

**RMIT University Vietnam**

**School of Science and Technology**

**EEET2482 Software Engineering Design**

**Laboratory 1 Report – Subtraction Calculator**

**Lecturer: Dr. Minh Ngoc Dinh**

**Group: Group 123**

**Hoang Minh Quan s3754450**

**Vu Thanh Quang s3760192**

**Le Ngoc Cao Son s3752823**

**Create Date: Nov 8, 2019**

# Introduction

In general, this laboratory actitivity requires us to cover and apply some basic programming topics of C++: procedural programming, iostream, strings, data types, loops and functions.

Procedural programming is a top-down programming paradigm that approaches and solves the problem by instructing the computer to follow step-by-step logical lists of procedures, which can be also known as functions, or routines.

In **<iostream>** standard library, **cout <<** and **cin** **>>** are functionality for writing to console and reading from standard input like keyboards.

Strings in C++ can be treated as objects of **string class** that, by definition, are sequences of characters. Therefore, a number of useful features and methods provided by the **string** class come in quite handy when dealing with strings.

Data types are simply similar to that of C, including primitive data types like: **int** for integer literals**, float** or **double** for floating point literals**, char** for integral number interpreted to ASCII characters, and **bool** for logical value either true or false**.**

Functions in C++ are nearly the same as that concept in other programming languages: small instructions performing a specific task that can be used (or called) multiple times. They take inputs (or no inputs at all) and return a value (or nothing for **void** functions) that can be used at somewhere else in the program when being called. The value returned type can be any of the data types in C++. Functions help us to remove redundant repetitive tasks and reduce the size of the program.

**While loop** and **for loop** are two common types of control flow for repetitive execution of a set of instructions under a certain condition.

In particular for this laboratory, we are expected to create a subtraction calculator that takes one-line input in an expression of **[arg1] op [arg2]**, in which **[arg1]** and **[arg2]** are integers in range of [-32,768 to 32,767] and **op** is a valid operator of [+, -, \*, /, %]**.** The program has to provide a simple user interface that continuously ask and prompt users to input their expression, from which print out its results or display a given form when users want the program to stop. It is also tested for the ability to check and display an appropriate error message to the console for each following circumstance of wrong user’s input:

1. *Valid Number Input*: Only test whether the input arguments can be converted into valid integers.
2. *Number Input Check Range*: The value of **[arg1]** and/or **[arg2]** is checked for being in range of [-32,768 to 32,767].
3. *Valid Operator Input*: The value of **op** is checked for being of [+, -, \*, /, %].
4. *Division by 0*: the value of **op** is ‘/’ and **[agr2]** is 0.
5. *Dummy Variable Check*: The one-line input contains extra variables rather than that in correct format **[arg1] op [arg2]**.

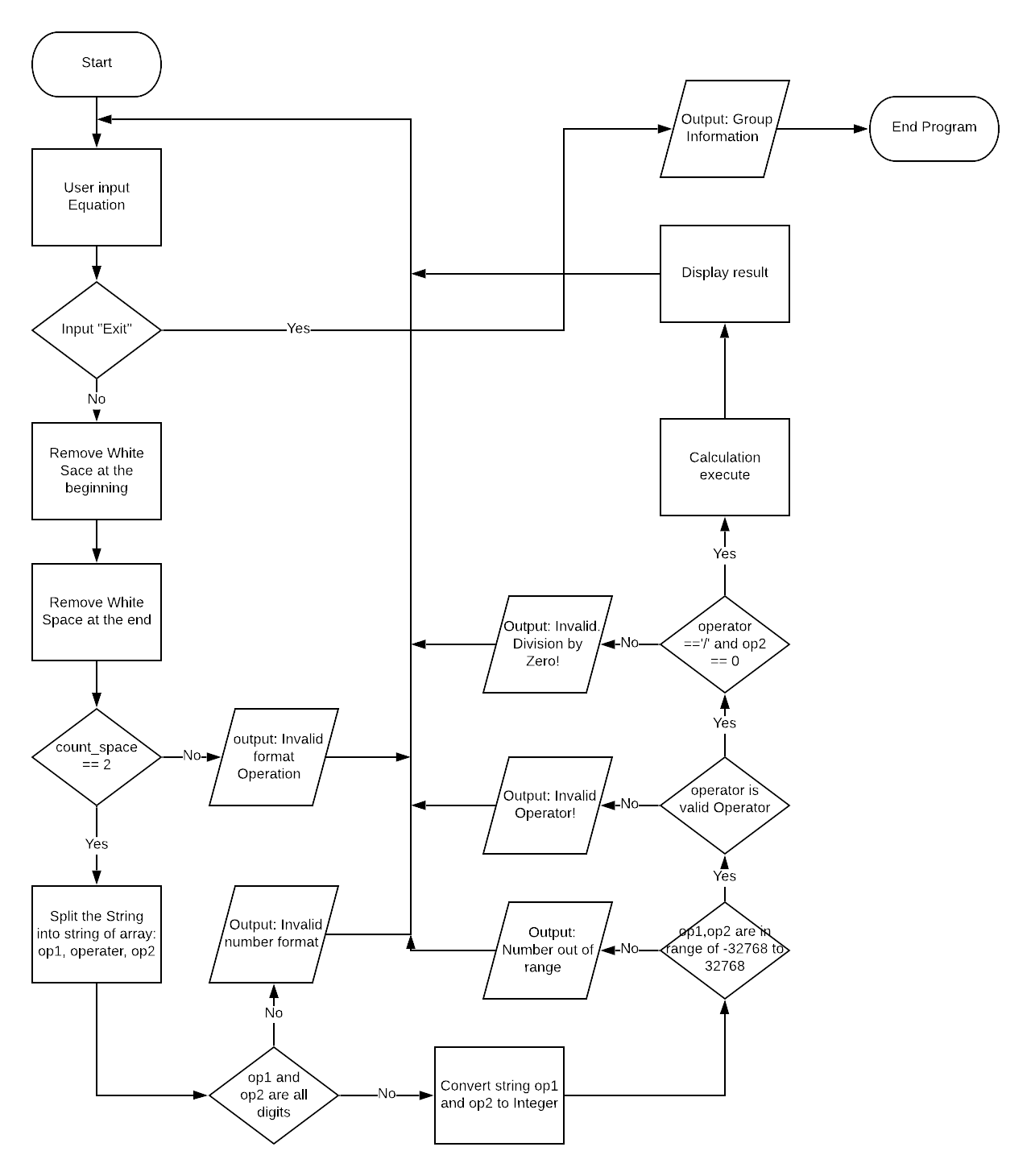
From this requirement, we intend to solve the problem using top-down design, in which the program is broken down to smaller functions. For the user interface, an infinite loop will be used to keep asking for the user’s input using **cin >>** functionality of **<iostream>** and permission to continue. The loop has a break when user input “Exit” and then, display a form. Inside the loop, our program first checks and ensures that all the arguments of the one-line input must meet the 5 given specifications, using suitable features from **string** class to seperate and check for the validation of **arg1**, **arg2**, and **op**. For each function that checks for the error, there will be a false returned if any argument is invalid. Then, arguments **arg1** and **arg2** will be converted to corresponding integers. Depending on the value of **op**, there will be a function that performs the operation taking **arg1** and **arg2** as its arguments and then, returning the result. The result will be printed out to the console using **cout <<** functionality.

