Streams

Object-Oriented Programming with C++

Why streams?

- Original C I/O used printf, scanf

Why streams?

- Original C I/O used printf, scanf
- Streams introduced in C++
  - C I/O libraries still work
- Advantages of streams
  - Better type safety
  - Extensible
  - More object-oriented

Why streams?

- Original C I/O used printf, scanf
- Streams introduced in C++
  - C I/O libraries still work
- Advantages of streams
  - Better type safety
  - Extensible
  - More object-oriented
- Disadvantages
  - More verbose (std::format comes back in C++20)
  - Might be slower
    - Turn off synchronization:

```
std::ios::sync_with_stdio(false);
```

C vs. C++

- C stdio operations work
  - Don't provide "object-oriented" features
  - No overloadable operators
- C++
  - Can overload inserters and extractors
- Moral
  - When converting C to C++, leave the I/O intact

What is a stream?

- Common logical interface to a device
  - One-dimension, unidirectional
  - Random access on file, but not on std::cin/cout

Stream naming convections

	Input	Output	Header
Generic	istream	ostream	<iostream>
File	ifstream	ofstream	<fstream>
C string (legacy)	istrstream	ostrstream	<strstream>
C++ string	istringstream	ostringstream	<sstream>

Stream operations

- Extractors
  - Read a value from the stream
  - Overload the >> operator
- Inserters
  - Insert a value into a stream
  - Overload the << operator
- Manipulators
  - Change the stream state

Kinds of streams

- Text streams
  - Deal with ASCII text
  - Perform some characters translation
    - e.g.: newline -> actual OS file representation
  - Include
    - Files
    - Character buffers
- Binary streams
  - Binary data
  - No translation

Predefined streams

- cin
  - standard input
- cout
  - standard output
- cerr
  - unbuffered error (debugging) output
- clog
  - buffered error (debugging) output

Examples

```
#include <iostream>
int i; float f; char c;
char buffer[80];
```

- **Read the next character**

```
cin >> c;
```

- **Read an integer**

```
cin >> i; // skips whitespace
```

- **Read a float and a string separated by whitespace**

```
cin >> f >> buffer;
```

Predefined extractors

- `istream >> lvalue`

Expression type	input format	C I/O
char	Character	%c
short, int	Integer	%d
long	Long decimal integer	%ld
float	Floating point	%g
double	Double precision flp.	%lg
char []	String	%s
*void **	Pointer	%p

- Extractors skip leading whitespace, in general

Defining a stream extractor

- Has to be a 2-argument free function
  - First argument is an `istream&`
  - Second argument is a *reference* to a value

```
istream& operator>>(istream& is, T& obj) {
    // specific code to read obj
    return is;
}
```

Defining a stream extractor

- Has to be a 2-argument free function
  - First argument is an `istream&`
  - Second argument is a *reference* to a value

```
istream& operator>>(istream& is, T& obj) {
    // specific code to read obj
    return is;
}
```

- Return an `istream&` for chaining

```
cin >> a >> b >> c;
((cin >> a) >> b) >> c;
```

Other input operators

- `int get()`
  - Returns the next character in the stream
  - Returns EOF if no characters left
  - Example: copy input to output

```
int ch;
while ((ch = cin.get()) != EOF)
  cout.put(ch);
```

- `istream& get(char& ch)`
  - Reads the next character into argument
  - Similar to `int get();`

More input operators

- `get(char *buf, int limit, char delim='\n')`
  - Read up to `limit` characters, or to `delim`
  - Appends a null character to `buf`
  - Does not consume the delimiter
- `getline(char *buf, int limit, char delim='\n')`
  - Read up to limit characters, or to `delim`
  - Appends a null character to `buf`
  - Does consume the delimiter
- `ignore(int limit=1, int delim = EOF)`
  - Skip over limit characters or to delimiter
  - Skip over delimiter if found

More input operators

- `int gcount()`
  - Returns number of characters just read

```
char buffer[100];
cin.getline(buffer, sizeof(buffer));
cout << "read" << cin.gcount()
     << " characters"
```

- `void putback(char c)`
  - Pushes a single character back into the stream
- `char peek()`
  - Examines the next character without consuming it

```
switch(cin.peek()) …
```

Predefined inserters

- Usage
  - `ostream << expression`

Expression type	Output format	C I/O
char	Character	%c
short, int	Integer	%d
long	Long decimal integer	%ld
float	Floating point	%g
double	Double precision flp.	%lg
char []	String	%s
*void **	Pointer	%p

Creating a stream inserter

Creating a stream inserter

- Has to be a 2-argument free function
  - First argument is an `ostream&`
  - Second argument is any value

```
ostream& operator<<(ostream& os, const T& obj) {
    // specific code to write obj
    return os;
}
```

- **Return an `ostream&` for chaining**

```
cout << a << b << c;
((cout << a) << b) << c;
```

Other output operators

- `put(char)`
  - Prints a single character
  - Examples

```
cout.put('a');
cerr.put('!');
```

- `flush()`
  - Force output of stream contents
  - Example

```
cout << "Enter a number";
cout.flush();
```

Formatting using manipulators

- Manipulators modify the state of the stream
  - `#include <iomanip>`
  - Effects hold (usually)
- Example

```
int n;
cout << "enter number in hexadecimal"
     << endl;
cin >> hex >> n;
```

Example

- A simple program

```
#include <iostream>
#include <iomanip>
int main() {
  cout << setprecision(2) << 1230.243 << endl;
  cout << setw(20) << "OK!";
  return 0;
}
```

- Prints

```
1.2e+03
                    OK!
```

Manipulators

Manipulator	Effect	Type
dec, hex, oct	Set numeric conversion	I, O
endl	Insert newline and flush	O
flush	Flush stream	O
setw(int)	Set field width	I, O
setfill(ch)	Change fill character	I, O
setbase(int)	Set number base	O
ws	Skip whitespace	I
setprecision(int)	Set floating point precision	O
setiosflags(long)	Turn on specified flags	I, O
resetiosflags(long)	Turn off specified flags	I, O

Creating manipulators

- You can define your own manipulators!

```
// skeleton for an output stream manipulator
ostream& manip(ostream& out) {
    ...
    return out;
}
ostream& tab(ostream& out) {
    return out << '\t';
}
cout << "Hello" << tab << "World!" << endl;
```

Stream flags control formatting

Flag	Purpose (when set)
ios::skipws	Skip leading white space
ios::left, ios::right	Justification
ios::internal	Pad between sign and value
ios::dec, ios::oct, ios::hex	Format for numbers
ios::showbase	Show base of number
ios::showpoint	Always show decimal point
ios::uppercase	Put base in uppercase
ios::showpos	Display + on positive numbers
ios::scientific, ios::fixed	Floating point format
ios::unitbuf	Flush on every write

Setting flags

- Using manipulators
  - `setiosflags(flags);`
  - `resetiosflags(flags);`
- Using stream member functions
  - `setf(flags);`
  - `unsetf(flags);`