De La Salle University
College of Computer Studies
Department of Software Technology

Codenames

CCPROG2 Machine Project (AY 2024-2025, Term 3)

Release Date: June 2, 2025

Due Date: July 28, 2025 (M) 0800

Prepared by: Kristine Kalaw

1 Project Specifications

You must create a C program based on the project specifications detailed below. This project serves as a culminating program activity to showcase the programming concepts learned in class. Through the project, students should be able to demonstrate the learning outcomes indicated in the syllabus.

For this project, you are allowed to do this **individually** or to be in a **group of 2 members** only. Should you decide to work in a group, the following mechanics apply:

- **Individual Solution:** Even if it is a group project, each student is still required to create their INITIAL solution for the MP individually without discussing it with any other students. This will help ensure that each student goes through the process of reading, understanding, solving the problem, and testing the solution.
- **Group Solution:** Once both students are done with their solution: they discuss and compare their respective solutions (ONLY within the group) – note that learning with a peer is the objective here – to see a possibly different or better way of solving a problem. They then come up with their group's final solution – which may be the solution of one of the students or an improvement over both solutions. Only the group's final solution, with internal documentation (part of the comment indicating whose code was used or was an improved version of both solutions), will be submitted as part of the final deliverables. It is the group solution that will be checked/assessed/graded. Thus, only 1 final set of deliverables should be uploaded by one of the members on the Canvas submission page. [Groupings will be created in AnimoSpace to facilitate this.]
- **Individual Demo Problem:** As each is expected to have solved the MP requirements individually before coming up with the final submission, both members should know all parts of the final code to allow each to INDIVIDUALLY complete the demo problem within a limited amount of time (to be announced nearer the demo schedule). This demo problem is given only on the day/time of the demo and may be different per member of the group. Both students should be present during the demo, not just to present their individual demo problem solution, but also to answer questions about their group submission.
- **Grading:** The MP grade will be the same for both students – UNLESS there is a compelling and glaring reason as to why one student should get a different grade from the other – for example, one student cannot answer questions properly OR does not know where or how to modify the code to solve the demo problem (in which case deductions may be applied or a 0 grade may be given – to be determined on a case-to-case basis).

1.1 Overview

You are tasked to create a text-based game for facilitating a game of Codenames. Within the program, users can (1) manage the game components; (2) view game statistics; and, of course, (3) play the game proper.

(image source: https://www.czechgames.com/games/codenames)

1.2 Codenames

Victory speaks in code…

In the actual board game Codenames, two teams compete to see who can make contact with all of their agents first. There are 25 **codename cards**, each bearing a single word, that are arranged in a 5x5 grid. The spymasters look at a **key card** showing the identity of each card, then take turns clueing their teammates. A clue consists of a single word and a number, with the number suggesting how many cards in play have some association with the given clue word. The teammates then identify one agent they think is on their team; if they're correct, they can keep guessing up to the stated number of times + 1; if the agent belongs to the opposing team or is an innocent bystander, the team's turn ends; and if they reveal the assassin, they lose the game. Spymasters continue giving clues until one team has identified all of their agents or the assassin has removed one team from play (source: http://czechgames.com/games/codenames).

For your MP, you are to create a program that helps in facilitating the game. This entails:

- Team management, in terms of who are the members and spymaster for the red and blue team
- Display a grid of 25 randomly selected codename cards in a 5x5 grid and select a key card
- The team's selection of a codename from the grid and revealing its identity
- Allow a team to continue guessing or end their turn
- Show the game results

1.3 Program Flow

When the program starts, it should display the main menu. The main menu contains **at least** the following options:

- **New Game**: If the user selects this option, the program should proceed to the game proper. See Section 1.3.1 for more details.
- **Top Spymasters**: If the user selects this option, the program should display the game statistics. See Section 1.3.2 for more details.
- **Exit**: If the user selects this option, the program should terminate without crashing. Note that this is the only valid way for the program to terminate.

```
Main Menu
  [1] New Game
  [2] Top Spymasters
  [0] Exit

>> 1
```

Sample interface for the main menu

1.3.1 Game Proper

When a game starts, the following flow should be observed:

