Lee Zong Yang 100073484

Table of Contents

Task 1

Description

This task is like task in problem set 1. It requires an infinite loop that receive input and only end when “Exit” command is received. In the loop, certain command can be used to modify the state of “Entity” class.

Concept

I choose while loop instead of GOTO because it is a better programming practice. Error handling of input had been implemented too. The concept to implement the function to analyse input is first tokenize the string and compare the distinct wording in the sentence.

Implementation

```cpp
#include <iostream>
#include "Entity.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;
    Entity u1(1, "Mytical Creater", 99999);
    Entity u2(2, "Pikachu", 100);
    std::cout << "Program Start\n";

    while (true) {
        tokens.clear();
        std::cout << "Command Message: <ObjectID> ID GET, <ObjectID> ID SET <VALUE>, <ObjectID> NAME GET, <ObjectID> NAME SET <VALUE>, <ObjectID> HP GET, <ObjectID> HP SET
          <VALUE>, EXIT" << 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);
        }

        // Check each token to determine what action to be taken
        if (tokens.size() == 3) {
            if (tokens[1] == "ID") {
                if (tokens[2] == "GET") {
                    if (stoi(tokens[0]) == u1.getEntityID()) {
                        std::cout << u1.getEntityID() << std::endl;
                    }
                    else if (stoi(tokens[0]) == u2.getEntityID()) {
                        std::cout << u2.getEntityID() << std::endl;
                    }
                }
            }
```

```
        else if (tokens[1] == "NAME") {
            if (tokens[2] == "GET") {
                if (stoi(tokens[0]) == u1.getEntityID()) {
                    std::cout << u1.getName() << std::endl;
                }
                else if (stoi(tokens[0]) == u2.getEntityID()) {
                    std::cout << u2.getName() << std::endl;
                }
            }
        }
        else if (tokens[1] == "HP") {
            if (tokens[2] == "GET") {
                if (stoi(tokens[0]) == u1.getEntityID()) {
                    std::cout << u1.getHP() << std::endl;
                }
                else if (stoi(tokens[0]) == u2.getEntityID()) {
                    std::cout << u2.getHP() << std::endl;
                }
            }
        }
    }
    else if (tokens.size() == 4) {
        if (tokens[1] == "ID") {
            if (tokens[2] == "SET") {
                if (stoi(tokens[0]) == u1.getEntityID()) {
                    u1.setEntityID(std::stoi(tokens[3]));
                }
                else if (stoi(tokens[0]) == u2.getEntityID()) {
                    u2.setEntityID(std::stoi(tokens[3]));
                }
            }
        }
        else if (tokens[1] == "NAME") {
            if (tokens[2] == "SET") {
                if (stoi(tokens[0]) == u1.getEntityID()) {
                    u1.setName(tokens[3]);
                }
                else if (stoi(tokens[0]) == u2.getEntityID()) {
                    u2.setName(tokens[3]);
                }
            }
        }
```

```
        else if (tokens[1] == "HP") {
            if (tokens[2] == "SET") {
                if (stoi(tokens[0]) == u1.getEntityID()) {
                    u1.setHP(std::stoi(tokens[3]));
                }
                else if (stoi(tokens[0]) == u2.getEntityID()) {
                    u2.setHP(std::stoi(tokens[3]));
                }
            }
        }
    }
    else if (tokens[0] == "EXIT") {
        std::cout << "Exit Program" << std::endl;
        break;
    }
  }
}
```