The remaining report’s structure will cover:

* 1. Flowchart: represents the program design including functions and subfunctions to illustrate our algorithm.
  2. Discussion & Results: This section will be divided into 4 subsections, in which we will indicate how we approached and analyzed the problem (subsection 1), how we work on the user interface (I/O) (subsection 2), how all the error checks were handled (subsection 3) and finally, how we connected all the functions and re-organized the entire source code for our program (subsection 4.
  3. Conclusion: A brief paragraph of how our program addressed and related to
  4. References: all the reference sources we have used to help us throughout this laboratory will be listed here in IEEE style.
  5. Appendices: this part will refer to our source code for the program.

# Flowchart

The Flowchart following present the structure of our software to verify the input mathematic expression and execute it. In this diagram the logical order of the processing of our software is presented. The program will take the user expression input as a string and validate the parts for operands and operator before executing the calculation. If the input string is invalid, the program will notice to the user the error and ask the user to input again. If the input expression is correct, the result will be calculated and display in the console and ask the user to input another expression. The program only ends when the user input “Exit” in the command line.



# Discussion & Results

To make sure the program is readable and efficient structure with well top design, our code is written as functions and each function will not exceed 20 lines of code. Therefore, our main is kept as short as possible. To avoid the usage of magic value, we define constant and global value with meaningful name instead of using direct value in the code. Moreover, to make our code more better practice, instead of define all function in the namespace std library, we call the method in the function only when we need it for the specific case, such as std::cout and std::string.

## 3.1 Program Analysis and Design

### 3.1.1 Get input user

Before asking the user to input the mathematic expression, we display the correct format of an expression in the console for the user. After that we will print out one line to tell the user to input the expression or use can type in “Exit” to end the program. Moreover, we notice the user that they cannot input more than 12 characters because if the user input more than that the expression will be in one of five errors that we tended to check.

### 3.1.2 Remove White Space at the end and at the beginning

In same cases, user could accidentally type the white space at the beginning or at the end of the expression, it could affect to the checking error format of the expression. In this case, if the expression is correct, we still must execute the calculation. Therefore, we will remove the white space at the beginning and at the end of the user input. We divided it into two functions:

In the function to remove the white space at the beginning, we loop from the beginning of the input string until we detect the first character. Then we erase the from the beginning of the string to the index before the first character [1]. If there no white space at the beginning, the function will not erase anything from the string. The same functional for the function to remove white space at the end of the input string.

### 3.1.3 Count the number within the input after removing space

After ensuring that there are no unnecessary white space exists in the input string, we check the number of the total white spaces in the expression. The correct format “op1 op op2” for the input equation defines that there is only one white space between the operand and the operator. So, we must check there are exact 2 white spaces in the input string.

In this function, we use for looping the loop throw each index of the string to check if that index is the white space. If it is a white space, the count\_space will be increased by one.

If the number of white spaces within the string after removing the ending and beginning white space is equal to 2 then the code will check other possible error in the next stage. Otherwise, the program will display error and ask the user to input again.

### 3.1.4 String split then store in the array of string

After ensuring that there two white spaces within the string. We must split the string by the white spaces [2]. We split the string into 3 string and store it in the array of string.

The size of the array of string is defined by 3, because we split the string by 2 white spaces. This function loops through the string and store each character, which is not the whitespace, into the current index of the string array. After the white space, the index will be increased by 1, the characters will be stored in the next index of string array.

### 3.1.5 Convert the first operand and the second operand into Integer

After splitting the initial string equation into string array, the first index and the third index supposed to be the operand. So, we must check if in the first index of the string array is included only digit or not. If there are not all digit, there will be the operand error included in the 3.3 Error handling). If there are only digits, we must convert the string into integer.

