CSCE 240: Advanced Programming Techniques

Lecture 17: (C++) Testing Strategies, HW 5 (given), PA 4(start)

PROF. BIPLAV SRIVASTAVA, AI INSTITUTE

14^{TH} MARCH 2023

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

- Main Section
  - Concept: Testing strategies
  - Concept: C++ considerations
  - Task: HW 5 – details (due March 17, 2022)
  - Task: Project – PA #4 starts - details

- Concluding Section
  - About next lecture – Lecture 18
  - Ask me anything

Introduction Section

Recap of Lecture 16

- Reviewed HW#4
- We looked at the concept of operators
  - Many types: right sidebar
  - Precedence order when evaluating

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>

Project Updates

- PA-2 and PA-3 graded
- Document how someone should replicate your code to test
  - Otherwise, test log (documentation) may not be seen completely
  - Data should be available, or steps mentioned to obtain data.
- Put test log (documentation)
  - Do extensive testing.

Main Section

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

Example – Calculating Fibonacci Number

- Concept in mathematics:
  - Fibonacci number of a number is the sum of F numbers of its two predecessors
  - Credit: https://en.wikipedia.org/wiki/Fibonacci_number

  - Popularized by Fibonacci around 1200 AD, known before in India as early as 450 BC

F_0	F_1	F_2	F_3	F_4	F_5	F_6	F_7	F_8	F_9	F_{10}	F_{11}	F_{12}	F_{13}	F_{14}	F_{15}	F_{16}	F_{17}	F_{18}	F_{19}	F_{20}
0	1	1	2	3	5	8	13	21	34	55	89	144	233	377	610	987	1597	2584	4181	6765

Implementing and Testing in C++ (V1)

```
int fibonacci(int n)

{

   return fibonacci(n-1) + fibonacci(n-2);

}
```

What can be wrong ?

F_0	F_1	F_2	F_3	F_4	F_5	F_6	F_7	F_8	F_9	F_{10}	F_{11}	F_{12}	F_{13}	F_{14}	F_{15}	F_{16}	F_{17}	F_{18}	F_{19}	F_{20}
0	1	1	2	3	5	8	13	21	34	55	89	144	233	377	610	987	1597	2584	4181	6765

Implementing and Testing in C++ (V2)

```
long fibonacci(unsigned int n)
{
   if (n < 2) return n;
   return fibonacci(n-1) + fibonacci(n-2);
}
```

Fixed for handling

- Negative numbers
- Larger return type

But may take too long

Implementing and Testing in C++ (V3) With Measuring Time

```
long fibonacci(unsigned int n)

{

    if (n < 2) return n;

    return fibonacci(n-1) + fibonacci(n-2);

}


int main ()

auto start = std::chrono::steady_clock::now();  // measures start time
long result = fibonacci(n);                     // calls function
cout << "f(" << n << ") = " << result << '\n’;  // prints result
auto  end = std::chrono::steady_clock::now(); // measures end time

 // prints time elapsed
```

Fixed for handling

- Negative numbers
- Larger return type

Reports time

* But time includes printing time

Code sample:
https://github.com/biplav-s/course-adv-proglang/blob/main/sample-code/CandC%2B%2B/Class17and18_TestingAdvPointers/src/Class17and18_TestingAdvPointers.cpp

Implementing and Testing in C++ (V4) With Measuring Time

```
long fibonacci(unsigned int n)

{

    if (n < 2) return n;

    return fibonacci(n-1) + fibonacci(n-2);

}


int main ()

auto start = std::chrono::steady_clock::now();  // measures start time
long result = fibonacci(n);                       // calls function
auto  end = std::chrono::steady_clock::now(); // measures end time

cout << "f(" << n << ") = " << result << '\n’;        // prints result


 // prints time elapsed
```

Fixed for handling

- Negative numbers
- Larger return type

Reports time

Testing Frameworks

- Java
  - JUnit
    - Code sample: https://github.com/biplav-s/course-adv-proglang-s23/tree/main/sample-code/Java/L17_Testing/src/autotest

- C++
  - Boost.Test
  - Google Test

Home Work 5

Due Thursday, March 16, 2022

Home Work (#5) – C++ - Background

- A *factorial* is a function that multiplies a number by every number below it. For a number N, it is denoted N!
  - Example: 4! = 4 x 3 x 2 x 1 = 24
- Factorial notation is used in many problems dealing with permutations and combinations
- Note:
  - 0! = 1
  - 1! = 1

- *Combination*: Number of ways **r** items can be selected from a set of size **n** where the order of picking does not matter
  - Example: Handshakes between 6 people = C^{6}_{2}
  - = (6!) / (2! * 4!) = (6 * 5 * 4!) / (2! * 4!) = 15
- Note:
  - r is smaller than n

$${}_nC_r = \frac{n!}{r!(n-r)!}$$

Credit: https://en.wikipedia.org/wiki/Combination

Home Work (#5) – C++ - Requirement

- So, write a program named: ***FactorialFun***
- It will support inputs/ arguments in two formats:
  - N: number // to find factorial of N
  - N: number, r: number // to find C^N_r
- Output:
  - Value // computed value
  - Time taken // time for processing

