Lee Zong Yang 100073484

Table of Contents

1. Task 1
2. Task 2
3. Task 3
4. Task 4
5. Task 5

**Task 1**

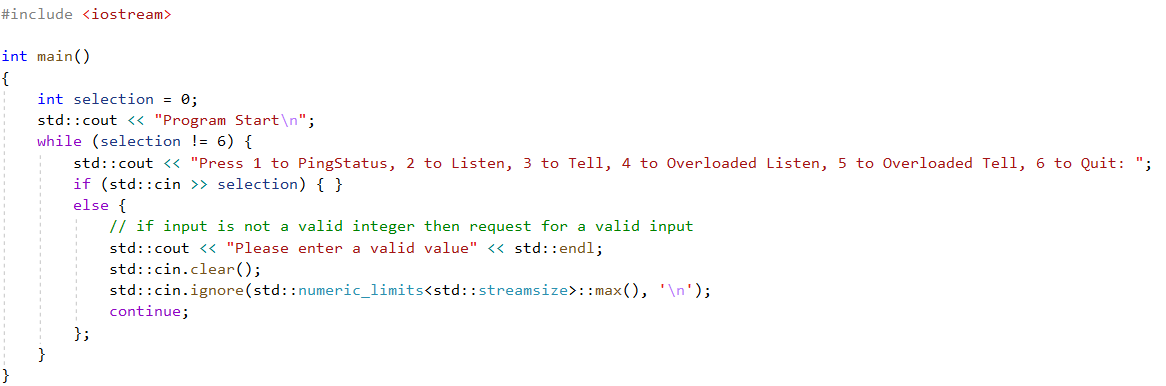
**Description**

Implement an event loop.

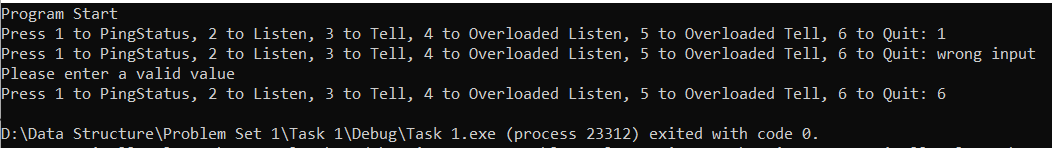
**Concept**

I choose while loop instead of GOTO because it is a better programming practice. Error handling of input had been implemented too.

**Implementation**



**Output**



**Troubleshooting**

The needs to handle not viable input. Managed to solve it by checking the validity of std::cin>>selection and clear it if not valid input had been entered.

