Contents

CSCI 1300 Syllabus

1 About The Course

1.1 Course Description

Teaches techniques for writing computer programs in higher level programming languages to solve problems of interest in a range of application domains. Appropriate for students with little to no experience in computing or programming.

1.2 Course Objectives

This class covers the basics of computer programming using C++. The focus will be on understanding the basic components of computer languages. Upon completion of the course, the student can:

- Design and construct algorithms for problem solving by applying processes of abstraction and program decomposition.
- Implement fundamental programming constructs, such as variables, expressions, conditionals, and iterative control structures, in a higher-level language.
- Evaluate and implement simple I/O, such as user input and file I/O, including the necessity for external input to a program and the role of external data storage.
- Design and explain the role of functions in program construction, including an understanding of parameter passing and return values.
- Describe the properties of data types, including the primitive types of numbers, characters, and booleans, as well as more complex data types, such as arrays, records, and strings.
- Use an Integrated Development Environment to produce code that is free of syntactical, logical, and run-time errors. Understand the process of debugging as part of software development.
- Design and create code using object-oriented design methodology, including an understanding of objects and classes, information encapsulation, and efficient class design.
- Use third-party code to accomplish a programming objective. This includes learning to read code written by another individual and modifying the code, or using third-party libraries.
- Develop an understanding that software development is a dynamic, social process, and that learning how to seek out information is a necessary skill for success.

1.3 Course Outline

This is a 4 credit hour course. As such, you can be expected to spend 4 hours per week "in class" (3 in full class lecture, 1 in recitation) and 12 hours per week on additional course-related activities. These activities will include reading sections from the course text, watching pre-recorded topic videos, working on various assignments, engaging with course staff in grading interviews, and other activities.

Brief list of topics to be covered:

- Computer architecture and environment
- Variables and data types
- Variables and operators
- Strings: indexing, iteration, comparing
- Control structures: if/else statements
- Control structures: switch statements
- Control structures: while loops
- Control structures: for loops, iteration
- Functions
- Arrays
- I/O Streams
- File I/O
- Classes and objects
- Classes, functions, methods
- Inheritance
- Vectors
- Recursion
- References

2 Assignments and Grading

2.1 Assignments

Your grade is based on your performance in course activities including recitations, homeworks, quizzes, projects, and exams.

Your grade in the course will be based on the following components:

- **Homeworks (25%)**. There are 8 homework assignments that will asses your knowledge about the weekly concepts taught in lecture.
- **Projects (15%)**. You will complete two projects during the semester. Project 1 will be worth 5% of your grade, and Project 2 will be worth 10% of your grade. These assignments will build upon and integrate the concepts covered in previous coding assignments and give you opportunities to practice your skills on real-world problems.
- **Weekly recitation activities (10%, drop lowest)**. You will complete pre-quizzes before recitation and in-class worksheets during the recitation section. These are graded assignments and may often require working with a partner. Attendance in recitations is required.
- **Midterms and final (40%)**. There will be three midterms and a final exam that will evaluate your overall knowledge of the course material. Dates for these, and topics covered, are listed in the table below. Midterms will be held in-person during lecture, and the final exam will be held in-person during the scheduled final exam time. You must be present in-person for all exams, they will not be offered remotely.
- **Class participation (10%, drop 3 lowest)**. Most activities will take place during lecture. Other activities will be published via Canvas.

2.2 Extra Credit

There will be a number of other opportunities to receive extra credit in the form of additional points on an assignment or within a category. Take advantage of each and every opportunity.

2.3 Submission Policies

All assignments are due by the date and time specified in the assignment. Homework can be submitted up to two days late for 80% of full credit. After two days, no late work is accepted without prior permission of the instructor.

Recitations

Recitation pre-quizzes will include questions to get you adjusted to the content taught in lecture the previous week. Pre-quizzes will be due at **9:00 am on Mondays**. Recitations will then address these questions and provide the correct answers, and then in recitation you will complete a paper worksheet provided by your TA. The worksheets will be due at **11:59 pm on Wednesdays**.

Homeworks

There is a homework or project due every week. Homework will be due at **5:00 pm on Fridays**. Coding assignments will often be given immediate feedback using an auto-grade feature, but may also involve a grading interview process outlined below. Code files should have your name, date, and assignment number included as comments at the top of the file.

There will be a quiz associated with every homework and project. These quizzes will include both theory and code questions. Quiz will be due at 5:00 pm on Sundays.

Homework Late Policy:

All homework assignments, except Homework 0, may be turned in up to 2 days (48 hours) late with a 20% penalty. Late extension homework coderunner assignments will be available at 5:01 pm on Fridays and may be submitted as late as Sunday at 5:00 pm. If a student would have received a 95% had they turned in their homework on time, a late submission will earn them a 75% instead.

Interview Grading:

Following a project submission, each student will sign up for a grading interview about their project. If you are not interviewed for all projects, then you get no credit for those activities. The grading interview will consist of questions about the work you will have submitted the previous week (or two, for projects). These questions are designed to test your understanding of the solution code submitted. They will also serve as an opportunity for you check in about other topics related to the course. It is your responsibility to look on Canvas and sign-up for an interview slot. If you wait, and the interview slots dry-up, then you missed your interview. Sign-up early, do not wait.

If you have to reschedule a grading interview, you must email the TA at least one day in advance. Documented emergency situations will be evaluated on a case-by-case basis.

You are responsible for scheduling the grading interview during the designated grading period, which is usually 1 week. Once the period expires you will not be able to have the grading interview and you will lose all the points for the assignment. You are responsible for finding the room and arriving on time. There is a 1- minute "grace period" for being late, after that it is 10% off for each minute the student is late, at 6 minutes late you get a zero.

Advice: Get to the appointment 5 to 10 minutes early and use the extra time to prepare.

Exams

As discussed in the Assignments section above, there are three midterms and a final in this class that account for 40% of your final grade. **You must average at least 67% on your in order to receive better**

than a D+ in the class, regardless of your scores on the other aspects of the class. Each exam is 10% of your grade and your final exam can be used to replace your lowest midterm grade if it is higher than one of your midterms. The midterms are an individual assessment of your ability to apply the programming skills learned in lectures, recitation assignments, and homework assignments. If you do not show a skill level that is at or above this threshold, you will not receive a grade better than D+ in the course overall. A grade of C- in this class is required to take the next class in the computer science sequence. In addition, you will not get a passing score for the class by just showing up for the exams. You still need to turn in all other class work, including all projects.

Midterm 1	Friday, Oct 4
Midterm 2	Friday, Oct 25
Midterm 3	Friday, Nov 15

2.4 Grading Scale

At the end of the course, letter grades will be assigned via the following formula:

Letter Grade	Percentage Grade
A	93-100
A-	90-92.99
B+	87-89.99
B	83-86.99
B-	80-82.99
C+	77-79.99
C	73-76.99
C-	70-72.99
D+	67-69.99
D	63-66.99
D-	60-62.99
F	<60

3 Course Requirements

3.1 Method of Instruction

Attendance at all class meetings is highly recommended, and attendance in recitations is required.

10% of your grade is participation in lecture, which will include small in-lecture activities that you must be present to complete.

You are responsible for knowing the material presented during class, even if you were not in attendance when the material was presented. Previous experience has shown that students who do not attend class regularly often receive a failing grade and have to repeat the class the following semester.

Read the daily lecture material and watch pre-recorded topic videos before lecture. Lecture sessions will utilize discussion and activities that build on this content, along with the materials from previous days. Lectures will be recorded and posted to Canvas for later use within the course.

Recitation activities will include discussion and problem-solving, as well as provide a chance for addressing questions about course content. You will complete a Canvas quiz during recitation.

Your TA(s) will take attendance for recitation each week. If you need to miss recitation, make arrangements with your TA(s), to attend a different recitation section for the week. You can find the list of TA' s and the times for each recitation on the start here page. If you miss recitation, you will not get credit for the Recitation assignment that week.

Recitation will have similar expectations for lecture. Be professional, be on time, and do not be disruptive.

Good tips for maintaining a professional environment:

- Decide before lecture if you are going to attend.

- If you attend, be professional, be on time, and do not be disruptive.
- Turn off your cell phone.
- Bring your laptop, but restrict its use to class activity.
- Put away newspapers and magazines.
- Refrain from disruptive conversation.
- In general, respect all interactions related to this class and the students that desire to partake.

3.2 Required Texts and Other Materials

Textbook

Brief C++ Late Objects, Third Edition, Cay S. Horstmann, Wiley. 2017. ISBN: 978-1118674260.

This book is available in digital format and print format.

It is presented in an Enhanced E-text format readable on computers, tablets, and Kindle devices through Canvas. All Enhanced E-text sections include many different forms of guidance to help students build confidence and tackle the task at hand, including Self Check and Practice activities along with end-of-section Review Exercises, Practice Exercises and Programming Projects.

Content in the new edition is much improved, and if you choose an older edition you will miss out on the interactive activities and exercises.

Software

We will be using VS Code as our primary Interactive Development Environment (IDE). This installation process is supported in the Getting Started and Week 1 materials of the course.

We will also be using Canvas as our Learning Management System (LMS). Although you only need to do minimal work directly on Canvas, you will need to know how to navigate Canvas to access course materials, see announcements, submit your homework, understand your grades, and view course notes and project descriptions.

You will also need to be expected to communicate and collaborate on Ed. You can post your questions on Ed in the appropriate thread for each week, each assignment and each problem. Your questions will be answered by another student or one of the members of the instructional team. Please also answer any questions that you see on Ed. As a group we are much more efficient than any one individual.

Other Materials

You must have a back-up system. Lost work, internet was broken, computer froze up, etc. are not valid reasons for missing due dates.

For this course it is highly recommended that you get a Dropbox account, a Google Drive account or invest in a USB memory stick, to save your files. Computer crashes absolutely WILL happen and you are responsible for saving and keeping backup copies of your work. Also, Google Drive and Dropbox keep a versioning system. If you accidentally delete your file, you will be able to recover it.

A limited amount of printing may be required in this class. You need to ensure that your printing account has sufficient funds for this. Your initial allocation may deplete quickly, depending on your other printing activities. If this causes problems, please contact the course staff at csci1300@colorado.edu .

4 Collaboration

4.1 Collaboration, Plagiarism, and Honor Code

Collaboration is highly recommended and a valuable skill to develop at this point in your life.

Collaboration is defined as working with other people to ask, and answer, questions about how coding is done. This can be accomplished verbally or with pseudo-code or with pictures or with flow charts or with

discussions about what syntax means... Collaboration is NOT defined as showing someone your source code, or by cutting-and-pasting code, or by a group project where everyone talks but only one person codes, or by using a paid tutor to show you code, or by using internet sites to copy code.

Although you must work with other students to fully understand coding, you must never ever reveal your source code, or copy source code, or leave your browser open, or leave your computer unattended. Stolen code results in ALL involved parties being sent to honor council.

The Computer Science Department at the University of Colorado at Boulder encourages collaboration among students. You are encouraged to work with a partner for some assignments (Final Project), but you are also free to tackle the work alone.

Many forms of collaboration are acceptable and encouraged. In computer science courses, it is usually appropriate to ask others—TAs, LAs, instructor, or other students—for hints and debugging help or to talk generally about problem solving strategies and program structure. In fact, we strongly encourage you to seek such assistance when you need it. Discuss ideas together, but do the coding on your own .

Rule 1: You must not submit or look at solutions or program code that are not your own. Do not search for online solutions. This is not an appropriate way to "check your work," "get a hint," or "see alternative approaches."

Rule 2: You must not share your solution code with other students, nor ask others to share their solutions with you. Do not leave copies of your work on public computers nor post your solution code on a public website.

Rule 3: You must indicate on your submission any assistance you received. It is fine to discuss ideas and strategies, but you should be careful to write your programs on your own.

Be aware: all submissions are subject to automated plagiarism detection. The entire point of the CU Honor Code is that we all benefit from working in an atmosphere of mutual trust. Do not take advantage of that trust. CU employs powerful automated plagiarism detection tools that compare assignment submissions with other submissions from the current and previous quarters. The tools also compare submissions against a wide variety of online solutions. These tools are effective at detecting unusual resemblances in programs, which are then further examined by the course staff. The staff then make the determination as to whether submissions are deemed to be potential infractions of the Honor Code.

To support students in collaboration the Department has created a Collaboration Policy that makes explicit when their collaborative behavior is within the bounds of the Collaboration Policy and when it is actually academic dishonesty, which would be considered a violation of the University' s Honor Code. All students of the University of Colorado at Boulder are responsible for knowing and adhering to the University's Honor Code. Violations of this policy may also include cheating, plagiarism, academic dishonesty, fabrication, lying, bribery, and threatening behavior. Collaboration on homework assignments is allowed and encouraged. Students are most successful when they are working with other students to understand new concepts. The ultimate goal is that you fully understand the code you develop.

Plagiarism includes using material from outside sources (e.g. Internet sources, chatGPT, other AI tools, chegg.com, coursehero.com, a tutor) without clear identification and citation. Unless otherwise specified, you may make use of outside resources (internet, other books, other people), but then you must give credit by citing your sources in the comments inside your code.

Citing examples (assuming // indicates beginning of code comment):

- `// Modified version from https://github.com/Phhere?MOSS-PHP`
- `// Adapted from Program #7.2 in book "Accelerated C++" by Stroustrup`
- `// Worked with Joe Smith from class to come up with algorithm for sorting`
- `//Received suggestions from stackExchange website (see http://⋯)`
- `//Worked with a tutor on the algorithm for the STORE function`

A good rule of thumb: "if it did not come from your brain, then you need to attribute where you got."

Note: you do not need to cite if you are adapting from slides for the course or the required textbook for the course of from the hired staff for the course.

All homework assignments, all quizzes, all labs, and all exams will be required to be completed without outside resources . These will be clearly marked as individual assignments: the Canvas submission is individual. Use of outside resources would violate the collaboration policy.

4.2 Adhering to the Collaboration Policy:

Some examples of violating the collaboration policy include, but are not limited to:

- Sharing a file (source code) with someone else.
- Submitting a file that someone else shared with you.
- Stealing a copy of someone else' s work and submitting as your own, even with modification.
- Copying outside resources.
- Using outside resources and not citing your sources.
- Posting on websites like chegg.com. coursehero.com or craigslist.org soliciting a solution to an (or part of an) assignment.
- Soliciting help with commenting your code also constitutes a violation of the collaboration policy.
- Copying solutions from website like chegg.com, coursehero.com, and other websites.
- Using a solution the "tutor" gave you.

Examples of collaborating correctly :

- Sharing pseudo-code.
- Asking another student for a helpful suggestion. Verbally, not written.
- Reviewing another student' s code for bugs/errors.
- Working together on the whiteboard, or paper, to figure out how to approach and solve the problem. In this case you must include that person' s name in your collaboration list at the top of your submission. This includes working with a tutor.

One way to know you are collaborating well is if everyone is developing the code solution individually. This collaboration policy requires that you be able to create the code, or solve the problem on your own before you submit your assignment. You can brainstorm a solution or algorithm with a friend, but the submitted code should not be the same code.

Even if collaboration is stated, it is a violation of the Honor Code to submit effectively identical code with another student or an outside source.

If two or more students submit the same solution, claiming they have been "working with the same tutor" , that constitutes a violation of the Honor Code.

Any discovered incidents of violation of this collaboration policy will be treated as violations of the University' s Academic Integrity Policy and will lead to an automatic academic sanction in the course and a report to both the College of Engineering and Applied Science and the Honor Code Council. The academic sanction will be an automatic F.

Note: The instructor reserves the right to change the policy and to apply different academic sanctions if the violation justifies it. Students who are found to be in violation of the Academic Integrity Policy can be subject to non-academic sanctions as well, including but not limited to university probation, suspension, or expulsion.

Collaboration boundaries are hard to define crisply, and may differ from class to class. If you are in any doubt about where they lie for a particular course, it is your responsibility to ask the course instructor.

Students that leave their computers unprotected are also subject to course sanctions mentioned above. When similar code is found, ALL parties are subject to sanctions. This includes the person whose code was "taken" . Protect your source code at all times. Do not forget to sign out of public computers or leave your own machine unattended.

5 Communication

Please send all general course questions to: csci1300@colorado.edu

As a member of the CU community you are expected to consistently demonstrate integrity and honor through your everyday actions.

5.1 Professional Email Expectations

Any email correspondence related to the class should be sent from a colorado.edu email address. Please note that we do not read email between 5pm and 9am, or during the weekends. You can expect a response within 24 - 48 hours during the week and within 48 - 72 hours if sent on the weekend.

Send email messages to faculty and staff using a professional format.

Tips for a professional email include:

- Always fill in the subject line with a topic that indicates the reason for your email to your reader.
- Respectfully address the individual to whom you are sending the email (e.g., Dear Professor Smith).
- Avoid email or text message abbreviations.
- Be brief and polite.
- Add a signature block with appropriate contact information.
- Reply to email messages with the previously sent message. This will allow your reader to quickly recall the questions and previous conversation.

5.2 Office Hours

Faculty, TAs, and staff members are very willing to assist with your academic and personal needs. However, multiple professional obligations make it necessary for us to schedule our availability.

Suggestions specific to interactions with faculty and staff include:

- Respect posted office hours. Plan your weekly schedule to align with scheduled office hours. There is a variety of availability throughout the week.
- Avoid disrupting ongoing meetings within faculty and staff offices. Please wait until the meeting concludes before seeking assistance.

Our office hours are structured as group meetings.

Group office hours are on Monday through Friday, where multiple students can join in for help, and one or more members of the instructional team will be available to answer questions.

Please check out our Office Hours page on Canvas for up to date availability.

What are Office Hours?

It is much like a classroom. It is a place to ask questions, explore further, discussion strategies, explore related topics and support your classmates and contribute to the class. Office hours are optional.

How do we use Office Hours?

- Discuss class content directly with the instructor and other students.
- Practice professional collaboration strategies.
- Instructors may use breakout groups or/and guide discussions, at their discretion.
- Instructors may decide which topics will be discussed based on what will optimize learning.
- Please wear appropriate attire and be aware of your environment.

What are Office Hours not?

- Answer forum.
- Tutoring session.
- Instructors cannot help you completely debug your code.
- Instructor may not reply to all inquiries and let other students speak as appropriate.

5.3 Ed

We will use Ed as the center of communication for this course. Whenever you have content related questions about the course, especially ones that you believe other students may also have, you should post these questions on Ed. You can find the link on Canvas.

Week 1: Introduction

Learning Goals

This week you will:

1. Become familiar with the Syllabus, to learn about the structure of the course and its rules and policies
2. Install and become familiar with your Integrated Development Environment (IDE) - Visual Studio Code
3. Be able to identify the parts of a computer and their function in the computer system
4. Write the "hello world" program in C++

1 Background

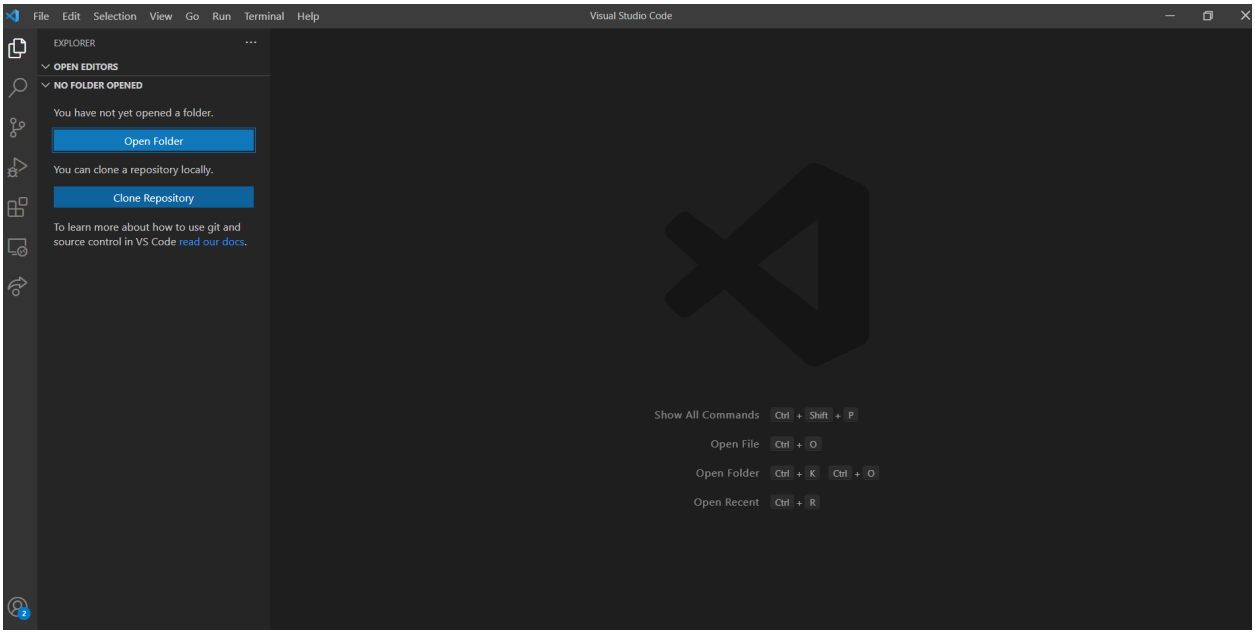
This week will be an introduction to a variety of foundational material for computer science. You will need to learn how to navigate the required software, basic variables, user input, and algorithms/pseudocode.

1.1 Navigating Terminals and Compiling Code

In Visual Studio Code, you can open an integrated terminal, which initially starts at the bottom of your workspace. But what is a terminal?

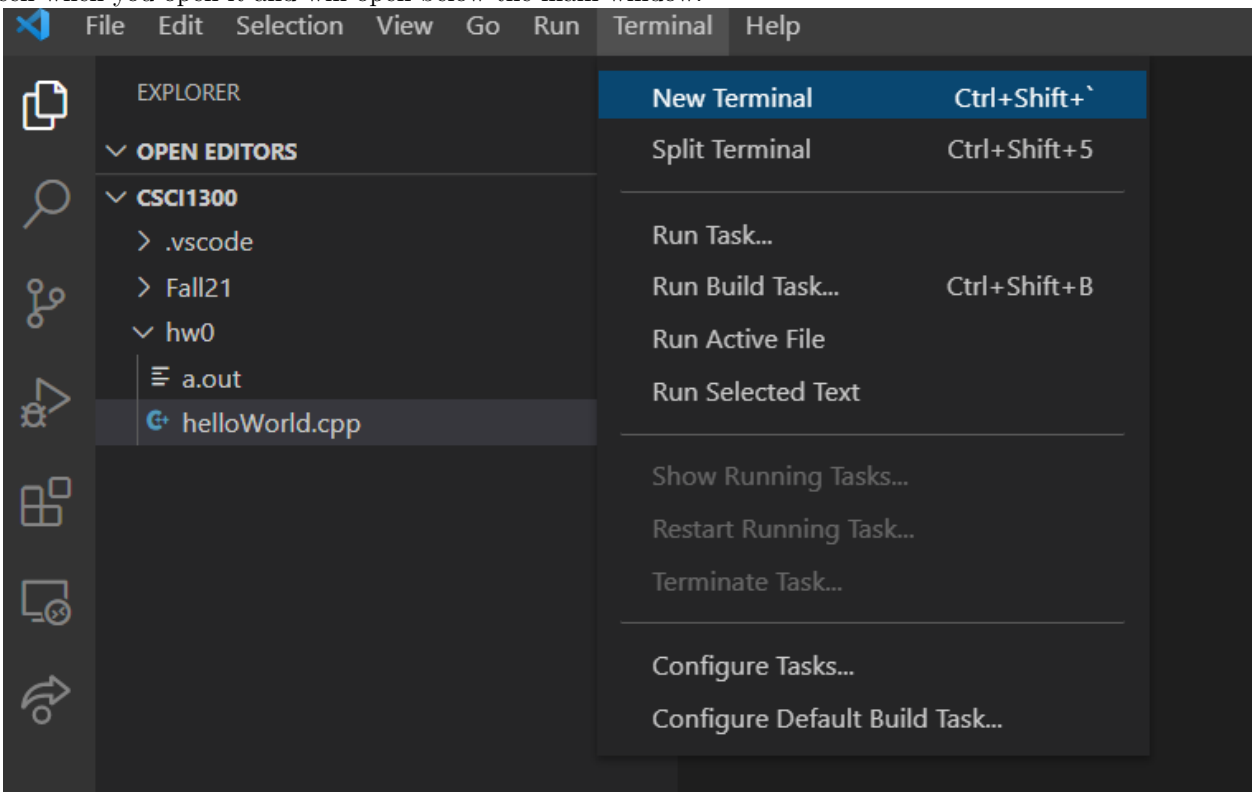
A **terminal** is an **interface** to your computer which allows you to execute any task on the computer directly through **commands**, without the use of a graphical user interface or GUI (like the file explorers on your system that you use to navigate to folders, create files etc). This allows you to directly execute tasks, which is often quicker than using the GUI.

VS Code has a built in terminal that you can open and use to interact with your computer's files and any code you write. To access this terminal, you will need to open VS Code. It may look like this the first time you open it:

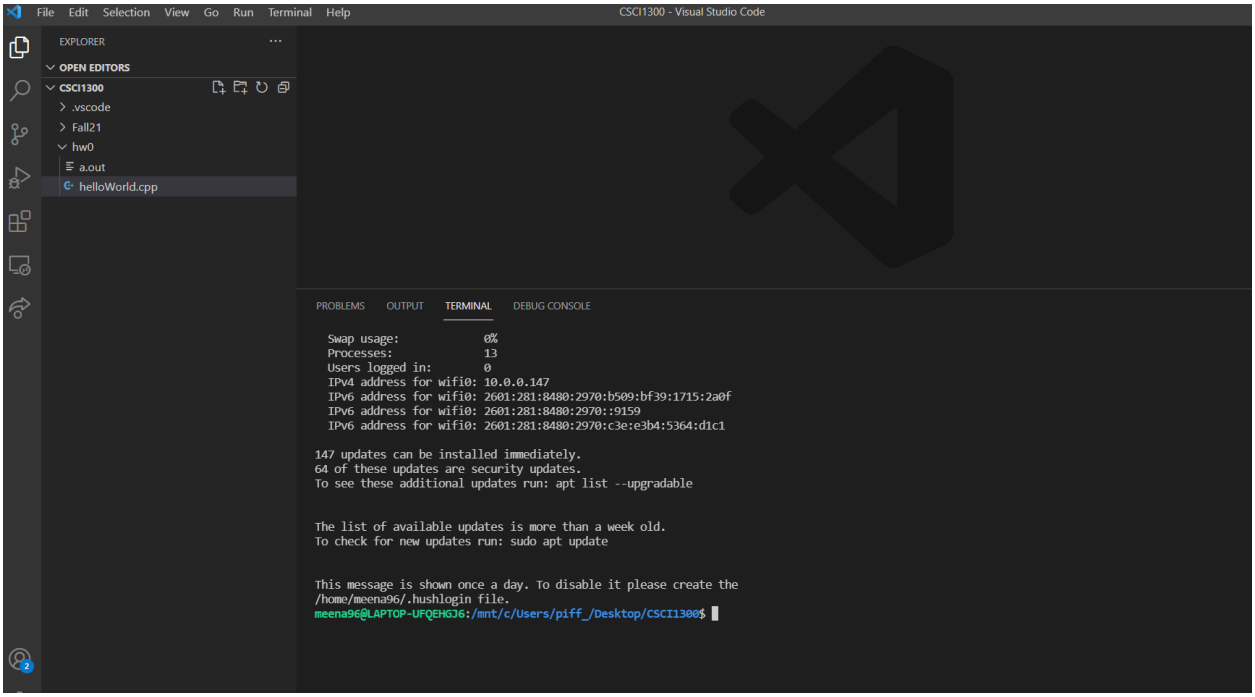


If you have not opened any folders yet in VS Code, you will need to start by opening one. You can do this by clicking "Open Folder" and navigating to whichever folder you would like to use. I would encourage you to make a new folder called "CSCI_1300" to keep all of your course materials organized.

Once you have a folder open, you can open a terminal. To open the Terminal tab, go to the menu bar, locate Terminal and click on it. Select "New Terminal" to open a new terminal. A terminal looks like a dark screen when you open it and will open below the main window.



The new screen should look something like this:



You will see your name and your device name on the terminal tab. Note that this can look different depending on your OS. The general anatomy of a terminal window is as follows:

1. Everything before the : tells you the username and the device name where you are logged in
2. Everything between the : and $ is your current directory (think of a directory as the folder you' re in). Note that you do not need to be in the same directory as the screenshots shown above. Yours will depend on your computer. Mac/Linux and Windows will look different, but both will state the current directory (or "folder") that you are in.
3. $ represents the end of the prompt, after which you can enter a command.

File browsing using the terminal is like using Windows explorer/Finder or clicking on folders and navigating to different folders on your machine.

In the terminal, instead of clicking on folders we use text commands to tell the computer what we want. If we want to go to a folder where we saved our last homework, we can type the commands to navigate to that folder and display its contents.

You can read file and folder locations by their "pathname", which is just a list of directories needed to get to your current location. Note: "folder" and "directory" mean the same thing, but most computer science texts will use the word "directory". There are two different ways of expressing pathnames that you will come across; one is a constant pathname, which describes how to access a location from anywhere. A second one is a relative path name, which tells you how to get to a location from your current location. If you are already in a folder. You may see these pathnames have something like "..", which tells the computer to go back a directory from the current location, but otherwise the style of these pathnames will look similar.

You will need to learn the basic text commands to navigate your computer system using the terminal. Important commands to learn include how to change which folder you are in, how to see the contents of your current folder, and more. Here is a list of commands and their meanings:

`ls`	list directory contents. Note, this a lowercase l, not an uppercase "i". This will 'list' everything in the current directory.
`cd [location]`	change directory, taking you to the specified [location].
`mkdir [name]`	make directory, which will make a new directory inside of your current location using the provided [name].
`cp [from here] [to here]`	copy a specified file from the first specified location to the second specified location.
`mv [from here] [to here]`	move a specified file from the first specified location to the second specified location.
`zip [zip file] [listed files]`	Zip all listed files into a new zip file called [zip file].
`rm [file name]`	permanently delete the file called [file name].

Note, the square brackets are not part of the command but only to illustrate where you should insert some other text.

Finally, you will need to know how to compile code from your terminal. Let us say that we have some code in a file called "helloWorld.cpp". In order to compile this program, we would have to navigate to the directory containing this file. Once there, you will enter the command:

```
g++ -std=c++17 helloWorld.cpp
```

`g++` is the compiler program.
`-std=c++17` is the version of C++ we want to use.
`helloWorld.cpp` is the file we wish to compile.
This command creates a file named a.out which is the compiled version of the code in helloWorld.cpp, which can be executed. You can run this compiled code using the following command:

```
./a.out
```

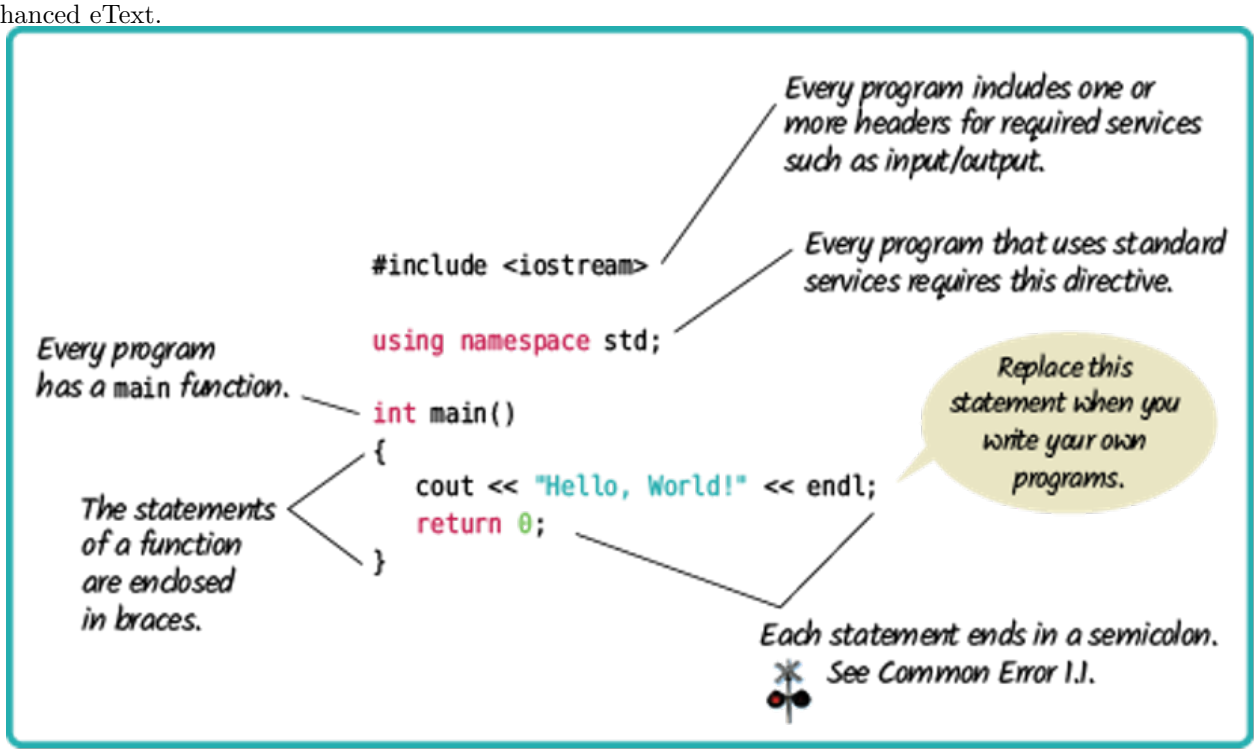
You can add the `-o` flag to your compiling command to give the output file a name. Additionally, in the near future you may also find it helpful to begin compiling with flags that tell you more information about possible errors in your code. These flags include:
`-Wall`
Wall is short for "Warn All", which will turn on most of the warnings in C++. This will help identify various possible ways your code might go wrong, including array bounds errors and other helpful messages.
`-Werror`
Werror will treat all warnings as errors. This will prevent you from skipping past the possible sources of error in your code, and you will need to make sure all warnings are resolved prior to compiling.
`-Wpedantic`
This flag enables warnings that alert you about language constructs that are not ISO C or ISO C++ standard compliant. This is particularly helpful to identify constructs that may not be uniform in other compilers, which could cause problems with your code on other machines. This will help prevent instances where your code works on your personal computer, but does not work on CodeRunner or on the grader's computer. All together, your command line prompt will look something like this:
`g++ -Wall -Werror -Wpedantic -o myName.out -std=c++17 myCodeFile.cpp`
Pro Tips:

- Tab Complete: if you're typing something in the command line that' s very long, but unique, you can hit tab when you're partially through and it will try to fill in the rest (kind of like autocomplete). If it doesn't, and you press tab twice, it tells you everything it has as options.
- Command history browsing: if you have typed a command and want to repeat it, just press the up arrow. It will bring up your last executed command. Pressing up again will go to the one before. Pressing down will go forward in time through the list.

1.2 Anatomy of a C++ Program

You will need a couple of components for the computer to be able to understand how to translate code that you can read into something it can actually do.

Here is a snapshot of a basic "Hello World" program from the course textbook, Brief C++: Late Objects, Enhanced eText.



2 Recitation

2.1 Software Setup

You will need to install and become familiar with the software we will be using this semester. Follow the appropriate installation guides in Appendix A for your particular computer (Windows or Mac) to install VS Code.

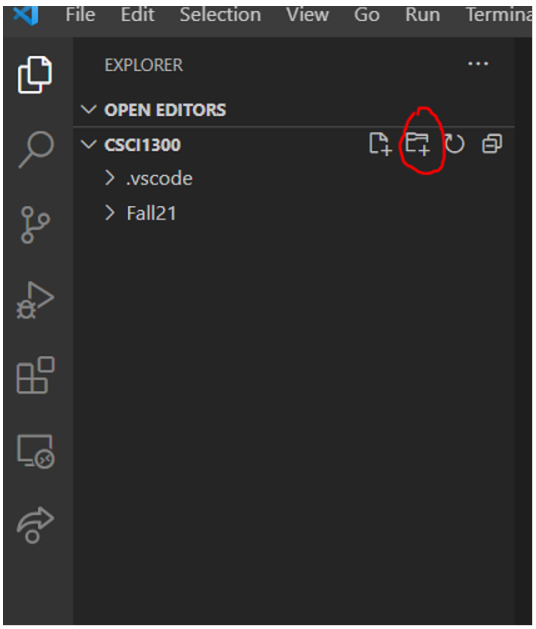
Once you have installed your software you will need to open VS Code and open the terminal. You will need VS Code and a working terminal. Take a screenshot and submit it on Canvas for this week's Recitation assignment.

3 Homework

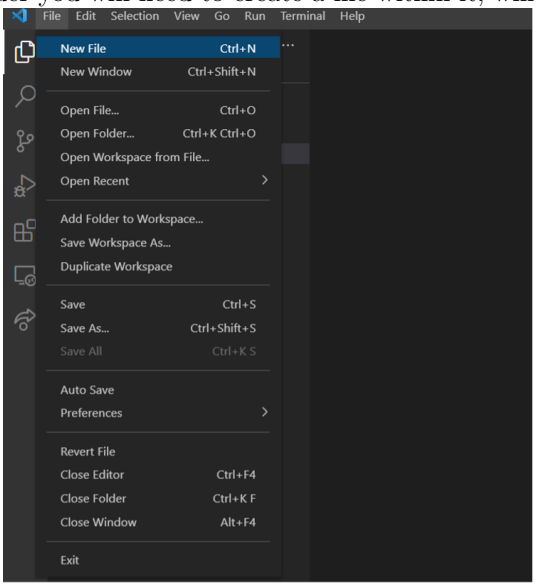
3.1 Hello World

The "Hello, World!" program is one of the simplest programs in a programming language, and it is often used to illustrate the basic syntax of a programming language. We will need to first create a folder to store our program file in and then create the file to write the program.

First, open VS Code. You can make a new folder by clicking this button:



Name the folder something intuitive, like “Week_1”, “Recitation_1”, or similar. Once you have your folder you will need to create a file within it, which you can do with the toolbar across the top:



You should name your file `helloWorld.cpp` – this file name structure is important. Files of uncompiled C++ code should end with `.cpp`, and the usage of a capital letter at the beginning of each word makes it legible even without spaces or punctuation. This style of writing names is often called “camel case”.

Type the following code into your file:

```
#include <iostream>

int main() {
    std::cout << "Hello, world!" << std::endl;
}
```

Save your file. A quick shortcut to save is Cmd + S (In Mac) or Ctrl + S (in Windows). Now you will need to compile and run your program. You can review the background information to see how to do this. Once you have completed this and successfully run your program, take a screenshot of your VS Code window including both the file and the terminal.

3.2 Hello World: Improved

Now, let us explore this program a little more. Add a statement to use the standard namespace. Insert `using namespace std;` at the beginning of the code, then we can remove the `std::` prefixes. Your new code should look like this:

```
#include <iostream>

using namespace std;

int main() {
    cout << "Hello, world!" << endl;
}
```

Run the program again using the steps above. Take an additional screenshot.

3.3 Hello CSCI 1300!

Let's modify the file from Recitation to print "Hello world! Hello CSCI 1300". The text inside of the quotation marks is printed as it is. It's case sensitive too! To see the updated output, compile and run again.

```
#include <iostream>

using namespace std;

int main() {
    cout << "Hello, world! Hello CSCI 1300" << endl;
}
```

Once again, take a screenshot. For this section you should have three screenshots for the different Hello World programs.

Create a ZIP file that contains these three screenshots, and the three .cpp files you created, and submit it on Canvas.

Week 2: Variables and Operators

Learning Goals

This week you will:

1. Be able to identify and understand variable types and operations on variables
2. Implement variables and operations in a C++ program to solve a computational problem
3. Write arithmetic expressions and assignment statements in C++
4. Appreciate the importance of comments and good code layout
5. Create programs that read and process input, and display the results

1 Background

1.1 Variables and Operators

This week we learned about variables and basic arithmetic operators in computer science.

Variables in computer science are similar to variables you have seen in other math and science courses. A **variable** is a value that can change, depending on conditions or on information passed to the program.

A computer stores variables differently depending on the type of information they are meant to contain, so we must declare the type of variable. This table will tell you the names of different types of variables in C++:

Type of variable	Declaration in C++	Size	Description
Integer	int	2 or 4 bytes	a number that has no decimal value
Float	float	4 bytes	a number that has short decimal values
Double	double	8 bytes	a number that can have longer decimals
Character	char	1 byte	stores a single letter/character using ASCII values
Boolean	bool	1 byte	stores either True or False

In order to create and use a variable, you must declare it using the appropriate declaration from the above table and then give it a name. You may also provide it an initial value, but this is not required to create a variable. Some examples include:

Example 1.1.1. Here is an example of declaring variables:

```
int myNumber = 0;
double myDecimal = 3.1415;
char myEmptyChar;
```

After declaring variables, you can access them without repeating their data type. You can also perform basic operations, including the ones listed in this table:

Operator	Description	Example
+	Adds together two values	x + y
+=	Adds to itself and stores the new value	x += y
-	Subtracts one value from another	x - y
*	Multiplies two values	x * y
/	Divides one value by another	x / y
%	Modulus; this returns the division remainder	x % y
++	Increment; this increases the value of a variable by 1	++x
--	Decrement; this decreases the value of a variable by 1	--x

Example 1.1.2. Here is an example of declaring variables and using operators:

```
int myNumber = 4;
int mySecondNumber = 5;
int mySum = myNumber+mySecondNumber;
int myThirdNumber = 6;
mySum = mySum + myThirdNumber;
int myFourthNumber = 7;
mySum += myFourthNumber;
```

By the end of this code, the variable `mySum` would contain 4+5+6+7, or 22.

1.2 Characters

Characters are a variable type that is made of a single byte, which means they can encode for a fairly limited number of characters. We can identify which characters we can store using an ASCII table shown below.

Characters can be referred to by placing the chosen character in single quotation marks, like this:

```
char myChar = 'a';
```

Characters can also can be accessed using numerical values. Since every sequence of binary numbers that encodes a character could also instead be interpreted as an integer, we can use those numbers to identify which character we are talking about. The integer values that correspond with each letter are shown in the ASCII table.

There are a few traits that you may find helpful as you move forward: firstly, observe that all lower case letters are listed in consecutive alphabetical order, and secondly that all upper case letters are similarly listed in consecutive alphabetical order. This means that the space between a lowercase ‘a’ and a capital ‘A’ is the same as the space between a lowercase ‘b’ and a capital ‘B’ and so on. This is useful for converting characters from lowercase to uppercase.

Decimal	Hex	Char	Decimal	Hex	Char	Decimal	Hex	Char	Decimal	Hex	Char
0	0	[NULL]	32	20	[SPACE]	64	40	@	96	60	`
1	1	[START OF HEADING]	33	21	!	65	41	A	97	61	a
2	2	[START OF TEXT]	34	22	"	66	42	B	98	62	b
3	3	[END OF TEXT]	35	23	#	67	43	C	99	63	c
4	4	[END OF TRANSMISSION]	36	24	$	68	44	D	100	64	d
5	5	[ENQUIRY]	37	25	%	69	45	E	101	65	e
6	6	[ACKNOWLEDGE]	38	26	&	70	46	F	102	66	f
7	7	[BELL]	39	27	'	71	47	G	103	67	g
8	8	[BACKSPACE]	40	28	(	72	48	H	104	68	h
9	9	[HORIZONTAL TAB]	41	29	)	73	49	I	105	69	i
10	A	[LINE FEED]	42	2A	*	74	4A	J	106	6A	j
11	B	[VERTICAL TAB]	43	2B	+	75	4B	K	107	6B	k
12	C	[FORM FEED]	44	2C	,	76	4C	L	108	6C	l
13	D	[CARRIAGE RETURN]	45	2D	-	77	4D	M	109	6D	m
14	E	[SHIFT OUT]	46	2E	.	78	4E	N	110	6E	n
15	F	[SHIFT IN]	47	2F	/	79	4F	O	111	6F	o
16	10	[DATA LINK ESCAPE]	48	30	0	80	50	P	112	70	p
17	11	[DEVICE CONTROL 1]	49	31	1	81	51	Q	113	71	q
18	12	[DEVICE CONTROL 2]	50	32	2	82	52	R	114	72	r
19	13	[DEVICE CONTROL 3]	51	33	3	83	53	S	115	73	s
20	14	[DEVICE CONTROL 4]	52	34	4	84	54	T	116	74	t
21	15	[NEGATIVE ACKNOWLEDGE]	53	35	5	85	55	U	117	75	u
22	16	[SYNCHRONOUS IDLE]	54	36	6	86	56	V	118	76	v
23	17	[ENG OF TRANS. BLOCK]	55	37	7	87	57	W	119	77	w
24	18	[CANCEL]	56	38	8	88	58	X	120	78	x
25	19	[END OF MEDIUM]	57	39	9	89	59	Y	121	79	y
26	1A	[SUBSTITUTE]	58	3A	:	90	5A	Z	122	7A	z
27	1B	[ESCAPE]	59	3B	;	91	5B	[	123	7B	{
28	1C	[FILE SEPARATOR]	60	3C	<	92	5C	\	124	7C	|
29	1D	[GROUP SEPARATOR]	61	3D	=	93	5D	]	125	7D	}
30	1E	[RECORD SEPARATOR]	62	3E	>	94	5E	^	126	7E	~
31	1F	[UNIT SEPARATOR]	63	3F	?	95	5F	_	127	7F	[DEL]

1.3 Strings

In C++, a `string` is a data type that represents sequences of characters instead of a numeric value (such as **`int`** or **`float`**). A string literal can be defined using double-quotes. So `"Hello, world!"`, `"3.1415"`, and `"int i"` are all strings. We can access the character at a particular location within a string by using square brackets, which enclose an index which you can think of as the address of the character within the string. Importantly, strings in C++ are indexed starting from zero. This means that the first character in a string is located at index 0, the second character has index 1, and so on. For example:

```
string s = "Hello, world!";
cout << s[0] << endl;  //prints the character 'H' to the screen
cout << s[2] << endl;  //prints the character 'l' to the screen
cout << s[9] << endl;  //prints the character 'r' to the screen
cout << s[12] << endl; //prints the character '!' to the screen
```

Note that when a character in a string is accessed with square brackets, the character has type char. For example:

```
string str = "Example"; //this is a string
char c = str[1]; //this is a char
```

There are many useful standard functions available in C++ to manipulate strings. One of the simplest is `length()`. We can use this function to determine the number of characters in a string.

```
string s = "Hello, world!";
int s_length = s.length();
cout << s_length << endl; //This line will print 13 to the screen
```

Notice how the length function is called.
The correct way:

```
s.length()
```

Common incorrect ways:

```
length(s)
s.length
```

Another useful function available for strings is `substr()`. This function allows us to access a subset, or a small portion, of a longer string. The substring function takes two arguments:

- The starting index of the substring you would like to capture
- The length of the substring you would like to capture (optional)

Note that the second argument is optional. If you don't pass a second argument to subtring, then the function will print the entirety of the string, beginning with the character at the position specified in the first argument. Note that `substr()` always returns a variable of type `string`, regardless of the length of the substring.

For example, consider the code below:

Example 1.3.1. Substring example code:

```
string str = "Coding is fun!";
cout << str.substr(0, 6) << endl;
cout << str.substr(6) << endl;
cout << str.substr(1, 1) << endl; //prints a string of length one
```

This will ouput the following:

```
Coding
 is fun!
o
```

Note: The second line of output begins with a space.

Both `length()` and `substr()` are special kinds of functions associated with objects, usually called methods, which we will discuss later in the course.

1.4 Terminal Input and Output

It is important to be able to take input from the user or to print output to the terminal as your program runs. You can do this using `cin` and `cout` for input and output, respectively.

The Hello World program you have seen provides an example of using `cout` to display output.

Example 1.4.1. Hello World for C++:

```
#include <iostream>

using namespace std;

int main() {
    cout << "Hello, world!" << endl;

    return 0;
}
```

You can similarly ask the user for a particular value and store that value in a variable using `cin`:

Example 1.4.2. An example of collecting user input to sum two integer variables and then printing their sum

```
#include <iostream>

using namespace std;

int main() {
    int firstNum;
    int secondNum;
    cout << "Please provide one number:" << endl;
    cin >> firstNum;
    cout << "Please provide a second number:" << endl;
    cin >> secondNum;
    cout << "The sum of these two numbers is " << firstNum + secondNum << endl;

    return 0;
}
```

1.5 Pseudocode and Algorithms

Pseudocode is used to develop algorithms. An algorithm is a procedure for solving a problem.

An algorithm describes actions to be executed and the order in which those actions are to be executed. In other words, an algorithm is merely the sequence of steps taken to solve a problem; like a recipe. An algorithm is not computer code. Algorithms are just the instructions which provide a clear path for you to write the computer code.

Example 1.5.1. An algorithm for adding two numbers together:
Step 1: Start
Step 2: Declare variables num1, num2, and sum.
Step 3: Read values num1 and num2.
Step 4: Add num1 and num2 and assign the result to sum.
Step 5: Display sum
Step 6: Stop

The main difference between an algorithm and pseudocode is that an algorithm is a step by step procedure to solve a given problem while pseudocode is a method of writing an algorithm. In other words, an algorithm is how to solve a problem while pseudocode is how to implement that solution.

Algorithm	Pseudocode
An unambiguous specification of how to solve a problem.	An informal high-level description of the operating principle of a computer program or other algorithm.
Helps to simplify and understand the problem.	A method of developing an algorithm.

Pseudocode is informal language that helps programmers develop algorithms (or recipes). Although there are no hard and fast rules for pseudocode, there are some suggestions to help make pseudocode more understandable and easy to read.

For instance, consider indenting all statements showing a “dependency” , like statements that use: While, do, for, if.

Several keywords are often used to indicate common input, output, and processing operations.

Input:	READ, OBTAIN, GET
Output:	PRINT, DISPLAY, SHOW
Compute:	COMPUTE, CALCULATE, DETERMINE
Initialize:	SET, INITIALIZE
Add one:	INCREMENT, BUMP

For looping and selection, the keywords that you might consider writing include:

- Do While...
- Do Until...
- Case...
- If⋯then...
- Call ... with (parameters)
- Call
- Return ....
- Return
- When

Try to indicate the end of loops and iteration by using scope terminators. For instance use if⋯(statements) ... endif.

Other words you may find useful while writing pseudocode include: Generate, Compute, Process, set, reset, increment, compute, calculate, add, sum, multiply, subtract, divide, print, display, input, output, edit, test, etc.

Be sure to indent if the indentation fosters understanding. Being clear is the purpose of pseudocode, and a very desirable goal to strive for.

Here are some examples of pseudocode:

Example 1.5.2. Will a grade pass?

```
If students grade is higher than or equal to 60
^^IThen Print, "Passed"
else
^^IPrint, "Failed"
```

Example 1.5.3. How do you find the area of a rectangle?

```
Read the length of the rectangle
Read the width of the rectangle
Compute the area of the rectangle as length times width.
```

You should write pseudocode whenever you are addressing problems in computer science. This allows you space to determine how to solve a problem without worrying about syntax and formatting, instead of having to figure out everything all at once.

2 PreQuiz

Problem 2.1. What is a variable?

Problem 2.2. Consider a variable tracking changing cost of an item on Ebay. What would be a good name for this variable?

Problem 2.3. Consider a variable that represents the roll of a 6-sided die. What possible values could this variable hold?

Problem 2.4. Consider a variable that tracks the temperature outside. What data type should you use? Are there multiple types that would work?

3 Recitation

3.1 Hello, Name!

You have recently learned how to accept user input and store it in a variable. Write a program where you ask the user for their name in the terminal, and then print `Hello, [name]!` in the terminal. You may find it helpful to start with the Hello World program from recitation.

A few example runs are shown below. In these examples, red text represents user-provided text (or text you would have to type in the terminal while running the program).

Sample Run 3.1.1

```
What is your name?
Mike
Hello, Mike!
```

Sample Run 3.1.2

```
What is your name?
samantha
Hello, samantha!
```

Problem 3.1.a. What steps would your program need to follow? Write these steps below.

Problem 3.1.b. Assume you have stored the user's name in a string variable called `name`. How would you write the cout statement to print `Hello, [name]!`?

Problem 3.1.c. Write out the steps above as a complete C++ program. Test your code. Does it work as expected? Keep testing until it does.

3.2 Converting Seconds to Days:Hours:Minutes:Seconds

A day has 86,400 seconds ($24 \times 60 \times 60$). Given a number of seconds in the range of 0 to 1,000,000 seconds, your program should print the time as days, hours, minutes, and seconds for a 24 hour clock. For example, 70,000 seconds is 0 days, 19 hours, 26 minutes, and 40 seconds. Your program should accept user input for the number of seconds to convert and then use that number in your calculations. Format your output as follows:

```
The time is W days, X hours, Y minutes, and Z seconds.
```

Problem 3.2.a. Explicitly list the variables you will need and their data types.

Problem 3.2.b. What arithmetic operators will be most useful for this problem?

Problem 3.2.c. Write out the steps you would use to solve this problem by hand as pseudocode.

Problem 3.2.d. Pick a random number between 0 and 1,000,000 for a sample run. Follow the steps you wrote from part c for this number to find your end result, and verify it. You can use your calculator, but make sure to write out each step.

Problem 3.2.e. Translate your pseudocode into a c++ program to solve the above code.

4 Homework

4.1 Identify and Correct the Errors

There are several snippets of code below where there is one error. You will need to identify and correct the error so that the code compiles. This code is also available directly in CodeRunner.

Problem 4.1.a. Spot the error:

```
#include <iostream>
using namespace std;

int Main()
{
    cout << "Hello, World!" << endl;
    return 0;
}
```

Problem 4.1.b. Spot the error:

```
#include <iostream>
using namespace std;

int main
{
    cout << "Hello, World!" << endl;
    return 0;
}
```

Problem 4.1.c. Spot the error:

```
#include <iostream>
using namespace std;

int main()
{
    cout << "Hello, World! << endl;
    return 0;
}
```

Problem 4.1.d. Spot the error:

```
#include <IOstream>
using namespace std;

int main()
{
    cout << "Hello, World!" << endl
    return 0;
}
```

Problem 4.1.e. Spot the error:

```cpp
#include <iostream>
using namespace;
int main()
{
    cout << "Hello, World!" < endl;
}
```

4.2 Fahrenheit to Celsius Converter

Create a program that convert temperatures from Fahrenheit to Celsius.

$$Celsius = (Fahrenheit - 32) * (5.0/9.0)$$

The answer-box in coderunner is pre-loaded with following solution template for this question.

```cpp
#include <iostream>

using namespace std;

int main(){
    // declare all the variable
    double fahrenheit, celsius;

    // prompt the user & get their input
    cout << "<add question>" << endl; // EDIT THIS LINE TO PROMPT USER
    cin >> fahrenheit;

    // temperature calculation
    celsius = <add equation>; // EDIT THIS LINE TO CALCULATE TEMPERATURE
    // hint: use (5.0/9.0) instead of (5/9)

    cout << "The temperature is " << celsius << " degree Celsius." << endl;
    return 0;
}
```

Here are a few sample runs of the program, with user input shown in red:

Sample Run 4.2.1

```
What is the temperature in Fahrenheit?
32
The temperature is 0 degree Celsius.
```

4.3 Calculate the Falling Time of an Object

Create a program that calculates the time for a falling object to hit the ground based on the height from which the object fell. The equation for this is:

$$time = \sqrt{\frac{(2 * height)}{9.8}}$$

- **time** is the amount of time the object fell in seconds.
- **height** is the height the object was dropped from in meters.

Hint: cmath library contains a square root function that you can utilize.

You should print the time to two decimal places. You can accomplish this by using the iomanip library and the setprecision function. Here is a sample run of the program:

Sample Run 4.3.1

```
How far did the object fall in meters?
4
The object fell for 0.90 seconds.
```

4.4 Volume Of A Fish Tank

Write a program that calculates the volume of a fish tank. Ask the user for three values: the length, the width, and the height of the tank in inches. Use these values to calculate the total volume of the tank in cubic inches. Then, translate cubic inches into gallons and tell the user how many gallons can fit in the fish tank. Print this capacity to one decimal place.

Hint: 1 cubic inch is 0.004329 gallons.

Sample Run 4.4.1

```
What is the length of the fish tank in inches?
20
What is the width of the fish tank in inches?
20
What is the height of the fish tank in inches?
20
This fish tank has a 34.6 gallon capacity.
```

4.5 Pool Water Management

You're in charge of maintaining a swimming pool with a minor leak. The pool has some water in it already and needs to be filled up. However, due to the leak, the water level decreases slightly every hour at a constant rate. Write a program takes the hour as an input (as an integer) and predicts the pool's water level over time.

Pool	**Initial water level (inches)**	**Fill rate (inches/hour)**	**Leakage rate (inches/hour)**
Indoor Pool	19	0.6	0.4
Outdoor Pool	22	0.3	0.1

Sample Run 4.5.1

```
How many hours have passed?
22
The indoor pool has 23.4 inches of water, and the outdoor pool has 26.4 inches of water.
```

4.6 Barter System

Bartering is the exchange of goods and services between two or more parties without the use of money. Below is the table of conversion values:

Items	**Values**
1 chicken	6 avocados
1 avocado	2 watermelon
1 watermelon	4 potatoes

1 chicken is equal to 6 avocados, 1 avocado is equal to 2 watermelon, and 1 watermelon is equal to 4 potatoes.

Your program should take the number of potatoes as an input(integer) and converts its value to the maximum number of chickens, avocados, watermelons, and potatoes that can be bought.

