De La Salle University
College of Computer Studies
Software Technology Department

Turn-based Tiled Shooter

CCPROG1 Machine Project

Game Mechanics

Tile based shooters are shooting games where the movement of the player is restricted to specific spaces. One such game is Megaman Starforce that restricts the player's movement to only horizontal spaces as shown in Figure 1. For this project, a simpler version of this turn-based game will be made. Players have the option to either move left, move right or shoot every turn. After the player makes a move, the enemy's turn begins where they can move based on a simple pattern.

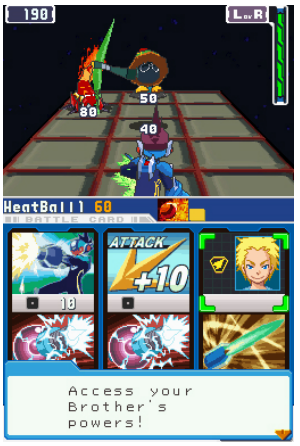


Figure 1: Megaman Starforce

To simplify the movement, both enemies and the player will move in a simple tile-based manner. The game will replicate this using ASCII symbols. There are 4 columns that the player and enemies can move from. The enemy has 5 rows they can move down from. The map to be used is illustrated in figure 2. Note, the illustration does not need to be followed perfectly but shown resembles something similar.

```
*
|X|X|X| |1
| | | | |2
| | | | |3
| | | | |4
| | | | |5
*-*-*-*-*
|P| | | |*
*
Score: 000
```

Figure 2: Sample of the initial map where X symbolizes the enemies and P symbolizes the player

The player can only move horizontally and will move one tile per movement. Firing the laser will destroy all enemies on the same column. The player cannot move and fire the laser on the same turn. There are no projectiles fired and shown on screen. Whenever an enemy is defeated, the player scores 10 points.

Enemies spawn from the top of the map and will move right, down, left, and down in a cycle. Only three enemies spawn per game at the top row of the map. Whenever an enemy is defeated, it will respawn at the top left corner of the enemy movable space. Upon respawn, the movement pattern of that specific enemy will reset. If two or more enemies are defeated in a single attack, the 2nd enemy will spawn to the right of the 1st and the 3rd enemy will spawn right of the 2nd. The score earned for each enemy defeated remains to be 10 points each even if multiple enemies are defeated in a single attack.

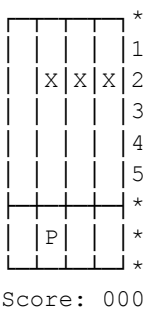
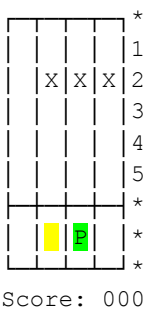
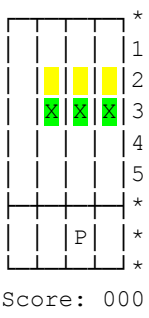
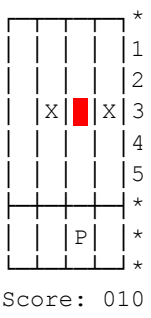
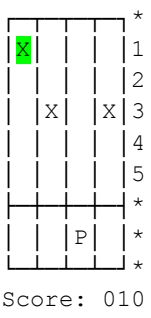
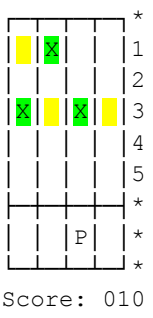
When the player reaches a score of 100 points, the player has won. If the enemies reach the bottom of the map, the player has lost. There is no need to ask the player if they wish to play the game again when reaching a game over.

Image reference:

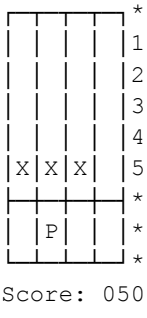
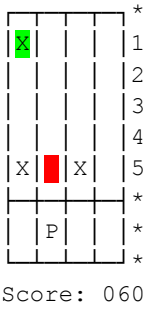
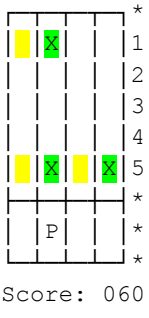
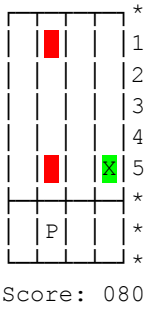
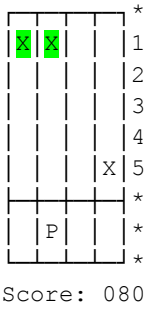
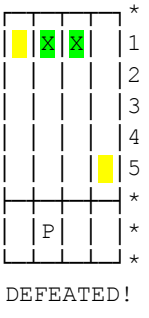
Wikipedia (2022). Mega Man Star Force. Taken from https://en.wikipedia.org/wiki/Mega_Man_Star_Force