**Task 2**

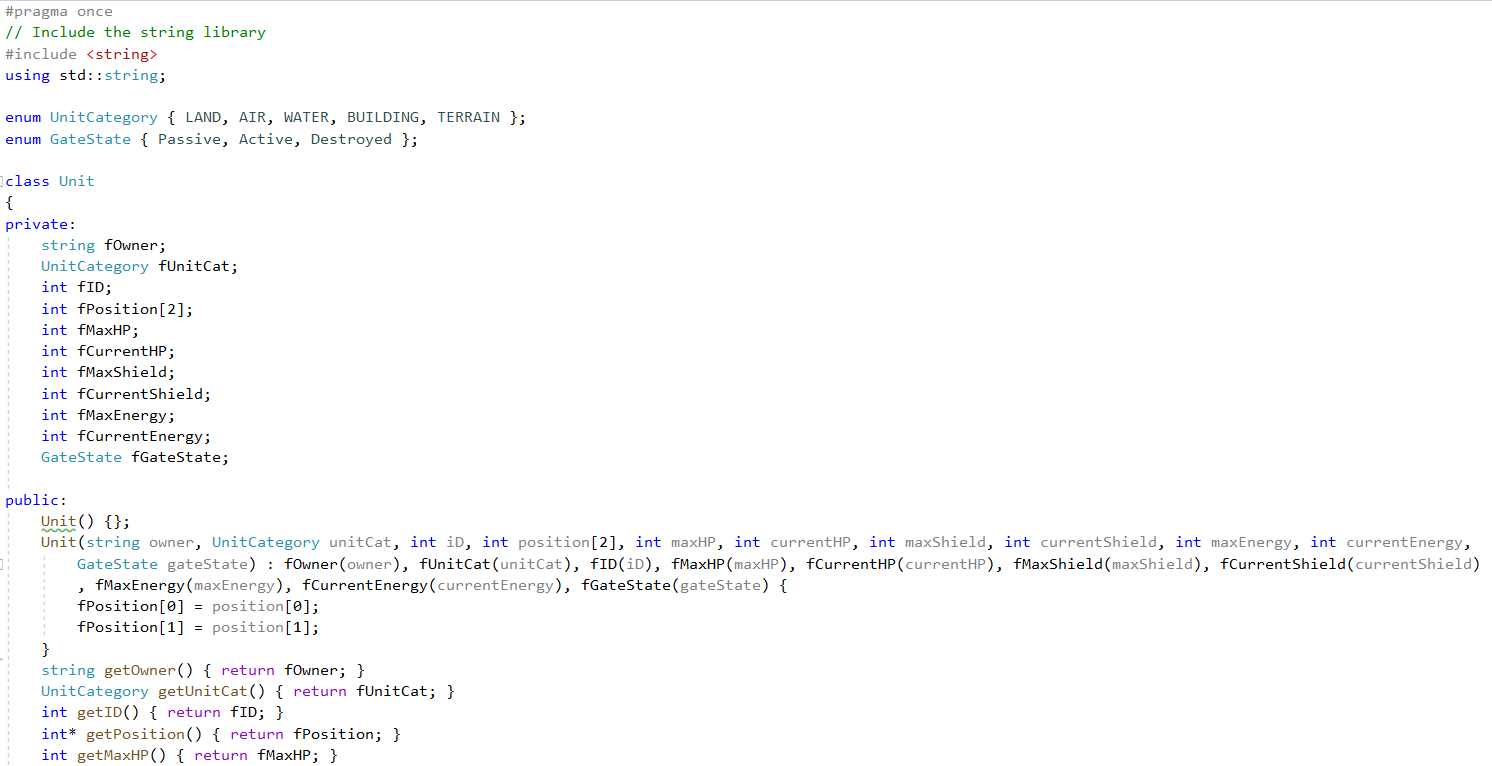
**Description**

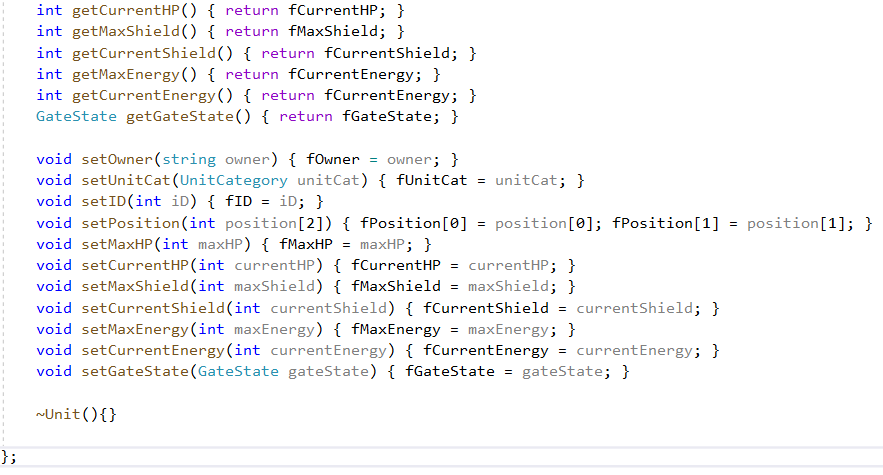
Implement a Unit base class to represent a generic Unit entity.

**Concept**

When creating a class, encapsulation need to be considered. Thus, I decided to declare all member field as private and it’s getter/setter, constructor, and destructor as public.

**Implementation**





**Troubleshooting**

Tried to find out a way to auto generate getter/setter but it is advisable to write it by hand.

**Task 3**

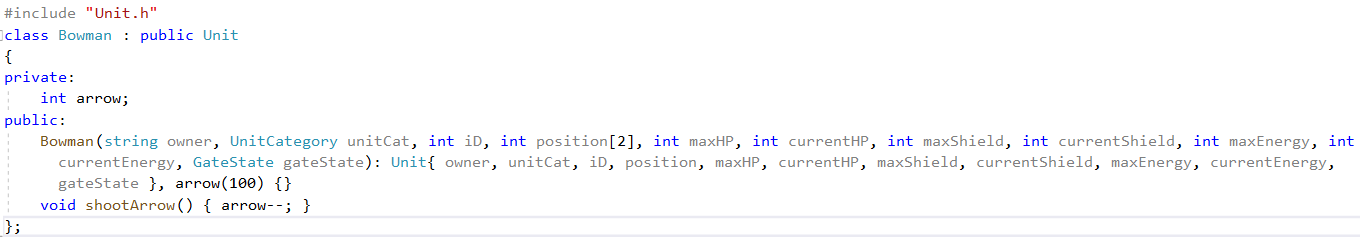
**Description**

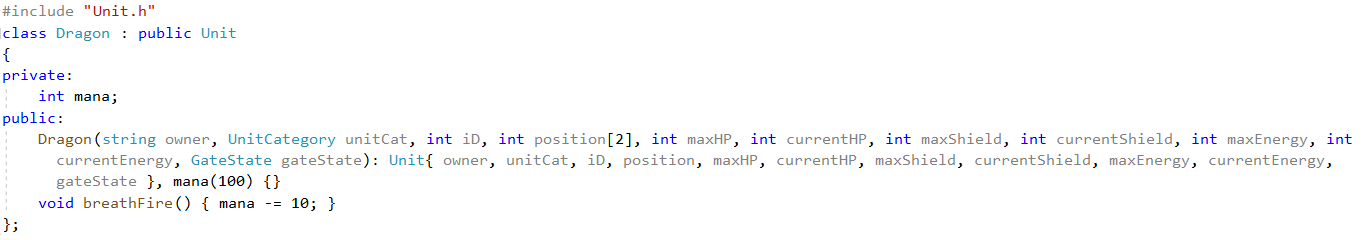
Implement 3 different derived classes of Unit base class.

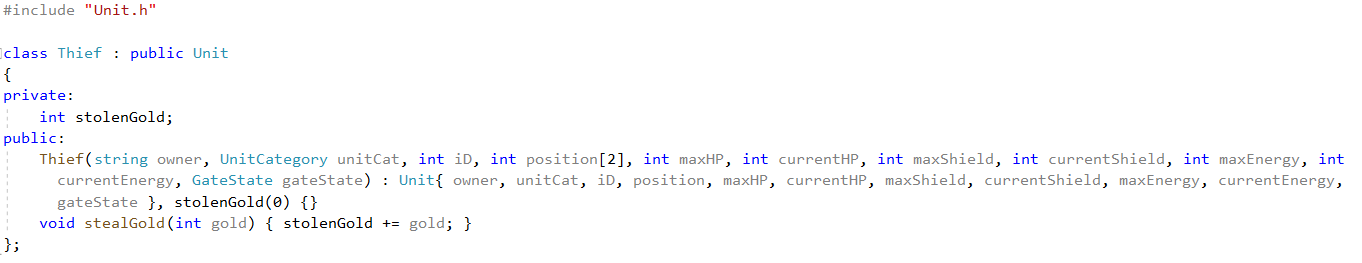
**Concept**

To reduce the work, I decided to use initializer list to initialize the members in base class.

**Implementation**







**Troubleshooting**

The only hard part is the devise of derived class.