Sample Run 4.6.1

```
Enter the number of potatoes:
200
Maximum number of chicken(s) 4, avocado(s) 1, watermelon(s) 0, potato(es) 0
```

Sample Run 4.6.2

```
Enter the number of potatoes:
100
Maximum number of chicken(s) 2, avocado(s) 0, watermelon(s) 1, potato(es) 0
```

Week 3: Booleans and Conditions

Learning Goals

This week you will:

1. Understand the Boolean data type and boolean operators
2. Understand relational operators
3. Be able to implement decisions using if statements

1 Background

1.1 Booleans

Booleans are a special data type that stores only "true" or "false". This true or false value can be stored in a boolean variable, or it can be the result of evaluating different expressions.

Relational Operators

A relational operator is a feature of a programming language that tests or defines some kind of relation between two entities. These include numerical equality (e.g., `5 == 5`) and inequalities (e.g., $4 \geq 3$). Relational operators will evaluate to either True or False based on whether the relation between the two operands holds or not. When two variables or values are compared using a relational operator, the resulting expression is an example of a boolean condition that can be used to create branches in the execution of the program. Below is a table with each relational operator' s C++ symbol, definition, and an example of its execution.

Operator	Meaning	Example
>	greater than	5 > 4 is TRUE
<	less than	4 < 5 is TRUE
>=	greater than or equal	4 >= 4 is TRUE
<=	less than or equal	3 <= 4 is TRUE
==	equal to	5 == 5 is TRUE
!=	not equal to	5! = 6 is TRUE

Logical Operators

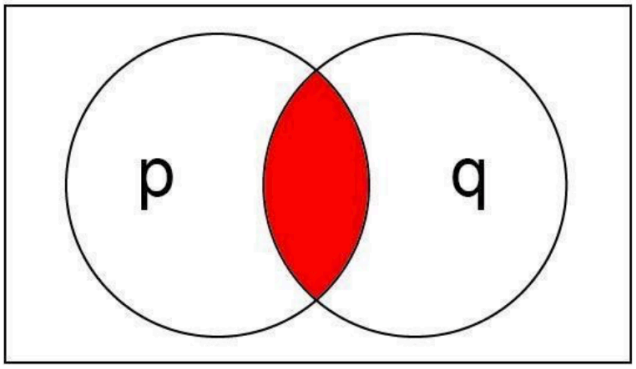
Logical operators are used to compare the results of two or more conditional statements, allowing you to combine relational operators to create more complex comparisons. Similar to relational operators, logical operators will evaluate to True or False based on whether the given rule holds for the operands. Below are some examples of logical operators and their definitions.

&&	AND	returns true if and only if both operands are true
||	OR	returns true if one or both operands are true
!	NOT	returns true if the operand is false and false if the operand is true

Every logical operator will have a corresponding truth table, which specifies the output that will be produced by that operator on any given set of valid inputs. Below are truth tables for each of the logical operators specified above.

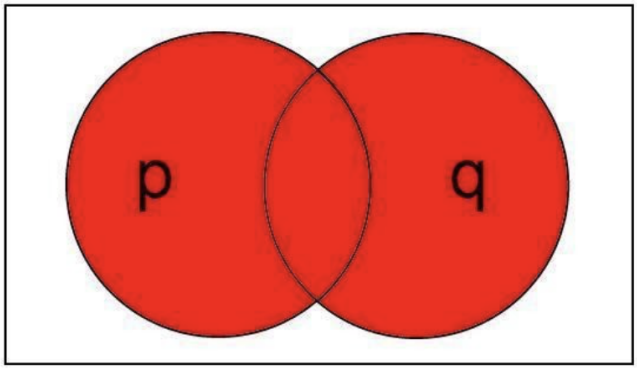
AND (`&&`): These operators return true if and only if both operands are True. This can be visualized as a venn diagram where the circles are overlapping.

p	q	p && q
True	True	True
True	False	False
False	True	False
False	False	False



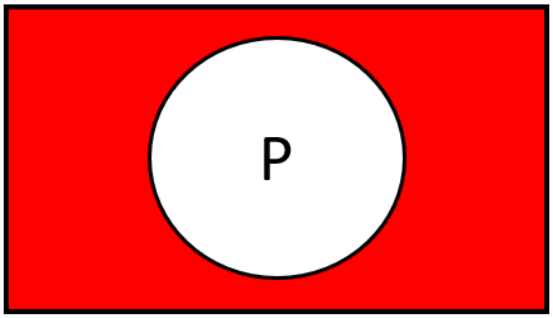
OR (`||`): These operators return True if one or both of the operands are True. This can be visualized as the region of a venn diagram encapsulated by both circles.

p	q	p || q
True	True	True
True	False	True
False	True	True
False	False	False



NOT (!): This operator returns the opposite of the operand. This can be visualized as the region of a venn diagram outside the circle. Unlike AND and OR, the NOT operator has only one operand.

p	!p
True	False
False	True



You can create truth tables for more complicated expressions by combining elements of these tables. You should begin with columns of the basic variables representing each possible combination of those variables, and then add columns to represent their modified values. For example, if you wanted to create a truth table for `!p && q` you could make a column for `p` and a column for `q` representing all possible combinations of true/false between the two variables. You can then create a third column for `!p`, and then perform the `&&` operation between the `!p` and `q` columns instead of the `p` and `q` columns, like this below:

p	q	!p	!p && q
True	True	False	False
True	False	False	False
False	True	True	True
False	False	True	False

For simple expressions, you can often work through the truth table in your head. However, knowing how to make truth tables will be helpful when you need more complicated expressions.

Using Booleans

There are two main ways you can use booleans: you can either assign them to a boolean variable, or you can use them directly as a condition (such as in an if statement). If you would like to evaluate a boolean expression and store it in a variable, you can do it like this:

```
bool myNewBoolean = (4 < 5); // this will evaluate to true
bool mySecondBoolean = (5 == 6); //this will evaluate to false
bool myFinalBoolean = (myNewBoolean && mySecondBoolean); //this will evaluate to false
```

You can string together increasingly complicated boolean equations either as a combination of boolean variables or as a combination of relational/logical expressions.

Booleans can also be represented using integers, and will print that way by default in C++. As an integer representation, 0 is false and 1 is true.

You can build if statements using boolean variables or boolean expressions.

1.2 Conditionals

Conditional statements, also known as decision statements or branching statements, are used to make a decision based on condition. A condition is an expression that evaluates to a boolean value, either true or false. Conditional Execution in C++ is a good online resource for learning about conditionals in C++.

You have seen one type of conditional expression already: switch statements. If statements, If/Else statements, and If/Else If/Else statements are a more complicated but also more dynamic way to make decisions in your code.

IF Statements

An if statement in C++ is composed of a condition and a body. The body is executed only if the condition is true. The condition appears inside a set of parentheses following the keyword "if" and the body appears within a set of curly brackets after the condition:

The general format for if statements is:

```
if ( <CONDITION> ){
    <BODY>
}
```

It is good practice to vertically align the closing curly bracket with the start of the if statement, and to indent the body.

The condition is interpreted as a boolean value, either true or false. Be careful, most expressions in C++ have a boolean interpretation. For instance, non-zero numeric values are true. Assignment operations (single equal sign) are interpreted as true as well. A common mistake is to use a single equals sign inside a condition when a double equals sign is intended.

Example 1.2.1. Here is an if statement that will check if a number is negative and change it to positive (i.e., find the absolute value):

```
if (num < 0){
    cout << "Changing sign" << endl;
    num = -1 * num;
}
```

IF-ELSE Statements

If statements may be paired with else statements in C++. If the condition associated with the if statement is false, the body associated with the else statement is executed. The else statement body is enclosed in a set of curly brackets:

```
 if ( <CONDITION> ){
    <BODY>
    // executed when CONDITION is true
}
else{
    <BODY>
    // executed when CONDITION is false
}
```

An if statement does not need an else statement, but there must be an if statement before every else statement.

Example 1.2.2. Here is an if/else statement that will check if a number can be a divisor before performing division:

```
if (num == 0) //notice the double equals!{
    cout << "Can't divide by 0!" << endl;
}
else{
    num = 1000 / num; //integer arithmetic
}
```

1.3 ELSE-IF Statements

Finally, an if statement may also be associated with any number of else-if statements. These statements each have an associated condition and an associated body. The body is executed if the condition is true and the conditions for all preceding if- and else-if statements in the same group are false. An else statement may be included at the end of the group but is not required. The else statement will be executed if all the previous conditions are false.

```
if ( <CONDITION> ){
    <BODY>
}
else if ( <CONDITION> ){
    <BODY>
}
else if ( <CONDITION> ){
    <BODY>
}
else{
    <BODY>
}
```

This is **not** logically the same as having multiple sequential if statements.

Example 1.3.1. These two if statements:

```
if ( <CONDITION A>){
   //do X
}
if ( <CONDITION B>){
   //do Y
}
```

are NOT logically the same as this if/else-if statement:

```
if( <CONDITION A>){
   //do X
}
else if ( <CONDITION B>){
   //do Y
}
```

In the first code section, both if statements are evaluated. If both CONDITION A and CONDITION B are true, we will do **both** X and Y. Meanwhile, in the second code block, if CONDITION A is true we will never evaluate CONDITION B, and therefore never do execute that code; here, we will **only** do X. Therefore, we need to use "else if" only when we want the two conditions to be mutually exclusive.

Example 1.3.2. Here is an if/else if/else statement to tell you if a number is positive, 0, or negative:

```
if ( num > 0 ){
    cout << "Positive" << endl;
}
else if ( num == 0 ){
    cout << "Zero" << endl;
}
else{
    cout << "Negative" << endl;
}
```

Nested If Statements

You can put if statements inside of other if statements (or if/else, or if/else if/else). The meaning of logical expressions can change when you are nesting if statements, so you should think through the truth tables for your if/else statements carefully.

```
if (booleanExpression1){
   //anything here will evaluate if booleanExpression1 is true
   if (booleanExpression2){
       //we will only evaluate this if statement if booleanExpression1 is true,
       //and then will only execute this statement if booleanExpression2 is ALSO true
   }
}
```

Nested if statements are essentially performing a logical "AND" operation on the two boolean expressions for the innermost if statement, but if only the first if statement is true you can still do other things.

1.4 Common Errors

Unintended behavior when accidentally using assignment operation (= instead of ==) in conditional statements:

Example 1.4.1. Here is some (incorrect) code:

```
int x = 5;
if (x = 1){ // one equal sign: changes value of x, will always evaluate to true

    cout << "The condition is true." << endl;
}
cout << "x is equal to " << x << endl;
```

The output of this would look like this:

Sample Run 1.4.1

```
The condition is true.
x is equal to 1
```

What you would ACTUALLY want is:

```
// CORRECT CODE
int x = 5;
if (x == 1) // two equal signs, performs comparison
{
    cout << "The condition is true." << endl;
}
cout << "x is equal to " << x << endl;
```

Which would output:

Sample Run 1.4.2

```
x is equal to 5
```

Remember, "=" is for assignment and "==" is for checking equality.

2 PreQuiz

Problem 2.1. What is the difference between arithmetic and relational operators?

Problem 2.2. What is the correct way to write a condition to determine if a variable `num` stores a number between 0 and 4, inclusive?

Problem 2.3. Fill in the blank in the code below with a condition that accounts for temperature in range of 25 (exclusive) to 50 (inclusive) degrees:

```
#include <iostream>
using namespace std;

int main()
{
    int temperature = 50;

    if (temperature > 85) {
        cout << "It's a hot day!";
    }
    if (temperature <= 85 && temperature > 50){
        cout << "It's a pleasant day.";
    }
    else if (_____________________________________________________) { //FILL IN THIS LINE
        cout << "It's a cool day.";
    }
    else {
        cout << "It's a cold day.";
    }

    return 0;
}
```

Problem 2.4. Complete the following truth table to evaluate the expressions for (a && !b).

a	b	!b	a && !b
T	T		
T	F		
F	T		
F	F		

Problem 2.5. Complete the following truth table to evaluate the expressions for (a || (!a && b)).

a	b	!a	!a && b	a || (!a && b)
T	T			
T	F			
F	T			
F	F			

3 Recitation

3.1 Spot The Error

Problem 3.1.a. The code snippet below is supposed to determine if a variable stores a value that is greater than, less than, or equal to 8. Identify the error(s) in the code below, and write the correct line(s).

```
#include <iostream>
using namespace std;

int main()
```

```
{
    int num = 6;

    if (num > 8) {
        cout << "The number is greater than 8." ;
    }
    else if (num = 8) {
        cout << "The number is equal to 8.";
    }
    else {
        cout << "The number is less than 8.";
    }

    return 0;
}
```

Problem 3.1.b. The code snippet below is supposed to determine if a variable stores a value for an angle that is obtuse, right, or acute. Identify the error(s) in the code below, and write the correct line(s).

```
#include <iostream>
using namespace std;

int main()
{
    int angle = 120;
    if (x>90) {
        cout<<"It is an obtuse angle." ;
    }
    elif(x=90) {
        cout<<"It is a right angle.";
    }
    else{
        cout<<"It is an acute angle.";
    }
}
```

Problem 3.1.c. The code snippet below is supposed to determine if a variable stores a value that is equal to zero or not. Identify the error(s) in the code below, and write the correct line(s).

```
#include <iostream>
using namespace std;

int main()
{
    int num = 7;

    if (num) {
        cout << "The number is zero.";
    }
    else {
        cout << "The number is not zero.";
    }

    return 0;
}
```

Problem 3.1.d. The code snippet below is supposed to determine if a variable stores a value that is equal to zero or not. Identify the error(s) in the code below, and write the correct line(s).

```
#include <iostream>
using namespace std;

int main()
{
    int num = 0;

    else {
        cout << "This is the 'else' block.";
    }
    if (num == 0) {
        cout << "The number is zero.";
    }
    else {
        cout << "The number is not zero.";
    }

    return 0;
}
```

Problem 3.1.e. The following code snippet is expected to accept a user provided integer and then state whether that number is even or odd. Identify the error(s) in the code below, and write the correct line(s).

```
#include <iostream>
using namespace std;

int main()
{
    int num;
    cout << "Provide an integer:" << endl;
    cin >> num;

    if (num/2){
        cout << "The number is even." << endl;
    }
    else {
        cout << "The number is odd." << endl;
    }

    return 0;
}
```

Problem 3.1.f. The following code snippet is expected to accept a user provided character and then state whether the corresponding grade passes or not. Identify the error(s) in the code below, and write the correct line(s).

```
#include <iostream>
using namespace std;

int main()
{
    char grade;
```

```
    cout << "Provide a grade (A, B, C, D, or F):" << endl;
    cin >> grade;

    if (grade == 'A' || 'B' || 'C'){
        cout << "This is a passing grade." << endl;
    }
    else if (grade == 'D'){
        cout << "This grade passes with conditions." << endl;
    }
    else {
        cout << "This is a failing grade." << endl;
    }

    return 0;
}
```

3.2 Step Tracking App

Your goal is to walk 10,000 steps every day but you aren't great at remembering to do it! So you decide to create a step tracking app that tracks your steps every day and will alert you based on how much you walked for the day. The program first asks how many steps you walked that day and then displays a message based on whether you have hit your goal for the day. Next, it will also tell you how many steps you have left to walk.

The following are the possible messages you will get based on your intake:

- If you've walked 5,000 steps or less, then you get:

  `You have not walked much today! Get those steps in! You have X steps left to walk.`

- If you' ve walked more than 5,000 steps but less than 10,000 steps, you get:

  `You're doing great, over half way there! You still have X steps left to walk.`

- If you've walked 10,000 steps or more, you get:

  `You've hit your goal for the day! Great job getting exercise!`

Note that **X** is the amount of steps left after subtracting how far you have walked.
Here is a sample run:

Sample Run 3.2.1

```
How many steps have you taken today?
3000
You have not walked much today! Get those steps in! You have 7000 steps left to walk.
```

Problem 3.2.a. Write an algorithm in pseudocode for the program above.

Problem 3.2.b. Imagine how a sample run of your program would look like. Write at least two examples.

Problem 3.2.c. Identify the values that you must test for. We call these values "boundary conditions".

Problem 3.2.d. Implement your solution in C++ using VS Code. Revise your solution, save, compile and run it again. Are you getting the expected result and output? Keep revising until you do. Make you sure you test for the values used in your sample runs, and for the boundary conditions.

4 Homework

4.1 Preparing for the Heat

You are planning to go for a run outside, but it's a hot day. Write a C++ program to determine whether you need to carry extra water based on the temperature.

The program should prompt the user to enter the temperature in degrees Fahrenheit. If the given temperature is above 85°F, display "You need to carry extra water." as it is considered hot weather for running, and then terminate. If the temperature is 85°F or below, the program will display "You don't need extra water." as carrying extra water may not be necessary, and then terminate.

Additionally, the program should perform input validation. If the user inputs a non-positive value for temperature (i.e., 0 or a negative number), the program should display "Invalid temperature." and terminate.

Sample Run 4.1.1

```
What is the temperature?
75
You don't need extra water.
```

Sample Run 4.1.2

```
What is the temperature?
90
You need to carry extra water.
```

Sample Run 4.1.3

```
What is the temperature?
-5
Invalid temperature.
```

4.2 Ordering Pizza

You've decided to order a pizza for dinner. The restaurant you're ordering from has three sizes of pizza: S, M, and L. Each size has a different base price and price per topping. The prices are indicated in the table below. Create a program to calculate the total cost of your pizza. The program should prompt you for the size of the pizza and the number of toppings, then output the cost.

Size	Base Price	Price Per Topping
S	8.00	0.99
M	10.00	1.99
L	14.00	2.99

The input should be a character (for size) and a non-negative integer (for the number of toppings). The program should accept both lowercase and uppercase inputs for pizza sizes (e.g., 's', 'm', 'l' or 'S', 'M', 'L'), and the output should be double.

Note: You can utilize the `toupper()` or `tolower()` functions to convert the input into either upper or lower case before comparing.

Note: The total cost should be formatted with a two-digit precision. You can use the `setprecision()` function with the fixed manipulator from `<iomanip>` library to do so.

Bad formatting: 10.8
Good formatting: $10.80
You will also need to perform input validation on the size of the cake – i.e., XL is not valid.

Sample Run 4.2.1

```
What size pizza would you like to order?
S
How many toppings do you want?
3
Your total is $10.97
```

Sample Run 4.2.2

```
What size pizza would you like to order?
XL
How many toppings do you want?
2
Invalid pizza size.
```

Sample Run 4.2.3

```
What size pizza would you like to order?
M
How many toppings do you want?
-2
Invalid number of toppings.
```

Sample Run 4.2.4

```
What size pizza would you like to order?
R
How many toppings do you want?
-2
Invalid pizza size and number of toppings.
```

4.3 Travel

You want to write a program that tells you if you can afford a road trip. In order to do this, you need to know your budget, how far you are driving, and how many nights you will stay.

To calculate the gas money, you estimate the road trip will cost you 16 cents per mile. After computing the gas money, you can determine your budget for each night. If you have less than $20 a night, you cannot afford to go. If you have at least $20 a night, you can afford to go camping during this trip. If you have at least $50 a night, you can afford a cheap motel. If you have at least $100 a night, you can afford a nice hotel.

You should perform basic input validation for all inputs by checking that they are non-negative numbers. If any of the inputs are invalid, the program should display "Invalid input(s)." Otherwise, the program should proceed with the calculations based on the provided values.

Sample Run 4.3.1

```
What is your budget?
500
How many miles will you drive?
800
How many nights do you want to spend there?
3
You can afford to stay in a nice hotel on this trip.
```

Sample Run 4.3.2

```
What is your budget?
500
How many miles will you drive?
800
How many nights do you want to spend there?
6
You can afford to stay in a cheap motel on this trip.
```

Sample Run 4.3.3

```
What is your budget?
100
How many miles will you drive?
800
How many nights do you want to spend there?
3
This trip is outside your budget.
```

Sample Run 4.3.4

```
What is your budget?
300
How many miles will you drive?
1200
How many nights do you want to spend there?
4
You can afford to go camping on this trip.
```

Sample Run 4.3.5

```
What is your budget?
-300
How many miles will you drive?
1200
How many nights do you want to spend there?
4
Invalid input(s).
```

4.4 Temperature Changes

You've decided to test the temperature outside every morning. Create a program that will tell you if the temperature over the last three days has increased, decreased, or neither. If the temperatures are increasing, print out "It's getting warmer!". If the temperatures are decreasing, then print out "It's getting cooler!". If the temperatures are not in any order or if two or more temperatures are the same, then print "The temperature is changing unpredictably."

The user should input 3 non-negative numbers (double) separated by spaces.

Sample Run 4.4.1

```
Enter temperatures over the last three days:
75.9 79.3 81.2
It's getting warmer!
```

Sample Run 4.4.2

```
Enter temperatures over the last three days:
45 30 18
It's getting cooler!
```

Sample Run 4.4.3

```
Enter temperatures over the last three days:
79 75 90
The temperature is changing unpredictably.
```

Sample Run 4.4.4

```
Enter temperatures over the last three days:
-79 75 90
Invalid temperature input.
```

4.5 Car Rental

You have decided to go on a weekend road trip with your friends. So, you plan to rent a car for this trip. The car rental company offers many options and has 4 categories. Each type of car has its specific base rate and price per day to rent the vehicle. Write a C++ program that calculates the total cost based on the car type and number of days.

Car Type	Base Price	Price per day
A	$80	$15
B	$110	$25
C	$160	$35
D	$220	$45

The total bill is calculated based on this formula:

$$\text{Total} = 1.23 \times (\text{base price} + \text{no. of days} \times \text{price per day})$$

The input should be a character (for car type), a non-negative integer (for the number of days you want to rent the car), and the output should be double.

Ensure you are doing input validation. The user should input car type from one among A, B, C, or D, and the minimum number of days to rent a car is 1. If the car type or the number of days is invalid, display "Please enter valid input." and exit the program.

Note: The total cost should be formatted with a two-digit precision. You can use the `setprecision()` function with the `fixed` manipulator from the `<iomanip>` library to do so.
Bad formatting: 10.8
Good formatting: $10.80

Sample Run 4.5.1

```
Which car type (A, B, C, or D) would you like to rent?
A
How many days would you like to rent this car?
6
Your total is $209.10
```

Sample Run 4.5.2

```
Which car type (A, B, C, or D) would you like to rent?
C
How many days would you like to rent this car?
11
Your total is $670.35
```

Sample Run 4.5.3

```
Which car type (A, B, C, or D) would you like to rent?
E
How many days would you like to rent this car?
1
Please enter valid input.
```

Sample Run 4.5.4

```
Which car type (A, B, C, or D) would you like to rent?
C
How many days would you like to rent this car?
-1
Please enter valid input.
```

Week 4: Functions

Learning Goals

This week you will:

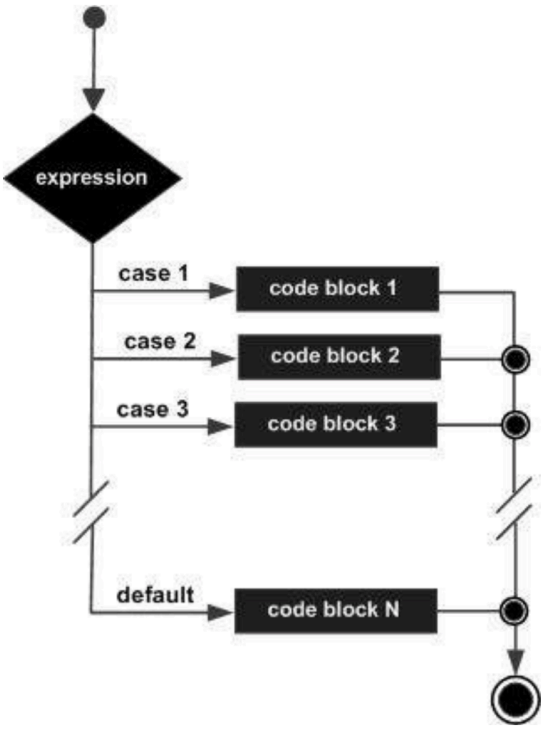
1. Understand switch statements
2. Understand basic functions:
   - Return types
   - Parameter lists
   - Function calls
3. Be able to identify the scope of a variable

1 Background

1.1 Switch Statements

Switch statements are an easy way to make decisions in a program. We can execute different sections of our code based on the value of a character or integer variable.

- If we are building a switch statement around an `int` variable, all of the cases must be defined using numbers.
- If we are building a switch statement around a `char` variable, all of the cases must be defined using characters. This means they must also use single quotes.



With the switch statement, the variable name is used once in the opening line. The **case** keyword is used to provide the possible values of the variable, which is followed by a colon and a set of statements to run if the variable is equal to a corresponding value.

An example of a simple switch statement:

Example 1.1.1. Switch statement syntax:

```
switch (n){
     case 1:
          // code to be executed if n == 1;
          break;
     case 2:
          // code to be executed if n == 2;
          break;
     default:
          // code to be executed if n doesn' t match any cases
}
```

Important notes to keep in mind when using switch statements :

- The expression provided in the switch should result in a constant value otherwise it would not be valid.
  - `switch(num)`
    - allowed (`num` is an integer variable)
  - `switch('a')`
    - allowed (takes the ASCII Value)
  - `switch(a+b)`
    - allowed, where `a` and `b` are **int** variables, which are defined earlier

- The `break` statement is used inside the switch to terminate a statement sequence. When a `break` statement is reached, the switch terminates, and the flow of control jumps to the next line following the switch statement.
- The `break` statement is optional. If omitted, execution will continue on into the next case. The flow of control will fall through to subsequent cases until a break is reached.
- The default statement is optional. Even if the switch case statement does not have a default statement, it would run without any problem.

Switch statements are a simple way to make decisions based exclusively on the equivalence of some characters or numbers. They are a simple form of **conditional statement**.

1.2 Functions

"Functions" in the context of math may be familiar, especially when formatted like this:

$$f(x) = x^2 + 5$$

Here we have labeled our function f, and it is a function of the variable x. You might have seen some where the functions are functions on multiple variables:

$$g(x, y) = x^2 + y^3$$

Functions in computer science are similar, but we can abstract them further so they apply to more than just numbers.

A function is a named block of code which is used to perform a particular task. The power of functions lies in the capability to perform that task anywhere in the program without requiring the programmer to repeat that code many times. This also allows us to group portions of our code around concepts, making programs more organized. You can think of a function as a mini-program.

There are two types of functions:

- Library functions
- User-defined functions

Library functions refer to pre-existing functions that you can use but did not write yourself. In order to use a library function, you must include the library that contains the function. For example, the C++ math library provides a `sqrt()` function to calculate the square root of a number. To use the `sqrt()` function, you must include the cmath library at the top of your program, e.g. `#include<cmath>`. Libraries other than the built-in C++ libraries can be found online.

C++ allows programmers to define their own functions. These are called user-defined functions. Every valid C++ program has at least one function, the `main()` function.

We pass values to functions via parameters. In general, the parameters should be all the information needed for the function to do its work. When that work is complete, we would like to use the result in other code. The function can return one value of the specified return type. A function may also return nothing, in which case its return type is void.

Here is the syntax for a function definition:

```
returnType functionName(parameterList)
{
    //function body
}
```

- The `returnType` is the data type that the function returns
- `functionName` is the actual name of the function

- `parameterList` refers to the type, order, and number of the parameters of a function. A parameter is like a placeholder. When a function is invoked, you pass a value to the parameter. This value is referred to as actual parameter or argument. Note: this can be a list of multiple items separated by commas.
- `//function body` contains a collection of statements that define what the function does. The statements inside the function body are executed when a function is called.

This may not immediately look like the functions you have seen in math, but they are actually quite close. Here is an example of the functions above written in C++:

Example 1.2.1. The function $f(x) = x^2 + 5$ written in C++:

```
double f(double x){
    return (x*x+5);
}
```

Example 1.2.2. The function $g(x, y) = x^2 + y^3$ written in C++:

```
double g(double x, double y){
    return (x*x+y*y*y);
}
```

I used `double` for the parameters and the return type because these functions in math would probably use decimals, but you could also use integers if you only wanted to use whole numbers. We can also use functions inside of other functions. Instead of just multiplying variables by themselves to do the exponents, we could use the `pow` function from `cmath`:

Example 1.2.3. The function $f(x) = x^2 + 5$ written using cmath:

```
#include<cmath>

double f(double x){
    return pow(x, 2)+5;
}
```

Example 1.2.4. The function $g(x, y) = x^2 + y^3$ written using cmath:

```
#include<cmath>

double g(double x, double y){
    return pow(x, 2) + pow(y, 3);
}
```

If you wanted to then use these functions to perform some calculations, you could use any value of x and y. You could use constant values, or you could use variables. Here are examples of ways you could use the function:

Example 1.2.5. Examples of calling the function $g(x, y)$:

```
#include<cmath>
```

```cpp
#include<iostream>

using namespace std;

double g(double x, double y){
    return pow(x, 2) + pow(y, 3);
}

int main(){
    double firstValue = g(4.0, 5.0);
    double secondValue = g(2, 3);
    double myX, myY;

    cout << "What value would you like for X?" << endl;
    cin >> myX;
    cout << "What value would you like for Y?" << endl;
    cin >> myY;
    cout << "The result would be " << g(myX, myY) << endl;

    return 0;
}
```

A function has its own **scope**. That means that the parameters of the function, as well as local variables declared within the function body, are not accessible outside the function. This is useful because it allows us to solve a small problem in a self-contained way. Parameter values and local variables disappear from memory when the function completes its execution. We can illustrate the order in which code executes with this example:

Example 1.2.6. An example with a simple function:

```cpp
#include <iostream>
using namespace std;

//funtion to add two numbers
int sum( int num_one, int num_two)
{
    int result = num_one + num_two;
    return result;
}

//main function
int main()
{
    //declare parameter value
    int parameter_var = 1;

    //call the function
    int sum_result = sum(parameter_var, 99);

    cout << "The sum is " << sum_result << endl;

```

```
22      return 0;
23  }
```

The code will begin executing in the main function – in this case, the first step is declaring a variable in line 15. When the program execution reaches line 18, the main function will pause, and the first line in the body of the `sum()` function will begin running. After the line `return result;` is reached, `sum()` will stop running, and the main program will resume execution where it left off. In this case, when `main()` resumes execution, the return value of `sum()` will be stored in `int sum_result`, and then the last two lines of the main function will run.

Functions can do more than just perform calculations; they can also perform operations on other variables, or they can be used to prevent yourself from copy/pasting the same code multiple times in your program. They do not even always need parameters. For example, if you wanted to write out a selection menu, you could write a function to print the menu so you do not have to type it out every time. In these cases, the parenthesis can be empty, like they are for our main functions.

As a general note: the function names f and g are not very good function names. You should generally pick better (clearer) names, and you should choose a naming style to be consistent. The naming style you choose should be different from how you name variables, so it is easier to read your code.We recommend using camel case to name your functions, like so:

Examples of function names we might use: `circleArea(), sumList(), findCapitalLetters()`

1.3 Testing Code

You must naturally test your code to make sure that it works correctly, and that it works correctly **all** the time. This may seem like a very difficult task, but there are several steps you can take to make sure you start correctly.

- Come up with a handful of test cases to use on your program.
- Consider the ways a user could use your program incorrectly. Use this to develop additional test cases.
- Consider extreme values, or "boundary conditions". Use these boundary conditions to develop yet more test cases.
- Test each part of your program independently. This is called *Unit Testing*. Do you have individual functions? Test each of them. Do you have major steps in your main function? Test each of them.

Boundary conditions are significant for many complex problems, and should test the extreme limits of what your problem may be applied to. Example: Are you supposed to examine a string? What happens if that string is empty? What happens if that string is thousands of digits long?

You should develop test cases for each boundary condition you can come up with. If there are several components to your program, you may need to identify boundary conditions for each of these components. Testing these boundary conditions of these pieces individually – whether they are functions or just blocks of code in your main function – is often easier than testing every independent combination of them.

Consider a program where you have 4 functions, and each function has 3 boundary conditions. To test each boundary condition for each of the 4 functions, you would need 12 tests. If however you only tested the program as a whole, you might need to check each combination of boundary conditions, which would end up becoming $3^4 = 81$ different tests. This is part of why **unit testing** is valuable.

As you develop extra functions, you should start by using your `main` function to test these functions. There will be 3 different types of test cases you should be expected to write depending on the return type of the function. Listed below is how we expect you to test different types of functions. The process will be different for if you are testing a `void` function, non-void functions that return an `int` or `bool`, and non-void functions that return a `double`.

Testing Void Functions

For void functions that have printed output (i.e. functions that use cout to print to the terminal), call the testing function in the main function. Your tests should include the expected output in comments. For these functions you will want to make sure that all expected outputs are successfully printed.

See the example code below:

Example 1.3.1. This is testing a function that prints whether a grade is passing or not.

```
void checkGrade(char grade){
    switch(grade){
        case 'a':
        case 'b':
        case 'c':
            cout << "You passed!" << endl;
            break;
        case 'd':
            cout << "You did not pass, but you were close." << endl;
            break;
        case 'f':
            cout << "You failed." << endl;
            break;
        default:
            cout << "That is not a valid grade." << endl;
    }
}

int main(){
    checkGrade('b'); //Should output "You passed!"
    checkGrade('d'); //Should output "You did not pass, but you were close."
    checkGrade('f'); //Should output "You failed."
    checkGrade('m'); //Should output "That is not a valid grade."
}
```

Testing Integer/Boolean Functions

For non-void functions that return a `bool` or `int`, use an `assert` statement from the `cassert` header (`#include <cassert>`) with a conditional expression.

Assert tests contain a conditional expression which will be true unless there is a bug in the program. If the conditional expression evaluates to false, then your program will terminate and show an error message.

For immediate purposes, functions that return a `bool` or `int` can be compared to a specific value using the equality operator ==.

Your test will look something like this:

`assert(<function call> == <value to compare to>);`

- `<function call>` is where you will call the function you want to test with its function parameters.
- `<value to compare to>` is the value you expect the function to return.
- == is the equality operator, and it compares the equality of both sides of itself.

See the sample code below:

Example 1.3.2. The below code shows examples of how to test integer functions with a simple addition function:

```
#include <iostream>
#include <cassert>
using namespace std;

int addInts(int num1, int num2)
{
    // add num1 and num2 before returning
    return num1 + num2;
}

int main()
{
    // test 1 for addInts
    assert(addInts(5, 6) == 11);
    // test 2 for addInts
    assert(addInts(10, 10) == 20);
}
```

Testing Double Functions

For non-void functions that return a double, use an `assert` statement from the `cassert` header (`#include <cassert>`) with a conditional expression and include the following function in your program:

Example 1.3.3. This is a required function to successfully test Double functions in C++:

```
/**
 * doublesEqual will test if two doubles are equal to each
 * other within two decimal places.
 */
bool doublesEqual(double a, double b, const double epsilon = 1e-2)
{
    double c = a - b;
    return c < epsilon && -c < epsilon;
}
```

Because the **double** type holds so much precision, it will be hard to compare the equality of a function that returns a **double** with another **double** value. To overcome this challenge, we can compare **double** values within a certain range of precision or decimal places. The function above compares the equality of two values `a` and `b` up to two decimal places. This function returns `true` if the values `a` and `b` are equal with each other up to two decimal places.

You will be expected to use this function in conjunction with assert statements to test functions that return the type double.

Your test will look something like this:

`assert(doublesEqual(<function call>, <value to compare to>));`

- `<function call>` is where you will call the function you want to test with its function parameters
- `<value to compare to>` is the double value you expect the function to return.

See the sample code below:

Example 1.3.4. This is code to test a function that finds the reciprocal of a value (i.e., divides 1 by that number).

```
#include <iostream>
#include <cassert>
using namespace std;
/**
 * doublesEqual will test if two doubles are equal to each other within
 * two decimal places.
 */
bool doublesEqual(double a, double b, const double epsilon = 1e-2)
{
    double c = a - b;
    return c < epsilon && -c < epsilon;
}
/**
 * reciprocal returns the value of 1 divided by the number
 * passed into the function.
 */
double reciprocal(int num)
{
    return 1.0 / num;
}
int main()
{
    // test 1 for reciprocal
    assert(doublesEqual(reciprocal(6), 0.16));
    // test 2 for reciprocal
    assert(doublesEqual(reciprocal(12), 0.083));
}
```

General Testing Tips

You will certainly come to a time in your coding career when your code does not work, and you just cannot figure out *why*.

In these times, there are a few possible options. First, your algorithm may be incorrect. If that is the case no amount of code testing will help you, and you will need to go back and think through how to solve the problem. If your algorithm is correct but your code is not, here are three tips:

1. If your code does not compile, start commenting out sections of your code. Keep going until it compiles, even if you have to go all the way back to an empty main function. Then, you can uncomment sections of your code until it fails to compile again. This will help you pinpoint *where* the issue is in your code, and once you know where it is you will be able to see *what* it is.
2. If your code compiles but has unexpected runtime errors, add output statements periodically throughout your code. When your program fails, you will know where your code stopped running based on which output statements failed to print.
3. If your code compiles and runs completely but the output is incorrect, go back through your code and print out the significant variables at each step in your code. You can then compare this to the test cases you worked out by hand and see where the code output differs from your algorithm.

2 PreQuiz

Problem 2.1. Select True or False:

A) **T/F:** When writing switch statements in C++ you must include a default case, otherwise your switch statement is not valid.

B) **T/F:** Switch statements can be built on integers, floats, and characters.

C) **T/F:** There is no difference between an if/else-if/else-if chain and an if/if/if chain in C++.

Problem 2.2. Given two conditions, `cond1` and `cond2`, the expression if `(cond1 && cond2)` will evaluate to true only if _____`cond1` and `cond2` are true. On the other hand, the expression if `(cond1 || cond2)` will evaluate to true if _____`cond1` or `cond2` is true.

Problem 2.3. How can you write a switch statement where multiple cases will execute the same block of code? Provide an example.

Problem 2.4. Fill in the blank(s) for the code below:

```
int choice;
cout << "Help our adventurer discover what to do next! Enter 1, 2, or 3." << endl;
______ >> choice;

switch(____________){
    case ________:
        cout << "You found a hidden treasure!" << endl;
        break;
    case 2:
        cout << "You discovered a secret passage!" << endl;
        __________
    case 3:
        cout << "You found a mystical artifact!" << endl;
        break;
    ________:
        cout << "That's not a valid case. Please try again." << endl;
}
```

3 Recitation

3.1 Spot The Error

Problem 3.1.a. Below is code that asks the user for the day of the week as a number (Monday is 1, Sunday is 7) and then prints a corresponding statement. Identify the error(s):

```
int day;
cout << "What number day of the week is it?" << endl;
cin >> day;
```

```
switch (day) {
  case '6':
    cout << "Today is Saturday";
    break;
  case 7:
    cout << "Today is Sunday";

  default:
    cout << "Looking forward to the Weekend";
}
```

Problem 3.1.b. Below is code with the same goal as the previous question, but different error(s). Identify the error(s):

```
int day = 4;
switch (day)
  case 6:
    cout << "Today is Saturday";
    break;
  case 7:
    cout << "Today is Sunday";
    break;
  default
    cout << "Looking forward to the Weekend";
```

Problem 3.1.c. The code below is meant to determine if an angle is acute, obtuse, or right. Spot the error(s):

```
#include <iostream>
using namespace std;

int main()
{
    int angle =40;
    if (x<90) {
        cout<<"It is an acute angle." ;
    }
    else if(x=90) {
        cout<<"It is a right angle.";
    }
    els{
        cout<<"It is an obtuse angle.";
    }
}
```

Problem 3.1.d. The code below implements an exclusive OR logical operation, which means that only one of the conditions may be true. Spot the error(s):

```
// This program implements XOR
#include iostream
using namespace std;

//Set the variable value to 1 when x or y is 1
int main(){
    int x = 1,y=0,value;
```

```
    if (x == 1){
        if(y==0)
        value = 1;

        else
        y == 0;

    if(x==0){
        if(y==0)
        value = 0;

        else
        value = 1;
    }

    cout < value < endl;
    return 0;
}
```

3.2 Final Velocity of a Rocket

Write a C++ program that will calculate the final velocity of a rocket after 20 seconds. The program will ask the user for the initial velocity (m/s) and the fuel type (A, B, C). The rate of acceleration will depend on the type of fuel and the initial velocity.

- If initial velocity is less than 10, then the acceleration rate for each fuel type is as follows
  - Fuel type A → 5 (m/s) per second
  - Fuel type B → 10 (m/s) per second
  - Fuel type C → 20 (m/s) per second
- If initial velocity is greater than or equal to 10 and less than or equal to 40, then the acceleration rate for each fuel type is as follows
  - Fuel type A → 6 (m/s) per second
  - Fuel type B → 12 (m/s) per second
  - Fuel type C → 24 (m/s) per second
- If initial velocity is greater than 40, then the acceleration rate for each fuel type is as follows
  - Fuel type A → 3 (m/s) per second
  - Fuel type B → 6 (m/s) per second
  - Fuel type C → 9 (m/s) per second

Below are some sample runs. User input is shown in bold.

Sample Run 3.2.1

```
Enter the initial velocity:
70
Enter the fuel type:
C
The final speed is 250 m/s.
```

Sample Run 3.2.2

```
Enter the initial velocity:
5
Enter the fuel type:
A
The final velocity is 105 m/s.
```

Problem 3.2.a. Write out the steps you would use to solve this problem by hand as pseudocode.

Problem 3.2.b. Pick possible inputs for your program. Follow the steps you wrote for these values to find your result, and verify it.

Problem 3.2.c. Identify two possible values that are "boundaries" in this problem that you will have to test. What should happen for these values?

Problem 3.2.d. Translate your pseudocode into a c++ program to solve the above code.

4 Homework

4.1 Car switch

Write a program that accepts a single character representing an automobile manufacturing company as input from the user. Then, the program should print out the text for the appropriate company

- Your program prompts the user with "Enter the first letter of the company: ", which asks for a character input.
- The input must be case sensitive, e.g., if the user enters 'b' instead of 'B', the output should be invalid and not "BMW".
- Your program prints output according to the following:
  - If the input is 'B', print "BMW"
  - If the input is 'V', print "Volkswagen"
  - If the input is 'H', print "Honda"
  - If the input is 'T', print "Tesla"
  - Any input value that is not 'B', 'V', 'H', or 'T' should print "Invalid"

Note: Code should NOT contain any if-else statements and should only utilize Switch Statements

Sample Run 4.1.1

```
Enter the first letter of the company:
B
Automobile manufacturer chosen: BMW
```

Sample Run 4.1.2

```
Enter the first letter of the company:
V
Automobile manufacturer chosen: Volkswagen
```

Sample Run 4.1.3

```
Enter the first letter of the company:
A
Automobile manufacturer chosen: Invalid
```

Sample Run 4.1.4

```
Enter the first letter of the company:
T
Automobile manufacturer chosen: Tesla
```

4.2 Instrument Price

You want to learn to play an instrument, but you need to know how much it will cost to buy it. The music store has the following table on their website:

Category	Instrument	Price
Brass	Trumpet	$570
	Trombone	$500
Woodwind	Flute	$425
	Saxophone	$225
Percussion	Snare Drum	$275
	Cymbals	$350

Write a menu-driven program that asks the user to input an instrument category and then an instrument. The program should give the user the price.

The user should input an integer in the range of the choices you give them (for example, a user cannot input 3 if you only have 2 choices). If the user inputs the correct range, the program should be prompted for the next set of choices. Once they make the final selection, the total should be printed to them as an integer with proper formatting, as shown in the sample run.

Note: Code should NOT contain any if-else statements and should only utilize Switch Statements

Sample Run 4.2.1

```
Select a category: (1)Brass (2)Woodwind (3)Percussion
1
Select an instrument: (1)Trumpet (2)Trombone
2
Your instrument will be $500
```

Sample Run 4.2.2

```
Select a category: (1)Brass (2)Woodwind (3)Percussion
3
Select an instrument: (1)Snare Drum (2)Cymbals
2
Your instrument will be $350
```

Sample Run 4.2.3

```
Select a category: (1)Brass (2)Woodwind (3)Percussion
5
Please enter a valid input.
```

Sample Run 4.2.4

```
Select a category: (1)Brass (2)Woodwind (3)Percussion
2
Select an instrument: (1)Flute (2)Saxophone
1
Your instrument will be $425.
```

4.3 Movie Night

You are preparing a movie night with your friends. Create a C++ program that helps you determine which movie to select.

Based on the given table, you, as the programmer, should give step-by-step choices to the user to proceed

further and make a movie selection. Once the user has selected the movie, the program should display a message: "You have selected the movie: <movie title>" where "<movie title>" is the movie the user has selected.

Additionally, ensure that the user's input is present in the range of choices. If the user inputs an invalid option, print "Please enter a valid input" and terminate the program.

Genre	Directors	Movies
(1) Animation	(1) Pete Docter	(1) Monsters, Inc.
		(2) Inside Out
	(2) Brad Bird	(1) The Incredibles
		(2) Ratatouille
	(3) Andrew Stanton	(1) Finding Nemo
		(2) WALL-E
(2) Adventure	(1) Steven Spielberg	(1) E.T. the Extra-Terrestrial
		(2) The BFG
	(2) Jon Favreau	(1) The Jungle Book (2016)
		(2) Elf
	(3) Robert Zemeckis	(1) Back to the Future
		(2) Who Framed Roger Rabbit

Sample Run 4.3.1

```
Select the genre: (1) Animation (2) Adventure
1
Select the director: (1) Pete Docter (2) Brad Bird (3) Andrew Stanton
1
Select the movie: (1) Monsters, Inc. (2) Inside Out
1
You have reserved the movie: Monsters, Inc.
```

Sample Run 4.3.2

```
Select the genre: (1) Animation (2) Adventure
2
Select the director: (1) Steven Spielberg (2) Jon Favreau (3) Robert Zemeckis
1
Select the movie: (1) E.T. the Extra-Terrestrial (2) The BFG
1
You have reserved the movie: E.T. the Extra-Terrestrial
```

Sample Run 4.3.3

```
Select the genre: (1) Animation (2) Adventure
4
Please enter a valid input
```

Sample Run 4.3.4

```
Select the genre: (1) Animation (2) Adventure
2
Select the director: (1) Steven Spielberg (2) Jon Favreau (3) Robert Zemeckis
2
```

```
Select the movie: (1) The Jungle Book (2016) (2) Elf
2
You have reserved the movie: Elf
```

4.4 Area of a room

You work in a construction company and have been tasked to develop a reusable C++ function that will allow engineers to calculate the area of the room they wish to renovate. For simplicity, we'll imagine every room as a rectangle.

The engineers will be asked to input the room's length and width in feet. Then, the dimensions will be passed into the function `calculateRoomArea()`, which will compute and return the area of the room.

Complete and submit the `calculateRoomArea()` function and `main()` to coderunner. You do not need to include the header.

```
double calculateRoomArea(double length, double width){

}
```

Function: `calculateRoomArea(double, double)`	`double calculateRoomArea(double length, double width)`
Purpose:	To calculate the area of the room.
Parameters:	**double** `length` - the length of the room. **double** `width` - the width of the room.
Return Value:	If successful, it returns the area of the room.
Error Handling:	- If `length` or `width` is non-positive, -1 is returned. Then, in `main()`, the program should display "Length or width is invalid. Area cannot be calculated."

Table 4.1: Function Details for calculateRoomArea

Sample Run 4.4.1

```
Enter the length of the room in ft:
4.5
Enter the width of the room in ft:
2
The area is: 9 sq ft.
```

Sample Run 4.4.2

```
Enter the length of the room in ft:
30
Enter the width of the room in ft:
40
The area is: 1200 sq ft.
```

Sample Run 4.4.3

```
Enter the length of the room in ft:
2
```

```
Enter the width of the room in ft:
0
Length or width is invalid. Area cannot be calculated.
```

Sample Run 4.4.4

```
Enter the length of the room in ft:
-10
Enter the width of the room in ft:
2
Length or width is invalid. Area cannot be calculated.
```

4.5 Estimate Sowing Time

Write a C++ program with a function to estimate the time it takes to sow seeds on farmland. The farming company provides 4 different kinds of sowing machines. Each machine can sow seeds at a different speed per square foot, as shown below.

Sowing Machine	Time taken per square foot
W	8 sq ft per 12 minutes
X	3 sq ft per 10 minutes
Y	2 sq ft per 7 minutes
Z	7 sq ft per 8 minutes

Your program should ask the user the area of the farm and the type of sowing machine the user intends to use in `main()`. Once the information is obtained, call the `calculateSowingTime()` function and pass in the appropriate values.

For submission, please paste in the whole program into coderunner. Please make sure `calculateSowingTime()` function exists and takes in the two parameters in the same order. Below is the function specification for `calculateSowingTime()`:

Function: `calculateSowingTime(double, char)`	`double calculateSowingTime(double area, char machine_type)`
Purpose:	To calculate the time taken to plant seeds all over the farmland.
Parameters:	**double** `area` - the area of the farmland. **char** `machine_type` - the type of machine selected.
Return Value:	If successful, it returns the time taken to plant seeds using a particular machine across the entire farmland.
Error Handling:	- If `area` is non-positive, 0 is returned. - If `machine_type` is invalid, 0 is returned.

Table 4.2: Function Details for calculateSowingTime

Sample Run 4.5.1

```
Enter area of the farmland in sq ft:
1200
Enter the type of sowing machine to be used:
W
The time taken is: 1800 minutes.
```

Sample Run 4.5.2

```
Enter area of the farmland in sq ft:
3244
Enter the type of sowing machine to be used:
X
The time taken is: 10813.33 minutes.
```

Sample Run 4.5.3

```
Enter area of the farmland in sq ft:
5000
Enter the type of sowing machine to be used:
A
Area or machine type is invalid. Time cannot be calculated.
```

Sample Run 4.5.4

```
Enter area of the farmland in sq ft:
1000.45
Enter the type of sowing machine to be used:
Z
The time taken is: 1143.37 minutes.
```

Week 5: Loops

Learning Goals

This week you will:

1. Learn about while loops
2. Learn about do-while loops
3. Learn about for loops

1 Background

Loops are a tool that allows us to repeat a block of code. The number of repetitions is controlled by the conditions of the loop, and can range anywhere from never executing, executing a fixed number of times, or executing an infinite number of times. Loops will execute for as long as their condition is satisfied; once the condition is not true, the loop will stop and the program will continue. The structure of these conditions can vary depending on the type of loop used. There are three main types of loop we use in C++; the while loop, the do-while loop, and the for loop.

1.1 While Loops

While loops are the most basic form of loops. Their structure looks like:

```
while (condition)
{
    //code to execute
}
```

Here, *while* is a C++ reserved word, *condition* should be a Boolean expression that will evaluate to either **true** or **false**, and the comment between the brackets is where we would add code to execute. If the condition is true, then the specified statement(s) within the loop are executed. After running once, the Boolean expression is re-evaluated. If the condition is true, the specified statement(s) are executed again. This process of evaluation and execution is repeated until the condition becomes **false**.

Example 1.1.1. Here is an example of a while loop where the condition is based on user input.

```
int userChoice = 1;
while (userChoice != 0)
{
   cout << "Do you want to see the question again?" << endl;
   cout << "Press 0 if no, any other number if yes." << endl;
   cin >> userChoice;
}
```

Entering 0 will terminate the loop, but any other number will cause the loop to execute again. Note

how we must initialize the condition before the loop starts. Setting userChoice = 1 ensures that the while loop will run at least once.

Example 1.1.2. Here is an example of a while loop where the condition is based on a counter.

```
int i = 0;
while (i < 5)
{
    cout << i << endl;
    i = i + 2;
}
```

Notice how you must manually initialize i=0 and then manually increment i by 2.

1.2 Do-While Loops

The do-while loop is a variant of the while loop. The critical difference with a do-while loop is that the block of code we wish to execute is written before the condition. The structure of a do-while loop looks like this:

```
do {
  // code block to be executed
}
while (condition);
```

In a do-while loop, the block of code is executed once before the condition is checked. This means we will never entirely skip a do-while loop. You will often find this type of loop to be useful when gathering user input, like the example shown below.

Example 1.2.1. Here is an example of the while loop from example 1.1.1. rewritten as a do-while loop.

```
int userChoice;
do {
   cout << "Do you want to see the question again?" << endl;
   cout << "Press 0 if no, any other number if yes." << endl;
   cin >> userChoice;
}
while (userChoice != 0);
```

Note that here we do not have to initialize user choice before the loop begins.

1.3 For Loops

You will frequently come across instances where you already know the number of iterations you would like your loop to complete, like example 1.1.2 shown above. In these cases, there is a special loop that has a counter built in rather than needing to keep track on your own. For loops have three elements:

- Initialization: It must initialize a counter variable to a starting value. Note: When initializing your counter, you can choose to either declare a new variable or use an existing variable. If you declare a new variable in your initialization, it will only exist in the **scope** of the loop; this means once the loop concludes, you cannot access that variable again.
- Condition: If it is true, then the body of the loop is executed. If it is false, the body of the loop does not execute and jumps to the next statement(s) just after the loop.

- Update: Updates the counter variable during each iteration

The basic structure of a for loop looks like this:

```
for (initialization; condition; update)
{
    //code to execute
}
```

Here, *for* is a C++ reserved word.

Example 1.3.1. Here is a section of code that will simply print the word "hello" five times:

```
for (int count = 0; count < 5; count++)
{
    cout << "hello" << endl;
}
```

Notice the following three parts of the for loop:

- count is initialized to 0,
- the conditional expression is count < 5
- count++ to increment the count value by one

Example 1.3.2. Here is an example that would work equivalently to example 1.1.2.:

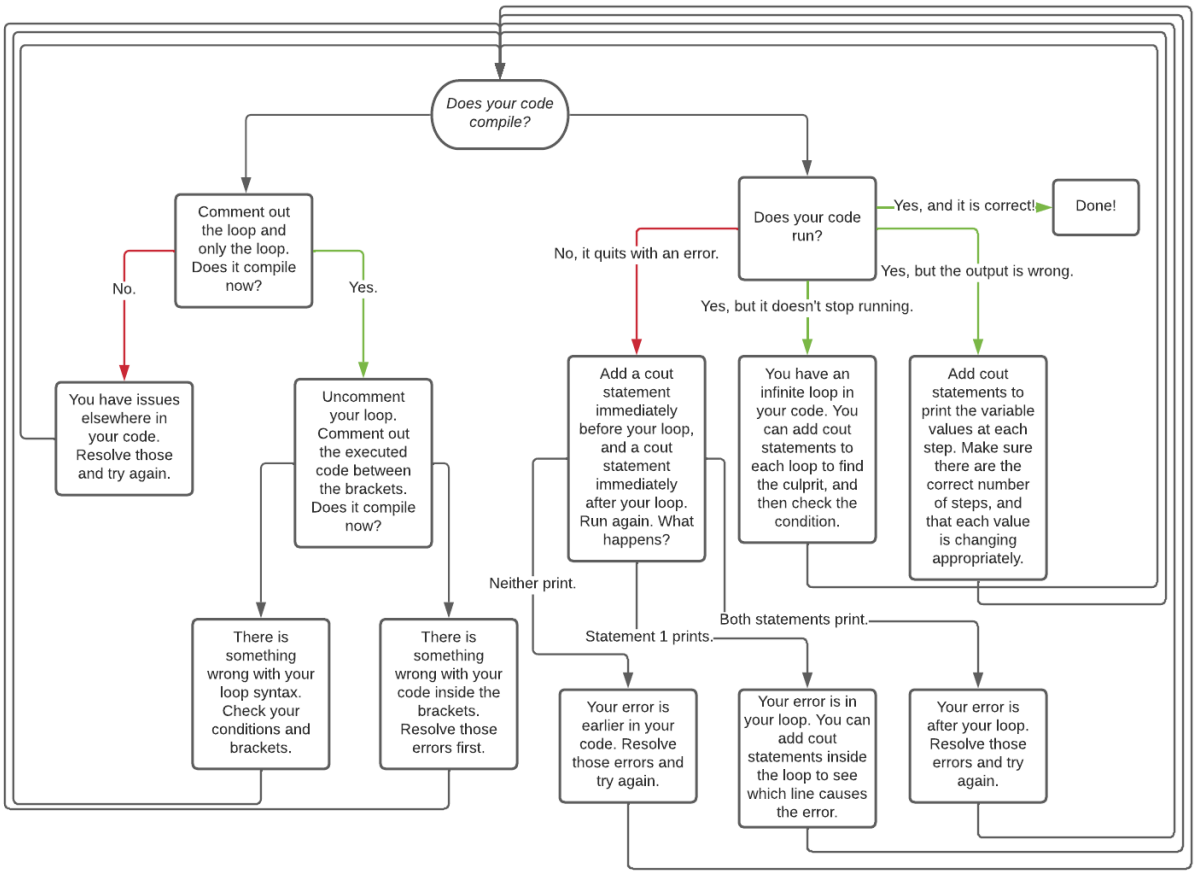
```
for (int i = 0; i < 5; i = i + 2)
{
    cout << i << endl;
}
```

1.4 Common Errors and Debugging Loops

Unique errors with loops include:

- errors with the loop syntax itself, such as your condition or the set of statements in your for loops. Check that you have semicolons and have declared all your variables that you use in this area.
- infinite loops, which will frequently present as either a program that never stops running or an error saying something along the lines of memory exceeded or time exceeded. Check your conditions, and make sure the variables that control your condition are updating correctly.
- incorrect numbers of iterations, which means you may not be updating your counter correctly or your logic may be incorrect.

Here is a flowchart for helping you figure out (and resolve) what is wrong with your loops:



Any errors that you may come across either within the bracketed section of your loop or outside of your loop are subject to the same rules as any code we have previously worked with. If you are struggling with errors in those areas, revisit debugging tips from the relevant sections to help you.

