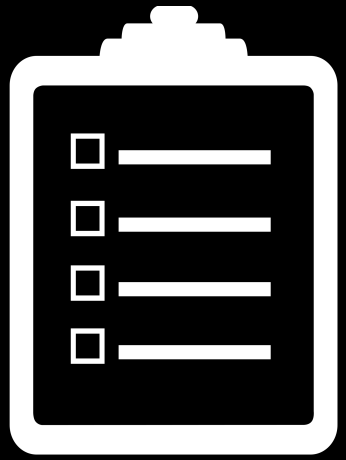
Streams

Ali Malik
malikali@stanford.edu

Game Plan



Recap

Purpose of Streams

Output Streams

Input Streams

Stringstream (maybe)

Announcements

Recap

Recap - Hello, world!

```cpp
#include <iostream>

int main() {
    std::cout << "Hello, world!" << std::endl;
    return 0;
}
```

Recap - Hello, world!

```cpp
#include <iostream>

int main() {
    std::cout << "Hello, world!" << std::endl;
    return 0;
}
```

These can get annoying to write for common names like cout, endl, string etc.

Recap - Hello, world!

```cpp
#include <iostream>

using std::cout;
using std::endl;

int main() {
    std::cout << "Hello, world!" << std::endl;
    return 0;
}
```

Recap - Hello, world!

```cpp
#include <iostream>

using std::cout;
using std::endl;

int main() {
    std::cout << "Hello, world!" << std::endl;
    return 0;
}
```

Whenever you use `cout`, the compiler will assume you mean `std::cout`

Recap - Hello, world!

```
#include <iostream>

using std::cout;
using std::endl;

int main() {
    std::cout << "Hello, world!" << std::endl;
    return 0;
}
```

Whenever you use `cout`, the compiler will assume you mean `std::cout`

Recap - Hello, world!

```
#include <iostream>

using std::cout;
using std::endl;

int main() {
    cout << "Hello, world!" << endl;
    return 0;
}
```

Whenever you use `cout`, the compiler will assume you mean `std::cout`

Recap - Hello, world!

```
#include <iostream>

using std::cout;
using std::endl;

int main() {
    cout << "Hello, world!" << endl;
    return 0;
}
```

Whenever you use `cout`, the compiler will assume you mean `std::cout`

The `using namespace std` directive is a bazooka version of this.

Streams

"Designing and implementing a general input/output facility for a programming language is notoriously difficult"

- *Bjarne Stroustrup*

Streams - Introduction

A stream is an abstraction for input/output.

You can think of it as a source (input) or destination (output) of characters of indefinite length.

Streams - Introduction

A stream is an abstraction for input/output.

You can think of it as a source (input) or destination (output) of characters of indefinite length.


```
<< "Hello, world!"
```

Streams - Introduction

A stream is an abstraction for input/output.

You can think of it as a source (input) or destination (output) of characters of indefinite length.

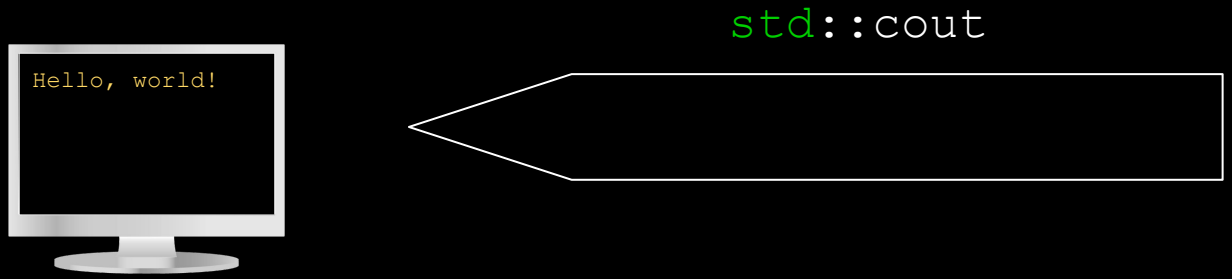
```
std::cout
```

```
"Hello, world!"
```

Streams - Introduction

A stream is an abstraction for input/output.

