AI INSTITUTE #AIISC
UNIVERSITY OF SOUTH CAROLINA

UNIVERSITY OF SOUTH CAROLINA

CSCE 240: Advanced Programming Techniques

Lecture 17: C++ Standard Library, Testing Strategies, HW4 (Given), PA4 (Start)

PROF. BIPLAV SRIVASTAVA, AI INSTITUTE

12TH MARCH 2024

Carolinian Creed: "I will practice personal and academic integrity."

Credits: Some material reused with permission of Dr. Jeremy Lewis. Others used as cited with thanks.

Organization of Lecture 17

- Introduction Section
  - Recap of Lecture 16
  - Mid-course pulse survey
- Main Section
  - Concept: Standard Library
  - HW 4 given
  - Discussion: Project
    - Start of PA #4
- Concluding Section
  - About next lecture – Lecture 18
  - Ask me anything

Introduction Section

Recap of Lecture 16

- We looked at the concept of operators
  - Many types
  - Precedence order when evaluating
  - Defining one's own operator
- Programming Assignment #3 completed

Mid Course Pulse Survey

- Link: https://forms.gle/z6toZVkeF3a2YzaW9
- Questions like (7)
  - Do you like the pace of the course ?
  - Do you like the content on which the course is focusing?
  - Should the number of HWs be reduced?
  - …
  - Any other comments?
- Please respond by Wednesday, 5pm

Main Section

Concept: C++ Standard Library

C++ reference

C++11, C++14, C++17, C++20, C++23, C++26 | Compiler support C++11, C++14, C++17, C++20, C++23, C++26

Language
- Keywords – Preprocessor
- ASCII chart
- Basic concepts
  - Comments
  - Names (lookup)
  - Types (fundamental types)
  - The `main` function
- Expressions
  - Value categories
  - Evaluation order
  - Operators (precedence)
  - Conversions – Literals
- Statements
  - `if` – `switch`
  - `for` – range-`for` (C++11)
  - `while` – `do-while`
- Declarations – Initialization
- Functions – Overloading
- Classes (unions)
- Templates – Exceptions
- Freestanding implementations

Standard library (headers)

Named requirements

Feature test macros (C++20)

Language support library
- Program utilities
- `source_location` (C++20)
- Coroutine support (C++20)
- Three-way comparison (C++20)
- Type support
- `numeric_limits` – `type_info`
- `initializer_list` (C++11)

Concepts library (C++20)

Diagnostics library
- `exception` – System error
- `basic_stacktrace` (C++23)

Memory management library
- `unique_ptr` (C++11)
- `shared_ptr` (C++11)
- `weak_ptr` (C++11)
- Memory resources (C++17)
- Allocators – Low level management

Metaprogramming library (C++11)
- Type traits – `ratio`
- `integer_sequence` (C++14)

General utilities library
- Function objects – `hash` (C++11)
- Swap – Type operations (C++11)
- Integer comparison (C++20)
- `pair` – `tuple` (C++11)
- `optional` (C++17)
- `expected` (C++23)
- `variant` (C++17) – `any` (C++17)
- String conversions (C++17)
- Formatting (C++20)
- `bitset` – Bit manipulation (C++20)
- Debugging support (C++26)

Strings library
- `basic_string` – `char_traits`
- `basic_string_view` (C++17)
- Null-terminated strings:
  - byte – multibyte – wide

Containers library
- `vector` – `deque` – `array` (C++11)
- `list` – `forward_list` (C++11)
- `map` – `multimap` – `set` – `multiset`
- `unordered_map` (C++11)
- `unordered_multimap` (C++11)
- `unordered_set` (C++11)
- `unordered_multiset` (C++11)
- Container adaptors
- `span` (C++20) – `mdspan` (C++23)

Iterators library

Ranges library (C++20)

Algorithms library
- Execution policies (C++17)
- Constrained algorithms (C++20)

Numerics library
- Common math functions
- Mathematical special functions (C++17)
- Mathematical constants (C++20)
- Basic linear algebra algorithms (C++26)
- Numeric algorithms
- Pseudo-random number generation
- Floating-point environment (C++11)
- `complex` – `valarray`

Date and time library
- Calendar (C++20) – Time zone (C++20)

Localization library
- `locale` – Character classification
- `text_encoding` (C++26)