The first character could be the ‘+’ symbol or the –‘symbol’ to define the operand has the positive or nagative value. For the Integer value of the operand, we loop through the string to get each digits first. To get the integer value of each digit from string, we use the ASCII value of that digit and subtract it to 48 because the ASCII value of 0 to 9 is 48 to 57 respectively [3]. After each loop, we add the value of the digit to the sum multiply by 10

### 3.1.6. Execute the equation based on three value op1, operand and the op2

After splitting the input expression into two operands and an operator and converting the operand strings into a valid integer, the program will check for possible errors of the operands and operator (referred to session 3.3 Error Handling). By checking for errors, the program can ensure that the input expression is valid and will execute it.

The expression execute function will take two integer variable has the value of two operands and a string variable has the value of the operator. The range of the operand must be in range of [-32,768 to 32,767] and the operator can be the plus, minus, divide, mode, or multiply symbol which will represent the corresponding calculation.

### 3.1.7 Return the result of the equation and ask user to input other equation

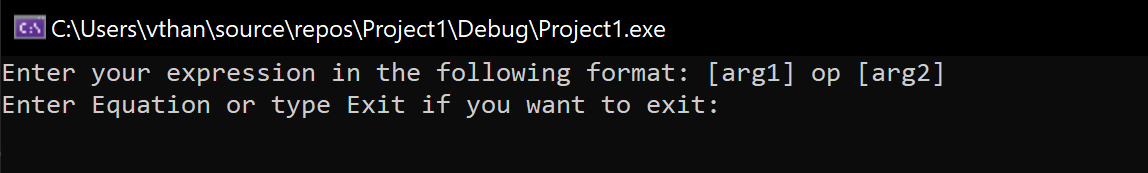
After executing the calculation, the program will display the result, which has the variable type of integer, in the console. If the calculation is in the case of division, the result will be rounded to integer automatically. Then in the console, the program will ask user to input another expression and continue to check and execute the input expression in the same flow as above.

### 3.1.8 Repeat the process above until user input “Exit”

The program will repeat to ask user to input mathematic expression again or tell the user to exit if he wants to exit the program. After the program exits, the group information and group details will be display in the console (referred to session 3.2 User InterFace).

3.2 User Interface (I/O)

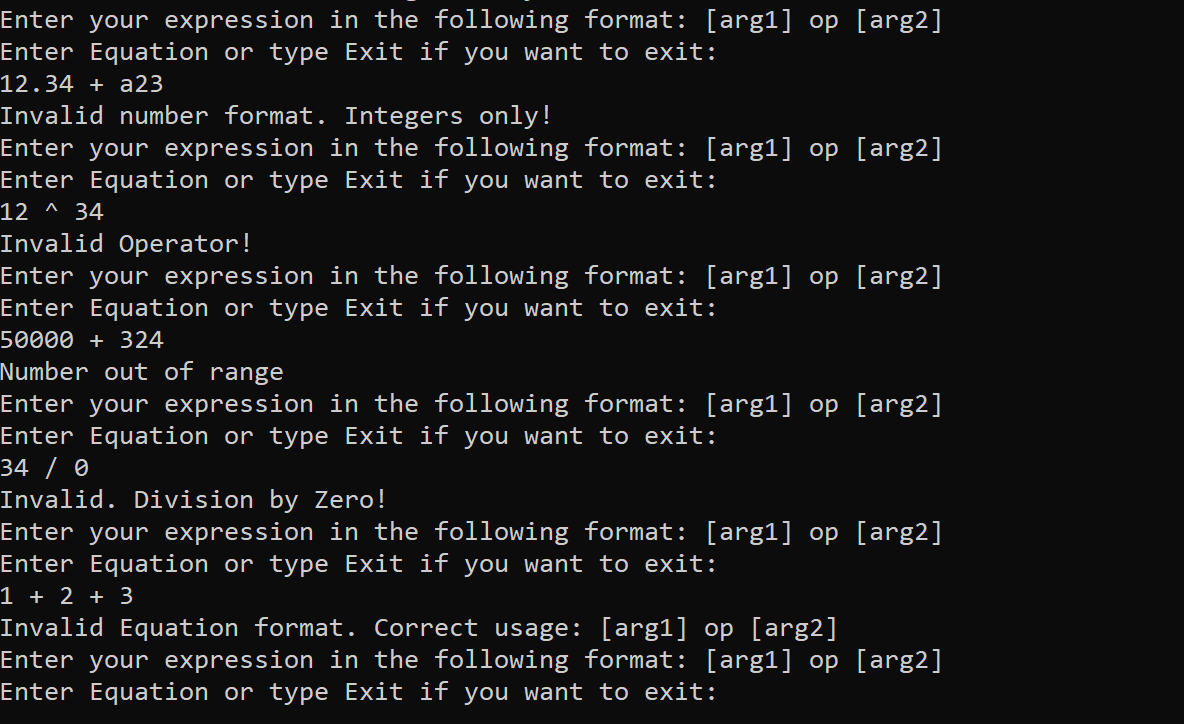
### 3.2.1 Beginning of the console when the program is executed



Before getting the input from the user, we must clarify the correct format of the input expression, which has 2 operands and one operator divided by exactly one white space. The output on the console before getting the input from user also tell him how to exit the program.