1. **Determine Blue Team members**
    - A team has a minimum of 2 players.
    - The players of the game can be new or existing. Existing players are chosen from a list. If it is a new player, ask for their username.
        - The username can be up to 36 characters and must be unique.
        - Note that in the next runs of the program, this "new player" should already be part of the existing player list (i.e., changes to the list of players persist).
        - Assume that the program can handle up to a maximum of 50 players.
    - The list of players should be found in **`players.txt`**. You are free to decide on the file format but take note of the following player data: username and number of games won as spymaster.

```
Current Blue Team members:
  + JaneBond
  + R3D_W1D0W

What would you like to do?
  [0] Done adding team members
  [1] <Add new player>
  [2] nu11
  [3] velvet-dagger

>> 1
```

```
New player username: spy-ce_gal21
```

Sample interface team player selection and creation

2. **From the Blue Team members, select 1 to be the spymaster**

```
Who will be the spymaster?
  [1] JaneBond
  [2] R3D_W1D0W
  [3] spy-ce_gal21

>> 2
```

Sample interface for spymaster selection

3. **Repeat steps 1 and 2 for the Red Team**

4. **Determine the 5x5 grid of codenames (words)**
   - The 25 unique words must be randomly selected from **`codenames.txt`** and displayed in a 5x5 grid.
   - This file is already provided for you. See Section 3.4 on how to access it.

5. **Select a key card**
   - A key card is randomly selected from the existing text files inside the **`keycard/`** folder. The filename for the textfiles within this folder is a number (i.e., **`keycard/<key_card_number>.txt`** See Section 1.5 for details about the file format.
   - Assume that the program can handle up to a maximum of 20 key cards.
   - The first 4 key cards are already provided for you. See Section 3.4 on how to access it.

6. **Game Start!**
   - The key card determines which team will start.
   - In each team's turn, there is the Spymaster Phase and the Agents Phase:
     - Spymaster Phase
       - During this phase, it is assumed that only the spymasters are looking at the screen.
       - The spymaster should be able to see the key card and the current state of the grid.
       - The spymaster should input a number to indicate the number of words associated with their clue.
     - Agents Phase
       - The remaining team members must not see the key card; they should only see the current state of the grid.
       - There must be a means for the team to select an unrevealed card, and after selection, the game should reveal the identity of that card (blue, red, bystander, assassin)

        - If the card belongs to the team's color, the Agent Phase continues until the team guesses up to the number provided during the spymaster phase, plus one.
        - If the card belongs to the other team's color or is a bystander, the current team's turn ends.
        - If the card is revealed to be an assassin, the current team loses.
    - Each team takes turns guessing words until all of one team's agents are found, or until they reveal the assassin, ending the game immediately.

```
AIR             [-BYSTANDER-]  MOON           LOCK           POINT

TOWER           LONDON         PLATE          YARD           [---BLUE----]

CAP             [---BLUE----]  [---BLUE----]  [----RED----]  DEGREE

[-BYSTANDER-]   WATER          [-BYSTANDER-]  ALIEN          SCIENTIST

[----RED----]   LINK           HORSE          [-BYSTANDER-]  GOLD
```

Sample interface for displaying the current state of the grid

7. **Show game results**
    - The winner of the game is revealed.

1.3.2 View Statistics

When the user is in this part of the program, it should have an interface to display the Top 5 Spymasters. These are the players who have been spymaster with the most number of wins

1.4 Interface

This project is text-based and should be run from the terminal or command line. Students have the flexibility to choose the type of user interface they wish to implement, provided it is intuitive and adheres to the specifications outlined in this document. Creativity in presenting the game is encouraged, as long as the specifications are followed.

There must be error checking for all player inputs. If the input comes from a list of options, then only the options should be allowed. If the input requires a string, then the input string must be valid. If the user enters an invalid input, the program should not crash. The program must display an error message and continue to prompt the user until an appropriate input is given.

1.5 Key Card File Format

The key card determines which team takes a turn first as well as the position of the team's agent on the 5x5 word grid. There are: 7 innocent bystanders, 1 assassin, 8 red agents, and 8 blue agents. Note that these are only 24 out of the 25 grid elements. The 25th colored agent depends on which team starts the game. For example, if you draw the key card wherein the blue team plays first, then there should be 9 blue agents and 8 red agents on the key card's grid.