2 PreQuiz

Problem 2.1. Explain the difference between defining a function and calling a function in C++. Provide an example of each.

Problem 2.2. What does the keyword `void` signify when used as a function's return type? When would you use it?

Problem 2.3. In your own words, explain why testing your functions matters. Make sure to mention boundary conditions in your explanation.

Problem 2.4. Fill in the blank(s) for the code below so the function successfully returns the square of the number passed to the function:

```
______ floatSquare(____ number) {
    return number * number;
}
```

Problem 2.5. Fill in the blank(s) for the code below such that the num1 is successfully divided by num2:

```
    int integerDivision(______ num1, _____ num2){
        return num1/num2;
    }
```

Problem 2.6. Fill in the blank(s) for the code below so that computeArea is successfully called in the blank below, passing in the variables length and width:

```
double computeArea(double length, double width) {
    return length * width;
}

int main() {
    double length = 5.0;
    double width = 3.0;

    // Fill in the blank to call computeArea and store the result
    double area = _________________________________;

    cout << "The area is: " << area << endl;
    return 0;
}
```

Problem 2.7. Identify the error in the error in our test for the isEven function implemented below:

```
#include <iostream>
#include <cassert>
using namespace std;

bool isEven(int num) {
    return num % 2 == 0;
}

int main() {
    assert(isEven(4) = true);
    return 0;
}
```

3 Recitation

3.1 Spot The Error

Problem 3.1.a. The program below will display the average of three values by calling the function `findMean`. Identify the error(s) in the code below, and write the correct line(s).

```
#include <iostream>

using namespace std;
```

```
double findMean(int a, int b, int c)
{
    int mean = (a+b+c) / 3.0;
    return mean;
}

int main()
{
    int average = avg(2,5,2);
    assert(average == 3)
    return 0;
}
```

Problem 3.1.b. The program below checks if the two strings given are the same. Identify the error(s) in the code below, and write the correct line(s).

```
#include <iostream>
#include <string>
using namespace std;

string passwordMatchCheck(string password, string confirmPassword)
{
    return password = confirmPassword;
}

int main()
{
   bool passwordMatch = passwordMatchCheck('Good',"Morning");
   cout << passwordMatch << endl;
}
```

Problem 3.1.c. The same company uses a member ID as the username for its employees. The employees all have an eight digit member ID, and the member ID cannot start with a 0.

```
#include <iostream>
#include <cassert>
using namespace std;

bool idLengthCheck(int ID)
{
    if (ID >= 9999999 || ID < 100000000)
    {
        return true;
    }
    return false;
}

int main()
{
    assert(idLengthCheck(12345678));
    assert(idLengthCheck("123456789") == False);
    return 0;
}
```

Problem 3.1.d. The program below will use two functions: one to check for password match and another to check if the ID is valid before registering the user. Assume the relevant functions have been defined successfully. Identify the error(s) in the code below, and write the correct line(s).

```cpp
#include <iostream>
#include <string>
#include <cassert>
using namespace std;

bool passwordMatchCheck(char, char);
bool idLengthCheck(char);

int main() {
    int ID;
    string password;
    string confirmPassword;

    cout << "Enter your member ID: ";
    cin >> ID;
    assert(idLengthCheck(ID));

    cout << "Enter your password: ";
    cin >> password;

    cout << "Confirm your password: ";
    cin >> confirmPassword;

    if (passwordMatchCheck(password, confirmPassword))
    {
        cout << "Password set successfully for " << username << "." << endl;
    }
    else if (!passwordMatchCheck(password, confirmPassword))
    {
        cout << "Passwords do not match." << endl;
    }
    else if(!idLengthCheck(ID))
    {
        cout << "ID is invalid." << endl;
    }

    return 0;
}

bool passwordMatchCheck(string password, string confirmPassword)
{
    // appropriate definitions
}

bool idLengthCheck(string password)
{
    // appropriate definitions
}
```

Problem 3.1.e. The program below is a working program that uses the `getPrice` function to compute the price of a wall frame of a given area and color. This code does not contain any syntax or logical errors. However, it has multiple style errors making the code very difficult to read. These errors can range from usage of unintended white space to having extraneous variables or clauses in your code. Identify the style error(s) in the code below and rewrite the below code to improve readability.

```cpp
#include <iostream>
#include <string>
#include <cassert>
using namespace std;

double getPrice(double area, string color){
assert(area>=0); double cost = 0.0;
if (color == "green"){
    cost = 4; }
    else if (color == "red")
    { cost = 3; }
    else if (color == "orange")
    {
    cost = 2;
    }
    else if (color == "blue")
    {
        cost = 1;
    } return area * cost;}

int main()
{
    string color, shape;
    int area_choice;
    double radius;
    double area = 0;

cout << "Enter the area of the frame: (1) 5x5 (2) 4x6 (3) 8x10" << endl;
cin >> area_choice;
    assert(
        area_choice == 1 || area_choice == 2 || area_choice == 3
    );
    if(area_choice == 1){area = 5*5; }
    else if (area_choice == 2){area = 4*6; }
    else if (area_choice == 3){area = 8*10; }

cout << "Enter the color of the frame: (green, red, orange, blue): ";
cin >> color;
    assert(
        color == "green" || color == "red" || color == "orange" || color == "blue"
    );

    double price = getPrice(area, color);

    cout << "You will receive a "<< color << " color frame with a price of $" << price << ". ";
    cout << "Thank you for your business."<<endl;

    return 0;
}
```

3.2 Halloween

In the mysterious town of "Mathville", surrounded by eerie forests, the annual "Halloween Night" celebration is approaching. This town is renowned for blending mathematics with the art of candy making, creating

unique candies adorned with mathematical designs.

As the spooky mastermind behind this exciting venture, you're tasked with ensuring the success of Halloween Night by calculating the ingredients for your candies. To achieve this, you'll employ two specialized functions to accurately calculate candy volumes of the pumpkin-shaped candies and the witch's hat candies.

- Equation for pumpkin candy volume (approximated as an ellipsoid): $Volume = \frac{4}{3}\pi abc$, where a, b, and c are the radii along the x, y, and z axes respectively.
- Equation for witch's hat candy volume (approximated as a cone): $Volume = \frac{1}{3}\pi r^2 h$, where r is the radius of the base, and h is the height.

```
/**
@brief Function to determine the volume of a pumpkin-shaped candy using its radii
@param radiusX Radius along the x-axis
@param radiusY Radius along the y-axis
@param radiusZ Radius along the z-axis
@return double - volume of the pumpkin-shaped candy
*/ double calculateVolumeOfPumpkinCandy(double radiusX, double radiusY, double radiusZ) { // Your
↪  code goes here. }

/**
@brief Function to determine the volume of a witch's hat-shaped candy using its base radius and
↪  height
@param radius Base radius of the witch's hat-shaped candy
@param height Height of the witch's hat-shaped candy
@return double - volume of the witch's hat-shaped candy
*/ double calculateVolumeOfWitchHatCandy(double radius, double height) { // Your code goes here.
↪  }
```

Problem 3.2.a. Write out the steps you would use to solve this problem by hand as pseudocode.

Problem 3.2.b. Pick two possible inputs for each of your two functions (four total). Follow the steps you wrote for these values to find your result, and verify it.

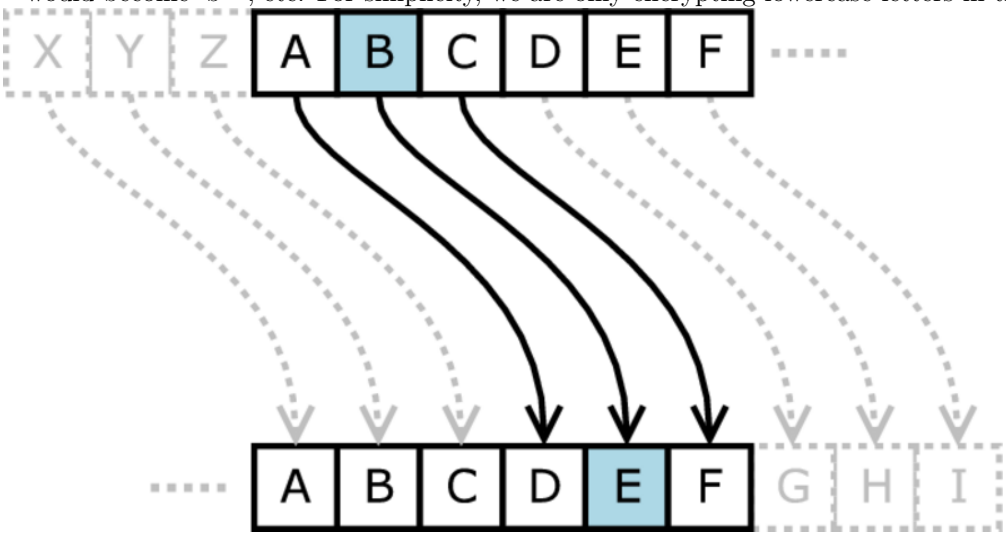
Problem 3.2.c. Translate your inputs and expected outputs into assert statements.

Problem 3.2.d. Translate your pseudocode into a c++ program to solve the above code, using your assert statements in your main function to verify that your program works as expected.

4 Homework

4.1 Encrypting Lowercase Characters

The Caesar cipher shifts each letter by a fixed number of positions within the alphabet. The shift value can be positive or negative and should wrap around the alphabet if necessary. For example, with a shift of 3, 'a' would be replaced by 'd' , 'b' would become 'e' , and so on. If the shift is -3, 'd' would be replaced by 'a' , 'e' would become 'b' , etc. For simplicity, we are only encrypting lowercase letters in this question.



Note: Characters like 'a' and 'z' have specific ASCII values ('a' is 97, 'z' is 122). The code uses these values indirectly by operating on character arithmetic rather than directly manipulating ASCII values.

Write a program that encrypts lowercase letters using the Caesar cipher. Prompt the user for a letter and a shift value in `main()`, and use the function specification listed below for encryption.

Function: `encryptLower(char, int)`	`char encryptLower(char letter, int shift_value)`
Purpose:	Encrypt the letter using the shift value. The function should not print anything.
Parameters:	**char** `letter` - The letter that is being encrypted. **int** `shift_value` - The shift value.
Return value:	If successful, returns the newly encrypted letter.
Error handling/ Boundary conditions:	- If the unencrypted letter is not lowercase, then the unencrypted letter is returned. In other words, return the original letter if it's not lowercase. - Hint: You may use the modulo (%) operation to handle the wrap-around, e.g., if the letter is a and the shift value is -1, the encrypted letter should result in z.

Example:	

	`// This is only an example usage, and you should develop your own main function`
	`// Assume the proper libraries are included.`
	`// Assume the proper implementation of encryptLower() is included.`
	
	`int main() {`
	`  char letter = 'r';`
	`  char encrypted_letter = encryptLower(letter, 5);`
	`  cout << "Letter " << letter << " was encrypted to " << encrypted_letter;`
	`  return 0;`
	`}`

Sample Output (from above example):	Letter r was encrypted to w

Develop and validate your solution in VS Code. Once you are happy with your solution, go to coderunner on Canvas and paste the whole program into the answer box!

Sample Run 4.1.1

```
Enter the lowercase character to encrypt:
a
Enter the encryption value:
-1
Letter a was encrypted to z
```

Sample Run 4.1.2

```
Enter the lowercase character to encrypt:
f
Enter the encryption value:
-3
Letter f was encrypted to c
```

Sample Run 4.1.3

```
Enter the lowercase character to encrypt:
a
Enter the encryption value:
5
Letter a was encrypted to f
```

Sample Run 4.1.4

```
Enter the lowercase character to encrypt:
a
Enter the encryption value:
27
```

```
Letter a was encrypted to b
```

Sample Run 4.1.5

```
Enter the lowercase character to encrypt:
D
Enter the encryption value:
10
Letter D was encrypted to D
```

4.2 Building a Snowman

In this problem, you are tasked with simulating the process of three snowballs rolling down a hill to create a snowman. Each snowball must grow to a specific size to form the snowman's head, mid-body, and lower-body.

Initially, each snowball starts with a size of 1 unit and grows at a variable rate. The growth rate begins at 1 unit per second and increases by 1 unit each second (for example, the snowball grows by 1 unit in the first second, 2 units in the second second, 3 units in the third second, and so on).

You need to calculate the amount of time each snowball will roll to reach the required sizes for the snowman's head, mid-body, and lower-body. Additionally, determine the total time needed for all three snowballs to achieve their respective sizes.

Note: You should check if the size input for each body part is positive in `main()`. If the input is non-positive (i.e., zero or negative), output `"Please enter a positive integer for <section> size:"`, where <section> is part of the snowman that received invalid input, and prompt for the input again. This ensures that only valid sizes are accepted for the snowman parts.

Function: `calculateTime(int)`	`int calculateTime(int target_size)`
Purpose:	The function will calculate the time needed for the snowball to reach each part's specified target_size. The function should not print anything.
Parameters:	**int** `target_size` - The desired size that the snowball needs to reach.
Return Value:	If successful, it returns the time it takes to reach the desired size of the snowball in seconds.
Error Handling:	The error handling for desired size will occur in `main()`, which means all target_size should be valid.

Example:	`// This is only an example usage, and you should develop your own main function`
	`// Assume the proper libraries are included.`
	`// Assume the proper implementation of calculateTime() is included.`
	
	`int main() {`
	`  int head_time = calculateTime(6);`
	`  cout << "Time to reach head size: " << head_time << " seconds" ;`
	`  return 0;`
	`}`

Sample Output (from above example):	Time to reach head size: 3 seconds

Develop and validate your solution in VS Code. Once you are happy with your solution, go to coderunner on Canvas and paste the whole program into the answer box!

Sample Run 4.2.1

```
Enter head size:
6
Enter mid-body size:
12
Enter lower-body size:
22
Time to reach head size: 3 seconds
Time to reach mid-body size: 5 seconds
Time to reach lower-body size: 6 seconds
Total time to create the snowman: 14 seconds
```

Sample Run 4.2.2

```
Enter head size:
-12
Please enter a positive integer for head size:
12
Enter mid-body size:
13
Enter lower-body size:
14
Time to reach head size: 5 seconds
Time to reach mid-body size: 5 seconds
Time to reach lower-body size: 5 seconds
Total time to create the snowman: 15 seconds
```

4.3 Print Collatz Sequence

For this question, you will write a program that prompts the user to enter an integer value between 10 and 500 (both exclusive). Your program should then print out a sequence of numbers between the given value and 1 (inclusive) following the pattern below: For a given number a_i,

- If a_i is even, then $a_{i+1} = floor(\frac{a_i}{2})$
- If a_i is odd, then $a_{i+1} = 3a_i + 1$

Your program should stop printing numbers once the sequence reaches the value of 1.

If the user enters a number that is not between 10 and 500 (both exclusive), print "Invalid input." and prompt the user for a number again.

This sequence is referred to as the Collatz sequence. The Collatz Conjecture states that this sequence will eventually reach the value of 1 for any starting number, a fact that has not yet been proven by modern mathematics!

Note: floor(x) means that x should be rounded down to the nearest integer.

Develop and validate your solution in VS Code. Once you are happy with your solution, go to coderunner on Canvas and paste the whole program into the answer box!

Sample Run 4.3.1

```
Enter a number between 10 and 500:
10
Invalid input.
Enter a number between 10 and 500:
20
10
5
16
8
4
2
1
Total steps: 7
```

Sample Run 4.3.2

```
Enter a number between 10 and 500:
122
61
184
92
46
23
70
35
106
53
160
80
40
20
10
5
16
8
4
2
1
Total steps: 20
```

4.4 Potion Crafting

You are developing a crafting system for an RPG video game where players can create two types of potions: Health Potions and Magic Potions. Each type of potion requires four key ingredients: Tealeaves, Sunflowers, Toadstools, and Pine Needles. However, the quantity of each ingredient needed differs slightly between the two types of potions.

Your task is to write a program that asks the user which potion they would like to prioritize—Health or Magic. If the user enters an invalid number for the type of potion, the program should ask them to re-enter like this: "Invalid input. Please select 1 or 2." The program will then prompt the user to input the number of each ingredient they currently have (Tealeaves, Sunflowers, Toadstools, and Pine Needles). Based on the available ingredients, calculate how many potions of the priority type can be crafted. Any leftover

ingredients should be used to craft as many of the other types of potion as possible. Finally, the program should output how many of the priority potions can be crafted and how many of the other potions can be made with the remaining ingredients.

The ingredient requirements for each type of potion are as follows:

Ingredients	Amount for Health Potion	Amount for Magic Potion
Tealeaves	6	2
Sunflowers	1	3
Toadstools	5	10
Pine Needles	2	1

Develop and validate your solution in VS Code. Once you are happy with your solution, go to coderunner on Canvas and paste the whole program into the answer box!

Sample Run 4.4.1

```
Select a potion crafting priority:
1. Health Potion
2. Magic Potion
2
How many Tealeaves do you have?
30
How many Sunflowers do you have?
40
How many Toadstools do you have?
12
How many Pine Needles do you have?
22
You can make 1 Magic potion(s) and 0 Health potion(s).
```

Sample Run 4.4.2

```
Select a potion crafting priority:
1. Health Potion
2. Magic Potion
3
Invalid input. Please select 1 or 2.
2
How many Tealeaves do you have?
10
How many Sunflowers do you have?
5
How many Toadstools do you have?
8
How many Pine Needles do you have?
3
You can make 0 Magic potion(s) and 1 Health potion(s).
```

Sample Run 4.4.3

```
Select a potion crafting priority:
1. Health Potion
2. Magic Potion
```

```
1
How many Tealeaves do you have?
10
How many Sunflowers do you have?
15
How many Toadstools do you have?
19
How many Pine Needles do you have?
27
You can make 1 Health potion(s) and 1 Magic potion(s).
```

4.5 Validate Integer

Design a function `validateInt` that accepts a string input and determines if it represents a valid integer by checking if each character in the string is a valid value. Your program should ask the user to input an integer, store it as a string, and then invoke the `validateInt` function to check its validity. The program should then print whether the string is a valid integer or not. (Negative integers are also valid integers).

Function: validateInt(string)	`bool validateInt(string input)`
Purpose:	Iterate through a string and verify if it is a valid integer or not. The function should not print anything.
Parameters:	**string** `input` - The string to be verified
Return value:	It returns true if the string is a valid integer. Otherwise, it returns false.
Error handling/ Boundary conditions:	If length of input = 0, false is returned

Example:	**Example 4.5.1.**
	`// Assume the proper libraries are included.`
	
	`// Assume the proper implementation of validateInt() is included.`
	
	`int main()`
	`{`
	`  string number;`
	`  cout << "Enter the integer : " << endl;`
	`  getline(cin, number);`
	`  if(!validateInt(number))`
	`  {`
	`  cout << "The entered string is not a valid integer!!" << endl;`
	`  }`
	`  else`
	`  {`
	`  cout << "The entered string is a valid integer!!" << endl;`
	`  }`
	`  return 0;`
	`}`
	
	Sample Run 4.5.1
	`Enter the integer :`
	`1234`
	`The entered string is a valid integer!!`

Develop and validate your solution in VS Code. Once you are happy with your solution, go to coderunner on Canvas and paste just the `validateInt()` function into the answer box!

Sample Run 4.5.2

```
Enter the integer :
-12
The entered string is a valid integer!!
```

Sample Run 4.5.3

```
Enter the integer :
```

```
23e56
The entered string is not a valid integer!!
```

Sample Run 4.5.4

```
Enter the integer :
32 56
The entered string is not a valid integer!!
```

Sample Run 4.5.5

```
Enter the integer :
-
The entered string is not a valid integer!!
```

Sample Run 4.5.6

```
Enter the integer :
456789
The entered string is a valid integer!!
```

Week 6: Arrays

Learning Goals

This week you will:

1. Learn how to make arrays of any data type

1 Background

An array is a data structure which can store other data types like double, int, char, and boolean, and string. Arrays have both a type and a size.

1.1 Making and Using Arrays

How to declare arrays

```
// data_type array_name[declared_size];
bool myBooleans[10];
string myStrings[15];
int myInts[7];
```

How to initialize arrays (method 1)

```
bool myBooleans[4] = {true, false, true, true};
```

If you do not declare the size inside the square brackets, the array size will be set to however many entries you provide on the right.

```
bool myBooleans[] = {true, false, true}; // size = 3
```

Note: the size specified in the brackets needs to match the number of elements you wrote in the curly brackets.

Example 1.1.1. When the specified size is larger than the actual number of elements, the elements provided in the curly brackets will be the first several elements in the array, while the additional elements will be filled with default values. If it' s an integer/double array, the default values are zero, while if it' s a string array, the default values are empty strings.

```
#include <iostream>
using namespace std;
int main()
{
    int intArray[5] = {1,2,3};
    for (int i = 0; i < 5; i ++)
    {
        cout << intArray[i] << " ";
```

```
    }
}
```

Output:

```
1 2 3 0 0
```

Example 1.1.2. When the specified size is smaller than the actual number of elements, there will be a compilation error.

```
#include <iostream>
using namespace std;
int main()
{
    int intArray[3] = {1,2,3,4,5};
}
```

Output:

```
error: excess elements in array initializer
int intArray[3] = {1,2,3,4,5};
                          ^
1 error generated.
```

How to Initialize Arrays (Method 2) You can also initialize elements one by one using a for loop:

```
int myInts[10];
for (int i = 0; i < 10; i++)
{
    myInts[i] = i;
}
//{0, 1, 2, 3, 4, 5, 6, 7, 8, 9}
```

How to Access Elements in an Array We have essentially already had practice with accessing elements in an array, as in C++, string is basically an array of characters. You can access elements in arrays using the same syntax you used for strings:

```
string greetings[] = {"hello", "hi", "hey", "what' s up?"};
cout << greetings[3] << endl;
```

Arrays and strings can also be iterated through in the same way.

Example 1.1.3. Iterating through an array:

```
string greetings[] = {"hello", "hi", "hey", "what' s up?"};
int size = 4;
for (int i = 0; i < size; i++)
{
    cout << greetings[i] << endl;
}
```

Iterating through a string:

```
string greeting = "Hello world!";
for (int i = 0; i < greeting.length(); i++){
```

```
        cout << greeting[i] << ", " << endl;
    }
```

2 PreQuiz

Problem 2.1. How does a "for" loop differ from a "while" loop?

Problem 2.2. What kind of loop is this loop an example of? Hint: You don't want to run this code on your machine.

```
#include <iostream>

int main() {
    while (true) {
        std::cout << "We are in the loop!" << std::endl;
    }
    return 0;
}
```

Problem 2.3. You have seen function tables designed to convey the information in a function on previous assignments. Now, given the code below, fill in the following table for the `createPyramid()` function. The function creates a pyramid pattern of asterisks (stars) with the given number of rows and returns the total number of stars printed.

```
#include <iostream>
#include <cassert>
using namespace std;

int createPyramid(int rows)
{
    if (rows > 0)
    {
        int totalStars = 0;
        for (int i = 1; i <= rows; i++)
        {
            // Print spaces
            for (int j = i; j < rows; j++)
                cout << "  ";
            // Print stars
            for (int k = 1; k <= (2 * i - 1); k++)
            {
                cout << "* ";
                totalStars++;
            }
            cout << endl;
        }
        return totalStars;
    }
    return 0;
}
```

Function: createPyramid(int)	
Purpose:	
Parameters:	
Return value:	
Error handling/ Boundary conditions:	
Example:	

```
int main()
{
    int totalStars = createPyramid(3);
    cout << "Total stars printed: " << totalStars << endl;
    return 0;
}
```

Sample Run 2.0.1

```
*
* * *
* * * * *
Total stars printed: 9
```

Problem 2.4. Write 5 assert statements for the function shown in the previous question.

Problem 2.5. The program intends to count and display the number of days in a given month that fall on **Tuesdays or Thursdays**.

- How many iterations does the for loop perform?
- How many times does the if condition evaluate to true?

```
#include <iostream>
using namespace std;


int countTuesdaysThursdays(int month_days)
{
    int count = 0;
    for (int day = 1; day <= month_days; day++)
    {
        // Check if the day falls on a Tuesday or Thursday
        if (day % 7 == 2 || day % 7 == 4)
        {
            count++;
        }
    }
return count;
}

int main() {
    int month_days = 31;
    int count = countTuesdaysThursdays(month_days);
```

```
    cout << "The number of Tuesdays and Thursdays in the month is: " << count << endl;
    return 0;
}
```

3 Recitation

3.1 Spot The Error

Problem 3.1.a. The program intends to prints all even numbers from 2 to N (both inclusive). Identify the error(s) in the code below, and write the correct line(s).

```
#include <iostream>
using namespace std;

void printEvenNumbers(int N)
{
    int i = 2;
    while (i <= N)
    {
        if (i % 2 == 0)
        {
            cout << i << " ";
        }
    }
    j++;
    return;
}

int main()
{
    int number;
    cout << "Enter a number: " << endl;
    cin >> number;
    printEvenNumbers(number);
    return 0;
}
```

Problem 3.1.b. The program monitors the pump status and fills the liquid until the liquid level reaches the threshold. Identify the error(s) in the code below, and write the correct line(s).

```
#include <iostream>
using namespace std;

int main()
{
    int liquid_level = 5;
    int threshold = 60;

    while (liquid_level <= threshold)
    {
        cout << "Pump is running. Liquid level: " << liquid_level << " units." << endl;
        liquid_level -= 5;
    }

    cout << "Pump stopped. Liquid level: " << liquid_level << " units." << endl;
```

```
    return 0;
}
```

Problem 3.1.c. The program intends to verify if two strings match, prompting the user to re-type the string until it matches the initial string. Identify the error(s) in the code below, and write the correct line(s).

```
#include <iostream>
#include <string>
using namespace std;

int main()
{
    string initial_string;
    string retype_string;

    cout << "Enter your string: ";
    cin >> initial_string;

    cout << "Enter your string again: ";
    cin >> retype_string;

    while (retype_string = initial_string)
    {
        cout << "Your strings do not match. Try again: " << endl;
        cin >> retype_string;
    }

    cout << "Your strings match!!!!" << endl;
    return 0;
}
```

Problem 3.1.d. The program intends to find the sum of all numbers from 1 to N (inclusive) and prints the result. Identify the error(s) in the code below, and write the correct line(s).

```
#include <iostream>
using namespace std;

int totalSum(int n)
{
    int result = 1;
    for(i = 1; i <= n; i + 1)
    {
        result += i;
    }
    return result;
}

int main()
{
    int n;
    cout << "Enter a positive integer (n): ";
    cin >> n;

    int result = totalSum(n);
```

```
    cout << "Sum of numbers from 1 to " << n << " is: " << result << endl;

    return 0;
}
```

Problem 3.1.e. The program intends to calculate the sum of numbers entered by the user until a negative number is encountered. Identify the error(s) in the code below, and write the correct line(s).

```
#include <iostream>
using namespace std;

int main()
{
    int num;
    int sum = 0;

    do
    {
        cout << "Enter a number (enter a negative number to stop): ";
        cin >> number;
        if (number >= 0)
        {
            sum += number;
        }

    }while number >= 0;

    cout << "Sum of the entered numbers: " << sum << endl;
    return 0;
}
```

3.2 Valid Double

Design a function `validateDouble` that accepts a string input and determines if it represents a valid double by iterating through the string. Valid doubles can start with a negative and can have up to one decimal point. Your program should ask the user to input a double, store it as a string and then invoke the `validateDouble` function to check its validity. The program should then print whether the string is a valid double or not. (Negative double are also valid doubles. You can reuse some parts of your `validateInt` function from recitation 5).

Function: validateDouble(string)	`bool validateDouble(string input)`
Purpose:	Iterate through a string and verify if it is a valid double or not.
Parameters:	**input** - The string to be verified
Return value:	It returns true is the string is a valid double. Otherwise returns false.
Error handling/ Boundary conditions:	If length of input = 0, false is returned

Example:	**Example 3.2.1.**
	`// Assume the proper libraries are included.`
	`// Assume the proper implementation of validateDouble() is included.`
	`int main()`
	`{`
	`  string number;`
	`  cout << "Enter the double : " << endl;`
	`  getline(cin, number);`
	`  if(!validateDouble(number))`
	`  {`
	`  cout << "The entered string is not a valid double!!" << endl;`
	`  }`
	`  else`
	`  {`
	`  cout << "The entered string is a valid double!!" << endl;`
	`  }`
	`  return 0;`
	`}`
	
	Sample Run 3.2.1
	`Enter the double :`
	`-123.4`
	`The entered string is a valid double!!`

Here are a few additional sample runs:

Sample Run 3.2.2
```
Enter the double :
```

```
-12
The entered string is a valid double!!
```

Sample Run 3.2.3

```
Enter the double :
23.56e
The entered string is not a valid double!!
```

Sample Run 3.2.4

```
Enter the double :
32 56
The entered string is not a valid double!!
```

Sample Run 3.2.5

```
Enter the double :
-.
The entered string is not a valid double!!
```

Problem 3.2.a. Write out the steps you would use to solve this problem by hand as pseudocode. Stating your modifications to your validateInt is also sufficient.

Problem 3.2.b. Pick three possible inputs for your program. Try to pick values that will test different aspects of your function. Follow the steps you wrote for these values to find your result, and verify it.

Problem 3.2.c. Translate each of the sample inputs and expected outputs you created into assert statements.

Problem 3.2.d. Implement your solution in C++ using VS Code. Revise your solution, save, compile and run it again. Are you getting the expected result and output? Keep revising until you do. Make sure you test for the values used in your sample runs, and for the boundary conditions.

Project 1: DNA Analysis

1 Introduction

You have recently been employed by Rhonda Labs, a leading research institute working to uncover the secrets of rare genetic diseases. Your first task is to write a program that analyzes DNA from various species and individuals, compares their similarities, and performs analyses on the given sequences.

Before you dive into the project, let's review a bit of biology to help you understand what you will be working with. DNA, or deoxyribonucleic acid, is the molecule that carries genetic information in almost all living organisms. It acts like a set of instructions that tells cells how to function, grow, and reproduce.

DNA is made up of four chemical bases:

- Adenine (A)
- Cytosine (C)
- Guanine (G)
- Thymine (T)

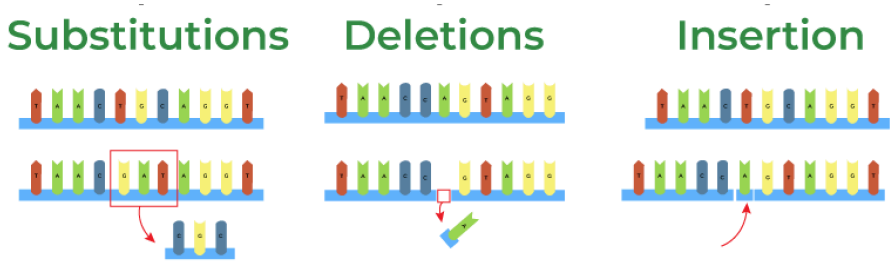


Figure 7.1: Types of DNA mutations

These bases pair together (A with T, C with G) to form the DNA double helix, which is the the overall structure of DNA. A DNA strand is made up of a sequence of these base pairs. Every organism has its own unique DNA sequence, though many organisms share similarities, especially related species. Sometimes, DNA sequences can change, leading to mutations, such as:

- Substitutions (one base is swapped for another)
- Insertions (extra bases are added)
- Deletions (bases are removed)

For this task, the steps are outlined and divided into individual questions. By the end of the project, you will have developed a cohesive program that a user can interact with by inputting their DNA sequences. Below is a breakdown of the steps:

1. Check whether your DNA strands are composed of valid bases

2. Develop functions to compare DNA strands and assess their similarity
3. Develop a function that compares two DNA sequences and identifies all types of mutations between them
4. Transcribe DNA to RNA and compute the reverse complement of a DNA strand
5. Find open reading frames within a DNA strand
6. Final step: make it more accessible and user-friendly!

2 Assignment

Warning: You are not allowed to use global variables for this project.

All function names, return types, and parameters must precisely match those shown. You may not use pass by reference or otherwise modify the function prototypes. You are welcome to create additional functions that may help streamline your code. You must well comment your code, and follow good formatting practices. You should create assert statements for each function to test your code in VS Code before moving to CodeRunner.

2.1 Question 1: `isValidBase()`

One of the first steps when working with DNA sequences is to check whether the data is valid or corrupted. You must ensure the sequences you are working with consist only of valid DNA bases: A, C, G, and T. You will write a function, `isValidBase()`, that checks if a character is a valid DNA base.

Function: `isValidBase(char)`	`bool isValidBase(char base)`
Purpose:	The function will determine whether a given character is a valid DNA base. The function should not print anything.
Parameters:	**char** `base` - The character to validate.
Return Value:	The function should return true if the character is a valid base (A, C, G, or T) and false otherwise.
Error handling/ Boundary conditions:	- The function should be case-sensitive, e.g., 'A' is a valid base, but 'a' is not. - **Note:** True is represented by 1 and false by 0 when you `cout` boolean variables.

For Question 1, develop and validate your solution in VS Code. Once you are happy with your solution, go to coderunner on Canvas and paste only the `isValidBase()` to the answer box!

Sample Runs 2.1.1

Function call	Expected return value
`isValidBase('A')`	`true`
`isValidBase('X')`	`false`
`isValidBase('a')`	`false`
`isValidBase('T')`	`true`

2.2 Question 2: `isValidStrand()`

Next, write the `isValidStrand()` function that checks if a string contains only valid DNA bases.

Function: `isValidStrand(string)`	`bool isValidStrand(string strand)`
Purpose:	The function will determine whether a given string consists only of valid DNA bases. The function should not print anything.
Parameters:	**string** `strand` - The DNA strand to validate.
Return Value:	The function should return true if the string is a valid DNA strand and false otherwise.
Error handling/ Boundary conditions:	- The input string is only considered valid if it consists only of A, C, T, and G bases. - If the string is empty, then it should not be considered a valid DNA strand, and your function should return false. - **Hint:** The function should make use of your `isValidBase` function.

For Question 2, develop and validate your solution in VS Code. Once you are happy with your solution, go to coderunner on Canvas and paste the `isValidBase()` and `isValidStrand()` to the answer box!

Sample Runs 2.2.1

Function call	Expected return value
`isValidStrand("ATGCTTCAA")`	`true`
`isValidStrand("CTTZ")`	`false`
`isValidStrand("")`	`false`
`isValidStrand("ATCG")`	`true`

2.3 Question 3: `strandSimilarity()`

Comparing DNA sequences allows researchers to identify similarities and differences that may be significant in understanding genetic relationships or disease mechanisms. In this step, you'll develop functions to compare DNA strands and assess their similarity.

For two DNA strands of equal length, the similarity score is calculated based on the number of matching bases at corresponding positions using the following formula:

$$\text{Similarity} = \frac{\text{total matches}}{\text{total positions}}$$

Example: Let's compare the following two DNA strands:

Position	Strand 1	Strand 2
1	G	G
2	A	T
3	T	T
4	C	C
5	A	A
6	G	A

Table 7.11: Positions of two DNA strands

The total number of matches is 4 out of 6 positions, resulting in a similarity score of $\frac{4}{6} = 0.667$.

The function `strandSimilarity()` compares two strands position by position, counting the number of positions where the bases are identical. This provides a direct measure of how similar the two sequences are.

Function: `strandSimilarity(string, string)`	`double strandSimilarity(string strand1, string strand2)`
Purpose:	The function will find the similarity between two DNA strands. The function should not print anything.
Parameters:	**string** `strand1` - The first DNA strand to compare. **string** `strand2` - The second DNA strand to compare.
Return Value:	The function should return the similarity score between the two strands.
Error handling/ Boundary Condition:	- The parameters should be two strings of equal length. If they are not equal in length, your function should return 0. - You may assume that the input to `strandSimilarity()` will always be a valid strand, i.e., you do not have to account for arbitrary strings.

For Question 3, develop and validate your solution in VS Code. Once you are happy with your solution, go to coderunner on Canvas and paste the `strandSimilarity()` and any helper function(s) you used to the answer box!

Sample Runs 2.3.1

Function call	Expected return value
`strandSimilarity("AGGT" , "CTGA")`	`0.25`
`strandSimilarity("CCTT" , "CCTT")`	`1`
`strandSimilarity("ATG" , "AAATTT")`	`0`
`strandSimilarity("CTGTAGAGCT" , "TAGCTACCAT")`	`0.2`

2.4 Question 4: `bestStrandMatch()`

In `bestStrandMatch()`, the strands can be different lengths, therefore, you'll want to compare overlapping sections and calculate similarity scores at each position, and to do that you'll slide the shorter strand along the longer strand. The maximum score across all positions indicates the best alignment between the two strands.

Function: `bestStrandMatch(string, string)`	`int bestStrandMatch(string input_strand, string target_strand)`
Purpose:	The function will find the best similarity between two DNA strands. The function should print out the best similarity score.
Parameters:	**string** `input_strand` - The input DNA strand to be checked against the target_strand (length greater than or equal to the target strand) **string** `target_strand` - The target DNA strand.
Return Value:	If the parameters are valid, returns an `int` representing the starting index of the substring in the input strand where the best alignment with target strand occurs.

Error handling/ Boundary conditions:	- If the input strand is shorter than the target strand, the function returns -1 as the alignment index and prints out "Best similarity score: 0.0". - This function should make use of the `strandSimilarity()` function. - You may assume that the input to `bestStrandMatch()` will always be a valid DNA sequence, i.e., you do not have to account for arbitrary strings.

For Question 4, develop and validate your solution in VS Code. Once you are happy with your solution, go to coderunner on Canvas and paste the `bestStrandMatch()` and any helper function(s) you used to the answer box!

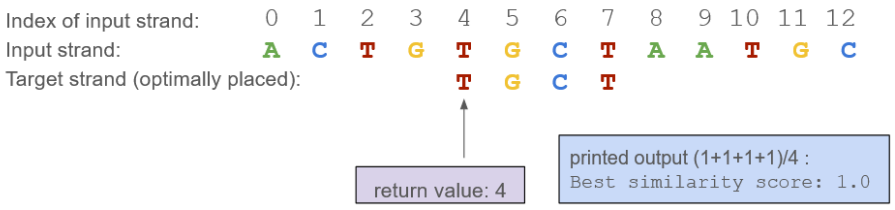


Figure 7.2: An illustration of bestStrandMatch() between two example strands

Sample Runs 2.4.1

Function call	Expected return value	Expected printed value
`bestStrandMatch("GATCAGT", "TCA")`	2	`Best similarity score: 1.0`
`bestStrandMatch("AACCTGAC", "ACT")`	1	`Best similarity score: 0.666667`
`bestStrandMatch("CTG", "CCCC")`	-1	`Best similarity score: 0.0`
`bestStrandMatch("ATCGTA", "TTCGAT")`	0	`Best similarity score: 0.5`

2.5 Question 5: Mutation Detection

Now, you want to be able to make deeper observations between DNA sequences. To do this, you will create a function that compares two DNA sequences and identifies all types of mutations between them. The function should align the sequences based on the best possible match and then process the sequences character by character, printing out mutations as they are detected.

Your function should be able to identify the following mutations:

- Substitution: When bases at the same position differ
- Insertion: When an extra base is present in the target strand
- Deletion: When a base from the input strand is missing in the target strand

The function should determine the longest of the two strands and use the `bestStrandMatch()` function to optimally align them. Upon alignment, the function should print out the best alignment index. After alignment, it should compare the sequences character by character to identify mutations. It detects substitutions

when bases at the same aligned position differ, deletions when extra bases are present in the input strand but not in the target strand, and insertions when bases are present in the target strand but missing from the input strand.

Function: identifyMutations(string, string)	`void identifyMutations(string input_strand, string target_strand)`
Purpose:	The function compares two DNA sequences to identify all types of mutations between them. It aligns the sequences based on the best possible match and processes them character by character, printing out any mutations as they are detected.
Parameters:	**string** `input_strand` - The input strand to be checked against the target **string** `target_strand` - The target strand
Return Value:	N/A
Error handling/ Boundary conditions:	-You may assume that the input and target strands are both valid DNA strands. - If no mutations are found, the function outputs "No mutations found."

For Question 5, develop and validate your solution in VS Code. Once you are happy with your solution, go to coderunner on Canvas and paste `identifyMutations()` and any helper function(s) to the answer box!

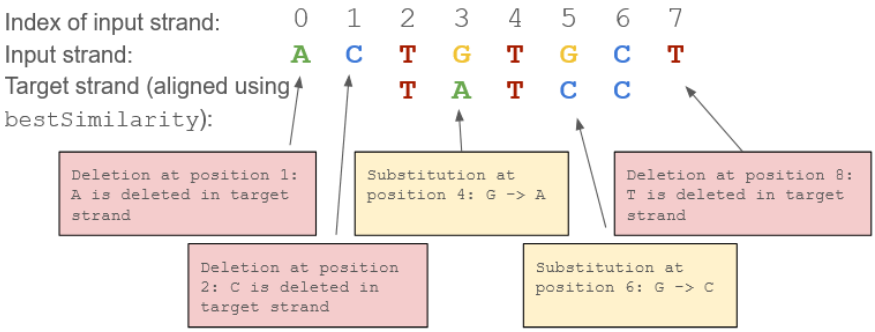


Figure 7.3: An illustration of identifyMutation() between two example strands

Sample Runs 2.5.1

Note: "Best Similarity Score:..." print line should come from your bestStrandMatch() function.

Function call	Expected printed value

`identifyMutations("GGG", "TAGGGTA")`	`Best similarity score: 1`
	`Best alignment index: 2`
	`Insertion at position 1: T is inserted in target strand`
	`Insertion at position 2: A is inserted in target strand`
	`Insertion at position 6: T is inserted in target strand`
	`Insertion at position 7: A is inserted in target strand`

`identifyMutations("AACCGG", "AACCGG")`	`Best similarity score: 1`
	`Best alignment index: 0`
	`No mutations found.`

`identifyMutations("TA", "TAGG")`	`Best similarity score: 1`
	`Best alignment index: 0`
	`Insertion at position 3: G is inserted in target strand`
	`Insertion at position 4: G is inserted in target strand`

`identifyMutations("TAGG", "TA")`	`Best similarity score: 1`
	`Best alignment index: 0`
	`Deletion at position 3: G is deleted in target strand`
	`Deletion at position 4: G is deleted in target strand`

`identifyMutations("AGTCACG", "AGCTACA")`	`Best similarity score: 0.571429`
	`Best alignment index: 0`
	`Substitution at position 3: T -> C`
	`Substitution at position 4: C -> T`
	`Substitution at position 7: G -> A`

2.6 Question 6: DNA Sequence Transformations

In this part, you will simulate the transcription process by converting a DNA sequence into an RNA sequence. This involves replacing every occurrence of thymine ('T') with uracil ('U') in the DNA strand.

Function: `transcribeDNAtoRNA(string)`	`void transcribeDNAtoRNA(string strand)`
Purpose:	The function will transcribe a DNA sequence to RNA and print the RNA sequence to the console. The function will replace all occurrences of 'T' with 'U'.
Parameters:	**string** `strand` - The DNA sequence to be transcribed.
Return Value:	N/A
Error handling/ Boundary conditions:	- You may assume that the input DNA strand is a valid DNA sequence.

For Question 6, develop and validate your solution in VS Code. Once you are happy with your solution, go to coderunner on Canvas and paste `transcribeDNAtoRNA()` and any helper function(s) to the answer box!

Sample Runs 2.6.1

Function call	Expected printed value
`transcribeDNAtoRNA("ATCGTACG")`	`AUCGUACG`
`transcribeDNAtoRNA("TTAA")`	`UUAA`
`transcribeDNAtoRNA("ACCG")`	`ACCG`
`transcribeDNAtoRNA("T")`	`U`

2.7 Question 7: Reverse Complement of a DNA Sequence

In this part, you will write a function to compute the reverse complement of a DNA strand. The reverse complement is obtained by reversing the DNA sequence and then replacing each base with its complement:

- 'A' is complemented by 'T'
- 'T' is complemented by 'A'
- 'C' is complemented by 'G'
- 'G' is complemented by 'C'

Function: `reverseComplement(string)`	`void reverseComplement(string strand)`
Purpose:	The function will compute the **reverse** complement for a DNA sequence and print the result to the console.
Parameters:	**string** `strand` - The DNA sequence for which the reverse complement will be computed.
Return Value:	N/A
Error handling/ Boundary condition:	You may assume that the input DNA strand is a valid DNA sequence.

For Question 7, develop and validate your solution in VS Code. Once you are happy with your solution, go to coderunner on Canvas and paste the `reverseComplement()` and any helper function(s) you used to the answer box!

Sample Runs 2.7.1

Function call	Expected printed value
`reverseComplement("ATCGTACG")`	`CGTACGAT`
`reverseComplement("AAACCC")`	`GGGTTT`
`reverseComplement("AGCT")`	`AGCT`
`reverseComplement("A")`	`T`

2.8 Question 8: Extracting Reading Frames

When working with DNA sequences, it is often necessary to focus on specific regions called open reading frames (ORF). These DNA regions are read in groups of three bases, called codons, which are later translated into amino acids during the process of protein synthesis. An ORF is a continuous sequence of DNA that begins with a start codon (ATG) and ends with a stop codon (TAA, TAG, or TGA).

Your task is to identify valid protein-coding regions in a DNA sequence. A valid ORF starts with the ATG codon and ends with one of the stop codons (TAA, TAG, or TGA), with the number of bases between the start and stop codons being divisible by 3.

Function: `getCodingFrames(string)`	`void getCodingFrames(string strand)`
Purpose:	The function will print out complete reading frames.
Parameters:	**string** `strand` - The DNA strand from which to extract reading frames.
Return Value:	N/A
Error handling/ Boundary condition:	- If no reading frames are found, the function should print "No reading frames found." - You may assume that the input DNA strand is a valid DNA sequence. - **Note:** There could be multiple ORF within a single DNA strand.

For Question 8, develop and validate your solution in VS Code. Once you are happy with your solution, go to coderunner on Canvas and paste the `getCodingFrames()` and any helper function(s) you used to the answer box!

Sample Runs 2.8.1

Function call	**Expected printed value**
`getCodingFrames("ATGCGGTAA")`	`ATGCGGTAA`
`getCodingFrames("AAACCC")`	`No reading frames found.`
`getCodingFrames("ATGCGTAGCTAAATGGGGTAG")`	`ATGCGTAGCTAA` `ATGGGGTAG`

2.9 Question 9: Tying It All Together

You have successfully written functions to help your research team analyze DNA sequences! However, now you want to make it more accessible and user-friendly. For this question, you will create a main function that allows a user to interact with your program by providing their DNA sequences. Your main function should present the user with a menu containing the following options:

1. Calculate the similarity between two sequences of the same length
2. Calculate the best similarity between two sequences of either equal or unequal length
3. Identify mutations
4. Transcribe DNA to RNA
5. Find the complement of a DNA sequence
6. Extract reading frames
7. Exit

Your menu should run on a loop, continually offering the user each option until they choose to exit. Be sure to use the functions you wrote in questions 1 through 6 as needed.

Note: Your main function should account for any user input that isn't a valid sequence. If user input is not a valid sequence, your program should print `"Invalid input. Please enter a valid sequence."` until the user enters a valid sequence. Additionally, functions that require strings of the same length should have their inputs validated for matching lengths before being called in the user menu. If the strands are not of the same size, the program should display `"Error: Input strands must be of the same length."` and return to the menu.

For Question 9, develop and validate your solution in VS Code. Once you are happy with your solution, go to coderunner on Canvas and paste you entire program to the answer box!

Sample Run 2.9.1

```
--- DNA Analysis Menu ---
1. Calculate the similarity between two sequences of the same length
2. Calculate the best similarity between two sequences of either equal or unequal length
3. Identify mutations
4. Transcribe DNA to RNA
5. Find the reverse complement of a DNA sequence
6. Extract reading frames
7. Exit
Please enter your choice (1 - 7):
8
Invalid input. Please select a valid option.
--- DNA Analysis Menu ---
1. Calculate the similarity between two sequences of the same length
2. Calculate the best similarity between two sequences of either equal or unequal length
3. Identify mutations
4. Transcribe DNA to RNA
5. Find the reverse complement of a DNA sequence
6. Extract reading frames
7. Exit
Please enter your choice (1 - 7):
7
Exiting program.
```

Sample Run 2.9.2

```
--- DNA Analysis Menu ---
1. Calculate the similarity between two sequences of the same length
2. Calculate the best similarity between two sequences of either equal or unequal length
3. Identify mutations
4. Transcribe DNA to RNA
5. Find the reverse complement of a DNA sequence
6. Extract reading frames
7. Exit
Please enter your choice (1 - 7):
1
Enter the first DNA sequence:
ATC
Enter the second DNA sequence:
AG
Error: Input strands must be of the same length.
--- DNA Analysis Menu ---
1. Calculate the similarity between two sequences of the same length
```

```
2. Calculate the best similarity between two sequences of either equal or unequal length
3. Identify mutations
4. Transcribe DNA to RNA
5. Find the reverse complement of a DNA sequence
6. Extract reading frames
7. Exit
Please enter your choice (1 - 7):

Exiting program.
```

Sample Run 2.9.3

```
--- DNA Analysis Menu ---
1. Calculate the similarity between two sequences of the same length
2. Calculate the best similarity between two sequences of either equal or unequal length
3. Identify mutations
4. Transcribe DNA to RNA
5. Find the reverse complement of a DNA sequence
6. Extract reading frames
7. Exit
Please enter your choice (1 - 7):
1
Enter the first DNA sequence:
AT
Enter the second DNA sequence:
AG
Similarity score: 0.5
--- DNA Analysis Menu ---
1. Calculate the similarity between two sequences of the same length
2. Calculate the best similarity between two sequences of either equal or unequal length
3. Identify mutations
4. Transcribe DNA to RNA
5. Find the reverse complement of a DNA sequence
6. Extract reading frames
7. Exit
Please enter your choice (1 - 7):

Exiting program.
```

Sample Run 2.9.4

```
--- DNA Analysis Menu ---
1. Calculate the similarity between two sequences of the same length
2. Calculate the best similarity between two sequences of either equal or unequal length
3. Identify mutations
4. Transcribe DNA to RNA
5. Find the reverse complement of a DNA sequence
6. Extract reading frames
7. Exit
Please enter your choice (1 - 7):
2
Enter the first DNA sequence:
```

```
AGTC
Enter the second DNA sequence:
ATCG
Best similarity score: 0.25
--- DNA Analysis Menu ---
1. Calculate the similarity between two sequences of the same length
2. Calculate the best similarity between two sequences of either equal or unequal length
3. Identify mutations
4. Transcribe DNA to RNA
5. Find the reverse complement of a DNA sequence
6. Extract reading frames
7. Exit
Please enter your choice (1 - 7):
7
Exiting program.
```

Sample Run 2.9.5

```
--- DNA Analysis Menu ---
1. Calculate the similarity between two sequences of the same length
2. Calculate the best similarity between two sequences of either equal or unequal length
3. Identify mutations
4. Transcribe DNA to RNA
5. Find the reverse complement of a DNA sequence
6. Extract reading frames
7. Exit
Please enter your choice (1 - 7):
3
Enter the first DNA sequence:
ATTTG
Enter the second DNA sequence:
ATCTG
Best similarity score: 0.8
Best alignment index: 0
Substitution at position 3: T -> C
--- DNA Analysis Menu ---
1. Calculate the similarity between two sequences of the same length
2. Calculate the best similarity between two sequences of either equal or unequal length
3. Identify mutations
4. Transcribe DNA to RNA
5. Find the reverse complement of a DNA sequence
6. Extract reading frames
7. Exit
Please enter your choice (1 - 7):
7
Exiting program.
```

Sample Run 2.9.6

```
--- DNA Analysis Menu ---
1. Calculate the similarity between two sequences of the same length
2. Calculate the best similarity between two sequences of either equal or unequal length
```

```
3. Identify mutations
4. Transcribe DNA to RNA
5. Find the reverse complement of a DNA sequence
6. Extract reading frames
7. Exit
Please enter your choice (1 - 7):
4
Enter the DNA sequence to be transcribed:
ATTC
The transcribed DNA is: AUUC
--- DNA Analysis Menu ---
1. Calculate the similarity between two sequences of the same length
2. Calculate the best similarity between two sequences of either equal or unequal length
3. Identify mutations
4. Transcribe DNA to RNA
5. Find the reverse complement of a DNA sequence
6. Extract reading frames
7. Exit
Please enter your choice (1 - 7):
7
Exiting program.
```

Sample Run 2.9.7

```
--- DNA Analysis Menu ---
1. Calculate the similarity between two sequences of the same length
2. Calculate the best similarity between two sequences of either equal or unequal length
3. Identify mutations
4. Transcribe DNA to RNA
5. Find the reverse complement of a DNA sequence
6. Extract reading frames
7. Exit
Please enter your choice (1 - 7):
5
Enter the DNA sequence:
ATTCG
The reverse complement is: CGAAT
--- DNA Analysis Menu ---
1. Calculate the similarity between two sequences of the same length
2. Calculate the best similarity between two sequences of either equal or unequal length
3. Identify mutations
4. Transcribe DNA to RNA
5. Find the reverse complement of a DNA sequence
6. Extract reading frames
7. Exit
Please enter your choice (1 - 7):
7
Exiting program.
```

Sample Run 2.9.8

```
--- DNA Analysis Menu ---
```

```
1. Calculate the similarity between two sequences of the same length
2. Calculate the best similarity between two sequences of either equal or unequal length
3. Identify mutations
4. Transcribe DNA to RNA
5. Find the reverse complement of a DNA sequence
6. Extract reading frames
7. Exit
Please enter your choice (1 - 6):
6
Enter the DNA sequence:
ATGCGGTAA
The extracted reading frames are:
ATGCGGTAA
--- DNA Analysis Menu ---
1. Calculate the similarity between two sequences of the same length
2. Calculate the best similarity between two sequences of either equal or unequal length
3. Identify mutations
4. Transcribe DNA to RNA
5. Find the reverse complement of a DNA sequence
6. Extract reading frames
7. Exit
Please enter your choice (1 - 7):
7
Exiting program.
```

Sample Run 2.9.9

```
--- DNA Analysis Menu ---
1. Calculate the similarity between two sequences of the same length
2. Calculate the best similarity between two sequences of either equal or unequal length
3. Identify mutations
4. Transcribe DNA to RNA
5. Find the reverse complement of a DNA sequence
6. Extract reading frames
7. Exit
Please enter your choice (1 - 7):
6
Enter the DNA sequence:
AGCTTTAA
The extracted reading frames are:
No reading frames found.
--- DNA Analysis Menu ---
1. Calculate the similarity between two sequences of the same length
2. Calculate the best similarity between two sequences of either equal or unequal length
3. Identify mutations
4. Transcribe DNA to RNA
5. Find the reverse complement of a DNA sequence
6. Extract reading frames
7. Exit
Please enter your choice (1 - 7):
7
Exiting program.
```

Sample Run 2.9.10

```
--- DNA Analysis Menu ---
1. Calculate the similarity between two sequences of the same length
2. Calculate the best similarity between two sequences of either equal or unequal length
3. Identify mutations
4. Transcribe DNA to RNA
5. Find the reverse complement of a DNA sequence
6. Extract reading frames
7. Exit
Please enter your choice (1 - 7):
1
Enter the first DNA sequence:
ATRBAD
Invalid input. Please enter a valid sequence.
Enter the first DNA sequence:
ATTA
Enter the second DNA sequence:
ATTAx
Invalid input. Please enter a valid sequence.
Enter the second DNA sequence:
ATCG
Similarity score: 0.5
--- DNA Analysis Menu ---
1. Calculate the similarity between two sequences of the same length
2. Calculate the best similarity between two sequences of either equal or unequal length
3. Identify mutations
4. Transcribe DNA to RNA
5. Find the reverse complement of a DNA sequence
6. Extract reading frames
7. Exit
Please enter your choice (1 - 7):
7
Exiting program.
```

Week 7: Arrays

Learning Goals

This week you will:

1. Learn how to make multidimensional arrays
2. Learn how to use arrays in functions
3. Be able to distinguish between pass by reference and pass by value

1 Background

1.1 Passing Arrays By Reference

Up until now, when calling functions, we have always **passed by value**. When a parameter is passed in a function call, a new variable is declared and initialized to the value passed in the function call.

Observe that the variable `x` in `main` and variable `x` in `AddOne` are separate variables in memory. When `AddOne` is called with `x` on line 10, it is the value of (i.e. 5) that is passed to the function. This value is used to initialize a new variable `x` that exists only in `AddOne`'s scope. Thus the value of the variable `x` in main's scope remains unchanged even after the function `AddOne` has been called.

Example 1.1.1. Pass by value:

```
void AddOne(int x){
    x = x + 1;
    cout << x << endl;
}

int main(){
    int x = 5;
    cout << x << endl;
    AddOne(x);
    cout << x << endl;
}
```

Output:

```
5
6
5
```

Passing By Reference Arrays, on the other hand, are passed by reference (a reference to the original array' s location in the computer' s memory). So, when an array is passed as a parameter, the original array is used by the function. Observe that there is only one array `X` in memory for the following example. When

the function `AddOne` is called on line 10, a reference to the original array `X` is passed to `AddOne`. Because the array `X` is passed by reference, any modifications done to `X` in `AddOne` are done to the original array. These modifications persist and are visible even after the flow of control has exited the function and we return to main.

