Institute of Technology, Carlow

Report

for

Set

Name: Huu Chi Huynh

Date: 05/02/2022

Tutor: Aine Byrne

Table of Contents

Introduction:

This report is how I prepared, implemented the **Set** data type, and how I implemented the Menu. This report was commissioned by Aine Byrne in the Discrete Structures and Algorithms module in IT Carlow and contains the following sections:

- Thorough description of the problem
- Detailed description of underlying data structures used to solve the problem
- Pseudocode of the Algorithms used
- Detailed description of the extra operations included in the toolbox
- Copy of the Java code
- Description of all the methods/routines which have been used
- Test data used and sample execution screenshots of outputs produced

Thorough description of the problem:

The problem I have been assigned to do is to make a **Set** data structure. The **Set** is data structure with **unique values** inside of it and has a **fixed size**.

I have to implement the **following** operations for the Set:

- add()
- contains()
- removeAll()
- print()

And also suggest and include **2** other operations for the **Set**. Which the 2 operations I chose being:

- remove()
- clear()

All of the operations have to be an option inside of **menu** aswell.

The menu having to interact with the **Set** data structure.

Detailed Description of the underlying data structures used to solve the problem:

I have used the **array** data structure to implement my **Set** data type in C++.

The **array** data structure is used to store multiple values in a variable.

All the values inside of an array must be the same data type as arrays are **stored in consecutive memory cells** and the array has a fixed size.

Each value inside of an array can be indexed by an int, starting from 0 to n. (n being the amount of values inside of an array.)

I have used the **unordered_map** data structure to implement storing of **Sets** in C++.

The **unordered_map** data structure is used to store values with the indexes being strings.

All of the keys in the **unordered_map** is unordered.

The **unordered_map** has a fixed size.

The key and the value of an entry can be any data type whether a string or a class.

Description of all the methods/routines which have been used:

add(key) :

Adds a new key to the keys array in the Set class, incrementing the count variable of the Set class by 1

contains(key) :

Checks if the current Set object has the key parameter by looping through the keys array of the Set class

removeAll(otherSet) :

Removes all the keys in the current set that are also present in the other set by looping through the current set and checking if the key exists in the other set and removing it from the current set's key array.

print(key) :

Displays the current set's keys by looping through it's keys array.

Detailed description of the extra operations included in the toolbox:

remove(key) : [MAIN]
Searches through the Set's keys and if it exists inside of the Set's keys then remove the key from the Set's keys.

equals(otherSet) : [MAIN]

Compares the set calling the function and the other set,

If they have the same count, size and keys then return true else

return false

search(key) : [USED FOR remove()]

Searches through the Set's keys and if it finds the key param then return its index

getSize() :

Return size of the Set

getCount() :

Return count of the Set

clear() :

Clears the Set's keys and sets the count to 0

Pseudocode of the Algorithms used

The following are already initialized inside of a Set object:

```
{
count = 5;
size = 10;
keys[10] = {“Hi”, “Bye”, “Hello”, “Hey”, “Cool”};
}
```

All these methods are in the Set class:

- **function** add(key)

```
{
    if (count < size) {
        if (!contains(key) {
            keys[count] = key;
            count++;
        }
        else {
            print(“Already in”)
        }
    }
    else {
        print(“Set is Full”)
    }
```

}

add(“Sup”)

Object:

{

count = 6;

size = 10;

keys[10] = {“Hi”, “Bye”, “Hello”, “Hey”, “Cool”, “Sup”};

}

- **function** contains(key)

```
{
        for (int i = 0; i < count; i++) {
                if (keys[i] == key) {
                        return true
                }
        }
        return false
}
```

contains (“Hi”)

Object:

{

count = 5;

size = 10;

keys[10] = {“Hi”, “Bye”, “Hello”, “Hey”, “Cool”};

}

Output:

true

- **function** removeAll(remSet)

{

previousKeys = keys

previousCount = count

keys = new Array(size) // New array to store keys that weren't removed

count = 0

for (int i = 0; i < previousCount; i++) {

if (not remSet.contains(previousKeys[i]) {

add(previousKeys[i])

}

}

}

removeAll(self)

Object:

{

count = 0;

size = 10;

keys[10] = {};

}

- **function** print()

{

for (int i = 0; i < count; i++) {

print("[i] = " + keys[i])

}

}

print()

Object:

{

count = 5;

size = 10;

keys[10] = {"Hi", "Bye", "Hello", "Hey", "Cool"};

}

Output:

[0] = Hi

[1] = Bye

[2] = Hello

[3] = Hey

[4] = Cool

- **function** remove(key)

{

```
int indexFound = search(key);
if (indexFound) {
        indexFound = indexFound - 1;
        for (int i = indexFound; i < count - 1; i++) {
                keys[i] = keys[i + 1];
        }
        count--;
        keys[count] = ""
}
```