Input/output library
- Print functions (C++23)
- Stream-based I/O – I/O manipulators
- `basic_istream` – `basic_ostream`
- Synchronized output (C++20)
- File systems (C++17)

Regular expressions library (C++11)
- `basic_regex` – Algorithms
- Default regular expression grammar

Concurrency support library (C++11)
- `thread` – `jthread` (C++20)
- `atomic` – `atomic_flag`
- `atomic_ref` (C++20) – `memory_order`
- Mutual exclusion – Semaphores (C++20)
- Condition variables – Futures
- `latch` (C++20) – `barrier` (C++20)
- Safe Reclamation (C++26)

Technical specifications
- **Standard library extensions** (library fundamentals TS)
  - `resource_adaptor` – `invocation_type`
- **Standard library extensions v2** (library fundamentals TS v2)
  - `propagate_const` – `ostream_joiner` – `randint`
  - `observer_ptr` – Detection idiom
- **Standard library extensions v3** (library fundamentals TS v3)
  - `scope_exit` – `scope_fail` – `scope_success` – `unique_resource`
- **Parallelism library extensions v2** (parallelism TS v2)
  - `simd`
- **Concurrency library extensions** (concurrency TS)
- **Transactional Memory** (TM TS)
- **Reflection** (reflection TS)

Credit: https://en.cppreference.com/w/cpp
Accessed: March 9, 2024

Many Implementations

Name	Organization	Homepage	Acronym	Licence	Latest release
GNU C++ Standard Library	GNU Project and Free Software Foundation	[1]	libstdc++	GPLv3	Unknown
LLVM C++ Standard Library	LLVM Developer Group	[2]	libc++	Apache License 2.0 with LLVM Exceptions	Every 2 weeks
NVIDIA C++ Standard Library	Nvidia	[3]	libcu++	Apache License 2.0 with LLVM Exceptions	October 12, 2022; 4 months ago
Microsoft C++ Standard Library	Microsoft	[4]	MSVC STL	Apache License 2.0 with LLVM Exceptions	Daily
HPX C++ Standard Library for Parallelism and Concurrency	STELLAR Group	[5]	HPX	Boost Software License 1.0	August 6, 2022; 6 months ago
Electronic Arts Standard Template Library	Electronic Arts	[6]	EASTL	BSD 3-Clause License	October 20, 2021; 16 months ago
Dinkum C++ Library	Dinkumware	[7]	Unknown	Commercial	Unknown
Cray C++ Standard Library	Cray User Group	[8]	Unknown	Commercial	Unknown

Credit: https://en.wikipedia.org/wiki/C%2B%2B_Standard_Library

Why Use Standard Library and Why Not ?

- Note: One can always implement a functionality themselves
- Reasons to reuse
  - Lesser development effort. Someone has created it.
  - Task needs specialized knowledge that the developer does not have
  - Usually, well tested.
  - Usually, efficient.
  - Well-documented. So, code using them easier to maintain
- Reasons not to reuse
  - Want to be in control of behavior and performance
  - Want to control code size/ memory footprint
  - Task needs specialized knowledge that the developer has

Credit: Adapted from 'Fundamentals of C++ Programming', Richard Halterman

Commonly Used: String

- Purpose: Make working with strings easy
- Examples
  - **Position**: front, back
  - **Size related**: size, capacity
  - **Character manipulation:** replace
  - **Search**: find
  - **Type conversion**: stoi, stof

Reference:
https://en.cppreference.com/w/cpp/string/basic_string

Credit: https://en.wikipedia.org/wiki/C%2B%2B_Standard_Library

C++ Standard Library

- Input/output
- Strings
- algorithm
- functional

Containers

- Sequence containers
- Associative containers
- Unordered associative containers

C standard library

- Data types
- Character classification
- Strings
- Mathematics
- File input/output
- Date/time
- Localization
- Memory allocation
- Process control
- Signals
- Alternative tokens
- Miscellaneous headers:
  - <assert.h>
  - <errno.h>
  - <setjmp.h>
  - <stdarg.h>

Commonly Used: String

- Code illustration
  - Front
  - Back
  - Size
  - Capacity
  - substr

Description: https://en.cppreference.com/w/cpp/string/basic_string
Demo: https://github.com/biplav-s/course-adv-proglang/blob/main/sample-code/CandC%2B%2B/Class15and16_OperatorSTL/src/Class15and16_OperatorSTL.cpp, demoStrings()