Output

```
Program Start
Command Message: <ObjectID> ID GET, <ObjectID> ID SET <VALUE>, <ObjectID> NAME GET, <ObjectID> NAME SET <VALUE>, <ObjectID> HP GET, <ObjectID> HP SET <VALUE>, EXIT
2 ID GET
2
Command Message: <ObjectID> ID GET, <ObjectID> ID SET <VALUE>, <ObjectID> NAME GET, <ObjectID> NAME SET <VALUE>, <ObjectID> HP GET, <ObjectID> HP SET <VALUE>, EXIT
2 ID SET 3
Command Message: <ObjectID> ID GET, <ObjectID> ID SET <VALUE>, <ObjectID> NAME GET, <ObjectID> NAME SET <VALUE>, <ObjectID> HP GET, <ObjectID> HP SET <VALUE>, EXIT
3 NAME GET
Pikachu
Command Message: <ObjectID> ID GET, <ObjectID> ID SET <VALUE>, <ObjectID> NAME GET, <ObjectID> NAME SET <VALUE>, <ObjectID> HP GET, <ObjectID> HP SET <VALUE>, EXIT
3 NAME SET ChangedPikaPika
Command Message: <ObjectID> ID GET, <ObjectID> ID SET <VALUE>, <ObjectID> NAME GET, <ObjectID> NAME SET <VALUE>, <ObjectID> HP GET, <ObjectID> HP SET <VALUE>, EXIT
3 NAME GET
ChangedPikaPika
Command Message: <ObjectID> ID GET, <ObjectID> ID SET <VALUE>, <ObjectID> NAME GET, <ObjectID> NAME SET <VALUE>, <ObjectID> HP GET, <ObjectID> HP SET <VALUE>, EXIT
3 HP GET
100
Command Message: <ObjectID> ID GET, <ObjectID> ID SET <VALUE>, <ObjectID> NAME GET, <ObjectID> NAME SET <VALUE>, <ObjectID> HP GET, <ObjectID> HP SET <VALUE>, EXIT
3 HP SET 10000
Command Message: <ObjectID> ID GET, <ObjectID> ID SET <VALUE>, <ObjectID> NAME GET, <ObjectID> NAME SET <VALUE>, <ObjectID> HP GET, <ObjectID> HP SET <VALUE>, EXIT
3 HP GET
10000
Command Message: <ObjectID> ID GET, <ObjectID> ID SET <VALUE>, <ObjectID> NAME GET, <ObjectID> NAME SET <VALUE>, <ObjectID> HP GET, <ObjectID> HP SET <VALUE>, EXIT
EXIT
Exit Program

D:\Swinburne Units\Data Structure\Problem Set 2\Task 1\Task 1\Debug\Task 1.exe (process 12248) exited with code 0.
To automatically close the console when debugging stops, enable Tools->Options->Debugging->Automatically close the console when debugging stops.
Press any key to close this window . . .
```

Troubleshooting

The logic of if else statement does require some time to consider thoroughly. To check for every corner case, it is a hassle.

Task 2

Description

Need to create two arrays, one is 1-dimension array, another is 2 dimension array. The purpose is to represent 2 different types of bag that an Entity can carry, and item will be represented by string element. The client (Entity) can only interact with the bags through '++', '- -', and '==' API.

Concept

The concept for implementation is Iterator pattern and Inheritance. The main purpose of Iterator pattern is to provide encapsulation and abstraction. In the future, the underlying storage may be change from array to linked list, but the client (Entity class) still can use the same API of the Iterator without changes to the client's code. A base class will be implemented to reduce the workload of implementing 2 different iterator sub class.

Implementation

Task 2.cpp

```cpp
// Task 2.cpp : This file contains the 'main' function. Program execution begins and ends there.
//

#include <iostream>
#include "Iterator1D.h"
#include "Iterator2D.h"

int main()
{
    const std::string oneDimArr[10] = {"zero", "one", "two", "three", "four", "five", "six", "seven", "eight", "nine"};
    const std::string set1[4] = { "zero", "one", "two", "three" };
    const std::string set2[4] = { "four", "five", "six", "seven" };
    const std::string set3[4] = { "eight", "nine", "ten", "eleven" };
    const std::string set4[4] = { "twelve", "thirteen", "fourteen", "fifteen" };
    const std::string* twoDimArr[] = { set1, set2, set3, set4 };

    std::cout << "1D Iterator" << std::endl;
    for (Iterator1D iter1D(oneDimArr, 10); !(iter1D == iter1D.end()); iter1D++) {
        std::cout << *iter1D << std::endl;
    }

    std::cout << std::endl << "2D Iterator" << std::endl;
    for (Iterator2D iter2D(twoDimArr, 16); !(iter2D == iter2D.end()); iter2D++) {
        std::cout << *iter2D << std::endl;
    }
}
```