`keycard/<key_card_number>.txt` should be in the following format:

```
<starting team><newline>
<newline>
<identity 01> <identity 02> <identity 03> <identity 04> <identity 05><newline>
<identity 06> <identity 07> <identity 08> <identity 09> <identity 10><newline>
<identity 11> <identity 12> <identity 13> <identity 14> <identity 15><newline>
<identity 16> <identity 17> <identity 18> <identity 19> <identity 20><newline>
```

```
<identity 21> <identity 22> <identity 23> <identity 24> <identity 25><newline>
<end of file>
```

- **`<starting team>`** is either **`B`** or **`R`** to indicate blue or red team, respectively
- **`<identity ##>`** is either **`B`**, **`R`**, **`I`**, or **`A`** to indicate blue agent, red agent, innocent bystander, or assassin, respectively

Sample **`keycard/<key_card_number>.txt`**, saved as **`keycard/01.txt`**

```
B<newline>
<newline>
B I B R R<newline>
R B R R B<newline>
I B A I I<newline>
B R I B A<newline>
B I B R I<newline>
<end of file>
```

1.6 Bonus Points

Students can earn **up to 10 bonus points (exceeding the maximum score)** by implementing additional features that will enrich the quality of the project. Bonus points will only be given for meaningful and non-trivial additions to the base project (i.e., simply using different font colors is considered trivial, and thus will not incur bonus points). Ideas for bonus features include, but are not limited to:

- Codenames Manager
  - This feature is a facility where users can manage the various game components such as key card management, player management, and/or codenames word list management.
- More interesting game statistics
  - Based on the current project specifications, the only other player data stored is the number of games won as spymaster. You can explore to store more player data and present more interesting statistics.
- Save and Continue game
  - An ongoing unfinished game may be saved for the meantime and continued sometime later.
- Good user experience (UX) design

Bonus points will only be given to projects that have met all the minimum requirements. The amount of bonus points is up to the discretion of the instructor. Ensure that additional features do not conflict with the minimum requirements stated in the specifications; otherwise, these may result in some of those requirements not being met. When in doubt, please consult your instructor.

1.7 Implementation Requirements

The project must abide by the following implementation guidelines:

- The program should be **fully implemented in the C programming language** (C99, NOT C++).
- Observe the appropriate use of conditional statements, loops, and functions.
  - **Do not use brute force.**
  - Codes must be modular (i.e., split into several logical functions). Codes that are not modularized properly will not be accepted, even if the program works properly.
- You can use these in the future when you have much more experience and wisdom in programming. However, **while you are in the PROG series, you are NOT ALLOWED to do the following:**
  - ✗ to declare and use global variables (i.e., variables declared outside any function)
  - ✗ to use **`exit`**, **`goto`** (i.e., to jump from code segments to code segments), **`break`** (except in **`switch`** statements), or **`continue`** statements
  - ✗ to use **`return`** statements to prematurely terminate a loop, function, or program

    - ✗ to use **`return`** statements in void functions
    - ✗ to call the **`main()`** function to repeat the process instead of using loops
- You may only use the standard C libraries.
- You may use topics outside the scope of CCPROG2, but this will be **self-study**.
- Adhere to coding conventions and **must include proper documentation** as indicated in Section 2.1.
- Debug and test your program exhaustively. The submitted program should have
    - **NO syntax errors and warnings**
        - It should compile properly with the command:

          `gcc -Wall -std=c99 <yourMP.c> -o <output>`
    - **NO logical errors**
        - Based on the test plan that the program was subjected to
        - As part of the requirements, you are required to create a test plan and execute your program against it. See Section 2.2 for more details.

2 Documentation

2.1 Internal Code Documentation

You are **required** to have the following internal documentation in your source code.

2.1.1 Program Structure

The structure of your program should look like this:

```
/**
 *  Description     : <short description of the project>
 *  Author/s        : <student1 full name (last name, first name)>
 *                    <student2 full name (last name, first name)>
 *  Section         : <your section>
 *  Last Modified   : <date when last revision was made>
 *  Acknowledgments : <list of references used in the making of this project>
 */

/* preprocessor directives> */

/* definitions (i.e., constants, typedefs, structs) */

/* function implementations */

int main()
{
  /* your project code */

  return 0;
}

/**
 * This is to certify that this project is my/our own work, based on my/our personal
 * efforts in studying and applying the concepts learned. I/We have constructed the
 * functions and their respective algorithms and corresponding code by myself/ourselves.
 * The program was run, tested, and debugged by my/our own efforts. I/We further certify
 * that I/we have not copied in part or whole or otherwise plagiarized the work of
 * other students and/or persons.
```

```
 *
 * <student1 full name (last name, first name)> (DLSU ID# <number>)
 * <student2 full name (last name, first name)> (DLSU ID# <number>)
 */
```

A template of the program structure is provided for you. See Section 3.4 on how to access it.

