looks just like the function heading as a statement (with ;)

```
double pow( double x, double y );
```

- As long as the ***declaration*** is before the function is used, you can put the *definition* anywhere
  - The definition is unchanged, still has heading and body
  - Convention is to put all function declarations together, followed by all function definitions (with `main` first)

```
// declare a function
int fun1();
int fun1(){
        cout<<"hi";
}
```

```
// allocate memory for a variable
int x;
x = 10;
```

Prototypes and Organization

```
// prototype
double get_side();

int main()
{
   double x;
   // call
   x = get_side();
   return 0;
}

// definition
double get_side()
{
   double input;
   cout << "Please enter a side of the triangle: ";
   cin >> input;
   return input;
}
```

Using Multiple Files

- It can be convenient to organize code into more than one file
  - Particularly for reusing code (libraries)
- Standard libraries are included with:

```
#include <name> (e.g. iostream, string, cmath, etc)
```

- Files in a project are included with:

```
#include “name” (e.g. main.h, functions.cpp, etc)
```

- Including a file merely inserts that file in place of the include directive
- Useful for putting functions in their own file

Exercise 1

- Write the prototype for a function that takes in a string and a number *n* and returns the *nth* word in the string
  - Name: `nth_word`
  - Parameters: `1 string, 1 number`
  - Return value: `1 string`

Exercise 2

- Complete the code below:

```
string sentence = “They switched from the Swingline to the Boston
   stapler, but I kept my Swingline stapler because it didn't bind
   up as much, and I kept the staples for the Swingline stapler
   and it's not okay because if they take my stapler then I'll set
   the building on fire…”;
string word;

// use the nth_word function to get the 14th word out of
//  sentence and store it in word
```