Iterator.h

```cpp
#pragma once
#include <string>
#include <iostream>

class Iterator
{
protected:
    const int fLength;
    int fIndex;

public:
    Iterator(const int aLength, int aStart) : fLength(aLength), fIndex(aStart) {
    }

    const std::string* f1DArrayElements;
    const std::string** f2DArrayElements;

    int getfIndex() const { return fIndex; }

    virtual const std::string& operator*() const { return ""; }

    Iterator& operator++() {
        fIndex++;
        return *this;
    }
    Iterator operator++(int) {
        Iterator temp = *this;
        fIndex++;
        return temp;
    }
    Iterator& operator--() {
        fIndex--;
        return *this;
    }
    Iterator operator--(int) {
        Iterator temp = *this;
        fIndex--;
        return temp;
    }
    virtual bool operator==(const Iterator& aOther) const { return false; };
    virtual Iterator begin() const { return *this; }
    virtual Iterator end() const { return *this; }
};
```

Iterator1D.h

```cpp
#pragma once
#include "Iterator.h"


class Iterator1D :
    public Iterator
{
private:


public:
    Iterator1D(const std::string* aArray, const int aLength, int aStart = 0): Iterator(aLength, aStart) {
        f1DArrayElements = aArray;
    }

    const std::string& operator*() const {
        return f1DArrayElements[fIndex];
    }

    bool operator==(const Iterator& aOther) const {
        return (fIndex == aOther.getfIndex()) && (f1DArrayElements == aOther.f1DArrayElements);
    }

    Iterator begin() const {
        return Iterator1D(f1DArrayElements, fLength);
    }

    Iterator end() const {
        return Iterator1D(f1DArrayElements, fLength, fLength);
    }
};
```

Iterator2D.h

```cpp
#pragma once
#include "Iterator.h"

class Iterator2D :
    public Iterator
{
private:


public:
    Iterator2D(const std::string** aArray,const int aLength, int aStart = 0) : Iterator(aLength, aStart) {
        f2DArrayElements = aArray;
    }

    const std::string& operator*() const {
        return f2DArrayElements[fIndex/4][fIndex%4];
    }

    bool operator==(const Iterator& aOther) const {
        return (fIndex == aOther.getfIndex()) && (f2DArrayElements == aOther.f2DArrayElements);
    }

    Iterator begin() const{
        return Iterator2D(f2DArrayElements, fLength);
    }

    Iterator end() const {
        return Iterator2D(f2DArrayElements, fLength, fLength);
    }
};
```

Output

```
1D Iterator
zero
one
two
three
four
five
six
seven
eight
nine

2D Iterator
zero
one
two
three
four
five
six
seven
eight
nine
ten
eleven
twelve
thirteen
fourteen
fifteen
```

Troubleshooting

The difficult part lies in the assignment of 2D array when creating Iterator2D class. The assignment must be done as copy the address of 2D array pointer, but not the value of all elements in the 2D array. This is to allow the comparison of pointer address for '==' operator overloading.

Task 3

Description

Create 2 Iterator pointers which will point to different iterator sub-class. Then set the iterator in the Entity to the created iterator through Entity's public member function. 3 more public methods to interact with the bag: InventoryNext – use the '++' operator, InventoryPrev – use the '- -' operator, InventoryGet – use the '*' operator.

Concept

First, the Iterator base class need to implement shared API for both derived iterator class. This can ensure the functionalities of the base class Iterator pointer. Polymorphism at work. Next, defensive programming need to be adopted to prevent the InventoryNext and InventoryPrev will not target an element out of bound. This could be achieved by checking through the "==" operator.

