VIETNAM NATIONAL UNIVERSITY - HO CHI MINH CITY
HO CHI MINH CITY UNIVERSITY OF TECHNOLOGY
FACULTY OF COMPUTER SCIENCE AND ENGINEERING

PROGRAMMING FUNDAMENTALS - CO1027

Assignment 1

HOW TO TRAIN YOUR DRAGON (PART I)

Version 1.0

Ho Chi Minh City, 07/2025

ASSIGNMENT SPECIFICATION

Version 1.0

1 Learning Outcomes

After completing this assignment, students will review and master the following concepts:

- Conditional structures
- Loop structures
- One-dimensional and two-dimensional arrays
- String processing
- Functions and function calls

2 Introduction

This assignment is adapted from the first part of the movie How to Train Your Dragon produced by DreamWorks, based on the famous book series of the same name by Cressida Cowell.

The story takes place on a remote, cold, and foggy island in the northern seas, where the Viking people have lived for many generations. This village is called Berk, a place that endures harsh storms all year round. The people of Berk are known as strong and brave warriors, always ready to defend themselves against sudden attacks from dragons — their long-time enemies.

The main character of the story, Hiccup Horrendous Haddock III, is the only son of the chief Stoick the Vast. While everyone expects him to become a strong warrior like his father, Hiccup is skinny, weak, and often causes trouble with his strange inventions. However, it is exactly his intelligence, creativity, and different way of thinking that change everything.

During a dragon attack one night, Hiccup uses his own invention to successfully shoot down a mysterious and extremely dangerous dragon — the Night Fury, also known to the villagers as the "nightmare." When Hiccup finds it, instead of killing it like Viking tradition requires, he decides to heal and take care of it. Little by little, Hiccup and the dragon — whom he names Toothless — build a relationship based on trust and mutual respect.

While training and learning together, Hiccup realizes that dragons are not as cruel as the people of Berk have always believed. In fact, they are forced to steal food to bring to a much larger and more frightening dragon. When Hiccup discovers this truth, he and Toothless decide to change the village's mindset, proving that dragons can become loyal, brave, and even lovable companions.

Besides Hiccup, there is also a group of close friends: Astrid — a strong and determined girl; Snotlout — confident and reckless; Fishlegs — gentle and knowledgeable about dragons; and the mischievous twins Ruffnut and Tuffnut. They not only join the tough training sessions in the Dragon Arena together, but also gradually learn to bond, cooperate, and discover their own hidden strengths and those of the dragons.

From tense battles to valuable lessons about friendship, courage, and empathy, the story opens up an inspiring adventure. And from that journey, a new future for Berk begins — where humans and dragons are no longer enemies, but become warriors standing side by side to protect their beloved village.

3 Input Data

3.1 Description of Input Data

The input data of the program is stored in a text file (.txt) in the line format of Windows, Linux, or macOS. The file name is stored in the variable `file_input`. This file contains the following information, in order line by line:

```
["Toothless", "Stormfly", "Hookfang", "Meatlug", "Barf and Belch"]
["Night Fury", "Deadly Nadder", "Monstrous Nightmare", "Gronckle", "Hideous Zippleback"]
[9; 8; 7; 7; 6]
[3; 2; 5; 1; 4]
[100; 80; 90; 70; 95]
["Hiccup", "Astrid", "Snotlout", "Fishlegs", "Ruff and Tuff"]
5
```

The input file sequentially and orderly represents detailed information of N dragons, including: name, dragon type, temperament, number of projectiles it can shoot, and damage. Specifically as follows:

- **Line 1 - `dragon_names`**: A list of dragon names in order, represented as strings.
- **Line 2 - `dragon_types`**: A list of dragon types corresponding to each dragon, represented as strings. Valid values include:
  - `"Night Fury"` $\rightarrow$ 1
  - `"Deadly Nadder"` $\rightarrow$ 2

  – `"Monstrous Nightmare"` → 3
  – `"Gronckle"` → 4
  – `"Hideous Zippleback"` → 5

  If the dragon type does not match any of the valid values above, it is mapped by default to type "Night Fury".