**Task 4**

**Description**

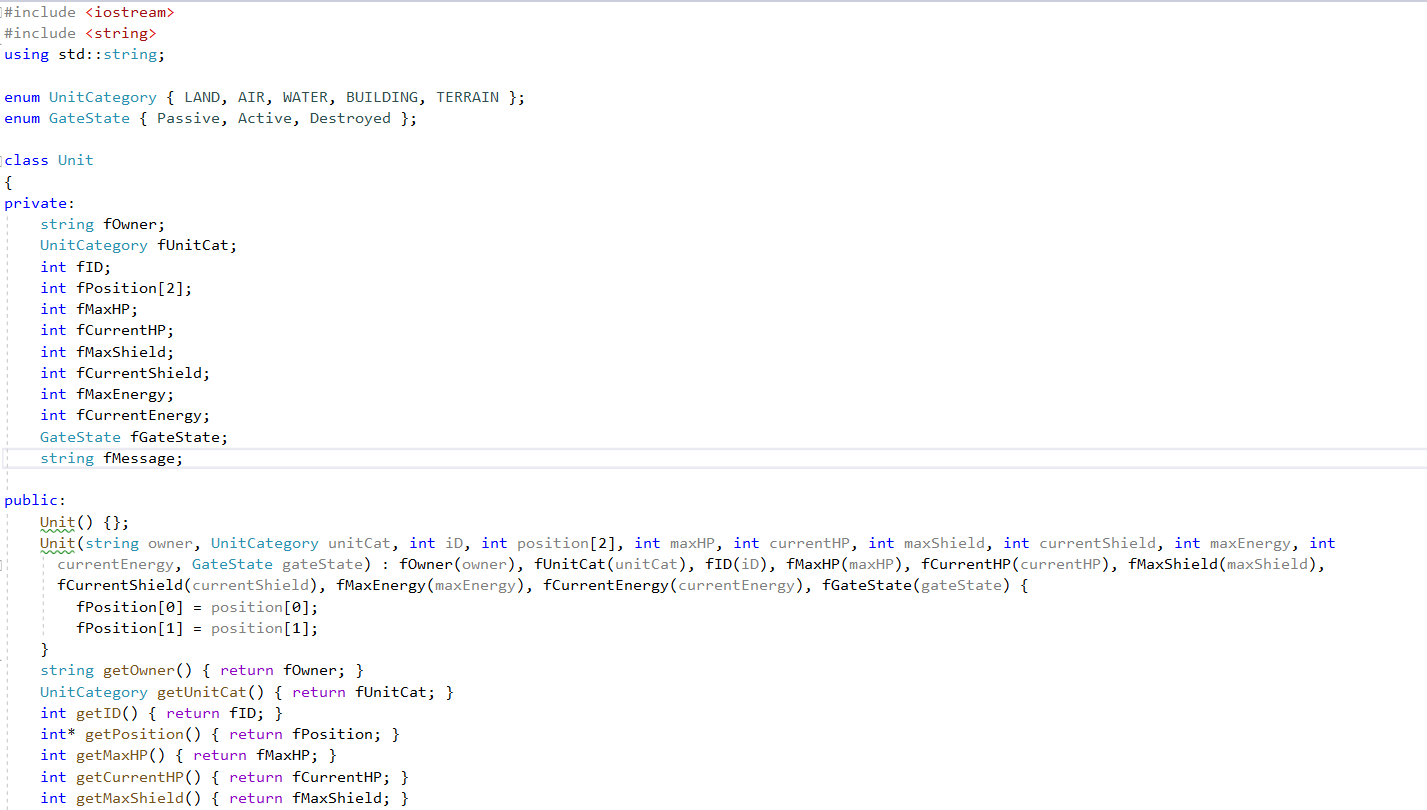
Add some private field and public function into Unit base class and test the functionalities of derived class.

