CCPROG1 Machine Project Specifications

Term 1 AY 2024 – 2025

Introduction – Mahjong:

Mahjong is a traditional Chinese game spanning thousands of years of history. The mechanics of Mahjong are simple, as the game is largely based on luck and some strategy. For this term's CCPROG1, you will implement a stripped-down version of the original mahjong game. If you are already familiar with the original game, we will still use some key ideas and terminologies as they were originally used despite the game's modifications.

Gameplay:

Your project will only use one ordered suit, contrary to the four suits in the game. You can use any ASCII symbols (https://www.rapidtables.com/code/text/ascii-table.html) of your choice (e.g., numbers, letters, or special characters) to represent ordered tiles of 9. You should use symbols that follow some order, such as numbers or letters, for a more intuitive gameplay.

Just like the original game, each tile has 4 identical copies. Concretely, there are 36 tiles (draw pile) in the game. Given the limited tiles, your project will only have two players.

The tile values will be used to form the tile combinations mentioned below.

Start of the game:

To start a game, each player gets 8 random tiles from the draw pile at the beginning. If a player gets 4 identical tiles (or a Kang), a player can reveal it and draw an additional tile from the draw pile.

You may assume that a game will begin with player 1 and transition to player 2 in a round-robin fashion, and so on, until the game ends.

During a Player's Turn:

At the start of a player's turn, the player draws a tile from the draw pile.

Alternatively, if the other player throws a tile into the discard pile, the current player starts his turn by getting the newly discarded tile if it forms one of the combinations mentioned below. The player must reveal the formed combination to the other player.

At the end of a turn, the current player throws a tile from his hand into the discard pile.

Tile combination:

A tile combination can only be formed upon drawing a tile from the draw pile or the most recent discarded tile from the opponent.

- **Pung (3 identical tiles)**

  A Pung is a revealed set of three identical tiles—for example, 1-1-1 and 2-2-2.

- **Kang (4 identical tiles)**

  For this project, we will use one type of Kang, the revealed Kang. Whenever a player declares Kang, the player must draw a tile from the Draw pile regardless of how the Kang was formed.

- **Chi (3 consecutive tiles)**

  A Chi is a revealed set of three consecutive tiles – for example, 1-2-3 and 5-6-7.

- **Hu (Winning Combination)**

  A player may declare Hu if their hand (revealed and concealed) comprises the tile combinations mentioned. Once a player declares Hu, they must reveal their hand to the other player as proof.

End Game

The game ends when a player declares Hu, or there are only 5 tiles left in the draw pile. If the game ends because of the latter condition, then the game is considered a draw.

Technical Requirements

- Discarding of tile should be index-based as shown below. Additionally, your program should be able to handle invalid user input.

```
74981436
87654321 ← Index of the tiles
```

- Your program should detect the Kang combination upon the first distribution of tiles before player 1 gets to draw from the draw pile and start playing the game.

```
12311451
Player 1, would you like to declare Kang for tile 1 [y/n]? y

// After Clear Screen //
234571111
54321KKKK ← Revealed tiles due to tile combination can't be discarded anymore; K for Kangs
******** ← Player 2's tiles are concealed while player 1's turn
```

- Your program should conceal the other player's tiles when it's not their turn

```
 ****1111 ← 1 Here is revealed because of the Kang declaration; * tiles are the unrevealed tiles of player 1 because it's player 2's turn.
123563812
987654321 ← Index of tiles
```

```
Player 2, please select a tile to discard: 7 ← discard 7th tile
```

- Your program should clear the screen for every player turn using the code below. Please inform your instructor if this code doesn't work on your computer.

```
void clrscr()
{
    system("@cls||clear");
}
```

- Your program must ask for Pung, Chi, Kang, or Hu based on the drawn or discarded tile whenever applicable.

```
234571111
54321KKKK
********     ← Player 2's tiles are concealed while player 1's turn
Player 1, please select a tile to discard: 1
// After Clear Screen //
Player 1 discarded tile 7

****1111
68123331

Would you like to declare Chi for tile 6-7-8 [y/n]?
```

Bonus Features (Maximum of additional 10 pts.):

Below are some of the bonus features you can implement:

- Sorted display player's hand [2 pts.]
- Recognition of at least three special Hu hand combination such as hands and tails (e.g. 111999111), snake (e.g. 123456789) and purely combination tiles (e.g. 111222333). You may define more special hand of your choice as long as they are as varied as the shown in the examples [5 pts.]
- Display discard pile history (using linked list) [3 pts.]
- Implementation of additional suit [10 pts.]