2.1.2 Functions and Function Comments

For **each function you define**, add the following comments. These serve as documentation to describe what the function does, what each parameter is for, and what is being returned, if any. If applicable, include pre-conditions as well. Pre-conditions refer to the assumed state of the parameters. Refer to this link for more information about C code documentation. Follow the format below when writing functions and function comments:

```
/**
 *  <Description of what the function does>
 *  @param <name1> <description of parameter 1>
 *  @param <name2> <description of parameter 2>
 *    ...
 *  @param <nameN> <description of parameter N>
 *  @return <description of the return value>
 *  @pre <list of pre-conditions>
 */
<return type> <function name>(<parameter list>)
{ // open brace is at the beginning of the new line, aligned with matching close brace
  <variable declarations> // variable declarations should be before the start of any
                          // statements, not inserted in the middle of the loop,
                          // to promote readability
  <rest of the function implementation>
} // close brace is aligned to the matching open brace
```

Below is a sample function with function comment adhering to the format indicated above:

```
/**
 *  Computes the average of the non-negative numbers from a given list of numbers
 *  @param arr The starting address of the array containing the list of numbers
 *  @param arrSize The size of the array
 *  @return The average of the non-negative numbers from the list of numbers
 *  @pre arr can include positive numbers, negative numbers, and zeros
 *  @pre arrSize is the correct size of arr
 */
float getAverage(int arr[], int arrSize)
{
  float average;
  int sum = 0, n = 0, i;

  for (i = 0; i < arrSize; i++)
      if (arr[i] >= 0)
      {
          sum += arr[i];
          n++;
      }
```

```
  if (n == 0) // prevent divide by zero
    average = 0;
  else
    average = sum / (float);

  return average;
}
```

2.1.3 Structures

For **each structure you define**, add the following comments. These serve as documentation to describe the structure and its members.

```
/**
 * <Description of the structure>
 */
struct coordinate
{
  <data type> <member1>; // Description of member1
  <data type> <member2>; // Description of member2
    ...
  <data type> <memberN>; // Description of memberN
};
```

Below is a sample structure with documentation adhering to the format indicated above:

```
/**
 * Represents a 2D point
 */
struct coordinate
{
  int x; // The x-coordinate of a point
  int y; // The y-coordinate of a point
};
```

2.2 Test Plan

As part of the requirements of this project, you are also required to submit a test plan document. This document shows the tests done to ensure the correctness of the program.

- It should be presented in a table format. Refer to Appendix A for an example.
    - A template of the test plan is provided for you. See Section 3.4 on how to access it.
- There should be **at least three test cases per function** (as indicated in the Test Description).
    - There is no need to test functions that are intended for interface display/design.
- Test descriptions are supposed to be unique and should indicate classes/groups of test cases on what is being tested.

3 Deliverables

3.1 Submission & Demo

- **The final MP deadline is July 28, 2025 (M) 0800 via AnimoSpace.** After the indicated time, the submission page will be locked and thus considered no submission (equivalent to 0 in the MP grade).
- The schedule for the demo will be announced at a later date.
- During the demo, you are expected to **appear on time**, be able to **answer questions about the implementation of the project**, and/or be able to **revise the program based on a given demo problem**. Note that more than just delivering a working program, the most important outcome of this project is for you to learn and have a good understanding of how your program works. Therefore, failure to meet any of these expectations could result in a grade of 0.
- During the MP demo, it is expected that the program can be compiled successfully and will run. **If the program does not run, the grade for the project is automatically 0**. However, a running program with complete features may not necessarily get full credit, as implementation (i.e., code) and other submissions (e.g., non-violation of restrictions evident in code, test script, and internal documentation) will still be checked.

3.2 Deliverables Checklist

The following are the deliverables:

- ☐ Upload via AnimoSpace submission a zip file (see right on how zip file content is organized) containing the following:
  - ☐ Source code
  - ☐ Test plan PDF
  - ☐ Text files
- ☐ Email this zip file as an attachment to YOUR own DLSU email address on or before the deadline

Your submission should have this folder structure before compressing it into a zip file:

```
src/
|-- keycards/
    |-- 01.txt
    |-- <#>.txt
|-- mp.c
|-- codenames.txt
|-- players.txt
|-- TestPlan.pdf
|-- <any other files and folders used as
part of this project>
```

3.3 Milestones

There is only 1 deadline for this project: **July 28, 2025 (M) 0800**, but the following are suggested targets:

MS-0: Design and Planning (June 7, 2025)

- ☐ Read through the project specifications

MS-1: Program Structure (June 14, 2025)