}

remove(“Hi”)

Object:

{

count = 4;

size = 10;

keys[10] = {“Bye”, “Hello”, “Hey”, “Cool”};

}

// Note: take 1 away from returned value to get index

- **function** search(key)

{

```
int indexFound = 0; // 0 == false
for (int i = 0; i < count; i++) {
        if (keys[i] == key) {
                indexFound = i + 1; // 0 + 1 == true
        }
}
return indexFound;
```

}

search(“Hey”)

Object:

{

count = 5;

size = 10;

keys[10] = { "Hi", "Bye", "Hello", "Hey", "Cool"};

}

Output:

3

- **function** equals(otherSet)

{

if (otherSet.getSize() == size && otherSet.getCount()) {

for (int i = 0; i < count; i++) {

if (not otherSet.contains(keys[i])) {

return false;

}

}

return true;

}

return false;

}

equals(self)

Object:

{

count = 5;

size = 10;

keys[10] = { "Hi", "Bye", "Hello", "Hey", "Cool"};

}

- **function** clear()

{

keys = new Array(size)

count = 0

}

clear()

Object:

{

count = 0;

size = 10;

keys[10] = {};

}

C++ Code

Menu.h

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

#include <iostream>
#include <unordered_map>// Map data type for storing Sets
#include <windows.h> // WinApi header (Colours)
#include <ctime>           // Current time for random
#include <algorithm> // For generating random string
#include "Colour.h"        // Colours for the console

class Menu
{
public:
	~Menu() {
		sets.erase(sets.begin(), sets.end());
	}
	void run();
	void generateSets();
	void createSet();
	void addToSet();
	void printSet();
	void removeAllFromSet();
	void containsInSet();
	void removeFromSet();
	void searchInSet();
	void clearSet();
	void equalsSet();

private:
	HANDLE hConsole; // Console window

	std::unordered_map<std::string, std::unique_ptr<Set>> sets; // Set storage

	bool continueOperation();
	bool findSet(std::string setName);
	std::string askForSet(std::string index);
	void listSets();
	void SendSuccess(std::string successMessage);
	void SendError(std::string errorMessage);
	void SendError(std::string errorMessage, std::string extraMessage);
};
```

Menu.cpp

```
#include "Menu.h"

/// <summary>
/// Generate random strings | Credits: https://stackoverflow.com/a/440240
/// </summary>
/// <param name="length">length of string</param>
```

```
/// <returns></returns>
std::string random_string(size_t length)
{
	auto randchar = []() -> char
	{
		const char charset[] =
			"0123456789"
			"ABCDEFGHIJKLMNOPQRSTUVWXYZ";
		const size_t max_index = (sizeof(charset) - 1);
		return charset[rand() % max_index];
	};
	std::string str(length, 0);
	std::generate_n(str.begin(), length, randchar);
	return str;
}

void Menu::run() {
	hConsole = GetStdHandle(STD_OUTPUT_HANDLE);

	int selection = 0;
	while (true) // MAIN LOOP
	{
		while (true) // Loop to check if user has inputted correct data type
		{
			// Starting text
			std::cout << "Text based Menu for Set class by Chi\n\t1) Create
Set\n\t2) Add to Set\n\t3) Print Set\n\t4) Remove All from Set\n\t5) Contains in
Set\n\t6) Remove From Set\n\t7) Search in Set\n\t8) Clear Set\n\t9) Equals Set\n\t10)
Generate 3 - 5 Sets\n\nEnter an Option: ";
			std::cin >> selection;
			if (!std::cin.fail()) {
				break;
			}
			SendError("Error: Numbers only.\n\n");
			std::cin.clear();
			std::cin.ignore(256, '\n');
		}
		std::cout << "\n";
		switch (selection) // Menu system
		{
		case 1:
			createSet();
			break;
		case 2:
			addToSet();
			break;
		case 3:
			printSet();
			break;
		case 4:
			removeAllFromSet();
			break;
		case 5:
			containsInSet();
			break;
		case 6:
			removeFromSet();
			break;
```

```
                case 7:
                        searchInSet();
                        break;
                case 8:
                        clearSet();
                        break;
                case 9:
                        equalsSet();
                        break;
                case 10:
                        generateSets();
                        break;
                default:
                        SendError("Invalid Selection\n\n");
                        break;
                }
        }
}