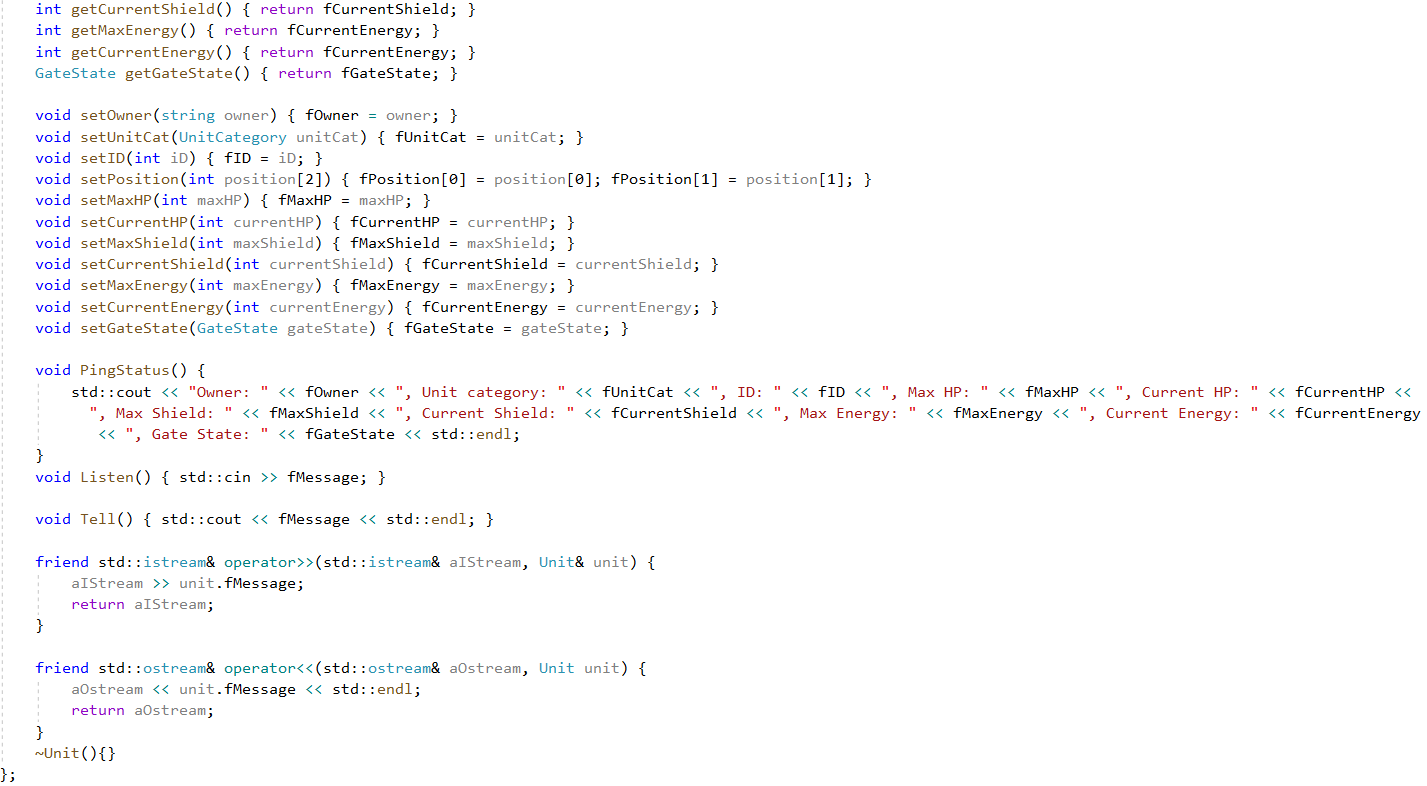
**Concept**

To let the derived classes to inherit the base class, the members of base class need to be protected or public. However, the fMessage need to be private as stated by requirement. Thus, I implemented public getter/setter and friend operator overload to allow the access from derived class.

