CS100 Recitation 11

GKxx

May 2, 2022

Contents

What have we learnt?

1. Getting Started	
2. Variables and Basic Types	
3. Strings, Vectors, and Arrays	std::string, std::vector, iterators
4. Expressions	
5. Statements	Exception handling (try-catch, throw)
6. Functions	
7. Classes	
8. The IO Library	fstream and stringstream
9. Sequential Containers	
10. Generic Algorithms	
11. Associative Containers	
12. Dynamic Memory	allocator and smart pointers
13. Copy Control	
14. Overloaded Operations and Conversions	
15. Object-Oriented Programming	
16. Templates and Generic Programming	
17. Specialized Library Facilities	
18. Tools for Large Programs	
19. Specialized Tools and Techniques	

A Federation of 4 Languages

Effective C++ Item 1: View C++ as a federation of languages.

- ☑ C
- ☑ Object-Oriented C++
- ☐ Template C++
- ☐ The STL

Operator Overloading

- At least one class-type parameter.
- Cannot change the **precedence** or the **associativity**.

Operators that may be overloaded:

+	-	*	/	%	^
&	|	~	!	,	=
<	>	<=	>=	++	--
<<	>>	==	!=	&&	||
+=	-=	/=	%=	^=	&=
|=	*=	<<=	>>=	[]	()
->	->*	new	new[]	delete	delete[]

Operator Overloading

Operators that may not be overloaded:

```
::   .   .*   ?:
```

Overloaded operator is a function:

- A special name: the `operator` keyword followed by the symbol of the operator.
- Non-member function: Operands are the parameters from left to right.
- Member function: The leftmost operand is implicitly bound to `this`. Other operands are the parameters from left to right.

Operator Overloading

Operators that may not be overloaded:

```
::   .   .*   ?:
```

Overloaded operator is a function:

- A special name: the `operator` keyword followed by the symbol of the operator.
- Non-member function: Operands are the parameters from left to right.
- Member function: The leftmost operand is implicitly bound to `this`. Other operands are the parameters from left to right.

More Effective C++ Item 7 says that never overload operator `&&`, `||` and `,`. Why?

Operator Overloading

We have seen that

- The IO library overloads `operator<<` and `operator>>`.
- The string library overloads `operator+` and `operator[]`.
  - Why won't '`"ABC" + "DEF"`' compile?

Contents

`iostream`, `cin` and `cout`

- `std::cin`: object of type `std::istream`.
- `std::cout`: object of type `std::ostream`.
- `std::istream` and `std::ostream` are **uncopyable** types.

`iostream`, `cin` and `cout`

- `std::cin`: object of type `std::istream`.
- `std::cout`: object of type `std::ostream`.
- `std::istream` and `std::ostream` are **uncopyable** types.
- Outputs can be chained together as in '`cout << a << b`'. Why?

iostream, cin and cout

- `std::cin`: object of type `std::istream`.
- `std::cout`: object of type `std::ostream`.
- `std::istream` and `std::ostream` are **uncopyable** types.
- Outputs can be chained together as in '`cout << a << b`'. Why?

```
inline std::ostream &operator<<
        (std::ostream &os, const Point2d &p) {
  os << "(" << p.get_x() << ", " << p.get_y() << ")";
  return os;
}
```

Test the State of `iostream`

On input failure, no error would be thrown, but we can test this by using the stream object as a condition.

```
struct Vector2d {
  double x, y, norm_l2;
};
inline std::istream &operator>>
        (std::istream &is, Vector2d &v) {
  is >> v.x >> v.y;
  // On input failure, set the object to a valid state.
  if (is)
    v.norm_l2 = std::sqrt(v.x * v.x + v.y * v.y);
  else
    v = Vector2d{};
  return is;
}
```

Examples

Read an unknown number of integers?

```
std::vector<int> v;
int x;
while (std::cin >> x)
  v.push_back(x);
```

Examples

Read an unknown number of integers?

```
std::vector<int> v;
int x;
while (std::cin >> x)
  v.push_back(x);
```

Read a line as a string?

```
std::string line;
std::getline(std::cin, line);
```

Examples

Read an unknown number of integers?

```
std::vector<int> v;
int x;
while (std::cin >> x)
  v.push_back(x);
```

Read a line as a string?

```
std::string line;
std::getline(std::cin, line);
```

- Note: `std::getline` reads until the first newline character (‘\n’), and throws away that newline character.
- What happens?

```
int n; std::cin >> n;
std::string line;
std::getline(std::cin, line);
```

Manipulators

`endl`, `flush` and the like are **manipulators**.

- `endl` outputs a newline character and flushes the buffer.
- `flush` only flushes the buffer.

More manipulators: (some defined in `<iomanip>`)

- `boolalpha`, `noboolalpha`
- `oct`, `hex`, `dec`, `showbase`, `noshowbase`, `setbase`
- `fixed`, `setprecision`, `scientific`
- ......