/// <summary>
/// Make 3 to 5 sets with a size of 1 to 5 with a random amount of keys
/// </summary>
void Menu::generateSets()
{
        std::srand(static_cast<unsigned>(time(nullptr)));             //Initalise random
seed
        int amountOfSets = 3 + rand() % 3;
        for (int i = 0; i < amountOfSets; i++) { // Generate 3 - 5 sets
                std::string randomString = random_string(5);    // Random 5 characters
                int randomSize = 1 + rand() % 5;                                        //
Make set with random 1 to 5 size
                sets[randomString] = std::make_unique<Set>(randomSize);// Create a unique
pointer for the set with size of random
                for (int i = 0; i < randomSize - rand() % randomSize; i++) {
                        sets[randomString]->add(random_string(5));
                }
        }
        SendSuccess(std::to_string(amountOfSets) + " Sets have been generated.\n\n");
}

/// <summary>
/// Ask user to input a name for the set,
/// if another set is found with the name then
/// repeat the question.
/// If not then continue to the next question
/// Ask the user for the set size.
/// If the inputted value is not a number then
/// repeat the question.
/// If not then create the set with the given values with
/// Set name being the index for the unordered_map to reference by and
/// Set Size being the size of the set
/// </summary>
void Menu::createSet() {
        std::string setName;
        int setSize = 1;

        while (true) {
                std::cout << "Enter the name of the new set: ";
```

```
		std::cin >> setName;
		if (!findSet(setName)) { // Check if set isn't inside of the
unordered_map
			break;
		}
		SendError("\nAnother Set with the same name was already made.\n\n");
		if (!continueOperation()) {
			return; // EXIT FUNCTION
		}
	}
	while (true) {
		std::cout << "\nEnter the size of the new set: \n";
		std::cin >> setSize;
		if (!std::cin.fail()) {
			break;
		}
		SendError("Error: Numbers only.\n");
		std::cin.clear();
		std::cin.ignore(256, '\n');
	}
	sets[setName] = std::make_unique<Set>(setSize); // Create a unique pointer for
the set
	SendSuccess("Set: " + setName + " | Size: " + std::to_string(setSize) + " was
created.\n\n");
}

/// <summary>
/// Ask user for set and ask them for a value to add into the Set,
/// if the Set is full, then display that it's full
/// if the Set already has the value, then display that it already has the value
/// if the Set doesn't have the value and it isn't full then display that it was
successful
/// </summary>
void Menu::addToSet() {
	listSets();
	std::string setName;
	bool continueAdd = false;

	setName = askForSet("");
	if (setName == "") { return; }

	while (true)
	{
		std::string value;
		std::cout << "Please enter the value that you are going to put into the
Set '" << setName << "':\n";
		std::cin >> value;
		std::cout << "\n";
		switch (sets[setName]->add(value)) {
		case SetResult::Success:
			SendSuccess("Successfully added '" + value + "' to Set: " +
setName + "\n");
			break;
		case SetResult::AlreadyIn:
			SendError("Unable to add '" + value + "' to Set: " + setName +
"\n", "Key is already inside of the Set.\n");
			break;
		case SetResult::Full:
```

```
                        SendError("Unable to add '" + value + "' to Set: " + setName +
"\n", "Set is full.\n");
                        break;
                }
                if (!continueOperation()) {
                        return; // EXIT FUNCTION
                }
        }
}

/// <summary>
/// Ask the user for a Set and
/// Call the print function in the Set class
/// </summary>
void Menu::printSet() {
        listSets();
        std::string setName;

        setName = askForSet("");
        if (setName == "") { return; }

        sets[setName]->print();
        std::cout << "\n";
}

/// <summary>
/// Ask for 2 Sets and remove all the keys inside of set1 that are also in set2
/// </summary>
void Menu::removeAllFromSet() {
        listSets();
        std::string setName1;
        std::string setName2;

        setName1 = askForSet("[1] ");
        if (setName1 == "") { return; }
        setName2 = askForSet("[2] ");
        if (setName2 == "") { return; }

        SendSuccess("Removed " + std::to_string(sets[setName1]-
>removeAll(sets[setName2])) + " keys inside of the Set: " + setName1 + "\n\n");
}

/// <summary>
/// Ask the user for Set and find the value they inputted inside of the Set
/// </summary>
void Menu::containsInSet() {
        listSets();
        std::string setName;
        std::string input;

        setName = askForSet("");
        if (setName == "") { return; }

        std::cout << "Enter a value that will want to find in the Set: " << setName <<
"\n";
        std::cin >> input;

        std::cout << "\n";
```

```
	if (sets[setName]->contains(input)) {
		SendSuccess("Found '" + input + "' inside of the Set: " + setName +
"\n");
	}
	else {
		SendError("'" + input + "' wasn't found inside of the Set: " + setName +
"\n");
	}
	std::cout << "\n";
}

