AI INSTITUTE #AIISC
UNIVERSITY OF SOUTH CAROLINA

UNIVERSITY OF SOUTH CAROLINA

CSCE 240: Advanced Programming Techniques
Lecture 21: Adv. Memory Management

PROF. BIPLAV SRIVASTAVA, AI INSTITUTE

28TH 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 21

- Introduction Section
  - Recap of Lecture 20
  - News / announcements / clarifications
- Main Section
  - Concept: Memory in a program
  - Concept: static memory management, vector
  - Concept: dynamic memory management
  - Task: Project – PA #5 starts
- Concluding Section
  - About next lecture – Lecture 22
  - Ask me anything

Introduction Section

Recap of Lecture 20

- We looked at
  - buffering of inputs and outputs
  - operator overloading
- Explained importance in building practical programs regardless of computer language

Student Question – "How Well Am I Doing Presently" ?

- Data available:
  - 3 graded PA (300 points),
  - 5 peer graded HWs (XXX points)
  - 1 Quiz (100 points)
  - Total points as of now: out of 400 points

Not sufficient data to give exact answer right now!

A = [900-1000]
B+ = [850-899]
B = [800-849]
C+ = [750-799]
C = [700-749]
D+ = [650-699]
D = [600-649]
F = [0-599]

Student Assessment Guideline

Tests	**1000 points**
Course Project: programming assign.(5) and report, in-class presentation	600 points
Class Participation and Home Work	200 points
Quizzes and Exams	200 points
Total	1000 points

Student Question – "How Well Am I Doing Presently" ?

- Data available:
  - 3 graded PA (300 points),
  - 5 peer graded HWs (XXX points) – we will do best of 4 of 6 later
  - 1 Quiz (100 points)
  - Total points as of now: out of 900 points

We used formula for an estimate:
[300] * 0.6 + [400] *0.2 + [100] * 0.2

Added column in BlackBoard: Current_weighted_score
- Max possible 400
- HW is unknown; will be 400 if you do at least 4
- Bonus marks not added

```
                                  // Best estimate
A   =   [900-1000]   // 270 - 300
B+  =   [850-899]    //
B   =   [800-849]    // 240 -
C+  =   [750-799]
C   =   [700-749]    // 210 -
D+  =   [650-699]
D   =   [600-649]    // 180 -
F   =   [0-599]
```

Main Section

Concept: Memory in a Program

Concepts Related to Program Memory

- Two type of content stored
  - **Code** (compiled, object): instructions about what is to be done
  - **Data**: variables and values.
- Three ways data is stored
  - **Common data:** global and static variables
  - **Heap**: a program has some memory set-aside by the operating system from which it can dynamically draw memory (with new operation) and free (with delete operation) as well. The size of the heap reduces and grows with these operations. Memory has to be explicitly freed.
  - **Stack**: local variables and function parameters live. When a function is called, space for local variables and parameters is allocated from stack and it is freed when the function returns. The size of the stack grows and shrinks during the program's execution as various functions execute, including recursively.

Illustration

- **Heap**:
- **Stack**: a, b

```
// Add the numbers and return the value
int addNumbers(int a, int b) {
return a + b;
}
```

- **Heap**:
- **Stack**: *a, *b, sum

```
int addNumbersAtFirstLocation
                    (int *a,
                     int *b) {
int sum = *a + *b;
*a = sum;

return sum;
}
```

Illustration

- **Heap**: buffer
- **Stack**: in_file_name,
  out_file_name,
  size, …

If buffer is not freed, it leads to memory leak

```
// Demonstrate reading and writing of file with buffers
void demoReadWriteBuffersFile() {
cout << "\n*** DEMO of reading from and writing to file using buffers ***\n\n";
// https://www.cplusplus.com/reference/ostream/ostream/write/
string in_file_name = "data/input.txt";
string out_file_name = "data/output.txt";
string line;

ifstream in_myfile(in_file_name);
ofstream out_myfile(out_file_name);
// get size of file
in_myfile.seekg (0,in_myfile.end);
long size = in_myfile.tellg();
cout << "Info: size of input file - " << size << endl;

// Now move to the beginning to start reading
in_myfile.seekg (0);

// allocate memory for file content
char* buffer = new char[size];

// read content of infile
// write to outfile

// release dynamically-allocated memory
delete[] buffer;

// Close files and flush content
out_myfile.close();
in_myfile.close();
}
```

Be Aware of Common Problems

- Type of memory being used
- How much memory will be used
- When will the memory be allocated and de-allocated
- Look out for variable initialization and values at any stage
- Do not perform repeated operations
  - Allocation: memory leak (if previous memory not freed)
  - De-allocation: logical error; can cause access exceptions

Static Memory

- Global variables
- Static variables
- Arrays
- Vectors: sequence container like arrays, but it can change size dynamically; storage management is handled automatically by the container / runtime

Credit: For vector, https://www.cplusplus.com/reference/vector/vector/

Vector: Important Methods

- size() – Returns the number of elements in the vector.
- max_size() – Returns the maximum number of elements that the vector can hold.
- capacity() – Returns the size of the storage space currently allocated to the vector expressed as number of elements.
- resize(n) – Resizes the container so that it contains ‘n’ elements.
- shrink_to_fit() – Reduces the capacity of the container to fit its size and destroys all elements beyond the capacity.
- empty() – Returns whether the container is empty.