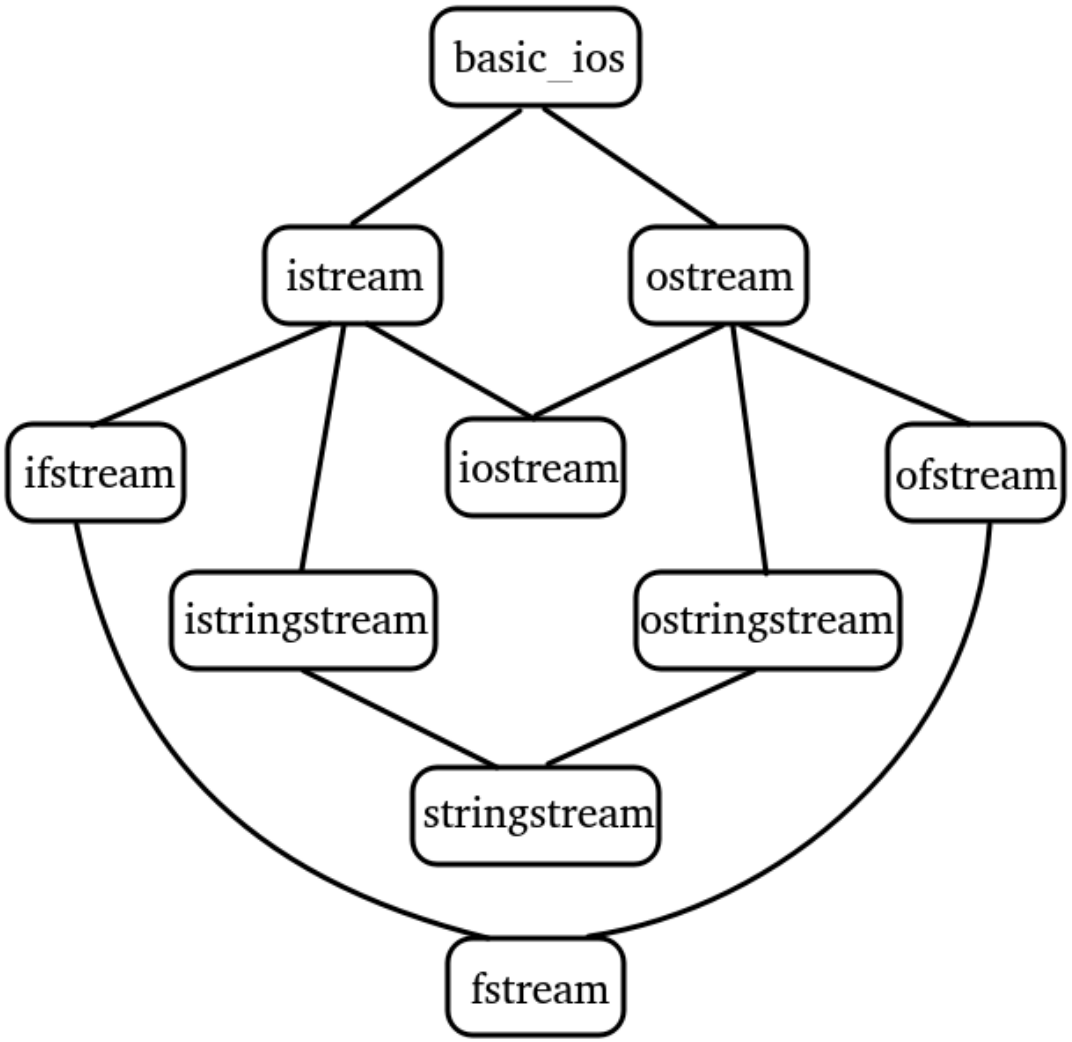
Contents

File Streams

Read an unknown number of integers from a file ‘student_score.txt’?

```
std::ifstream infile("student_score.txt");
// Equivalent way:
// std::ifstream infile;
// infile.open("student_score.txt");
std::vector<int> score;
int x;
while (infile >> x)
  score.push_back(x);
infile.close();
```

Inheritance



- Multiple inheritance
- Virtual inheritance
- What can we know from this?

Real World Example

Read a '.tex' file. Change math from '$...$' to '\(...\)'.

```
std::ifstream infile("hw3.tex");
std::ofstream result("result.tex");
bool in_math = false;
std::string line;
while (std::getline(infile, line)) {
  // process the line
}
infile.close();
result.close();
```

File Modes

Append something to a file, instead of overwriting it?

```
std::ofstream out_file("name.txt", std::ofstream::app);
```

File Modes

Append something to a file, instead of overwriting it?

```
std::ofstream out_file("name.txt", std::ofstream::app);
```

`in`	Open for input
`out`	Open for output
`app`	Seek to the end before every write
`ate`	Seek to the end immediately after the open
`trunc`	Truncate the file
`binary`	Do IO operations in binary mode

- *C++ Primer* 8.2.2

Contents

Stringstreams

Read data from a string, or generate a string by writing different kinds of data.

```
struct Person_info {
  std::string name;
  std::vector<std::string> phones;
};
std::string line;
std::vector<Person_info> people;
while (std::getline(std::cin, line)) {
  Person_info info;
  std::istringstream record(line);
  record >> info.name;
  std::string phone;
  while (record >> phone)
    info.phones.push_back(phone);
  people.push_back(info);
}
```

Stringstreams

Convert some `double` or `int` to a string?

```
inline std::string convert(double value) {
  std::ostringstream oss;
  oss << value;
  return oss.str();
}
```

Stringstreams

Convert some `double` or `int` to a string?

```
inline std::string convert(double value) {
  std::ostringstream oss;
  oss << value;
  return oss.str();
}
```

It works, but `std::to_string` is a better choice!

Contents

Sequential Containers

The standard library provides the following sequential containers:

`vector`	Flexible-size array.
`deque`	Double-ended queue.
`list`	Doubly-linked list.
`forward_list`	Singly-linked list.
`array`	Encapsulation of built-in array.
`string`	A specialized container containing characters.

Consistent Interfaces