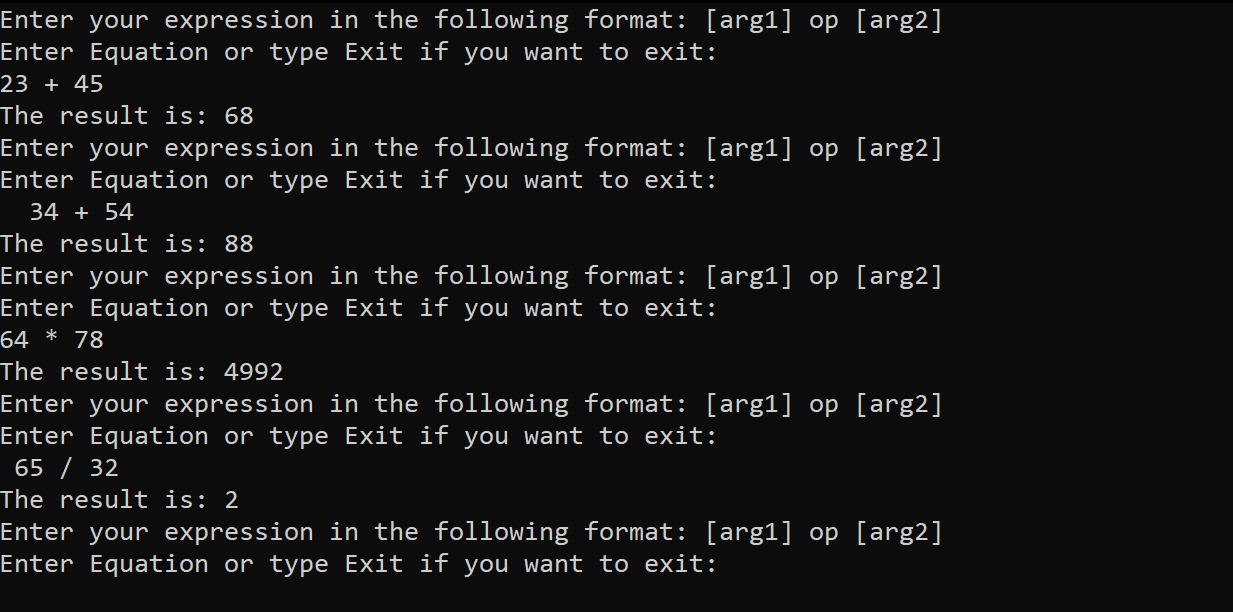
### 3.2.2 Display error message if user put invalid input.

The program will display the kind of error according to the case the invalid input expression. Our software will check for 5 errors before verifying the input expression: Invalid Number Input, Number Input Range Check, Invalid Operator, Division by 0, and Dummy Variable Check. For each error check, if the input is in one of these cases of error, the program will display like the example following in the corresponding order:



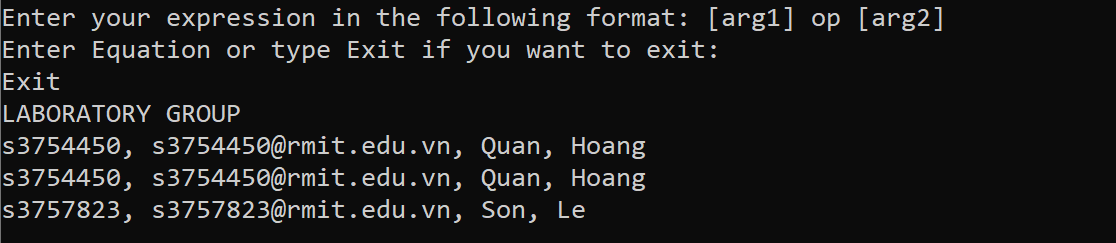
### 3.2.3 Otherwise display the result of the input equation.

After the display does not display error for the specific cases, the program will calculate the result of the expression and print it out on the console for the user. Then ask the user to input again or tell him to type in “Exit” in the command line to exit



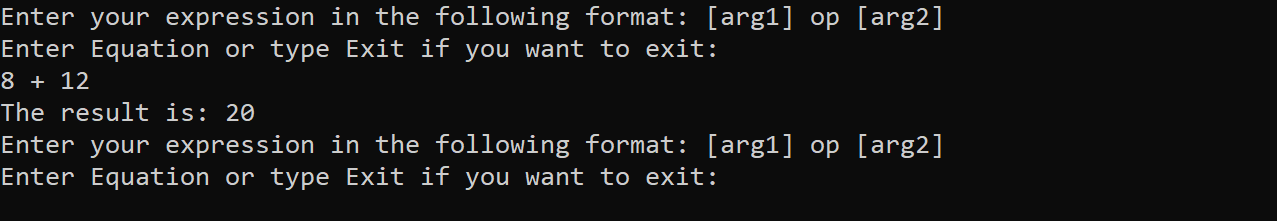
### 3.2.4 If input “Exit” display Course name and team information before display the ending message.

If the user wants to end the program, the console will show the option to exit the program by type in the command line “Exit”. After the command line “Exit” to the console, the console will display the our Group Name ༼ つ ◕\_◕ ༽つᕦ(⇀‸↼‶)ᕤ and our group member details (Unfortunately, the cout function of C++ can not display the special Unicode characters from other languages)



### 3.2.6 Otherwise, repeat the process above until the user input “Exit”

If the user writes in the command line “Exit”, then the program will continue to ask the user to input the mathematic expression and tell him how to end the program



