C++ Basic I/O

CSE 1320

Alex Dillhoff

University of Texas at Arlington

Streams

Formatted input and output is provided by the `iostream` library.

Other useful stream libraries, such as one for working with files, are provided by `fstream` and `sstream`

Output

There is a way to provide output for every built-in type as well as user-defined types.

The `<<` ("put to") operator works with objects of type `ostream`.

The default objects are `cout`, the standard output stream, and `cerr`, the standard error stream.

Output

Sending some output to a stream is simple to do with the << operator.

```
// print 1000 to standard output
cout << 1000;

// Print a character
char a = 'c';
cout << a;
```

Output

Complex output can be combined using the << multiple times.

```
string name {"Naomi"};
string ship {"Rocinante"};
cout << name << " boarded the ";
cout << ship << "\n";
```

Input

For input, C++ provides `istreams`.

These work with all basic data types as well as user-defined types.

The `istream` objects are used with the >> ("get from") operator.

Input

The syntax is very simple and easy to read.

```
int selection = 0;
cin >> selection;

double price = 0.0;
cin >> price;
```

Input

It is even more convenient to read strings than in C.

```
string name;
cout << "Enter character name: ";
cin >> name;
```

Input

Just as with `scanf` in C, `cin` will stop reading after a space character.

If you need to read an entire line (up to the newline character), C++ provides the `getline()` function.

Input

`getline` **example**

```
string name;
cout << "Enter character name: ";
getline(cin, name);
```

I/O for User-defined Types

C++ provides the programmer the tools to overload the standard I/O operators for user-defined types.

This makes it much easier and modular to work with complex classes and `struct`s.

I/O for User-defined Types

Consider the following struct.

```
struct Ship {
    string name;
    int id;
};
```

I/O for User-defined Types

We can easily overload the << operator to define the behavior of printing our custom type.

```
ostream& operator<<(ostream& os, const Ship& s) {
    return os << "{\"" << s.name << "\", " << s.id << "}";
}
```

I/O for User-defined Types

Now an object of type `Ship` is much easier to print.

```
Ship s {"Rocinante", 1234};
cout << s << endl;
```

Output

```
{"Rocinante", 1234}
```

File I/O

The C++ standard library provides 3 classes for file I/O:

1. `ifstream` for reading.
2. `ofstream` for writing.
3. `fstream` for reading and writing.

File I/O

They all have similar interfaces. A file can be opened as follows:

```
// Open a file for reading
fstream fs {"file.txt", ios_base::in};
```

The first input is the filename. The second input is called the *stream mode*.

File I/O

There are several different stream modes available.

- `ios_base::app` - append
- `ios_base::ate` - open and seek to end
- `ios_base::binary` - binary mode
- `ios_base::in` - reading
- `ios_base::out` - writing
- `ios_base::trunc` - truncate file to 0 length

File I/O

Once open, you can add or read data from files using the same operators that can be used with other streams.

```
ofstream ofs {"out.txt"};
ofs << "Write to file\n";
ofs.close();
```

File I/O

Lines can be read from a file just as they are with other streams as well.

```
ifstream ifs {"in.txt"};
string input;
getline(ifs, input);
ifs.close();
```