Commonly Used: Mathematical Functions

- Purpose: Make numerical computation easy

- Examples
  - **Basic**: abs, mod, nan (not a number), round, nearestint, infinity
  - **Exponential**: exp, log
  - **Power:** pow, sqrt, hypot (computes square root of the sum of the squares of two or three)
  - **Trigonometric**: sin, cos, tan, atan
  - **Floating point**: round, floor, ceil

Description: https://en.cppreference.com/w/cpp/numeric/math
Demo: https://github.com/biplav-s/course-adv-proglang/blob/main/sample-code/CandC%2B%2B/Class15and16_OperatorSTL/src/Class15and16_OperatorSTL.cpp, demoMaths()

Commonly Used: Mathematical Functions

- Code illustration
  - Sqrt -- square root
  - Cbrt -- cubic root
  - Round
  - Nearbyint
  - Infinity, nan

- Support for complex numbers - example
  - **Description**: https://en.cppreference.com/w/cpp/numeric/complex

Sometimes Used: Algorithmic Functions

- Purpose: Make ready implementation of popular algos

- Examples
  - Sequence operations: count, find, search
  - Sorting: sort
  - Partitioning
  - Permutation
  - Set operations
  - Numeric

Notes

- auto: a *placeholder* datatype defined in C++11 whose actual type is inferred from initialization
  - https://learn.microsoft.com/en-us/cpp/cpp/auto-cpp?view=msvc-170
- use of templates, which will be explained in a later class

Sometimes Used: Algorithmic Functions

- Code illustration
  - Sort
  - permutation

Description: https://en.cppreference.com/w/cpp/algorithm
Demo: https://github.com/biplav-s/course-adv-proglang/blob/main/sample-code/CandC%2B%2B/Class15and16_OperatorSTL/src/Class15and16_OperatorSTL.cpp, demoAlgos()

Sometimes Used: Container Functions

- Purpose: Make implementation of useful containers easily available

- Examples
  - Array
  - List - https://en.cppreference.com/w/cpp/container/list
  - Vector
  - Map (also called HashMap or dict in other languages)
  - Priority_queue

Description: https://en.cppreference.com/w/cpp/container
Demo: https://github.com/biplav-s/course-adv-proglang/blob/main/sample-code/CandC%2B%2B/Class15and16_OperatorSTL/src/Class15and16_OperatorSTL.cpp, demoContainer()

Concept: Testing Strategies

Testing – What is It ?

- Ensure software works
  - As asked
    - Customer wanted – requirement
    - Developer says it works – specification
  - On diverse data
    - Test data
    - Unseen data
  - Under various conditions
    - Ideal condition (as and if customer stipulates)
    - Typical operating condition
  - Without harm

Important Types of Testing

- Unit testing
  - Purpose: Check a basic functionality is working. Example, a function or programming assignment in course project
  - Developer does on their own
- Integration testing
  - Purpose: Ensure different components of project work together. Example, complete course project
  - Developer or dedicated tester performs
- Functional testing
  - Purpose: business requirement is met. Checks output, not intermediate results
  - Tester performs
- Acceptance testing
  - Purpose: business requirement is met both functionally and non-functionally like performance, throughput. Checks output, not intermediate results
  - Tester performs; customer performs
- Regression testing
  - Purpose: ensure existing functionality is preserved; especially after a code change
  - Tester performs

We are mostly doing unit and integration testing in the course

How to Perform Testing

- Manual Testing
  - Common testing practice; usually the default if not specified otherwise
  - Common for unit and system testing

- Automated Testing
  - Needs specification of expected outcome
  - Common for performance and regression testing

We are mostly doing **unit and integration** manual testing in the course

When to Stop Testing

- Code coverage is over a limit: when desired percentage of code has been exercised by test cases
  - Code Coverage = (Number of lines of code executed) / (Total Number of lines of code in the system component) * 100
- Number of bugs discovered exceeds a count
- All high priority bugs are identified and fixed

Home Work 4

Due Thursday, March 14, 2024

Home Work (#4) – C++ - Background

- Email programs parse Email headers and show content. The headers have **parts** (e.g., CC, To, From) that are part of a standard and also proprietary extensions.
- Defined with IETF RFC - https://datatracker.ietf.org/doc/html/rfc5322
  - description https://www.tutorialspoint.com/rfc-5322-internet-message-format
  - Examples for Microsoft Outlook and Gmail are shown.
- Let us assume that parts which are common to both are the standard and those unique are proprietary. So, "CC" is common and "X-MS-Has-Attach" is unique.
- Write a program, ***EmailInformationExtractor,*** which, when given a message header from either of the two programs, and a part name, will read the value of the message part.

