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!

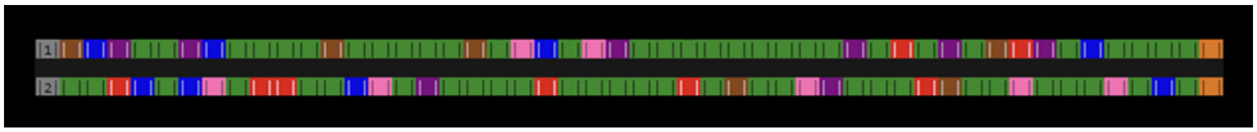

Figure 8.4: Example of the 2-lane Board when players choose **different path types**

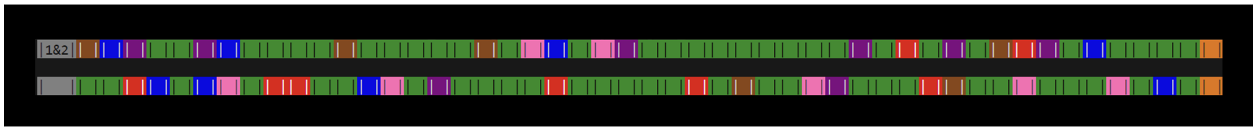

Figure 8.5: Example of the 2-lane Board when players choose **the same path type**

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.

- **Tuesday, 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:

```
#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;
}
```

```
// 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.