/// <summary>
/// Ask user for set.
/// If they have entered a valid Set then
/// Ask user to delete a key from the Set,
/// If the key isn't found then warn them
/// If the key is found then remove the key from the Set and send them a success
message
/// </summary>
void Menu::removeFromSet() {
	listSets();
	std::string setName;
	std::string input;

	setName = askForSet("");
	if (setName == "") { return; }
	sets[setName]->print();
	std::cout << "\n";

	while (true) {
		while (true) {
			std::cout << "Enter a value you want to remove from the Set: " <<
setName << "\n";
			std::cin >> input;
			if (sets[setName]->remove(input)) {
				break;
			}
			SendError("'" + input + "' wasn't found inside of the Set: " +
setName + "\n\n");
			if (!continueOperation()) {
				return;
			}
		}
		SendSuccess("'" + input + "' was removed from the Set: " + setName +
"\n\n");
		if (!continueOperation()) {
			return;
		}
	}

}

/// <summary>
/// Ask user for set,
/// Ask user for key to find in set.
/// If key is found in Set then print out it's index position
/// Else send error
/// </summary>
```

```
void Menu::searchInSet() {
	listSets();
	std::string setName;
	std::string input;

	setName = askForSet("");
	if (setName == "") { return; }
	sets[setName]->print();
	std::cout << "\n";

	int keyIndex = 0;

	while (true) {
		std::cout << "Enter a value that will want to find it's index in the Set:
" << setName << "\n";
		std::cin >> input;
		keyIndex = sets[setName]->search(input);
		if (keyIndex) {
			break;
		}
		SendError("'" + input + "' wasn't found inside of the Set: " + setName +
"\n\n");
		if (!continueOperation()) {
			return;
		}
	}
	SendSuccess("'" + input + "' was found inside of the Set: " + setName + " at
index position: " + std::to_string(keyIndex - 1) + "\n\n");
}

/// <summary>
/// Ask user for set and clear the set
/// </summary>
void Menu::clearSet() {
	listSets();
	std::string setName;

	setName = askForSet("");
	if (setName == "") { return; }
	sets[setName]->clear();

	SendSuccess("Set: '" + setName + "' was cleared!\n\n");
}

/// <summary>
/// Ask user for two set names and call the Set equals method with the second Set as
the parameter
///    If the first set equals second set then
/// Send success else Send error
/// </summary>
void Menu::equalsSet() {
	listSets();
	std::string setName1;
	std::string setName2;

	setName1 = askForSet("[1] ");
	if (setName1 == "") { return; }
	setName2 = askForSet("[2] ");
```

```
	if (setName2 == "") { return; }

	if (sets[setName1]->equals(sets[setName2])) {
		SendSuccess("Set: '" + setName1 + "' equals to Set: " + setName2 +
"!\n\n");
	}
	else {
		SendError("Set: '" + setName1 + "' does not equals to Set: " + setName2 +
"!\n\n");
	}
}

/// <summary>
/// Ask user if they would like to continue operation
/// </summary>
/// <returns>True = continue/ False = stop</returns>
bool Menu::continueOperation() {
	bool continueOp = false;
	while (true) {
		std::cout << "Would you like to continue:\n\t0 = No\n\t1 = Yes\n\nEnter
an Option: ";
		std::cin >> continueOp;
		if (!std::cin.fail()) {
			break;
		}
		SendError("\nError: 0 or 1\n\n");
		std::cin.clear();
		std::cin.ignore(256, '\n');
	}
	std::cout << "\n";
	return continueOp;
}

/// <summary>
/// Checks if the set is available in the unordered_map
/// </summary>
/// <param name="setName">Set Name</param>
/// <returns>True = Found | False = not Found</returns>
bool Menu::findSet(std::string setName) {
	return sets.find(setName) != sets.end();
}

/// <summary>
/// Ask the user for a set name,
/// If a set is found then break out of the loop,
/// If there is no set then the user can exit out of the loop
/// </summary>
/// <param name="index"></param>
/// <returns></returns>
std::string Menu::askForSet(std::string index) {
	std::string setName;
	while (true)
	{
		std::cout << index << "Enter the name of your set: ";
		std::cin >> setName;
		// This if statement checks if the name of the set is in the
unordered_map
		if (findSet(setName)) {
```

```
                        break;
                }
                SendError("\nSet not found.\n\n");
                if (!continueOperation()) {
                        return ""; // EXIT FUNCTION
                }
        }
        std::cout << "\n";
        return setName;
}

/// <summary>
/// Loop through all of the keys inside of the sets unordered_map
/// </summary>
void Menu::listSets() {
        std::string setsString;
        for (std::unordered_map<std::string, std::unique_ptr<Set>>::iterator iter =
sets.begin(); iter != sets.end(); ++iter)
        {
                setsString = setsString + "||" + iter->first;
        }
        setsString = setsString + "||\n\n";
        std::cout << "Set Names: " << setsString;
}