Sample Screens

Scenario 1

					
The positions of the player and enemy pieces.	In the player's turn, the player moves to the right.	During the enemy's turn, the enemies move down.	Player selects to shoot and so destroys the enemy in front of it and scores 10 points.	It then respawns at the top row.	During the enemy's turn, the enemy now moves.

Scenario 2

					
The positions of the player and enemy pieces.	In the player's turn, the player fires, scores a point and the enemy respawns.	During the enemy's turn, the enemy moves.	Player selects to shoot down two enemies and scores twice.	The destroyed enemies respawn beside each other.	When the enemy moves down, the player loses.

For full sample runs of the game, these can be found in Appendix B.

Project overview

This game is to be developed in the C programming language. There are several rules when developing this application.

- **Arrays are not permitted to be used for this project**. Use of arrays will result in a zero grade.
- When a new turn is made, the screen output is printed below the text of the game. There is no need to clear the screen for every action.
- **The use of scanf for input is required**. Do not use methods that get real-time input.
    - The use of kbhit() and getch() are not permitted.
- The use of the ASCII art borders for the game is optional. Use of it in the game will not count as bonus points.
- A menu screen is optional but permitted.
- More complicated AI for the enemies are not required.
- Input validation is required. However, assume that the player will only input whole numbers.
    - When a move is invalid (even moving beyond boundaries count as an invalid move), the turn does not count and the enemy does not move.

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. For topics not covered, it is left to the student to read ahead, research, and explore by him/her-self.

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

- To use arrays (static or dynamic, both are not allowed),
- 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),
- 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. This is also called a recursion.

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 other 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?
7. and others…