Example 1.1.2. Pass by reference example:

```
void AddOne(int X[]){
   X[0] = X[0] + 1;
   cout << X[0] << endl;
}

int main(){
    int X[4] = {1, 5, 3, 2};
    cout << X[0] << endl;
    AddOne(X);
    cout << X[0] << endl;
}
```

Output:

```
1
2
2
```

When we pass a one-dimensional array as an argument to a function we also provide its length. For two-dimensional arrays, in addition to providing the length (or number of rows), we will also assume that we know the length of each of the subarrays (or the number of columns). A function taking a two-dimensional array with 10 columns as an argument then might look something like this:

```
void TwoDimensionalFunction(int matrix[][10], int rows){...}
```

1.2 Multidimensional Arrays

In C++ we can declare an array of arrays known as a multidimensional array. Multidimensional arrays store data in tabular form.

How to Declare Multidimensional Arrays

```
// data_type array_name[dimension_1][dimension_2]....;
int myInts[7][5];
bool myBooleans[10][15][12];
string myStrings[15][10];
```

How to Initialize Multidimensional arrays (Method 1)

```
int myInts[2][2] = {1, 2, 3, 4};
```

The 2D array in this case will be filled from left to right from top to bottom.

```
int myInts[2][2] = {{1, 2}, {3, 4}};
```

You can also initialize a 2D array by explicitly separating the rows as shown above.

How to Initialize Multidimensional arrays (Method 2) You can also initialize elements using nested loops:

```
int myInts[2][2];
for(int i=0; i < 2; i++)
{
    for(int j=0; j < 2; j++)
    {
        myInts[i][j] = i + j;
    }
}
```

The above code will create the following 2D array: 0, 1, 1, 2.

How to Access Elements in a Multidimensional array You can use `myInts[i][j]` to access the ith row and jth column of a 2D array

Multidimensional arrays can be iterated using nested loops as shown below:

```
int myInts[2][2] = {{0, 1}, {1, 2}};
int res = 0;
for(int i=0; i < 2; i++)
{
    for(int j=0; j < 2; j++)
    {
        res = res + myInts[i][j];
    }
}
cout << "Result is " << res;
```

Output: `Result is 4`

2 PreQuiz

Problem 2.1. True or False: The following is valid C++ and will not return an error.

```
#include <iostream>
using namespace std;
int main()
{
bool intArray[3] = {true,false,false,false,true};
}
```

Problem 2.2. True or False: The following is valid C++ and will not return an error.

```
#include <iostream>
using namespace std;
int main()
{
bool intArray[7] = {true,false,false,false,true};
}
```

Problem 2.3. The program prints the contents of a string array. Fill in the blank accordingly:

```
#include <iostream>
using namespace std;

int main()
{
    // Create a character array of size 6 and initialize it with the characters 'h', 'e', 'l',
    ↪  'l', 'o', '\0'
```

```
    _____ // FILL IN THIS LINE

    cout << "The contents of the array are: ";
    for(int i = 0; i < 5; i++)
    {
        cout << message[i] << " ";
    }
    return 0;
}
```

Problem 2.4. The program below creates an array of strings and then prints the first letter of every string, the second letter of every string, so on and so forth. Fill in the blank accordingly.

```
#include<iostream>
using namespace std;

int main()
{
    string fruits[] = {"Apple", "Banana", "Cherry", "Date", "Fig", "Grape", "Mango"};

    int longest_string = fruits[0].length();
    for (int i = 1; i < 7; i++){
        if (_____________){ //FILL IN THIS LINE
            longest_string = fruits[i].length();
        }
    }

    for (int i = 0; i < longest_string; i++){
        for (int j = 0; j < 7; j++){
            if (i < static_cast<int>(fruits[j].length())){
                cout << _____________ << endl; // FILL IN THIS LINE
            }
        }
    }

}
```

3 Recitation

3.1 Spot The Error

Problem 3.1.a. Given two positive integers `x` and `y`, this programs prints all the integer points (i, j) in the rectangle formed by (0, 0) and (x, y). Identify the error(s) in the code below, and write the correct line(s).

```
    #include <iostream>
    using namespace std;

    int main()
    {
        int x = 3, y = 4;

        for(int i = 0; i >= x; j++)
        {
```

```
        for(int j = 0; j <= y; j++)
        {
            cout << "(" << i << ", " << j << ")  ";
        }
        cout << endl;
    }

}
```

Problem 3.1.b. The program prints the contents of an array and then calculates the sum of all the elements. Identify the error(s) in the code below, and write the correct line(s).

```
#include <iostream>
using namespace std;

int main()
{
    int numbers[5] = {10, 20, 30, 40, 50};

    cout << "The contents of the array are: ";
    for (int i = 0; i <= 5; i++)
    {
        cout << numbers << " ";
    }
    cout << endl;

    for (int i = 0; i <= 5; i++)
    {
        int sum = 0;
        sum += numbers;
    }

    cout << "Sum = " << sum << endl;
    return 0;
}
```

Problem 3.1.c. The program finds and prints all prime factors of a given number `num`. Identify the error(s) in the code below, and write the correct line(s).

```
#include <iostream>
#include <cmath>

using namespace std;

void primeFactors(int num)
{
    int n;
    while (n % 2 == 0)
    {
        cout << 2;
        n = n / 2;
    }

    for (int i = 3; i <= sqrt(n); i++)
    {
        while (n % i == 0)
        {
            cout << i << " ";
            n = n / i;
        }
    }

    if (n > 2)
    {
        cout << n;
    }
    cout<<endl;
}

int main()
{
    int num = 315;
    primeFactors(num);
    return 0;
}
```

Problem 3.1.d. The program prints the product of the length of contents of a string array. Identify the error(s) in the code below, and write the correct line(s).

```
#include <iostream>
#include <string>
using namespace std;

int main()
{
    string languages[6] = {"C++","Python","Java","Matlab","Julia"};
    int product = 0;
    int total = languages.length();

    for(int i = 0; i <= total; i++)
    {
```

```
            product *= languages[i].length;
        }

        cout << "Product of lengths = " << product << endl;
        return 0;
    }
```

3.2 Combinations

Create two integer arrays `set1` and `set2` in the `main()`. The length of `set1` and `set2` should be 5 and 2, respectively. Prompt the user to enter 5 integers that go into `set1`. Do the same with 2 integers for `set2`. Then use the arrays to print all the possible pairs of the elements in `set1` with the elements in `set2`.

Example output (red is user input):

Sample Run 3.2.1

```
Please enter 5 integers for the first set:
1 2 3 4 5
Please enter 2 integers for the second set:
10 20
1-10 1-20
2-10 2-20
3-10 3-20
4-10 4-20
5-10 5-20
```

Problem 3.2.a. Write out the steps you would use to solve this problem by hand as pseudocode.

Problem 3.2.b. Pick possible inputs for your program. Choose as many inputs as you think you need to thoroughly test your program. Follow the steps you wrote for these values to find your result, and verify it.

Problem 3.2.c. Implement your solution in C++ using VS Code. Revise your solution, save, compile and run it again. Are you getting the expected result and output? Keep revising until you do. Make you sure you test for the values used in your sample runs, and for the boundary conditions.

Week 8: File I/O

Learning Goals

This week you will:

1. Learn how to open file streams
2. Learn how to read files line by line
3. Learn how to use get/unget

1 Background

1.1 File Input/Output

So far in class, we've been using the `iostream` standard library. This library has provided us with methods like `cin` and `cout`. `cin` is the method that reads from standard input (i.e. in the terminal via the keyboard) and `cout` is for writing to standard output.

In this background section we'll cover file input, which will allow you to read information from a file. To do this, we'll need to include C++'s fstream library, which is short for "file stream".

Reading Lines From A File

Step 1. Make a stream object.

Create an object (a variable) of file stream type. If you want to open a file for reading only, then the ifstream object should be used (short for "input file stream").

```
// create an input file stream object
ifstream file_input;
```

Step 2. Open a file.

Once you have a file stream object, you need to open the file. To do this, use the ifstream object's `open()` method (function), which takes only one parameter: the file name as a string (surrounded by " " if the file name is given directly).

```
// open myTextFile.txt with the file stream object
file_input.open("myTextFile.txt");
```

Step 3. Checking for open files.

It is always good practice to check if the file has been opened properly and take an appropriate action if not. To check if a file was successfully opened, you may use the `fail()` or `is_open()` methods.

`fail()`: This method will return a boolean value true if the file failed to open and false otherwise.

```
if (file_input.fail()) // true when file fails to open
{
    cout << "Could not open file." << endl;
```

```
    return -1; // return to terminate the program; -1 to indicate that the program didn't
    ↪  function as expected
}
// do things with the file
```

`is_open()`: This method will return a boolean value true if the file has successfully opened and false otherwise.

```
if (file_input.is_open()) // true when file opens sucessfully
{
    // do things with the file
}
else
{
    cout << "Could not open file." << endl;
}
```

Step 4. Read lines from the file.

To read a line from the file, you can use `getline(file_input, line)` which returns true as long as an additional line has been successfully assigned to the variable line. Once no more lines can be read in, getline returns false. So we can set up a while loop where the condition is the call to getline.

`.eof()`: This method will return a boolean value true if all the data in the file was processed and false otherwise.

```
string line = "";
int line_idx = 0;
// read each line from the file
while (!file_input.eof()) // continue looping as long as there is data to be processed in the
↪  file
{
    // get the next line from the file and store in 'line' variable
    getline(file_input, line);

    // print each line read from the file
    cout << line_idx << ": " << line << endl;

    // increment index(count of lines in the file)
    line_idx++;
}
```

Step 5. Closing a file.

When you are finished processing your files, it is recommended to close all the opened files before the program is terminated. You can do this by using the `.close()` function on your file stream object.

```
// closing the file
file_input.close();
```

Step 6. Putting it all together. If we put all the previous steps together, this is the final piece of code we get.

```
// create an input file stream object
ifstream file_input;

// open myTextFile.txt with the file stream object
file_input.open("myTextFile.txt");
```

```cpp
// check if file opened successfully
if (file_input.fail())
{
    cout << "Could not open file." << endl;
    return -1;
}
else
{
    // do things with the file
    string line = "";
    int line_idx = 0;

    // read each line from the file
    while (!file_input.eof())
    {
        // gets line of text from file_input, stores it in line
        getline(file_input, line);

        // print each line read from the file
        cout << line_idx << ": " << line << endl;

        // increment index (count of lines in the file)
        line_idx++;
    }
}

// closing the file
myTextFile.close();
```

2 PreQuiz

Problem 2.1. When declaring 2D arrays in C++, it is possible to make one dimension flexible, but the other dimension must remain fixed. Which specific dimension should be hard coded in this scenario? And why is this required?

Problem 2.2. How can you access a specific element in a 3D array?

Problem 2.3. What is the output of the program below?

```cpp
#include <iostream>
using namespace std;

void mysteryFunction(int arr[], int size)
{
    int temp = arr[size-1];
    int i = size-1;
    while(i > 0)
    {
       arr[i] = arr[i-1];
       i--;
    }
    arr[0] = temp;
}

int main()
```

```
{
    const int N = 7;
    int myArr[N] = {4, 6, 8, 0, 3, 2, 1};

    mysteryFunction(myArr, N);

    // Display the result
    cout << "The result is: " << endl;
    for (int i = 0; i < N; i++)
    {
        cout << myArr[i] << " ";
    }

    return 0;
}
```

Problem 2.4. Consider the following code snippet. What are the values of a, b and c?

```
const int N=4;
int matrix[N][N] = {{1, 2, 3, 4},
                    {5, 6, 7, 8},
                    {9, 10, 11, 12},
                    {13, 14, 15, 16}};
int a = matrix[0][0] + matrix[1][1] + matrix[2][2] + matrix[3][3];
int b = matrix[0][3] * matrix[3][0];
int c = matrix[2][3];
```

Problem 2.5. (a) You have seen function tables designed to convey the information in a function on previous assignments. Now, given the code shown in part b of this question, fill in the following table for the `elementWiseProduct()` function

Function: elementWiseProduct(int[][], int[][], int, int)	
Purpose:	.
Parameters:	
Return value:	.
Error handling/ Boundary conditions:	.
Example:	**Example 2.0.1.** **Sample Run 2.0.1**

(b) Write 4 assert statements to test the function after the function call in the given code.

```
#include <iostream>
#include <cassert>
using namespace std;
```

```
void elementWiseProduct(int a[][3], int b[][3], int row, int col)
{
    for (int i = 0; i < row; i++)
    {
        for (int j = 0; j < col; j++)
        {
            a[i][j] = a[i][j] * b[i][j];
        }
    }
}
int main()
{
    const int N=2;
    const int M=3;

    int matrix1[N][M] = {
        {1,2,3},
        {2,3,1}
    };
    int matrix2[N][M] = {
        {2,3,1},
        {2,3,1}
    };

    // Call the elementWiseProduct function with the two matrices
    elementWiseProduct(matrix1, matrix2, N, M);

    // Display the result matrix
    cout << "The result is: "<< endl;
    for (int i = 0; i < 2; i++)
    {
        for (int j = 0; j < 3; j++)
        {
            cout << matrix1[i][j] << " ";
        }
        cout << '\n';
    }

    // include assert statements here
    assert(_____________________________________);
    assert(_____________________________________);
    assert(_____________________________________);
    assert(_____________________________________);

    return 0;
}
```

3 Recitation

3.1 Spot The Error

Problem 3.1.a. The program below intends to prints average of the scores. Identify the error(s) in the code below, and write the correct line(s).

```
#include <iostream>
#include <string>
using namespace std;
int main()
{
    int size = 6;
    double scores[size] = {85.4, 90.3, 100, 89, 74.5, 95.0, 82.3};
    double sum = 0;
    for(int i = 0; i < size; i++)
    {
        sum += scores[i];
    }
    int avg = sum / 6.0;
    cout << "Average = " << avg << endl;
    return 0;
}
```

Problem 3.1.b. The program below displays transpose of a given matrix. The transpose of a matrix is simply a flipped version of the original matrix by switching its rows with its columns. Identify the error(s) in the code below, and write the correct line(s).

```
#include <iostream>
using namespace std;

// Function to calculate the transpose of a matrix
double transposeMatrix(int matrix[][3], int n, int m)
{
    for (int i = 0; i < n; i++)
    {
        for (int j = i + 1; j < m; j++)
        {
            temp = matrix[i][j];
            matrix[i][j] = matrix[j][i];
            matrix[j][i] = temp;
        }
    }
    return matrix;
}

int main()
{
    const int rows = 3;
    const int cols = 3;
    int originalMatrix[rows][cols] =
    {
        {1, 1, 1},
        {2, 2, 2},
        {3, 3, 3}
```

```
    };

    // Calculate the transpose matrix using the function
    int result[rows][cols] = transposeMatrix(originalMatrix[3][3], rows, cols);

    // Display the transpose matrix
    cout << "Transpose Matrix:" << endl;
    for (int i = 0; i < rows; i++)
    {
        for (int j = 0; j < cols; j++)
        {
            cout << result[i][j] << " ";
        }
        cout << endl;
    }

    return 0;
}
```

Problem 3.1.c. The program below tries to print all the items specified in the item array. Identify the error(s) in the code below, and write the correct line(s).

```
#include <iostream>
using namespace std;

int main()
{
    int N = 3;
    char item[] = {"book", "pen", "pencil", "eraser"};

    //printing all the items
    for (int i = 0; i < N; i++)
    {
        cout << "The item list has " << items[j] << endl;
    }
    return 0;
}
```

Problem 3.1.d. The program below prints the strings that have length equal to 4. Identify the error(s) in the code below, and write the correct line(s).

```
#include <iostream>
#include <string>
using namespace std;

int main()
{
    const int N = 6;
    string animals[N] = {"lion", "cat", "bear", "dog", "elephant", "fox"};
    for (int i = 0; i < N; i++)
    {
        if (animals.length()[i] == 4)
        {
            cout << animals[i] << endl;
        }
```

```
        }
        return 0;
    }
```

3.2 Coding Telephone

For this activity, you will need to break into groups of approximately 6-8. The basic idea of the game is simple: We will give you some starting piece of code, with a short region of the code marked for "Coding Telephone". The first person in your group will have to translate the code entirely to comments, leaving no actual code in that region behind. They will pass this comment file on to the second person in your group, who will translate the comments back into code. Then the next person will translate the code to comments, so on and so forth until everyone in your group has participated. Then you will attempt to compile your code and run it, and see what happens. Think of it like the game Telephone, but with code.

You will get the starting code from Canvas. The only person who should view this original code is the first person in your group. Make sure to keep the code secret at each stage so others cannot see it. You will each have five minutes to translate your code. During the time you are not coding, you are welcome to work on the other questions in this recitation assignment.

Once your group has completed the game, work together to answer the following questions:

Problem 3.2.a. Did your final code compile? If not, what error did it fail with? If this is an easy to fix error, fix it and move on to the next question.

Problem 3.2.b. If your code compiled, did it run? If not, what error did it fail with? If so, what did it print?

Problem 3.2.c. Compile and run the original code, and compare it with the code you ended with. How did it change? Where were some areas of miscommunication?

Submit your answers to these questions along with your final Telephone code for this problem on Canvas.

3.3 Matrix Sum

The sum of two matrices is found by adding together corresponding entries in each matrix, as shown above. (For example, the numbers at (row 1, column 1) of each matrix are added together to get the (row 1, column 1) number of the sum matrix.)

Write a function that accepts two 2x3 integer arrays as input parameters and displays the sum matrix (displayed in 2 rows). Example header:

```
void matrixSum(int a[2][3], int b[2][3])
```

Then, create a main() function that asks the user to input the values for each matrix one row at a time. Store these in two arrays, and pass them to matrixSum. Example output (red is user input)

Sample Run 3.3.1

```
Enter values for matrix 1, one row at a time:
1 2 3
4 5 6
Enter values for matrix 2, one row at a time:
2 4 6
8 10 12
The sum is:
3 6 9
12 15 18
```

Problem 3.3.a. Write out the steps you would use to solve this problem by hand as pseudocode.

Problem 3.3.b. Pick possible inputs for your program. Choose as many inputs as you think you need to thoroughly test your program. Follow the steps you wrote for these values to find your result, and verify it.

Problem 3.3.c. Implement your solution in C++ using VS Code. Revise your solution, save, compile and run it again. Are you getting the expected result and output? Keep revising until you do. Make you sure you test for the values used in your sample runs, and for the boundary conditions.

4 Homework

Warning: You are not allowed to use global variables for this project.

All function names, return types, and parameters must precisely match those shown. You may not use pass by reference or otherwise modify the function prototypes. You are welcome to create additional functions that may help streamline your code.

4.1 Most Popular Word

A word is said to be most popular if it occurs more frequently than any other word. Write a function named `mostPopularWord()` that finds the most popular word in an array and prints the word, its frequency, and the indices where it was found. You may assume there's always one correct answer, and the array will be non-empty. If it has equal occurrence, then we always consider the most recent one.

Note: You do not need sorting to solve this.

Function: mostPopularWord(string[], const int)	`void mostPopularWord(string words[], const int WORDS_SIZE)`
Purpose:	Print the word with the highest frequency, its count, and its index positions in the array.
Parameters:	**string** `words`: Array of words **const int** `WORDS_SIZE`: The size of word array
Return value:	- The function should print the word with the highest frequency, its count, and its index positions in the array. - The function doesn't return any value to the main function.
Error Handling/ Boundary Conditions:	- If two or more words have the same frequency, the word that appears last will be considered the most popular; see sample run 4.1.3. - You may assume the words[] array is always non-empty.

Example:	**Example 4.1.1.**
	`// Assume the proper libraries are included`
	`// Assume the proper implementation of mostPopularWord() is included`
	`int main()`
	`{`
	`  const int WORDS_SIZE = 4;`
	`  string words[WORDS_SIZE] = {"mail", "text", "spam", "spam"};`
	`  mostPopularWord(words, WORDS_SIZE);`
	`  return 0;`
	`}`
	
	Sample Run 4.1.1
	`The most popular word: spam`
	`Frequency: 2`
	`Found at pos: 2 3`

For Question 1, develop and validate your solution in VS Code. Once you are happy with your solution, go to coderunner on Canvas and paste mostPopularWord() and any helper function(s) to the answer box!

Sample Run 4.1.2

Test code:

```
    const int WORDS_SIZE = 5;
    string words[WORDS_SIZE] = {"hello", "world", "hello", "world", "hello"};
```

Output:

```
The most popular word: hello
Frequency: 3
Found at pos: 0 2 4
```

Sample Run 4.1.3

Test code:

```
    const int WORDS_SIZE = 5;
    string words[WORDS_SIZE] = {"apple", "corn", "corn", "apple", "lettuce"};
```

Output:

```
The most popular word: apple
Frequency: 2
Found at pos: 0 3
```

Explanation:

The last word that appeared with a frequency of 2 is "apple" at index 3. Therefore, based on our rules, that is the word that will be displayed.

4.2 Fitness Statistics

Bob recently started a daily fitness routine to improve his overall health. Every day, he tracks the number of steps he takes, and after a few weeks of consistent effort, he now wants to analyze his data to gain some insights into his progress.

To help him, Bob needs two(2) functions written in C++ that will allow him to understand his fitness journey better.

Function 1: Bob is curious about how his total step count has increased over time. This function will calculate the cumulative sum of his daily step count, showing him the total distance he's covered since he started tracking.

Function: stepCountCumulativeSum(int[], const int, int[])	`void stepCountCumulativeSum(int daily_steps[], const int NUM_DAYS, int cumulative_steps[])`
Purpose:	Calculates the cumulative sum of the daily step counts and records the running total in the cumulative_steps array. The cumulative sum represents the total number of steps Bob has taken up to each respective day, providing a running total of his overall progress. The function should not print anything.
Parameters:	**int** `daily_steps[]`: array of Bob's daily step count. **const int** `NUM_DAYS`: Size of daily_steps array and cumulative_steps array **int** `cumulative_steps[]`: Array to be filled with Bob's cumulative step count.
Return value:	The function doesn't return any value.

Example:

Example 4.2.1.

```
// Assume the proper libraries are included
// Assume the proper implementation of
↪  stepCountCumulativeSum() is included

int main()
{
    const int NUM_DAYS = 5;
    int daily_steps[NUM_DAYS] = {5000, 4000, 5000,
    ↪  2000, 4000};
    int cumulative_steps[NUM_DAYS];
    stepCountCumulativeSum(daily_steps, NUM_DAYS,
    ↪  cumulative_steps);
    // We are printing the contents of the
    ↪  cumulative_steps array here.
    return 0;
}
```

Sample Run 4.2.1

```
5000
9000
14000
16000
20000
```

Sample Run 4.2.2

Test code:

```
const int NUM_DAYS = 3;
int daily_steps[NUM_DAYS] = {10000, 2000, 6000};
int cumulative_steps[NUM_DAYS];
```

Output:

```
10000
12000
18000
```

Sample Run 4.2.3

Test code:

```
const int NUM_DAYS = 4;
int daily_steps[NUM_DAYS] = {4000, 2000, 8000, 3000};
int cumulative_steps[NUM_DAYS];
```

Output:

```
4000
6000
```

```
14000
17000
```

Function 2: Bob is also interested in how consistent he has been with his daily steps. This function will calculate the mean of his daily step count and return the deviation, showing him which days he was above or below his regular activity level.

Function: stepCountDeviation(int[], const int, const int)	`double stepCountDeviation(int daily_steps[], const int NUM_DAYS, const int OPTIMAL_STEP_COUNT)`
Purpose:	Calculates the mean of daily step counts and returns the deviation deviation = mean - optimal step count The function should not print anything.
Parameters:	**int** `daily_steps[]`: array of Bob's daily step count. **const int** `NUM_DAYS`: Size of daily_steps array **const int** `OPTIMAL_STEP_COUNT`: Desired step count per day.
Return value:	The function will return a **double**, which is the calculated deviation

Example:	**Example 4.2.2.**
	`// Assume the proper libraries are included`
	`// Assume the proper implementation of stepCountDeviation() is included`
	`int main()`
	`{`
	`  const int NUM_DAYS = 5;`
	`  int daily_steps[NUM_DAYS] = {5000, 4000, 5000, 2000, 4000};`
	`  const int OPTIMAL_STEP_COUNT = 5000;`
	`  double deviation = stepCountDeviation(daily_steps, NUM_DAYS, OPTIMAL_STEP_COUNT);`
	`  // We are printing the deviation up to 3 decimal places`
	`  cout << fixed << setprecision(3) << deviation << endl;`
	`  return 0;`
	`}`
	
	Sample Run 4.2.4
	`-1000.000`
	
	Explanation: mean = (5000 + 4000 + 5000 + 2000 + 4000)/5 = 4000
	deviation = mean - OPTIMAL_STEP_COUNT = 4000 - 5000 = -1000

For Question 2, develop and validate your solution in VS Code. Once you are happy with

your solution, go to coderunner on Canvas and paste stepCountCumulativeSum(), stepCountDeviation(), and any helper function(s) to the answer box!

Sample Run 4.2.5

Test code:

```
const int NUM_DAYS = 3;
int daily_steps[NUM_DAYS] = {10000, 2000, 6000};
const int OPTIMAL_STEP_COUNT = 5000;
```

Output:

```
1000.000
```

Sample Run 4.2.6

Test code:

```
const int NUM_DAYS = 4;
int daily_steps[NUM_DAYS] = {4000, 2000, 8000, 3000};
const int OPTIMAL_STEP_COUNT = 750;
```

Output:

```
3500.000
```

4.3 Normal Heart Rate

As a pediatrician, you need to monitor newborns' heart rates, which should fall between 70 and 190 bpm. To simplify this task, write a function printNormalHeartRates() that takes two arrays: one with patient names and another with their heart rates. The function will print the names of patients with normal heart rates.

Function: printNormalHeartRates (string, int, int)	`void printNormalHeartRates(string patients[], int heart_rate[], const int NUM_PATIENTS)`
Purpose:	Prints out the patients with normal heart rates. Where normal heart rate is between 70 and 190, both inclusive. You may assume that both patients[] and heart_rate[] arrays will be the same size.
Parameters:	**string** `patients[]`: Array of strings which contains the patient's names. **int** `heart_rate[]`: Array of int which contains their heart rate **const int** `NUM_PATIENTS`: An integer that gives the number of elements in the patient and heart rate arrays
Return value:	- The function doesn't return any value. - Your function should print out all patients whose heart rate is within the range
Error handling/ Boundary conditions:	- You may assume that the patients[] array and heart_rate[] array are non-empty - You may also assume that the size of the patients[] array and heart_rate[] array are always equal

Example:

Example 4.3.1.

```
// Assume the proper libraries are included
// Assume the proper implementation of
↪  printNormalHeartRates() is included

int main()
{

    string patients[3] = {"Jamie", "Sebastien",
    ↪  "Shaun"};
    int heart_rate[3] = {90, 70, -12};

    printNormalHeartRates(patients, heart_rate, 3);

    return 0;
}
```

Sample Run 4.3.1

```
Jamie 90
Sebastien 70
```

For Question 3, develop and validate your solution in VS Code. Once you are happy with your solution, go to coderunner on Canvas and paste printNormalHeartRates() and any helper function(s) to the answer box!

Sample Run 4.3.2

Test code:

```
string patients[4] = {"Joe", "Jack", "Amy", "Bob"};
int heart_rate[4] = {70, 80, 190, 100};
printNormalHeartRates(patients, heart_rate, 4);
```

Output:

```
Jack 80
Amy 190
Bob 100
```

Sample Run 4.3.3

Test code:

```
string patients[6] = {"Chloe Kim", "Rene Rinnekangas", "Shaun White", "A", "B", "C"};
int heart_rate[6] = {190, 70, 191, 69, 71, 189};
printNormalHeartRates(patients, heart_rate, 6);
```

Output:

```
Chloe Kim 190
```

```
Rene Rinnekangas 70
B 71
C 189
```

4.4 Best Time to Buy and Sell a House

Flipping is a real estate strategy that involves buying homes and selling them for a profit in a short period of time. You want to maximize your profit by choosing a single day to buy a house and choosing a different day in the future to sell that house.

Function: maxProfit(int[], const int)	`int maxProfit(int prices[], const int NUM_MONTHS)`
Purpose:	Find the maximum profit achieved from buying and selling a house. The function should not print anything.
Parameters:	**int** `prices[]`: Array of prices of the house every month **const int** `NUM_MONTHS`: The size of prices array
Return value:	It should return the maximum profit achieved through one house flip.
Error handling/ Boundary conditions:	- Return 0 if no profit can be achieved. - You may assume that the `prices[]` array is always non-empty

Example:	**Example 4.4.1.**
	`// Assume the proper libraries are included`
	`// Assume the proper implementation of maxProfit() is included`
	`int main()`
	`{`
	`  const int NUM_MONTHS = 6;`
	`  int prices[NUM_MONTHS] = {700000, 100000, 500000, 300000, 600000, 400000};`
	`  int profit = maxProfit(prices, NUM_MONTHS);`
	`  cout << "Maximum profit: " << profit << endl;`
	`  return 0;`
	`}`
	
	Sample Run 4.4.1
	`Maximum Profit: 500000`
	
	Explanation:
	Buy on month 2 (price = 100000) and sell on month 5 (price = 600000), profit = 600000-100000 = 500000.
	Note: Buying on month 2 and selling on month 1 is not allowed because you must buy before you sell.

For Question 4, develop and validate your solution in VS Code. Once you are happy with your solution, go to coderunner on Canvas and paste maxProfit() and any helper function(s) to the answer box!

Sample Run 4.4.2
Test code:

```
const int NUM_MONTHS = 5;
int prices[NUM_MONTHS] = {700000, 600000, 400000, 300000, 100000};
```

Output:

```
Maximum Profit: 0
```

Explanation:
In this case, no profitable transactions are possible, and the max profit = 0.

Sample Run 4.4.3
Test code:

```
const int NUM_MONTHS = 6;
int prices[NUM_MONTHS] = {700000, 300000, 400000, 100000, 400000, 100000};
```

Output:

```
Maximum Profit: 300000
```

Explanation:
Buy on month 4 (price = 100000) and sell on month 5 (price = 400000), profit =
400000-100000 = 300000.

4.5 Splitting a String

When you're processing data, it's useful to break up a text string into pieces using a delimiter. Write a function split() that takes a string, splits it at every occurrence of a delimiter, and then populates an array of strings with the split pieces, up to the provided maximum number of pieces.

Function: split(string, char, string[], const int)	`int split(string input_string, char separator, string arr[], const int ARR_SIZE)`
Purpose:	Break a text string into pieces using the given delimiter and populate each piece in the array. The function does not print anything.
Parameters:	**string** `input_string`: The text string containing data separated by a delimiter **char** `separator`: The delimiter marking the location where the string should be split up **string** `arr[]`: The array that will be used to store the input text string's individual string pieces **const int** `ARR_SIZE`: The number of elements that can be stored in the array
Return value:	- The function returns an `int`, the number of pieces the input text string is split into. - Populate `arr[]` with the all the element.
Error Handling/ Boundary Conditions:	- Return 1 if the delimiter character is not found. Place the entire string in the array as the first element - Return -1 if the string is split into more pieces than the size of the array. Fill the array with as many pieces of the split string as possible. - Return 0 if an empty string is provided.

Example:

Example 4.5.1.

```
// Assume the proper libraries are included
// Assume the proper implementation of split() is
↪  included

void printArray(string arr[], const int NUM_ELEMENTS)
{
    for (int i=0; i < NUM_ELEMENTS; i++)
    {
        cout << "arr["<< i << "]:" << arr[i] << endl;
    }
}

int main()
{
    string input_string = "ABCDEFG";
    char separator = ' ';
    const int ARR_SIZE = 3;
    string arr[ARR_SIZE];
    // num_splits is the value returned by split
    int num_splits = split(input_string, separator,
    ↪  arr, ARR_SIZE);
    cout << "Function returned value: " << num_splits
    ↪  << endl;
    if(num_splits != -1)
    {
        printArray(arr, num_splits);
    }
    else
    {
        printArray(arr, ARR_SIZE);
    }
}
```

Sample Run:

Sample Run 4.5.1

```
Function returned value: 1
arr[0]:ABCDEFG
```

For Question 5, develop and validate your solution in VS Code. Once you are happy with your solution, go to coderunner on Canvas and paste split() and any helper function(s) to the answer box!

Sample Run 4.5.2

Test code:

```
string testcase = "RST,UVW,XYZ";
```

```
char separator = ',';
const int ARR_SIZE = 3;
string arr[ARR_SIZE];
// num_splits is the value returned by split
int num_splits = split(testcase, separator, arr, ARR_SIZE);
cout << "Function returned value: " << num_splits << endl;
for (int i=0; i < ARR_SIZE; i++){
  cout << "arr["<< i << "]:" << arr[i] << endl;
}
```

Output:

```
Function returned value: 3
arr[0]:RST
arr[1]:UVW
arr[2]:XYZ
```

Sample Run 4.5.3

Test code:

```
string testcase = "Bangkok,Berlin,Birmingham,Bogota,Busan,Baton
↪  Rouge,Beaumont,Boise,Budapest";
char separator = ',';
const int ARR_SIZE = 5;
string arr[ARR_SIZE];
// num_splits is the value returned by split
int num_splits = split(testcase, separator, arr, ARR_SIZE);
cout << "Function returned value: " << num_splits << endl;
for (int i=0; i < ARR_SIZE; i++){
  cout << "arr["<< i << "]:" << arr[i] << endl;
}
```

Output:

```
Function returned value: -1
arr[0]:Bangkok
arr[1]:Berlin
arr[2]:Birmingham
arr[3]:Bogota
arr[4]:Busan
```

Explanation:

```
The string can be split into more pieces than the size of the array; thus, we returned -1
and filled the array with as many pieces as possible.
```

4.6 School Exam Scores

You' re helping your teacher analyze student exam scores across 10 subjects. Each row represents a student' s scores in these subjects. Your task is to identify the first student whose average score falls below a defined threshold. Write a function minAverg() that returns the index of the first student with an average score below the threshold.

Function: minAverg(int, int, double)	`int minAverg(int scores[][10], const int NUM_STUDENTS, double threshold)`
Purpose:	A function to return the index of a student whose average score is less than or equal to the threshold.
Parameters:	**int** `scores[][10]`: A 2D integer array with 10 columns for storing the scores for each of the 10 subjects **const int** `NUM_STUDENTS`: An integer number of rows for each student **double** `threshold`: A floating number threshold for low average score.
Return value:	The function returns an **int** representing the index of the first student with a low average score that is less than or equal to the threshold.
Error handling/ Boundary conditions:	- The function returns -1 if there is no average score that is less than or equal to the threshold - You may assume that the score[] array is non-empty

Example:	**Example 4.6.1.**
	`// Assume the proper libraries are included`
	`// Assume the proper implementation of minAverg() is included`
	`int main()`
	`{`
	`  int scores[2][10] = {`
	`  {80, 90, 85, 95, 88, 82, 75, 89, 91, 87},`
	`  {1, 2, 3, 4, 5, 6, 7, 8, 9, 10}`
	`  };`
	`  cout << "returned student index: " << minAverg(scores, 2, 10.0);`
	`  return 0;`
	`}`
	
	Sample Run 4.6.1
	`returned student index: 1`

For Question 6, develop and validate your solution in VS Code. Once you are happy with your solution, go to coderunner on Canvas and paste minAverg() and any helper function(s) to the answer box!

Sample Run 4.6.2

Test code:

```
int scores[3][10] = {
      {10, 1, 1, 1, 1, 1, 1, 1, 1, 1},
      {2, 2, 2, 2, 2, 2, 2, 2, 2, 20},
      {3, 3, 3, 3, 3, 30, 3, 3, 3, 3}
};
```

```
cout << "returned student index: " <<  minAverg(scores, 2, 25);
```

Output:

```
returned student index: 0
```

Sample Run 4.6.3

Test code:

```
    int scores[4][10] = {
    {50, 60, 55, 65, 70, 60, 58, 62, 55, 65},
    {70, 80, 75, 85, 90, 65, 78, 82, 88, 74},
    {10, 20, 15, 25, 12, 30, 18, 22, 17, 15},
    {100, 110, 105, 115, 120, 90, 108, 102, 114, 104},
};
cout << "returned student index: " <<  minAverg(scores, 5, 40);
```

Output:

```
returned student index: 2
```

Sample Run 4.6.4

Test code:

```
int scores[5][10] = {
    { 1, 1, 1, 1, 1, 1, 1, 1, 1, 1 },
    { 1, 1, 1, 1, 1, 1, 1, 1, 1, 1 },
    { 1, 1, 10, 1, 1, 1, 1, 1, 1, 1 },
    { 1, 1, 1, 1, 1, 1, 1, 1, 1, 1 },
    { 1, 1, 1, 1, 1, 1, 1, 1, 1, 1 }
};
cout << "returned student index: " << minAverg(scores, 5, 9);
```

Output:

```
returned student index: 0
```

Week 9: File I/O

Learning Goals

This week you will:

1. Learn how to write to a file

1 Background

1.1 File Output

We discussed reading from a file last week. Here, we will show how we can write to a file. First, you define an `ofstream` variable and open the file you would like to write to. Then you can write to the output file using the same operation you used with `cout`, `<<`, and `endl`.

Example 1.1.1.

```
// Create the output file object
ofstream file_out;
// Opening the output file
file_out.open("output.txt");

// Writing something to the output file
file_out << "writing a sentence to the output file" << endl;

// Writing the value of the variable to the file
string test_string = "you can also write the content of the variable";
file_out << test_string << endl;
file_out.close();
```

The output.txt will have the following content:

Sample Run 1.1.1

```
writing a sentence to the output file
you can also write the content of the variable
```

2 PreQuiz

Problem 2.1. What are file streams? How many major steps are involved in working with a file in C++?

Problem 2.2. The program needs to print all the contents of the file `log.txt` present in the current folder.

```
#include<iostream>
#include<fstream>
```

```
using namespace std;

int main()
{
    // make an input file stream from the file name given above.
    ifstream file_in("log.txt");
    string text;
    while(_____________) // FILL IN THIS LINE
    {
        cout << text << endl;
    }
    //close the file
    _____________; // FILL IN THIS LINE
    return 0;
}
```

Problem 2.3. The program needs to write all the contents of the file input.txt to the file output.txt. Assume that both these files are present in the current folder.

```
#include<iostream>
#include<fstream>
using namespace std;

int main()
{
    // make an input file stream from the file input.txt.
    _____________; // FILL IN THIS LINE
    // make an output file stream to the file output.txt.
    _____________; // FILL IN THIS LINE
    string text;

    // read the text from the input file stream.
    while(getline(file_in, text))
    {
        file_out << text << endl;
    }
    return 0;
}
```

Problem 2.4. The program below takes in cities and their coordinates from a file and prints them one at a time to the screen. The cities and coordinates are stored in a file called `cities.txt`, which looks something like:

```
London
51.5072 N 0.1276 W
New Delhi
28.6139 N 77.2090 E
Cairo
30.0444 N 31.2357 E
Sydney
33.8688 S 151.2093 E
```

The program loops through every line in the file. If the line starts with a letter we know it must be a city name. If it doesn't, it must be a coordinate line. Fill in the blanks appropriately.

```cpp
#include<iostream>
#include<fstream>
using namespace std;

int main(){
    ifstream file_in;
    file_in.open("cities.txt");
    double latitude = 0, longitude = 0;
    char north_or_south, east_or_west;
    char peek;
    string cityName;

    while (!file_in.fail()){
        // Peek at the next character
        file_in.get(peek);
        // Put the character back into the stream
        _________________; //FILL IN THIS LINE
        if (_____________________){ //FILL IN THIS LINE
            getline(file_in, cityName);
        }
        else{
            file_in >> __________________ >> north_or_south >> longitude >> _________________; //FILL
            ↪  IN THIS LINE
            file_in.get(peek); //this line clears the new line character at the end of a
            ↪  coordinate line
            cout << cityName << " is located at " << latitude  << north_or_south;
            cout << ", " << longitude << east_or_west << endl;
        }
    }
}
```

3 Recitation

3.1 Spot The Error

Problem 3.1.a. The program is supposed to flip the sign of all the numbers in an array. Identify the error(s) in the code below, and write the correct line(s).

```cpp
#include <iostream>
#include <string>
using namespace std;

void flipSign(int numbers[])
{
    for (int i = 0; i < 14; i++)
    {
        numbers[i] = -1.0 * numbers[i];
    }
    return;
}

int main()
{
```

```
    const int length = 4;
    int numbers[] = {1, 2, 3, 4};

    cout << "The contents of the array before changing: ";
    for (int i = 0; i < length; i++)
    {
        cout << numbers[i] << " ";
    }
    cout << endl;

    flipSign(numbers);

    cout << "The contents of the array after changing: ";
    for (int i = 0; i < length; i++)
    {
        cout << numbers[i] << " ";
    }

    return 0;
}
```

Problem 3.1.b. The program below reads a file visitors.txt and prints out the busiest day. Each line in the file has the format:

`dayOfWeek <space> visitor1,visitor2,..,visitorN.`

Identify the error(s) in the code below and write the correct line(s).

```
#include <iostream>
#include <string>
using namespace std;

int main()
{
    ifstream my_file("visitors.txt");
    string full_line;
    string days[] = {"Monday", "Tuesday", "Wednesday", "Thursday", "Friday"};
    int vis[5] = {0, 0, 0, 0, 0};
    int i = 0;
    int traffic = 0;
    int j = 0;

    while (getline(my_file, full_line))
    {
        for(int i = 0; i < static_cast<int>(full_line.length()); i+=1)
        {
            if(full_line[i] == ' ' && i < static_cast<int>(full_line.length())-1)
            {
                visitors[i]++;
            }
            if(full_line[i] == ",")
            {
                visitors[i]++;
            }
        }
        if (visitors[i] < traffic)
        {
            traffic = visitors[i];
            j = i;
        }
        i++;
    }
    cout << days[j] << " is the busiest day of the week at the motel." << endl;

    return 0;
}
```

Problem 3.1.c. The program appends and prepends underscores for every word in the given message string. Assume the message is maximum 4 words. Identify the error(s) in the code below, and write the correct line(s). Note, `split()` is a function from Homework 5.

```
#include <iostream>
#include <string>
#include <cassert>
using namespace std;

string appendPrepend(string message)
{
    const int LENGTH = 4;
    string message_fragments[LENGTH] = {};
    string empty_word = "";
    split(message, ' ', message_fragments[], LENGTH)
    assert(message_fragments[4] == empty_word)

    string answer, word;
    for(int i == 0; i < LENGTH; i++)
    {
        answer += "_" + message + "_";
    }
    int first_word_length = message_fragments[0].length();
    int second_word_length = message_fragments[1].length();
    assert(message_fragments[1] = answer.substr(first_word_length+3, second_word_length))

    return answer;
}

int main()
{
    cout << "Please enter the string message:" << endl;
    string message;
    getline(cin, message);
    cout << appendPrepend(message);
}
```

3.2 Center of Mass

The file `coordinates1.txt` contains a list of comma-separated X, Y and Z coordinates for a given geometric body in each column respectively. Your goal is to find the center of mass of the body by computing the average of the X, the Y and the Z coordinates. In order to do this, you must:

1. Read each line in as a string,
2. Use your `split()` function from Homework 5 to separate each line at the commas,
3. Use your `validateDouble()` function from Q2 to confirm that the pieces of each line translate into doubles, and then you can use `stod` to translate those valid strings to doubles.

Report the average of your X coordinates, Y coordinates and Z coordinates as your center of mass. Example output (red is user input):

Sample Run 3.2.1

```
// If file contains two lines:
// 3.25,4.19,-3.56
// 1.04,2.31,5.12
The center of mass is at: 2.145, 3.25, 0.78!
// 2.145 = (3.25+1.04)/2.0
// 3.25 = (4.19+2.31)/2.0
// 0.78 = (-3.56+5.12)/2.0
```

Sample Run 3.2.2

```
// If file contains these lines:
// 5.00,0,-0.8
// -3,3.3,-0.75
// 1,-1.0,3.8
// 3.50,0.67,-2
The center of mass is at: 1, 1.65, -0.025!
```

Sample Run 3.2.3

```
// If file contains these lines:
// 0,5.00,-0.8
// -3,3.3,0.75
// abc,-1.0,2.8
Invalid value detected!
```

There are several coordinate files included in the week 9 module which you can use to test your code. You can also make your own input files.

Problem 3.2.a. Write out the steps you would use to solve this problem by hand as pseudocode.

Problem 3.2.b. Write three possible lines you can include in your file to later test your program. Try to pick values that will test different aspects of your program. Follow the steps you wrote for these values to find your result, and verify it.

Problem 3.2.c. Implement your solution in C++ using VS Code. Revise your solution, save, compile and run it again. Are you getting the expected result and output? Keep revising until you do. Make sure you test for the values used in your sample runs, and for the boundary conditions. Use the coordinates files on github to test your code.

Week 10: Structs and Objects

Learning Goals

This week you will:

1. Learn how to create structs
2. Learn how to access data in structs
3. Learn how to write functions that use structs
4. Learn the basics of object oriented programming
5. Learn about data members, member functions, and constructors.

1 Background

1.1 Structs

In C++, we can define a structure using the keyword struct like so:

```
struct State
{
    string name;
    int area;
};                  // <-- semicolon after struct definition
```

This defines a new type, State, that you can use for declaring variables, e.g.

```
//create a State variable with no name or area
State emptyState;

//create a State variable with a name and area
State colorado{"Colorado", 104094};
```

The variables `emptyState` and `colorado` both have two named attributes, called members - `name` and `area`. We can access each member using dot notation, e.g.

```
//set members for empty State
emptyState.name = "Texas";
emptyState.area = 268596;

//get members for colorado
cout << colorado.name << " has an area of " << colorado.area << " square miles." << endl;
```

Expected output:

```
Colorado has an area of 104094 square miles.
```

If we want to compare two structs, we cannot do so directly. Instead, we must compare each data member individually to see if they match, e.g.

```
//check each data member by one
if(colorado.name == emptyState.name && colorado.area == emptyState.area)
{
    cout << "These are the same state!" << endl;
}
else
{
    cout << "These are not the same state!" << endl;
}
```

Expected output:

```
These are not the same state!
```

1.2 Classes

When writing complex programs, sometimes the built-in data types (such as int, char, string) don' t offer developers enough functionality or flexibility to solve their problems. A solution is for the developer (you - yes, you!) to create your own custom data types to use, called classes. Classes are user-defined data types, which hold their own data members and member functions, which can be accessed and used by creating an instance of that class. A class is like a blueprint for an object, customized for whatever particular problem the programmer is working on.

String, for example, is a class in C++ which holds data (the characters comprising the string) and supports useful member functions like substr which operate on this data.

Below is an example of the basic definition of a class in C++.

```
class ClassName{
    public:
        //The public interface
        //Member functions

    private:
        //The private data members
}; //must end with a semicolon
```

1.3 Anatomy of a Class

Class Name

A class is defined in C++ using the keyword **class** followed by the name of the class. The body of the class is defined inside the curly brackets and terminated by a semicolon at the end. This class name will be how you reference any variables or objects you create of that type. For example:

`ClassName objectName;`

The above line would create a new object (variable) with name `objectName` and of type `ClassName`, and this object would have all the properties that you have defined within the class `ClassName`.

Access Specifiers

Access specifiers in a class are used to set the accessibility of the class members. That is, they restrict access by outside functions to the class members.

Consider the following analogy:

Imagine an intelligence agency like the CIA, that has 10 senior members. Such an organization holds various sorts of information, and needs some way of determining who has access to any given piece of

information. Some information concerning classified or dangerous operations may only be accessible to the 10 senior members of the agency, and not directly accessible by any other person. This data would be **private**.

In a class, like in our intelligence agency, these private data members are only accessible by a class's member functions and not directly accessible by any object or function outside the class.

Some other information may be available to anyone who wants to know about it. This is **public** data. Even people outside the CIA can know about it, and the agency might release this information through press releases or other means. In terms of our class, this public data can be accessed by any member function of the class, as well as outside functions and objects, even other classes. The public members of a class can be accessed from anywhere in the program using the direct member access operator (.) with the object of that class.

Data Members and Member Functions

Data members are the data variables and member functions are the functions used to manipulate these variables; together, data members and member functions define the properties and behavior of the objects in a Class.

The data members declared in any class definition can be fundamental data types (like `int`, `char`, `float`, etc.), `arrays`, or even other classes.

For example, for string objects, the data member is a `char array[]` that holds all of the individual letters of your string. Some of a string' s member functions that we have used already are functions like `.substr()` and `.length()`.

Accessing Data Members

Keeping with our intelligence agency example, the code below defines a class that holds information for any general agency. In the main function, we then create a new `IntelligenceAgency` object called `CIA`, and we set its `organizationName` to "CIA" by using the access operator (.) and the corresponding data member' s name. However, we cannot access the `classifiedIntelligence` data member in the same way. Not everyone has access to that private data.

Instead, we need to use a member function of the `IntelligenceAgency` class, `getClassifiedIntelligence()`, in order to see that information. This allows us to control the release of private information by our `IntelligenceAgency`.

Additionally, the main function includes four different ways of creating objects with descriptions in the comments next to it.

```
class IntelligenceAgency
{
    public:
        IntelligenceAgency();         // Default constructor
        IntelligenceAgency(string classified_intelligence_input); // Parameterized constructor
        string organizationName;
        string getClassifiedIntelligence();
        void setClassifiedIntelligence(string classifiedIntelligenceInput);

    private:
        string classifiedIntelligence;
};

int main()
{
    IntelligenceAgency CIA;
    CIA.organization_name = "CIA";
    cout << CIA.classifiedIntelligence; // gives an error
    cout << CIA.getClassifiedIntelligence();
```

```
    // four types of constructor calls
    IntelligenceAgency abc; // creating an IntelligenceAgency object with the default constructor
    IntelligenceAgency xyz = IntelligenceAgency(); // creating an IntelligenceAgency
    //object with the default constructor
    IntelligenceAgency pqr("PQR"); // creating an IntelligenceAgency
    //object with a paramaterized constructor
    IntelligenceAgency rst = IntelligenceAgency("RST"); // creating an IntelligenceAgency
    //object with a paramaterized constructor
}
```

1.4 Defining Member Functions

You may have noticed that we declared various member functions in our class definition, but we did not specify how they will work when called. The body of the function still needs to be written. The solution is to define a function, such as `getClassifiedIntelligence()`, corresponding to our class' s functions. But how does our program know that these functions are the same as the ones we declared in our class? You as the developer need to explicitly tell the computer that these functions you are defining are the same ones you declared earlier.

```
string IntelligenceAgency::getClassifiedIntelligence()
{
    return classifiedIntelligence;
}
void IntelligenceAgency::setClassifiedIntelligence(string classifiedIntelligenceInput)
{
    classifiedIntelligence = classifiedIntelligenceInput;
}
```

We start the definition as we do any function, with the return type. Then, we have to add a specifier `IntelligenceAgency::` that lets our program know that this function and the one we declared inside the class are the same. We can see that this function returns the class' `classifiedIntelligence` to the user.

Functions like `getClassifiedIntelligence()` are called accessor/getter functions. This is because they retrieve or 'get' the private data members of a class object and return it to the program so that these values may be used elsewhere.

The second function, `setClassifiedIntelligence(string classifiedIntelligenceInput)`, is called a mutator/setter function. These allow functions from other parts of our program to modify or 'set' the private data members of our class objects. Getters and setters are the functions that our program uses to interact with the private data members of our class objects.

1.5 Structs and Classes

A class can have a struct as a data member, much like how a class could have any other type of data member. It's important to make sure that the class header file (the .h file) can see the definition of the struct. This can be accomplished by defining the struct inside of the .h file, like below:

```
// filename: example.h
struct State
{
    string name;
    int area;
};              // don't forget the semicolon

class Example
{
```

```
    private:
        State my_state_;
        //other data members

    public:
        //setter accepts a State parameter
        void setState(State new_state);

        //getter returns the State member variable
        State getState();

        //other member methods
};
```

Any file that includes the .h file shown above will be able to use instances of the State struct, so you will not need to define the State struct anywhere else.

1.6 When to use Structs vs Classes

While structs and classes are similar, there are some key differences between them. When deciding whether you should use one over the other, consider the functionality you'd like to achieve. A good rule of thumb is to ask whether the data type you're defining will need to have any methods or private variables. If it will, then you should create a class. If all members can be public and there are no methods, then a struct may be better.

2 PreQuiz

Problem 2.1. True or False: When writing to a file, you must always manually close the file using `file_out.close()` to ensure the data is saved correctly.

Problem 2.2. True or False: The `.eof()` method for `ifstream` returns a boolean value.

Problem 2.3. What is the difference between `ifstream` and `ofstream`?

Problem 2.4. Fill in the blanks of the following code. The goal of this program is to read the contents of a file input.txt and write them to another file output.txt. Your task is to fill in the blanks to complete the program.

```
#include <iostream>
#include <fstream>
#include <string>

using namespace std;

int main() {
    // Declare input file stream
    ifstream inputFile;

    // Declare output file stream
    ofstream outputFile;

    // Open the input file (input.txt)
    inputFile._____("input.txt"); //FILL IN THIS LINE

    // Check if the input file opened successfully
```

```
    if (!inputFile.is_open()) {
        cout << "Error opening input file!" << endl;
        return 1;
    }

    // Open the output file (output.txt)
    outputFile._____("output.txt"); //FILL IN THIS LINE

    // Declare a string to hold each line of the file
    string line;

    // Read the input file line by line
    while (getline(_____, line)) {
        // Write each line to the output file
        outputFile << _____ << endl;
    }

    // Close both files
    inputFile._____();
    outputFile._____();

    cout << "File copied successfully!" << endl;
    return 0;
}
```

Problem 2.5. The following is a C++ program that takes user input and writes it to a log file called log.txt. The user will enter multiple lines of text, and the program should save each line to the file. Fill in the blanks of the following code.

```
#include <iostream>
#include <fstream>
#include <string>
using namespace std;

int main() {
    // Declare an ofstream object for file output
    ofstream logFile;

    // Open the file "log.txt" for writing
    logFile._____("log.txt");

    // Check if the file opened successfully
    if (!logFile.is_open()) {
        cout << "Error opening file!" << endl;
        return 1;
    }

    string userInput;
    cout << "Enter text to log (type 'exit' to stop):" << endl;

    // Keep reading user input until "exit" is entered
    while (true) {
        getline(cin, userInput);
```

```
        if (userInput == "exit") {
            break;
        }

        // Write the user input to the file
        logFile << __________ << endl;
    }

    // Close the log file
    logFile.__________();

    cout << "Log saved to log.txt." << endl;
    return 0;
}
```

3 Recitation

3.1 Spot the Error

Problem 3.1.a. The following code writes a string to the file output.txt. Identify and fix the error.

```cpp
#include <iostream>
#include <fstream>
using namespace std;

int main() {
    ofstream file_out;
    file_out.open("output.txt");

    string message = "Hello, World!";
    file_out << message << endl;

    return 0;
}
```

Problem 3.1.b. The following code intends to write a sentence and a variable's content to a file. Identify and fix the error.

```cpp
#include <iostream>
#include <fstream>
using namespace std;

int main() {
    ofstream file_out;
    file_out.open("output.txt");

    string test_string = "C++ file handling is fun!";

    file_out("Writing this sentence to the file") << endl;
    file_out(test_string) << endl;

    file_out.close();
    return 0;
}
```

Problem 3.1.c. The following code should open the file log.txt and write some data into it. Identify and fix the error.

```cpp
#include <iostream>
#include <fstream>
using namespace std;

int main() {
    ofstream file_out.open("log.txt");

    string log_message = "This is a log entry.";
    file_out << log_message << endl;

    file_out.close();
    return 0;
}
```

3.2 Center of Mass with File Output

For this assignment you should create a coordinates2.txt file with the following contents:

```
3.25,4.19,-3.56
1.04,2.31,5.12
abc,2.31,5.12
5.00,0,-0.8
2.35,7.45,1.22
```

The file `coordinates2.txt` now contains a list of comma-separated X, Y, and Z coordinates for a given geometric body. Your goal is to find the center of mass of the body by computing the average of the X, the Y, and the Z coordinates. Additionally, you are required to log the results and handle invalid data entries using file I/O in C++.

In order to do this, you must:

1. Read each line in as a string from the file `coordinates2.txt`.
2. Use your `split()` function to separate each line at the commas.
3. Use your `validateDouble()` function to confirm that the pieces of each line translate into valid doubles.
4. Accumulate the valid X, Y, and Z values to compute the center of mass.
5. If any line is invalid (e.g., a non-numeric or incomplete coordinate set), log that line into the file `error_log.txt` and continue with the next line.
6. After processing all lines, calculate and display the center of mass. Additionally, write the center of mass, the number of valid entries, and a completion message to the file `summary.txt`.

Expected contents of each file:

1. summary.txt:
   - Write the computed center of mass (average X, Y, and Z).
   - Write the number of valid coordinate entries processed.
   - Write a completion message indicating the process was successful.
2. error_log.txt:
   - Log each invalid line (if any) with the message `"Invalid entry: "` followed by the contents of the line.

For example, if the file `coordinates2.txt` contains:

```
3.25,4.19,-3.56
1.04,2.31,5.12
abc,2.31,5.12
5.00,0,-0.8
```

The contents of `summary.txt` will be:

Sample Run 3.2.1

```
Center of Mass:
X: 3.095
Y: 2.25
Z: 0.92

Number of valid entries: 3
```

```
Process completed successfully!
```

The contents of `error_log.txt` will be:

Sample Run 3.2.2

```
Invalid entry: abc,2.31,5.12
```

If invalid data is found in any line, the program should skip processing that line and log it in `error_log.txt`. The program should still calculate the center of mass from all valid lines. If no valid coordinates are found, write the message `"No valid coordinates processed!"` to `summary.txt` and skip writing the center of mass.

Problem 3.2.a. Write out the steps you would use to solve this problem by hand as pseudocode.

Problem 3.2.b. Write three possible lines you can include in your file to later test your program. Try to pick values that will test different aspects of your program. Follow the steps you wrote for these values to find your result, and verify it.