Example invocation

```
>  FactorialFun 4
24
Time for processing: 0.023 seconds


>  FactorialFun 6 2
15
Time for processing: 0.0034 seconds
```

Home Work (#5) – C++ - Code Design

- Create test cases, i.e., input/ output pairs, to test for boundary conditions
- Use exception to handle likely errors – user may give any input

Discussion: Course Project

Course Project – Building and Assembling of Prog. Assignments in Health

- **Project**: Develop collaborative assistants (chatbots) that offer useful information about diseases
- Specifically, use the CDC dataset on diseases at: https://wwwnc.cdc.gov/travel/diseases
  - For polio, it is: https://wwwnc.cdc.gov/travel/diseases/poliomyelitis
  - Each student will choose two diseases (from 47 available).
  - Each student will also use data about the disease from WebMD. Example for polio - https://www.webmd.com/children/what-is-polio
  - Programming assignment programs will: (1) extract data about a disease from two sites, (2) process it, (3) make content available in a command-line interface, (4) handle any user query and (5) report on interaction statistics.
- *Other sources for disease information are possible. Example – NIH https://www.ninds.nih.gov/health-information/disorders*

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 and
- Prog 5: report statistics on interaction of a session, across session

Review of Assignments PA1,PA2, PA2 - Feedback

- Do not put *.class, a.out or .exe in git; it is a binary
- Put a Readme.md or Readme.txt in your assignment's main directory so that the reviewer knows what is the main file, where is the data, how is your program invoked, etc
- Avoid hardcoding in code
  - Paths an absolute no-no
  - Data based string extraction
    - [Students have hardcoded line number, character offset, or simply written values in code (manual extraction). Regex hardcoding most common.]
    - Will make code hard to generalize; no one else will be able to reuse
    - Regex makes extraction easy to understand and simpler
    - Loading extraction logic (regex, string indexes) from a config file makes code easy to generalize

Externalizing Extraction Logic From Code

Loading extraction logic (regex, string indexes) from a config file makes code easy to generalize

Configuration file (Data)

```
# Format: entity name, regex pattern
# Format: entity name, line, start index, end index
Malaria, (M|m)alaria
Phone-number, 13, 23, 47
```

Code

```
1. Read configuration file
2. Read data stream
3. For each pattern
       extract entity value from data stream
4. Close files
5. # Do rest of the processing
```

Now, to extract a new pattern or change extraction rule, we just have to modify the configuration file!

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
  - Match to the closest supported query (I1-I12) and a confidence estimate
  - If confidence greater than a threshold
    - 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 the disease", "What is ***malaria***?" => (Type-I1)
    - "What can I do after travel?" => (Type-I4)
    - "what is the treatment? " => (Type-I10)
    - "Tell me about ***malaria*** vaccine" => (Type-12)
    - …
    - "**Tell me everything**" => ***Give all information extracted***

14 intents: I1 to I12, tell everything and quit

Programming Assignment # 4

- Goal: **make an utterance to query [**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 queries in Q **// 14 until now**
    - Split u into words
    - For each query q in Q
      - Split q into words
      - 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 and TA

- Use concepts learned in class
  - Exceptions

Concluding Section

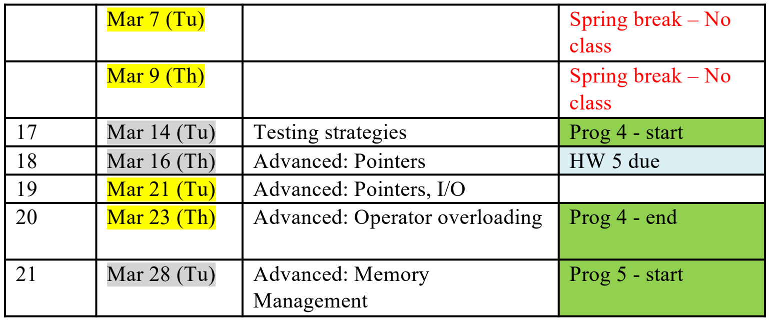
Lecture 17: Concluding Comments

- We looked at common testing types
- Considered an example and different pitfalls
- Gave HW5, due on Thursday (March 16, 2023)
- Gave PA 4, due on Thursday (March 23, 2023)

About Next Lecture – Lecture 18

Lecture 18: Advanced - Pointers

- Pointers
  - Pointer management
  - Function pointers
  - Shared pointers
- HW #5 review



	Mar 7 (Tu)		Spring break – No class
	Mar 9 (Th)		Spring break – No class
17	Mar 14 (Tu)	Testing strategies	Prog 4 - start
18	Mar 16 (Th)	Advanced: Pointers	HW 5 due
19	Mar 21 (Tu)	Advanced: Pointers, I/O	
20	Mar 23 (Th)	Advanced: Operator overloading	Prog 4 - end
21	Mar 28 (Tu)	Advanced: Memory Management	Prog 5 - start