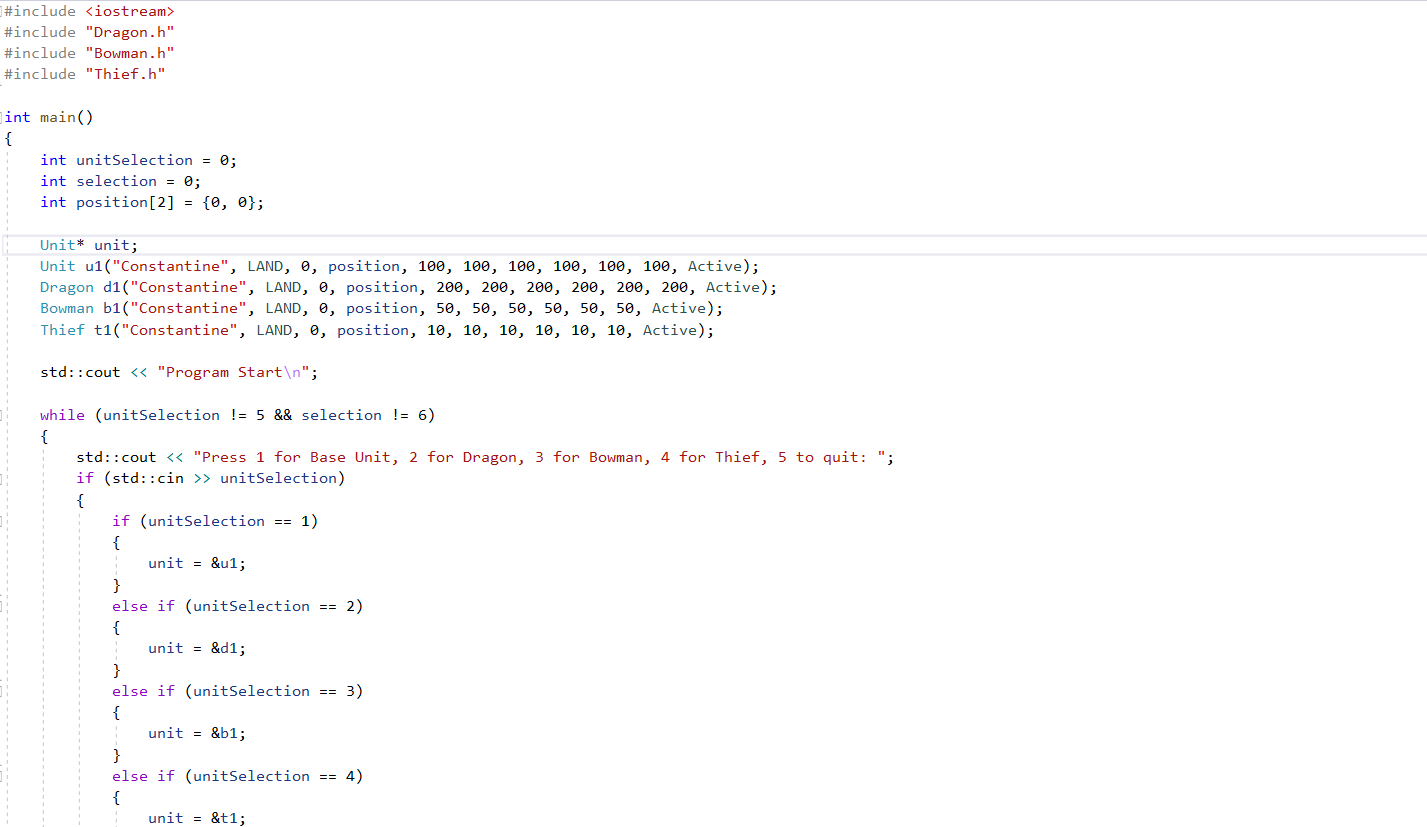
**Code**

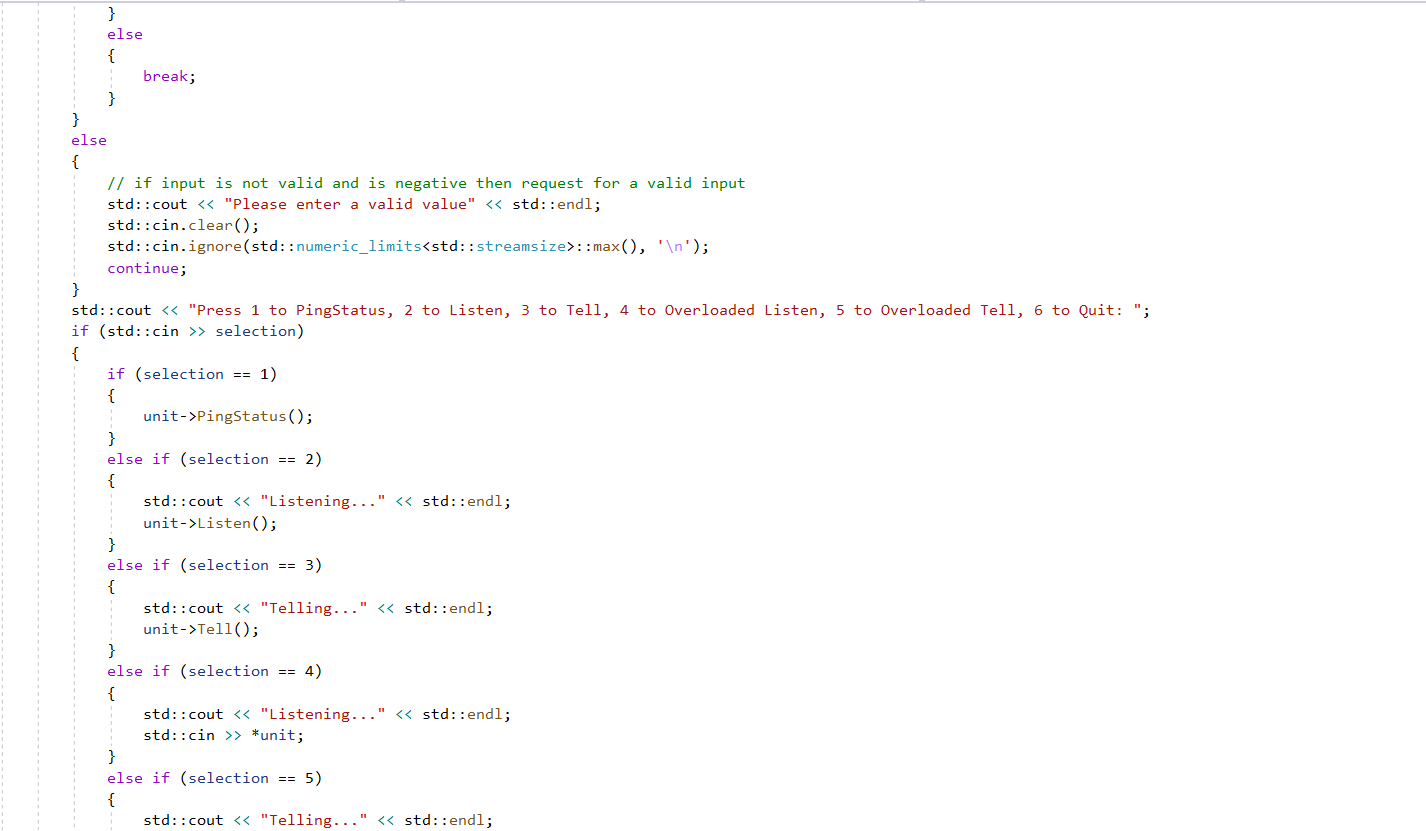
**Unit.h**

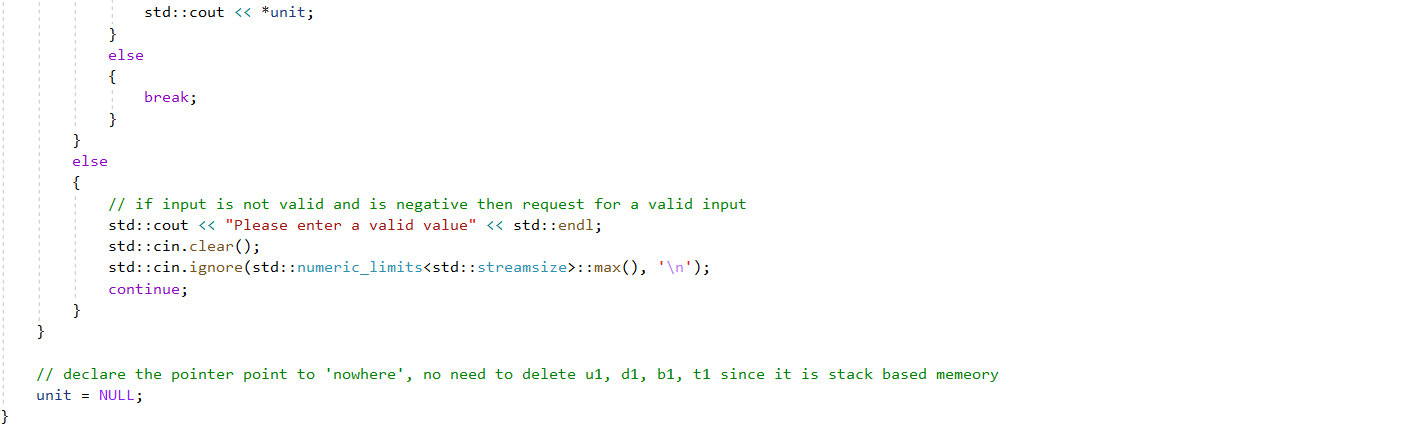




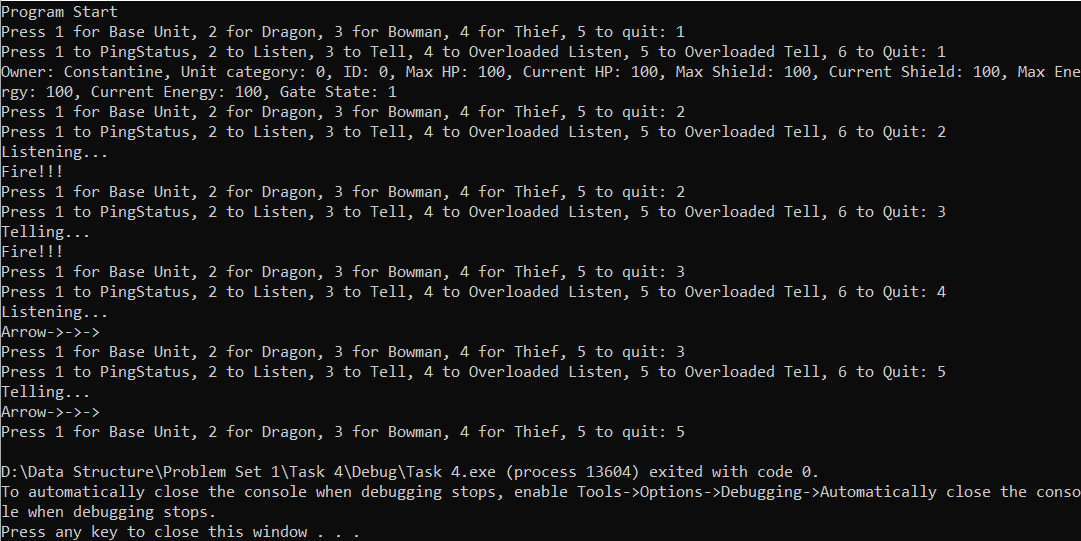
**Main.cpp**







**Output**



**Troubleshooting**

The decision of whether declare the units as stack based memory or heap based