Important Reminders

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 <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.**)
   - Consistently employ coding conventions
   - Include internal documentation (i.e., comments)
3. Deadline for the project is the **8:00AM of December 5, 2022 (Monday)** via submission through **AnimoSpace**. After this time, submission facility is locked and thus 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> - <ID Number> - <Section>
*******************************************************************************/
```

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.

Function	#	Description	Sample Input Data	Expected Output	Actual Output	P/F
addTwoNumbers	1	Two positive numbers	numA: 5 numB: 10	15	15	P
	2	Zero and a positive number.	numA: 0 numB: 7	7	7	P
	3	Two negative numbers.	numA: -2 numB: -3	-5	5	F

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

1. Testing where both inputs are positive integer inputs.
2. Testing where one number has the value zero and the other is a positive number.
3. Testing where both inputs are negative numbers.

The following test descriptions are incorrectly formed:

- Too specific: testing with nNumA containing 5 and nNumB containing 10
- Too general: testing if function can generate correct sum of two numbers
- Not necessary -- since already defined in pre-condition: testing with nNumA or nNumB 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 the questions convincingly 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 the running program). Thus, if the program does not compile successfully using gcc -Wall 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 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 a bonus.
    - 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 other people that are not online). **You should develop your own codes from scratch by yourself.**

Appendix A: Questions and Clarifications

Instruction Clarifications

1. What counts as brute force for this project? (See Important Reminders point 2, subpoint 2)
    - Hard coding every single enemy position per turn
    - Hard coding every single bullet action based on the turn number
    - Anything else in a similar vein of logic flow where every single combination is programmed rather than an elegant solution.
2. Can we ask help from classmates, relatives and experts?
    - This is not allowed. Even when it is mechanics that are asked, the line between cheating and asking for help is a thin one. Ask the teacher when stuck.
3. Can I share or look at other people's work?
    - Do not share any work until the end of the term once the grades are given.
4. Where can we find the rubric for grading?
    - The general rubric can be found in Canvas or in Appendix C.
    - The more specific breakdown of the rubric (such as the contents of Program Correctness) will only be shown during the demo.
5. N/A

Mechanic Clarifications

6. What counts as bonus points?
    - Only killing one enemy when a laser is fired.
    - Creating power ups that create score multipliers for combos.
    - Random generation spawning of units at the top row.
7. Do all enemies follow the same movement pattern?
    - No, each enemy will have its own movement pattern. This is because the movement pattern is reset whenever an enemy dies and respawns.
8. What if two enemies overlap, what will the program do?
    - Allow the enemies to overlap and only draw one enemy. Do not add collision detection.
9. Why is it that in the sample runs the enemies spawn at the 2nd column when it is destroyed?
    - This is because the player and enemy turn happened in succession. Please look at Sample Screens Scenario 1 for a step by step execution of steps from one turn to another.
10. Do we need to restart the game once victory or defeat is reached?
    - Not a requirement but you may do so.
11. N/A

Coding Clarifications

12. Are we allowed to use structures "type def" or classes?
    - Classes are not allowed in C but C++ and structures are permitted but it will not be needed.
13. N/A

Delivery Clarifications

14. N/A

Appendix B: Sample Screen output

The following are sample runs for the program. The exact interface and spacings of the prompts need not be followed for as long as there are prompts shown. The output is read from left to right then top to bottom.

Run 1

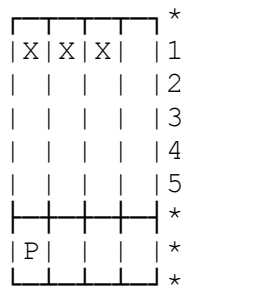
```
┌─┬─┬─┬─┬─┐*
|X|X|X| |1
| | | | |2
| | | | |3
| | | | |4
| | | | |5
├─┼─┼─┼─┼─┤*
|P| | | |*
└─┴─┴─┴─┴─┘*
Score: 000
Actions:
 1 - move left
 2 - move right
 3 - fire laser
Input action: 2
┌─┬─┬─┬─┬─┐*
| |X|X|X|1
| | | | |2
| | | | |3
| | | | |4
| | | | |5
├─┼─┼─┼─┼─┤*
| |P| | |*
└─┴─┴─┴─┴─┘*
Score: 000
Actions:
 1 - move left
 2 - move right
 3 - fire laser
Input action: 2
┌─┬─┬─┬─┬─┐*
| | | | |1
| |X|X|X|2
| | | | |3
| | | | |4
| | | | |5
├─┼─┼─┼─┼─┤*
| | |P| |*
└─┴─┴─┴─┴─┘*
Score: 000
```

```
Actions:
 1 - move left
 2 - move right
 3 - fire laser
Input action: 2
┌─┬─┬─┬─┬─┐*
| | | | |1
|X|X|X| |2
| | | | |3
| | | | |4
| | | | |5
├─┼─┼─┼─┼─┤*
| | | |P|*
└─┴─┴─┴─┴─┘*
Score: 000
Actions:
 1 - move left
 2 - move right
 3 - fire laser
Input action: 1
┌─┬─┬─┬─┬─┐*
| | | | |1
| | | | |2
|X|X|X| |3
| | | | |4
| | | | |5
├─┼─┼─┼─┼─┤*
| | |P| |*
└─┴─┴─┴─┴─┘*
Score: 000
Actions:
 1 - move left
 2 - move right
 3 - fire laser
Input action: 1
┌─┬─┬─┬─┬─┐*
| | | | |1
| | | | |2
| |X|X|X|3
| | | | |4
| | | | |5
├─┼─┼─┼─┼─┤*
| |P| | |*
└─┴─┴─┴─┴─┘*
Score: 000
```

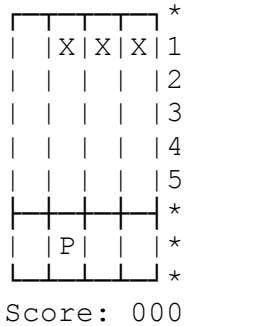
```
Actions:
 1 - move left
 2 - move right
 3 - fire laser
Input action: 3
┌─┬─┬─┬─┬─┐*
| |X| | |1
| | | | |2
| | | | |3
| | |X|X|4
| | | | |5
├─┼─┼─┼─┼─┤*
| |P| | |*
└─┴─┴─┴─┴─┘*
Score: 010
Actions:
 1 - move left
 2 - move right
 3 - fire laser
Input action: 3
┌─┬─┬─┬─┬─┐*
| |X| | |1
| | | | |2
| | | | |3
| |X|X| |4
| | | | |5
├─┼─┼─┼─┼─┤*
| |P| | |*
└─┴─┴─┴─┴─┘*
Score: 020
Actions:
 1 - move left
 2 - move right
 3 - fire laser
Input action: 3
┌─┬─┬─┬─┬─┐*
| |X|X| |1
| | | | |2
| | | | |3
| | | | |4
| | |X| |5
├─┼─┼─┼─┼─┤*
| |P| | |*
└─┴─┴─┴─┴─┘*
Score: 040
```