- **Line 3 -** `dragon_temperament`: A list of temperament scores for each dragon, represented as integers. Each value must fall within the range $[0, 10]$. If a value is out of bounds, it is clamped to the nearest valid limit.
- **Line 4 -** `ammo_counts`: A list of the number of shots each dragon can fire in a single charge, represented as integers. Each value must fall within the range $[0, 1000]$. If a value is out of bounds, it is clamped to the nearest valid limit.
- **Line 5 -** `dragon_damages`: A list of damage values corresponding to the five dragon types, represented as integers. Each value must be in the range $[0, 1000]$. If a value is out of bounds, it is clamped to the nearest valid limit. The list must contain exactly 5 elements, representing types 1 through 5.
- **Line 6 -** `rider_names`: A list of rider names corresponding to each dragon, represented as strings.
- **Line 7 -** `N`: The number of dragons.

Note: All string elements in the input arrays are only allowed to contain the following characters: uppercase and lowercase letters (A–Z, a–z), digits (0–9), spaces, and the 8 special characters (`! @ # $ % ^ & *`).

4 Tasks

4.1 Task 1: Read input data (3.0 points)

Before proceeding with training and battle tasks, dragon trainers need to collect and process information about dragon individuals, their types, ammo counts, damage values, and rider names. The first task is to implement a function that reads and processes data from the input file.

Function Description

- **Function name**:

```
int readFile(const string filename,
                Dragon dragons[],
                int dragonDamages[5],
                int &N);
```

- **Objective**: Read data from the input file and store information for each dragon individual into the `dragons` array, as well as read the `dragonDamages` array. The variable `N` will be assigned the actual number of dragons after reading.
- **Data structure**:

```
struct Dragon {
    string name;
    int type;
    int temperament;
    int ammoCount;
    string riderName;
};
```

- **Input parameters**:
  - `filename`: Name of the input data file.
  - `dragons[]`: Array storing dragon individuals (name, type, temperament, ammo count, and rider name).
  - `dragonDamages[5]`: Array storing the damage of 5 dragon types (from type 1 to 5).
  - `N`: Will be assigned the actual number of dragons after file reading.
- **Return value**: A success or error code, defined as follows:
  1. 1: Success
  2. 2: File does not end with `.txt`
  3. 3: File not found
  4. 4: Missing line
  5. 5: `dragonNames[]` array has missing or extra elements
  6. 6: `dragonTypes[]` array has missing or extra elements
  7. 7: `dragonTemperament[]` array has missing or extra elements
  8. 8: `ammoCounts[]` array has missing or extra elements
  9. 9: `dragonDamages[5]` does not contain exactly 5 elements
  10. 10: `riderNames[]` array has missing or extra elements

11. `100 + i`: `i` is the index of the 2nd element in `dragonNames[]` that contains special characters
12. `500 + k`: `k` is the total number of special characters in all elements of `dragonTypes[]`
13. `900 + i`: `i` is the index of the first element in `riderNames[]` that contains 2 or more consecutive spaces
14. `1000 + k`: `k` is the total number of extra spaces (2 or more consecutive) in `dragonNames[]`

- **Assumption**: If the input file has an error, it contains only one error from the above list. Other types of errors are not considered.
- **Note**: If students fail to complete this task correctly, resulting in corrupted or improperly formatted dragon data, then all subsequent tasks that use such data will be penalized **50% of the points**.

Example 4.1

Success Example (Return 1)

Input data:

```
["Toothless",␣"Stormfly",␣"Hookfang",␣"Meatlug",␣"Barf␣and␣Belch"]
["Night␣Fury",␣"Deadly␣Nadder",␣"Monstrous␣Nightmare",␣"Gronckle",␣"Hideous␣Zippleback"]
[9;␣8;␣7;␣7;␣6]
[3;␣2;␣5;␣1;␣4]
[100;␣80;␣90;␣70;␣95]
["Hiccup",␣"Astrid",␣"Snotlout",␣"Fishlegs",␣"Ruff␣and␣Tuff"]
5
```