You can think of it as a source (input) or destination (output) of characters of indefinite length.



The Idea Behind Streams

The Idea Behind Streams

You can write data of multiple types to stream objects.

```
cout << "Strings work!" << endl;
cout << 1729 << endl;
cout << 3.14 << endl;
cout << "Mixed types - " << 1123 << endl;
```

In particular, any primitive type can be inserted into a stream! For other types, you need to explicitly tell C++ how to do this.

The Idea Behind Streams

How does this work?

Idea:

- Input from user is in text form (`string`)
- Output from user is in text form (`string`)
- Intermediate computation needs to be done on object type

The Idea Behind Streams

Streams allow a C++ programmer to convert between the string representation of data, and the data itself.

Types of Streams

Output Streams

Can only receive data.

- The `std::cout` stream is an example of an output stream.
- All output streams are of type `std::ostream`.

Send data using stream insertion operator: `<<`

Insertions converts data to string and sends to stream.

Output Streams

We can use a `std::ostream` for more than just printing to a console.

You can send the data to a file using a `std::ofstream`, which is a special type of `std::ostream`.

Output Stream Example

(`output.cpp`)

Input Streams

Quick test!

How familiar is this:

```
int x;
std::cin >> x;
```

Input Streams

Can only give you data.

- The `std::cin` stream is an example of an input stream.
- All input streams are of type `std::istream`.

Pull out data using stream extraction operator: `>>`

Extraction gets data from stream as a string and converts it into the appropriate type.

Input Streams

Just like with `std::ostream`, we can use a `std::istream` for more than just console IO.

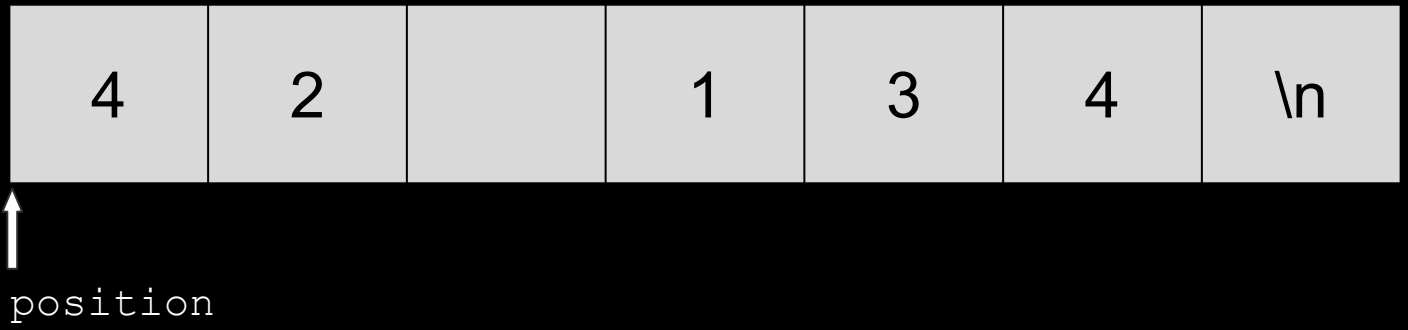
You can read data from a file using a `std::ifstream`.

Input Stream Example

(`input.cpp`)

Input Streams

To understand a `std::istream`, think of it as a sequence of characters.