```
Actions:
 1 - move left
 2 - move right
 3 - fire laser
Input action: 3
┌─┬─┬─┬─┬─┐*
| |X| | |1
| | |X| |2
| | | | |3
| | | | |4
| | | |X|5
├─┼─┼─┼─┼─┤*
| |P| | |*
└─┴─┴─┴─┴─┘*
Score: 050
Actions:
 1 - move left
 2 - move right
 3 - fire laser
Input action: 3
┌─┬─┬─┬─┬─┐*
| |X| | |1
| |X| | |2
| | | | |3
| | | | |4
| | | | |5
├─┼─┼─┼─┼─┤*
| |P| | |*
└─┴─┴─┴─┴─┘*
DEFEATED!
```

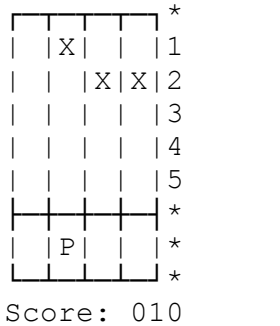
Run 2



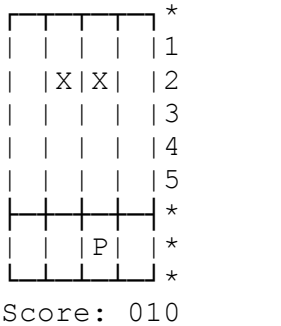

```
Score: 000
Actions:
 1 - move left
 2 - move right
 3 - fire laser
Input action: 2
```



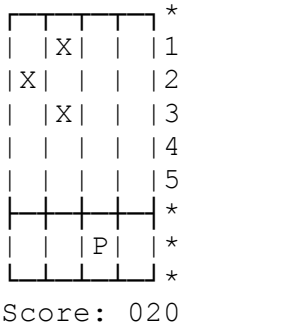

```
Score: 000
Actions:
 1 - move left
 2 - move right
 3 - fire laser
Input action: 3
```



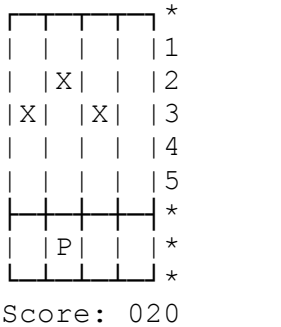

```
Score: 010
Actions:
 1 - move left
 2 - move right
 3 - fire laser
Input action: 2
```




```
Score: 010
Actions:
 1 - move left
 2 - move right
 3 - fire laser
Input action: 3
```

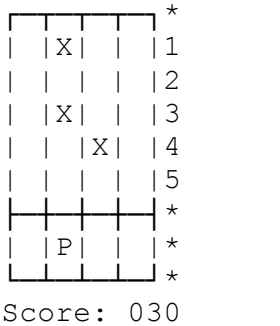



```
Score: 020
Actions:
 1 - move left
 2 - move right
 3 - fire laser
Input action: 1
```

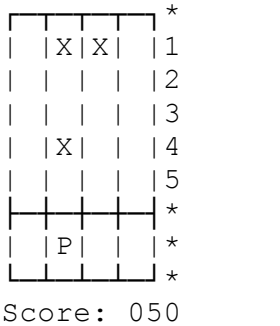