Problem 3.2.c. Implement your solution in C++ using VS Code. Save, compile, and run it. Test the program with different input files, including boundary conditions.

4 Homework 7

Warning: You are not allowed to use global variables for this assignment.

All function names, return types, and parameters must precisely match those shown. You may not use pass by reference or otherwise modify the function prototypes. You are welcome to create additional functions that may help streamline your code.

All files used in this assignment are in a zipped folder titled `'homework_7_input_files.zip'` located in Canvas under the week 10 module.

4.1 National Park

This question may require the use of file streams, loops, and arrays.

Write a C++ program to read a file of national park names and print them to the console in the same order as they appear in the file.

Note: There are 63 national parks in the USA, which is why our `MAX_SIZE` is set to 63.

For this question, the answer box on the coderunner is preloaded with the following solution template. You are only required to fill in the appropriate blanks. Additionally, you may use the `national_parks_1.txt` file to test out your code. **All the files used in this assignment, including `national_parks_1.txt`, are available in Canvas under week 10 module**. You may also create more files for further testing.

Example 4.1.1.

```
#include <iostream>
#include <fstream>

using namespace std;

void listNationalParks(string filename)
{
    // initialize variables
    ifstream file_in(filename); // opens a file

    if (file_in.fail())
    {
        cout << "Failed to open file." << endl;
        return;
    }

    const int MAX_SIZE = 63;
    string parks[MAX_SIZE];

    string park;

    int idx = 0;

    // Fill in the blank below with code to read from the file into the 'park' variable
    _____________________________
    {
        parks[idx] = park;
        idx += 1;
```

```
        // Fill in the blank below with code to check if you are trying to store more values
        // than possible in the parks array
        __________________________
        {
            break;
        }

    }

    for (int i = 0; i < idx; ++i)
    {
        cout << i << ": " << parks[i] << endl;
    }

    file_in.close();
}

int main()
{
    string filename;
    cout << "Enter national parks filename" << endl;
    cin >> filename;
    listNationalParks(filename);
    return 0;
}
```

For Question 1, develop and validate your solution in VS Code. Once you are happy with your solution, go to coderunner on Canvas and replace the blank with your solution. The answer box is preloaded with the template above.

4.2 Museum Attendance

This question may require the use of file streams, loops, and arrays.

Your former high school took students on a museum trip. Before boarding the bus, the teachers made a list of all the students' names. After visiting the museum, the teachers asked the students to board the bus and made a new list of all the students' names. Before leaving, the teachers wanted to compare the two lists to make sure no student was left behind. Write a C++ program to help them do this quickly and easily. Your program compares the two attendance sheets and reports if any student is still inside the museum.

Note: The largest file has 24 names; therefore, declare your array with size 30 to ensure it accommodates all test cases.

Function: `compareAttendanceSheet(string, string)`	`void compareAttendanceSheet(string first_attendance_file, string second_attendance_file)`
Purpose:	Given two attendance sheets, the function will compare and print out any individuals who are present in the first sheet but missing in the second.
Parameters:	**string** `first_attendance_file` - The first attendance sheet. **string** `second_attendance_file` - The second attendance sheet.
Return Value:	N/A.
Error Handling:	- If either file does not exist, print "Failed to open attendance files". - If a name exists in the second_attendance_file, it will also be present in the first_attendance_file. In other words, no new names are added to the second_attendance_file that were not in the first. - Hint: The largest file has 24 names; therefore, declare your array with size 30 to ensure it accommodates all test cases.

Example:	**Example 4.2.1.**
	`// This is only an example usage, and you should develop your own main function`
	`// Assume the proper libraries are included.`
	`// Assume the proper implementation of compareAttendanceSheet() is included.`
	`int main() { compareAttendanceSheet("example_1.txt", "example_2.txt"); return 0; }`

Sample Input:	Sample from example_1.txt: Jorden Owen Waylon Mccoy Dario Harding Jameson Odonnell Andrew Wilkinson Sample from example_2.txt: Jorden Owen Waylon Mccoy Jameson Odonnell Andrew Wilkinson
Sample Output:	Students yet to board the bus are Dario Harding

For Question 2, develop and validate your solution in VS Code. Once you are happy with your solution, go to coderunner on Canvas and paste compareAttendanceSheet() and any

helper function(s) to the answer box!

Sample Run 4.2.1

Inputs: Example_3.txt:

```
Anya Garcia
Oliver Jones
Sofia Smith
```

Example_4.txt:

```
Anya Garcia
Sofia Smith
Oliver Jones
```

Output:

```
Every student has boarded the bus. It's time to leave.
```

Sample Run 4.2.2

Inputs:

```
does_not_exist_1.txt
does_not_exist_2.txt
```

Output:

```
Failed to open attendance files
```

4.3 Compare Trails

This question may require the use of file streams, split(), loops, and functions

Today is your first day as a software engineering intern with the National Park Service. You've started on a busy day, and after the initial meetings, your boss asked you to develop a function to print some statistics about some of the long hikes around the world. You've been given a text file, which contains a list of different hikes. Each line includes the name of a long hike, its length (as a whole number) in miles, and its elevation gain (as a whole number) in feet, each separated by the delimiter "|".

Hint: The split() function from homework 6 could be helpful.

Function: `printHikeStats(string)`	`void printHikeStats(string file_name)`
Purpose:	Given a text file that contains information on the trails, the function will print the following: - Number of valid lines read from the file. A valid line must contain exactly three fields: the hike name, length (as an integer), and elevation gain (as an integer). - Name and the length of the longest hike. - Name and the length of the shortest hike. - Name and elevation gain per mile of the steepest hike. This number should be rounded to one decimal point (use setprecision).
Parameters:	**string** `file_name` - The file name of the text file.
Return Value:	N/A.

Error Handling:	- If the file does not exist, print "Could not open file." - If there is a tie, choose the first one.

Example:	**Example 4.3.1.**
	`// This is only an example usage, and you should develop your own main function`
	`// Assume the proper libraries are included.`
	`// Assume the proper implementation of printHikeStats() is included.`
	`int main() { printHikeStats("long_hikes.txt"); return 0; }`

Sample Input:	Content of long_hikes.txt: The Appalachian Trail|2180|464500 The South West Coast Path|630|115000 Continental Divide Trail|3100|457000 Great Himalayan Trail|1056|289000 The Colorado Trail|578|89000 The Pacific Crest Trail|2650|824370
Sample Output:	Number of lines read: 6. Longest hike: Continental Divide Trail at 3100 miles. Shortest hike: The Colorado Trail at 578 miles. Steepest hike: The Pacific Crest Trail at 311.1 feet gained per mile.

For Question 3, develop and validate your solution in VS Code. Once you are happy with your solution, go to coderunner on Canvas and paste printHikeStats() and any helper function(s) to the answer box!

Sample Run 4.3.1

Content of hikes_blanks1.txt:

```
The South West Coast Path|630|115000


Continental Divide Trail|3100|457000
Great Himalayan Trail|1056|289000
```

Input:
hikes_blanks1.txt
Output:
Longest hike: Continental Divide Trail at 3100 miles.

```
Shortest hike: The South West Coast Path at 630 miles.
Steepest hike: Great Himalayan Trail at 273.7 feet gained per mile.
```

Sample Run 4.3.2

Input:

```
file_does_not_exist.txt
```

Output:

```
Could not open file.
```

4.4 Restaurant Struct

This question may require the use of structs and functions.

You are tasked with designing a program to evaluate restaurants based on a food quality rating, cleanliness rating, and wait time rating, ultimately providing an overall rating. Write a C++ program to accept the name of a restaurant, its food quality rating (0-10), cleanliness rating (0-10), and wait time rating (0-5). Use a function to calculate the overall rating using the function specification and formula given below:

$$\text{Overall Rating} = 0.5 \times \text{food quality} + 0.3 \times \text{cleanliness} + 0.2 \times \text{wait time} \tag{11.1}$$

You must create a struct named Restaurant with the following attributes: the name of the restaurant, food quality rating, cleanliness rating, wait time rating, and overall rating.

Member Type	Member Name	Description
string	`name`	Restaurant's name
int	`food_quality`	Restaurant's food quality rating (0-10)
int	`cleanliness`	Restaurant's cleanliness rating (0-10)
int	`wait_time`	Restaurant's wait time rating (0-5)
double	`overall`	Restaurant's overall rating

Next, create a function that will calculate the overall score of each restaurant.

Function: `getOverallRating(Restaurant)`	`double getOverallRating(Restaurant restaurant)`
Purpose:	Given a object of type `Restaurant`, calculate its overall rating.
Parameters:	**Restaurant** `restaurant` - A `Restaurant` object that contains all the information of the object.
Return Value:	The overall rating of the restaurant. The function should not print anything.
Error Handling:	- If any of the ratings in `restaurant` are not within its bound, return -1. The rating bounds are: food quality (0-10), cleanliness (0-10), wait time (0-5)

Example:	**Example 4.4.1.**
	`// This is only an example usage, and you should develop your own main function`
	`// Assume the proper libraries are included.`
	`// Assume the proper implementation of getOverallRating() and Restaurant struct is included.`
	`int main() { Restaurant r; r.name = "McDonalds"; r.food_quality = 4; r.cleanliness = 7; r.wait_time = 5; r.overall = getOverallRating(r); if(r.overall == -1) { cout << "Invalid rating(s) entered." << endl; } else { cout << "Restaurant: " << r.name << " Overall: " << r.overall << endl; } return 0; }`

Sample Output:	Restaurant: McDonalds Overall: 5.1

For Question 4, develop and validate your solution in VS Code. Once you are happy with your solution, go to coderunner on Canvas and paste Restaurant struct, getOverallRating(), and any helper function(s) to the answer box!

Sample Run 4.4.1

The sample inputs will be given in the order of name, food quality, cleanliness, and wait time, respectively.

Input:

```
Aloy Thai
9
8
3
Output:
Restaurant: Aloy Thai Overall: 6.8
```

Sample Run 4.4.2

```
The sample inputs will be given in the order of name, food quality, cleanliness, and wait
time, respectively.
Input:
Invalid
2
3
-1
Output:
Invalid rating(s) entered.
```

4.5 List of Restaurants

This question may require the use of file streams, split(), loops, arrays, and structs.

In this question, we will use the same `struct` for `Restaurant` from Question 4. You will be given a file, which contains the name of the restaurant, followed by its food quality, cleanliness rating, and wait time rating, all separated by ~. The task will be to evaluate the best restaurant from within the list.

Part A:

First, write a function that reads the restaurant details from a file and populates the required array for further analysis.

Hint: The split() function from homework 6 could be helpful.

Function: `readRestaurantDetails( string, Restaurant[], const int)`	`int readRestaurantDetails(string filename, Restaurant restaurant[], const int MAX_RESTAURANTS)`
Purpose:	Read all the restaurant details from a given file and calculate the overall rating for all restaurants.
Parameters:	**string** `filename` - The text file that contains all the details of the restaurants. **Restaurant**[] `restaurants` - The array that will be populated with all the restaurant objects. **const int** `MAX_RESTAURANTS` - The maximum number of restaurants that can be present in the file containing restaurant details.
Return Value:	If successful (function parameters are correct), return the number of restaurants added to the array. The function should not print anything.
Error Handling:	- If the file does not exist, -1 is returned. - If there is an incorrect number of attributes for a given line of the file, skip the line.

Example:

Example 4.5.1.

```
// This is only an example usage, and you should develop
↪  your own main function
// Assume the proper libraries are included.
// Assume the proper implementation of
↪  readRestaurantDetails(), getOverallRating(), and
↪  Restaurant struct is included.

int main()
{
    Restaurant restaurants[30];

    int res_size =
    ↪  readRestaurantDetails("restaurants.txt",
    ↪  restaurants, 30);

    // Checking if the file was opened correctly
    if (res_size == -1)
    {
        cout << "Failed to open file." << endl;
    }
    else
    {
        for (int i = 0; i < res_size; ++i) {
            cout << "Restaurant: " << restaurants[i].name
            ↪  << " ";
            cout << "Ratings: ";
            cout << restaurants[i].food_quality << " ";
            cout << restaurants[i].cleanliness<< " ";
            cout << restaurants[i].wait_time << " ";
            cout << "Overall: ";
            cout << restaurants[i].overall << " ";
            cout << endl;
        }
    }
}
```

Sample Output:

Restaurant: Wendys Ratings: 6 3 2 Overall: 4.3
Restaurant: Chick-fil-a Ratings: 9 5 5 Overall: 7
Restaurant: Snarfburger Ratings: 10 4 3 Overall: 6.8
Restaurant: Cosmo's Pizza Ratings: 7 3 2 Overall: 4.8

Part B:

Then, create a feature where we can find the best restaurant according to your criteria. The possible options are:

Option 1: "Food Quality"
Option 2: "Cleanliness"
Option 3: "Wait Time"
Option 4: "Overall"

Function: `getBest(Restaurant[], int, string)`	`int getBest(Restaurant restaurants[], int arr_size, string metric)`
Purpose:	Given an array, the function will find and return the index of the best restaurant according to the metric.
Parameters:	**Restaurant**[] `restaurants` - The array that has been populated with the details of all the restaurants. **int** `arr_size` - The maximum number of restaurants that are present in the restaurants array. **string** `metric` - The metric used to find the best restaurant.
Return Value:	If successful (function parameters are correct), return the index of the best restaurant according to the metric. The function should not print anything.
Error Handling:	- If the metric is not a valid option (not "Food Quality," "Cleanliness," "Wait Time," or "Overall"), -1 is returned. - If the line does not have exactly four attributes—name, food quality, cleanliness, and wait time—skip the line. - If there is a tie, choose the first one. - Note: in this assignment, a higher wait time rating means a shorter wait, which indicates better service. For example, a wait time rating of 5 indicates excellent service with minimal wait.

Example:

Example 4.5.2.

```
// This is only an example usage, and you should develop
↪  your own main function
// Assume the proper libraries are included.
// Assume the proper implementation of getBest(),
↪  readRestaurantDetails(), getOverallRating(), and
↪  Restaurant struct is included.

int main()
{
    Restaurant restaurants[30];

    int arr_size =
    ↪  readRestaurantDetails("restaurants.txt",
    ↪  restaurants, 30);

    // Checking if the file was opened correctly
    if (arr_size == -1)
    {
        cout << "Failed to open file." << endl;
    }
    else if (arr_size == 0)
    {
        cout << "Empty file." << endl;
    }
    else
    {
        int idx = getBest(restaurants, arr_size, "Food
        ↪  Quality");

        if (idx == -1)
        {
            cout << "Invalid metric." << endl;
        }
        else
        {
            cout << "Restaurant: " <<
            ↪  restaurants[idx].name << " ";
            cout << "Ratings: ";
            cout << restaurants[idx].food_quality << " ";
            cout << restaurants[idx].cleanliness << " ";
            cout << restaurants[idx].wait_time << " ";
            cout << "Overall: ";
            cout << restaurants[idx].overall << " ";
            cout << endl;
        }
    }
}
```

Sample Output:	Restaurant: Snarfburger Ratings: 10 4 3 Overall: 6.8

For Question 5, develop and validate your solution in VS Code. Once you are happy with your solution, go to coderunner on Canvas and paste the Restaurant struct, getOverallRating(), readRestaurantDetails(), getBest(), and any helper function(s) to the answer box!

Sample Run 4.5.1

The content of restaurants_1.txt:

```
Wendys 6 3 2
Chick-fil-a 9 5 5
McDonalds 2 2 2 2
Snarfburger 10 4 3
Cosmo's Pizza 7 3 2
Pizza Hut 1 2
```

```
Input:
restaurants_1.txt
Cleanliness
Output:
Restaurant: Chick-fil-a Ratings: 9 5 5 Overall: 7
```

Week 11: Objects Continued

Learning Goals

This week you will:

1. Learn the differences between header and source files
2. Compile multiple files and organizing code across multiple files
3. Learn how to nest objects
4. Learn how to work with arrays of objects

1 Background

1.1 Header and Source Files

Creating our own classes with various data members and functions increases the complexity of our program. Putting all of the code for our classes as well as the main functionality of our program into one .cpp file can become confusing for you as a programmer, and we need ways of reducing the visual clutter that this creates. This is why, as we increase the complexity of a program, we might need to create multiple files: header and source files, which capture the *definition* and *implementation* of the class, respectively.

Header Files

Header files have .h as their filename extensions. In a header file, we declare one or more of the complex structures (classes) we want to develop. In a class, we define member functions and member attributes. These functions and attributes are the building blocks of the class.

Example 1.1.1. Example File 0: ClassName.h

```
#include <iostream>
using namespace std;
class ClassName
{
    public:
    .
    .
    .
    private:
    .
    .
    .
};
```

Example 1.1.2. Example File 1: IntelligenceAgency.h

```
#include <iostream>
using namespace std;
class IntelligenceAgency
{
    public:
        IntelligenceAgency();         // Default constructor
        IntelligenceAgency(string classifiedIntelligenceInput); // Parameterized const
        string organization_name;
        string getClassifiedIntelligence(){return classifiedIntelligence};
        void setClassifiedIntelligence(string classifiedIntelligenceInput)
        {classifiedIntelligence=classifiedIntelligenceInput};

    private:
        string classified_intelligence;
};
```

Source File

Source files are recognizable by the .cpp extension. In a source file, we implement the class defined in the header file. Since we are splitting the development of actual code for the class into a definition (header file) and an implementation (source file), we need to link the two somehow.

```
//IntelligenceAgency Example
#include "IntelligenceAgency.h"
```

Or, more generally,

```
//General Example
#include "ClassName.h"
```

In the source file, we will include the header file that defines the class so that the source file is "aware" of where we can retrieve the definition of the class. We must define the class definition in every source that wants to use our user defined data type (our class). When implementing each member function, our source files must tell the compiler that these functions are actually the methods defined in our class definition using the syntax that we showed earlier.

How To Compile Multiple Files

Once you create a project with one or more separate classes, it will be necessary to write multiple files (.h and .cpp) and test them before submitting them. You need to compile and execute your code using the command line. Make sure that you start by changing directories so that you are in the folder where your solution's files are stored. In this example, our folder will be called **project2**. To change to this directory, use:

```
cd project2/
```

When compiling from the command line, you need to specify every `.cpp` file in your project. This means that when you call the g++ compiler, you need to explicitly name the files you're compiling:

```
g++ -std=c++17 file1.cpp file2.cpp main.cpp
```

The compiling command results in the creation of an executable file. Note that header files (.h) are NOT included in compilation commands. If you did not specify a name for this executable, it will be named `a.out` by default. To execute this file, use the command:

```
./a.out
```

You can add the `-o` flag to your compiling command to give the output file a name:

```
g++ -o myName.out -std=c++17 file1.cpp file2.cpp main.cpp
```

And then run it with:

```
./myName.out
```

At this stage, you may also find it helpful to begin compiling with flags that tell you more information about possible errors in your code. These flags include:

`-Wall`

Wall is short for "Warn All", which will turn on most of the warnings in C++. This will help identify various possible ways your code might go wrong, including array bounds errors and other helpful messages.

`-Werror`

Werror will treat all warnings as errors. This will prevent you from skipping past the possible sources of error in your code, and you will need to make sure all warnings are resolved prior to compiling.

`-Wpedantic`

This flag enables warnings that alert you about language constructs that are not ISO C or ISO C++ standard compliant. This is particularly helpful to identify constructs that may not be uniform in other compilers, which could cause problems with your code on other machines. This will help prevent instances where your code works on your personal computer, but does not work on CodeRunner or on the grader's computer. All together, your command line prompt will look something like this:

```
g++ -Wall -Werror -Wpedantic -o myName.out -std=c++17 file1.cpp file2.cpp main.cpp
```

2 PreQuiz

Problem 2.1. True or False: In C++, you must include a semicolon at the end of both struct and class definitions.

Problem 2.2. True or False: In a class, public members are only accessible by member functions within the same class.

Problem 2.3. True or False: Structs in C++ can only contain public data members, while classes can have both public and private data members.

Problem 2.4. Fill in the blanks in the code below:

```cpp
#include <iostream>
#include <string>
using namespace std;

struct ____ {  // Struct name
    string ____;  // Member for the name of the location
    int ____;     // Member for the population
};

int main() {
    ____ myLocation{"____", ____};  // Initialize with name and population

    cout << "Location: " << myLocation.____ << endl;
    cout << "Population: " << myLocation.____ << endl;

    return 0;
}
```

Problem 2.5. Fill in the blanks in the code below:

```cpp
#include <iostream>
using namespace std;

class ____ {  // Class name
public:
```

```
    ____ (int a) {  // Constructor to initialize age
        ____ = a;
    }

    void ____ (int a) {  // Setter method for age
        age = ____;
    }

    int ____() const {  // Getter method for age
        return ____;
    }

private:
    int ____;  // Private data member to store age
};

int main() {
    ____ person(25);  // Create a Person object with age 25
    person.____(30);  // Set the person's age to 30
    cout << "The person's age is: " << person.____() << endl;  // Get and display the age
    return 0;
}
```

3 Recitation

3.1 Spot the Error

Problem 3.1.a. The following code defines a class with a getter method to return a private member. Identify and fix the error.

```
#include <iostream>
using namespace std;

class MyClass {
public:
    MyClass(int val) { value = val; }

    int getValue() { return value; }

private:
    int value;
};

int main() {
    const MyClass obj(42);
    cout << obj.getValue() << endl;
    return 0;
}
```

Problem 3.1.b. The following code is supposed to define a class with a constructor to initialize a private data member. Identify and fix the error.

```
#include <iostream>
using namespace std;
```

```
class MyClass {
public:
    MyClass(int val) { valueX = val; }

    int getValue() const { return value; }

private:
    int value;
};

int main() {
    MyClass obj(10);
    cout << obj.getValue() << endl;
    return 0;
}
```

Problem 3.1.c. The following code is supposed to define a class with a private data member and a setter method. Identify and fix the error.

```
#include <iostream>
using namespace std;

class MyClass {
    int value;

    void setValue(int newValue) {
        value = newValue;
    }
};

int main() {
    MyClass obj;
    obj.setValue(20);
    return 0;
}
```

3.2 Coffee Shop Order System

This exercise focuses on using structs and basic arrays in C++ to create a simple coffee shop ordering system. You will define structs to represent drinks and orders, as well as functions to manage and calculate the order's total.

To begin, define a struct called `Drink`. The `Drink` struct should have three fields: a `string name` for the drink' s name (e.g., "Latte" or "Espresso"), a `string size` for the drink' s size ("small," "medium," or "large"), and a `double price` for the drink' s cost. Next, create a function called `displayDrink` that takes a `Drink` object as an argument and outputs its details in a readable format such as:

```
[Size] [Name]: $[Price]
```

For example, a medium latte priced at $3.50 should display as `Medium Latte: $3.50`.

Now, create a second struct called `Order` to represent a customer' s order. The `Order` struct should contain an array of `Drink` objects and an integer `numDrinks` that tracks the number of drinks in the order. To keep things simple, assume a maximum of 10 drinks per order. Define `numDrinks` with an initial value of 0.

Next, write a function called `addDrink` that takes an `Order` and a `Drink` as arguments, adds the drink to the order' s `drinks` array, and increments `numDrinks`. If the order already has 10 drinks, output an error message and prevent additional drinks from being added.

Now create a function called `calculateTotal` that takes an `Order` as a parameter and returns the total price of all drinks in the order by summing the `price` of each `Drink` in the `drinks` array. Finally, implement a function called `displayOrder` that prints out each drink in the order using `displayDrink` and then displays the total order cost.

In the `main` function, test your structs and functions by creating a few `Drink` objects, an `Order` object, and adding the drinks to the order. Finally, call `displayOrder` to print the details of the order and verify that the total is calculated correctly.

Problem 3.2.a. Write out the steps you would use to solve this problem by hand as pseudocode.

Problem 3.2.b. Write three possible lines you can include in your file to later test your program. Try to pick values that will test different aspects of your program. Follow the steps you wrote for these values to find your result, and verify it.

Problem 3.2.c. Implement your solution in C++ using VS Code. Save, compile, and run it. Test the program with different input files, including boundary conditions.

4 Homework 8

Warning: You are not allowed to use global variables for this assignment.

All function names, return types, and parameters must precisely match those shown. You may not use pass by reference or otherwise modify the function prototypes. You are welcome to create additional functions that may help streamline your code.

Note: Questions 1 through 3 are grouped together as Part I, and Questions 4 through 6 are grouped together as Part II.

In Part I, you'll develop a basic Library System by creating a Library class to manage a collection of books. In Part II, you'll develop a Quest Style Game.

4.1 Library Class

Create a class Library by splitting the code into the following files:

- A header file `Library.h` to declare the definition of the class
- An implementation file `Library.cpp` to implement the class defined in the header
- A driver file `LibraryDriver.cpp` that contains the main function

For Question 1, develop and validate your solution in VS Code. Once you are happy with your solution, go to coderunner and paste `Library.h` and `Library.cpp` files into the answer box. (Note: You are not expected to complete the implementation of `getAveragePageCount()` function for Q1, but do include the function declaration within the header file).

The Library class comprises the following attributes:

Data members (private)

Member Type	Member Name	Description
`string`	`_name`	The name of the Library
`const static int`	`_MAX_BOOKS`	The maximum number of books the library can hold; will be 10 for this question
`int[]`	`_page_counts`	Array containing the page counts of books in the library. The size of this array is `_MAX_BOOKS`
`int`	`_current_books`	The current number of books in the Library

Member Functions (public)

Function	Description
`Default constructor`	Creates a new instance of `Library` by setting `_name` to an empty string, `_current_books` to 0, and each entry in `_page_counts` to 0.
`Library(string)`	Creates a new instance of `Library` with `_name` as the string parameter. The `_current_books` is set to 0, and each entry in the `_page_counts` array is set to 0.
`Library(string, int[], int)`	Creates a new instance of `Library` with `_name` as the string parameter, `_current_books` as the integer parameter, and fills the `_page_counts` array. See Function Specification table below for more details.

`string getName()`	Returns the `_name` of the `Library`.
`int getCurrentBooks()`	Returns the `_current_books` of the `Library`.
`void setName(string)`	Sets the `_name` to the value of the provided string parameter.
`int getPageCount(int)`	Returns the page count at a specified index in `_page_counts`, or -1 if the index is out of bounds.
`bool addPageCount(int)`	Returns `true` if the new page count can be added to the `_page_counts` array. If `_current_books` is already equal to `_MAX_BOOKS`, returns `false`. See Function Specification table below for more details.
`double getAveragePageCount()`	Calculates and returns the average page count of the books in the `Library`. See Question 2 for more details.

Function Specifications:

Function: `Library(string, int[], int)`	`Library(string name, int page_counts[], int arr_size)`
Purpose:	This parameterized constructor creates a new instance of the `Library` class. - Sets `_name` to `name`. - Sets `_current_books` to `arr_size` if `arr_size` doesn't exceed `_MAX_BOOKS`; otherwise, `_current_books` is set to `_MAX_BOOKS`. - Copies elements from `page_counts[]` into the `_page_counts[]` array, up to the size indicated by `_current_books`.
Parameters:	`string name` - The name of the Library. `int page_counts[]` - Array containing page counts of books in the library. `int arr_size` - The size of the `page_counts[]` array.
Return Value:	N/A
Error Handling:	If `arr_size` exceeds `_MAX_BOOKS`, only the first `_MAX_BOOKS` elements are stored in the `_page_counts[]` array. Do NOT print any other statements to handle this scenario.

Example:	**Example 4.1.1.**
	`// Assume the proper libraries are included`
	`// Assume the proper implementation of the class is included`
	
	`int main() {`
	`  string name = "City Library";`
	`  int arr_size = 5;`
	`  int page_counts[arr_size] = {250, 300, 150, 500, 400};`
	`  Library new_library = Library(name, page_counts, arr_size);`
	`}`
	
	Expected Contents of `new_library` Object:
	`_name` = "City Library"
	`_MAX_BOOKS` = 10
	`_current_books` = 5
	`_page_counts[]` = {250, 300, 150, 500, 400, 0, 0, 0, 0, 0}

Function: `addPageCount(int)`	`bool addPageCount(int pages)`
Purpose:	Adds the provided page count to the `_page_counts` array.
Parameters:	`int pages` - The number of pages to be added.
Return Value:	`bool` : Returns a boolean indicating whether the new page count was successfully added to the `_page_counts` array.
Error Handling:	- If `_current_books` is already equal to `_MAX_BOOKS`, do not add or modify any contents, and return `false`. - If the `pages` count is non-positive, do not add or modify any contents, and return `false`. - Return `true` if the new entry is successfully added.

Example:

Example 4.1.2.

```
// Assume the proper libraries are included
// Assume the proper implementation of the Library class
↪  is included

int main() {
    string name = "City Library";
    int arr_size = 5;
    int page_counts[arr_size] = {250, 300, 150, 500, 400};
    Library new_library = Library(name, page_counts,
    ↪  arr_size);
    cout << new_library.addPageCount(90) << endl;
}
```

Expected Return Value:

`true`

Expected Contents of `new_library` Object after Function Call:

`_name` = "City Library"
`_MAX_BOOKS` = 10
`_current_books` = 6
`_page_counts[]` = {250, 300, 150, 500, 400, 90, 0, 0, 0, 0}

4.2 Find Average Page Count of Books in a Library

Write a member function `getAveragePageCount()` in the Library class to find the average page count of the books in `_pageCounts`.

For Question 2, develop and validate your solution in VS Code. Once you are happy with your solution, go to coderunner on Canvas paste code from `Library.h` and `Library.cpp` into the answer box.

Function: `getAveragePageCount()`	`double getAveragePageCount()`
Purpose:	Calculates and returns the average page count of the books in the `Library`.
Parameters:	This member function takes no parameters.
Return Value:	`double` : The average page count of all books in the library. The function should not print anything.
Error Handling:	Return `0` if the `_current_books` is zero.
Example:	**Example 4.2.1.** Note: This is only an example usage of the function; you need to develop your own main function to fulfill the requirement for this problem. **Expected Output:** The expected output after the function call: 320.00

```
// Assume the proper libraries are included
// Assume the proper implementation of the library class
↪  is included

int main()
{
    string name = "City Library";
    int arr_size = 5;
    int page_counts[arr_size] = {250, 300, 150, 500, 400};
    Library new_library = Library(name, page_counts,
    ↪  arr_size);
    cout << fixed<< setprecision(2) <<
    ↪  new_library.getAveragePageCount() << endl;
}
```

4.3 Library with the Largest Average Page Count

Write a function `findLibraryWithLargestAveragePageCount()` to find the `Library` in the array with the largest average page count.

For Question 3, develop and validate your solution in VS Code. Once you are happy with your solution, go to coderunner on Canvas and paste code from `Library.h`, `Library.cpp`, and the function `findLibraryWithLargestAveragePageCount` into the answer box.

Function: `findLibraryWithLargestAveragePageCount(Library[], const int)`	`string findLibraryWithLargestAveragePageCount(Library libraries[], const int ARR_SIZE)`
Purpose:	Finds the name of the library in the `libraries` array with the largest average page count of the libraries.
Parameters:	`Library libraries[]` - Array of Library objects. `const int ARR_SIZE` - The size of the `libraries` array.
Return Value:	`string` : The name of the library with the largest average page count. The function should not print anything.
Error Handling:	- If multiple libraries have the same largest average page count, return the name of the first library with the largest average page count found in the array. - All libraries[] array will be non-empty.

Example:

Example 4.3.1.

```
// Assume the proper libraries are included
// Assume the proper implementation of the Library class
↪  is included

int main() {
    const int ARR_SIZE = 2;
    int page_counts1[3] = {250, 300, 350};
    int page_counts2[4] = {200, 300, 250, 400};
    Library library1 = Library("Downtown Library",
    ↪  page_counts1, 3);
    Library library2 = Library("Uptown Library",
    ↪  page_counts2, 4);
    Library libraries[ARR_SIZE] = {library1, library2};
    cout <<
    ↪  findLibraryWithLargestAveragePageCount(libraries,
    ↪  ARR_SIZE) << endl;
}
```

Expected Output:

```
Downtown Library
```

Explanation:

Average page count of `library1` = 300.00
Average page count of `library2` = 287.50

`library1` has the largest average page count, so we return the name of `library1`, i.e., `Downtown Library`.

4.4 Character Class

Develop a class called Character that will represent either a player or a creature. This class will be split into:

- A header file `Character.h` to declare the definition of the class
- An implementation file `Character.cpp` to implement the class defined in the header
- A driver file `CharacterDriver.cpp` that contains the main function

For Question 4, develop and validate your solution in VS Code. Once you are happy with your solution, go to coderunner on Canvas and paste `Character.h` and `Character.cpp` into the answer box.

Data members (private)

Member Type	Member Name	Description

`string`	`_name`	The characters name
`double`	`_health`	The characters health points
`int`	`_mana`	The amount of magical energy that the character has `_mana`
`char`	`_status`	Current status: 'A' for Active, 'C' for Cursed, 'F' for Frozen
`bool`	`_isCreature`	Specifies if the character is a mystical creature

Member Functions (public)

Function	Description
`Default constructor`	Initializes `_mana` and `_health` to 0, `_name` to an empty string, `_status` to 'A', and `_isCreature` to false
`Character(string, double, int, char, bool)`	Initializes the data members with the provided values. See Function Specification table below for more details
`string getName()`	Returns the `_name` of the `character`.
`double getHealth()`	Returns the `_health` of the character.
`int getMana()`	Returns the `_mana` of the character.
`char getStatus()`	Returns the current status of the character.
`bool getIsCreature()`	Returns if the character is a creature
`void setName(string name)`	Sets the `_name` to the value of the provided string parameter.
`void setHealth(double health)`	Sets the amount of `_health` the character possesses to the given value health only if it is a non-negative value, else it is not changed
`void setStatus(char status)`	Sets the status to one of the allowed characters (A, C, or F) to the given value status only if it is one among A, C or F, else it is not changed
`void setMana(int mana)`	Sets the amount of `_mana` the character possesses to the given value mana only if it is a non-negative value, else it is not changed
`void setIsCreature(bool is_enemy)`	Sets if the character is a creature
`void printStats()`	Prints out the character's stats See Function Specification table below for more details

Function Specifications:

Function: `Character(string, double, int, char, bool)`	`Character(string name, double health, int mana, char status, bool isCreature)`
Purpose:	This parameterized constructor creates a new instance of the `Character` class and sets the data members as provided.

Parameters:	`string name` - The name of the character. `double health` - Health points of the character. `int mana` - Amount of magical energy the character has. `char status` - The current status of the character ('A' for Active, 'C' for Cursed, 'F' for Frozen). `bool isCreature` - Specifies if the character is a mystical creature.
Return Value:	N/A
Error Handling:	`_health` is set to the value of `health` only if it is non-negative; otherwise, it is set to 0. `_mana` is set to `mana` only if it is non-negative; otherwise, it is set to 0. `_status` is set to `status` if it is 'A', 'C', or 'F'; otherwise, it is set to 'A'.

Example:	**Example 4.4.1.** Note: This is only an example usage of the function; you need to develop your own main function to fulfill the requirement for this problem.
	
	`// Assume the proper libraries are included`
	`// Assume the proper implementation of the Character class is included`
	
	`int main() { Character character1("Elara", 8.2, 12, 'F', true); }`
	
	Expected Contents of `character1` Object:
	`_name = "Elara"`
	`_health = 8.2`
	`_mana = 12`
	`_status = 'F'`
	`_isCreature = true`

Function: `printStats()`	`void printStats()`
Purpose:	This member function prints the stats of the character. All `double` values are printed up to 2 decimal places.
Parameters:	This function takes no parameters.
Return Value:	`void` : The function does not return any value.

Example:	**Example 4.4.2.**
	`// Assume the proper libraries are included`
	`// Assume the proper implementation of the Character class is included`
	`int main() { Character character1("Elara", 8.2, 12, 'F', true); character1.printStats(); }`
	
	Expected Output:
	Elara's stats:
	Health: 8.20
	Status: F
	Mana: 12
	Is Creature: Yes

4.5 Game Class

Create a class Game by splitting the code into the following files:

- A header file `Game.h` to declare the definition of the class
- An implementation file `Game.cpp` to implement the class defined in the header
- A driver file `GameDriver.cpp` that contains the main function to test the class

For Question 5, develop and validate your solution in VS Code. Once you are happy with your solution, go to coderunner on Canvas and paste `Game.h`, `Game.cpp`, `Character.h`, and `Character.cpp` into the answer box.

Data members (private)

Member Type	Member Name	Description
`Character`	`_players[2]`	Array storing player objects
`Character`	`_creatures[2]`	Array storing mystical creature characters
`int`	`_num_players`	Current number of players in the game
`int`	`_num_creatures`	Current number of creatures in the game

Member Functions (public)

Function	Description
`Default constructor`	Creates a new instance of `Game` with empty `_players` and `_creatures` arrays and sets `_num_players` and `_num_creatures` to 0.
`Game(Character[], Character[], int, int)`	Creates a new instance of `Game` with the provided parameters. See Function Specification table below for more details.
`int getNumPlayers()`	Returns the current number of players `_num_players`.
`int getNumCreatures()`	Returns the current number of creatures `_num_creatures`.
`void setPlayersList (Character[], int)`	Sets the `_players` array with the provided array of `Character` objects. If the number of objects exceeds 2, only the first two objects are considered. Updates `_num_players` accordingly.
`void setCreaturesList (Character[], int)`	Sets the `_creatures` array with the provided array of `Character` objects. If the number of objects exceeds 2, only the first two objects are considered. Updates `_num_creatures` accordingly.
`bool setPlayer(int, Character)`	Replaces a player object at the specified index in the `_players` array. See Function Specification table below for more details.
`Character getPlayer(string)`	Returns an object from the `_players` array based on the provided name. If no object matches the name, returns a new blank `Character` object.
`bool setCreature (int, Character)`	Replaces a creature object at the specified index in the `_creatures` array. See Function Specification table below for more details.

`Character getCreature(string)`	Returns an object from the `_creatures` array based on the provided name. If no object matches the name, returns a new blank `Character` object.
`int findPlayer(string)`	Returns the index of the player object in the `_players` array based on the provided name. See Function Specification table below for more details.
`int findCreature(string)`	Returns the index of the creature object in the `_creatures` array based on the provided name. See Function Specification table below for more details.
`void printAllStats()`	Prints stats of all the players and creatures. See Function Specification table below for more details.

Function Specifications:

Function: `Game(Character[], Character[], int, int)`	`Game(Character players[], Character creatures[], int num_players, int num_creatures)`
Purpose:	This parameterized constructor creates a new instance of the `Game` class. - Sets `_num_players` to `num_players`. - Sets `_num_creatures` to `num_creatures`. - Assigns elements from the `players[]` array to the `_players[]` array. - Assigns elements from the `creatures[]` array to the `_creatures[]` array.
Parameters:	`Character players[]` - Array of player objects. `Character creatures[]` - Array of creature objects. `int num_players` - Number of players in the array. `int num_creatures` - Number of creatures in the array.
Return Value:	N/A
Error Handling:	If the size of either the `players[]` or `creatures[]` array is greater than 2, only the first two elements are stored in the respective arrays.

Example:	**Example 4.5.1.** Note: This is only an example usage; you need to develop your own function to fulfill the requirement.
	
	`// Assume the proper libraries are included`
	`// Assume the proper implementation of the Character and Game classes is included`
	
	`int main() {`
	`  Character player1("Elise", 100.0, 50, 'A', false);`
	`  Character player2("Pawin", 80.0, 30, 'A', false);`
	`  Character creature1("Goblin", 60.0, 20, 'C', true);`
	`  Character creature2("Dragon", 150.0, 100, 'F', true);`
	
	`  Character players[2] = {player1, player2};`
	`  Character creatures[2] = {creature1, creature2};`
	
	`  Game new_game(players, creatures, 2, 2);`
	`}`
	
	Expected Contents of `new_game` Object:
	`_num_players` = 2
	`_num_creatures` = 2
	`_players[]` = { "Elise", "Pawin" }
	`_creatures[]` = { "Goblin", "Dragon" }
	
	Note: For demonstration purposes, we are only displaying the names of the `_players[]` and `_creatures[]`.

Function: `setPlayer(int, Character)`	`bool setPlayer(int index, Character new_player)`
Purpose:	Replaces the player object in the `_players` array at the specified index with the new `Character` object.
Parameters:	`int index` - Index in the `_players` array. `Character new_player` - New `Character` object to replace the existing one.
Return Value:	`bool` : Returns `true` if the replacement was successful; otherwise, returns `false` if the index is out of bounds.

Example:	

Example 4.5.2.

```
int main() {
    Character player1("Elise", 100.0, 50, 'A', false);
    Character player2("Pawin", 80.0, 30, 'A', false);
    Character new_player("Clara", 120.0, 40, 'A', false);
    Character players[2] = {player1, player2};
    Character creatures[1] = {Character("Goblin", 60.0,
    ↪  20, 'C', true)};
    Game game(players, creatures, 2, 1);
    int index = game.findPlayer("Elise");
    cout << "Status of setting player at index " << index
    ↪  << ": " << game.setPlayer(index, new_player) <<
    ↪  endl;
}
```

Expected Output:

```
Status of setting player at index 0: 1
```

Function: `setCreature(int, Character)`	`bool setCreature(int index, Character new_creature)`
Purpose:	Replace the creature object in the `_creatures` array at the specified index with the new `Character` object.
Parameters:	`int index` - Index of the creature object in the `_creatures` array. `Character new_creature` - The new `Character` object which replaces an older object in the `_creatures` array.
Return Value:	`bool` : Returns `true` if the new object replaces an older object in the `_creatures` array; returns `false` if the index is out of bounds.
Error Handling:	Returns `false` if the attribute index is not within the size of the `_creatures` array.

Example:

Example 4.5.3.

```
int main() {
    Character player1("Brian", 100, 50, 'A', false);
    Character player2("Sam", 80, 100, 'A', false);
    Character creature1("Hydra", 200, 60, 'C', true);
    Character creature2("Minotaur", 150, 30, 'C',
    ↪  true);
    Character players[2] = {player1, player2};
    Character creatures[2] = {creature1, creature2};
    int num_players = 2;
    int num_creatures = 2;
    Game new_game = Game(players, creatures,
    ↪  num_players, num_creatures);

    int index = new_game.findCreature("Minotaur");
    cout << "Status of setting creature at index " <<
    ↪  index << " is " <<
    ↪  new_game.setCreature(index, creature1) <<
    ↪  endl;

    index = new_game.findCreature("Phoenix");
    cout << "Status of setting creature at index " <<
    ↪  index << " is " <<
    ↪  new_game.setCreature(index, creature2) <<
    ↪  endl;
}
```

Expected Output:

Status of setting creature at index 1 is 1
Status of setting creature at index -1 is 0

Function: `findPlayer(string)`	`int findPlayer(string name)`
Purpose:	Finds the index of the player object in the `_players` array whose name matches the provided name.
Parameters:	**string** `name` - Name of the player
Return Value:	`int` : Returns the index of the object in the `_players` array. Returns -1 if the name doesn't match any object.

Example:	**Example 4.5.4.**
	`int main() {`
	`  Character player1("Brian", 100, 50, 'A', false);`
	`  Character player2("Sam", 80, 100, 'A', false);`
	`  Character players[2] = {player1, player2};`
	`  int num_players = 2;`
	`  int num_creatures = 0;`
	`  Game new_game = Game(players, nullptr, num_players, num_creatures);`
	`  cout << "Status of finding player with name Brian: " << new_game.findPlayer("Brian") << endl;`
	`  cout << "Status of finding player with name Jenna: " << new_game.findPlayer("Jenna") << endl;`
	`}`
	Expected Output:
	Status of finding player with name Brian: 0
	Status of finding player with name Jenna: -1

Function: `findCreature(string)`	`int findCreature(string name)`
Purpose:	Finds the index of the creature object in the `_creatures` array whose name matches the provided name.
Parameters:	**string** `name` - Name of the creature
Return Value:	`int` : Returns the index of the object in the `_creatures` array. Returns -1 if the name doesn't match any object.
Example:	**Example 4.5.5.** `int main() { Character creature1("Hydra", 200, 60, 'C', true); Character creature2("Minotaur", 150, 30, 'C', true); Character creatures[2] = {creature1, creature2}; int num_players = 0; int num_creatures = 2; Game new_game = Game(nullptr, creatures, num_players, num_creatures); cout << "Status of finding creature with name Hydra: " << new_game.findCreature("Hydra") << endl; cout << "Status of finding creature with name Phoenix: " << new_game.findCreature("Phoenix") << endl; }` **Expected Output:** Status of finding creature with name Hydra: 0 Status of finding creature with name Phoenix: -1

Function: `printAllStats()`	`void printAllStats()`
Purpose:	Prints the stats of all players and creatures in the game, see sample run below for exact format. Each character's stats are separated by a dashed line, "----------".
Parameters:	This member function takes no parameters.
Return Value:	`void` : The function does not return any value.

Example:

Example 4.5.6.

```
int main() {
    Character player1("Elise", 100.0, 50, 'A', false);
    Character player2("Pawin", 80.0, 30, 'A', false);
    Character creature1("Goblin", 60.0, 20, 'C', true);

    Character players[2] = {player1, player2};
    Character creatures[1] = {creature1};

    Game game(players, creatures, 2, 1);
    game.printAllStats();
}
```

Expected Output:

Elise's stats:
Health: 100.00
Status: A
Mana: 50
Is Creature: No

Pawin's stats:
Health: 80.00
Status: A
Mana: 30
Is Creature: No

Goblin's stats:
Health: 60.00
Status: C
Mana: 20
Is Creature: Yes

4.6 Loading Characters

To begin the game, load characters from text files to populate the player and creature lists.

Create a players.txt file with the following content:

```
name|health|mana|status
Brian|100|50|A
Jenna|80|100|F
```

Create a creatures.txt file with the following content:

```
name|health|mana|status
Goblin|100|50|A
Frog|80|100|A
```

Note: All files used in this problem are in a zipped folder titled `'homework_8_input_files.zip'` located in Canvas under the week 11 module.

Write a function `loadCharacters()` that reads data from a text file and fills an array of Character objects. The text file contains information about different characters, with each character's information separated by the character | and each character listed on a new line. The function specification is provided below.

For Question 6, develop and validate your solution in VS Code. Once you are happy with your solution, go to coderunner on Canvas and paste `Character.h`, `Character.cpp`, `Game.h`, `Game.cpp`, and `loadCharacters` function into the answer box.

Function: `loadCharacters(string, Character[], const int, bool)`	`bool loadCharacters(string filename, Character characters[], const int CHARACTERS_SIZE, bool is_creature)`
Purpose:	Reads data from the specified text file and fills the `characters` array with `Character` objects. Sets `_isCreature` based on the `is_creature` parameter.
Parameters:	**string** `filename` - Name of the text file to be read. **Character** `characters[]` - Array of `Character` objects to be filled. **const int** `CHARACTERS_SIZE` - The maximum size of the `characters` array. **bool** `is_creature` - Specifies if each character read from the file is a creature.
Return Value:	`bool` : Returns `true` if characters were successfully read and stored in the array; `false` if the file cannot be opened.
Error Handling:	- Returns `false` if the file cannot be opened. - Ignores any empty lines in the file. - The first line of the file is ignored as it contains headers.

Example:

Note: This is only an example usage of the function; you need to develop your own function to fulfill the requirement for this problem.

Sample text file:

```
name|health|mana|status
Brian|100|50|A
Jenna|80|100|F
```

Example 4.6.1.

```
int main() {
    string filename = "characters.txt";
    const int CHARACTERS_SIZE = 5;
    Character characters[CHARACTERS_SIZE];
    bool is_creature = false;
    cout << "Function returned value: " <<
    ↪  loadCharacters(filename, characters,
    ↪  CHARACTERS_SIZE, is_creature) << endl << endl;
    // Print the contents of the characters array
    for (int i = 0; i < CHARACTERS_SIZE; i++) {
        if (characters[i].getName() != "") {
            characters[i].printStats();
            cout << endl;
        }
    }
}
```

Expected Output:

Function returned value: 1

Brian's stats:
Health: 100.00
Mana: 50
Status: A
Is Creature: No

Jenna's stats:
Health: 80.00
Mana: 100
Status: F
Is Creature: No

Week 12: Objects Continued

Learning Goals

This week you will:

1. Continue practicing with Objects

1 PreQuiz

Problem 1.1. True or False: In C++, you must explicitly define a constructor for a struct, or it will not compile.

Problem 1.2. True or False: In C++, you must explicitly define a constructor for a class, or it will not compile.

Problem 1.3. True or False: Within C++, we separate our code to capture the definition in header files and the implementation within source files.

Problem 1.4. Short Answer: What is the difference between Data Members and Member Functions?

Problem 1.5. Short Answer: Explain why a struct might be preferred over a class when defining a simple data container in C++.

Problem 1.6. Code Analysis: Identify the issue(s) in the following code snippet.

```
struct City {
    string name;
    int population;

    City(string cityName) {
        name = cityName;
    }
};

int main() {
    City c1("Paris");
    cout << c1.population;
    return 0;
}
```

Problem 1.7. Fill in the blanks in the code below to complete a class that models a guitar with its brand and the number of strings:

```
#include <iostream>
using namespace std;
```

```
class ____ {
public:
    ____ (string b, int s) {
        brand = b;
        strings = s;
    }

    void ____ (int s) {
        ____ = s;
    }

    int ____() const {
        return ____;
    }

    string ____() const {
        return ____;
    }

private:
    string ____;
    int ____;
};

int main() {
    ____ guitar("Fender", 6);
    cout << "Brand: " << guitar.____() << endl;
    cout << "Strings: " << guitar.____() << endl;
    return 0;
}
```

2 Recitation

This week you will create a class to represent a player in the game you are creating for your final project. Your final project will be based on the Game of Life, redesigned to take place in the world of the Lion King. Players will journey across the African Savannah, each as a young lion eager to prove they' re ready to step up as the next 'Pride Leader' after Simba' s retirement. Along the way, they' ll make strategic decisions, face unexpected challenges, and collect Pride Points as they grow and refine their Leadership Traits—Stamina, Strength, and Wisdom. For your player objects you will need to keep track of a few details including the name of the player, their in-game age, and then a few statistics to keep track of in-game points: stamina, wisdom, strength, and total points. All of these numerical values can be integers.

2.1 Header File

You will need to start by composing a header file where you write your class interface. We suggest having these data members:

```
string _name;
int _strength, _stamina, _wisdom, _pride_points, _age;
```

You will then need to add member functions to your public interface. We will be focusing on four key types of functions:

1. Constructors

2. Getters
3. Setters
4. Other member functions

You can start with functions you think would be valuable in all four categories. We will also encourage particular functions as the activity continues.

You should create a source file and a driver file at this point. For each of the following sections you will go to one location around the room to work on that component. You may work on the following sections in any order.

In each section, we will outline minimum functions. However, you are allowed to be flexible in how you choose to design this class – if there is something you believe would be better for usage in your final project, talk to your LAs or TA about your alternative functions.

2.2 Constructors

You will need to create a default constructor and at least one parameterized constructor. You should construct the Players so that they start in the accepted range of starting values. All new Player objects should be restricted to these values:

- Age: 1-20 years old
- Stamina: 100 - 1000 Points
- Strength: 100 - 1000 Points
- Wisdom: 100 - 1000 Points

As the Players progress through the game their stats may deviate from these values, but they should start in this range.

The two suggested constructors are as follows:

Constructor Functions (public)

Function	Description
`Default constructor`	Creates a new instance of `Player` by setting `_name` to an empty string, and all numerical values to their minimum accepted value.
`Player(string name, int strength, int stamina, int wisdom);`	Creates a new instance of `Player` with `_name`, `_strength`, `_stamina` and `_wisdom` coming from the parameters. If the parameters are outside of the accepted range of values, use the minimum accepted value instead. `_pride_points` should default to 0, and `_age` should default to 1.

2.3 Getters

You should create getters for all of your data members. We suggest these:

Getter Functions (public)

Function	Description
`string getName();`	return the `_name` data member.
`int getStrength();`	return the `_strength` data member.
`int getStamina();`	return the `_stamina` data member.

`int getWisdom();`	return the `_wisdom` data member.
`int getPridePoints();`	return the `_pride_points` data member.
`int getAge();`	return the `_age` data member.

2.4 Setters

You should create setters for all of your data members. For each of these functions you should determine whether you would like to return confirmation if a value is updated (create a boolean function) or not (create a void function).

You might find it more helpful to make these addition functions instead, where we add the parameter to the data member instead of replacing the data member. This may make it easier for you to update values as the game progresses. For example, if you wanted to increase the stamina value by 50, you could either use normal getters and setters like so:

```
player1.setStamina(player1.getStamina()+50); //get the current stamina value, add 50 to it.
↪  Then set the stamina to that value.
```

Or you could use an add to stamina function:

```
player1.addStamina(50); //adds 50 stamina to whatever the current stamina value is.
```

You may even want to create both versions. This is entirely up to you. We will suggest the void versions of classic setters below, but feel free to make changes as you see fit:

Setter Functions (public)

Function	Description
`void setName(string name);`	update the stored name for your Player
`void setStrength(int strength);`	update the stored strength
`void setStamina(int stamina);`	update the stored stamina
`void setWisdom(int wisdom);`	update the stored wisdom
`void setPridePoints(int pride_points);`	update the stored pride points
`void setAge(int age);`	update the stored age

2.5 Other Member Functions

You should determine what other member functions might be helpful that go beyond simply getting and setting data values. Three functions are strongly encouraged:

Other Functions (public)

Function	Description
`void trainCub(int strength, int stamina, int wisdom);`	This function can be called with a player chooses to train their cub at the start of the game instead of going straight to the pride lands. This should increase the cub's strength, stamina, and wisdom by the parameters given. It should also decrease the pride points by -5,000.

`void toPrideLands();`	This function can be called when a player chooses to go straight to the pride lands at the start of the game and skip training. This should give the player an immediate boost of +5,000 Pride Points, but come at the cost of reduced Strength (-2,000), Wisdom (-2,000), and Stamina (-1,000).

`void printStats();`	This function should print all of the data members in an organized stats message. You can design this message however you like. Here are a few examples:
	
	Example 2.5.1. Simple stats:
	`Zuri, age 14`
	`Strength: 8700`
	`Stamina: 7100`
	`Wisdom: 13000`
	`Pride Points: 4000`
	
	Example 2.5.2. Stats with a border:
	`~*~*~*~*~*~*~*~*~*~*~*~*~*~*~*~*~*~*~*~*~*~`
	`~*~ Simba, age 3 ~*~`
	`~*~ Strength: 1400 ~*~`
	`~*~ Stamina: 2400 ~*~`
	`~*~ Wisdom: 1000 ~*~`
	`~*~ Pride Points: 5000 ~*~`
	`~*~*~*~*~*~*~*~*~*~*~*~*~*~*~*~*~*~*~*~*~*~`
	
	Example 2.5.3. Stats with ASCII art:
	 
	`PLAYER STATS:`
	
	`Nala, 4 years old`
	`Strength: 4000`
	`Stamina: 3000`
	`Wisdom: 3500`
	`Pride Points: 4200`

enumitem

Project 2: The Circle of Life - Create a Board Game

Learning Goals

This project will get you familiar with:

1. Objects
   - Designing objects and complete member function lists
   - Making objects that store other objects
2. User interface design

Warning: You are not allowed to use global variables, pointers, or pass-by references for this assignment.

1 Introduction

For the Final Project, you will implement a text-based, 2-player board game in C++ that draws inspiration from Disney's *The Lion King*.

Players will journey across the African Savannah, each as a young lion eager to prove they're ready to step up as the next 'Pride Leader' after Simba's retirement. Along the way, they'll make strategic decisions, face unexpected challenges, and collect Pride Points as they grow and refine their Leadership Traits—Stamina, Strength, and Wisdom. Whether navigating tricky terrain or helping a fellow lion in need, each choice will

shape their journey. The player who earns the most Pride Points by demonstrating they have what it takes to guide the pride's future will be chosen as the next 'Pride Leader', ensuring the pride is thriving and keeping the circle of life moving forward!

Unlike previous assignments, this project has no answer key, prebuilt test cases, or set questions. Instead, you'll be given a list of requirements to meet, and it's up to you to fulfill them in the most creative way possible.

This project will be worth more of your grade than any previous assignment. We recommend starting early to give yourself enough time to complete it. This is worth 10% of your final grade.

2 Game Play

The goal of this game is simple: as a player, you'll lead your lion on one of two distinct paths—either through Cub Training or Straight to the Pride Lands. Each path offers its own unique set of advantages and challenges, shaping your journey and influencing your growth along the way. As you advance, you'll cross different terrain: grasslands, drylands, wetlands, and unknown terrain, which introduce unexpected obstacles that may either help or hinder your progress. Navigate these challenges wisely to prove that you have what it takes to become the next 'Pride Leader.'

We have outlined the critical components of the game below, but there are numerous opportunities for creative expression in how you choose to define the details of your game. There is no CodeRunner that you have to match precisely, so take advantage of the flexibility here. Whenever we do not explicitly provide details on a given mechanic in the game, please use your own intuition to design an interesting and cohesive game.

2.1 Starting the Game

This is a two-player game. First, display all available characters and their stats. Then, prompt the players to enter their names and select a lion character. Both players choose a character from the predefined list of five character lions (see character.txt), with each featuring various personal attributes such as name and age (see Character Selection Menu). After selecting a character and before the first turn, players also choose their Path Type, which is either Cub Training or Straight to the Pride Lands (see Path Types). Each path type will contain the same number of special tiles (unique events that affect Leadership Traits) and regular tiles (random events that affect Pride Points). However, the arrangement of the tiles will differ depending on the path chosen (See Visual Board Representation).