- ☐ Menu options and transitions
- ☐ Preliminary outline of the functions to be created

MS-2: Game Proper, Part 1 (June 28, 2025)

- ☐ Declaration/implementation of data structures
- ☐ Game Proper > Team creation
- ☐ Game Proper > Display grid
- ☐ Game Proper > Display keycard

MS-3: Game Proper, Part 2 (July 5, 2025)

- ☐ Game Proper > Spymaster Phase

- [ ] Game Proper > Agent Phase
- [ ] Game Proper > Show game results

MS-4: File Handling (July 12, 2025)

- [ ] Load players from a file
- [ ] Save a new player to a file
- [ ] Load codenames from a file
- [ ] Load key cards from a file
- [ ] Optional: Implement bonus features

MS-5: View Statistics (July 19, 2025)

- [ ] View Statistics > Top Spymasters
- [ ] Optional: Implement bonus features

MS-6: Finalization (July 26, 2025)

- [ ] Integration testing (test program as a whole, not per function)
- [ ] Double-check that all base requirements of the project are met
- [ ] Optional: Implement bonus features
- [ ] Zip and submit the final MP-related files

Note that each milestone assumes fully debugged and tested code/function, code written following coding convention and included internal documentation, and documented tests in the test plan. It is expected that at least one submission per target milestone is done in AnimoSpace. Even if this is not graded, this serves as version control and proof also of continuing efforts.

3.4 MP Starter Files

The MP starter files contain the following:

- A template of the MP program structure
- The list of codenames (**`codenames.txt`**)
- The first 4 key cards (**`keycard/01.txt`** to **`keycard/04.txt`**)
- A template of the test plan
    - Refer to Appendix B for the guide on how to export the template as a PDF.

4 Collaboration and Academic Honesty

A student cannot discuss or ask about design or implementation with other persons, with the exception of the teacher and their groupmate. Copying other people's work and/or working in collaboration with other teams are not allowed and are punishable by a grade of 0.0 for the entire CCPROG2 course and a case may be filed with the Discipline Office. Comply with the policies on collaboration and AI usage as discussed in the course syllabus.

5 Grading

For grading, refer to the MCO rubrics indicated in the syllabus

Acknowledgments / Disclosure

The following Generative AI tools were used to assist in the making of this document.

ChatGPT

- ChatGPT was used mainly for streamlining and/or sentence construction. The instructor validated the AI-generated output and modified it as needed.

Appendix A: Test Plan Format Sample

Function	#	Test Description	Test Input	Expected Output	Actual Output	P/F
getAverage	1	arr contains only non-negative numbers	arr = {1,2,3,4,5,0} arrSize = 6	2.5		
	2	arr contains only negative numbers	arr = {-1,-2,-3,-4,-5} arrSize = 5	0.0		
	3	arr contains a mix of positive numbers, negative numbers, and zeros	arr = {-1,0,5,-4,7,1} arrSize = 6	3.25		
	4	arr contains only zeros	arr = {0, 0, 0} arrSize = 3	0.0		
anotherFunction	1	...	...	...		
	2	...	...	...		
...	...	...	...	...		

Given the sample function in Section 2.1.2, the following are 4 distinct classes of tests:

1. Testing with arr containing only non-negative numbers
    - Non-negative numbers (positive numbers and zeros) are the only values that can be part of the average computation
2. Testing with arr containing only negative numbers
3. Testing with arr containing a mix of positive numbers, negative numbers, and zeros.
4. Testing with arr containing only zeros
    - Zero is a non-negative number

The following test descriptions are **incorrectly** formed:

- **Too specific:** Testing with arr = {1, 2, 3, 4, 5}
- **Too general:** Testing if the function can correctly compute the average of the non-negative numbers
- **Not necessary (because already defined in the pre-condition):** Testing with a negative arrSize

The test plan template can be accessed in Section 3.4. Refer to Appendix B for the guide on how to export the template as a PDF.

Appendix B: Export Test Plan Guide

1. In the **TestPlan** sheet:

   File > Download > PDF (.pdf)
2. Adjust the settings to:
   - Export: Current Sheet
   - Paper size: Legal (8.5 x 14)
   - Page orientation: Landscape
   - Scale: Fit to width
   - Margins: Narrow
   - Formatting dropdown
     - Leave as is
   - Headers& footers dropdown
     - Check: Current date
     - Check: Current time
     - Uncheck: Repeat frozen rows
3. Click Export.