

# CSCE 240: Advanced Programming Techniques

## Lecture 23: Templates

---

PROF. BIPLAV SRIVASTAVA, AI INSTITUTE

2<sup>ND</sup> APRIL 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 23

---

- Introduction Section
  - Recap of Lecture 22
  - News / announcements / clarifications
- Main Section
  - Concept: Templates
  - Concept: Class Templates
  - Concept: Function Templates
  - Task: Project – PA #5 due
- Concluding Section
  - About next lecture – Lecture 24
  - Ask me anything

# Introduction Section

---

# Recap of Lecture 22

---

- We discussed code optimization considerations
  - Memory optimization
  - Runtime optimization
  - Code maintenance ease
- Looked at examples
  - Sorting
  - Tax preparation software
  - Project

# Announcement - 1

---

- McNair Junior Fellows program: All details and applications are on: <http://www.cec.sc.edu/mjf> | **Deadline April 19<sup>th</sup>, 2024 ! (30 grantees in 2023)**
  - The program, in its 10<sup>th</sup> year since its foundation, and in its 6<sup>th</sup> year as an official CEC program, provides supports for undergraduate students up to 3k\$ in summer funds and runs activities that helps the students further explore research (as well as research posters, state of the art and other research initiation programs).
- Summer Internships
  - You can apply to fellowship and work with faculty ON YOUR IDEA
  - You can work with faculty ON THEIR IDEA and get paid
  - You can work on your idea with a faculty to mentor (with/ without fellowship)
  - Faculty can augment salaries

# Announcement - 2

---

- PA 4 assessed
  - Please follow instructions carefully: not many following it for code organization
- More towards the end of class

# Main Section

---

# Goals of Templates

---

- Generalize on coding best practices
- Improved developer productivity

*Without impacting code runtime*



# Steps for Using Templates

---

- The programmer creates functions with templates
- The compiler (effectively) creates specific functions for different types the functions are invoked with
- The user calls the functions (mostly) seamlessly

# Concept: Function Templates

---

# Simple Template

---

- Code example

Code: [https://github.com/biplav-s/course-adv-proglang/blob/main/sample-code/CandC%2B%2B/Class23\\_Templates/src/Class23\\_Templates.cpp](https://github.com/biplav-s/course-adv-proglang/blob/main/sample-code/CandC%2B%2B/Class23_Templates/src/Class23_Templates.cpp)

- Option 0

Credit: Fundamentals of Programming C++, Richard L. Halterman, Chap. 19

```
template <class T>
bool less_than(T a, T b) {
    return a < b;
}
```

```
template <typename T>
bool less_than(const T& a, const
T& b) {
    return a < b;
}
```

# Medium (-ly Complex) Template

---

- Code example
- Discussion:  
what happens with string?

```
template <typename T>
T sum(const vector<T>& v) {
    T result = 0;
    for (T elem : v)
        result = result + elem;
    return result;
}
```

Code: [https://github.com/biplav-s/course-adv-proglang/blob/main/sample-code/CandC%2B%2B/Class23\\_Templates/src/Class23\\_Templates.cpp](https://github.com/biplav-s/course-adv-proglang/blob/main/sample-code/CandC%2B%2B/Class23_Templates/src/Class23_Templates.cpp)

- Option 1

Credit: Fundamentals of Programming C++, Richard L. Halterman, Chap. 19

# Concept: Class Templates

---

# Class Template

---

- Generic classes which do encapsulation of similar capability
  - Data members
  - Functions
- Commonly used to implement new data structures

Code: [https://github.com/biplav-s/course-adv-proglang/blob/main/sample-code/CandC%2B%2B/Class23\\_Templates/src/Class23\\_Templates.cpp](https://github.com/biplav-s/course-adv-proglang/blob/main/sample-code/CandC%2B%2B/Class23_Templates/src/Class23_Templates.cpp)

- Options 2 and 3

# Code Example

- Code example
- Discussion:
  - what happens with string?
  - With multiple class templates?