Implementation

Task 3.cpp

```
#include <iostream>
#include "Entity.h"
#include "Iterator1D.h"
#include "Iterator2D.h"

int main()
{
    const std::string oneDimArr[10] = { "zero", "one", "two", "three", "four", "five", "six", "seven", "eight", "nine" };
    const std::string set1[4] = { "zero", "one", "two", "three" };
    const std::string set2[4] = { "four", "five", "six", "seven" };
    const std::string set3[4] = { "eight", "nine", "ten", "eleven" };
    const std::string set4[4] = { "twelve", "thirteen", "fourteen", "fifteen" };
    const std::string* twoDimArr[] = { set1, set2, set3, set4 };

    Iterator* pIter1D = new Iterator1D(oneDimArr, 10);
    Iterator* pIter2D = new Iterator2D(twoDimArr, 16);

    Entity u1(1, "Mytical Creater", 99999);
    Entity u2(2, "Pikachu", 100);
    u1.grab(pIter1D);
    u2.grab(pIter2D);

    std::cout << "1D Iterator Test" << std::endl;
    std::cout << u1.InventoryGet() << std::endl;
    std::cout << u1.InventoryPrev() << std::endl;
    std::cout << u1.InventoryNext() << std::endl;

    std::cout << std::endl << "2D Iterator Test" << std::endl;
    std::cout << u2.InventoryGet() << std::endl;
    std::cout << u1.InventoryPrev() << std::endl;
    std::cout << u2.InventoryNext() << std::endl;
    std::cout << u2.InventoryNext() << std::endl;
    std::cout << u2.InventoryNext() << std::endl;
    std::cout << u2.InventoryNext() << std::endl;
    std::cout << u2.InventoryNext() << std::endl;
    std::cout << u2.InventoryNext() << std::endl;
    std::cout << u2.InventoryNext() << std::endl;
    std::cout << u2.InventoryNext() << std::endl;
    std::cout << u2.InventoryNext() << std::endl;
    std::cout << u2.InventoryNext() << std::endl;
    std::cout << u2.InventoryNext() << std::endl;
    std::cout << u2.InventoryNext() << std::endl;
    std::cout << u2.InventoryNext() << std::endl;
```

```
    std::cout << u2.InventoryNext() << std::endl;
    std::cout << u2.InventoryNext() << std::endl;
    std::cout << u2.InventoryNext() << std::endl;
    std::cout << u2.InventoryNext() << std::endl;
    std::cout << u2.InventoryNext() << std::endl;
    std::cout << u2.InventoryNext() << std::endl;

}
```

Iterator.h

```cpp
#pragma once
#include <string>
#include <iostream>

class Iterator
{
protected:
    const int fLength;
    int fIndex;

public:
    Iterator(const int aLength, int aStart) : fLength(aLength), fIndex(aStart) {
    }

    const std::string* f1DArrayElements;
    const std::string** f2DArrayElements;

    int getfIndex() const { return fIndex; }

    virtual const std::string& operator*() const { return ""; }

    Iterator& operator++() {
        fIndex++;
        return *this;
    }

    Iterator operator++(int) {
        Iterator temp = *this;
        fIndex++;
        return temp;
    }

    Iterator& operator--() {
        fIndex--;
        return *this;
    }

    Iterator operator--(int) {
        Iterator temp = *this;
        fIndex--;
        return temp;
    }

    virtual bool operator==(const Iterator& aOther) const { return false; };

    virtual Iterator begin() const { return *this; }
    virtual Iterator end() const { return *this; }
};
```

Iterator1D.h

```cpp
#pragma once
#include "Iterator.h"


class Iterator1D :
    public Iterator
{
private:


public:
    Iterator1D(const std::string* aArray, const int aLength, int aStart = 0) : Iterator(aLength, aStart) {
        f1DArrayElements = aArray;
    }

    const std::string& operator*() const {
        return f1DArrayElements[fIndex];
    }

    bool operator==(const Iterator& aOther) const {
        return (fIndex == aOther.getfIndex()) && (f1DArrayElements == aOther.f1DArrayElements);
    }

    Iterator begin() const {
        return Iterator1D(f1DArrayElements, fLength);
    }

    Iterator end() const {
        return Iterator1D(f1DArrayElements, fLength, fLength);
    }
};
```