Return value: 1

Example 4.2

Example with missing line (Return 4)

Input data:

```
["Toothless",␣"Stormfly",␣"Hookfang",␣"Meatlug",␣"Barf␣and␣Belch"]
["Night␣Fury",␣"Deadly␣Nadder",␣"Monstrous␣Nightmare",␣"Gronckle",␣"Hideous␣Zippleback"]
[9;␣8;␣7;␣7;␣6]
[3;␣2;␣5;␣1;␣4]
[100;␣80;␣90;␣70;␣95]
["Hiccup",␣"Astrid",␣"Snotlout",␣"Fishlegs",␣"Ruff␣and␣Tuff"]
```

Return value: 4

Example 4.3

Example with missing element in dragonNames (Return 5)

Input data:

```
["Toothless",␣"Stormfly",␣"Hookfang",␣"Meatlug"]
["Night␣Fury",␣"Deadly␣Nadder",␣"Monstrous␣Nightmare",␣"Gronckle",␣"Hideous␣Zippleback"]
[9;␣8;␣7;␣7;␣6]
[3;␣2;␣5;␣1;␣4]
[100;␣80;␣90;␣70;␣95]
["Hiccup",␣"Astrid",␣"Snotlout",␣"Fishlegs",␣"Ruff␣and␣Tuff"]
5
```

Return value: 5

Example 4.4

Example with invalid dragonDamages (Return 9)

Input data:

```
["Toothless",␣"Stormfly",␣"Hookfang",␣"Meatlug",␣"Barf␣and␣Belch"]
["Night␣Fury",␣"Deadly␣Nadder",␣"Monstrous␣Nightmare",␣"Gronckle",␣"Hideous␣Zippleback"]
[9;␣8;␣7;␣7;␣6]
[3;␣2;␣5;␣1;␣4]
[100;␣80;␣90;␣70]
["Hiccup",␣"Astrid",␣"Snotlout",␣"Fishlegs",␣"Ruff␣and␣Tuff"]
5
```

Return value: 9

Example 4.5

Example with special character in 2nd dragon name (Return 101)

Input data:

```
["Toothless",␣"Storm@fly",␣"Hookfang",␣"Meatlug",␣"Barf␣and␣Belch"]
["Night␣Fury",␣"Deadly␣Nadder",␣"Monstrous␣Nightmare",␣"Gronckle",␣"Hideous␣Zippleback"]
[9;␣8;␣7;␣7;␣6]
[3;␣2;␣5;␣1;␣4]
[100;␣80;␣90;␣70;␣95]
["Hiccup",␣"Astrid",␣"Snotlout",␣"Fishlegs",␣"Ruff␣and␣Tuff"]
5
```

Return value: 101

Example 4.6

Example with 3 total special characters in dragonTypes (Return 503)

Input data:

```
["Toothless",␣"Stormfly",␣"Hookfang",␣"Meatlug",␣"Barf␣and␣Belch"]
["Night\$␣Fury",␣"Deadly!␣Nadder",␣"Monstrous␣Nightmare",␣"Gr@onckle",␣"Hideous␣Zippleback"]
[9;␣8;␣7;␣7;␣6]
[3;␣2;␣5;␣1;␣4]
[100;␣80;␣90;␣70;␣95]
["Hiccup",␣"Astrid",␣"Snotlout",␣"Fishlegs",␣"Ruff␣and␣Tuff"]
5
```

Return value: 503

Example 4.7

Example with double spaces in rider name (Return 904)

Input data:

```
["Toothless",␣"Stormfly",␣"Hookfang",␣"Meatlug",␣"Barf␣and␣Belch"]
["Night␣Fury",␣"Deadly␣Nadder",␣"Monstrous␣Nightmare",␣"Gronckle",␣"Hideous␣Zippleback"]
[9;␣8;␣7;␣7;␣6]
[3;␣2;␣5;␣1;␣4]
[100;␣80;␣90;␣70;␣95]
["Hiccup",␣"Astrid",␣"Snotlout",␣"Fishlegs",␣"Ruff␣␣and␣Tuff"]
5
```