/// <summary>
/// Send success message with green colour
/// </summary>
/// <param name="successMessage">Raw string</param>
void Menu::SendSuccess(std::string successMessage) {
        SetConsoleTextAttribute(hConsole, GREEN);
        std::cout << successMessage;
        SetConsoleTextAttribute(hConsole, WHITE);
}

/// <summary>
/// Send error message with red colour
/// </summary>
/// <param name="errorMessage">Raw string</param>
void Menu::SendError(std::string errorMessage) {
        SetConsoleTextAttribute(hConsole, RED);
        std::cout << errorMessage;
        SetConsoleTextAttribute(hConsole, WHITE);
}

/// <summary>
/// Send Error message with red and grey colour
/// </summary>
/// <param name="errorMessage">Raw string</param>
/// <param name="extraMessage">Raw string</param>
void Menu::SendError(std::string errorMessage, std::string extraMessage) {
        SetConsoleTextAttribute(hConsole, RED);
        std::cout << errorMessage;
        SetConsoleTextAttribute(hConsole, GREY);
        std::cout << extraMessage;
        SetConsoleTextAttribute(hConsole, WHITE);
}
```

Set.h

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

enum class SetResult {
	Success,
	Full,
	AlreadyIn
};

class Set
{
public:
	Set() {
		this->size = 1;
		keys = (new std::string[1]);
	};
	Set(int size) {
		this->size = size;
		keys = (new std::string[size]);
	};

	~Set() {
		delete[] keys;
	}
	SetResult add(std::string key);
	bool contains(std::string key);
	int removeAll(std::unique_ptr<Set> &otherSet);
	void print();
	int search(std::string key);
	bool remove(std::string key);
	void clear();
	bool equals(std::unique_ptr<Set>& otherSet);

	const int getSize() { return size; }
	const int getCount() { return count; }

private:
	int size = 0;
	int count = 0;
```

```
	// When debugging, the size of the keys are set from the constructor
	std::string* keys;
};
```

Set.cpp

```
#include "Set.h"

/// <summary>
/// Check if the count is less than size,
/// If not then return SetResult::Full.
/// If less then check if the key is not inside of the keys array.
/// If found then return SetResult::AlreadyIn.
/// Else add the key to the keys array and increase the count returning
SetResult::Success
/// </summary>
/// <param name="key"></param>
/// <returns></returns>
SetResult Set::add(std::string key) {
	if (count < size)
	{
		if (!contains(key))
		{
			keys[count] = key;
			count++;
			return SetResult::Success; // Returning this enum for Menu app
		}
		else {
			return SetResult::AlreadyIn;
		}
	}
	else {
		return SetResult::Full;
	}
}

/// <summary>
/// Loops through the keys array using linear search
/// checking if the key param is equal to the looped key
/// If the key is found inside of the array then return true else false
/// </summary>
/// <param name="key"></param>
/// <returns>True = found | False = not found</returns>
bool Set::contains(std::string key) {
	for (int i = 0; i < count; i++) {
		if (keys[i] == key) {
			return true;
		}
	}
	return false;
}

/// <summary>
/// Keep reference of previous keys array and previous count.
/// Reset the current keys and count.
```

```
/// Loop through the previous keys array and check if it is inside of the other set.
/// If the key isnt found inside of the other set then add that key to keys array
/// After finished looping through previous keys array,
/// Delete the previous keys array and return the amount of keys removed
/// </summary>
/// <param name="otherSet">Other set to compare and remove the current Set's
keys</param>
/// <returns>Amount of keys removed</returns>
int Set::removeAll(std::unique_ptr<Set> &otherSet) {
	std::string* previousKeys = keys;
	int previousCount = count;
	count = 0;
	keys = (new std::string[size]);
	for (int i = 0; i < count; i++) {
		bool found = otherSet->contains(previousKeys[i]);
		if (!found) { // If the other set doesn't contain the key then add the
key to the new set
			add(previousKeys[i]);
		}
	}
	delete[] previousKeys; // Delete previous keys array
	return previousCount - count; // Amount of keys removed from the keys array
}

/// <summary>
/// Go through the keys array printing all of the Set contents
/// </summary>
void Set::print() {
	std::cout << "Set contents: \n";
	for (int i = 0; i < count; i++) {
		std::cout << "[" << i << "]" << " = " << keys[i] << "\n";
	}
}

/// <summary>
/// Searches through the keys array using linear search and if the key passed in the
function is found then
/// return the index of the key + 1
///
/// TAKE 1 away from RESULT
/// </summary>
/// <param name="key"></param>
/// <returns>keyIndex + 1 [Take 1 away from it to get index]</returns>
int Set::search(std::string key) {
	int indexFound = 0; // If key isn't found then indexFound will be 0 which is
false
	for (int i = 0; i < count; i++) {
		if (keys[i] == key) {
			indexFound = i + 1;
			break;
		}
	}
	return indexFound; // - 1 if in use
}

/// <summary>
/// Loop from its index to count - 1 replacing the current key with the next key
/// </summary>
```

```
/// <param name="key"></param>
/// <returns></returns>
bool Set::remove(std::string key) {
	int indexFound = search(key); // No need to check if the count is equal to or
below 0
	if (indexFound) {
		indexFound -= 1;
		// Loop from the indexFound to the count - 1 moving the key of i + 1 to
index i
		for (int i = indexFound; i < count - 1; i++) {
			keys[i] = keys[i + 1];
		}
		count--;

		// The count index will always be the an empty string when this happens.
		// And to ensure that the program doesn't index outside of the keys array
		keys[count] = "";
		return true; // Return if remove is successful
	}
	return false;
}

/// <summary>
/// Reset count to 0 and
/// Delete all keys array and create a new one
/// [Not to be used with removeAll]
/// </summary>
void Set::clear() {
	count = 0;
	delete[] keys;
	keys = (new std::string[size]);
}

/// <summary>
/// Checks if the current set is equal to the other Set by
/// Looping through the current Set and checking if the other set doesn't contain the
key
/// </summary>
/// <param name="otherSet"></param>
/// <returns></returns>
bool Set::equals(std::unique_ptr<Set>& otherSet) {
	if (otherSet->getCount() == count && otherSet->getSize()) {
		for (int i = 0; i < count; i++) {
			if (!otherSet->contains(keys[i])) {
				return false;
			}
		}
		return true;
	}
	return false;
}
```

