PART 4

Templates

C++ / Python Programming

Session #2

@Institut d’Astrophysique de Paris

OUTLINE

1. Templates
   a. Function Template
   b. Class Template
2. C++ standard libraries

Introduction

- **Templates** – extend the re-usability letting it accepts using different types of objects but one type at the same type.

- **STL** – Standard Template Library

  a. standard library of C++

  b. contains lots of useful classes

  c. mainly based on templates

  d. provide basic classes

  - istream, ostream, cin, cout, ... <iostream>
  - string <string>

  e. data structure

  - vector, list, stack, set, map, etc
  - with template parameters
  - using the iterator concept

Templates

- templates facilitate generic programming
- generic programming: algorithms written in terms of types to be specified later (i.e., algorithms are generic in sense of being applicable to any type that meets only some very basic constraints)
- extremely important language feature
- avoids code duplication
- leads to highly efficient and customizable code
- promotes code reuse
- C++ standard library makes very heavy use of templates (actually, most of standard library consists of templates)
- many other libraries make heavy use of templates (e.g., Boost)

Function Templates

- function template is family of functions parameterized by one or parameters
- each template parameter can be: non-type (integral constant), type, template
- syntax for template function has general form:

  template <*parameter list*> *function declaration*
- *parameter list*: parameters on which template function depends
- *function*: function declaration
- type parameter designated by **class** or **typename** keyword
- template parameter designated by **template** keyword
- non-type (integral constant) parameter designed by its type (e.g., **int**)
- function template definitions usually appear in header file

Function Templates

- consider following functions:

```
int max(int x, int y)
   { return x > y ? x : y; }

double max(double x, double y)
   { return x > y ? x : y; }

// more similar-looking max functions...
```

- each of above functions has *same general form*; that is, for some type T, we have:

```
T max(T x, T y)
  {   return x > y ? x : y; }
```

- would be nice if we did not have to repeatedly type, debug, test, and maintain nearly identical code
- in effect, would like code to be parameterized on type T

Example: Function Templates

```
// compute minimum of two values
  template <class T>
  T  min (T  x,T  y){
      return x < y ? x : y;
}

// compute square of value
template <typename T >
T sqr(T  x) {
  return x * x;
}
```

```
// swap two values
template <class T>
void swap(T& x, T& y){
  T tmp = x;
  x = y;
  y = tmp;
}

// increment value by constant
template <int N = 1, typename T>
T& increment_by(T& n) {
n += N;
return n;
}
```

Class Templates

- class template is family of classes parameterized on one or more parameters
- each template parameter can be: non-type (integral constant), type, template
- syntax has general form: **template** <*parameter list*> *class*
- *parameter list*: parameter list for class
- *class*: class/struct declaration or definition
- compiler only generates code for class template when it is instantiated (i.e., used)

C++ Standard Libraries

- C++ standard library provides huge amount of functionality (orders of magnitude more than C standard library)
- uses std namespace (to avoid naming conflicts)
- functionality can be grouped into following sub-libraries:
  1. language support library (e.g., exceptions, memory management)
  2. diagnostics library (e.g., exceptions, error codes)
  3. general utilities library (e.g., date/time)
  4. strings library (e.g., C++ and C-style strings)
  5. localization library (e.g., date/time formatting and parsing, character classification)
  6. algorithms library (e.g., searching, sorting, merging, set operations, heap operations, minimum/maximum)
  7. numerics library (e.g., complex numbers, math functions)
  8. input/output (I/O) library (e.g., streams)
  9. thread support library (e.g., threads, mutexes, condition variables, futures)
  10. containers library (e.g., sequence containers and associative containers)

Commonly used Libraries

Language support library:

Header file	Description
cstdlib	run-time support, similar to stdlib.h from C (e.g., exit)
exception	exception handling support (e.g., set_terminate, current_exception)
limits	properties of fundamental types (e.g., numeric_limits)
initializer_list	initializer_listclasstemplate

Containers, iterators and Algorithms library:

Header file	Description
array	array class
vector	vector class
deque	deque class
list	list class
set	set classes (i.e. set, multiset)
map	map classes (i.e. map, multimap)
unordered_set	unordered set classes (i.e., unordered_set, unordered_multiset)
unordered_map	unordered map classes(i.e., unordered_map, unordered_multimap)
iterator	iterators (e.g., reverse_iterator, back_inserter)
algorithm	algorithms (e.g., min, max, sort)

References

1. A tower of C ++ - Bjarne Stroustrup
2. Effective Modern C ++ - Scott Meyers
3. Thinking in C ++ - Bruce Eckel
4. Websites:
   - https://en.cppreference.com/w/
   - http://www.cplusplus.com/
   - https://www.tutorialspoint.com/cplusplus/
   - https://www.onlinegdb.com/online_c++_compiler
   - https://www.geeksforgeeks.org