Return value: 904

Example 4.8

Example with 3 total extra spaces in dragonNames (Return 1003)

Input data:

```
["Toothless",␣"Stormfly",␣"Hookfang",␣"Meat␣␣lug",␣"Barf␣␣␣and␣Belch"]
["Night␣Fury",␣"Deadly␣Nadder",␣"Monstrous␣Nightmare",␣"Gronckle",␣"Hideous␣Zippleback"]
[9;␣8;␣7;␣7;␣6]
[3;␣2;␣5;␣1;␣4]
[100;␣80;␣90;␣70;␣95]
["Hiccup",␣"Astrid",␣"Snotlout",␣"Fishlegs",␣"Ruff␣and␣Tuff"]
5
```

Return value: 1003

4.2 Task 2: Find the strongest dragon above temperament threshold (1.0 point)

In this task, students must implement a function to identify the name of the dragon with the **highest power** among those whose **temperament** is greater than or equal to a threshold `T`. Dragon information is stored in the `Dragon` struct array, including type, ammo count, temperament, and rider name.

Function Description

- **Function name**:

```
string findKthStrongestDragon(Dragon dragons[],
                              int dragonDamages[5],
                              int N,
                              int T);
```

- **Input parameters**:
  - `dragons[]`: Array containing dragon individuals.
  - `dragonDamages[5]`: Array containing the damage corresponding to each dragon type (type 1 to 5).
  - `N`: Number of dragons in the array.
  - `T`: Minimum threshold of the temperament value.
- **Return value**:
  - Name of the dragon with the highest power among those with `temperament` $\geq$ `T`.
  - If no dragon meets the condition, return the string `"None"`.

Power Calculation Formula

- **Night Fury** (type = 1):
  `power = (ammo × damage) + temperament × 3`
- **Deadly Nadder** (type = 2):
  `power = (ammo × damage) + temperament × 2`
- **Monstrous Nightmare** (type = 3):
  `power = (ammo × damage) + temperament`
- **Gronckle** (type = 4):
  `power = (ammo × damage) + temperament / 2`

- **Hideous Zippleback** (type = 5):
  `power = (ammo × damage × 0.9) + temperament × 1.5`

Note

- If multiple dragons have the same maximum power, the one appearing first in the array takes precedence.
- Only return the dragon's name as a string.

Example 4.9

Example 1: A qualified dragon exists, return the strongest one

Input data:

```
["Toothless",␣"Stormfly",␣"Hookfang",␣"Meatlug",␣"Barf␣and␣Belch"]
[1;␣2;␣3;␣4;␣5]
[9;␣8;␣7;␣4;␣6]
[3;␣2;␣5;␣1;␣4]
[100;␣80;␣90;␣70;␣95]
["Hiccup",␣"Astrid",␣"Snotlout",␣"Fishlegs",␣"Ruff␣and␣Tuff"]
5
```

With `T = 6`, the power of qualified dragons:

- Toothless (type 1): $(3100) + 93 = 327$
- Stormfly (type 2): $(280) + 82 = 192$
- Hookfang (type 3): $(590) + 7 = 457$
- Barf and Belch (type 5): $(4950.9) + 61.5 = 375$

Meatlug is excluded because temperament < 6.

Return value: `"Hookfang"`

Example 4.10

Example 2: No qualified dragons with temperament $\geq$ T

Input data:

```
["Toothless",␣"Stormfly",␣"Hookfang"]
[1;␣2;␣4]
[3;␣2;␣4]
[1;␣2;␣3]
[100;␣80;␣70;␣60;␣50]
["Hiccup",␣"Astrid",␣"Snotlout"]
3
```

With `T = 6`, no dragons qualify.

Return value: `"None"`

4.3 Task 3.1: Evaluate compatibility (1 point)

At the Fire Arena, warriors must prepare a strong dragon team to compete. However, a strong dragon does not necessarily mean it can be tamed. Each dragon has its own temperament (hot-tempered, cold, wise, etc.), while the warrior has their own riding and controlling skills. If the compatibility score is too low, the dragon will resist and not obey commands.