Code Demonstration

demoVector () - option 8

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

Dynamic Memory

Caution with delete() - 1

- The delete operator never should be used to free up memory that was not allocated by a previous call to new.

```
int list[10];
// ...
*p = list; // p points to list
// ...
// ...
delete [] p; // Logic error, attempt to deallocate p's memory
```

Problem: program will access another space and mess up with runtime

Credit: Fundamentals of Programming C++, Richard L. Halterman, Chap. 18

Caution with delete() - 2

- delete must not be used to deallocate the same memory more than once

```
int *p = new int[10];
 *q = p; // q aliases p
// ...
// Do some stuff with p and/or q
// ...
delete [] p; // Free up p's memory
// ...
// Do some other stuff
// ...
delete [] q; // Logic error, q's memory already freed!
```

Problem: program will access another space (in heap here)

Credit: Fundamentals of Programming C++, Richard L. Halterman, Chap. 18

Caution with delete() - 3

- Memory previously deallocated via delete should never be accessed

```
int *list = new int[10];
// ...
// Use list, then
// ...
delete [] list; // Deallocate list's memory
// ...
// Sometime later
// ...
int x = list[2]; // Logic error, may sometimes works!
```

Problem: unexpected behavior

Credit: Fundamentals of Programming C++, Richard L. Halterman, Chap. 18

Summary: Caution with delete()

- The delete operator never should be used to free up memory that was not allocated by a previous call to new.
- delete must not be used to deallocate the same memory more than once
- Memory previously deallocated via delete should never be accessed

Credit: Fundamentals of Programming C++, Richard L. Halterman, Chap. 18

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 sessions **[prog5-sessionlogger]**

Objective in Programming Assignment # 5:
Record what happens in a chat session and provide summary

- A user may interact with your chatbot for one question or twenty. How did your chatbot do?
- **Record chat your system makes with each user and report on user session as well total usage statistics (since the chatbot was created)**

Approach Suggested

- Under data folder,
  - have a sub-folder called **chat_sessions**
    - When a person starts a chat session (i.e., starts your program and until does not quit), create a file with the "<data>_<time>.txt" as the name. Save the user's utterance and the system's reply there in the order they come. Close this file when the user session ends.
    - Calculate statistics: # user_utterance, #system_utterance and time duration of session
  - have a file called **chat_statistics.csv**.
    - Have a header with columns: S.No, chat_file, # user_utterance, #system_utterance and time taken
    - For each chat file in chat_sessions, there will be a row with the chat statistics you have calculated

Objective in Programming Assignment # 5:
Record what happens in a chat session and provide summary

Approach Suggested

- Under data folder,
  - have a sub-folder called **chat_sessions**
    - When a person starts a chat session (i.e., starts your program and until does not quit), create a file with the "<data>_<time>.txt" as the name. Save the user's utterance and the system's reply there in the order they come. Close this file when the user session ends.
    - Calculate statistics: # user_utterance, #system_utterance and time duration of session
  - have a file called **chat_statistics.csv**.
    - Have a header with columns: S.No, chat_file, # user_utterance, #system_utterance and time taken
    - For each chat file in chat_sessions, there will be a row with the chat statistics you have calculated

- Goal: report statistics on interaction of a session, across sessions [Name: **prog5-sessionlogger**]
- One can invoke it with arguments
  - **prog5-sessionlogger –summary**
    - **There are 12 chats to date with user asking 23 times and system respond 24 times. Total duration is 456 seconds.**
  - **prog5-sessionlogger –showchat-summary 2**
    - **Chat 2 has user asking 2 times and system respond 2 times. Total duration is 4 seconds.**
  - **prog5-sessionlogger –showchat 2**
    - **Chat 2 chat is:**

      …
  - **prog5-sessionlogger –showchat 200**
    - **ERROR: there are only 12 chat sessions. Please choose a valid number.**

Programming Assignment # 5

- Code organization
  - Create a folder in your GitHub called "**prog5-sessionlogger**"
  - Have sub-folders: src (or code), data, doc, test
  - Have data directory as shown in previous slide
    - ./data/**chat_sessions/**
    - ./data/ **chat_statistics.csv**
  - 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
  - File operations
  - Dynamic memory

Class Exercise – 10 Mins

- Think about memory management and buffering need in PA5
  - Size of utterances (by user, by system)
  - Size of files
    - **chat_statistics.csv**
    - Number and size of chat sessions
- Any other considerations ?

Concluding Section

Lecture 21: Concluding Comments

- We looked at memory management
  - Different types of memories available to a program
  - Summary of vector
  - Care to be taken with deletes
- PA5 due April 4, 2023 (Tuesday)

About Next Lecture – Lecture 22

Lecture 22: Advanced: Coding for Efficiency

- Memory optimization
- Runtime optimization
- Code maintenance ease

20	Mar 23 (Th)	Advanced: Operator overloading	Prog 4 - end
21	Mar 28 (Tu)	Advanced: Memory Management	Prog 5 - start
22	Mar 30 (Th)	Advanced: Code efficiency	
23	Apr 4 (Tu)	Advanced: Templates	Prog 5 - end