Iterator2D.h

```cpp
#pragma once
#include "Iterator.h"

class Iterator2D :
    public Iterator
{
private:


public:
    Iterator2D(const std::string** aArray, const int aLength, int aStart = 0) : Iterator(aLength, aStart) {
        f2DArrayElements = aArray;
    }

    const std::string& operator*() const {
        return f2DArrayElements[fIndex / 4][fIndex % 4];
    }

    bool operator==(const Iterator& aOther) const {
        return (fIndex == aOther.getfIndex()) && (f2DArrayElements == aOther.f2DArrayElements);
    }

    Iterator begin() const {
        return Iterator2D(f2DArrayElements, fLength);
    }

    Iterator end() const {
        return Iterator2D(f2DArrayElements, fLength, fLength);
    }
};
```

Entity.h

```cpp
#pragma once
// Include the string library
#include <iostream>
#include <string>
#include "Iterator.h"
using std::string;

class Entity
{
private:
    int fEntityID;
    string fName;
    int fHP;
    Iterator* fIter;

public:
    Entity() {};
    Entity(int EntityID, string Name, int HP) : fEntityID(EntityID), fName(Name), fHP(HP) {}

    void grab(Iterator* iter) {
        fIter = iter;
    }

    const std::string& InventoryNext() {
        (*fIter)++;
        if (*fIter == (*fIter).end()) {
            (*fIter)--;
        }
        return *(*fIter);
    }

    const std::string& InventoryPrev() {
        if (*fIter == (*fIter).begin()) {}
        else {
            (*fIter)--;
        }
        return *(*fIter);
    }

    const std::string& InventoryGet() {
        return *(*fIter);
    }
```

```cpp
    void setEntityID(int ID) {
        fEntityID = ID;
    }

    void setName(string Name) {
        fName = Name;
    }

    void setHP(int HP) {
        fHP = HP;
    }

    int getEntityID() {
        return fEntityID;
    }

    string getName() {
        return fName;
    }

    int getHP() {
        return fHP;
    }

    ~Entity() {}

};
```

Output

Troubleshooting

When designing the base class of Iterator class, I had faced a major problem due to return value of function could not be abstract class. So, I could not tag the virtual function as abstract function instead had to resort to template function. The member variable for 1D or 2D array had to be at Iterator base class too. This may be unsafe in term of programming guideline, but there is no other way if want to reuse the code through inheritance. A work around may be downcasting the object after check its typeid(), but downcasting is bad practice too. Another approach may be using composition for code reuse, but inheritance is required for this task.

Appendix

Task 1

Task 1.cpp

```
// Task 1.cpp : This file contains the 'main' function. Program execution begins and
ends there.
//

#include <iostream>
#include "Entity.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;
    Entity u1(1, "Mytical Creater", 99999);
    Entity u2(2, "Pikachu", 100);
    std::cout << "Program Start\n";

    while (true) {
        tokens.clear();
        std::cout << "Command Message: <ObjectID> ID GET, <ObjectID> ID SET <VALUE>,
<ObjectID> NAME GET, <ObjectID> NAME SET <VALUE>, <ObjectID> HP GET, <ObjectID> HP SET
<VALUE>, EXIT" << 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);
}

// Check each token to determine what action to be taken
if (tokens.size() == 3) {
    if (tokens[1] == "ID") {
        if (tokens[2] == "GET") {
            if (stoi(tokens[0]) == u1.getEntityID()) {
                std::cout << u1.getEntityID() << std::endl;
            }
            else if (stoi(tokens[0]) == u2.getEntityID()) {
                std::cout << u2.getEntityID() << std::endl;
            }
        }
    }
    else if (tokens[1] == "NAME") {
        if (tokens[2] == "GET") {
            if (stoi(tokens[0]) == u1.getEntityID()) {
                std::cout << u1.getName() << std::endl;
            }
            else if (stoi(tokens[0]) == u2.getEntityID()) {
                std::cout << u2.getName() << std::endl;
            }
        }
    }
    else if (tokens[1] == "HP") {
        if (tokens[2] == "GET") {
            if (stoi(tokens[0]) == u1.getEntityID()) {
                std::cout << u1.getHP() << std::endl;
            }
            else if (stoi(tokens[0]) == u2.getEntityID()) {
                std::cout << u2.getHP() << std::endl;
            }
        }
    }
}
else if (tokens.size() == 4) {
    if (tokens[1] == "ID") {
        if (tokens[2] == "SET") {
            if (stoi(tokens[0]) == u1.getEntityID()) {
                u1.setEntityID(std::stoi(tokens[3]));
            }
            else if (stoi(tokens[0]) == u2.getEntityID()) {
                u2.setEntityID(std::stoi(tokens[3]));
            }
        }
    }
    else if (tokens[1] == "NAME") {
        if (tokens[2] == "SET") {
            if (stoi(tokens[0]) == u1.getEntityID()) {
                u1.setName(tokens[3]);
            }
            else if (stoi(tokens[0]) == u2.getEntityID()) {
                u2.setName(tokens[3]);
            }
        }
```

```
            }
            else if (tokens[1] == "HP") {
                if (tokens[2] == "SET") {
                    if (stoi(tokens[0]) == u1.getEntityID()) {
                        u1.setHP(std::stoi(tokens[3]));
                    }
                    else if (stoi(tokens[0]) == u2.getEntityID()) {
                        u2.setHP(std::stoi(tokens[3]));
                    }
                }
            }
        }
        else if (tokens[0] == "EXIT") {
            std::cout << "Exit Program" << std::endl;
            break;
        }
    }
}
```

