AI INSTITUTE #AIISC
UNIVERSITY OF SOUTH CAROLINA

UNIVERSITY OF SOUTH CAROLINA

CSCE 240: Advanced Programming Techniques
Lecture 19: Advanced Pointers, Input/ Output

PROF. BIPLAV SRIVASTAVA, AI INSTITUTE

19TH 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 19

- Introduction Section
  - Recap of Lecture 18
- Main Section
  - Concept: Pointer arrays
  - Concept: Function Pointers
  - Concept: Buffering
  - Task: Project – PA #4 ongoing – check on issues
- Concluding Section
  - About next lecture – Lecture 20
  - Ask me anything

Introduction Section

Recap of Lecture 18

- We looked pointers and references
  - Pointers are useful for dynamic behavior - memory management, function invocation, …
  - Pointers and references
  - Pointer arrays
  - Pointer based swapping of numbers and user-defined types
- Reviewed HW4
- Checked on PA4, due on Thursday (March 21, 2024)

Main Section

Concept: Pointers – Advanced (Contd.)

Function Pointers

- Functions can be treated as data
  - Passed using pointers
  - Selected dynamically and iterated
- Example
  - **int** (*f_ptr)(**int**, **int**); // declaring a function variable
  - f_ptr = &add; // assigning a value, i.e., function – add here - which matches the function signature
    // i.e., arguments and return type
  - f_ptr(a, b) // invoking the function

Function Arrays

- Group of functions can be manipulated in an array

- Example
  - **int** (*f[3])(**int**, **int**); // Declaring variable

  - f[0] = &add; // Assigning
  - f[1] = &multiply; // Assigning
  - f[2] = &subtract; // Assigning

  - f[i](a, b) // Invoking

Review: Pointers and Examples

- int *a; // a is a pointer to int
- int **a; // a is a pointer to a pointer to a
- int *a[10]; // a is an array of size 10 of pointer to integers
- int (*a)[10]; // a is a pointer to an array of size 10 to integers
- char *(*fp)(int, float *); // fp is a pointer to a function, passing an integer and a pointer to a float,
  // returning a pointer to a char

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

Arguments: 1 through 6

Practical Advice: http://c-faq.com/decl/spiral.anderson.html

Tip for Deciphering Pointer Statements

There are three simple steps to follow:

Starting with the unknown element, move in a spiral/clockwise direction; when ecountering the following elements replace them with the corresponding english statements:

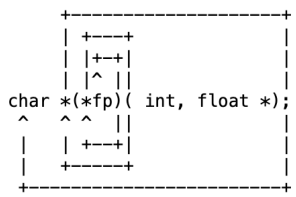
1. [X] or [] => Array X size of... or Array undefined size of... (type1, type2) => function passing type1 and type2 returning... * => pointer(s) to...
2. Keep doing this in a spiral/clockwise direction until all tokens have been covered.
3. Always resolve anything in parenthesis first!

Credit - Practical Advice: http://c-faq.com/decl/spiral.anderson.html

Example #1: Simple declaration

``str is an array 10 of pointers to char''

Example #2: Pointer to Function declaration



``fp is a pointer to a function passing an int and a pointer to float returning a pointer to a char''

Further Exploration

- Tutorials
  - https://www.cplusplus.com/doc/tutorial/pointers/
  - https://www.cprogramming.com/tutorial/function-pointers.html

- Books
  - The Annotated C++ manual, https://www.stroustrup.com/arm.html
  - The C++ Programming Language (4th Edition), Addison-Wesley ISBN 978-0321563842. May 2013, https://www.stroustrup.com/C++.html
  - Fundamentals of C++ Programming , by Richard L. Halterman https://archive.org/details/2018FundamentalsOfCppProgramming/page/n333/mode/2up

Concept: Adv Testing Strategies

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

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

Class Programming Exercise

Fibonnaci numbers (FN) with pointers: write a program which given an input number **n**, will put the FN(n) in its location

```
void fibonacci(int *n)

{

   // Old code: return fibonacci(n-1)
                          + fibonacci(n-2);

}
```

Concept: Adv. I/O - Buffering

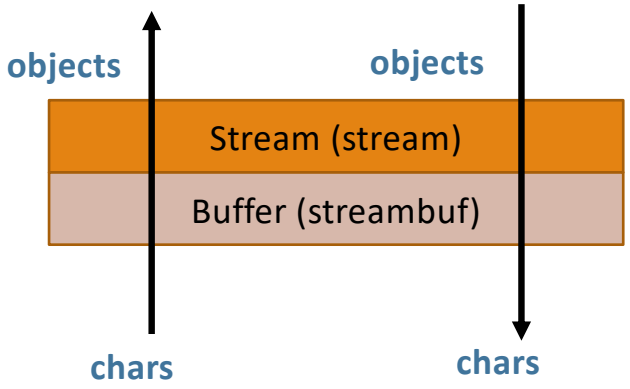
I/O and Memory Organization

- Computer has access to both memory (temporary storage) and disk (permanent storage)
- Properties
  - Faster to write data to memory than to disk.
  - Faster to write one block of N bytes to disk in a single operation than it is to write N bytes of data one byte at a time using N operations

Credit: Fundamentals of Programming C++, Richard L. Halterman

Why Buffer Input or Output

- Improve performance by leveraging characteristics of memory
  - Better to allocate / free memory in storage-appropriate blocks rather than what programmer wants
  - Do it while providing convenient abstraction



- Developer has to be aware of
  - buffer size // impacts I/O performance or memory usage
  - Initial and last values // In case last chunk is less than buffer size
  - Clearing off of the buffer // Affects what is read/ written at the end; flush the values
- Buffered reading/ writing supported in most languages

Code Examples

- Buffering in C style
- Buffering in C++, with streams

Code: https://github.com/biplav-s/course-adv-proglang/blob/main/sample-code/CandC%2B%2B/Class19To22_AdvTopics/src/Class19To22_AdvTopics.cpp

Arguments: 0 through 3

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

Review of Assignments PA1,PA2, PA3 - 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

Suggestion: 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
IBM, (I|i)bm
```

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!

Concluding Section

Lecture 19: Concluding Comments

- We looked at function pointers and function arrays
- We looked at testing strategies and considerations in a small problem - FN
- Re-looked at I/O and discussed buffering
- Reviewed PA4

About Next Lecture – Lecture 20

Lecture 20: Advanced: Operator Overloading

- Adv I/O
  - Buffered writing
- Adv: operator overloading
- Prog 4 ends

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)