# 3.3 Error Handling.

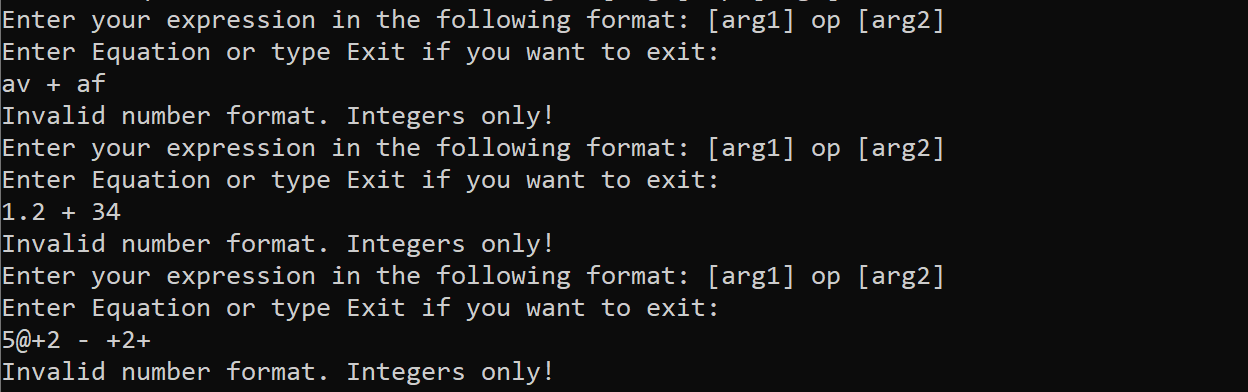
In this session, we clarify the error check cases of our software and explain the algorithm for each case of the error.

# 3.3.1 Valid Number Input

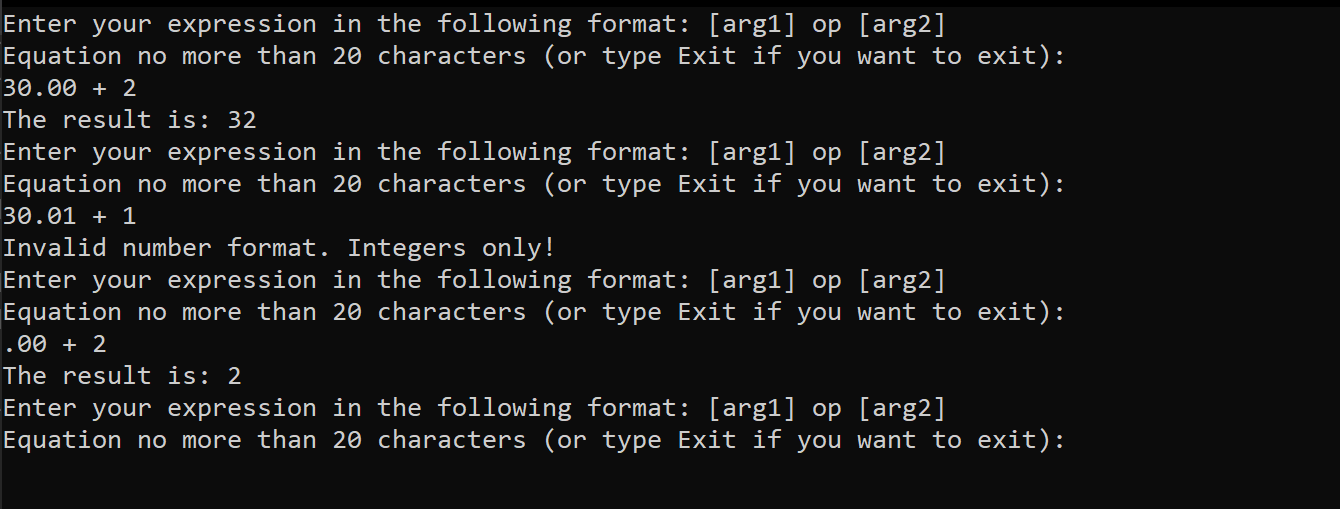
For the purpose of checking for valid number input, we have created a Boolean returned function called *isNumber()* that takes an operand as a string. The function will return true if the operand is determined as a valid number, or false vice versa.

To be evaluated as an integer, all the character in the operand passed in must be a digit, except for the first character could be ‘+’ or ‘-’. In particular, we looped over each character in the operand, if the first character is ‘+’ or ‘-’ we will skip over to the next one. For every character that was iterated, we used the function *isdigit(),* which takes a character as its argument, to check for digit validation. If any character is not a digit, *isNumber()* will immediately return false.

This function later on will be called to check the first operand and the second one. If one of which is not an integer, there will be a message to the console. Some instances of invalid number can be seen as follow:



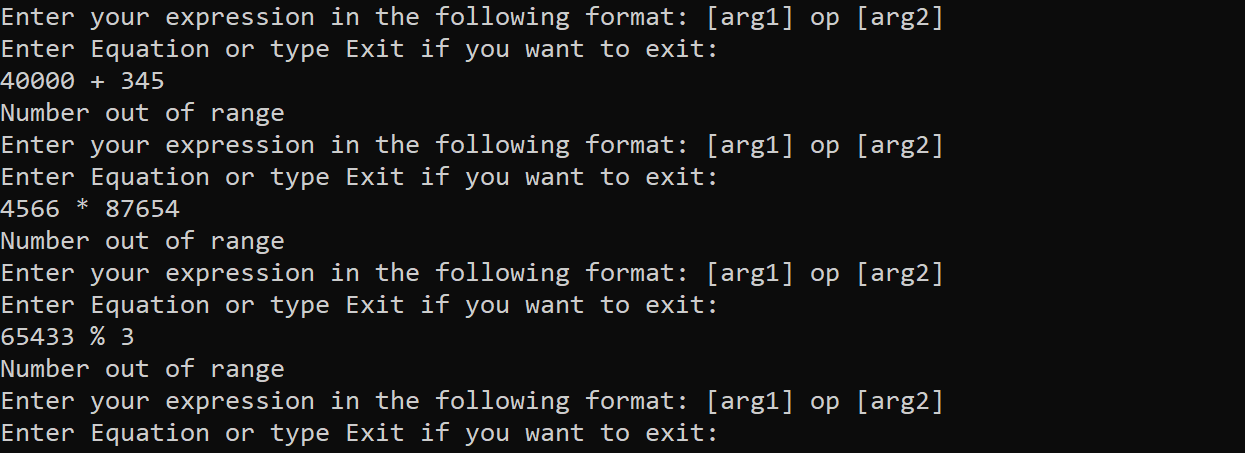
However, there are some special cases in which some numbers such as 30.00 and .00 are valued as valid number input. To cover these cases in our error check, we have to carefully check when it comes to a floating point occurring in the number: the loop will continue iterating to the next characters from which any digit but zero will eventually make the whole number an invalid case. This is due to the fact that 30.00 is considered as valid whereas 30.01 is not. An example of these cases can be seen as follow:



# 3.3.2 Valid Number Range

From the program’s specification, integers must lie in the range of [-32,768 to 32,767]. Therefore, having been converted to number using the function *int convert\_to\_Integer (string str)* as referred to 3.3.1, operands will be passed respectively as arguments into a function called *bool isInRange (int num)* to check whether being in the given range. *Bool isInRange (int num )* returns false if numbers are out of range.

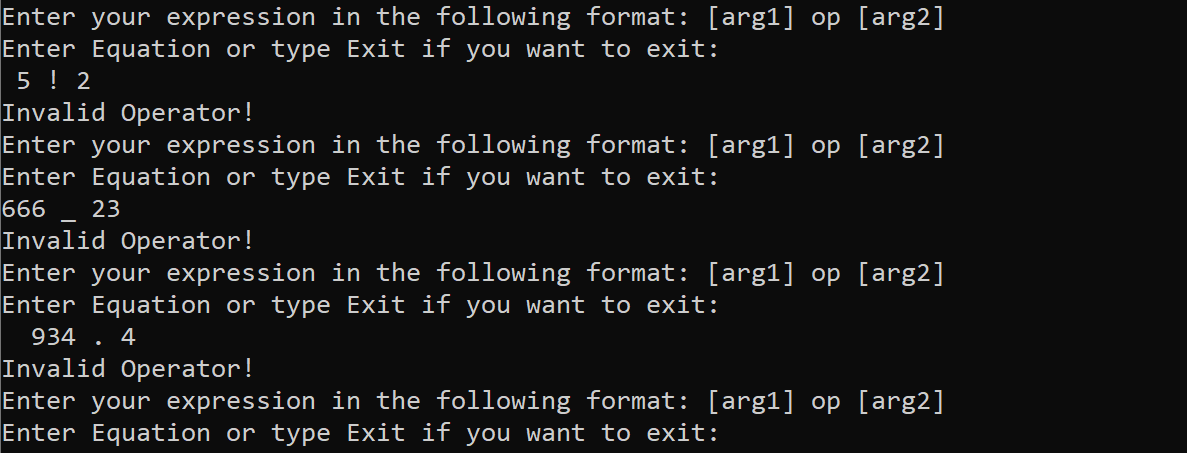
In our main program, this function will be used as a condition for an if-statement that will print out the message to console if numbers are not in range:



# 3.3.3 Valid Operator Input

To validate the operator, we create a function named *bool isValidOp (string str)*, in which a local array of valid operator is initialized ,including “+”, “–“, “\*”, ‘‘/’’, ‘‘%’’, and then, looped over each of its elements to compare to the operator that is passed into the function as an argument. The function will immediately return true if the operator is equal to one of the elements in the given array of valid operators, return false when the loop stops and no matched operator is found.

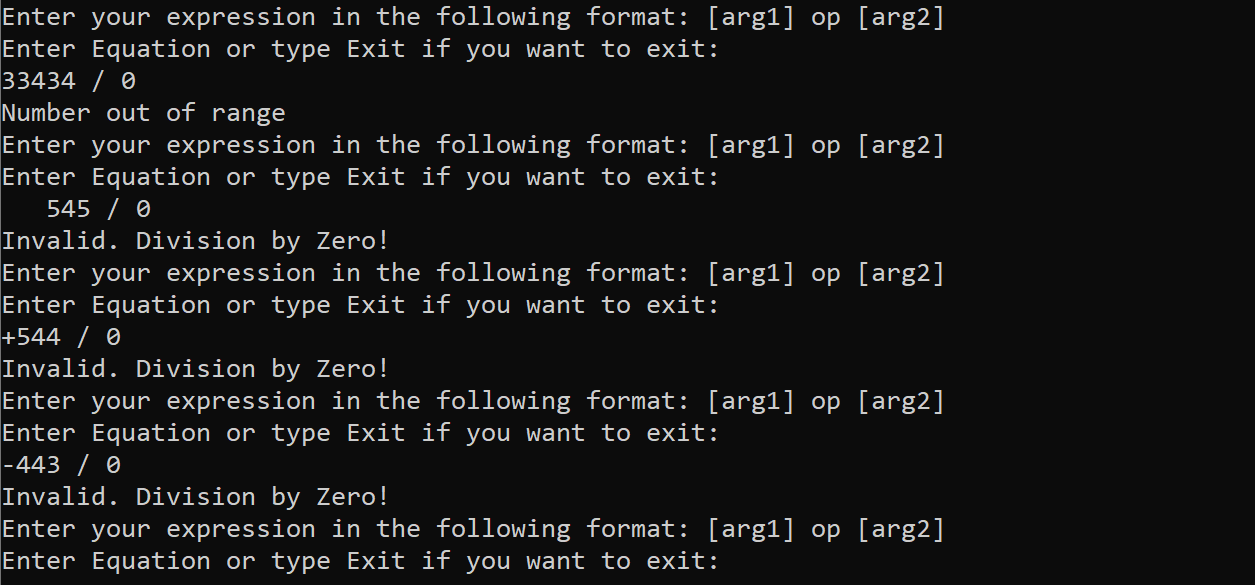
In the main program, once this function returns false in an if-statement there will be a message to the console:



# 3.3.4 Division By 0 Check

Once valid integers and operators and dummy variables (referred to 3.3.5) were all checked, division by zero will be checked next whether the operator and the second operand are “/” and “0” respectively or not. A Boolean returned function named *bool divisionByZero (string op, int num2)* will validate this condition, which returns the boolean value of the expression *(op == “/”) && (num2 == 0);*

An error message will be displayed to the console if this function returns true. For example, when user inputs “5 / 0”, the console will print out:

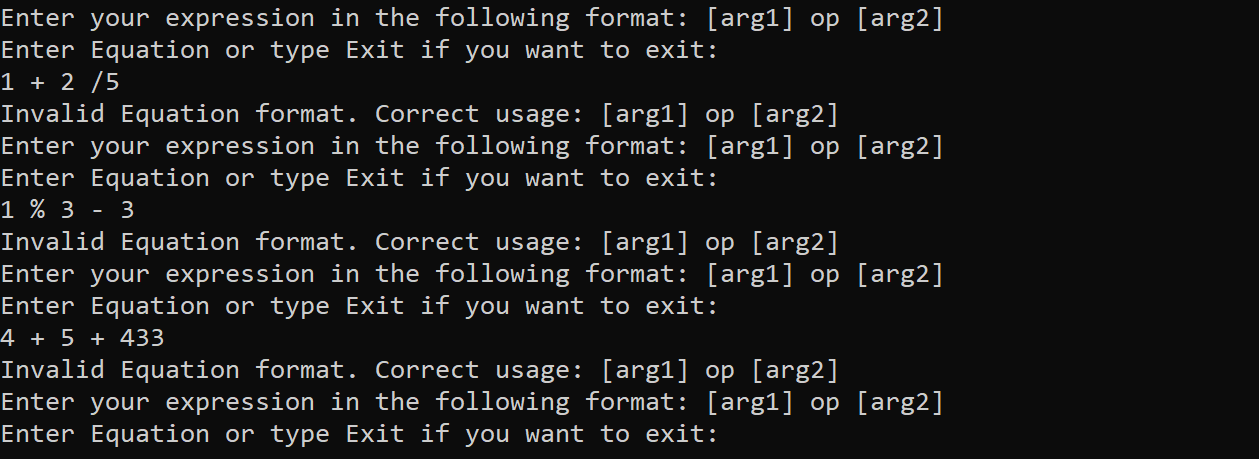


# 3.3.5 Dummy Variables Check

Dummy variable error occurs when there are extra operands or operator in the mathematic expression such as “1 + 2 - 4”. Therefore, in order to handle this type of error, we would count the number of spaces in the input line that has been already removed from all the beginning and ending spaces. The total number of spaces counted in the input line must only two, as the standard input line format is **[arg1] op [arg2]**.

For usage, once all the operands and operator are checked and input line is free from beginning and ending spaces (referred to 3.1.2), *int cout\_space (string str)* (referred in details to 3.1.3) that returns number of spaces will be used as condition to validate this error check.

Take “1 + 2 / 5” as an error test case. A message will be displayed as follow:



# 3.4 Source Code Organized.

Once all the essential functions and error checks are fully and well covered, we begin to re-organize and allocate them into a void function called *void check\_input\_and\_execute\_equation ()* that will be in charge of the main execution of our program. Inside this function, a while loop will be used for the purpose of keeping the program working and not shutting down under any circumstance unless user decides to enter the word “Exit. Inside this loop, an if-else flow control would be applied so as to cover all the error cases (referred to 3.3) and then, eventually do the computation and print out the results. Thus, after displaying the user interface (referred to 3.2.1) and asking for input (referred to 3.1.1), the input line must be checked for the exit command first, and then cut out the beginning and ending spaces (referred to 3.1.2). Then, we will handle the dummy variables (referred to 3.3.5) to ensure the right format of input line. After that, the function *string\* splitStringbyWhiteSpace (string str)* (referred to) will split the input line into three elements that will be validated next in the following order: first check for valid number input (referred to 3.3.1), number range check (referred to 3.3.2), valid operator check (referred to 3.3.3), and finally division by zero (referred to 3.3.4). The order of these error checking cases is supposed to manage the errors efficiently and for better code readability which hence make it easier to debug afterwards. Whenever an element of the input line is caught by these error checking functions, an appropriate message will be displayed to the console (referred to 3.3) and the program will start over to ask for another input line. For further details, our source code will be referred to in the Appendices.

# Conclusion

To conclude, this laboratory activities provide the basic concept of software engineering design and fundamental of C++ programming language. In this laboratory, we are required to design a software to calculate the result of mathematic input expression. The most challenge task of this software is to verify the user input expression. The program takes the input as a string and must convert it to three part of the calculation. We also must handle the white spaces at the beginning and at the end of the input string in case that the user accidentally enters some white spaces. Then we must verify the input by checking for 5 errors including the operands error, operator error, and the format error. Besides the lectures and the laboratory, our team must perform the critically thinking as a software engineering as well as the most efficient algorithms for our software design. Therefore, the program is very efficient and readable from the top down design. As the result, our software performed correctly in calculation as well as all cases of error checking.