Microsoft Outlook Header

- Received: from DS7PR19MB5853.namprd19.prod.outlook.com …
- Authentication-Results: dkim=none (message not signed
- Received: from …
- Content-Type: application/ms-tnef; name="winmail.dat"
- Content-Transfer-Encoding: binary
- From: "Sri Naga Sushmitha, Satti" <SATTI@cse.sc.edu>
- To: "Srivastava, Biplav" <BIPLAV.S@sc.edu>
- CC: "Baldwin, Randi" <baldwin@cse.sc.edu>
- Subject: Re: Possible need for … 240
- Thread-Topic: Possible need for printout for .. 240
- Thread-Index: … +AAAIRpoAAAp/ggAAAJH0=
- Date: Tue, 15 Feb 2022 13:52:33 +0000
- Message-ID: <…>
- References: …
- In-Reply-To: <…>
- Accept-Language: en-US
- Content-Language: en-US
- X-MS-Has-Attach:
- X-MS-Exchange-Organization-SCL: -1

Home Work (#4) – C++ - Requirement

- So, program name: ***EmailInformationExtractor***
- Inputs:
  - message header
  - Part name
- Output:
  - Value
- Hint
  - Use regex
  - Use standard libraries

Gmail Header

- Delivered-To: biplav.srivastava@gmail.com
- Received: by 2002:a05:7000:1f97:0:0:0:0 with SMTP …
- X-Google-Smtp-Source: ABdhPJz/…
- Received: from m08b.cvent-planner.com …
- From: Reply-To:To:Message-ID:Subject:MIME-Version:
- Content-Type: List-Unsubscribe; /Tvkdd8/15SWlBA=; …
- Date: Thu, 17 Feb 2022 23:56:12 +0000
- From: AAAI Staff <aaai22@aaai.org>
- Reply-To: <aaai22@aaai.org>
- To: Biplav Srivastava <biplav.srivastava@gmail.com>
- Message-ID: <..>
- Subject: AAAI-22 General Information
- MIME-Version: 1.0
- Content-Type: multipart/alternative; ..
- Content-Type: text/plain; charset=UTF-8
- Content-Transfer-Encoding: quoted-printable

Home Work (#4) – C++ - Code Design

- Create 3 classes:
  - Base class with common parts: BaseEmailHeaderType
  - Children classes with custom parts: GmailHeaderType, OutlookHeaderType
- Use exception to handle likely errors

Discussion: Course Project

Course Project – Knowing About Companies

- **Project**: Develop collaborative assistants (chatbots) that offer useful information about companies
- Specifically, use the EDGAR dataset on companies at: https://www.sec.gov/edgar/searchedgar/companysearch.
  - For Apple, it is: https://www.sec.gov/edgar/browse/?CIK=320193&owner=exclude
- **Each student will choose two companies (from thousand available).**
- Programming assignment programs will: (1) extract data about two companies from 10-k, (2) process it, (3) make content available in a command-line interface, (4) handle any user query and (5) report on interaction statistics.

Core Programs Needed for Project

- Prog 1: extract data from the district **[prog1-extractor]**
- Prog 2: process it (extracted data) based on questions **[prog2processor]**
- Prog 3: make content available in a command-line interface **[prog3-ui]**
- **Prog 4: handle any user query [prog4-userintent2querymapper]**
- Prog 5: report statistics on interaction of a session, across session

Objective in Programming Assignment # 4:
Remove Requirement on User to Know Supported Queries!