Code: [https://github.com/biplav-s/course-adv-proglang/blob/main/sample-code/CandC%2B%2B/Class23 Templates/src/Class23 Templates.cpp](https://github.com/biplav-s/course-adv-proglang/blob/main/sample-code/CandC%2B%2B/Class23%20Templates/src/Class23%20Templates.cpp)

- Option > 3

Credit: Adapted from <https://www.programiz.com/cpp-programming/class-templates>

```
// Class template with single parameter
template <class A>
class SingleClassTemplate {
    private:
        A a;

    public:
        SingleClassTemplate(A aa) : a(aa)
        {} // constructor

        void printValues() {
            cout << "\ta = " << a << endl;
        }
};
```

# Discussion

---

- Templates are meant to increase developer productivity
- One can define for functions or classes
- Can have one or more types
- Compiler generates type-specific code; hence, little-to-no impact on code performance
- Be aware of the initial values and operators being defined for the types
  - Unexpected errors may happen



# Class Exercise – 10 Mins

---

- **Objective:** Discuss where we can use in our homeworks and assignments
- Function templates
- Class templates

# 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 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

# Guidance on Readme

---



# Code Structure for any Assignment

---

- File: Readme.md    // (or Readme.txt or readme.md)
  - // Has information about – who, what, why, how
  - // - In our case: code for course and PA#, author, layout of sub-folders, etc
  - // - (optional) have readme in major sub-dirs as needed
- ./data
  - // Data needed or created by the program
  - // - (optional) have sub-folder for input and output, if both present
  - // - (optional) others as logically needed
- ./doc
  - // Document assumptions, algorithm
- ./src
  - // (or ./code) – Has the program source code
  - // - (optional) divide further into header files, source code,
- ./test
  - // Has information about how to test your code
  - // - a transcript of code's running and its input/ output
  - // - (optional) a code to check the working of code on a test case
  - // - (optional) test cases

# Feedback: Folders Management

---

1. Write a good readme
2. Make your project well-organized
3. Configure the class path well

# Feedback: Feature Implementation Methods

---

## **Input reprocessing**

Step 1: splitting the input into words by identifying space  
Step 2: converting all letters into upper case or lower case  
Step 3: use spell checking to find closest word

**Method 1:** looking up keyword and computing confidence

### **Coding implementation:**

Two loops:

- outer is for determining which question is being asked
- inner: if one question is asked appropriately

Pros: straightforward

Cons: lack of flexibility, slow

**Method 2:** regular expression

Step 1: search based on pre-defined regex

**Method 3:** advanced comparison algorithm

-Levenshtein distance (character-level): it can compare two words distance

e.g., “dise” vs “disease”, “information” vs “info”

**Takeaway:** make sure your program can answer simple questions first, and then try with more sophisticated algorithms.

# Feedback: More on Features

---

## Question rephrasing:

- Requiring the rephrased question can pass confidence;
- interaction with user by giving hint:  
“I assume you are asking for the contact information, Y/N?”

## Confidence computing:

- counting by one point if one word matched;
- different weights  
e.g. allocating more weight to salient keywords  
“disease” or “vaccine”

## Codes reusability:

- Put individual classes into separate head files;
- Think about the reusability issue before coding;

## Data Structure: for storing read answers

- Separate files for each answer type
- 2D array
- Nested dictionary (Jaya, Python)

## Language filter: to deal with non-English character

# Concluding Section

---

# Lecture 23: Concluding Comments

---

- Programming practice for project assignments based on PA#4
- We discussed
  - Templates
  - Class templates
  - Functional templates

# About Next Lecture – Lecture 24

---

# Lecture 24: AI/ ML

- AI as a decision-support
- ML, Deep Learning and now, ChatGPT/ LLM craze
- AI/ML and programming – what to be aware of

19	Mar 19 (Tu)	Advanced: Pointers, I/O	
20	Mar 21 (Th)	Advanced: Operator overloading	Prog 4 – end
21	Mar 26 (Tu)	Advanced: Memory Management	Prog 5 – start HW 5 due
22	Mar 28 (Th)	Advanced: Code efficiency	
23	Apr 2 (Tu)	Advanced: Templates	
24	Apr 4 (Th)	AI / ML and Programming	Prog 5 – end
25	Apr 9 (Tu)	Project code summary – student presentation for reuse Review material for Quiz 2	HW 6 due Prog 6 – assembling start
26	Apr 11 (Th)	In class test	Quiz 2 – In class
27	Apr 16 (Tu)	Project presentation	Prog 6 - due
28	Apr 18 (Th)	Project presentation	Last day of class (April 22 per bulletin)
	Apr 23 (Tu)		Reading Day
29	Apr 25 (Tu)	9am – Final Overview	Examination