For each turn in the game, you should display the Main Menu, including the current position of players on the board and a relevant Main Menu for the space the player is currently on (see Main Menu in Feature Requirements).

2.2 Character Selection Menu

The `characters.txt` file contains a list of playable lions that players can choose from(See the example of the character file below). Each lion entry includes the following attributes: Name, Age, Strength, Stamina, Wisdom, and Pride Point. When players select a character, they will start the game with the predetermined point value for Leadership Traits and Pride Point associated with that character (see example ranges below). Ensure that each character can only be chosen once, and update the menu to show only the available characters.

Example of `characters.txt`:

```
playerName|age|strength|stamina|wisdom|pridePoints
Apollo|5|500|500|1000|20000
Mane|8|900|600|600|20000
Elsa|12|900|700|500|20000
Zuri|7|600|500|900|20000
Roary|18|1000|500|500|20000
```

Each player begins with 20,000 Pride Points before choosing their Path Type. The starting number of Leadership Traits varies for each character. The Path Type chosen can increase or decrease the number of Leadership Traits and Pride Points from the characters' starting values. It's important to note that the values for Stamina, Strength, and Wisdom cannot go below 100 Points throughout the entire game. If the values are below 100, default them to 100.

2.3 Path Types

During game setup, ask each player to choose one of two path types. These path types will impact the starting stats for the player as well as the arrangement of the tiles they will follow on the game board. More details on how each path will impact the game board are available in section 3.2 (Visual Board Representation).

The two path options are listed below:

Cub Training:

This path equips your lion character with essential Leadership Traits—Stamina, Strength, and Wisdom—needed for future leadership. The training requires an investment of -5,000 Pride Points from the starting number of Pride Points; this symbolizes the time and resources dedicated to developing these skills instead of gaining Pride Points. This path also adds 500 Stamina Points, 500 Strength Points, and 1,000 Wisdom Points to the starting amount of your character's Leadership Traits before you start the journey. Choosing this path allows your character to grow and mature, gaining valuable abilities at the expense of early progress. Although this path slows down your initial Pride Point accumulation, the boost in key traits and the opportunity to select an advisor for mentorship prepare your character for greater challenges and future leadership potential.

- Advisor Choice: If the player selects Cub Training, they will be prompted to choose an advisor who grants a unique special ability that protects them during random events that have a negative influence on their Pride Points (please see Advisor List).

Straight to the Pride Lands:

This option lets your lion character jump directly into life on the Savannah with an immediate boost of +5,000 Pride Points from the starting number of Pride Points, allowing early progression and quick success in achieving intermediate goals. This path adds 200 Stamina Points, 200 Strength Points, and 200 Wisdom Points to the starting amount of your character's Leadership Traits before you start the journey, leaving your character with fewer resources to prepare for more difficult situations. Also, you do not get an initial Advisor if you choose this path. Although this path offers a strong head start, it lacks the long-term resilience and special abilities that could be gained through mentorship in Cub Training, making it a riskier approach to becoming a Pride Leader.

2.4 Advisor List

The advisors have special abilities that can protect you in case of a negative random event (see the example below of the `random_events.txt` file). You should choose your advisor wisely.

1. Rafiki - Invisibility (the ability to become un-seen)
2. Nala - Night Vision (the ability to see clearly in darkness)
3. Sarabi - Energy Manipulation (the ability to shape and control the properties of energy)
4. Zazu - Weather Control (the ability to influence and manipulate weather patterns)
5. Sarafina - Super Speed (the ability to run 4x faster than the maximum speed of lions)

Example of `random_events.txt`:

```
Description | PathType (0 = cubTraining; 1 = straight to the pride lands; 2 = either path) |
Advisor (0 = none; 1 = Rafiki; 2 = Nala; 3 = Sarabi; 4 = Zazu; 5 = Sarafina) | PridePoints (lose or gain)

Desert storm sweeps through the territory|2|4|-500
Fatigue from intense training with pride warriors|0|3|-200
Challenging night watch duty under pitch-black conditions|1|2|-400
Extra energy from bountiful season|1|0|800
Observed a rare natural phenomenon|0|0|600
Gained wisdom from observing Rafiki' s rituals|1|0|500
```

For example, players on either path could encounter "Desert storm sweeps through the territory", but if you have Zazu as your advisor, you will safely bypass that event. If you do not have Zazu as advisor, you will suffer -500 pride points.

However, only players that chose to do cub training would be able to encounter "Fatigue from intense training with pride warriors". They would be safe if they had Sarabi as an advisor and skip the event, but if they had any other advisor, they would lose 200 pride points.

2.5 Core Concepts of the Game

1. The game starts by selecting your preferred lion character from a list of available character names.
2. Select a path type for your chosen Lion character from the two path types: Cub Training or Straight to the Pride Lands. Each of these paths differs in what they offer.
3. The map is a two-lane path (one lane for each path type). Depending on the path type chosen, both players can be on the same path or on different paths (See Visual Board Representation for details). Players navigate by spinning a virtual spinner (players can land on any number from (1-6) to determine how far to move forward. You can do this by using a random number generator.
4. Players alternate taking turns playing each round of the game.
5. There are special tiles on the game board where unique events happen that can affect your Leadership Traits or other aspects of the game (please see Special Tiles section)
6. The players must manage their total Stamina, Strength, Wisdom, and Pride Points throughout the game.
7. If you did not choose the cub training path, advisor selection is available on some special tiles where players can select an advisor (see Counseling Tile under Special Tiles). Advisors have special abilities in the random event files, depending on the path type of the character that holds it. If a player already has an advisor, you cannot select that same advisor.
8. Negative events occasionally occur, causing players to lose Leadership Traits or Pride Points that can setback their journey to becoming the Pride Leader.
9. Once all players reach Pride Rock, represented by the Orange Tile, the player who has the highest Pride Points wins the game.

3 Feature Requirements

The minimum requirements for this final project are given in the following sections. You are not allowed to use pointers, global variables, or pass-by-reference in the project.

3.1 Interactive Components

- Players move forward on the board by spinning a virtual spinner, similar to the image shown below. The number of tiles they land on determines how many tiles they advance on their turn. You can do this by using a random number generator (you do not need to actually draw a spinner – printing the random output is sufficient).

- Design the two distinct starting paths for players to choose from—one path requires an initial investment for *greater resources* (such as less Pride Points but more Leadership Trait Points) that lead to long-term rewards, while the other provides an immediate reward with higher starting points but *fewer resources* (such as more Pride Points but less Leadership Trait Points), impacting potential growth over time.
- Respond to in-game events and challenges, such as choosing actions, providing answers, or otherwise reacting to encountered situations, which may include solving riddles or making strategic choices based on the scenario presented.

Main Menu Provide an interactive menu with at least five options. For example, when players open the main menu, they may see the following options:

1. Check Player Progress: Review Pride Point and Leadership Trait stats.
2. Review Character: Check your character name and age.
3. Check Position: Display board to view current position.
4. Review your Advisor: Check who your current advisor is on the game.
5. Move Forward: For each player's turn, access this option to spin the virtual spinner.

At least 2 of the options should have secondary layers. For example:

- Review Your Advisor could have an additional option that displays the advisor' s ability and prompts the player to confirm using it.

- Check Player Progress has an additional option to convert Leadership Traits to Pride Points (see winning the game details on the purpose of converting Leadership Traits to Pride Points).

Example of the Main Menu:

```
Main Menu: Select an option to continue
1. Check Player Progress (1)
2. Review Character (2)
3. Check Position (3)
4. Review your Advisor (4)
5. Move Forward (5)

Please choose an option using the corresponding number:
```

3.2 Visual Board Representation

Step 1. Set Up the Initial Board Structure

- Use the provided board files, `board.cpp` and `board.h`, to display the 52-tile trail on a 2-lane path (one lane for each path type, with a total of 104 tiles) (see Figure 1).
- Ensure that the Starting Tile (Gray Color) and Ending Tile(Orange Tile) are visible on both lane paths. You may want to label which path corresponds to Cub Training, and which path corresponds to Straight to the Pride Lands.
- Ensure that the board contains a minimum of 102 tiles in various colors (e.g., Pink, Green, Blue) to represent different tile types.
- **Randomize the tile arrangement**: The tiles on the map should be randomized each time the game starts. Additionally, ensure that the distribution rules for the tiles differ between the two paths (Cub Training and Straight to the Pride Lands), creating unique gameplay experiences for each path. You should design your own rules for the tile generation based on what you think makes sense and would create a well-balanced game. Should players who have advisors and went through Cub Training have fewer challenges right away, while those who went straight to the pride lands are more likely to struggle early on? You must **randomize each path**, and the distribution rules for each path **must be different**.
- The provided example board currently displays starting tile, regular tile, special tile, and ending tile templates. You need to implement the functioning of each of these tile types on the board. For each lane, ensure there are at least 20 special tiles with unique characteristics spread randomly across the board. Each tile should have a distinct purpose; examples are detailed below in the "Tile Details/Descriptions" subsection.
- The Final Tile is named "Pride Rock." This tile signifies the game's endpoint and should convert the Leadership traits to Pride Points to determine and congratulate the winner!

Figure 8.4: Example of the 2-lane Board when players choose **different path types**

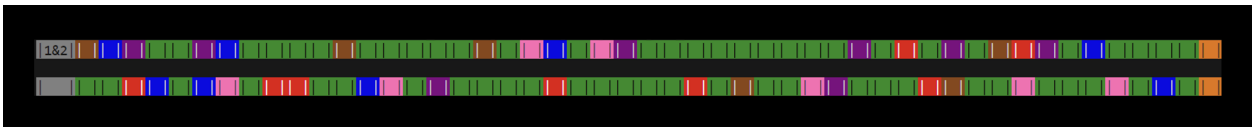


Figure 8.5: Example of the 2-lane Board when players choose **the same path type**

Step 2. Implement Player Representation on the Board

Add functionality to visually represent each player's position on the board using numbers:

- Use "1" to represent Player 1 and "2" to represent Player 2.
- The Final Tile is colored orange, and it represents Pride Rock.
- The Starting Tile is colored gray, and it represents the Starting Point after both players have decided on their path choices.
- Update each player's position on the board and display the board at the end of every turn. The position should reflect any forward or backward moves resulting from the game spinner or event outcomes.

3.3 Tile Details/Descriptions

Regular Tiles: These tiles represent random events that have probability-based outcomes.

- Regular tiles are colored green to represent the grassy lands of the Savannah; anything can happen in this open landscape. For each path, ensure there is a minimum of 20 regular tiles on the board. When triggered, randomly select an event from the `random.txt` file's event list that impacts the player's Pride Points either positively or negatively. Implement a 50% chance for an event to be triggered when landing on these regular tiles. The random events are dependent on the path type. In addition, the current player's advisor protects against random events that have a negative influence on Pride Points.

Special Tiles: These tiles are full of surprises, adding twists and turns to your journey. There should be at least 20 special tiles with random positions across each lane of the board. When triggered, events from the special tiles impact the player's current positions, Leadership Traits (Stamina, Strength, and Wisdom), or other attributes. The special tiles are listed below:

- Oasis Tile (blue color tile): You've found a peaceful oasis! This grants the player an extra turn to keep moving forward—take a deep breath and relax; you also gain 200 Stamina, Strength, and Wisdom Points.
- Counseling Tile (pink color tile): Welcome to the land of enrichment - when landing on this tile, your Stamina, Strength, and Wisdom Points increase by 300, and you get to choose an advisor from the available list of advisors. If you already have an advisor, you can switch your advisor out for a different one from the list or keep your original advisor. Don't forget - an advisor can protect you from random events that negatively impact your Pride Points.
- Graveyard Tile (red color tile): Uh-oh, you've stumbled into the Graveyard! This forces the player to move back 10 tiles and lose 100 Stamina, Strength, and Wisdom Points.
- Hyenas Tile (brown color tile): The Hyenas are on the prowl! They drag you back to where you were last, and the journey comes at a cost. This returns the player to their previous position. In addition, the player's Stamina Points decrease by 300 Points.
- Challenge Tile (purple color tile): Time for a test of wits! Land here, and you'll face a riddle randomly pulled from the `riddles.txt` file. Answer correctly, and you'll earn a boost of 500 Points to your Wisdom Trait—your cleverness pays off!

Example of the `riddles.txt` file:

```
Question(answer specifications)|Answer
I can hold multiple functions, but I' m not a list. What am I? (single word, lowercase)|class
I start a comment in C++. What am I? (symbol)|//
I am the number of bits in a byte. What am I? (integer)|8
```

4 Implementation Requirements

4.1 Class Structures:

These classes are strongly encouraged, but this is not an exhaustive list of classes that may be valuable in your code.

- `Board` Class: Represents the game board with an array or vector of `Tile` objects, each with specific properties and types.
- `Tile` Class: Represents each tile, with attributes for type (e.g., Green, Pink, Blue) and references to any potential effects (e.g., challenges, random events). This could also be a struct.
- `Player` Class: Tracks player attributes, including Pride Points, Stamina, Strength, and Wisdom. Includes methods for adjusting these attributes based on events.

4.2 Data Members and Methods:

- Ensure each required class has at least 4 data members, with appropriate getters, setters, and constructors.
- Use an array or vector of objects from one user-defined class within another (e.g., an array of Tile objects within the `BoardClass`).

4.3 File I/O:

- Write game stats to a file at the end of the game.
- Read files such as `random_events.txt`, `riddles.txt`, and `characters.txt`

4.4 Code Structure and Flow Control:

Include the following:

- 6+ if-else statements for game events, advisor choice, and board tile actions.
- 6+ loops, including at least 2 nested loops, to manage board traversal, player turns, and challenge encounters.

4.5 Final Evaluation:

- **All players reach final tile**: Calculate each player's total Pride Points. For every 100 points in Stamina, Strength, or Wisdom, add 1,000 Pride Points to their Pride Points total.
- **Declare the winner**: Display the name of the lion with the highest Pride Points as the winner, along with each player' s final stats.

Note: For any requirements not explicitly detailed, use your own creativity to design an interesting and cohesive gameplay experience.

5 Group Work Overview

You are allowed to work with (at most) one other student who is currently in CSCI 1300 this semester (in any section). You may also work by yourself if you choose.

If you choose to work in a group, there are additional requirements. We expect that you will contribute equally to the project. Both group members must submit a zip file for the project, and the solution files can be the same. Indicate your partner' s name in the comments at the top of each code file.

5.1 Group Requirements

In addition to the requirements reviewed above, if you work in a group, you must also implement a sorting algorithm and apply it to a task in your program, and at least one other customization in your code. You should not use a Library function or any external resources to implement the sorting algorithm. Possible customizations are listed below.

One situation where the sorting functionality would be useful is for a ranking task, for example, ranking all the players who have completed this game based on their final pride point values.

Note:

- If you work in a team and you do not implement a sorting task, 50 points will be deducted from your point total.
- We expect that you will be contributing to the project equally. Both group members must submit a zip file for the project, and the solution files can be the same. Indicate your partner' s name in the comments at the top of each code file. Both partners will book an interview grading appointment together, and TAs will grade you individually.

6 Customization

This game is inspired by *The Lion King*, and you also have the flexibility to expand on this theme. Listed below are some suggestions on how to shake things up and put your own spin on the project! **Note:** Any changes you make cannot reduce the complexity of the game or the information in files.

- **Game Balancing**: We encourage you to explore the balance of your game. Feel free to tweak any numbers in this document to make the game more competitive or interesting.
- **Modify the text file**: You can also modify and add to the txt files (random_event, riddles, characters, etc.). Ensure that the number of elements in each file meets or exceeds the current minimum specified.
- **Game Complexity:** You can also shake things up in more creative ways to add more complexity to the game. Just ensure that anything you change does not reduce the current complexity of the game or information, only changes or expands upon the base outlined here.
- **Change the theme**: The game could take place in an entirely different world or environment, like a deep ocean, outer space, or ancient civilizations.
- **Alter the characters**: Instead of lions, you could have characters from another animal group, fantasy beings, or even human explorers, each with their own attributes.
- **Customize the challenges**: Introduce unique challenges that fit your new theme. Instead of solving riddles, you can compete in skills-based contests or games like rock-paper-scissors or tic-tac-toe.
- **Add unique interactions**: Think about different ways the players can interact with the game, such as special abilities or items that fit the new setting.
- **Modify the Game Board**: Increase the number of tiles or change tile colors, customize the name of the final tile, or change the player representations on the map to symbols rather than numbers. Make sure to stick to the requirements of the number of tiles and at least 2 lanes on the board.

- **Create more classes/structs:** You can also add more classes and structs to represent the complexity you want to implement in your game.

This is a chance to make something truly your own while still meeting the required features. Feel free to explore new ideas, but remember to maintain the core complexity and integrity of the game. Get creative and have fun designing your unique version!

7 Extra Credit

The following extra credit opportunities are available for this project.

1. **Presentation (10 points):** Present your project during recitation or record a video of your presentation (3-5 minutes).
2. **Unique Special Tiles for each Path Type (5 points):** You can add more functionalities that allow each of the path types (cub training and straight to the pride lands) to have unique special tiles. These special tiles will only appear on the path lane of players who choose the path type.
3. **Multiplayer game - More than 2 (5 points):** Implement functionalities in the game to allow for more than 2 players.
4. **Extra Boost on Positive Random Event(5 points):** You can boost the number of pride points gained on positive random events. The boost should be specific to the advisor the player currently has.
5. **Same Tile Constraint (5 Points):** If both players are on the same tile number, they can battle for more Pride Points and who gets to stay on the tile.
6. **Additional Factors that Alter Event Outcomes (5 points):** Add different factors that can alter event outcomes, such as age having an effect on Pride Points.

   Here is an example of what this might look like:

```
Gained insight from observing cub play. This applies to characters under the age of 5.
Strengthened muscles through special training exercises.
    This applies to characters between the ages of 5 and 10.
Passed down wisdom to younger pride members. This applies to characters over the age of 15.
```

8 Timeline

- **Friday, November 22th at 11:59 pm:** Submit class files & Code Skeleton. The instructions for the skeleton are outlined below:
  - Describe the classes in your program, detailing all data members and member functions. Provide complete class interfaces in header (.h) files as blueprints, including accessible functions but not detailed implementations. Implement basic functions like getters and setters in source (.cpp) files. Submit both the header (.h) and source (.cpp) files for each class.
  - For complex class functions, define the function signature, return type, and the concept/objective of each function (TIP: It might be helpful to use the function table format from the workbook). We suggest you pseudocode each complex function (this will only help you!).
- **Friday, December 6th at 11:59 pm:** Interview Grading Sign-Up deadline. You must sign up for an interview grading timeslot no later than Friday, December 6th at 11:59 pm. The interviews will take place between December 12th and December 18th. If you do not sign up or miss your interview, then no points will be awarded for the project.
  - During the interview grading, TAs will be playing your game and asking questions about it. They will also ask about your implementations and conceptual questions.

- **Wednesday, December 11th at 11:59 pm:** Final Deliverables. Your project will be due on Tuesday, December 11th, at 11:59 pm. You must submit a zip file to the Project 2 assignment on Canvas, including all .h and .cpp files. The submission should compile and run. TAs will also be grading comments and code style.
- **Monday or Tuesday, December 9th or 10th:** Project presentation. You may present your project during your recitation for extra credit.
- **Saturday, December 14th at 11:59 pm:** Project Report. Write a 1-2 page report containing the following reflection questions:
  - How did you prepare for the project?
  - How did you develop our code skeleton? In what way(s) did you use your code skeleton?
  - Reflect on how you could have done better or how you could have completed the project faster or more efficiently.
  - In addition, write a paragraph answering the following question, in the context of the Project in CSCI 1300: Did you have any false starts, or begin down a path only to have to turn back when figuring out the strategy/algorithm for your Final Project program? Describe in detail what happened, for example, what specific decision led you to the false starts, or, if not, why do you think your work had progressed so smoothly. In either case, give a specific example.
  - The report should be a 1-inch margin, single space, 12pt font size, Times New Roman. Submit a report as PDF to Project2 Report on Canvas.

9 Project 2 Points

Project 2 is worth 200 points. Here is a summary of the points.

Criteria	**Points**
Code Skeleton	10
Minimum Implementation Requirements	25
Game functionality	90
Game Compilation, Algorithm, Comments, Style, Interview Questions	75
Total	200

Note:

- If your code does not compile, you cannot score above 50 points for the project
- The use of global variables, pointers, or pass-by-reference will result in a 0 on the entire project

Week 13: Vectors

Learning Goals

This week you will:

1. Learn what vectors are
2. Practice using vectors

1 Background

1.1 Vectors

Let's start with something we already know about - Arrays.

To recap, an array is a contiguous series that holds a fixed number of values of the same datatype.

A vector is a template class that uses all of the syntax that we used with vanilla arrays, but adds in functionality that relieves us of the burden of keeping track of memory allocation for the arrays. It also introduces a bunch of other features that makes handling arrays much simpler.

First things first. We need to include the appropriate header files to use the vector class.

```
#include <vector>
```

We can now move on to declaring a vector. This is general format of any vector declaration:

```
vector <datatype_here> name(size);
```

The size field is optional. Vectors are dynamically-sized, so the size that you give them during initialization isn't permanent - they can be resized as necessary.

You can access elements of a vector in the same way you would access elements in an array, for example array[4]. Remember, indices begin from 0.

The C++ vector class comes with several member functions available in the C++ reference guide, but following are the ones you will need in this week:

- `size()` return the size of a vector
- `at()` takes an integer parameter for index and returns the value at that position

Adding elements to the vector is done primarily using two functions

`push_back()` takes in one parameter (the element to be added) and appends it to the end of the vector. Here is an example:

Example 1.1.1. How to use `push_back()` with vectors:

```
vector <int> vector1; // initializes an empty vector
vector1.push_back(1); //Adds 1 to the end of the vector.
vector1.push_back(3); //Adds 3 to the end of the vector.
vector1.push_back(4); //Adds 4 to the end of the vector.
cout<< vector1.size(); //This will print the size of the vector - in this case, 3.
// vector1 looks like this: [1, 3, 4]
```

`insert()` can add an element at some position in the middle of the vector.

Example 1.1.2. How to use `insert` with vectors:

```
// vectorName.insert(vectorName.begin() + position, element)
vector1.insert(vector1.begin() + 1, 2);
cout << vector1.at(1) << endl; // 2 is at index=1
// vector1 looks like this: [1, 2, 3, 4]
```

Here, the `begin` function returns an iterator to the first element of the vector. Due to the nature of an iterator, this allows for the utility of using `begin()` to refer to the first element and `begin()+k` would refer to the kth element in the vector, starting at 0.

Elements can also be removed.

`pop_back()` deletes the last element in the vector.

Example 1.1.3. How to use `pop_back()` with vectors:

```
vector <int> vector1; // initializes an empty vector
vector1.push_back(1); //Adds 1 to the end of the vector.
vector1.push_back(3); //Adds 3 to the end of the vector.
vector1.push_back(4); //Adds 4 to the end of the vector.
vector1.pop_back(); //Removes the last element of the vector.
//vector1 looks like this: [1, 3]
```

`erase()` can delete a single element at some position, which is shown below using the iterator function of `begin()` to erase the first element of the vector.

Example 1.1.4. How to use `erase()` with vectors:

```
// vector_name.erase(vector_name.begin() + position)
vector1.erase(vector1.begin() + 0);
cout << vector1.at(0) << endl; //2 is at index=0
// vector1 looks like this: [2, 3, 4]
```

It may be useful to think of vectors relationship to arrays as something similar to strings vs arrays of characters; they are similar concepts, but with added utility and flexibility that is helpful. Vectors are also passed by value (like strings) instead of passed by reference (like arrays). This might look something like:

Example 1.1.5. Full vector example:

```
void myVecEditFunction(vector <int> vec){
   vec.erase(vec.begin());
   //vec now contains the original vector minus the starting element
}

...

int main(){
   vector <int> originalVector = {1, 2, 3};
   myVecEditFunction(originalVector);
   //originalVector still looks like [1, 2, 3]
}
```

1.2 Randomness

Random numbers are a valuable tool for a number of applications, including writing games where we want random chance to be a factor. There are limitations in being able to make a truly random number generator with code, but we have tools to get close enough.

`rand()` function is an inbuilt function in C++ STL, which is defined in header file `<cstdlib>`. `rand()` is used to generate a series of random numbers. The random number is generated by using an algorithm that gives a series of non-related numbers whenever this function is called. The `rand()` function is used in C++ to generate random numbers in the range `[0, RAND_MAX)`.

`RAND_MAX`: It is a constant whose default value may vary between implementations but it is granted to be at least 32767.

The syntax for the function is: `int rand(void);` where `int` is the return type and the parameter list is void (i.e. needs to parameters).

However in order to ensure that the random sequence of numbers is unique each time, we must choose a unique starting seed for the random generator.

`srand()` function is an inbuilt function in C++ STL, which is defined in `<cstdlib>` header file. `srand()` is used to initialize random number generators. The `srand()` function sets the starting point for producing a series of pseudo-random integers. If `srand()` is not called, the `rand()` seed is set as if `srand(1)` were called at the program start. Any other value for seed sets the generator to a different starting point.

Here are the two function prototypes for `srand()` to see the syntax:

```
void srand( unsigned seed );
int srand( unsigned int seed);
```

Seeds the pseudo-random number generator used by rand() with the value seed. Parameter **seed**: A seed for a new sequence of pseudo-random numbers to be returned by successive calls to rand()

Return value: This function returns a pseudo-generated random number.

Note: The pseudo-random number generator should only be seeded once, before any calls to rand(), and at the start of the program. It should not be repeatedly seeded or reseeded every time you wish to generate a new batch of pseudo-random numbers.

Standard practice is to use the result of a call to `srand(time(0))` as the seed. However, `time()` returns a `time_t` value which varies every time and hence the pseudo-random number varies for every program call.

Here are a few examples of using randomness.

Example 1.2.1. A short example to roll a die:

```
int main(){
    //declare variables
    int dieRoll;
    //random seed
    srand(time(0));

    dieRoll = rand()%6; //randomly generate a number 0 through 5
    dieRoll+=1; //add 1 to make it now store a number 1 through 6

    //printing a random number stored in a variable
    cout << "Our dice roll is " << dieRoll << endl;

    //printing the random number directly
    cout << "Our D20 rolled " << rand()%20+1 << endl;
}
```

Example 1.2.2. A long example to play Rock, Paper, Scissors:

```
int main(){
    //declare variables
    char userChoice; //to store the user's choice of R, P or S
    int compChoice; //to store the computer's choice
    srand(time(0));

    //ask the user for their choice
    cout << "Rock (R), Paper (P), or Scissors (S)?" << endl;
    cin >> userChoice;

    compChoice = rand()%3; //randomly choose a number 0, 1 or 2

    //Arbitarily choosing that 0 = R, 1 = P, 2 = S for comparison
    switch(userChoice){
        case 'R':
            switch(compChoice){
                case 0:
                    cout << "Both chose rock-- tie!" << endl;
                    break;
                case 1:
                    cout << "You chose rock, the computer chose ";
                    cout << "paper -- you lose." << endl;
                    break;
                case 2:
                    cout << "You chose rock, the computer chose ";
                    cout << "scissors -- you win!" << endl;
            }
            break;
        case 'P':
            switch(compChoice){
                case 0:
                    cout << "You chose paper, the computer chose ";
                    cout << "rock -- you win!" << endl;
                    break;
                case 1:
                    cout << "Both chose paper -- tie!" << endl;
                    break;
                case 2:
                    cout << "You chose paper, the computer chose ";
                    cout << "scissors -- you lose." << endl;
            }
            break;
        case 'S':
            switch(compChoice){
                case 0:
                    cout << "You chose scissors, the computer ";
                    cout << "chose rock -- you lose." << endl;
                    break;
                case 1:
                    cout << "You chose scissors, the computer ";
                    cout << "chose paper -- you win!" << endl;
```

```
                    break;
                case 2:
                    cout << "Both chose scissors -- tie!" << endl;
            }
            break;
        default:
            cout << "Invalid choice." << endl;
    }

}
```

1.3 Function Parameters: Pass By Reference

In C++, you can pass parameters to functions either by value or by reference. Passing by reference allows you to modify the original variable within the function.

The syntax is:

```
{
    // Function body
}
```

Example:

```
#include <iostream>
using namespace std;

// Function to increment a number by value
void incrementByValue(int num)
{
    num++;
}

// Function to increment a number by reference
void incrementByReference(int &num)
{
    num++;
}

int main()
{
    int x = 5;

    cout << "Before increment: " << x << endl;
    incrementByValue(x); // passing x by value
    cout << "After increment: " << x << endl;

    cout << "---" << endl;

    cout << "Before increment: " << x << endl;
    incrementByReference(x); // passing x by reference
    cout << "After increment: " << x << endl;

    return 0;
}
```

Sample Run 1.3.1

```
Before increment: 5
After increment: 5
---
Before increment: 5
After increment: 6
```

Key points to remember:

- Parameters passed by reference are indicated by & in the function declaration.
- Changes made to the reference parameter within the function affect the original variable.
- Pass by reference avoids unnecessary copying of the argument to the parameter variable, which can be more efficient.
- Arrays are passed by reference by default (so there is no preceding & for an array parameter).
- References cannot be reassigned to refer to a different variable once initialized.

2 PreQuiz

Problem 2.1. True or False: Header files in C++ define the implementation details of functions and classes.

Problem 2.2. True or False: When compiling multiple C++ files, header files (.h) are included in the compilation command.

Problem 2.3. True or False: Source files in C++ use the .cpp extension and implement the class defined in the corresponding header file.

Problem 2.4. Short Answer: Explain the purpose of using both header files (.h) and source files (.cpp) in organizing C++ programs. How do they contribute to code clarity and maintenance in larger projects?

Problem 2.5. Fill in the blanks for this code which defines a State class which has both a default and parameterized constructor:

```cpp
#include <iostream>
#include <string>
using namespace std;

class State {
public:
    State() {
        ______ = "Unknown";
        ______ = 0;
        ______ = 0.0;
    }

    State(string stateName, int statePopulation, double stateArea) {
        ______ = stateName;
        ______ = statePopulation;
        ______ = stateArea;
    }

    void displayInfo() {
        cout << "State: " << ______ << endl;
```

```
        cout << "Population: " << ______ << endl;
        cout << "Area: " << ______ << " sq miles" << endl;
    }

private:
    string name;
    int population;
    double area;
};

int main() {
    ______ defaultState;
    defaultState._________();

    ______ customState("California", 39538223, 163696);
    customState._________();

    return 0;
}
```

Problem 2.6. Fill in the blanks for this code which is based on the last problem and has been changed to use both a header and source file.

Here is the header file:

```
#include <string>
using namespace std;

class State {
public:
    // Default constructor
    ______ ();

    // Non-default constructor
    ______ (string stateName, int statePopulation, double stateArea);

    void ______ ();  // Declare the method to display state information

private:
    string ______;     // Variable to hold the state's name
    int ______;        // Variable to hold the state's population
    double ______;     // Variable to hold the state's area
};
```

Here is the corresponding source file:

```
// State.cpp
#include "State.h"
#include <iostream>
using namespace std;

// Default constructor implementation
State::______() {
    name = "Unknown";
    population = 0;
    area = 0.0;
}
```

```cpp
// Non-default constructor implementation
State::______(string stateName, int statePopulation, double stateArea) {
    name = ______;
    population = ______;
    area = ______;
}

// displayInfo method implementation
void State::______() {
    cout << "State: " << ______ << endl;
    cout << "Population: " << ______ << endl;
    cout << "Area: " << ______ << " sq miles" << endl;
}
```

3 Recitation

3.1 The Map Class

This assignment is part of your final project. You will need to expand on the map class PROVIDED ON CANVAS. Take time to get familiar with the code and then begin modifying the class.

You are a player in the game tasked to explore the tiles on the map. Write a C++ program where you will generate a map that satisfies the minimum requirements specified in the project (see the Project 2 section of this workbook). You will need to have two lanes: one to represent when a player goes to Cub Training, and one to represent when a player goes Straight to the Pride Lands.

Write code to enable a user to navigate through one lane on the map and visit the tiles. You should print out placeholder information for each type of tile, but you do not have to fully implement all of the tile types just yet. Make sure that your map is randomized for each new game, and that the generation rules are different for the two paths. For example, you could do something like this:

Straight to the Pridelands:

- 20 Grasslands tiles; 29 special tiles; 1 start and 1 end tile. There are more special tiles in this path, because they do not have an advisor to put them on the clearer path.
- The player is more likely to go to a bad tile, since they do not have an advisor to help them avoid those tiles. They are more likely to make these mistakes early, before they learn. Therefore, for the first half of the lane there is a 25% chance for a special tile to be a graveyard tile, and a 25% chance for the special tile to be a hyena tile. In the second half of the lane, these odds are reduced to a 15% chance for the graveyard, and a 15% chance for the hyena tile.
- Since the players do not already have an advisor, they are more likely to seek one out. They have a 20% chance of the random tile being an advisor tile.
- They are likely to learn from their mistakes and have an easier time finding an oasis as the game goes on. In the first half of the lane, there is a 5% chance of a special tile being an oasis tile. For the second half, there is a 25% chance of finding the oasis.
- Challenge tiles are uniformly likely with a 25% chance throughout the lane.

Cub Training:

- 29 Grassland tiles; 20 special tiles; 1 start and 1 end tile.
- These players are less likely to visit a negative tile throughout the game, because their advisors will help them avoid them. So they have a 20% chance for a graveyard and a 20% for a hyena tile at any point during the game.

- These players already have an advisor and are less likely to seek one out, so there is a 15% chance for a tile to be an advisor tile.
- These players are likely steered towards the easy path with an oasis at the beginning of their journey by their advisor. In the first half of the lane each special tile has a 25% chance of being an oasis. In the second half, they have a 15% chance of being an oasis.
- These players are more likely to want to test themselves after their training. In the first half of the lane there is a 20% chance of finding a challenge tile. In the second half, there is a 30% chance.

This is only a suggestion – you are more than welcome to design and balance the lane generation in any way you see fit. Your lanes must be random, and the two paths must have different generation rules, but beyond this is up to you. Read the Project 2 document carefully to ensure you meet all requirements before finalizing your game board class.

Any additional time you have can be dedicated to this week's homework, which is the Code Skeleton for your final project.

4 Homework: Code Skeleton

You will need to submit your code skeleton for your final project on Canvas for your homework this week. You should think about your code skeleton as an outline for how you will approach your project. You will need to determine how you are going to break down your code – an outline for the major sections of your main function and what types of objects and functions will help you make each part.

For your code skeleton you will need:

- For each class you create, your .h files should be complete with all the data members and member functions you will be using for each class.
- For the class implementation .cpp files, you should fully implement simple functions like your getters, setters, and constructors. For more complex functions, you can include the function prototype with detailed comments.
- For your game driver, you should have an approximate outline of how you will approach your main function's code. Pseudocode-like comments are fine for this. An example of expected comments is available in the code skeleton assignment on Canvas.

Week 15: Sorting Algorithms and Recursion

Learning Goals

This week you will:

1. Learn about pass by reference parameters
2. Learn about sorting algorithms and how to discuss the "speed" of an algorithm
3. Learn how to use recursive functions

1 Background

1.1 Pass By Reference Parameters

In C++, you can pass parameters to functions either by value or by reference. Passing by reference allows you to modify the original variable within the function.

The basic syntax is:

```
void functionName(<data_type> &parameter_name)
{
    // Function body
}
```

Example 1.1.1.

```
#include <iostream>
using namespace std;

// Function to increment a number by value
void incrementByValue(int num)
{
    num++;
}

// Function to increment a number by reference
void incrementByReference(int &num)
{
    num++;
}

int main()
{
```

```
    int x = 5;

    cout << "Before increment: " << x << endl;
    incrementByValue(x); // passing x by value
    cout << "After increment: " << x << endl;

    cout << "---" << endl;

    cout << "Before increment: " << x << endl;
    incrementByReference(x); // passing x by reference
    cout << "After increment: " << x << endl;

    return 0;
}
```

Sample Run 1.1.1

```
Before increment: 5
After increment: 5
---
Before increment: 5
After increment: 6
```

Key points to remember:

- Parameters passed by reference are indicated by & in the function declaration.
- Changes made to the reference parameter within the function affect the original variable.
- Pass by reference avoids unnecessary copying of the argument to the parameter variable, which can be more efficient.
- Arrays are passed by reference by default (so there is no preceding & for an array parameter).
- References cannot be reassigned to refer to a different variable once initialized.

1.2 Sorting Algorithms

There are several different strategies to sort unordered lists of data. They generally fall into two categories based on how long it takes for them to run proportionate to the number of items they are designed to sort.

When discussing how long an algorithm takes to run, we use something called "Big O Notation". To use Big O, you need to determine the size of the input, which is designated as n. The number of singular operations to execute your algorithm as a function of the input size defines how long it takes – for example if you have to look at every element in your list only once, you would need n operations. If you had to compare every element to every other element in your list, you would need n^2 operations. Big O then removes any constants or smaller terms, as we are only concerned with the limit of the behavior – i.e., what happens as your input gets exceedingly large.

When using Big-O, the two categories of sorting algorithms become clear: algorithms that generally either require that you compare every element against every other element, meaning they take $O(n^2)$ time, or algorithms that continuously break your data set in halves and compare these smaller pieces, which would take $O(n * log(n))$ time, as seen in Quick Sort.

Bubble Sort

Bubble Sort is generally considered the simplest sorting algorithm. It works by repeatedly swapping the adjacent elements if they are in the wrong order, only comparing two at a time.

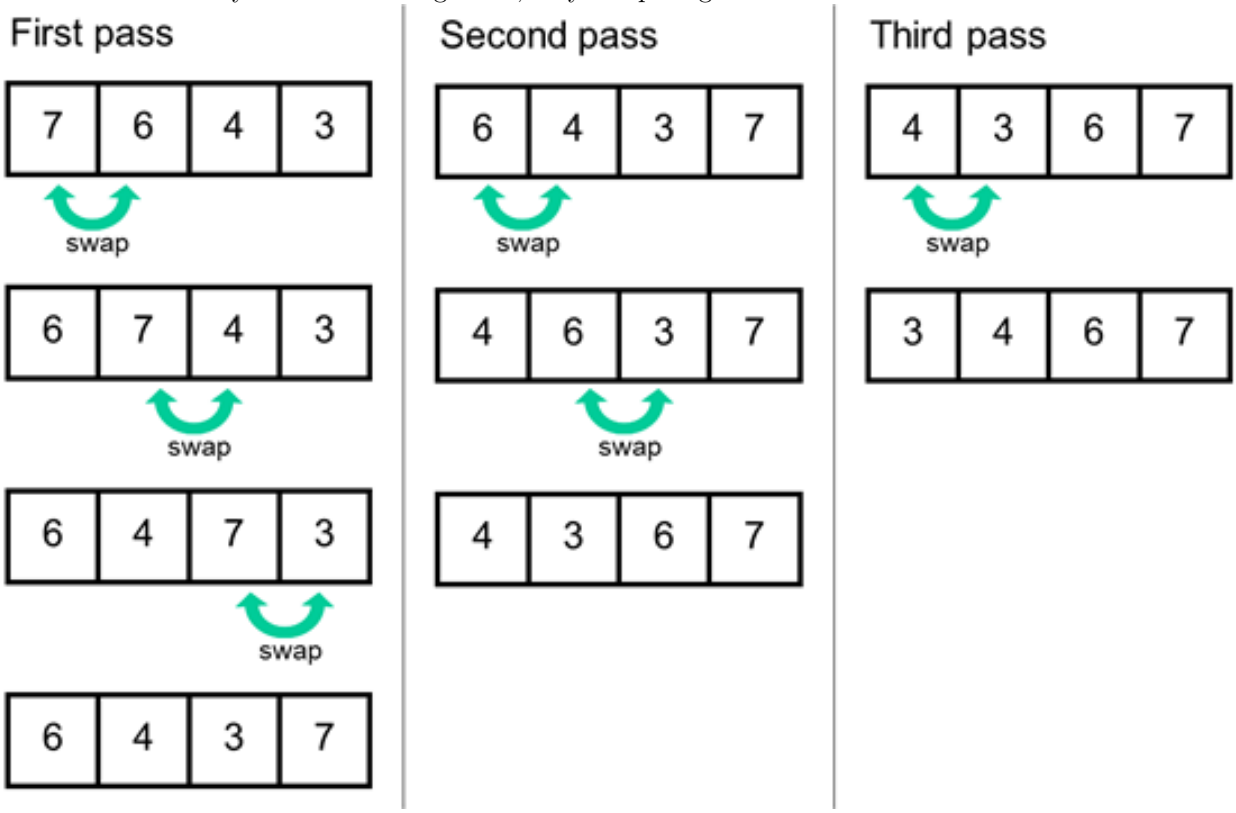


Image Source: https://www.computersciencebytes.com/sorting-algorithms/bubble-sort/

Bubble sort requires comparing n elements against n elements, and so its computational complexity described using Big-O is $O(n^2)$.

Selection Sort

Selection Sort involves picking the smallest (or largest) number out of the unsorted elements and then putting it in its own list, one after the other, so that you find the smallest, then the second smallest, and so on and so forth.

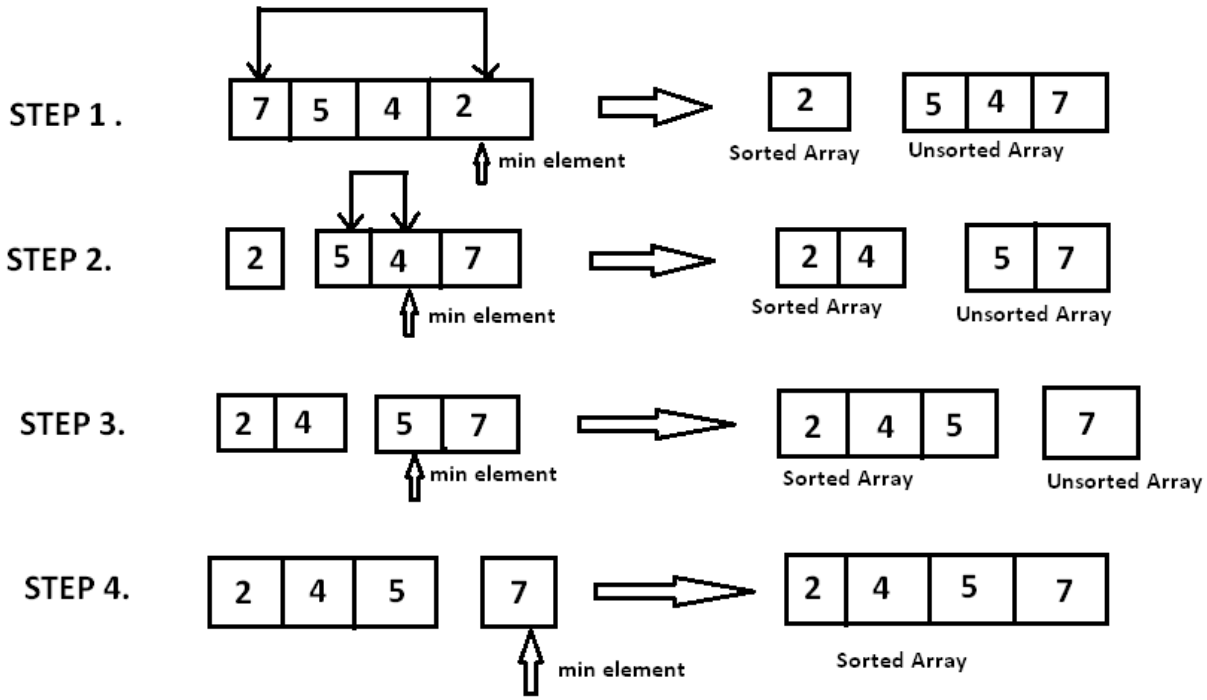


Image Source: https://www.hackerearth.com/practice/algorithms/sorting/selection-sort/tutorial/

For the first level of selection sort, you must search all n elements to find the smallest one. For the second level, you must search $n-1$ elements, then $n-2$, and so on and so forth. This will ultimately require $\frac{n^2}{2}$ operations. We do not worry about constants such as $\frac{1}{2}$ in Big-O, and so this is still a computational complexity of $O(n^2)$.

Below are four algorithms chosen to represent the different styles of sorting algorithms available to you.

Insertion Sort

Insertion sort works by splitting the list of elements into two pieces: one piece that has already been sorted, and one piece that has not been sorted yet. The algorithm goes through the unsorted list one element at a time and inserts it into the correct position in the sorted list until all elements have been sorted.

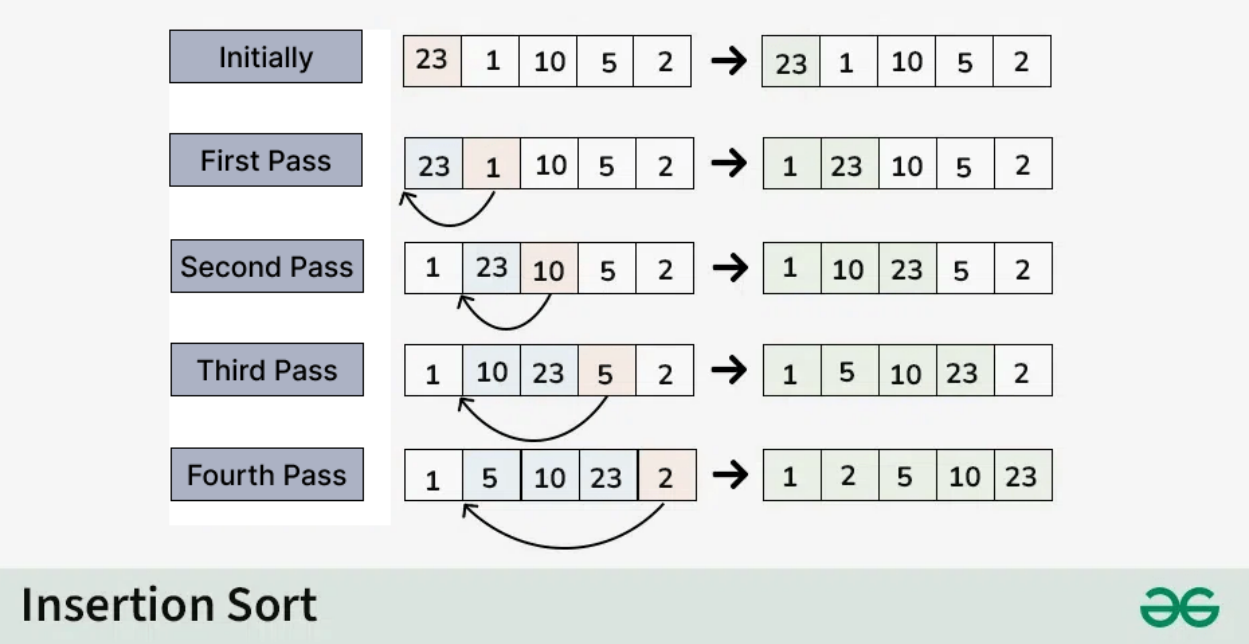
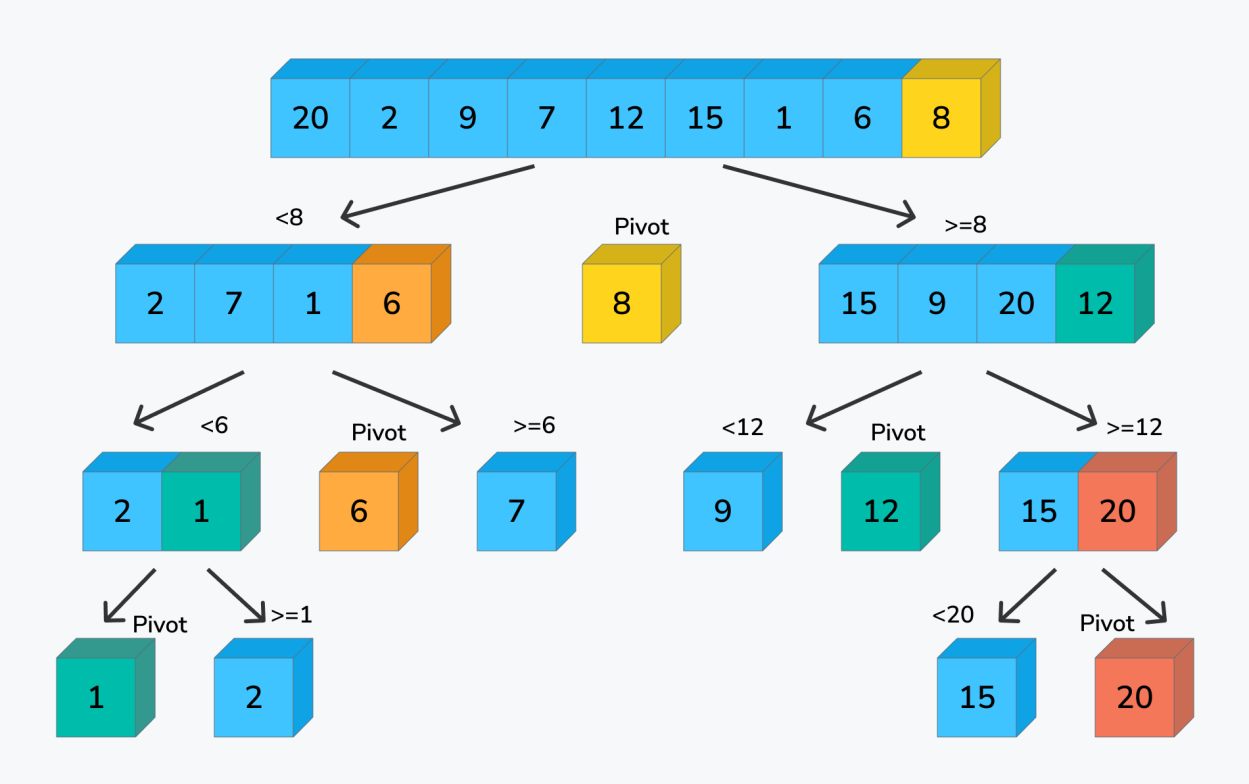


Image Source: https://www.geeksforgeeks.org/insertion-sort-algorithm/

The first element is already considered sorted. For the second element, you must compare against 1 element to decide where to put it; the third requires 2 comparisons; this pattern continues until the last comparison, which would be against $n-1$ elements. The formula $1+2+...+n-1$, when removing constants, also reduces to a Big-O of $O(n^2)$.

Quick Sort

Quick Sort is a little more complex, but it is faster than the previous algorithms, hence the name. Quick sort works by splitting our list around a randomly chosen "pivot" into two smaller lists for us to sort, where one list is smaller than the pivot and the other list is larger than the pivot. We then split those smaller lists around their own pivots again, resulting in four lists. This continues until we run out of elements in our lists.



As you can see, this sorting creates a tree-like structure. Each layer contains n elements and requires n operations comparing those elements against their respective pivots. However, the trick that makes this faster than the previous algorithms lies in the number of repetitions for this search: instead of requiring n separate comparisons for each element, as you have seen in the previous three sorting algorithms, you only need as many repetitions as there are layers in this tree. Since each iteration of the tree is splitting the number of elements in half, there are $log(n)$ layers, resulting in a total time complexity of $O(n * log(n))$.

1.3 Recursion

A recursive function is one which calls itself. Recursion can be used to accomplish a repetitive task, like loops. Indeed, it turns out that anything you can do with a loop, you could also do with recursion, and vice versa. However, some algorithms are easier to express with loops, and others are easier to express with recursion. You'll want both in your toolkit to write elegant, simplistic, short code.

Every recursive function includes two parts:

- **base case:** A simple non-recursive occurrence that can be solved directly. Sometimes, there are multiple base cases.
- **recursive case:** A more complex occurrence that can be described using smaller chunks of the problem, closer to the base case.

To write a recursive function, we often use the following format:

```
returnType functionName(arguments)
{
    if (/*baseCase?*/)
    {
        return /*baseCase result*/;
    }
    else
```

```
        {
            // some calculations, including a call to functionName
            // with "smaller" arguments.
            return /*general result*/
        }
    }
```

Consider the following simple recursive function, which calculates the sum of the numbers 1, 2, 3, ⋯, n:

```
int sumNumsRecursive(int num)
{
    // base case: if the number is 0, we will return 0
    if(num == 0)
    {
        return  0;
    }
    else
    {
        // recursive case: where we try to find the result of the previous step
        return num + sumNumsRecursive(num - 1);
    }
}
```

The following examples show the final returned value and intermediate recursive function calls.

For example, if `num = 3`. recursive call 1: `sumNumsRecursive(3)` will return `3 + sumNumsRecursive(2)` (we are running the else statement since num is 3). recursive call 2: When `sumNumsRecursive(2)` is called, it will return `2 + sumNumsRecursive(1)`. recursive call 3: Similarly, `sumNumsRecursive(1)` will `return 1 + sumNumsRecursive(0)`. recursive call 4: Finally, `sumNumsRecursive(0)` will return 0, by definition of the base case.

Next if we go back the chain and replace `sumNumsRecursive(0)` with 0, we will have `sumNumsRecursive(1) = 1 + 0`. Therefore, `sumNumsRecursive(1) = 1`. Similarly, `sumNumsRecursive(2) = 2 + sumNumsRecursive(1)`, where `sumNumsRecursive(1) = 1`. Therefore, `sumNumsRecursive(2) = 2 + 1`; thus, `sumNumsRecursive(2) = 3`. Lastly, `sumNumsRecursive(3) = 3 + 3`; the second 3 is the result of `sumNumsRecursive(2)`. Therefore, `sumNumsRecursive(3) = 6`.

Below is the same explanation in a different format.

```
sumNumsRecursive(3) = 3 + sumNumsRecursive(2) // running the else statement
                    = 3 + 2 + sumNumsRecursive(1)
                    // sumNumsRecursive(2) = 2 + sumNumsRecursive(1)
                    = 3 + 2 + 1 + sumNumsRecursive(0)
                    // sumNumsRecursive(1) = 1 + sumNumsRecursive(0)
                    = 3 + 2 + 1 + 0
                    // sumNumsRecursive(0) = 0 (base case)
                    = 6
```

2 Homework 9

Warning: You are not allowed to use global variables for this assignment.

This homework is extra credit, you will receive credit for the questions you have completed.

All function names, return types, and parameters must precisely match those shown. You may not use pass by reference or otherwise modify the function prototypes. You are welcome to create additional functions that may help streamline your code.

2.1 Carbon Emission Reduction

This question may require the use of recursion, logical and relational operators, if-else statements, declaring and calling a function, and processing user input. Note: you are not allowed to use loops for this question.

You are tasked with estimating the future reduction in your organization's carbon emissions. Based on historical data, the reduction of carbon emissions for the next year can be predicted from the current year using the following equation:

$$\text{next\_year\_reduction} = \text{current\_reduction} + 0.5 \times \text{current\_reduction} + 10$$

Develop a recursive function that takes the current year's carbon emission reduction and the number of years forward for which the reduction needs to be predicted and returns the estimated reduction.

Function: `emissionReduction(double, int)`	`double emissionReduction(double current_reduction, int years)`
Purpose:	To estimate future reductions in carbon emissions based on current reduction efforts and projected yearly improvements.
Parameters:	**double** `current_reduction` - The current year's reduction in carbon emissions. **int** `year` - number of years forward
Return Value:	Returns the estimated reduction in carbon emissions after the specified number of years.
Error Handling:	- If `current_reduction` and/or `year` is negative, -1 is returned. - If `year` is equal to 0, return original `current_reduction` - It should not print anything.

Example:	Note: This is only an example of the function; you need to develop your own main function to fulfill the requirement for this problem.
	Sample Code:
	Example 2.1.1.
	`// Assume the proper libraries are included.`
	`// Assume the proper implementation of emissionReduction() is included.`
	`int main()`
	`{`
	`  cout << "Predicted reduction in carbon emissions: " << emissionReduction(100, 3) << endl;`
	`  return 0;`
	`}`
	Sample Run 2.1.1
	`Predicted reduction in carbon emissions: 385`

For Question 1, develop and validate your solution in VS Code. Once you are happy with your solution, go to coderunner on Canvas and paste emissionReduction() and any helper function(s) to the answer box!

Sample Run 2.1.2

```
Inputs:
Current reduction: 200
```

```
Years: 5
Output:
Predicted reduction in carbon emissions: 444.204
Explanation:
emissionReduction(200, 0) = 200
emissionReduction(200, 1) = 200 + 0.5 * 200 + 10 = 310
emissionReduction(200, 2) = 310 + 0.5 * 310 + 10 = 475
emissionReduction(200, 3) = 475 + 0.5 * 475 + 10 = 722.5
emissionReduction(200, 4) = 722.5 + 0.5 *722.5 + 10 = 1093.75
emissionReduction(200, 5) = 1093.75 + 0.5 * 1093.75+ 10 = 1650.62
```

2.2 Vector Shuffle

This question may require the use of functions, vectors, and loops.

Your task is to write a C++ function that takes the elements of two vectors and returns a vector of integer values that combines the two input vectors by alternating between values from each of the two vectors. If the vectors are of different sizes, the remaining elements from the longer vector should be appended to the result vector.

Function: `vectorShuffle(vector<int>, vector<int>)`	`vector<int> vectorShuffle(vector<int> vec1, vector<int> vec2)`
Purpose:	To shuffle the elements of two vectors and return a new vector containing elements from both vectors in an interleaved fashion.
Parameters:	**vector<int>** `vec1` - The first vector of integers **vector<int>** `vec2` - The second vector of integers
Return Value:	- **vector<int>** Returns a new vector containing elements from both `vec1` and `vec2` - The function should not print anything.
Error Handling:	- If one vector is shorter than the other (including size 0), alternate as long as you can and append the remaining elements from the longer vector. - If both vectors are empty (size 0), the return value should be an empty vector. - The first element of the returned vector should come from the first input argument.