4.3.1 Function Description

- **Function Name**:

```
void compatibilityCheck(Dragon dragons[], string warriorName,
                        int warriorSkill);
```

- **Input Parameters**:
  - `dragons[]`: An array containing information about the dragon individuals (name, type, temperament, ammo count, and rider).
  - `warriorName`: Name of the warrior.
  - `warriorSkill`: Skill score of the warrior.
- **Return result**: Print the compatibility table of the warrior with all current dragon individuals.

4.3.2 Classification rules

Formula:

```
compatibility = (10 - | dragon.temperament - warrior.skill |) / 2
```

- **Not Compatible:** compatibility ≤ 4
- **Compatible:** compatibility > 4

4.3.3 Output format

Example 4.11

The program needs to print in the following format:

```
Warrior       Dragon        Compatibility       Review
Hiccup        Toothless     5.00                Compatible
Hiccup        Stormfly      3.95                Not Compatible
...
```

Students will be provided with the function `printCompatibilityTable()` to print the table in the above format.

4.4 Task 3.2: Find a buddy (1 point)

The Warriors team begins to look for dragons suitable for each member. To be able to join the battle together, each warrior needs to find the dragon with the highest compatibility score for themselves. However, not every dragon can be chosen multiple times: each dragon can only be matched with exactly one warrior.

4.4.1 Function Description

- **Function Name**:

```
void buddyMatching(Dragon dragons[], string warriors[][3]);
```

- **Input Parameters**:

  - `dragons[]`: An array containing information about the dragon individuals (name, type, temperament, ammo count, and rider).
  - `warriors`: List of warriors. Each warrior includes 3 pieces of information: Name, skill score, and ID.
- **Return value**: Print out the pairing result table of Warrior - Dragon.

4.4.2 Detailed Description

Progress:

- Calculate the compatibility score between each warrior and each dragon. Students will use the function `compatibilityCheck()` to support this task.
- For each warrior (considered in the order they appear in the `warriors` list), select the dragon with the highest compatibility score from the available dragons.
- Once a dragon has been selected by a warrior, it cannot be chosen again by the following warriors.

Selection Rules:

- Each dragon can only be assigned to at most one warrior.
- Priority is given to selecting the dragon with the highest compatibility score.
- If there are multiple dragons with the same highest score, the one appearing earlier in the list is preferred.

4.4.3 Output format

Example 4.12

The program needs to print in the following format:

```
Warrior       Dragon        Compatibility       Review
Hiccup        Toothless     5.00                Compatible
Astrid        Stormfly      4.95                Compatible
Snotlout      Hookfang      5.00                Compatible
Fishlegs      Meatlug       4.75                Compatible
RuffNTuff     BarfBelch     4.60                Compatible
```

Students will be provided with the function `printCompatibilityTable()` to print the table in the above format.

4.5 Task 4: ROUND 1 - THE SYNERGY TEST (1 point)

After each warrior has chosen a suitable dragon, the Warriors team officially enters Round 1 at the Fire Arena. In this round, the warrior-dragon pairs will participate in a challenge to test their compatibility before being allowed to advance to the next round.

Objective: Write a function to calculate the total time that each warrior needs to complete the challenge (including the time to go, the time to return, and the time to move between the cells).

4.5.1 Function Description

- **Function Name**:

```
void computeChallengeTime(string warriors[][3], int map[10][10]);
```

- **Input Parameters**:
  - `warriors`: List of warriors. Each warrior includes 3 pieces of information: Name, skill points, and ID.
  - `map`: The map of the challenge.
- **Return value**: Print a table showing the compatibility levels of the warrior with all existing dragon individuals.

4.5.2 Detailed Description

Challenge context: The pairs will have to fly across a treasure map built in the forest behind the Berk village mountain. The map is divided into 100 cells (10 x 10), each cell representing a small area.