Bonus features will only be checked if all of the basic features are working. Bonus features cannot alter the requirements and features of the project.

How to Approach the Machine Project

Step 1: Problem analysis and algorithm formulation

Read the MP Specifications again! Identify clearly what are the required information from the user, what kind of processes are needed, and what will be the output (s) of your program. Clarify with your professor any issues that you might have regarding the machine project.

When you have all the necessary information, identify the necessary functions that you will need to modularize the project. Identify the required data of these functions and what kind of data they will return to the caller. Write your algorithm for each of these modules/functions as well as the algorithm for your main program.

Step 2: Implementation

In this step, you are to translate your algorithm into proper C statements. While implementing, you are to perform the other phases of program planning and design (discussed in the other steps below) together with this step.

Follow the coding standard indicated in the course notes (Modules section in AnimoSpace).

You may choose to type your program in a text editor or an IDE (i.e. Dev-C IDE) at this point. Note that you are expected to use statements taught in class. You can explore other libraries and functions in C as long as you can clearly explain how these work. You may also use arrays, should these be applicable and you are able to properly justify and explain your implementation using these. For topics not covered, it is left to the student to read ahead, research, and explore by himself.

Note though that you are NOT ALLOWED to do the following:

- to declare and use global variables (i.e., variables declared outside any function),
- to use goto statements (i.e., to jump from code segments to code segments), [gotoxy() is allowed]
- to use the break or continue statement to exit a block. Break statement can only be used to break away from the switch block,
- to use the return statement or exit statement to prematurely terminate a loop or function or program,
- to use the exit statement to prematurely terminate a loop or to terminate the function or program, and
- to call the main() function to repeat the process instead of using loops.

It is best that you perform your coding "incrementally." This means:

- Dividing the program specification into subproblems, and solving each problem separately according to your algorithm;
- Code the solutions to the subproblems one at a time. Once you're done coding the solution for one subproblem, apply testing and debugging.

Documentation

While coding, you have to include internal documentation in your programs. You are expected to have the following:

- File comments or Introductory comments
- Function comments
- In-line comments

Introductory comments are found at the very beginning of your program before the preprocessor directives. Follow the format shown below. Note that items in between < > should be replaced with the proper information.

```
/*
     Description:      <Describe what this program does briefly>
     Programmed by:    <your name here>   <section>
     Last modified:    <date when last revision was made>
     Version:          <version number>
    [Acknowledgements: <list of sites or borrowed libraries and
sources>]
*/
<Preprocessor directives>

<function implementation>

int main()
{
   return 0;
}
```

Function comments precede the function header. These are used to describe what the function does and the intentions of each parameter and what is being returned, if any. If applicable, include pre-conditions as well. Pre-conditions refer to the assumed state of the parameters. Follow the format below when writing function comments:

```
/*    <Description of function>

      Precondition: <precondition / assumption>

      @param <name> <purpose>

      @return <description of returned result>

*/

<return type>

<function name> (<parameter list>)

      :
```

Example:

```
/* This function computes for the area of a triangle

   Precondition: base and height are non-negative values
```

```
   @param base is the base measurement of the triangle in
cm
   @param height is the height measurement of the triangle
in cm
   @return the resulting area of the triangle
*/
float
getAreaTri (float base,
            float height)
{
    ...
}
```

In-Line Comments are ot**her comments in major parts of the code. These are expected to explain the purpose or algorithm of groups of related code, esp. for long functions.**

Step 3: Testing and Debugging

Submit the list of test cases you have used. For each feature of your program, you have to fully test it before moving to the next feature. Sample questions that you should ask yourself are:

1. What should be displayed on the screen if the user inputs an order?
2. What would happen if I input incorrect inputs? (e.g., values not within the range)
3. Is my program displaying the correct output?
4. Is my program following the correct sequence of events (correct program flow)?
5. Is my program terminating (ending/exiting) correctly? Does it exit when I press the command to `quit`? Does it exit when the program's goal has been met? Is there an infinite loop?
6. and others...

Important Points to Remember:

1. You are required to implement the project using the C language (C99 and NOT C++). Make sure you know how to compile and run in both the IDE (DEV-C++) and the command prompt (via
   `gcc -Wall -std=c99 <yourMP.c> -o <yourExe.exe>`
2. The implementation will require you to:
   - Create and Use Functions

Note: Non-use of self-defined functions will merit a grade of **0** for the **machine project**. Too few self-defined functions may merit deductions. A general rule is to create a separate function for each option described above, unless some features are too similar that one function can serve the purpose for two [or more] of the options. Note that functions whose tasks are only to display are not included in the count for creating user-defined functions.