Example:	Note: This is only an example of the function; you need to develop your own main function to fulfill the requirement for this problem. Sample Code:

Example 2.2.1.

```
// Assume the proper libraries are included.

// Assume the proper implementation of vectorShuffle() is
↪  included.

int main()
{
    vector<int> vec1 = {1, 3, 5};
    vector<int> vec2 = {2, 4, 6, 8, 10};
    vector<int> result = vectorShuffle(vec1, vec2);
    if (result.size() == 0)
    {
        cout << "No elements found" << endl;
    }
    else
    {
        cout << "Shuffled vector:" << endl;
        for (int i = 0; i < int(result.size()); i++)
        {
            cout << result.at(i) << " ";
        }
        cout << endl;
    }
    return 0;
}
```

Sample Run 2.2.1

```
Shuffled vector:
1 2 3 4 5 6 8 10
```

For Question 2, develop and validate your solution in VS Code. Once you are happy with your solution, go to coderunner on Canvas and paste the vectorShuffle() and any helper function(s) into the answer box!!

Sample Run 2.2.2

```
Inputs:
vector<int> vec1;
vector<int> vec2 = 2, 4, 6, 8, 10;
Output:
Shuffled vector:
2 4 6 8 10
```

Sample Run 2.2.3

```
Inputs:
vector<int> vec1;
vector<int> vec2;
Output:
No elements found
```

Sample Run 2.2.4

```
Inputs:
vector<int> vec1 = 1, 1, 3;
vector<int> vec2 = 2, 4, 6, 8, 10;
Output:
Shuffled vector:
1 2 1 4 3 6 8 10
```

2.3 The Magical List

This question may require the use of functions, vectors, and loops.

Alex is a curious programmer who loves solving puzzles. One day, while tinkering with numbers, Alex decides to create a magical list that follows a set of peculiar rules. Here's how the game works:

Alex starts with an empty list. They ask a friend to keep entering positive integers, one at a time. The program will continuously prompt the user with "Please enter a number:" until -1 is entered and each number influences the list in some way:

1. If the list is empty, Alex adds the number to the list immediately.
2. If the number is divisible by both 3 and 5, Alex does both: remove the first and last numbers from the list (if there is a number).
3. If the number is divisible by 5, Alex removes the first number from the list (if there is one).
4. If the number is divisible by 3, Alex removes the last number from the list (if there is one).
5. If none of the above applies, Alex simply adds the number to the end of the list.

However, there's a catch! If someone enters a negative number that isn't `-1`, or zero, Alex scolds them with: **"The number should be a positive integer or -1."**

When someone finally enters `-1`, Alex stops the game. Then, Alex proudly displays the final list of numbers, separated by spaces. Note: the printing will happen in main().

Function: `processMagicalList()`	`vector<int> processMagicalList()`
Purpose:	To process a series of user inputs according to the rules and return the final list of integers.
Parameters:	None. All inputs are provided by the user interactively during execution.
Return Value:	**vector<int>** Returns a vector containing the final list of numbers after applying the rules.

Error Handling:	- The program should prompt the user with "Please enter a number:" for each input. - If the user enters a non-positive number (excluding `-1`), display `"The number should be a positive integer or -1."`. - The function should not print the vector. Printing should be handled by the main function.
Example:	Note: This is only an example of the function; you need to develop your own main function to fulfill the requirement for this problem. Sample Code:

Example 2.3.1.

```
// Assume the proper libraries are included.

// Assume the proper implementation of
↪  processMagicalList() is included.

int main()
{
    vector<int> result = processMagicalList();
    if (result.size() == 0)
    {
        cout << "The vector is empty." << endl;
    }
    else
    {
        cout << "The elements in the vector are: ";
        for (int i = 0; i < int(result.size()); i++)
        {
            cout << result[i] << " ";
        }
        cout << endl;
    }
    return 0;
}
```

Sample Run 2.3.1

```
The elements in the vector are:
8
```

For Question 3, develop and validate your solution in VS Code. Once you are happy with your solution, go to coderunner on Canvas and paste the processMagicalList() function into the answer box!

Sample Run 2.3.2

```
Inputs:
3, 5, 15, 8, -1
Output:
```

```
Please enter a number:
Please enter a number:
Please enter a number:
Please enter a number:
Please enter a number:
The elements in the vector are:
8
```
Explanation:

1. **3:** The list is empty, so 3 is added to the list.
Current list: [3]

2. **5:** The number is divisible by 5. The first number, 3, is removed from the list.
Current list: [] (empty)

3. **15:** The number is divisible by both 3 and 5. Since the list is empty, nothing is removed. 15 is added to the list.
Current list: [15]

4. **8:** The list is not empty, so 8 is added to the end of the list.
Current list: [15, 8]

5. **-1:** The game ends. The list contains numbers, so the program prints the elements: ``The elements in the vector are: 8''.
Final list: [8]

Sample Run 2.3.3

Inputs:
```
0 -5 6 -1
```
Output: Please enter a number:
```
The number should be a positive integer or -1.
Please enter a number:
The number should be a positive integer or -1.
Please enter a number:
Please enter a number:
The elements in the vector are:
6
```

Sample Run 2.3.4

Inputs:
```
10, 9, 4, -1
```
Output:
```
Please enter a number:
Please enter a number:
Please enter a number:
Please enter a number:
The elements in the vector are:
4
```
Explanation:

1. **10:** The number is divisible by 5. The first number in the list is removed. Since the list is empty, nothing happens.

```
Current list: [] (empty)

2. 9: The number is divisible by 3. The last number in the list is removed. Since the list
is empty, nothing happens.
Current list: [] (empty)

3. 4: The number is not divisible by 3 or 5, so it is added to the end of the list.
Current list: [4]

4. -1: The game ends. The program prints: ``The elements in the vector are: 4''.
Final list: [4]
```

2.4 T Ball Players

This question may require the use of arrays, logical and relational operators, if-else statements, declaring and calling a function.

Coach Prime is getting ready for the new T-ball season and needed to finalize his roster. The team is open to kids between the ages of 4 and 6, and parents had already sent in names and ages of their children.

Write a function, printEligiblePlayers, to help Coach Prime filter out the eligible players from the list of names and ages. Your function should identify players between 4 and 6 years old (inclusive) and print their names along with their ages, so Prime Kelly can focus on training her future stars.

Function Specification :

Function: `printEligiblePlayers(string, double, int)`	`void printEligiblePlayers(string names[], double ages[], int num)`
Purpose:	To identify players between 4 and 6 years old (inclusive) and print their names along with their ages
Parameters:	**string** `names[]` - an array of strings to hold names of players **double** `ages[]` - an array of floating point numbers (as doubles) to hold ages of players **int** `num` - an integer that gives the number of elements in the players and ages arrays
Return Value:	The function does not return anything
Error Handling:	- The size of names and ages will be the same represented by num.

Example:	Note: This is only an example of the function; you need to develop your own main function to fulfill the requirement for this problem. Sample Code:

Example 2.4.1.

```
// Assume the proper libraries are included.
// Assume the proper implementation of
↪  printEligiblePlayers() is included.

int main()
{
    string names[4] = {"Joe", "Jack", "Amy", "Bob"};
    double ages[4] = {4.0, 5.6, 6.0, 4.2};
    printEligiblePlayers(names, ages, 4);
}
```

Sample Run 2.4.1

```
Joe 4
Jack 5.6
Amy 6
Bob 4.2
```

For Question 4, develop and validate your solution in VS Code. Once you are happy with your solution, go to coderunner on Canvas and paste printEligiblePlayers() and any helper function(s) to the answer box!

Sample Run 2.4.2

```
Inputs:
string names[4] = "Joe", "Jack", "Amy", "Bob";
double ages[4] = 3.4, 5.6, 6.4, 4.2;
printEligiblePlayers(names, ages, 4);

Output:
Jack 5.6
Bob 4.2
```

2.5 Music App

For this question, you'll develop a basic Music App System by creating a Library class to manage a collection of songs. For this question, you will create a Song class.

The **Song** class comprises the following attributes:

Data members (private)

Member Type	Member Name	Description
`string`	`_title`	The name of the song
`string`	`_artist`	The name of the song who sang the song

`int`	`_downloads`	A count of the number of times the song was download

Member Functions (public)

Function	Description
`Default constructor`	Creates a new instance of `Song` by setting `_title` to an empty string, `_artist` to an empty string and `_downloads` to 0.
`Song(string)`	Creates a new instance of `Song` by setting `_title` as the string parameter, `_artist` to an empty string and `_downloads` to 0.
`Song(string, string, int)`	Creates a new instance of `Song` by setting `_title` as the first string parameter, `_artist` as the second string parameter and `_downloads` to the integer parament. See Function Specification table below for more details.
`string getTitle()`	Returns the `_title` of the `Song`.
`string getArtist();`	Returns the `_artist` of the `Song`.
`int getDownloads();`	Returns the number of `_downloads` of the `Song`.
`void setTitle(string)`	Sets the `_title` to the value of the provided string parameter.
`void setArtist(string)`	Sets the `_artist` to the value of the provided string parameter.
`void setDownloads(int)`	Sets the `_downloads` to the value of the provided int parameter.
`double grossRevenue(double)`	Calculates the revenue generated by the `Song` by taking in a price per download and returning the total revenue. Formula: revenue = price $\times$ *_downloads*. See the Function Specification table below for more details.

Function Specifications:

Function: `Song(string, string, int)`	`Song(string title, string artist, int downloads)`
Purpose:	This parameterized constructor creates a new instance of the `Song` class. - Sets `_title` to `title`. - Sets `_artist` to `artist` Sets `_downloads` to `downloads`
Parameters:	`string title` - The title of the Song. `string artist` - The artist who sang the Song. `int downloads` - Number of times the songs has been downloaded.
Return Value:	N/A
Error Handling:	- If the `downloads` count is non-positive, set _download to 0.

Example:	**Example 2.5.1.**
	`// Assume the proper libraries are included`
	`// Assume the proper implementation of the class is included`
	`int main() { string title = "Aint Got It Like That"; string artist = "Earl St. Clair"; int downloads = 5; Song new_song = Song(title, artist, downloads); }`
	
	Expected Contents of `new_song` Object:
	`_title` = "Aint Got It Like That"
	`_artist` = "Earl St. Clair"
	`_downloads` = 5

Function: `double grossRevenue(double)`	`double grossRevenue(double price)`
Purpose:	Calculates the revenue generated by the `Song` using the formula: revenue = price x _downloads.
Parameters:	`int price` - The price per download of a song.
Return Value:	`double` : Returns a double representing the revenue generated from the `Song` downloads.
Error Handling:	- If the `price` amount is non-positive, do not add or modify any contents, and return -1.

Example:

Example 2.5.2.

```
// Assume the proper libraries are included
// Assume the proper implementation of the Library class
↪ is included

int main() {
    string _title = "Get Schwifty";
    string _artist = "Rick Sanchez";
    int _downloads = 1000;
    double price = 1.1;
    Song new_song = Song(_title, _artist, _downloads);

    cout << fixed<< setprecision(2) <<
    ↪ new_song.grossRevenue(price) << endl;
}
```

Expected Output:

The expected output after the function call:
1100.00

For Question 4, develop and validate your solution in VS Code. Once you are happy with your solution, go to coderunner and paste the Song Class header (.h file) and all the implementations (.cpp file). Do not include your driver file (main())

2.6 College Admission

This question may require the use of file streams, logical and relational operators, if-else statements, declaring and calling a function, and processing user input.

You are part of a college admission board and have been requested to write a program to recommend prospective students for admission based on specific criteria using data from a text file. Each line in the file contains student data in the format:

`Student Name, SAT, GPA, Interest, High School Quality, Sem1, Sem2, Sem3, Sem4.`

1. **Overall Score Calculation**

   Use the following formula to calculate each student' s score:

$$\text{Score} = (0.4 \times \text{GPA}) + (0.3 \times \frac{\text{SAT}}{1600}) + (0.2 \times \text{High School Quality}) + (0.1 \times \text{Interest})$$

   Students with a score of **2.5 or higher** are selected under the "score" criteria.

2. **Outlier Detection**

   Flag students as "outliers" if the interest is 0 and they have a score of 1.5 or higher.

3. **Grade Improvement Detection**

   Students are flagged for "grade improvement" if their semester grades show a consistent increase (i.e., $Sem1 < Sem2 < Sem3 < Sem4$) and they have a score of **1.5 or higher**.

4. **Output**

   For each student, print their name, score (to 6 decimal places), and the criteria ("score", "outlier", or "grade improvement") if applicable. If no criteria are met, print just the name and score.

Required Functions :

- Use a predefined `split()` function available on Canvas to parse the file.
- Implement the `processAdmission()` function to handle calculations and criteria checking.
- Hint: you may use the to_string function to convert numbers (int, double) into strings. For example, `string example = to_string(1.0);`.

Function Specification :

Function: `string processAdmission(string)`	`string processAdmission(string line)`
Purpose:	To analyze a student's data, calculate their admission score, and determine whether they meet the criteria for admission based on score, outlier, or grade improvement.
Parameters:	**string** `line` - A single line of comma-separated student data containing the student' s name, SAT, GPA, interest, high school quality, and semester grades. Note: we have skipped the header in main(); therefore, you may assume that all `line` that are passed can be parsed.
Return Value:	Returns a string formatted as: `Name,Score,[criteria]`
Error Handling:	- Flag students as "outliers" if Interest = 0 and the score is 1.5 or higher. - Flag students for "grade improvement" if the grade of each semester is higher than the previous semester and the student has a score of 1.5 or higher. - Assumes valid input format for the line. The function does not handle cases where the line is improperly formatted or contains invalid data types. - The function should not print anything.

Example:

Note: This is only an example of the function; you need to develop your own main function to fulfill the requirement for this problem.
Sample Code:

Example 2.6.1.

```
// Assume the proper libraries are included.
// Assume the proper implementation of processAdmission()
↪  and split() is included.

int main(){
    string fileName;
    string line;

    cout<<"Enter the file name:"<<endl;
    cin>>fileName;

    ifstream file(fileName);
    if(file.is_open()){
        cout<<endl<<"Results:"<<endl;
        getline(file, line); // Skip header
        while (getline(file, line)){
            cout << processAdmission(line) << endl;
        }
    } else {
        cout<<"Could not open file."<<endl;
    }

    return 0;
}
```

Sample Run 2.6.1

```
Input: text8.txt
File Content:

Student,SAT,GPA,Interest,High School Quality,Sem1,Sem2,Sem3,Sem4
Anne Mutant,1550,3.89,0,9,97,87,97,87
Ayla Ranefer,1370,4,3,7,85,86,91,95

Output:

Results:
Anne Mutant,3.646625,score
Ayla Ranefer,3.556875,score
```

For Question 6, develop and validate your solution in VS Code. Once you are happy with your solution, go to coderunner on Canvas and paste processAdmission(), split() and any helper function(s) to the answer box!

Sample Run 2.6.2

Input: file-does-not-exist.txt
File Content: //file does not exist

Output:

```
Could not open file.
```

Sample Run 2.6.3

Input: text.txt
File Content:

```
Student,SAT,GPA,Interest,High School Quality,Sem1,Sem2,Sem3,Sem4
Abbess Horror,1300,3.61,10,7,95,86,91,94
Anastasia Kravinoff,1500,3.74,0,0,92,86,81,90
Adelicia von Krupp,900,4,5,2,88,92,83,72
Anna Frankenstein,1050,2.42,5,4,90,91,93,98
```

Output:

```
Results:
Abbess Horror,4.087750,score
Anastasia Kravinoff,1.777250,outlier
Adelicia von Krupp,2.668750,score
Anna Frankenstein,2.464875,grade improvement
```

Appendix A: Software Installation Guide

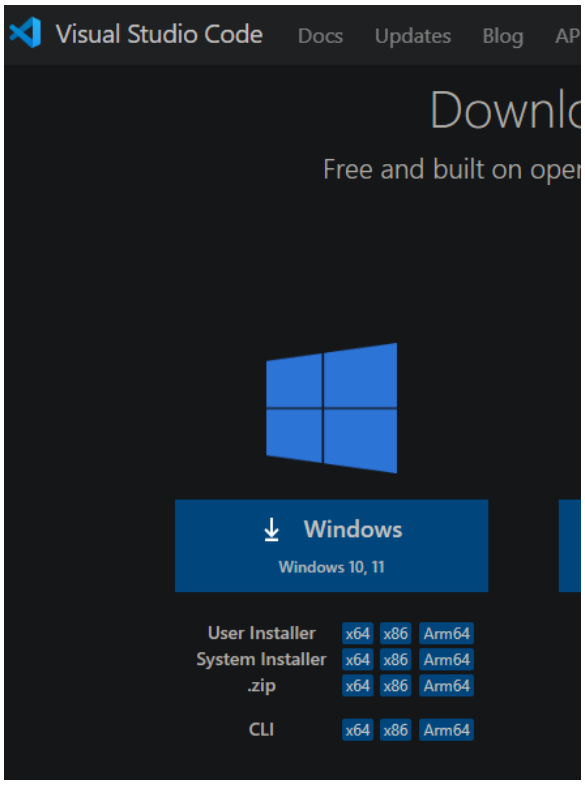
1 VS Code Set Up

1.1 Windows Installation

Important: Before proceeding with this document, make sure that you have run Windows Update within your Windows 10 or 11 environment. You must have the latest updates installed.

Step 1: Install VS Code

- Go to the VS code download page, and download for Windows.



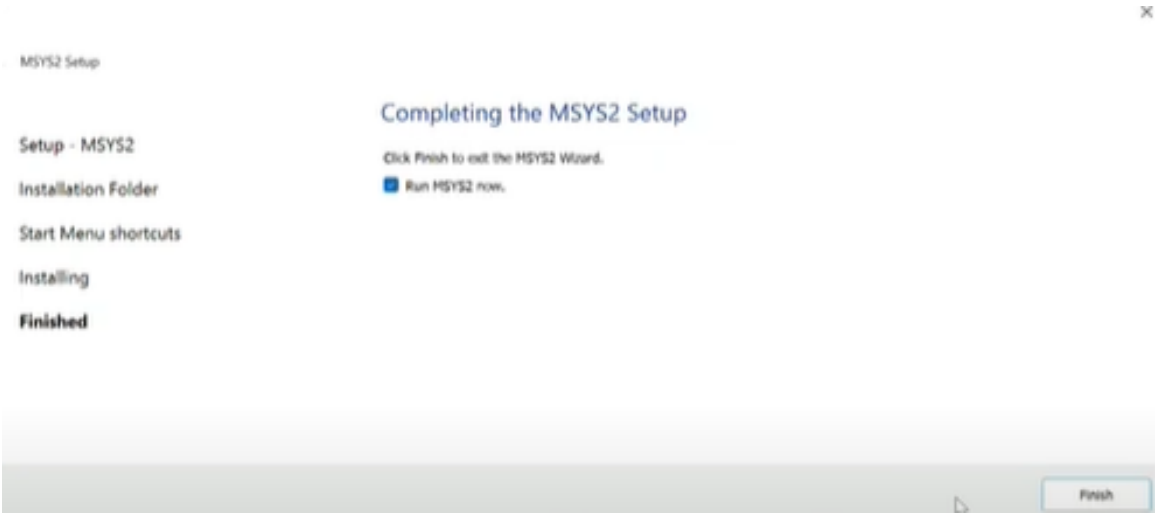
- Run the installer and accept all of the default settings.
- Click on Install and wait for Visual Studio Code to finish installing, then close the installer.

Step 2: Installing MinGW

This section is based on this guide from Microsoft: https://code.visualstudio.com/docs/cpp/config-mingw

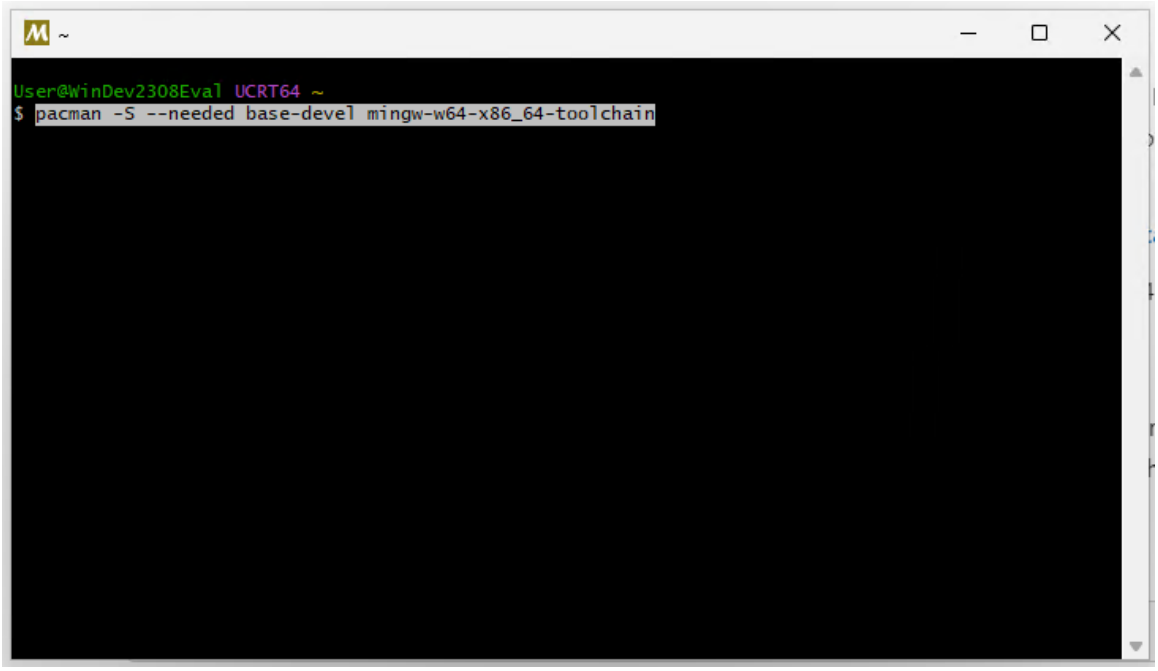
MinGW is a Windows C/C++ compiler tool set that will allow us to compile our C/C++ code into a .exe file.

- First Install MinGW from this link
- Open the installer and choose the Defaults for all settings.
- At the end of the installation run msys2 and then run the following command:



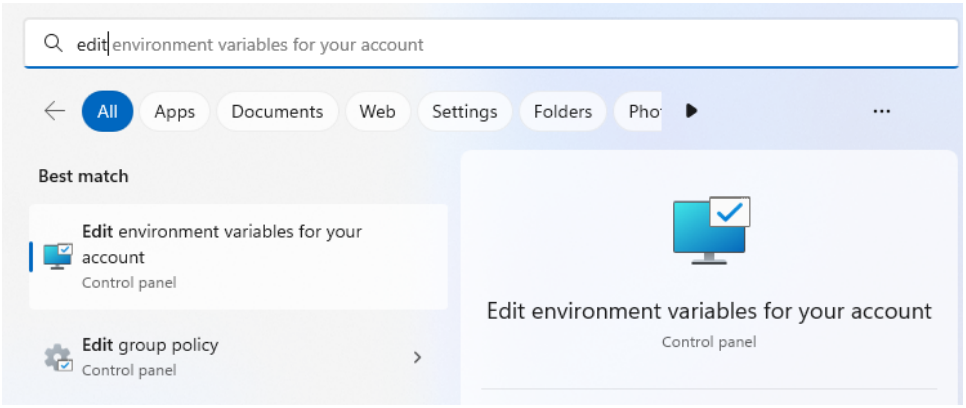
(Shift + Insert is the paste shortcut in MSYS2' s terminal)

```
pacman -S --needed base-devel mingw-w64-x86_64-toolchain
```



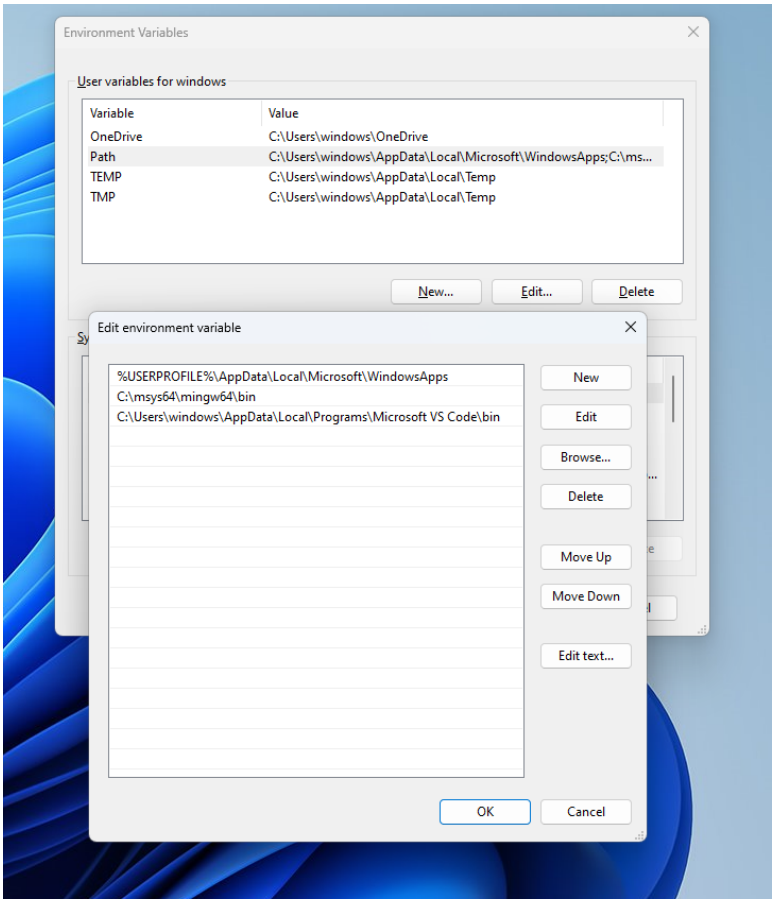
- Press enter when prompted to install all of the default packages, then press Y to confirm the install. This will take 1 to 5 minutes to finish. Once the install completes you can close msys2.

- Now we need to add msys2 to window' s PATH variable. Press the Windows key and begin typing "Edit environment variables for your account" until you see this option.



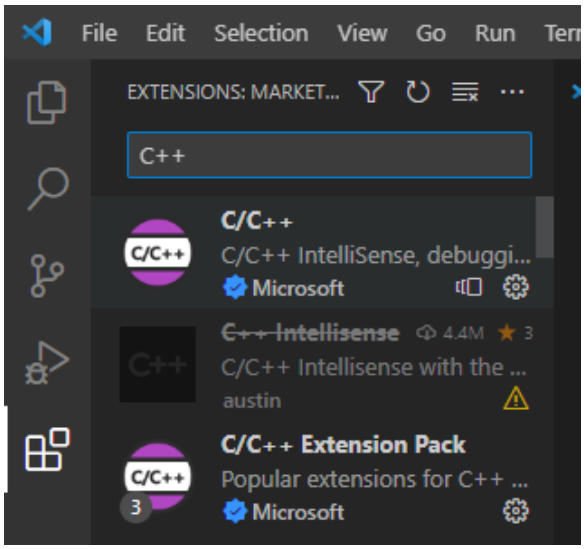
Now Select the "PATH" variable and click edit, in the window that opens click "New" and enter the following path for the default installation location of Msys2.

```
C:\msys64\mingw64\bin
```

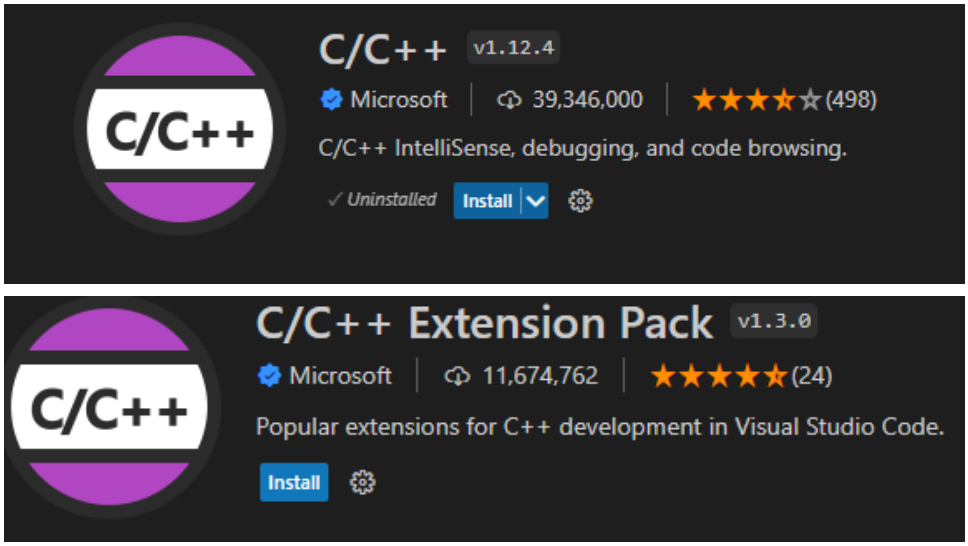


Step 3: Adding VS code extensions

- After you Reboot open VScode and select the extensions tab. (5th from the top), and search for "C++". We need to install the "C/C++" and "C/C++ Extension Pack" both from Microsoft.



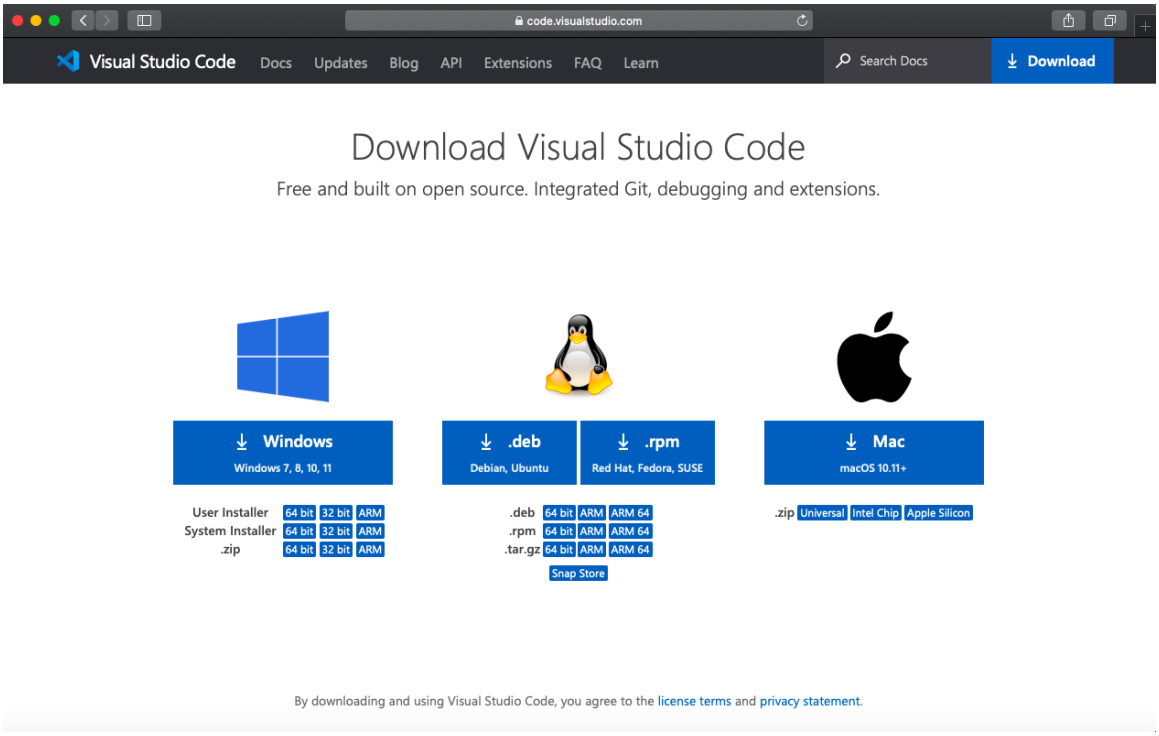
- Select the extension then click on install, these will provide Syntax Highlighting and other useful tools when working in C++



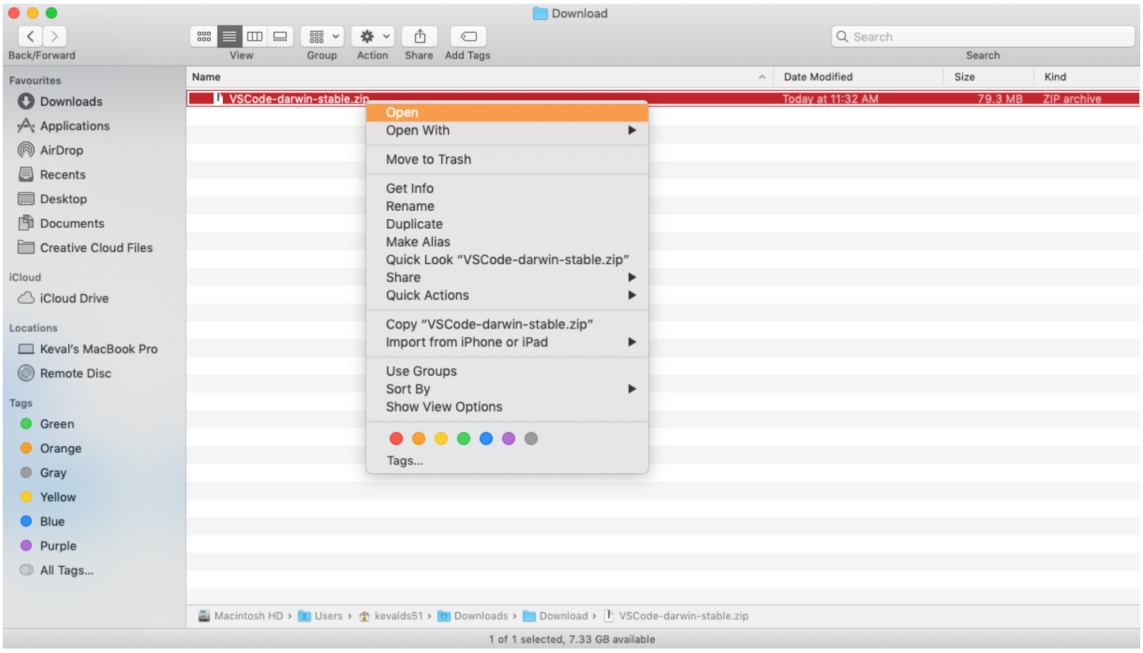
1.2 Mac Installation

Step 1: Installing VS Code

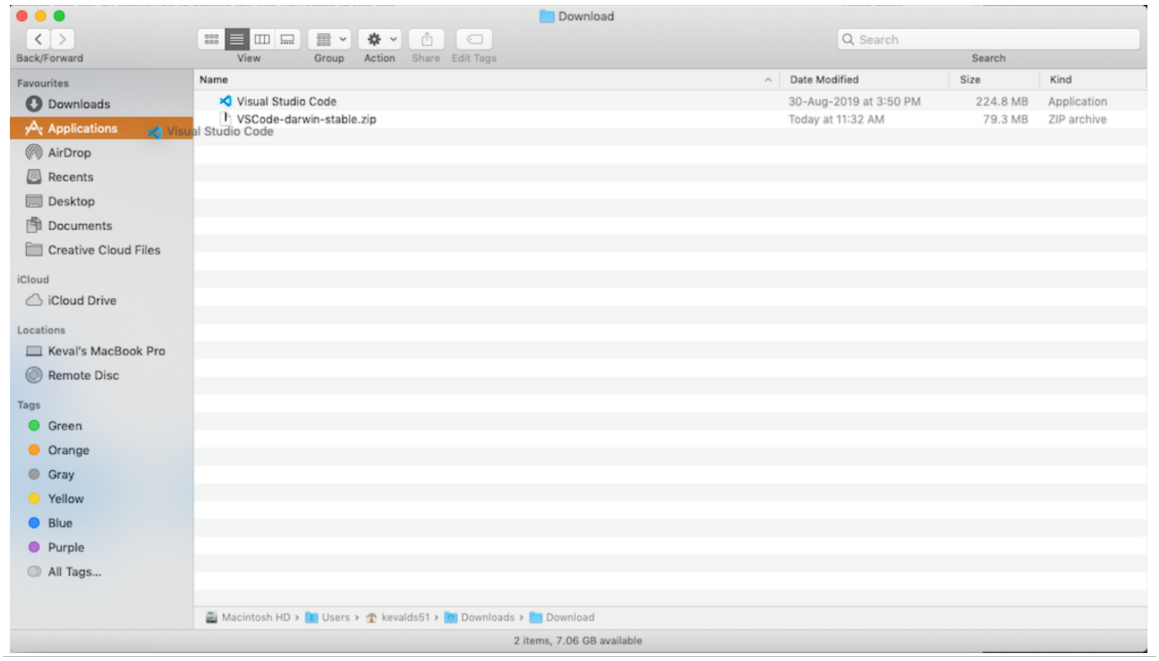
- Go to https://code.visualstudio.com/Download , and download for Mac.



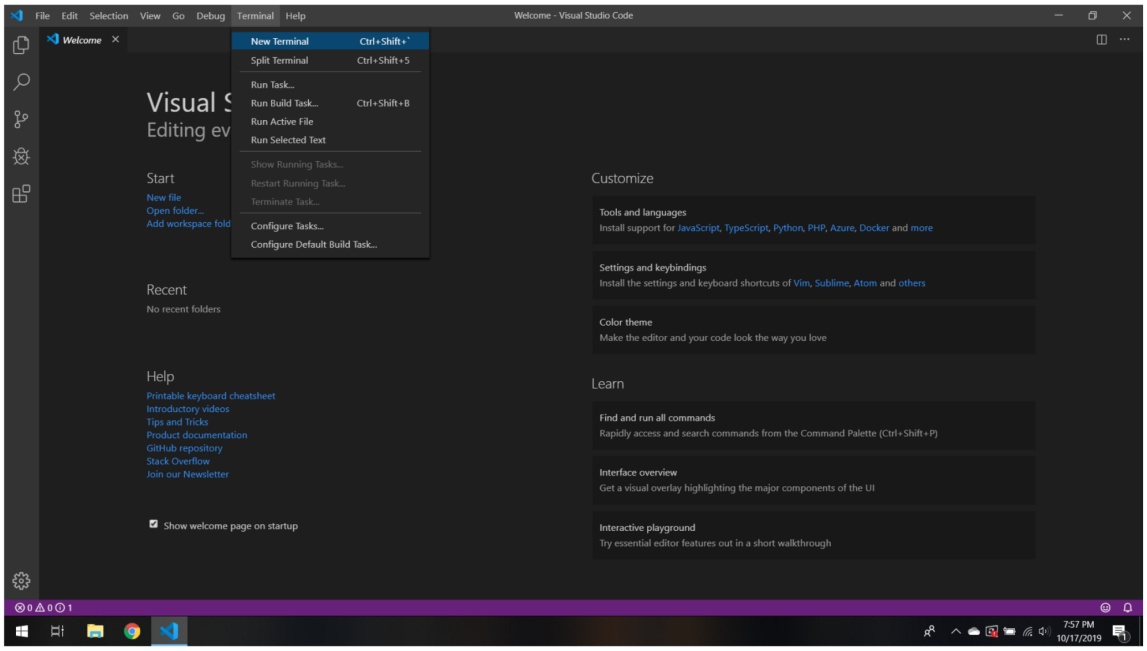
- After the download has finished, unzip the folder by double-clicking on it.



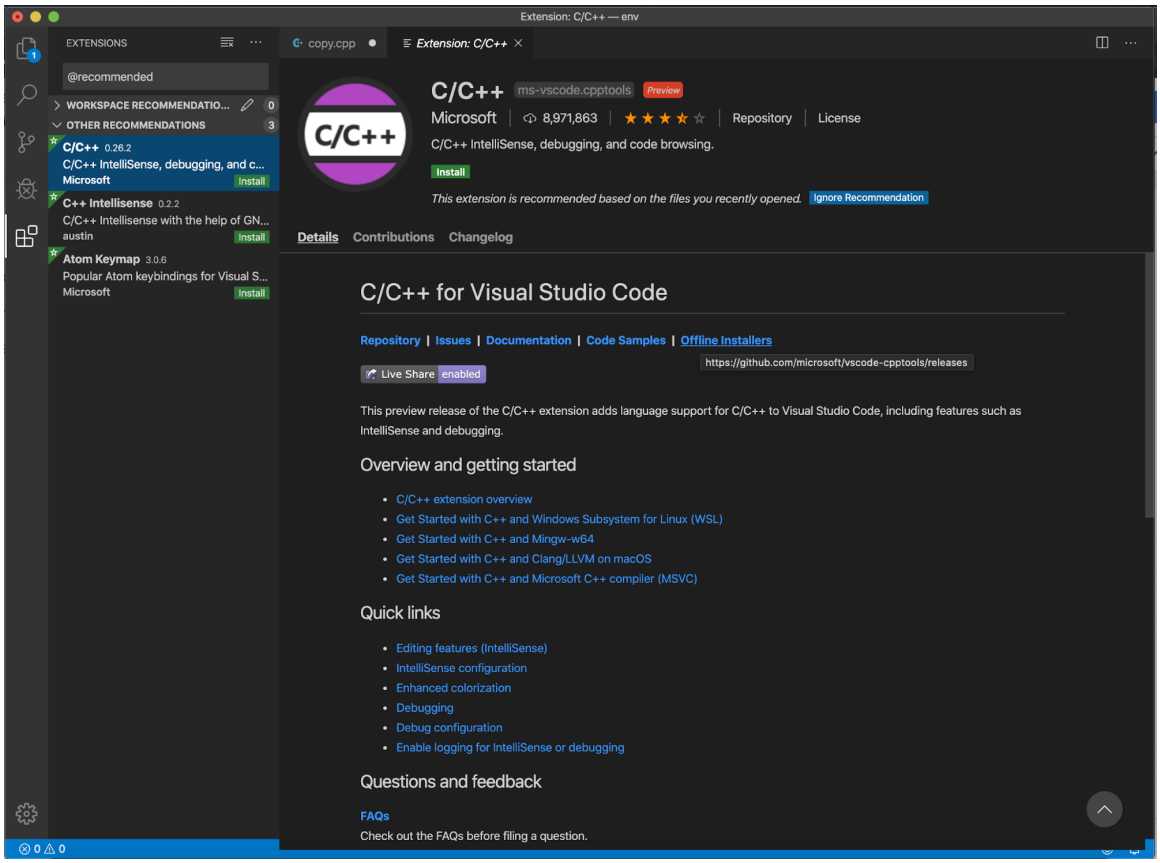
- Now you can see the "Visual Studio Code" application. Drag and drop this icon to the "Applications" folder of your computer.



- Double click on the ” Visual Studio Code” icon to launch the application. (You might need to right click and select “open” if you cannot launch the program). Next, select the ” New Terminal” option to open the terminal window.

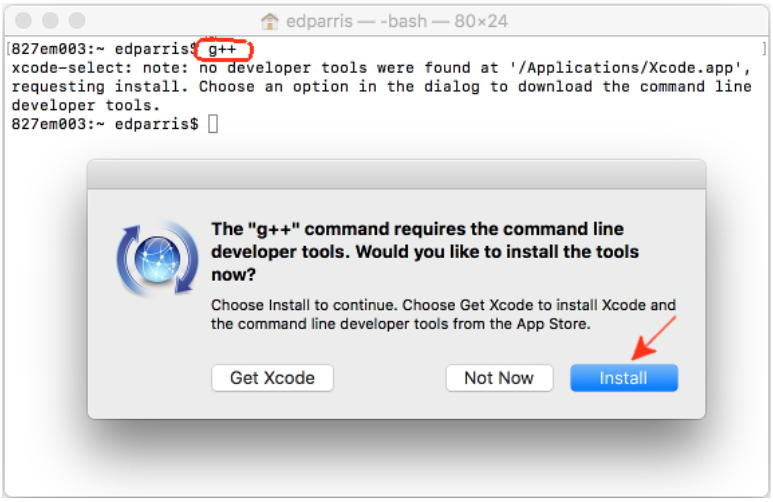


- Install C/C++ extension. In the toolbar on the left hand side of the screen click on the bottom icon for Extensions. Search for C/C++ and click install.



Step 2: Install g++

- Open a Terminal window.(One way is to press Command+Space, type Terminal in the search field, and press the Return key.)
- In the Terminal window type `g++` and press the Return key. We will see an alert box like this:

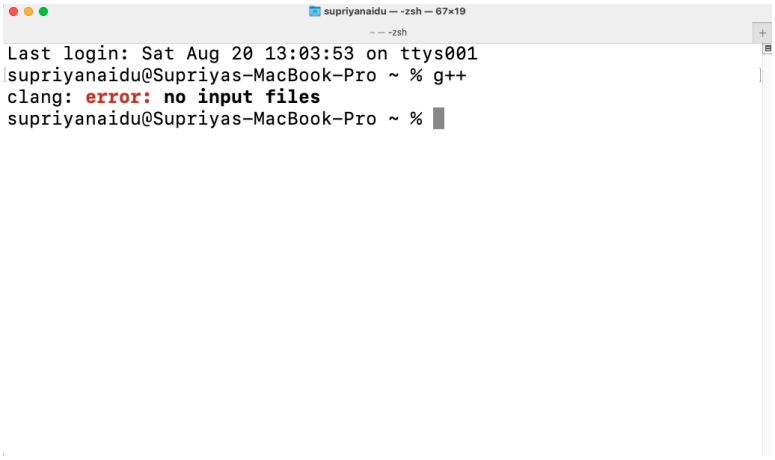


- Choose Install to get only the command line tools unless you want to learn Xcode.

  Xcode can be installed later from the App Store.

  After installation, type g++ in the Terminal, press the Return key, and verify the terminal prints the message, "no input files".

```
$ g++
clang: error: no input files
```



2 Debugger Installation

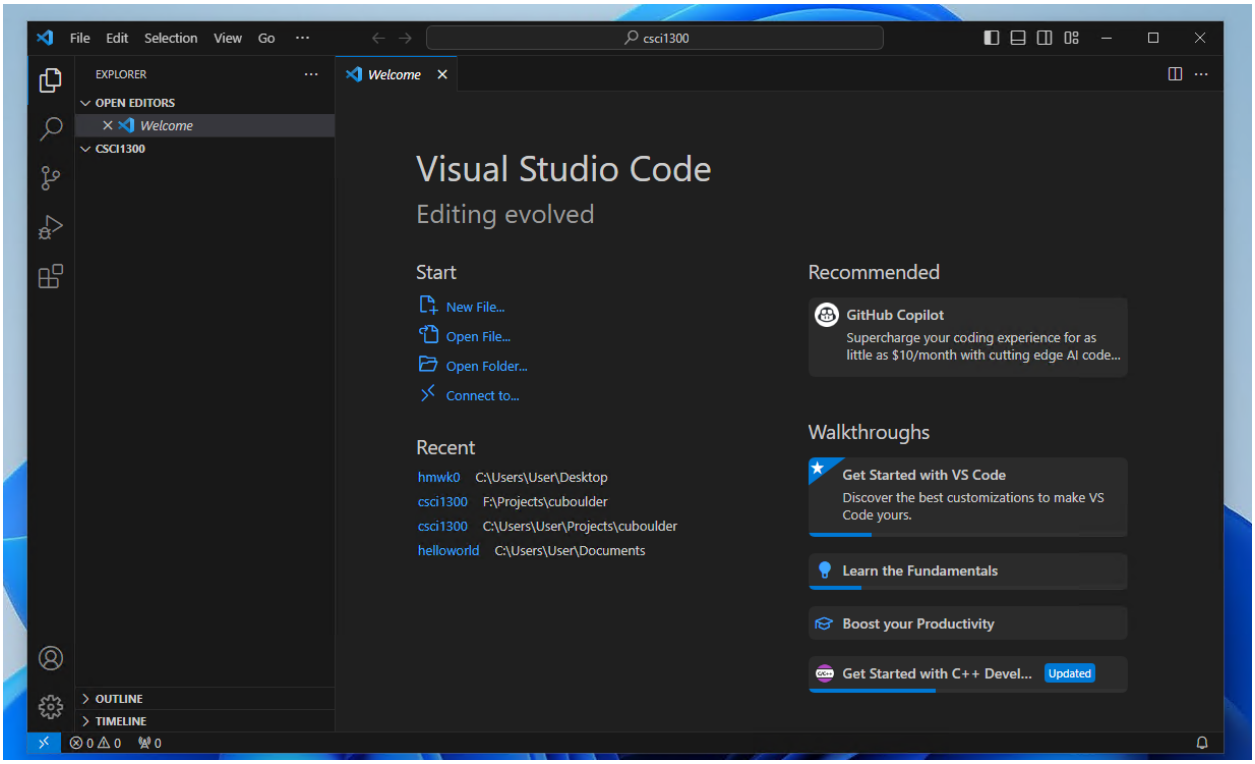
A debugger is an incredibly powerful tool you can use to help find issues with your code and fix them. It allows you to do step-by-step examination of your code, inspecting variable values at different points, and understanding how your code actually runs when it is executed.

2.1 Windows Installation

Windows users will use a debugger called gdb to debug their C++ code. This program has been installed with your compiler in homework0. The steps below will demonstrate the process of setting up the debugger in VSCode on Windows.

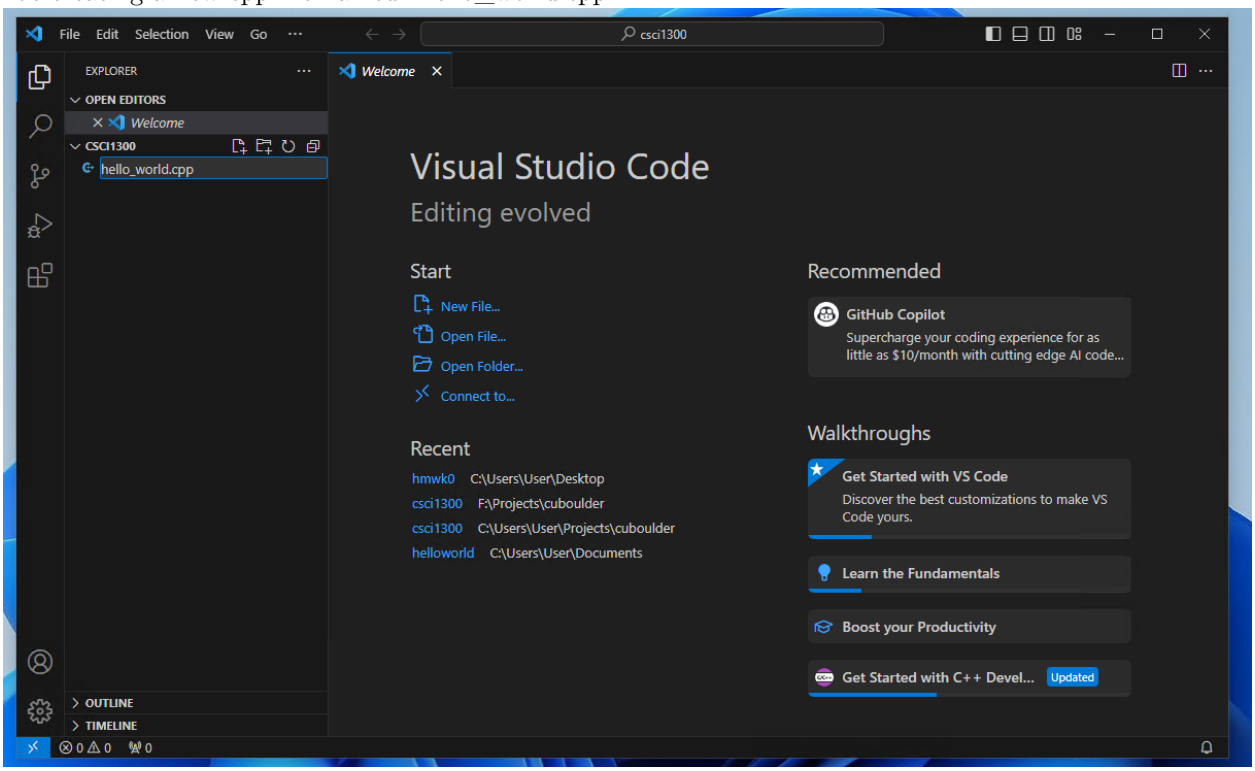
Step 1: Open Code Directory with VS Code

Begin by opening VS Code. Open the folder containing your code. For demonstration purposes, we will be using the "csci1300" folder located in Desktop. Once you have navigated to the target folder and selected it, click "Select Folder" on the lower right corner of the folder selection dialog. You should see something resembling the image below.



Step 2: Open A File to Debug

If you already have a cpp file in this folder, double click that file in the left pane. In this demonstration, we will be creating a new cpp file named "hello_world.cpp".

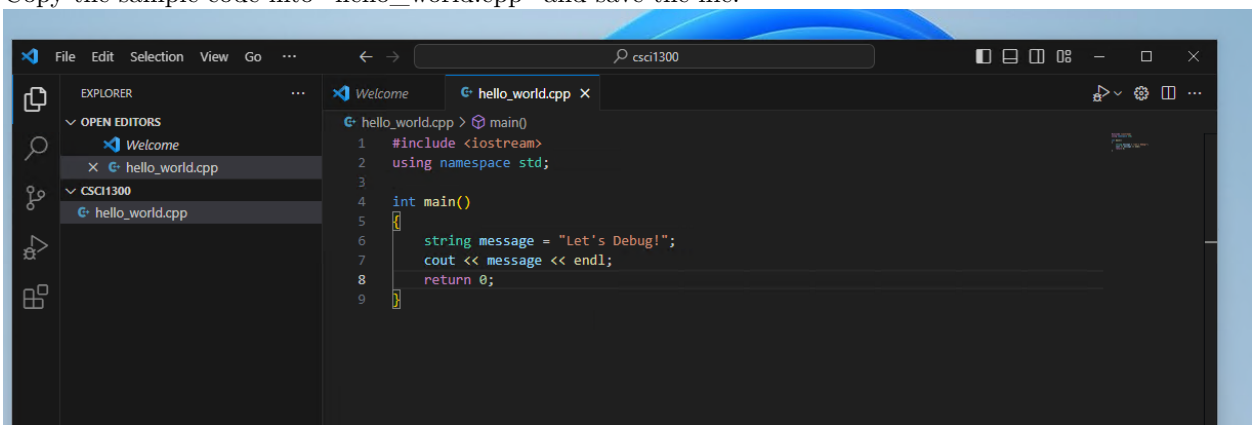


The sample code we will be using is as follows. This code declares a string variable named "message"

and prints it to the terminal.

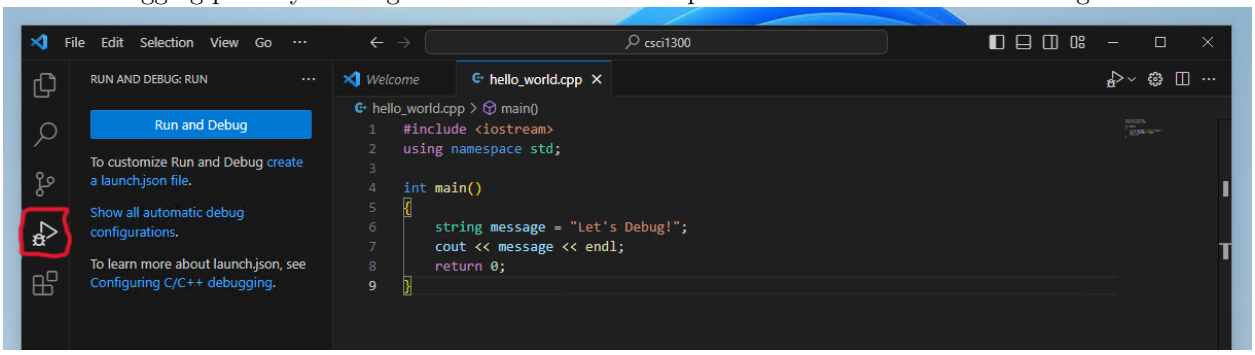
```cpp
#include <iostream>
using namespace std;

int main()
{
    string message = "Let's Debug!";
    cout << message << endl;
    return 0;
}
```

Copy the sample code into “hello_world.cpp” and save the file.



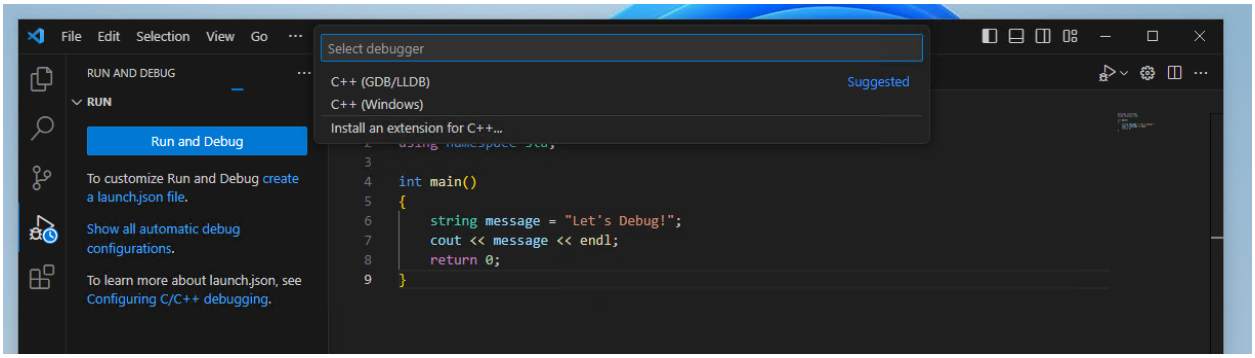
Step 3: Click “Run and Debug”

Go to the debugging pane by clicking on the icon in the red square. Then click “Run and Debug”.