Entity.h

```
#pragma once
// Include the string library
#include <iostream>
#include <string>
using std::string;

class Entity
{
private:
	int fEntityID;
	string fName;
	int fHP;

public:
	Entity() {};
	Entity(int EntityID, string Name, int HP) : fEntityID(EntityID), fName(Name),
fHP(HP) {}

	void setEntityID(int ID) {
		fEntityID = ID;
	}

	void setName(string Name) {
		fName = Name;
	}

	void setHP(int HP) {
		fHP = HP;
	}

	int getEntityID() {
		return fEntityID;
	}

	string getName() {
		return fName;
	}

	int getHP() {
		return fHP;
	}
```

```
	~Entity() {}

};
```

Task 2

Task 2.cpp

```
// Task 2.cpp : This file contains the 'main' function. Program execution begins and
ends there.
//

#include <iostream>
#include "Iterator1D.h"
#include "Iterator2D.h"

int main()
{
    const std::string oneDimArr[10] = {"zero", "one", "two", "three", "four", "five",
"six", "seven", "eight", "nine"};
    const std::string set1[4] = { "zero", "one", "two", "three" };
    const std::string set2[4] = { "four", "five", "six", "seven" };
    const std::string set3[4] = { "eight", "nine", "ten", "eleven" };
    const std::string set4[4] = { "twelve", "thirteen", "fourteen", "fifteen" };
    const std::string* twoDimArr[] = { set1, set2, set3, set4 };

    std::cout << "1D Iterator" << std::endl;
    for (Iterator1D iter1D(oneDimArr, 10); !(iter1D == iter1D.end()); iter1D++) {
        std::cout << *iter1D << std::endl;
    }

    std::cout << std::endl << "2D Iterator" << std::endl;
    for (Iterator2D iter2D(twoDimArr, 16); !(iter2D == iter2D.end()); iter2D++) {
        std::cout << *iter2D << std::endl;
    }
}
```

Iterator.h

```
#pragma once
#include <string>
#include <iostream>

class Iterator
{
protected:
	const int fLength;
	int fIndex;

public:
	Iterator(const int aLength, int aStart) : fLength(aLength), fIndex(aStart) {
	}

    const std::string* f1DArrayElements;
    const std::string** f2DArrayElements;

    int getfIndex() const { return fIndex; }

    virtual const std::string& operator*() const { return ""; }
```

```
    Iterator& operator++() {
        fIndex++;
        return *this;
    }
    Iterator operator++(int) {
        Iterator temp = *this;
        fIndex++;
        return temp;
    }
    Iterator& operator--() {
        fIndex--;
        return *this;
    }
    Iterator operator--(int) {
        Iterator temp = *this;
        fIndex--;
        return temp;
    }
    virtual bool operator==(const Iterator& aOther) const { return false; };
    virtual Iterator begin() const { return *this; }
    virtual Iterator end() const { return *this; }
};
```

