

CSCE 240: Advanced Programming Techniques

Lecture 19: Advanced Pointers, Input/ Output

PROF. BIPLAV SRIVASTAVA, AI INSTITUTE

21ST 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 19

- Introduction Section
 - Recap of Lecture 18
 - Class Pulse Survey
- 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 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)

Course Mid-Point Pulse Survey

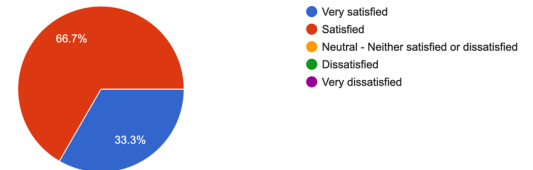
- a) Do you like the pace of the course ? - **Y**
- b) Do you like the content on which the course is focusing? - **Y**
- c) Should the number of HWs be reduced? - **N**
- d) What more topic(s) will you like to be covered? - [Open ended]
- e) Want changed? Any other feedback? - [Open ended]

Pointers, multi-threading, [and additionally - AI and ML]

HWs on Thurs/review on Tues (more time), grading of HW, somewhat more focus on coding,

“I like the class”, “This course is structured well, I feel like I am learning concepts while working towards a larger goal.”

How satisfied are you with the course?
3 responses



Participation: 50%

Actions on Survey

- Course process
 - HW6 will be on a Thursday
- Material changes
 - One lecture on AI/ML
 - Class-appropriate new material on multi-threading

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>

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. I/O - Buffering

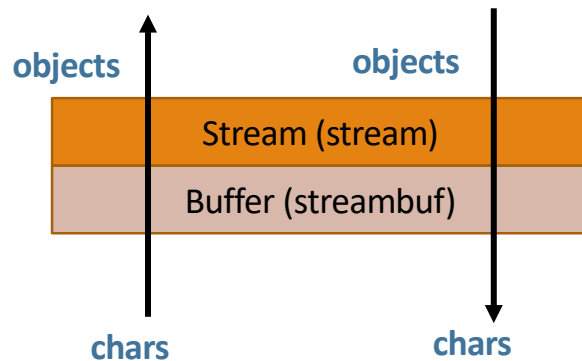
I/O and Memory Organization

- 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 – 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, user 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-I2)
- ...
- **"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 19: Concluding Comments

- We looked at class survey results and made some changes
- We looked at function pointers and function arrays
- Re-looked at I/O and discussed buffering
- Checked on PA4, due on Thursday (March 24, 2022)

About Next Lecture – Lecture 20

Lecture 20: Advanced: Operator Overloading

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

	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