Programming in C++

C++ Overview

Curtis Larsen

Utah Tech University—Computing

Spring 2025

Objectives

Objectives:

- Recognize C++ Language Elements
- Recognize Basic Concepts
- Recognize Keywords
- Recognize Preprocessor Directives
- Recognize Expressions
- Recognize Statements
- Recognize Functions
- Recognize Classes
- Recognize Templates
- Recognize Standard Library

Basic Concepts

C++ Programs

- Header files (declarations)
- Implementation files (implementation)
- Translation
- Begin with `main` function

Comments

```
/* C-style comment */
// C++-style comment
/* multi-line comment
   multi-line comment
   multi-line comment */
```

Identifiers

- First character must be one of: `A-Za-z_`
- All other characters must be one of: `0-9A-Za-z_`

Types

- Fundamental: `bool`, `void`
- Fundamental Integer Sizes: `char`, `short`, `int`, `long`, `long long`
- Fundamental Integer Signs: `signed int`, `unsigned int`, `signed X`, `unsigned X`
- Fundamental Floating-point: `float`, `double`, `long double`
- Compound Types: Pointers, Arrays, Enumerations, Classes, and Function Pointers

main

```
int main() {
  return 0;
}


int main(int argc, char* argv[]) {
  return 0;
}
```

Keywords

Subset of Keywords

and	auto	bool	break
case	catch	char	class
const	continue	default	delete
do	double	dynamic_cast	else
enum	extern	false	float
for	if	int	long
namespace	new	not	nullptr
operator	or	private	protected
public	reinterpret_cast	return	short
signed	sizeof	static	static_cast
struct	switch	template	this
throw	true	try	typedef
typeid	typename	union	unsigned
virtual	void	while	xor

Preprocessor Directives

Preprocessor

Part of the translation process. It's a "pre" step.

- `#include`
- `#define`
- `#ifdef`
- `#ifndef`
- `#endif`

Expressions

Operators

Common Operators

`a = b`	`a += b`	`a -= b`	`a *= b`	`a /= b`	`a %= b`
`++a`	`--a`	`a++`	`a--`	`+a`	`-a`
`a + b`	`a - b`	`a * b`	`a / b`	`a % b`	
`!a`	`a && b`	`a || b`	`a()`	`a, b`	`a ? b : c`
`a == b`	`a != b`	`a < b`	`a > b`	`a >= b`	`a <= b`
`a[...]`	`*a`	`&a`	`a->b`	`a.b`	

Values

lvalue : an expression that evaluates to a location

- variable
- `a = b` has location of `a`
- `a.m`
- `a[...]`
- Many other assignment operator expressions.

rvalue : an expression that evaluates to a value but not a location

- literals `10`, `3.14`
- function call that returns a non-reference value
- `a + b`, `a && b` and other binary operator expressions
- Many other experessions

Statements

Forms

- expression
- compound
- declaration
- ...

Forms

```
// Expression ending with ;
expression;

// Compound
{
  statements;
}

// Declaration
type name;
type name = value;
type name(value);
```

if

```
if (condition) { statements-true; }

if (condition) {
  statements-true;
} else {
  statements-false;
}

if (condition1) {
  statements-true1;
} else if (condition2) {
  statements-true2;
} else {
  statements-false;
}
```

switch

```
switch (expression) {
  case value1:
    statements1;
    break;
  case value2:
    statements2;
    break;
  default:
    statements;
    break;
}
```

for

```
for (init-statement; condition; increment-expression) {
    statements;
}

// same as
{
  init-statement;
  while ( condition ) {
    statements;
    increment-expression;
  }
}
```

for

```
for (item-declaration : range) {
    statements;
}

// example
std::vector<int> v = { 2, 3, 5, 7, 11 };
int sum = 0;
for (const int& item : v) {
  sum += item;
}
```

while

```
while(condition) {
  statements;
}

do {
  statements;
} while(condition);
```

Functions

Functions

```
// Declaration
return-type function-name(parameter-list);

// Definition
return-type function-name(parameter-list) {
  statements;
  return value; // if not void return-type
}

// Call
lvalue = function-name(argument-list);
```

Classes

Class Declaration

```
class class-name {
access-specifier:
  declarations;
access-specifier:
  declarations;
// ...
}; // <- ; is *very* important here.

// Example
class Data {
public:
  void setValue(const int v);
  int getValue() const;
private:
  int mValue;
};
```

Method Implementation

```
return-type class-name::function-name(parameter-list) {
  statements;
  return value; // if not void return-type
}

// Examples
void Data::setValue(const int v) {
  mValue = v;
}

int Data::getValue() const {
  return mValue;
}
```

Templates

Generic Programming

```
template < typename T >
class Data {
public:
  void setValue(const T v);
  T getValue() const;
private:
  T mValue;
};
template < typename T >
void Data<T>::setValue(const T v) {
  mValue = v;
}
template < typename T >
T Data<T>::getValue() const {
  return mValue;
}
```

Standard Library

Functionality

- Containers: vectors, maps, sets, lists, etc.
- Input/Output: streams, files, etc.
- Types: string
- Algorithms: sort, search, etc.
- C Standard Library

Example

```
#include <iostream>
#include <numeric>
#include <vector>

template <typename T>
T multiply(const T& a, const T& b) { return a * b; }

int main() {
  std::vector<int> v = { 2, 3, 5, 7, 11, 13 };
  int sum = std::accumulate(v.begin(), v.end(), 0);
  int product = std::accumulate(v.begin(), v.end(),
                                1, multiply<int>);
  std::cout << "sum: " << sum << std::endl;
  std::cout << "product: " << product << std::endl;
  return 0;
}
```