Iterator1D.h

```
#pragma once
#include "Iterator.h"


class Iterator1D :
    public Iterator
{
private:


public:
    Iterator1D(const std::string* aArray, const int aLength, int aStart = 0):
Iterator(aLength, aStart) {
        f1DArrayElements = aArray;
    }

    const std::string& operator*() const {
        return f1DArrayElements[fIndex];
    }

    bool operator==(const Iterator& aOther) const {
        return (fIndex == aOther.getfIndex()) && (f1DArrayElements ==
aOther.f1DArrayElements);
    }

    Iterator begin() const {
        return Iterator1D(f1DArrayElements, fLength);
    }

    Iterator end() const {
        return Iterator1D(f1DArrayElements, fLength, fLength);
    }
};
```

Iterator2D.h

```
#pragma once
#include "Iterator.h"

class Iterator2D :
    public Iterator
{
private:


public:
    Iterator2D(const std::string** aArray,const int aLength, int aStart = 0) :
Iterator(aLength, aStart) {
        f2DArrayElements = aArray;
    }

    const std::string& operator*() const {
        return f2DArrayElements[fIndex/4][fIndex%4];
    }

    bool operator==(const Iterator& aOther) const {
        return (fIndex == aOther.getfIndex()) && (f2DArrayElements ==
aOther.f2DArrayElements);
    }

    Iterator begin() const{
        return Iterator2D(f2DArrayElements, fLength);
    }

    Iterator end() const {
        return Iterator2D(f2DArrayElements, fLength, fLength);
    }
};
```

Task 3

Task 3.cpp

```
// Task 3.cpp : This file contains the 'main' function. Program execution begins and
ends there.
//

#include <iostream>
#include "Entity.h"
#include "Iterator1D.h"
#include "Iterator2D.h"

int main()
{
    const std::string oneDimArr[10] = { "zero", "one", "two", "three", "four", "five",
"six", "seven", "eight", "nine" };
    const std::string set1[4] = { "zero", "one", "two", "three" };
    const std::string set2[4] = { "four", "five", "six", "seven" };
    const std::string set3[4] = { "eight", "nine", "ten", "eleven" };
    const std::string set4[4] = { "twelve", "thirteen", "fourteen", "fifteen" };
    const std::string* twoDimArr[] = { set1, set2, set3, set4 };

    Iterator* pIter1D = new Iterator1D(oneDimArr, 10);
    Iterator* pIter2D = new Iterator2D(twoDimArr, 16);
```

```
    Entity u1(1, "Mytical Creater", 99999);
    Entity u2(2, "Pikachu", 100);
    u1.grab(pIter1D);
    u2.grab(pIter2D);

    std::cout << "1D Iterator Test" << std::endl;
    std::cout << u1.InventoryGet() << std::endl;
    std::cout << u1.InventoryPrev() << std::endl;
    std::cout << u1.InventoryNext() << std::endl;

    std::cout << std::endl << "2D Iterator Test" << std::endl;
    std::cout << u2.InventoryGet() << std::endl;
    std::cout << u1.InventoryPrev() << std::endl;
    std::cout << u2.InventoryNext() << std::endl;
    std::cout << u2.InventoryNext() << std::endl;
    std::cout << u2.InventoryNext() << std::endl;
    std::cout << u2.InventoryNext() << std::endl;
    std::cout << u2.InventoryNext() << std::endl;
    std::cout << u2.InventoryNext() << std::endl;
    std::cout << u2.InventoryNext() << std::endl;
    std::cout << u2.InventoryNext() << std::endl;
    std::cout << u2.InventoryNext() << std::endl;
    std::cout << u2.InventoryNext() << std::endl;
    std::cout << u2.InventoryNext() << std::endl;
    std::cout << u2.InventoryNext() << std::endl;
    std::cout << u2.InventoryNext() << std::endl;
    std::cout << u2.InventoryNext() << std::endl;
    std::cout << u2.InventoryNext() << std::endl;
    std::cout << u2.InventoryNext() << std::endl;
    std::cout << u2.InventoryNext() << std::endl;
    std::cout << u2.InventoryNext() << std::endl;
    std::cout << u2.InventoryNext() << std::endl;

}
```