memory did took some time to make. In the end, I choose stack based memory to get rid of the hassle of memory management.

**Task 5**

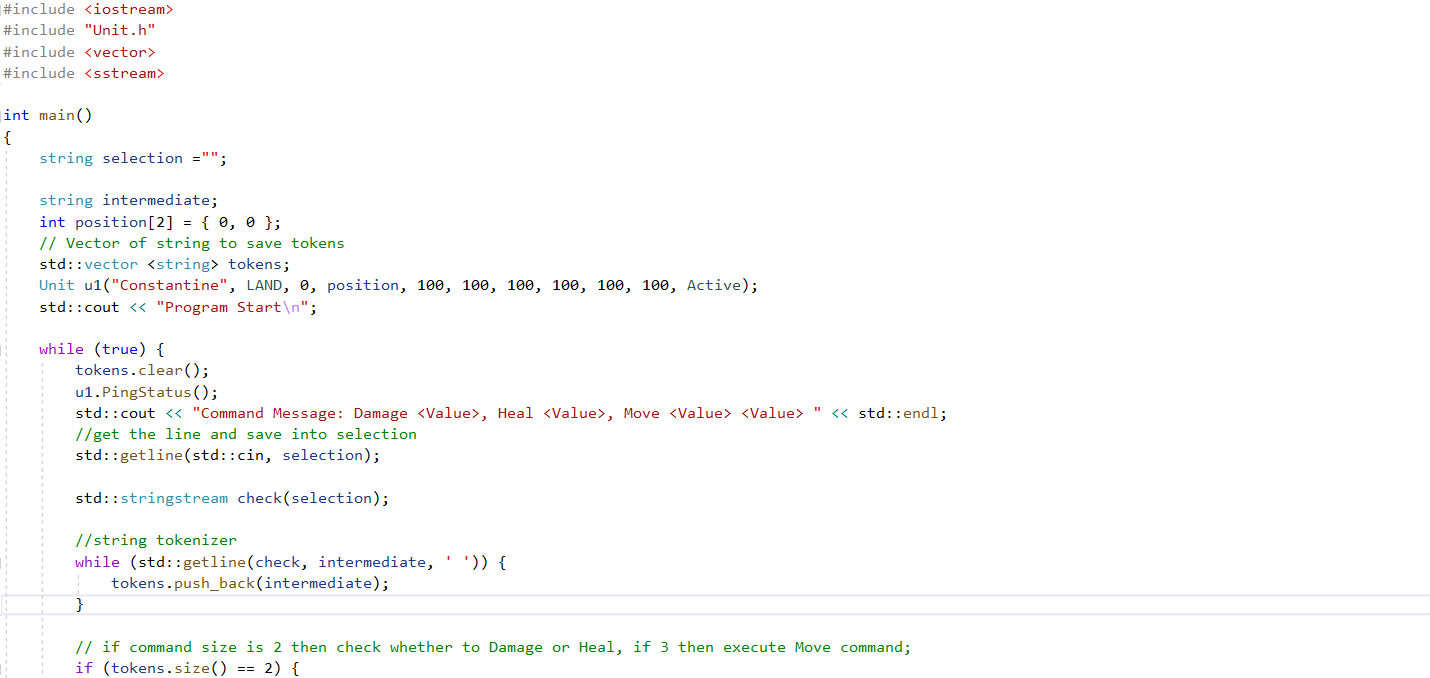
**Description**

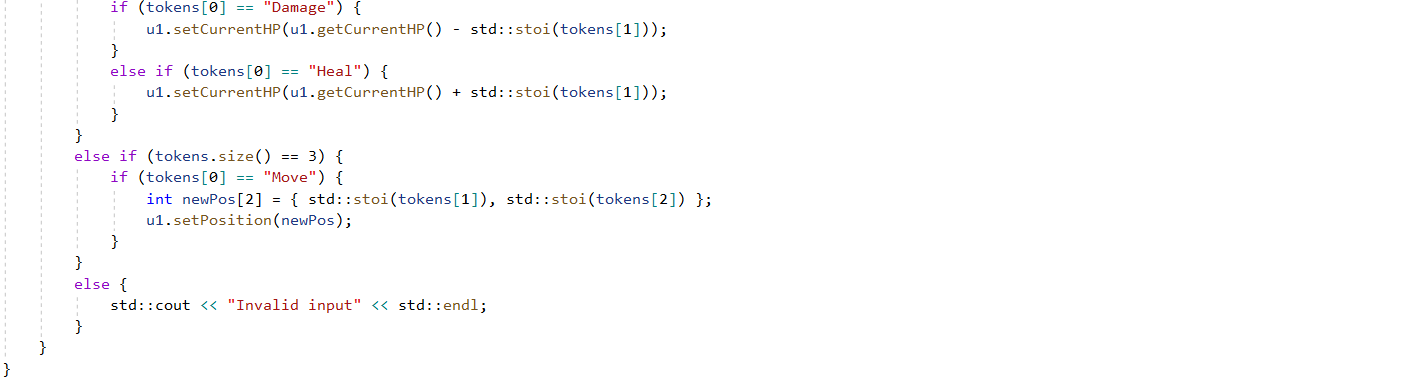
The main objective is to parse the command line input and execute corresponding function.