- Appropriately use conditional statements, loops and other constructs discussed in class (Do not use brute force solution. **You are not allowed to use goto label statements, exit statements. You are required to pass parameters to functions and not allowed to declare global or static variables.**) Refer to Step 2 on Implementation for other details and restrictions.
- Consistently employ coding conventions
- Include internal documentation (i.e., comments)

3. Deadline for the project is **7:59AM of November 25, 2024 (Monday)** via submission through **AnimoSpace**. Late submission up to max of 3 days will incur deductions. That is, submissions from 8:00AM of November 25 to 7:59AM of November 26 will incur 10% deduction, submissions from 8:00AM of November 26 to 7:59AM of November 27 will incur additional 20% deduction (total of 30% deduction), submissions from 8:00AM of November 27 to 7:59AM of November 28 will incur additional additional 30% deduction (total of 60% deduction in MP score), and submissions from 8:00AM of November 28 will not be accepted anymore as facility is locked. Once locked, no MP will be accepted anymore and this will result to a **0.0** for your machine project.
4. The following are the deliverables:

Checklist:

- ❒ Upload in AnimoSpace by clicking **Submit Assignment** on Machine Project and adding the following files:
  - ❒ source code*
  - ❒ test script**
- ❒ email the softcopies of everything as attachments to **YOUR own email address** on or before the deadline

Legend:

***Source Code also includes the internal documentation. The **first few lines of the source code** should have the following declaration (in comment) **BEFORE** the introductory comment:

```
/*****************************************************************************
*

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

                                <your full name>, DLSU ID# <number>
```

****
/

Test Script should be in a table format, with header as shown below. There should be **at least 3 distinct test classes (as indicated in the description) **per function**. There is no need to create test scripts for functions that only perform displaying on screen (like menu where there are no conditions being checked; but for the ASCII art which is displayed depending on size and on add-ons, there are conditions checked, thus should be included in the test script).

Function Name	#	Test Description	Sample Input (either from the user or to the function)	Expected Result	Actual Result	P/F
getAreaTri	1	**base and height measurements are less than 1.**	**base = 0.25** **height = 0.75**	**...**	**...**	
	2	:::				
	3					

Test descriptions are supposed to be unique and should indicate classes/groups of test cases on what is being tested. Given the function getAreaTri(), the following are 3 distinct classes of tests:

i.) testing with base and height values smaller than 1

ii.) testing with whole number values for base and height

iii.) testing with floating point number values for base and height, larger than 1.

The following test descriptions are incorrectly formed:

Too specific: testing with base containing 0.25 and height containing 0.75

Too general: testing if function can generate correct area of triangle

Not necessary -- since already defined in pre-condition: testing with base or height containing negative values

5. **MP Demo:** You will demonstrate your project on a specified schedule during the last weeks of classes. Being unable to show up on time during the demo or being unable to answer convincingly the questions during the demo will merit a grade of **0.0** for the **MP**. The project is initially evaluated via black box testing (i.e., based on output of running program). Thus, if the program does not compile successfully using gcc -Wall -std=c99 and execute in the

command prompt, a grade of 0 for the project will be incurred. However, a fully working project does not ensure a perfect grade, as the implementation (i.e., correctness and compliance in code) is still checked.

6. Any requirement not fully implemented and instruction not followed will merit deductions.
7. This is an **individual project**. Working in collaboration, asking other people's help, and/or copying other people's work are considered as cheating. Cheating is punishable by a grade of **0.0** for CCPROG1 course, aside from which, a cheating case may be filed with the Discipline Office.
8. The above description of the program is the basic requirement. A maximum of 10 points will be given as bonus. Use of colors may not necessarily incur bonus points. Sample additional features could be:
    - Creating a Graphical User Interface to look like a Vending Machine, use of other C libraries is allowed
    - Proper use of arrays and/or structures
    - Saving and loading the updated price and inventory count to file.

Note that any additional feature not stated here may be added but **should not conflict with whatever instruction was given in the project specifications**. Bonus points are given upon the discretion of the teacher, based on the difficulty and applicability of the feature to the program. Note that **bonus points can only be credited if all the basic requirements are fully met** (i.e., complete and no bugs).

HONESTY POLICY AND INTELLECTUAL PROPERTY RIGHTS

Honesty policy applies. Please take note that you are NOT allowed to borrow and/or copy-and-paste – in full or in part any existing related program code from the internet or other sources (such as printed materials like books, or source codes by others including that from AI). **You should develop your own codes from scratch by yourself.**