Main.cpp

```
/// <summary>
```

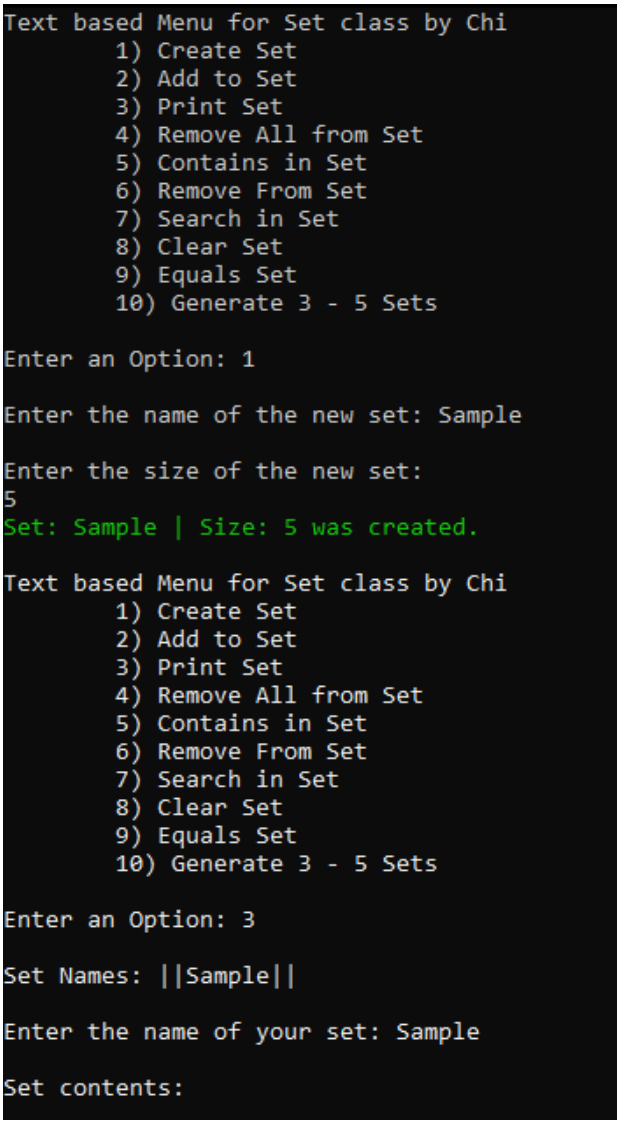
```
/// Name: Huu Chi Huynh
/// Date: 07/02/2022
/// </summary>

#include "Menu.h"
#include "Set.h"
#include <iostream>

int main() {
	Menu menu = Menu{};
	menu.run();
	system("pause");
	return 0;
}
```