- Until now, use needed to know what the program supports.
- **Can the system adapt rather than ask the user to adapt ?**
- **Approach Suggested**
  - Take user's utterance
  - Understand query and company of interest
  - Match to the closest supported query
    - ***Intents***: [Parts and Items] **+ 3 more**
    - Also, add a confidence estimate
  - If confidence greater than a threshold and if the company is supported
    - Run the query,
  - Otherwise
    - Ask user to re-phrase and ask again

- Program should do the following:
  - Run in an infinite loop until the user wants to quit
  - Handle any user response
    - **[#1]** User can quit by typing "Quit" or "quit" or just "q"
    - User can enter any other text and the program has to handle it. The program should write back what the user entered and say – "I do not know this information".
  - Handle known user query
    - "Tell me about ***IBM***" or "What are the risk factor for ***IBM***?" => (Part 1), or (Part 1: Item 2), accordingly
    - "What markets does ***IBM*** operate in?", "Are there aby disclosures from ***IBM***?" => (Part 2)
    - "who are the directors? " => (Part 3: Item ..) // assume company, or tell of all companies, or ask …
    - "Tell me about ***IBM's*** statements" => (Part 4)
    - …
    - "**Tell me everything**" => ***Give all information extracted***

Intents: [Parts and intents] +
tell everything, chitchat and quit

Content Reference: Queries for (Answers) Data We Have

- What does the (company) do? // Answers in Part 1
  - What is the (company's) business?
  - What are (company's) risk factors?
  - What does (company) own?
  - …
- Where does (company) operate? // Answers in Part 2
  - What has (company) disclosed?
- How is (company) structured? // Answers in Part 3
  - Who is (company's) CEO?
  - How much does (person) earn?
  - …
- What was in (company) statements? // Answers in Part 4
  - …

Concepts: 10-K, Parts, Items

Parts

- Part 1: Business Background and Risks
  - Item 1: Business
  - Item 2: Risk factors
  - Item 3: Properties
  - Item 4: Legal Proceedings
- Part 2: Operations and Disclosures
  - .. Market
  - .. Disclosures
- Part 3: Company Structure
  - Directors
  - Compensation
- Part 4: Financial Statements
  - Statements

Hint: Programming Assignment # 4

- Goal: **make an utterance to intent query mapper** [Name: **prog4-userintent2querymapper**]

- Program **may** do the following – pseudo-code
  - Run in an infinite loop until the user wants to quit
  - Get a user utterance. We will call it u
  - See if u matches to supported intents in Q **// 3 + financial doc info type**
    - Split u into words
    - For each information type – supported query q - in Q
      - Split q into words - w
      - Check how many words of u and w match **// one can also consider partial match**
      - Compute a percentage of match
    - q_i: let this be the query with the highest match percentage
    - If q_i > 0.7 **// 0.7: parameter**
      - Consider it to be the query. Inform user and execute; give information (result)
    - Else
      - Tell user cannot understand u. Example: rephrase and try again.

Programming Assignment # 4

- Code organization
  - Create a folder in your GitHub called "**prog4-userintent2querymapper**"
  - Have sub-folders: src (or code), data, doc, test
  - Write a 1-page report in ./doc sub-folder
  - Put a log of system interacting in ./test
  - Send a confirmation that code is done by updating Google sheet; optionally, send email to instructor

- Use concepts learned in class
  - Classes
  - Exceptions
  - UML Diagrams

Concluding Section

Lecture 17: Concluding Comments

- We looked at the c++ standard library
  - Many types of functionality
  - String, I/O, Mathematical libraries most commonly used
- Remember that many implementations of C++ standard library, usually based on different OS or hardware
  - Implements changing specs
- Be ready to implement one's own (rather than reuse), if necessary, for performance
- Discussed Testing Background

About Next Lecture – Lecture 18

Lecture 18: C++ Standard Libraries

- Advanced concepts on pointers
- Advanced concepts on testing strategies
- HW #4 – will be reviewed

12	Feb 15 (Th)	OO – Constructor, Destructor	Prog 2 – end
13	Feb 20 (Tu)	Review: inheritance, Polymorphism	Prog 3 - start
14	Feb 22 (Th)	In class test	Quiz 1 – In class
15	Feb 27 (Tu)	In class Project Review: PA1 and PA2	
16	Feb 29 (Th)	OO – operators, access control	Prog 3 - end Semester - Midpoint
	Mar 5 (Tu)		Spring break – No class
	Mar 7 (Th)		Spring break – No class
17	Mar 12 (Tu)	C++ standard library, Testing strategies	Prog 4 - start
18	Mar 14 (Th)	Advanced: Pointers	HW 4 due
19	Mar 19 (Tu)	Advanced: Pointers, I/O	
20	Mar 21 (Th)	Advanced: Operator overloading	Prog 4 – end (March 26, 2023)