Rules for the map:

- Each cell can contain an item with a code matching a specific warrior (code from 0 to 4), or it might not contain any item for anyone.
- Each cell has an ID calculated using the formula:

```
ID = (row + col) % 5
```

- Warriors are only allowed to collect items that have a code matching their own code.
- Additionally, warriors cannot fly into cells whose ID equals their own code, regardless of whether the cell has an item or not.

Item collection process:

- Each time a warrior finishes picking up an item, they must return to the starting position to drop the item before going back out to continue.
- If the current cell does not contain an item, the warrior simply flies to the next cell without needing to return.

Rules for calculating movement time:

- Time from the starting position to the cell containing the item (seconds):

```
time = [1 + (row + col * 2)] * 5
```

- Time flying back from that cell to the starting position:

```
time = [1 + (row + col * 2)] * 5
```

- If not picking up an item, it only takes 5 seconds to move to the next cell.

Example 4.13

Example of the map (ID in each cell):

```
0 1 2 3 4 0 1 2 3 4
1 2 3 4 0 1 2 3 4 0
2 3 4 0 1 2 3 4 0 1
3 4 0 1 2 3 4 0 1 2
4 0 1 2 3 4 0 1 2 3
0 1 2 3 4 0 1 2 3 4
1 2 3 4 0 1 2 3 4 0
2 3 4 0 1 2 3 4 0 1
3 4 0 1 2 3 4 0 1 2
4 0 1 2 3 4 0 1 2 3
```

4.5.3 Output format

- A summary table of completion times for all warriors.
- Select the 4 fastest warriors (eliminate the slowest one).
- Print this table sorted by time from smallest to largest.

Example 4.14

```
Warrior       Total time (secs)
Hiccup        123
Fishlegs      130
Astrid        150
Snotlout      175
```

4.6 Task 5.1: Evaluate the damage dealt (1 point)

After completing the Synergy Test round, the warriors and their dragons enter round 2 to assess their attacking ability. Each warrior will coordinate with their dragon to calculate the maximum damage they can inflict on the opponent. The damage score is calculated based on the dragon's base damage, the number of attacks, and the warrior's skill. This is an important factor to evaluate the true combat capability of each pair.

4.6.1 Function Description

- **Function Name**:

```
void fighterDamage(Dragon dragons[], string warriors[][3], int teamsDamage[]);
```

- **Input Parameters**:
  - `dragons[]`: An array containing information about the dragon individuals (name, type, temperament, ammo count, and rider).
  - `warriors`: List of warriors. Each warrior includes 3 pieces of information: Name, skill points, ID.
  - `teamsDamage`: List of the warriors' power values.
- **Return value**: Store the damage information that each Warrior-Dragon team can deal into the array `teamsDamage`. At the same time, print it out in the format shown below.

4.6.2 Formula:

```
damage = (dragon.damage * dragon.ammoCount) + (warrior.skill * 5)
```

4.6.3 Output format

Example 4.15

The program should print in the following format:

```
Hiccup-Toothless: damage = 125
Astrid-Stormfly: damage = 98
...
```

4.7 Task 5.2: ROUND 2 – SEARCHING FOR BERK'S LEGACY (2 points)

(Task 5.2 is expected to be updated for students within the next 2 days and no later than July 16, 2025.)

4.8 Conclusion

After many days of grueling training and intense battles, the young warriors of Berk have finally found the strongest and most loyal dragon companions. Together, they not only overcame strength tests, riding skills, and teamwork challenges, but also proved the steely will and courage worthy of true Viking warriors. The perfect combination between the warriors and their dragons created spectacular aerial maneuvers, powerful attacks, and an unbreakable spirit of unity. However, the real journey has only just begun. Ahead of them lie harsher trials, mysterious opponents from the cold northern seas, and strange creatures never seen before. Will they be strong enough to protect their tribe, explore new lands, and continue writing the legend of Berk? Get ready and stay tuned for part 2 of this grand assignment to join them in thrilling and unexpected adventures waiting just beyond the horizon!

Enjoy working on your Assignment!