**Concept**

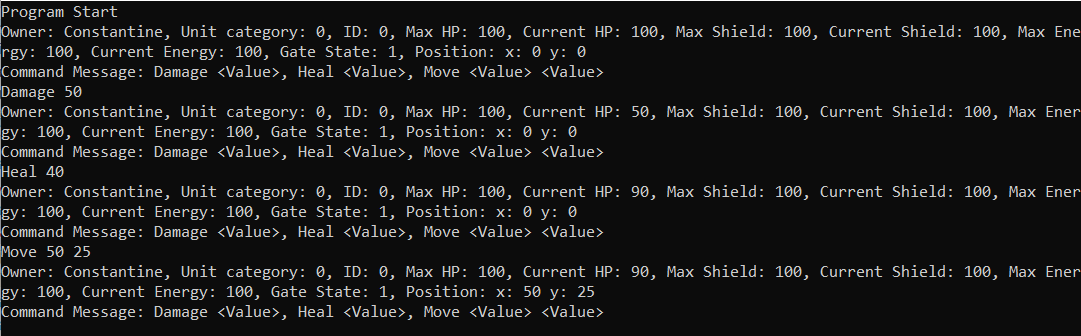
I choose to use third party library to tokenize the string instead of implementing my own string parser as an idiom say “Don’t reinvent the wheel.”

**Implementation**





**Output**



**Troubleshooting**

The finding of string tokenizer solution took some time.

**Appendix**

**Unit.h**

#pragma once

// Include the string library

#include <iostream>

#include <string>

using std::string;

enum UnitCategory { LAND, AIR, WATER, BUILDING, TERRAIN };

enum GateState { Passive, Active, Destroyed };

class Unit

{

private:

string fOwner;

UnitCategory fUnitCat;

int fID;

int fPosition[2];

int fMaxHP;

int fCurrentHP;

int fMaxShield;

int fCurrentShield;

int fMaxEnergy;

int fCurrentEnergy;

GateState fGateState;

string fMessage;

public:

Unit() {};

Unit(string owner, UnitCategory unitCat, int iD, int position[2], int maxHP, int currentHP, int maxShield, int currentShield, int maxEnergy, int currentEnergy, GateState gateState) : fOwner(owner), fUnitCat(unitCat), fID(iD), fMaxHP(maxHP), fCurrentHP(currentHP), fMaxShield(maxShield), fCurrentShield(currentShield), fMaxEnergy(maxEnergy), fCurrentEnergy(currentEnergy), fGateState(gateState) {

fPosition[0] = position[0];

fPosition[1] = position[1];

}

string getOwner() { return fOwner; }

UnitCategory getUnitCat() { return fUnitCat; }

int getID() { return fID; }

int\* getPosition() { return fPosition; }

int getMaxHP() { return fMaxHP; }

int getCurrentHP() { return fCurrentHP; }

int getMaxShield() { return fMaxShield; }

int getCurrentShield() { return fCurrentShield; }

int getMaxEnergy() { return fCurrentEnergy; }

int getCurrentEnergy() { return fCurrentEnergy; }

GateState getGateState() { return fGateState; }

void setOwner(string owner) { fOwner = owner; }

void setUnitCat(UnitCategory unitCat) { fUnitCat = unitCat; }

void setID(int iD) { fID = iD; }

void setPosition(int position[2]) { fPosition[0] = position[0]; fPosition[1] = position[1]; }

void setMaxHP(int maxHP) { fMaxHP = maxHP; }

void setCurrentHP(int currentHP) { fCurrentHP = currentHP; }

void setMaxShield(int maxShield) { fMaxShield = maxShield; }

void setCurrentShield(int currentShield) { fCurrentShield = currentShield; }

void setMaxEnergy(int maxEnergy) { fMaxEnergy = maxEnergy; }

void setCurrentEnergy(int currentEnergy) { fCurrentEnergy = currentEnergy; }

void setGateState(GateState gateState) { fGateState = gateState; }

void PingStatus() {

std::cout << "Owner: " << fOwner << ", Unit category: " << fUnitCat << ", ID: " << fID << ", Max HP: " << fMaxHP << ", Current HP: " << fCurrentHP << ", Max Shield: " << fMaxShield << ", Current Shield: " << fCurrentShield << ", Max Energy: " << fMaxEnergy << ", Current Energy: " << fCurrentEnergy << ", Gate State: " << fGateState << ", Position: " << "x: " << fPosition[0] << " y: " << fPosition[1] << std::endl;

}

void Listen() { std::cin >> fMessage; }

void Tell() { std::cout << fMessage << std::endl; }

friend std::istream& operator>>(std::istream& aIStream, Unit& unit) {

aIStream >> unit.fMessage;

return aIStream;

}

friend std::ostream& operator<<(std::ostream& aOstream, Unit unit) {

aOstream << unit.fMessage << std::endl;

return aOstream;

}

~Unit(){}

};

**Bowman.h**

#pragma once

#include "Unit.h"

class Bowman : public Unit

{

private:

int arrow;

public:

Bowman(string owner, UnitCategory unitCat, int iD, int position[2], int maxHP, int currentHP, int maxShield, int currentShield, int maxEnergy, int currentEnergy, GateState gateState): Unit{ owner, unitCat, iD, position, maxHP, currentHP, maxShield, currentShield, maxEnergy, currentEnergy, gateState }, arrow(100) {}

void shootArrow() { arrow--; }

};

**Dragon.h**

#pragma once

#include "Unit.h"

class Dragon : public Unit

{

private:

int mana;

public:

Dragon(string owner, UnitCategory unitCat, int iD, int position[2], int maxHP, int currentHP, int maxShield, int currentShield, int maxEnergy, int currentEnergy, GateState gateState): Unit{ owner, unitCat, iD, position, maxHP, currentHP, maxShield, currentShield, maxEnergy, currentEnergy, gateState }, mana(100) {}

void breathFire() { mana -= 10; }

};

**Thief.h**

#pragma once

#include "Unit.h"

class Thief : public Unit

{

private:

int stolenGold;

public:

Thief(string owner, UnitCategory unitCat, int iD, int position[2], int maxHP, int currentHP, int maxShield, int currentShield, int maxEnergy, int currentEnergy, GateState gateState) : Unit{ owner, unitCat, iD, position, maxHP, currentHP, maxShield, currentShield, maxEnergy, currentEnergy, gateState }, stolenGold(0) {}

void stealGold(int gold) { stolenGold += gold; }

};

**Task 1(main.cpp)**

#include <iostream>

int main()

{

int selection = 0;

std::cout << "Program Start\n";

while (selection != 6) {

std::cout << "Press 1 to PingStatus, 2 to Listen, 3 to Tell, 4 to Overloaded Listen, 5 to Overloaded Tell, 6 to Quit: ";

if (std::cin >> selection) { }

else {

// if input is not a valid integer then request for a valid input

std::cout << "Please enter a valid value" << std::endl;

std::cin.clear();

std::cin.ignore(std::numeric\_limits<std::streamsize>::max(), '\n');

continue;

};

}

}

**Task 4(main.cpp)**

#include <iostream>

#include "Dragon.h"

#include "Bowman.h"

#include "Thief.h"

int main()

{

int unitSelection = 0;

int selection = 0;

int position[2] = {0, 0};

Unit\* unit;

Unit u1("Constantine", LAND, 0, position, 100, 100, 100, 100, 100, 100, Active);

Dragon d1("Constantine", LAND, 0, position, 200, 200, 200, 200, 200, 200, Active);

Bowman b1("Constantine", LAND, 0, position, 50, 50, 50, 50, 50, 50, Active);

Thief t1("Constantine", LAND, 0, position, 10, 10, 10, 10, 10, 10, Active);

std::cout << "Program Start\n";

while (unitSelection != 5 && selection != 6)

{

std::cout << "Press 1 for Base Unit, 2 for Dragon, 3 for Bowman, 4 for Thief, 5 to quit: ";

if (std::cin >> unitSelection)

{

if (unitSelection == 1)

{

unit = &u1;

}

else if (unitSelection == 2)

{

unit = &d1;

}

else if (unitSelection == 3)

{

unit = &b1;

}

else if (unitSelection == 4)

{

unit = &t1;

}

else

{

break;

}

}

else

{

// if input is not valid and is negative then request for a valid input

std::cout << "Please enter a valid value" << std::endl;

std::cin.clear();

std::cin.ignore(std::numeric\_limits<std::streamsize>::max(), '\n');

continue;

}

std::cout << "Press 1 to PingStatus, 2 to Listen, 3 to Tell, 4 to Overloaded Listen, 5 to Overloaded Tell, 6 to Quit: ";

if (std::cin >> selection)

{

if (selection == 1)

{

unit->PingStatus();

}

else if (selection == 2)

{

std::cout << "Listening..." << std::endl;

unit->Listen();

}

else if (selection == 3)

{

std::cout << "Telling..." << std::endl;

unit->Tell();

}

else if (selection == 4)

{

std::cout << "Listening..." << std::endl;

std::cin >> \*unit;

}

else if (selection == 5)

{

std::cout << "Telling..." << std::endl;

std::cout << \*unit;

}

else

{

break;

}

}

else

{

// if input is not valid and is negative then request for a valid input

std::cout << "Please enter a valid value" << std::endl;

std::cin.clear();

std::cin.ignore(std::numeric\_limits<std::streamsize>::max(), '\n');

continue;

}

}

// declare the pointer point to 'nowhere', no need to delete u1, d1, b1, t1 since it is stack based memeory

unit = NULL;

}

**Task 5(main.cpp)**

#include <iostream>

#include "Unit.h"

#include <vector>

#include <sstream>

int main()

{

string selection ="";

string intermediate;

int position[2] = { 0, 0 };

// Vector of string to save tokens

std::vector <string> tokens;

Unit u1("Constantine", LAND, 0, position, 100, 100, 100, 100, 100, 100, Active);

std::cout << "Program Start\n";

while (true) {

tokens.clear();

u1.PingStatus();

std::cout << "Command Message: Damage <Value>, Heal <Value>, Move <Value> <Value> " << std::endl;

//get the line and save into selection

std::getline(std::cin, selection);

std::stringstream check(selection);

//string tokenizer

while (std::getline(check, intermediate, ' ')) {

tokens.push\_back(intermediate);

}

// if command size is 2 then check whether to Damage or Heal, if 3 then execute Move command;

if (tokens.size() == 2) {

if (tokens[0] == "Damage") {

u1.setCurrentHP(u1.getCurrentHP() - std::stoi(tokens[1]));

}

else if (tokens[0] == "Heal") {

u1.setCurrentHP(u1.getCurrentHP() + std::stoi(tokens[1]));

}

}

else if (tokens.size() == 3) {

if (tokens[0] == "Move") {

int newPos[2] = { std::stoi(tokens[1]), std::stoi(tokens[2]) };

u1.setPosition(newPos);

}

}

else {

std::cout << "Invalid input" << std::endl;

}

}

}