Colour.h

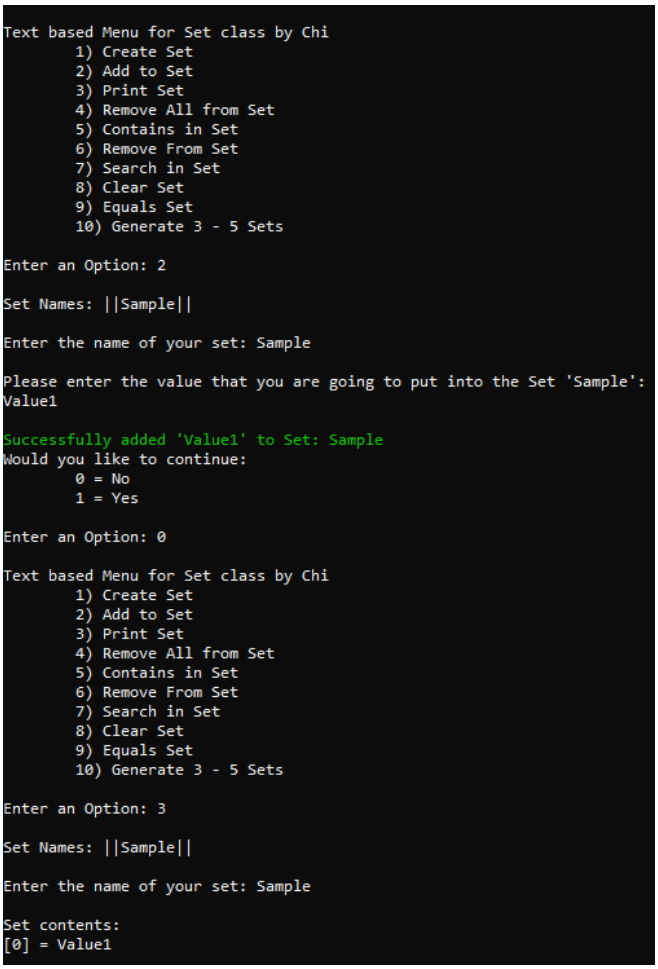
```
#pragma once
#define RED 4
#define GREY 8
#define GREEN 10
#define WHITE 15
```

Test data used and sample execution screenshots of outputs produced:

Create Set:



Add to Set and print:



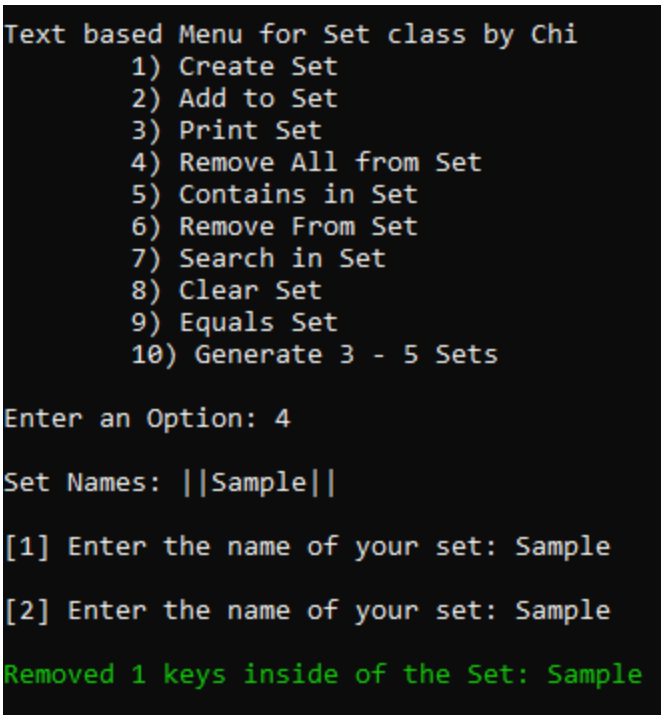
Attempt to add same value to Set:

```
Text based Menu for Set class by Chi
        1) Create Set
        2) Add to Set
        3) Print Set
        4) Remove All from Set
        5) Contains in Set
        6) Remove From Set
        7) Search in Set
        8) Clear Set
        9) Equals Set
        10) Generate 3 - 5 Sets

Enter an Option: 2

Set Names: ||Sample||

Enter the name of your set: Sample

Please enter the value that you are going to put into the Set 'Sample':
Value1

Unable to add 'Value1' to Set: Sample
Key is already inside of the Set.
```

Contains:

```
Text based Menu for Set class by Chi
        1) Create Set
        2) Add to Set
        3) Print Set
        4) Remove All from Set
        5) Contains in Set
        6) Remove From Set
        7) Search in Set
        8) Clear Set
        9) Equals Set
        10) Generate 3 - 5 Sets

Enter an Option: 5

Set Names: ||Sample||

Enter the name of your set: Sample

Enter a value that will want to find in the Set: Sample
Value1

Found 'Value1' inside of the Set: Sample
```

RemoveAll:


Text based Menu for Set class by Chi
1) Create Set
2) Add to Set
3) Print Set
4) Remove All from Set
5) Contains in Set
6) Remove From Set
7) Search in Set
8) Clear Set
9) Equals Set
10) Generate 3 - 5 Sets
Enter an Option: 4
Set Names: ||Sample||
[1] Enter the name of your set: Sample
[2] Enter the name of your set: Sample
Removed 1 keys inside of the Set: Sample