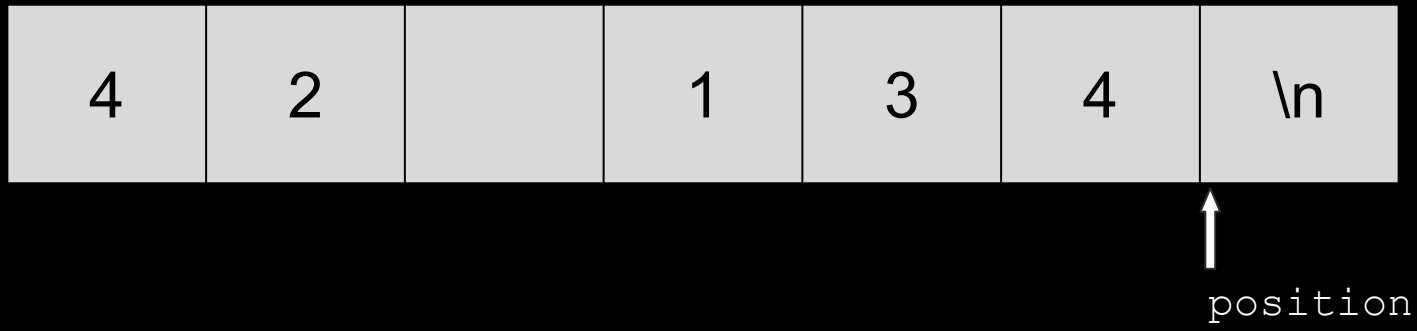
Input Streams

Extracting an integer will read as many characters as possible from the stream.


```
// input is an istream
int value;
input >> value;      // value is now 42
```

Input Streams

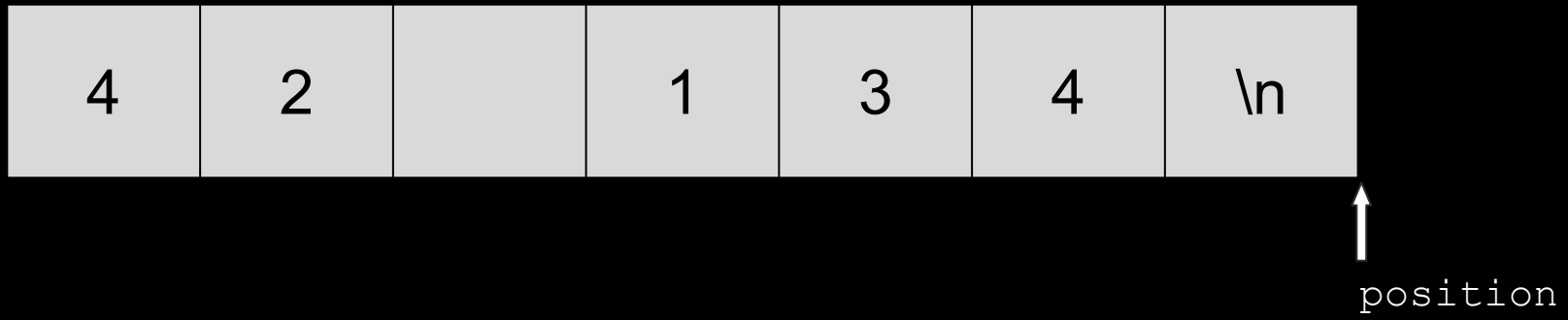
Extracting again will skip over any whitespace when reading the next integer.




```
// input is an istream
int value;
input >> value;      // value is now 134
```

Input Streams

When no more data is left, the fail bit will be set to true and `input.fail()` will return true.




```
// input is an istream
int value;
input >> value;      // value is now ??
```

Input Streams

More Input Stream Examples

(`input.cpp`)

Reading Data From a File

There are some quirks with extracting a string from a stream.

Reading into a string using >> will only read a single word, not the whole line.

To read a whole line, use

```
getline(istream& stream, string& line);
```

Input Streams

More Input Stream Examples

(`input.cpp`)

Input Stream

Think carefully when mixing >> and `getline`!

Using >> can have some weird bugs so next lesson we will talk about a way to avoid it by using `getline` and string streams.

Some Questions to Ponder

What happens if you read into the wrong type?

Can you extract user defined types (e.g. classes) from a stream?

Can you control how output stream output the data we give them?

Is there a stream that might be both an input and output stream?

Find out next time!

Next Time

Streams - The Details