VG101 — Intoduction to Computers & Programming

C++ Review

Liu Yihao

UM-JI (Summer 2018)

July 26, 2018

Namespace

- C: function names conflicts among different libraries
- C++: introduction of namespace
- Each library/program has its own namespace
- Standard library: std

Namespace

```cpp
namespace lyh {
    int foo(int a, int b) { return a + b; }
}
namespace szh {
    int foo(int a, int b) { return a - b; }
}
using namespace lyh;
int main() {
    int a = lyh::foo(1, 2); // a = 3
    int b = szh::foo(1, 2); // b = -1
    int c = foo(1, 2); // c = 3
    using namespace szh;
    int d = foo(1, 2); // error: call of overloaded foo(int, int)
    ↪  is ambiguous
    int e = lyh::foo(1, 2); // e = 3
    int f = szh::foo(1, 2); // f = -1
    return 0;
}
```

String

Improvements on strings:

- Strings in C: array of characters
- Many limitations, low level manipulations
- New type in C++: string

```cpp
#include <string>
using namespace std;
string g="good "; string m="morning";
cout << g + m + "!\n";
```

More possibilities: search and learn how to use strings in C++

Input Stream

- Standard Input: `std::cin` (*#include <iostream>*)
- File Input: `std::ifstream` (*#include <fstream>*)
- Input String Stream: `std::istringstream` (*#include <sstream>*)

- Use operator >> in input stream.
- Use `std::getline` to read line by line.
- Use >> `std::ws` to consume whitespace.

Output Stream

- Standard Output: `std::cout` (`#include <iostream>`)
- File Output: `std::ofstream` (`#include <fstream>`)
- Output String Stream: `std::ostringstream` (`#include <sstream>`)

- Use operator << in output stream.
- Use << `std::flush` to flush the output stream.
- Use << `std::endl` to output `'\n'` and flush the output stream.

Input/Output Manipulators

Manipulators are helper functions that make it possible to control input/output streams using operator << or operator >>.

You should `#include <iomanip>` for these four manipulators.

setbase	changes the base used for integer I/O
setfill	changes the fill character
setprecision	changes floating-point precision
setw	changes the width of the next input/output field

Input/Output Manipulators

boolalpha	switches between textual and numeric representation of booleans
showbase	controls whether prefix is used to indicate numeric base
showpoint	controls whether decimal point is always included in floating-point representation
showpos	controls whether the + sign used with non-negative numbers
skipws	controls whether leading whitespace is skipped on input
uppercase	controls whether uppercase characters are used with some output formats
unitbuf	controls whether output is flushed after each operation
internal/left/right	sets the placement of fill characters
dec/hex/oct	changes the base used for integer I/O
fixed/scientific/hexfloat/defaultfloat	changes formatting used for floating-point I/O

Example: std::setprecision

```cpp
#include <iostream>
#include <iomanip>
#include <cmath>
#include <limits>
int main()
{
    const long double pi = std::acos(-1.L);
    std::cout << "default precision (6): " << pi << '\n'
              << "std::setprecision(10): " << std::setprecision(10)
              ↪  << pi << '\n'
              << "max precision:         "
              << std::setprecision(std::numeric_limits<long
              ↪  double>::digits10 + 1)
              << pi << '\n';
}
```

```
default precision (6): 3.14159
std::setprecision(10): 3.141592654
max precision: 3.141592653589793239
```

Object Oriented Programming

- Everything is an object libraries
- Objects communicate between them by sending messages
- Each object has its own type
- Object of a same type can receive the same message

An object has two main components:

- The data it contains, what is known to the object, its attributes or data members
- The behavior it has, what can be done by the object, its methods or function members

Class and Instance

Class:

- Defines the family, type or nature of an object
- Equivalent of the type in “traditional programming”

Instance:

- Realization of an object from a given class
- Equivalent of a variable in “traditional programming”

Constructor and Destructor

Constructor:

- Method that initialises an instance of an object
- Used for a proper default initialization
- Definition: no type, name must be `classname`
- Important note: can have more than one constructor
- Question: What's an explicit constructor?

Destructor:

- Called just before the object is destroyed
- Used for clean up (e.g. release memory, close a file etc...)
- Definition: no type, name must be ∼`classname`
- Question: What's an pure virtual destructor?

Private

Reminder on private members:

- Everything private is only available to the current class
- Derived classes cannot access or use them

Private inheritance:

- Default type of class inheritance
- Any public member from the base class becomes private
- Allows to hide “low level” details to other classes

Public

Reminder on public members:

- They are available to the current class
- They are available to any other class

Public inheritance:

- Anything public in the base class remains public
- Nothing private in the base class can be accessed

Problem:

- Private is too restrictive while public is too open
- Need a way to only allow derived classes and not others

Protected

Protected members:

- Compromise between public and private
- They are available to any derived class
- No other class can access them

Possible to bypass all this security using keyword friend:

- Valid for both functions and classes
- A class or function declares who are its friends
- Friends can access protected and private members
- As much as possible do not use friend

Visibility

Attributes and methods:

Visibility	Base Classes	Derived Classes	Others Classes
Private	Yes	No	No
Protected	Yes	Yes	No
Public	Yes	Yes	Yes

Inheritance:

Base class	Public Inherit	Private Inherit	Protected Inherit
Private	-	-	-
Protected	Protected	Private	Protected
Public	Public	Private	Protected

Virtual Function

New keyword: `virtual`

- Virtual function in the base class
- Function can be redefined in derived class
- Preserves calling properties

Drawbacks:

- Binding: connecting function call to function body
- Early binding: compilation time
- Late binding: runtime, depending on the type, more expensive
- virtual implies late binding

Abstract class

Pure virtual function (method):

- Write a totally **abstract** class “at the top”
- This class has virtual member functions without any definition
- The method definition is replaced by $= 0$

A class with any pure virtual function defined, or with any pure virtual function inherited but not implemented is called an abstract class. An abstract class can not be instantiated.

You can only use pointer of an abstract class in most situations. The calls to virtual functions in the base class will be redirected to the functions with the same name in the derived classes.

Multiple Inheritance

Simple idea: a class can inherit from multiple classes

- Create a hierarchy without diamond problem
- Declare the derived classes as `virtual`
- Important: if the diamond problem appears in a diagram, redesign the whole hierarchy

A kind reminder: If you can design a hierarchy without multiple inheritance to solve the problem, never use it to make your hierarchy more complex.

Containers Library

Sequence containers:

- vector
- list
- deque

Associative containers:

- set
- map

Unordered associative containers:

- unordered_set
- unordered_map

Container adaptors:

- stack
- queue
- priority_queue

Algorithms Library

The algorithms library defines functions for a variety of purposes (e.g. searching, sorting, counting, manipulating) that operate on ranges of elements. Note that a range is defined as [first, last) where last refers to the element past the last element to inspect or modify.

Here are some useful functions:

sort	sorts a range into ascending order
swap	swaps the values of two objects
for_each	applies a function to a range of elements
count	returns the number of elements satisfying specific criteria
find	finds the first element satisfying specific criteria
search	searches for a range of elements
merge	merges two sorted ranges
max_element	returns the largest element in a range

Dynamic Memory Management

Low level memory management: new and delete.

- Memory for a variable: `int *p = new int;`
- Memory for an array: `int *p = new int[10];`
- Return `NULL` on failure
- Release the memory: `delete p` or `delete[] p`

Smart pointers: enable automatic, exception-safe, object lifetime management.

unique_ptr	smart pointer with unique object ownership semantics
shared_ptr	smart pointer with shared object ownership semantics

Pairs and Tuples

To be continued.