5 Requirements

To complete this assignment, students must:

1. Read this entire specification document.
2. Download the file initial.zip and extract it. After extraction, students will receive the following files: main.cpp, main.h, dragon.h, dragon.cpp, and sample data files for reading. Students must submit only two files, dragon.h and dragon.cpp, and must not modify the file main.h when running or testing the program.
3. Students must use the following command to compile the program:
   g++ -o main main.cpp dragon.cpp -I . -std=c++11
   Students must use the following command to run the program:
   ./main tnc_tc_01_input.txt
   The above commands are used in the command prompt/terminal to compile and run the program. If students use an IDE to run the program, they need to pay attention to the following: include all necessary files in the project/workspace of the IDE; modify the IDE's compile command accordingly. IDEs usually provide buttons for building (Build) and running (Run) the program. When clicking Build, the IDE will execute a corresponding compile command, which typically only compiles main.cpp. Students need to find a way to configure the IDE to change the compile command: add dragon.cpp, add the option -std=c++11, and include -I .

4. The program will be evaluated on a Unix-based platform. The evaluation environment and the student's local compiler might differ from the actual grading environment. The submission platform on BKeL is configured to match the actual grading environment. Students must test their programs on the BKeL submission system and fix all errors that occur there to ensure correct results when graded.
5. Modify the files dragon.h and dragon.cpp to complete this project and ensure the following two requirements are met:
   - There is only one **include** statement in the file `dragon.h`, which is:

     #include "main.h"

   - In the file `dragon.cpp`, there is only one include statement:

     #include "dragon.h"

   - Apart from the two includes above, no other **#include** statements are allowed in these files.
   - Implement the functions described in the tasks of this project assignment.
6. Students are encouraged to write additional functions to complete this project assignment.

6 Submission

Students only need to submit two files: dragon.h and dragon.cpp, before the deadline specified in the "Assignment 1 - Submission" link. There will be some simple test cases used to check students' submissions to ensure that the results can be compiled and executed. Students can submit as many times as they want, but only the last submission will be graded. Because the system cannot handle too many submissions at the same time, students are encouraged to submit as early as possible. Students will take full responsibility if they submit close to the deadline (within one hour before the deadline). After the submission deadline, the system will be closed and students will not be able to submit anymore. Submissions via other methods will not be accepted.

7 Additional Regulations

1. Students must complete this assignment on their own and must not allow others to copy or steal their results. Otherwise, students will be subject to the university's regulations on academic dishonesty.

2. All decisions made by the instructor in charge of the assignment are final.
3. Students will not be provided with the test cases after grading.
4. The content of this assignment will be harmonized with some questions in the Final Exam with similar topics.
5. If students use code generated by AI programs or automatic code generation tools without fully understanding how it works and its meaning, this assignment score will be zero.

8 Harmony for the Assignment

Some questions in the final exam will be directly related to this assignment.

Students must complete this assignment by themselves. If a student cheats in this assignment, they will be unable to answer related Harmony questions in the final exam and will receive a score of 0 for the assignment.

Students **must** answer the Harmony questions in the final exam. Failure to do so will result in a score of 0 for the assignment and failure in the course. **No exceptions or justifications will be accepted.**

9 Academic Integrity Policy

This assignment must be completed independently. Students will be considered to have committed academic misconduct if:

- There is an unusual similarity between submitted code. In such cases, **all involved submissions** will be considered as cheating. Therefore, students must protect their code from being copied.
- A student submits another student's work under their own account.
- A student does not understand the code they submitted, except for the starter code provided in the initial setup. Students may refer to any resources but must fully understand all the code they submit. If a student does not fully understand the source code they are referencing, they are explicitly warned **not to use it** and should instead rely on what they have learned to write the program.
- A student accidentally submits another student's work using their own account.
- A student uses code generated by automated tools without understanding its meaning.

If a student is found guilty of academic misconduct, they will receive a score of 0 for the entire course (not just this assignment).

NO EXCEPTIONS WILL BE ACCEPTED, AND NO JUSTIFICATIONS WILL BE CONSIDERED!

After each assignment submission, some students will be randomly selected for an interview to verify that they personally completed their submitted work.

10 Changes from the Previous Version

...