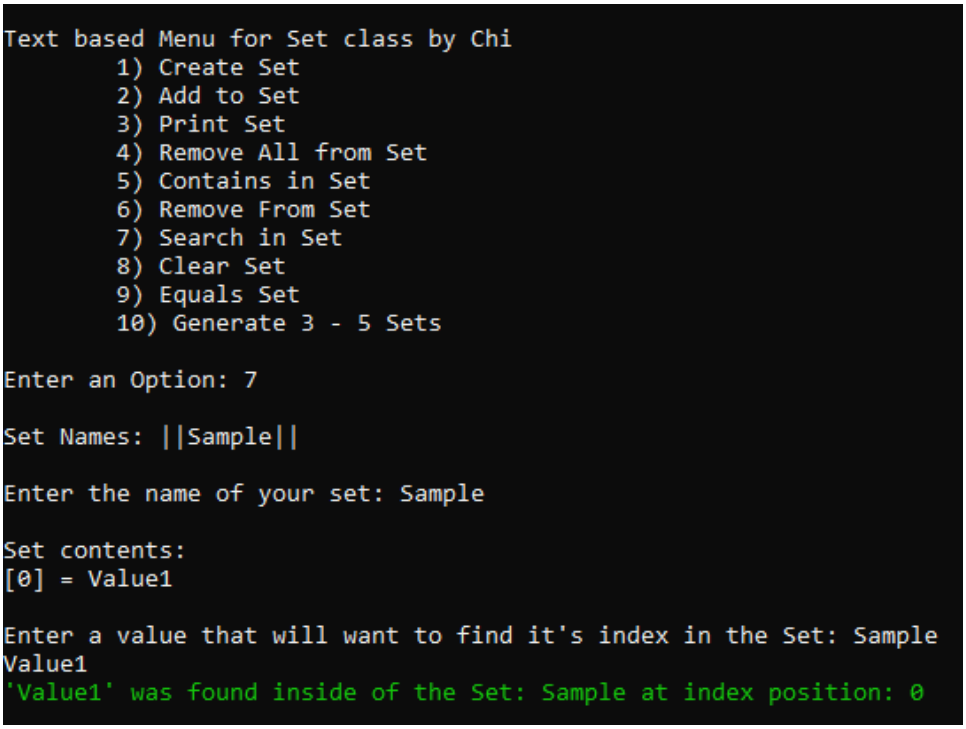
Equals:

```
Set Names: ||Sample||Example||

Enter the name of your set: Example

Please enter the value that you are going to put into the Set 'Example':
Value1

Successfully added 'Value1' to Set: Example
Would you like to continue:
        0 = No
        1 = Yes

Enter an Option: 0

Text based Menu for Set class by Chi
        1) Create Set
        2) Add to Set
        3) Print Set
        4) Remove All from Set
        5) Contains in Set
        6) Remove From Set
        7) Search in Set
        8) Clear Set
        9) Equals Set
        10) Generate 3 - 5 Sets

Enter an Option: 9

Set Names: ||Sample||Example||

[1] Enter the name of your set: Sample

[2] Enter the name of your set: Example

Set: 'Sample' equals to Set: Example!
```

Remove:

```
Text based Menu for Set class by Chi
        1) Create Set
        2) Add to Set
        3) Print Set
        4) Remove All from Set
        5) Contains in Set
        6) Remove From Set
        7) Search in Set
        8) Clear Set
        9) Equals Set
        10) Generate 3 - 5 Sets

Enter an Option: 6

Set Names: ||Sample||Example||

Enter the name of your set: Example

Set contents:
[0] = Value1

Enter a value you want to remove from the Set: Example
Value1
'Value1' was removed from the Set: Example

Would you like to continue:
        0 = No
        1 = Yes

Enter an Option: 0

Text based Menu for Set class by Chi
        1) Create Set
        2) Add to Set
        3) Print Set
        4) Remove All from Set
        5) Contains in Set
        6) Remove From Set
        7) Search in Set
        8) Clear Set
        9) Equals Set
        10) Generate 3 - 5 Sets

Enter an Option: 3

Set Names: ||Sample||Example||

Enter the name of your set: Example

Set contents:
```

Search:


Text based Menu for Set class by Chi
1) Create Set
2) Add to Set
3) Print Set
4) Remove All from Set
5) Contains in Set
6) Remove From Set
7) Search in Set
8) Clear Set
9) Equals Set
10) Generate 3 - 5 Sets
Enter an Option: 7
Set Names: ||Sample||
Enter the name of your set: Sample
Set contents:
[0] = Value1
Enter a value that will want to find it's index in the Set: Sample
Value1
'Value1' was found inside of the Set: Sample at index position: 0