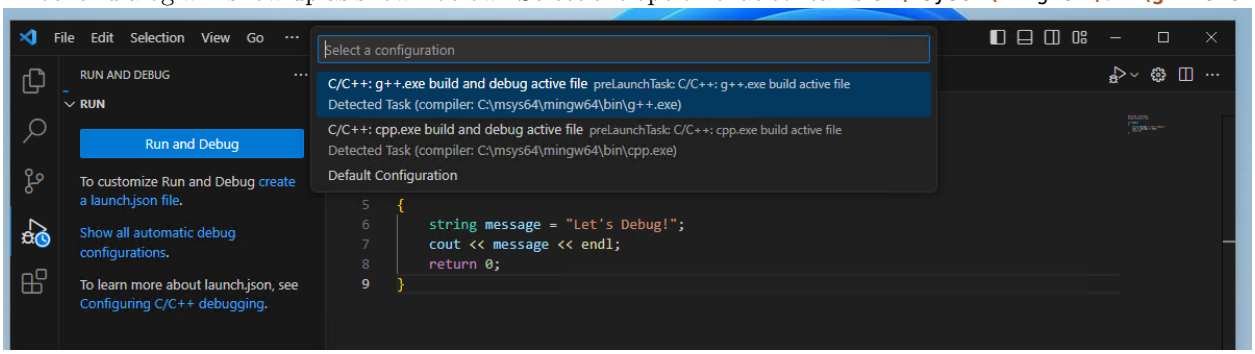


Step 4: Select Debugging Option

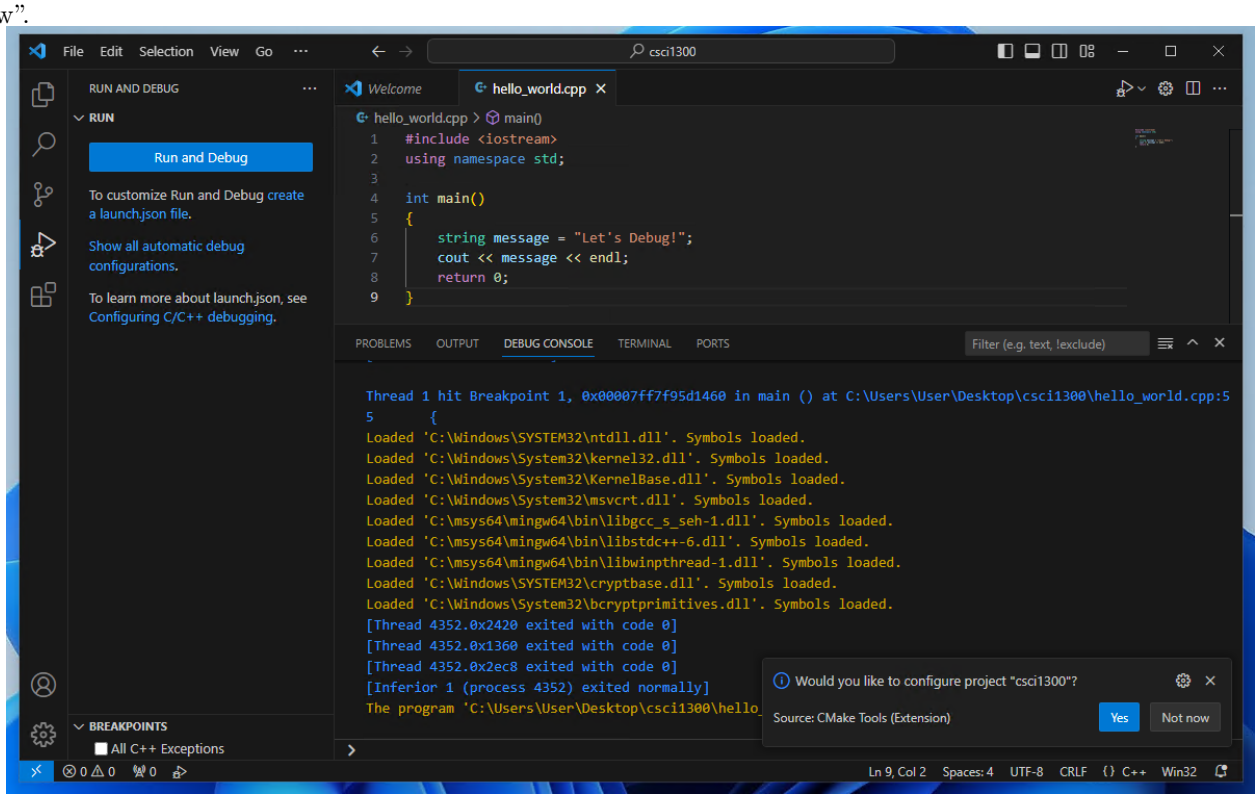
After clicking “Run and Debug”, a dialog will appear as shown below. Select “C++ (GDB/LLDB)”



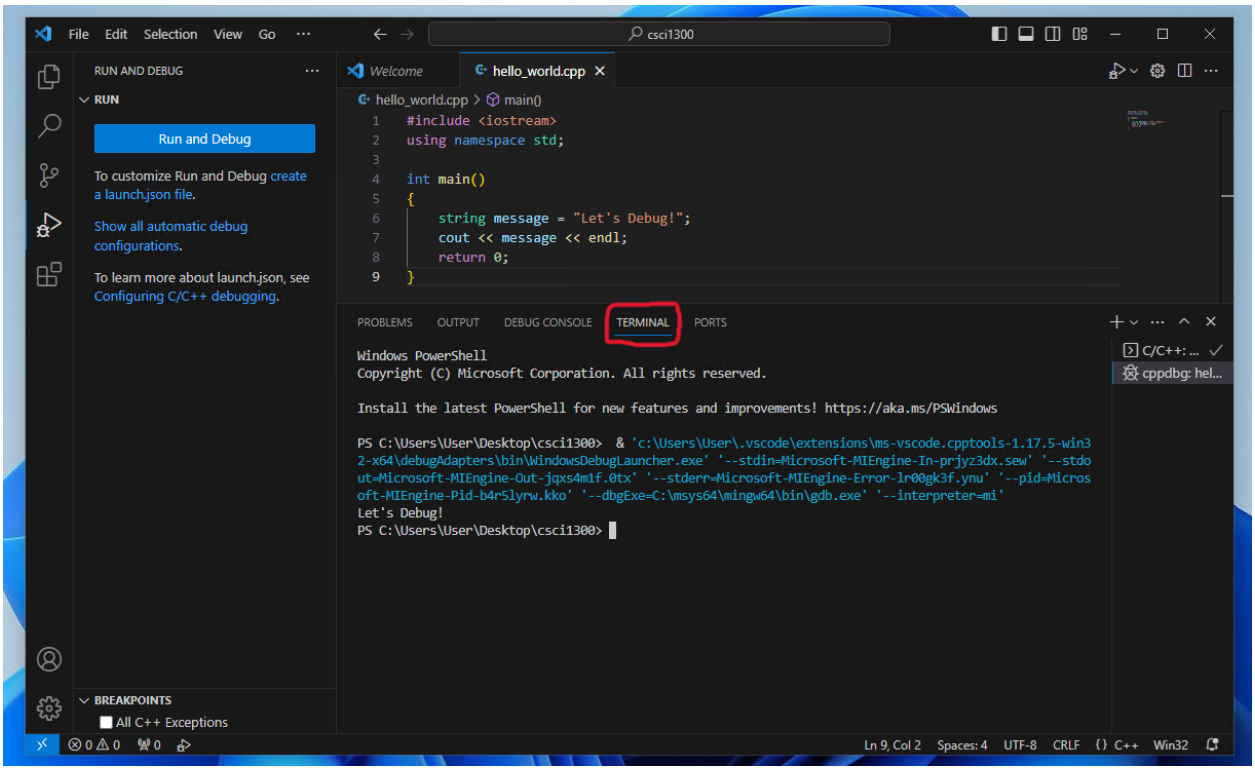
Another dialog will show up as shown below. Select the option that contains `C:\msys64\mingw64\bin\g++.exe`



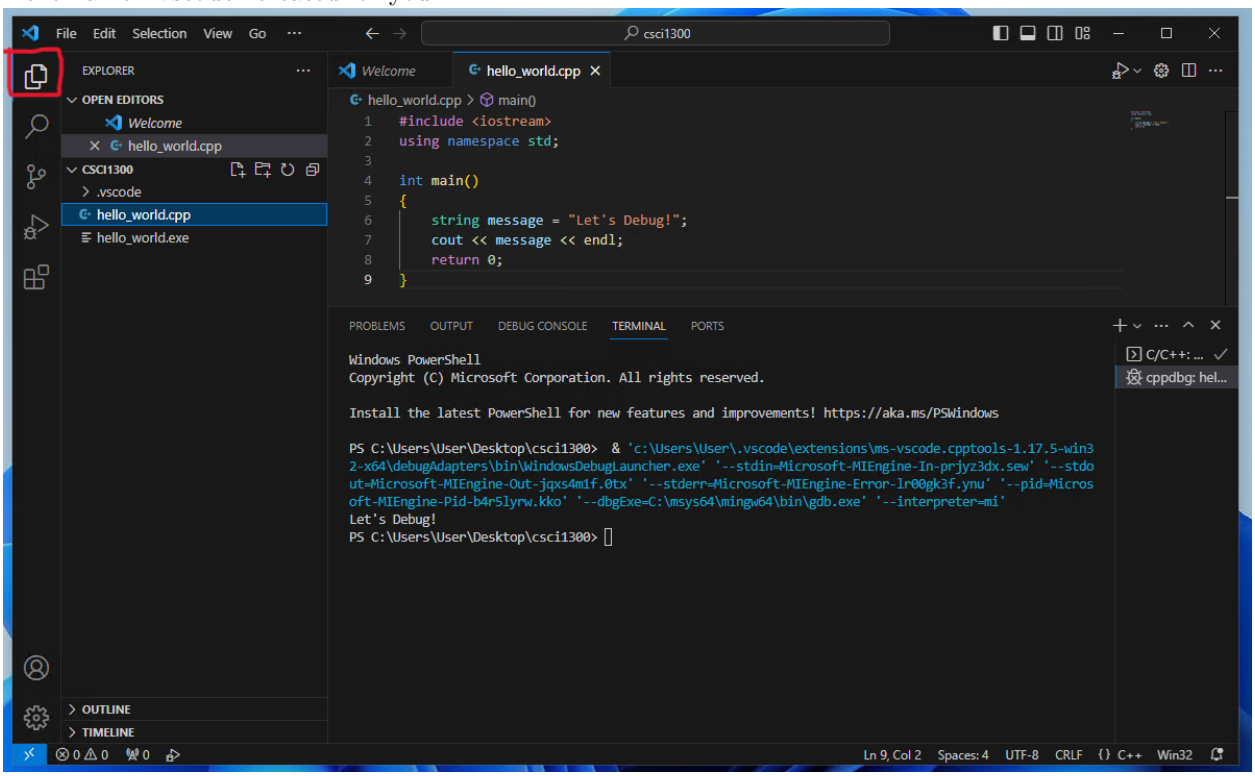
The result will be as shown below. For the dialog at the bottom right corner of the screen click "Not now".



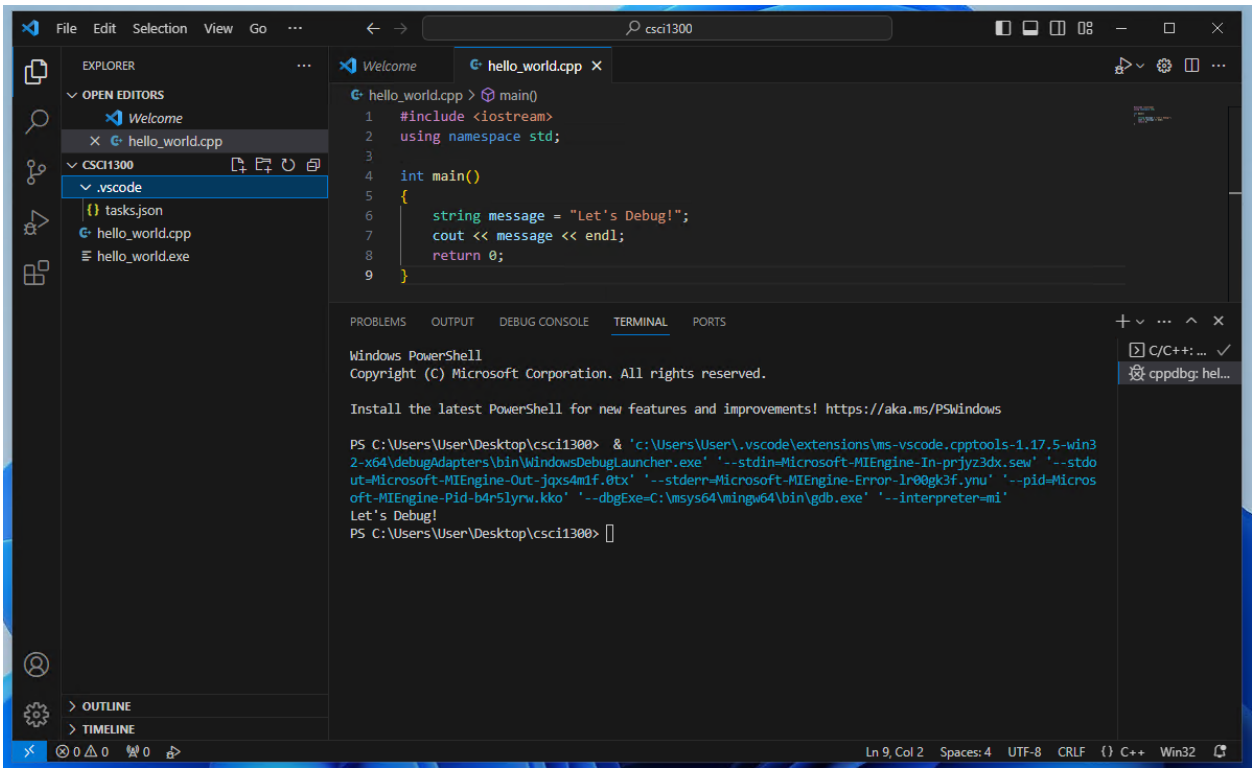
Click "TERMINAL" in the center of the screen. You will see the message in the terminal "Let's Debug!". At this point you have successfully setup and ran your debugger in VSCode. To make the debugger to the same compilation flags that we use, we will have to edit a configuration file called "tasks.json".



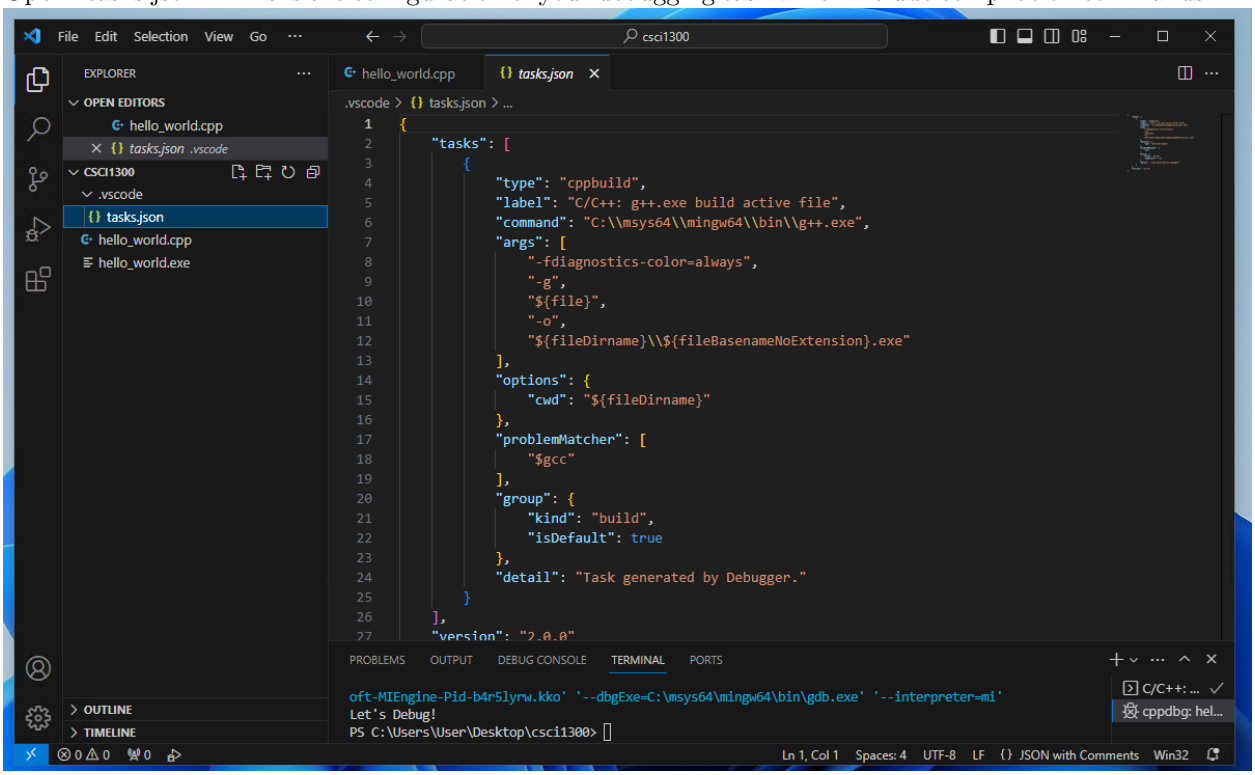
Navigate back to the file explorer pane by clicking the first icon on the left pane. There will be a folder with the name ".vscode" created for you.



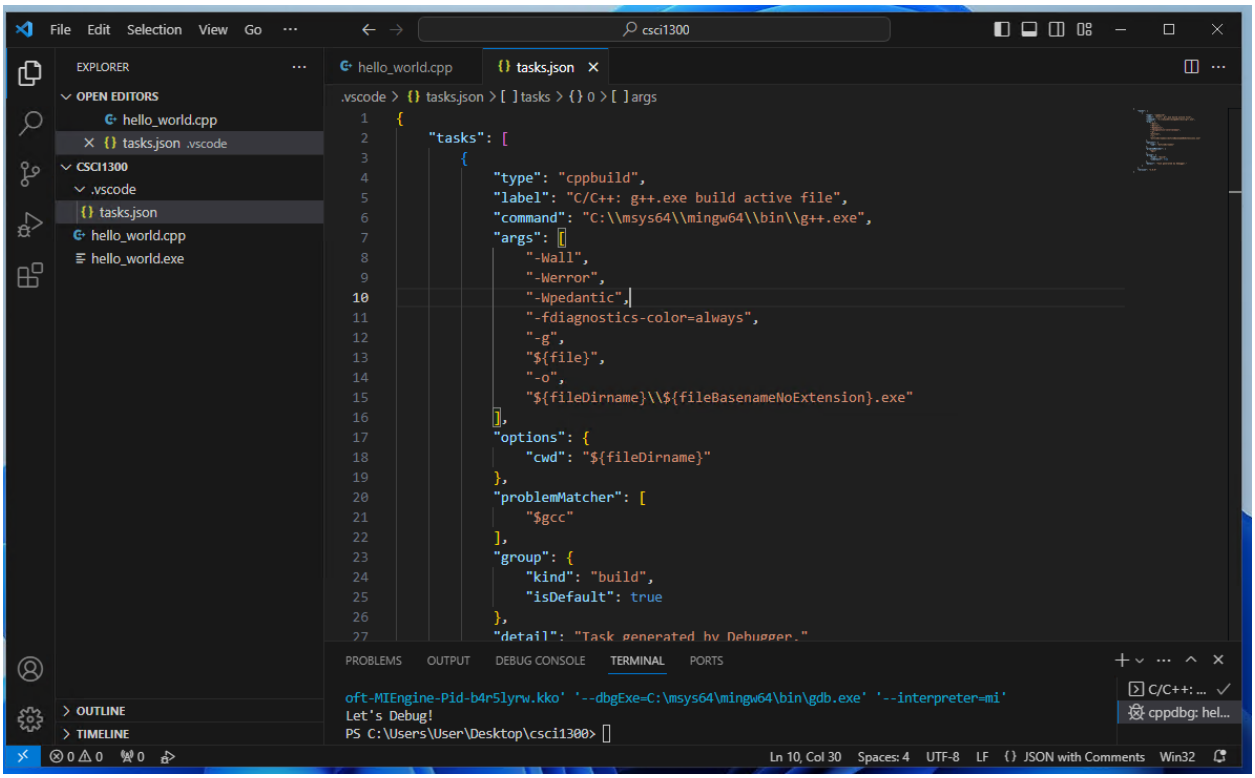
Click the folder to view the files inside. You will find "tasks.json" inside.



Open “tasks.json”. This is the configuration of your debugging task which include compilation commands.



Add the values “-Wall”, “-Werror” and “-Wpedantic” in the “args” section as shown below. Save the file after you have completed editing.

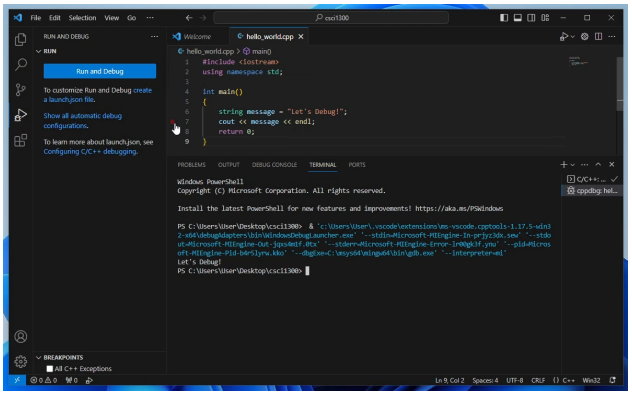


Step 5: Adding breakpoints

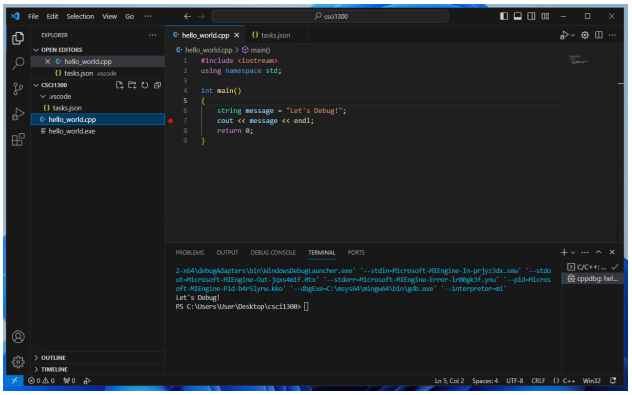
Breakpoints are line or specific code that we want to inspect. Once a breakpoint is reached during runtime, the program will pause for you to inspect the value of each variable. Once you are done inspecting that particular point, you may continue the program by pressing continue.

Navigate back to your cpp file by clicking the tab of you file at the center top of the screen. To add a breakpoint, hover your mouse over the line number of your code that you would like to inspect. The image below is selecting line 7. After clicking, the red dot will not disappear when you move your mouse away.

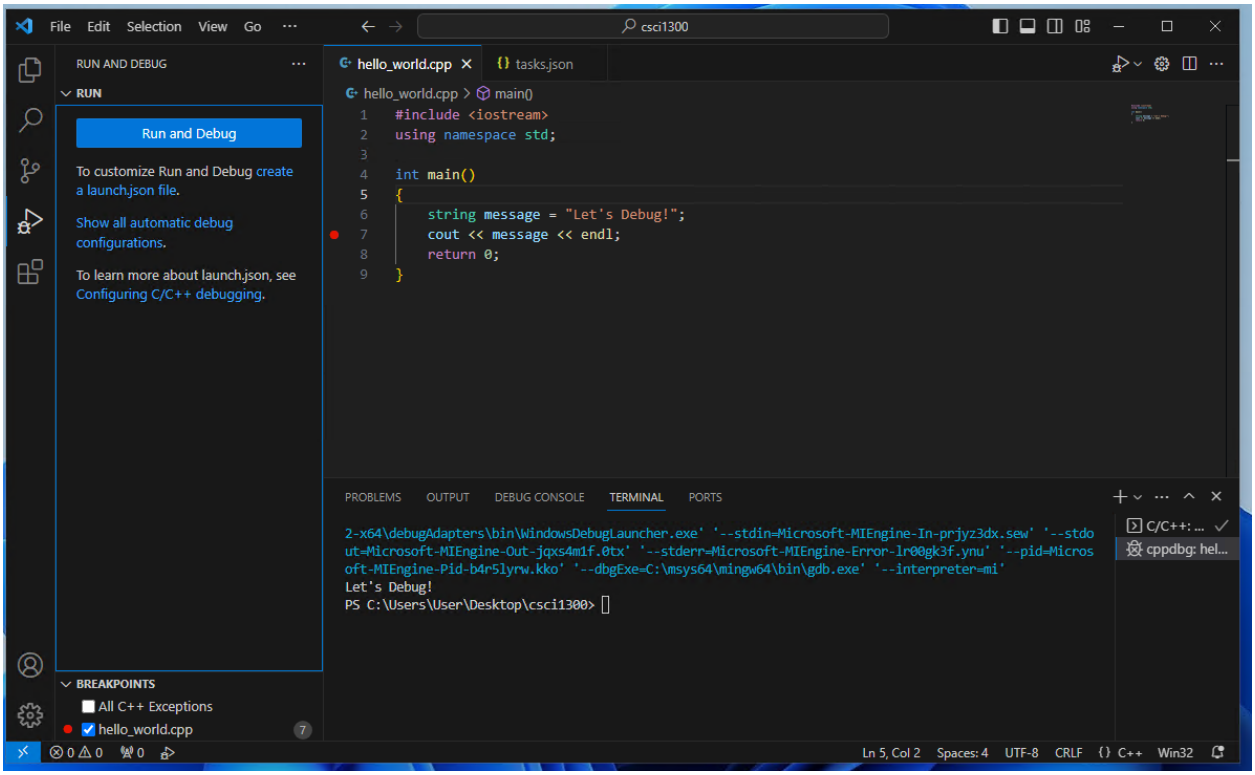
Selecting line of code



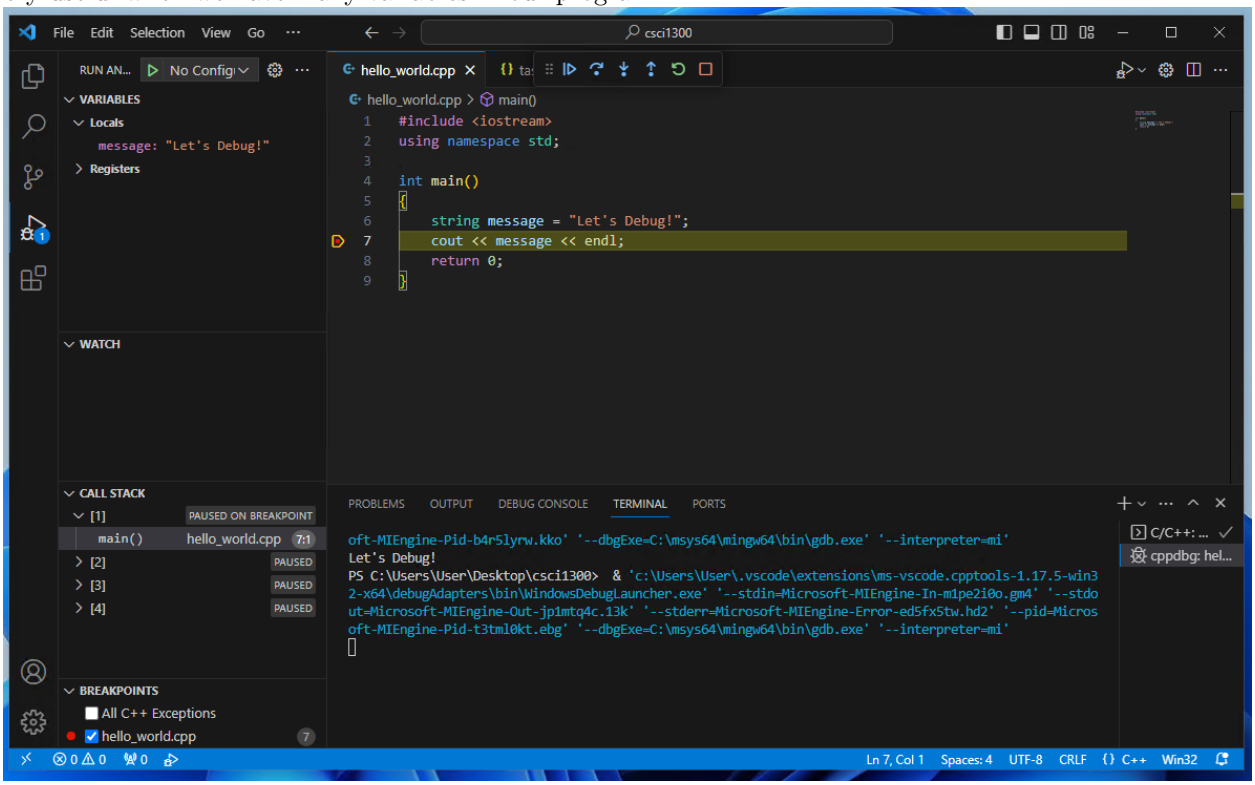
Breakpoint added to line of code



Now we can run the debugger by navigating to the debugging pane and clicking ”Run and Debug” as in step 3 and 4.



Notice that our program pauses at the line the breakpoint was added. There is also a list of variables on the left pane. For this instance, the variable "message" is being shown with the value "Let's Debug!". This is very useful when we have many variables in our program.



The menu to control the execution of your program looks like this:

The buttons, left to right, are as follows:

- Continue: go until the next breakpoint is encountered
- Step Over: go to the next line, skipping the details of functions
- Step Into: go to the next line, if necessary jumping into a function that is called
- Step Out: jump to the end of the function your are in, returning to wherever that function was called from
- Restart: start the program over from the beginning
- Stop: quit debugging

Most of the time you can do most of your debugging with just the Continue and the Step Over buttons as long as you have put breakpoints at all the points of your code you care about.

2.2 Mac Installation

MacOS users will use a debugger called lldb to debug their C++ code.

Step 1: Make a File

Write this code into a new file called "hello_world.cpp."

```cpp
#include <iostream>
using namespace std;

int main()
{
    string message = "Let's Debug!";
    cout << message << endl;
    return 0;
}
```

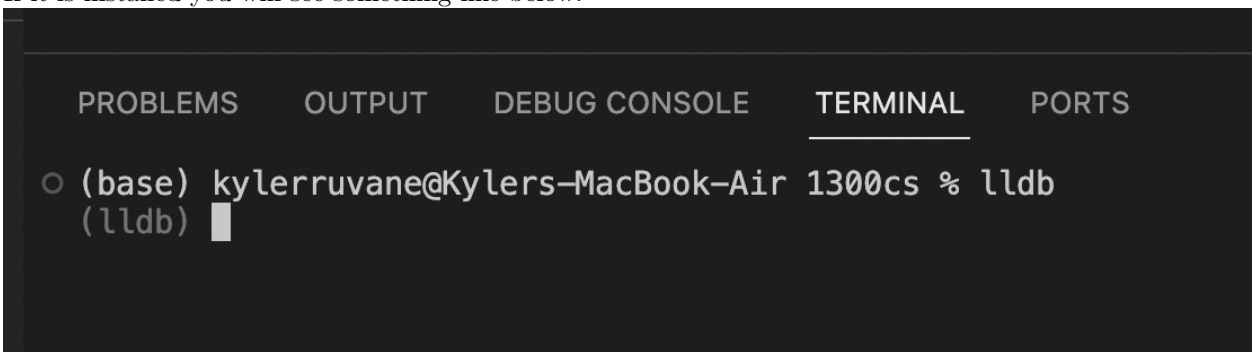
Step 2: Instal lldb

This is probably already installed on your computer. Lets double check and install it if we need to.

Open a terminal in vscode by clicking "Terminal > New Terminal" in your top bar. In your terminal type:

```
lldb
```

If it is installed you will see something like below.



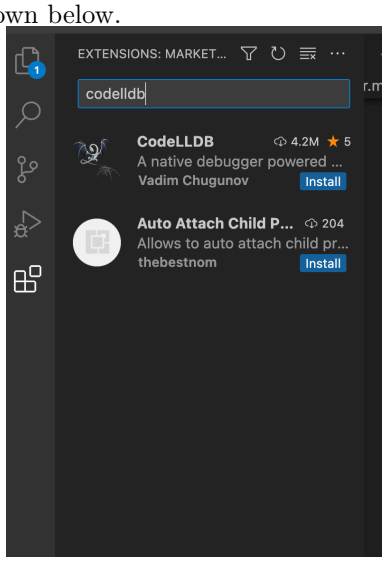
You can exit this program by typing

```
exit
```

If it is not installed, your mac should prompt you if you want to install it. Say yes and go through the install. Verify it is installed by typing lldb in your terminal again and seeing the above screenshot.

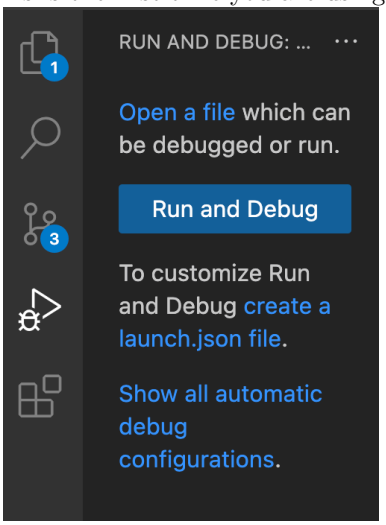
Step 3: Install VSCode Extension: CodeLLDB

Open the extensions tab on the left of VS Code and Search "CodeLLDB" and click install for the top result shown below.

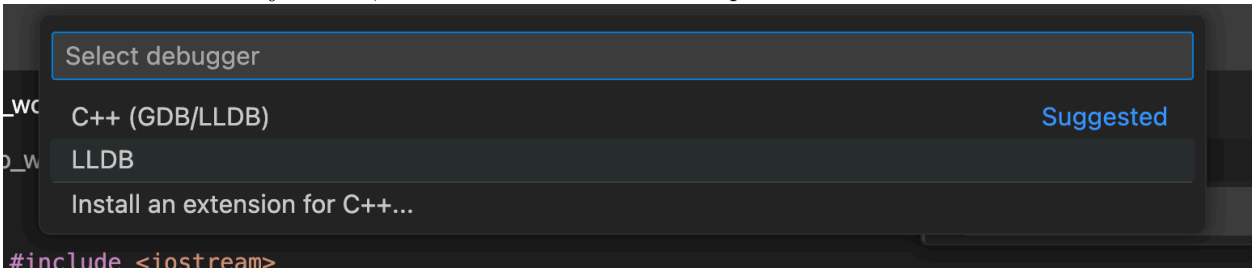


Step 4: Click the "Run and Debug" Tab

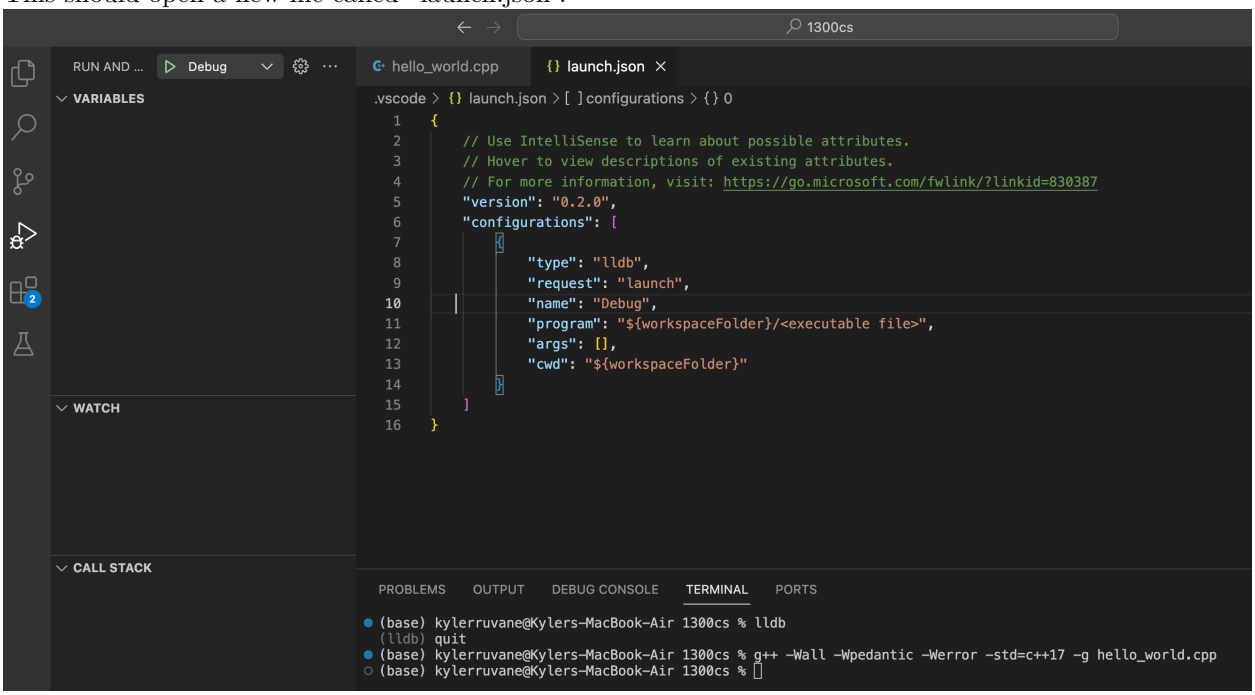
If this is the first time you are using this tab, it will look like this:



Press "create a launch.json file", and then choose the second option "LLDB".



This should open a new file called “launch.json”.



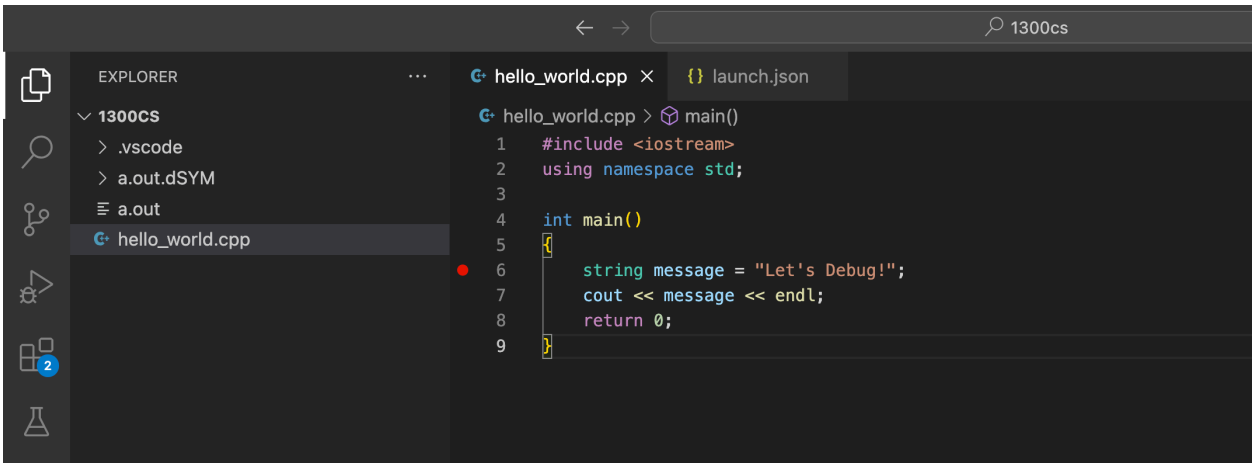
Delete everything in this file and replace it with:

```
{
    "version": "0.2.0",
    "configurations": [
        {
            "type": "lldb",
            "request": "launch",
            "name": "a.out debug",
            "program": "${workspaceFolder}/a.out",
            "args": [ ],
            "cwd": "${workspaceFolder}"
        }
    ]
}
```

Save the file with ”Command + S” or ”File > Save”.

Step 5: Add Breakpoints to your Code

The red dot to the left of line 6 is a ’breakpoint’. Add your own by hovering your mouse just to the left of the line number you want to add a breakpoint to.

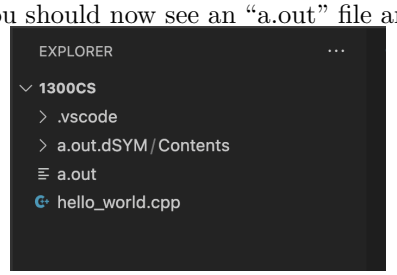


Step 6: Compile Your Code with -g

In your terminal type

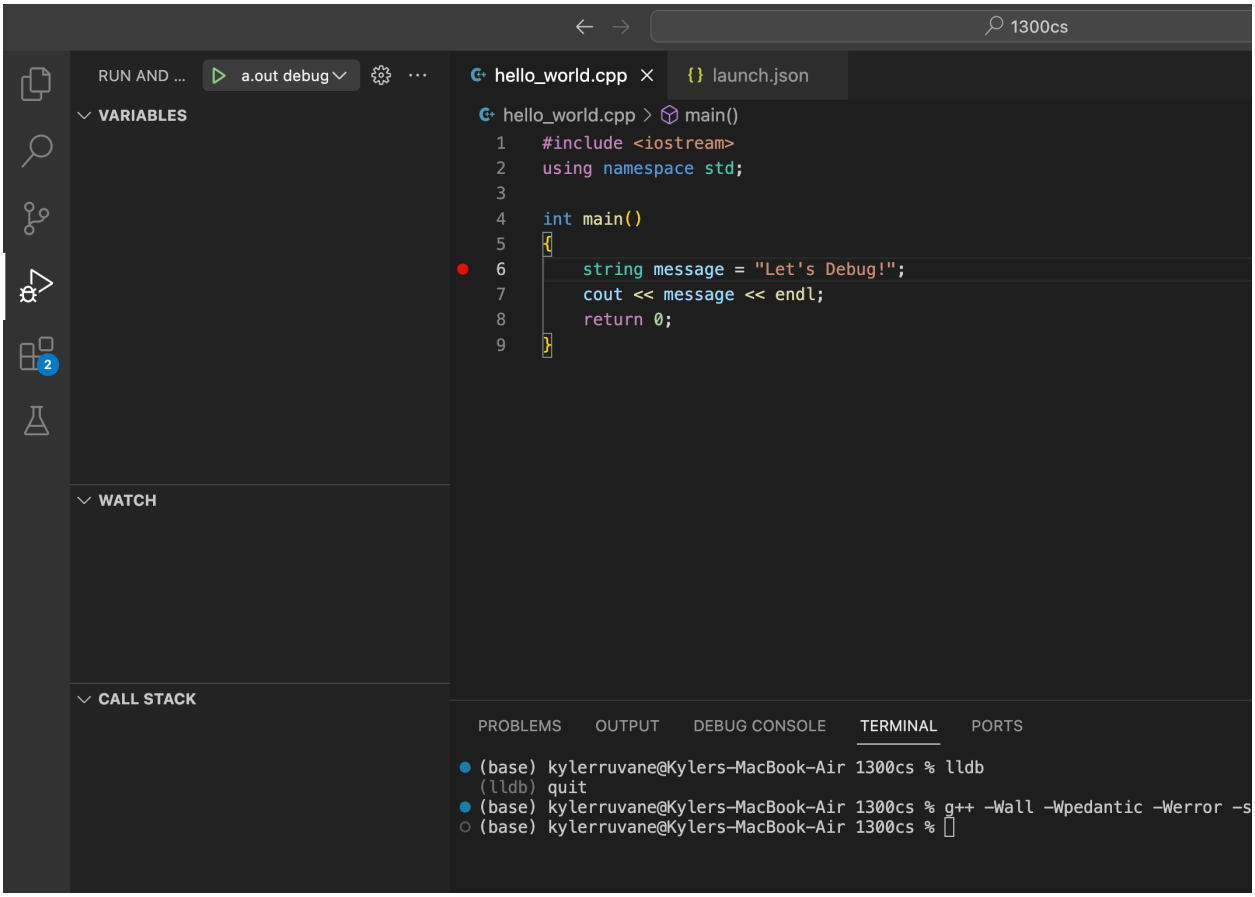
```
g++ -Wall -Wpedantic -Werror -std=c++17 -g hello_world.cpp
```

Note we have added the flag `-g` which tells the compiler to do extra stuff so we can debug the program. You should now see an "a.out" file and an 'a.out.dSYM" folder.



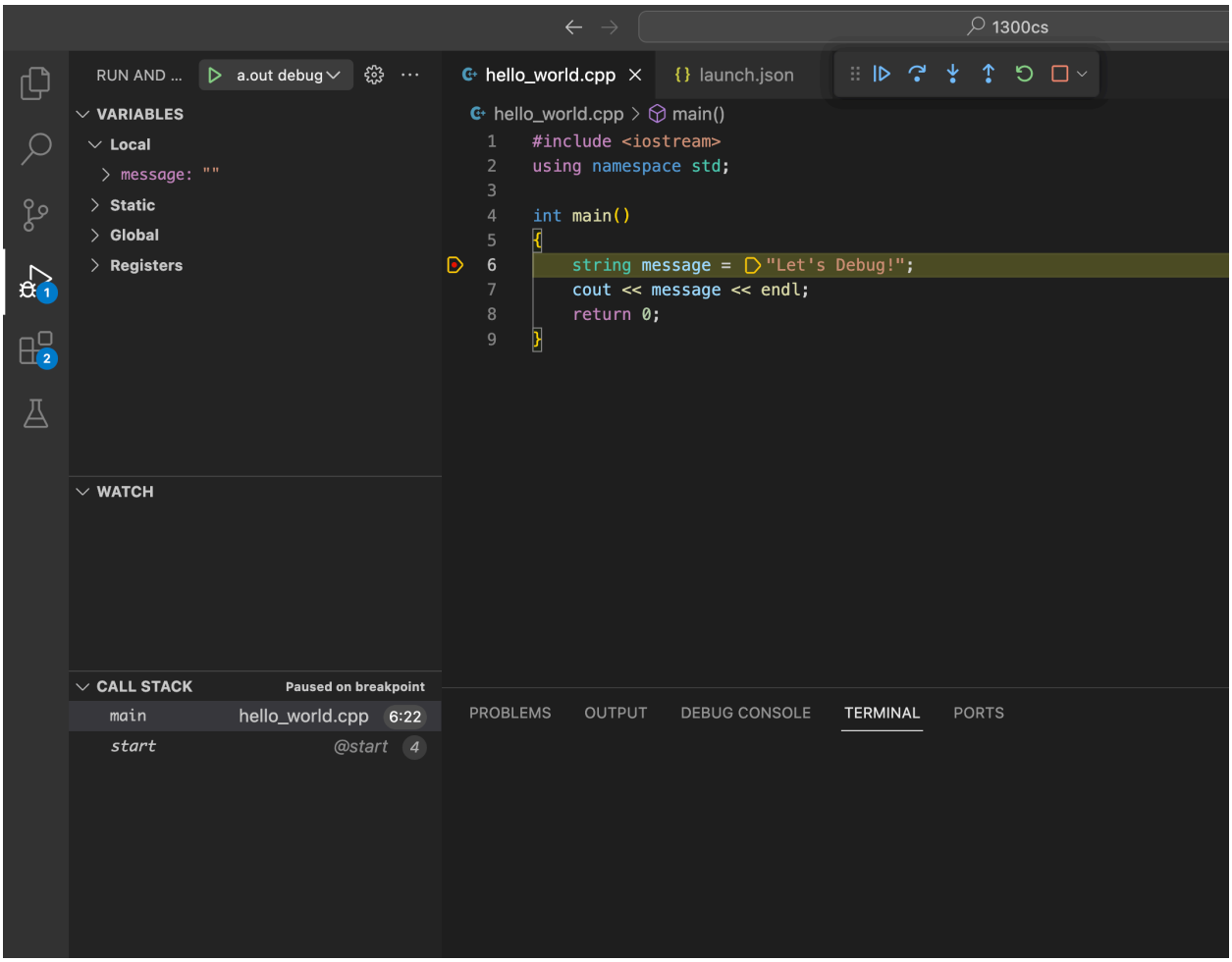
Step 7: Press the "Run and Debug" tab

This is the sideways triangle with a lady-bug on it.



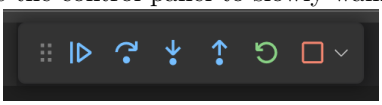
Step 8: Press the "a.out" Green Triangle Button

You are now debugging!



Step 9: Control Your Program Live

Your program will have paused at the first breakpoint it encountered, highlighting the line it is stuck on. Use the control panel to slowly walk through your code execution. The control panel looks like this:



The buttons, left to right, are:

- Continue: go until the next breakpoint is encountered
- Step Over: go to the next line, skipping the details of functions
- Step Into: go to the next line, if necessary jumping into a function that is called
- Step Out: jump to the end of the function your are in, returning to wherever that function was called from
- Restart: start the program over from the beginning
- Stop: quit debugging

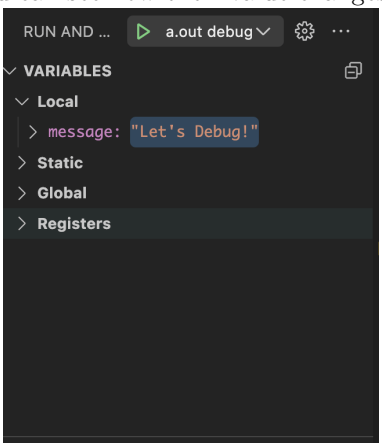
Most of the time you can do most of your debugging with just the Continue and the Step Over buttons as long as you have put breakpoints at all the points of your code you care about.

What Should I Look For?

The most important two pieces of information you get from debugging is 1. What lines of code are being executed when I run it and 2. What are the values of my variables during execution.

For 1. you can see this in the main .cpp file with the active line being highlighted yellow.

For 2. you can see this in the left panel where all of your in-scope variables will be listed with their value. You can see how their value changes as you step through the program execution.



Appendix B: Formatting Guide

1 Naming Conventions

Give variables descriptive names, such as `first_name` or `homework_score`. Avoid one-letter names like `a` or `x`, except for loop counter variables such as `i`.

Example 1.0.1. Poor variable names would be:

```
int thing = 16;
double a = 2.2;
string x = "Michael";
```

Better variable names would be:

```
int cups_per_gallon = 16;
double lbs_per_kilo = 2.2;
string first_name = "Michael";
```

2 Whitespace

"Whitespace" refers to all of the empty space you can use to organize your code including new lines, indentation, and extra spacing. Good whitespace usage helps reduce the strain on the reader's eyes. The compiler ignores whitespace, which allows you can place things anywhere and format them however you want. To maximize readability, there are a few guidelines for whitespace usage.

Indentation: Increase your indentation by one increment of each brace "{" and decrease it once on each closing brace "}". Use Tab to increase indent and Shift+Tab to decrease indentation. You can also increase indent for multiple lines by highlighting them and pressing CTRL+] on Windows and CMD+] on Macs, and decrease the indent for multiple lines by highlighting them and pressing CTRL+[on Windows and CMD+[on Macs.

New Lines: You should use one blank line to separate blocks of code. You can separate sections of code based on the tasks they are meant to complete. You should also put any brackets on their own lines to help visually break up your code.

Example 2.0.1. Here is a poor example of whitespace usage:

```
int main(){int number = 0;
if(number < 5){
cout << "Less than 5" << endl;
}else{
cout << "Greater than 5" << endl;
}
number++;
}
```

Here is a better example of whitespace usage:

```
int main()
{
    int number = 0;
    if(number < 5)
    {
        cout << "Less than 5" << endl;
    }
    else
    {
        cout << "Greater than 5" << endl;
    }
    number++;
}
```

3 Commenting

Your code should be well-commented. Use comments to explain what you are doing, especially if you have a complex code section. These comments are intended to help other developers understand how your code works. Comments can also help you remember your own code if you have to go back and read it a week or a month after writing it. Single-line comments should begin with two forward slashes (//). Multi-line comments begin with one forward slash and an asterisk (/* ... comments here ... */).

Example 3.0.1. Here is a single line comment:

```
// CSCI 1300 Fall 2024
```

Here is a multi-line comment:

```
/*
Algorithm:
Input: two numbers
Output: sum of input numbers

1. Ask the user to enter a number
Save in variable number_1
2. Ask the user to enter a number
Save in variable number_2
3. Compute sum
sum = number_1 + number_2
4. Display sum to user
*/
```

Appendix C: Syntax Guide

This syntax guide was designed with heavy inspiration from the Coding with Harry C++ syntax guide.

1 Basics

The basic start for a C++ program is:

```
#include <iostream>
using namespace std;

int main() {
    return 0;
}
```

Declaring variables:

```
bool my_bool;
bool my_bool_initialized = false;
char my_char;
char my_char_initialized = 'A';
int my_int;
int my_int_initialized = 1;
double my_double;
double my_double_initialized = 3.5;
float my_float;
float my_float_initialized = -2.1;
string my_string;
string my_string_initialized = "Hello!";
```

Terminal output:

```
cout << [statement to print];
cout << variable_contents;
cout << "Hello!";
```

Terminal input:

```
cin >> variable_input;
getline(cin, string_input);
```

Comments:

```
// It's a single line comment

/* It's a multi-linecomment*/
```

Compiling:

```
g++ -Wall -Werror -Wpedantic -std=c++17 file1.cpp file2.cpp ...
```

2 Decisions

Decisions are decided based off of boolean values. In this guide, any time `< condition >` appears, it should be replaced fully (including the angle brackets) with something that expresses to a boolean value.

If statements:

```
if ( < condition > ){
    //code
}
```

If-Else statements:

```
if ( < condition > ){
    //code
}
else {
    //more code
}
```

If-Else-If statements:

```
if ( < condition > ){
    //code
}
else if ( < condition2 > ){
    //more code
}
```

Integer switch statements. The case values can vary:

```
switch(integer_variable){
    case 1:
        //commands
        break;
    case 2:
        //commands
        break;
    //any further cases
    default:
        //commands
}
```

Character switch statements. The case values can vary:

```
switch(character_variable){
    case 'A':
        //commands
        break;
    case 'B':
        //commands
        break;
    //any further cases
    default:
        //commands
}
```

3 Functions

Functions are small packages of code that can be called multiple times. The general format of a function requires a return type (which is a data type), a function name, and a list of parameter inputs, like so:

```
<return type> FunctionName(<parameter 1>, <parameter 2>, ...);
```

There can only be one return type and it must be a data type. It can be any data type you have seen so far, or it can also be "void" which means the function returns nothing. There can be as many parameters as you like, and they must be written as <parameter i> = <data type of parameter i> <name of parameter i>.

Function prototypes are just the function declaration followed by a semicolon.Here are a few function prototype examples:

```
int AddNumbers(int numOne, int numTwo);
void PrintMenu(string menu);
double calcCircum(double radius);
```

When implementing the function, it is instead written with brackets:

```
<return type> FunctionName(<parameter list>){
    /* code for function */
    return <something of return type>;
}
```

When calling the function, you provide the list of arguments:

```
int sum = AddNumbers(firstVal, secondVal);
sum = (4, 5);
```

4 Strings

Declaring strings:

```
string my_string = "Hello World";
```

Getting the length of strings:

```
int string_length = myString.length();
```

Appending two strings:

```
string first_name = "Harry ";
string last_name = "Potter";
string full_name = first_name.append(last_name);
```

Accessing or changing characters:

```
char first_character = my_string[0];
my_string[1] = 'a'; //my_string now stores "Hallo World"
```

5 Loops

While loops repeat code as long as a condition is true:

```
while (< condition >)
{
/* code block to be executed */
}
```

Do-While loops will execute the block of code, and then execute it again as long as a condition is true:

```
do
{
/* code block to be executed */
} while (< condition >);
```

For loops will execute a block of code a given number of times. This is done by creating a variable, modifying that variable and go until a particular condition is false.

```
for (<create and initialize variable>; <condition>; <modify variable>){
    /* code block to be executed */
}
```

For example, to count from 0 to 9:

```
for (<create and initialize variable>; <condition>; <modify variable>){
    /* code block to be executed */
}
```

The keyword break terminates a loop:

```
    break;
```

The keyword continue skips the rest of the current iteration of the loop:

```
    continue;
```

6 Objects

To define a class:

```
class Class_Name {
    public: // Access specifier
        // member functions

    private:
        //data members
};
```

To create an object of that class:

```
Class_Name object_name;
```

Constructors describe how a new object's data members should be initialized. It does not have a return type. It must match the class name.

```
class Class_Name {
    public: // Access specifier
        Class_Name(); //default constructor
        Class_Name(<parameter list>); //parameterized constructor

    private:
        //data members
};
```

To call a member function on a particular object:

```
    object_name.functionName();
```

7 Libraries

To use a library you need to know the library name and include it.

```
#include<LibraryName>
#include "libraryFileYouWrote.h"
```

7.1 Math

The library is called <cmath>.

The square root function:

```
double square_root = sqrt(169);
```

The power function (returns the value of x raised to the power y):

```
double power = pow(x, y);
```

7.2 File I/O

The library is called <fstream>.

Creating and writing to a text file:

```
#include <iostream>
#include <fstream>
using namespace std;

int main() {
// Create and open a text file
ofstream my_file("filename.txt");

// Write to the file
my_file << "File Handling in C++";

// Close the file
my_file.close();
}
```

Opening a file and reading one line:

```
#include <iostream>
#include <fstream>
using namespace std;

int main() {
// Open a file to read from
ifstream my_file("filename.txt");

string file_line;

getline(my_file, file_line); //stores the first line in the file in file_line

// Close the file
my_file.close();
}
```