CSCE 240: Advanced Programming Techniques
Lecture 20: Advanced Input/ Output, Operators

PROF. BIPLAV SRIVASTAVA, AI INSTITUTE

23RD 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 20

- Introduction Section
  - Recap of Lecture 19
- Main Section
  - Concept: Buffering continued
  - Concept: Operator overloading
  - Task: Project – PA #4 due
- Concluding Section
  - About next lecture – Lecture 21
  - Ask me anything

Introduction Section

Recap of Lecture 19

- We reviewed HW 5
- We looked at pointers
  - Pointers and references
  - Pointer arrays
  - Pointer based swapping of numbers and user-defined types
- Checked on PA 4, due on Thursday (March 23, 2023)
  - **Now extended to Sunday (March 26, 2023)**

Announcements

- New course from Fall 2023

  CSCE 581 - Trusted Artificial Intelligence (3 Credits)
  AI Trust – responsible/ethical technology, fairness/ lack of bias, explanations (XAI), machine learning, reasoning, software testing, data quality and provenance, tools and projects.

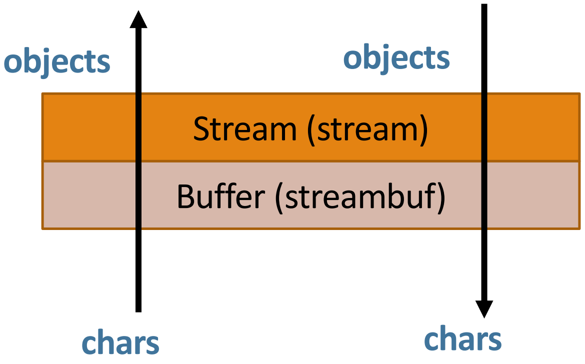
  Prerequisites: C or better in CSCE 240 and CSCE 350
  Prerequisite or Corequisite: D or better in CSCE 330

Main Section

Concept: Adv. I/O – Buffering (Continued)

Why Buffer Input or Output

- 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


objects

objects

Stream (stream)

Buffer (streambuf)

chars

chars

- 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

Operations on Stream

- Position
  - **get**: position of the next character to be fetched into the sequence (extraction)
  - **put**: position of the next character to be deposited into the sequence (insertion)
- Operations
  - **seek**: move pointer with a given offset
  - **tell**: inform about the position of pointer

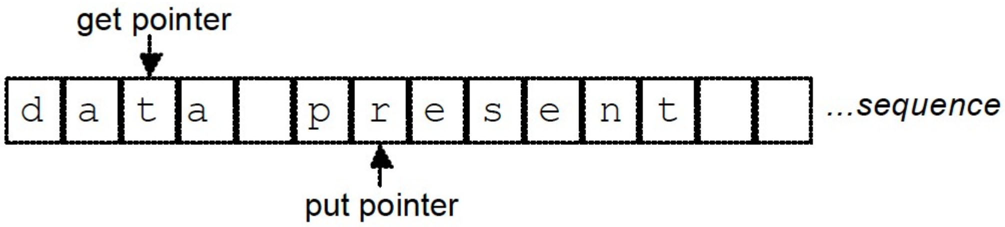


Image credit: C++ Essentials, Sharam Hekmat

Code Examples

- Steam write operations (option – 4)
- Reading and writing
  - with no buffering (option – 5)
  - with buffer size same as file length; extremely memory efficient (option – 6)

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

Discussion on Streams and Buffers

- Streams give a very convenient interface over I/O
  - Hides details of the physical systems (disks, displays, printer, string, web-connected resource)
  - But performance can be a challenge
- Buffers give a way to manage performance
  - Relies on differential speeds of access of I/O devices
  - Design issues about size of buffers, practical issues of initialization of content, flushing content (write situation)

Concept: Operator Overloading

Operator Overloading – What

- Overloading happens when we have multiple functions of the same name
  - Functions distinguished by signature, i.e., parameters and return types
  - Constructors are the common form of overloaded functions

- Operator overloading
  - When operators are overloaded
  - Examples: <<, >>, [], +, - , …

Operator Overloading - Why

- Commonly used with user defined types / classes
- Provide convenience to user, improve usability
- Avoid meaningless / error-prone behavior, especially when operator behavior is inherited due to class hierarchy

Example 1 – Strings

- Suppose you are working with text. Can be in any language.
  - You want to refer to strings and their relationships to each other
  - **Example**: combining two strings
- String representation:
  - Array of characters
- Operation
  - +, -, …

Example 2 – Point and Operations

- Suppose you are working in Geometry. Can be in any dimension.
  - You want to refer to points and their relationships with each other
  - **Example**: a point that is twice away from another point, with respect to a reference
- Point representation: 2-D: Cartesian Geometry
  - (x, y)
  - (angle, distance)
- Operation
  - +, -

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

Argument: 7

Class Exercise – 10 Mins

- Implement operators
  - * with a Point argument: multiples x and y of two points (self and argument) respectively, respectively
  - ^ with an int argument: multiples x and y of the point passed argument

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 [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 + 2 more) 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 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
  - Exceptions

Concluding Section

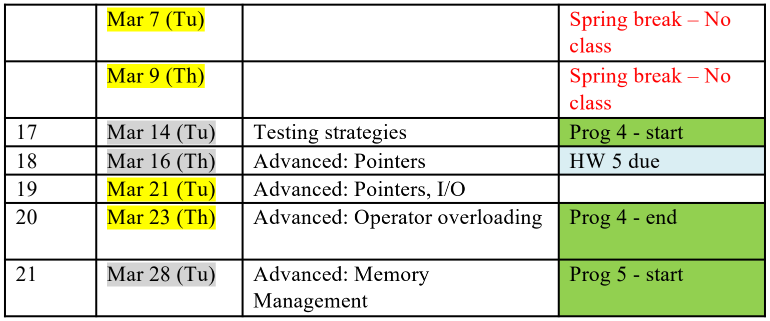
Lecture 20: Concluding Comments

- We looked at buffering for inputs and outputs
- We looked at operator overloading
- Both useful across OO programming languages
- PA4 due

About Next Lecture – Lecture 21

Lecture 21: Advanced: Memory Mgmt

- Fixed memory
  - Vectors
  - Arrays
- Dynamic memory
  - List
  - User defined types
- Freeing memory
- PA 5 starts



	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