Iterator.h

```
#pragma once
#include <string>
#include <iostream>

class Iterator
{
protected:
    const int fLength;
    int fIndex;

public:
    Iterator(const int aLength, int aStart) : fLength(aLength), fIndex(aStart) {
    }

    const std::string* f1DArrayElements;
    const std::string** f2DArrayElements;

    int getfIndex() const { return fIndex; }

    virtual const std::string& operator*() const { return ""; }

    Iterator& operator++() {
        fIndex++;
        return *this;
```

```
    }

    Iterator operator++(int) {
        Iterator temp = *this;
        fIndex++;
        return temp;
    }

    Iterator& operator--() {
        fIndex--;
        return *this;
    }

    Iterator operator--(int) {
        Iterator temp = *this;
        fIndex--;
        return temp;
    }

    virtual bool operator==(const Iterator& aOther) const { return false; };

    virtual Iterator begin() const { return *this; }
    virtual Iterator end() const { return *this; }
};
```

Iterator1D.h

```
#pragma once
#include "Iterator.h"


class Iterator1D :
    public Iterator
{
private:


public:
    Iterator1D(const std::string* aArray, const int aLength, int aStart = 0) :
Iterator(aLength, aStart) {
        f1DArrayElements = aArray;
    }

    const std::string& operator*() const {
        return f1DArrayElements[fIndex];
    }

    bool operator==(const Iterator& aOther) const {
        return (fIndex == aOther.getfIndex()) && (f1DArrayElements ==
aOther.f1DArrayElements);
    }

    Iterator begin() const {
        return Iterator1D(f1DArrayElements, fLength);
    }

    Iterator end() const {
        return Iterator1D(f1DArrayElements, fLength, fLength);
    }
};
```

Iterator2D.h

```
#pragma once
#include "Iterator.h"

class Iterator2D :
    public Iterator
{
private:


public:
    Iterator2D(const std::string** aArray, const int aLength, int aStart = 0) :
Iterator(aLength, aStart) {
        f2DArrayElements = aArray;
    }

    const std::string& operator*() const {
        return f2DArrayElements[fIndex / 4][fIndex % 4];
    }

    bool operator==(const Iterator& aOther) const {
        return (fIndex == aOther.getfIndex()) && (f2DArrayElements ==
aOther.f2DArrayElements);
    }

    Iterator begin() const {
        return Iterator2D(f2DArrayElements, fLength);
    }

    Iterator end() const {
        return Iterator2D(f2DArrayElements, fLength, fLength);
    }
};
```

Entity.h

```
#pragma once
// Include the string library
#include <iostream>
#include <string>
#include "Iterator.h"
using std::string;

class Entity
{
private:
       int fEntityID;
       string fName;
       int fHP;
       Iterator* fIter;

public:
       Entity() {};
       Entity(int EntityID, string Name, int HP) : fEntityID(EntityID), fName(Name),
fHP(HP) {}

       void grab(Iterator* iter) {
              fIter = iter;
       }

       const std::string& InventoryNext() {
```

```
		(*fIter)++;
		if (*fIter == (*fIter).end()) {
			(*fIter)--;
		}
		return *(*fIter);
	}

	const std::string& InventoryPrev() {
		if (*fIter == (*fIter).begin()) {}
		else {
			(*fIter)--;
		}
		return *(*fIter);
	}

	const std::string& InventoryGet() {
		return *(*fIter);
	}

	void setEntityID(int ID) {
		fEntityID = ID;
	}

	void setName(string Name) {
		fName = Name;
	}

	void setHP(int HP) {
		fHP = HP;
	}

	int getEntityID() {
		return fEntityID;
	}

	string getName() {
		return fName;
	}

	int getHP() {
		return fHP;
	}

	~Entity() {}

};
```