```
Score: 020
```

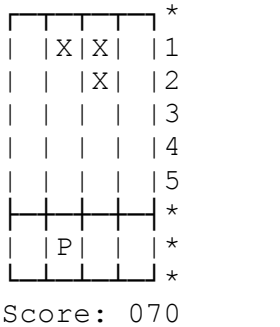
```
Actions:
 1 - move left
 2 - move right
 3 - fire laser
Input action: 3
```




```
Score: 030
Actions:
 1 - move left
 2 - move right
 3 - fire laser
Input action: 3
```

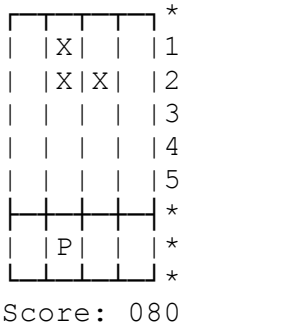



```
Score: 050
Actions:
 1 - move left
 2 - move right
 3 - fire laser
Input action: 3
```

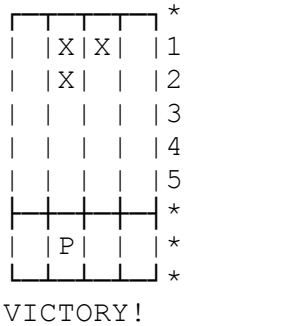



```
Score: 070
```

```
Actions:
 1 - move left
 2 - move right
 3 - fire laser
Input action: 3
```




```
Score: 080
Actions:
 1 - move left
 2 - move right
 3 - fire laser
Input action: 3
```




```
VICTORY!
```

Appendix C: General Rubric

The general rubric can be found in the Rubrics page in the Canvas course.

CCPROG1 MP						
Criteria	**Ratings**					**Pts**
Program Correctness	**75 to >40.0 pts** **Exemplary** The application meets all the requirements specified in the project specification. The code is syntactically and logically correctly for all cases. Implementation of the program follows the indicated guidelines and does not violate indicated restrictions. The implementation also exhibits appropriate use of programming constructs.	**40 to >25.0 pts** **Satisfactory** The code works for typical input, but fails for minor special cases; the major requirements are met, though some minor ones are not. Some implementation of the program violates indicated restrictions.	**25 to >10.0 pts** **Developing** The code sometimes fails for typical input. Many parts of the program implementation violate indicated restrictions and some parts of the solution are not implemented using appropriate programming constructs.	**10 to >0.0 pts** **Beginning** The code often fails, even for typical input. Most indicated restrictions were violated. Note: Program that does not run and /or implemented incorrectly (based on specifications and restrictions) automatically gets 0 for this course output.	**0 pts** **No submission or did not compile successfully**	75 pts
Readability	**10 to >8.0 pts** **Exemplary** The program conforms to a coding standard that promotes code readability. Internal documentation is comprehensive. Note: See references for some relevant information on code readability.	**8 to >5.0 pts** **Satisfactory** Minor code formatting does not exhibit consistency in coding standard. Not all functions / program features have proper internal documentation.	**5 to >0.0 pts** **Developing** Minimal internal documentation and code readability.	**0 pts** **Beginning or None** No internal documentation and code is not readable.		10 pts
Effective Communication / Concept Understanding	**5 to >3.0 pts** **Exemplary** Answers to questions are correct, reasonable, and reflective of the code. The justifications provided are sound.	**3 to >2.0 pts** **Satisfactory** Answers to questions are correct, but some justifications provided are weak.	**2 to >1.0 pts** **Developing** Answers to questions are correct, but cannot justify solution (e.g., solution via trial and error, rather than proper understanding and application of concepts).	**1 to >0 pts** **Beginning** Correct understanding of the problem, but was unable to explain workings of code provided. Note: Failure to explain and justify workings of the code submitted will automatically merit 0 for this course output.		5 pts
Test Cases	**10 to >8.0 pts** **Exemplary** All test cases are indicated correctly and completely with correctly formed descriptions.	**8 to >5.0 pts** **Satisfactory** All test cases are indicated correctly but some are results are incomplete or missing, or some descriptions are general, or some cases are missing.	**5 to >0.0 pts** **Developing** Test cases are indicated but most cases are missing, or results of most cases are missing, or descriptions are too general.	**0 pts** **Beginning or None** No test case document submitted.		10 pts
Bonus	**10 to >0.0 pts** **Bonus features**	**0 pts** **No bonus feature implemented**				10 pts
						Total Points: 110