# 5. References

[1] StackOverFlow, “Remove spaces from std::string in C++”, Accessed on: Nov 8, 2019. [Online]. Available at: <https://stackoverflow.com/questions/83439/remove-spaces-from-stdstring-in-c>

[2] Fluent C++, “How to split a string in C++”, April 21, 2017, Accessed on: Nov 8, 2019. [Online]. Available at: <https://www.fluentcpp.com/2017/04/21/how-to-split-a-string-in-c/>

[3] StackOverFlow, “Convert an int to ASCII character”, Accessed on: Nov 8, 2019. [Online]. Available at: <https://stackoverflow.com/questions/4629050/convert-an-int-to-ascii-character>

# Appendices.

#include <iostream>

const int MAXIMUM\_PARTS\_DIVIDE\_BY\_SPACE = 3;

const int MAXIMUM\_CHAR\_INPUT = 50;

const int WHITE\_SPACE = 32;

const int MINIMUM\_RANGE = -32768;

const int MAXIMUM\_RANGE = 32768;

const char PLUS\_SYMBOL = '+';

const char MINUS\_SYMBOL = '-';

const char MULTIPLY\_SYMBOL = '\*';

const char DIVIDE\_SYMBOL = '/';

const char MOD\_SYMBOL = '%';

const char FLOATING\_POINT\_SYMBOL = '.';

const char ASCII\_NUMBER\_0 = 48;

const char ASCII\_NUMBER\_9 = 57;

const int GET\_INTEGER\_VALUE\_FROM\_CHARACTER = 48;

std::string user\_input();

void ending\_program\_output();

bool isNumber(std::string s);

bool isValidOp(std::string s);

bool divisionByZero(std::string op, int num2);

int convert\_to\_Integer(std::string str);

bool isInRange(int num);

//bool isDummyVar(std::string str);

int performingCalculation(int num1, int num2, char op);

std::string removeEndingSpace(std::string str);

std::string remove\_beginning\_white\_space(std::string str);

std::string\* spliteStringbyWhiteSpace(std::string str);

int count\_space(std::string str);

void check\_input\_and\_execute\_equation();

bool check\_exit\_command(std::string str);

int main()

{

check\_input\_and\_execute\_equation();

ending\_program\_output();

return 0;

}

void check\_input\_and\_execute\_equation() {

while (true) {

std::string input\_string = user\_input();

//char answer[MAXIMUM\_CHAR\_INPUT];

//std::cout << "Enter your expression in the following format: [arg1] op [arg2]" << std::endl;

//std::cout << "Equation (or type Exit if you want to exit): " << std::endl;

//std::cin.get(answer, MAXIMUM\_CHAR\_INPUT);

if (check\_exit\_command(input\_string))

break;

else {

std::string str\_no\_beginning\_space = remove\_beginning\_white\_space(input\_string);

std::string str\_no\_end\_begin\_space = removeEndingSpace(str\_no\_beginning\_space);

if (count\_space(str\_no\_end\_begin\_space) != 2) {

std::cout << "Invalid Equation format. Correct usage: [arg1] op [arg2]" << std::endl;

std::cin.get();

continue;

}

std::string\* listOfOperands = spliteStringbyWhiteSpace(str\_no\_end\_begin\_space);

if ((!isNumber(\*listOfOperands)) || (!isNumber(\*(listOfOperands + 2)))) {

std::cout << "Invalid number format. Integers only!" << std::endl;

std::cin.get();

continue;

}

else if ((!isInRange(convert\_to\_Integer(\*listOfOperands))) || (!isInRange(convert\_to\_Integer(\*(listOfOperands + 2))))) {

std::cout << "Number out of range" << std::endl;

std::cin.get();

continue;

}

else if (!isValidOp(\*(listOfOperands + 1))) {

std::cout << "Invalid Operator!" << std::endl;

std::cin.get();

continue;

}

else if (divisionByZero((\*(listOfOperands + 1)), convert\_to\_Integer(\*(listOfOperands + 2)))) {

std::cout << "Invalid. Division by Zero!" << std::endl;

std::cin.get();

continue;

}

//else if (isDummyVar(str\_no\_end\_begin\_space)) {

// std::cout << "Wrong format of input. Correct usage: [arg1] op [arg2]" << std::endl;

// std::cin.get();

// continue;

//}

else {

std::cout << "The result is: " << performingCalculation(convert\_to\_Integer(\*listOfOperands),

convert\_to\_Integer(\*(listOfOperands + 2)), (\*(listOfOperands + 1))[0])

<< std::endl;

std::cin.get();

}

}

}

}

std::string user\_input() {

char user\_string[MAXIMUM\_CHAR\_INPUT];

std::cout << "Enter your expression in the following format: [arg1] op [arg2]" << std::endl;

std::cout << "Enter Equation or type Exit if you want to exit: " << std::endl;

std::cin.get(user\_string, MAXIMUM\_CHAR\_INPUT);

return user\_string;

}

void ending\_program\_output() {

std::cout << "LABORATORY GROUP " << std::endl;

std::cout << "s3754450, s3754450@rmit.edu.vn, Quan, Hoang" << std::endl;

std::cout << "s3754450, s3754450@rmit.edu.vn, Quan, Hoang" << std::endl;

std::cout << "s3757823, s3757823@rmit.edu.vn, Son, Le" << std::endl;

}

bool isNumber(std::string number) {

bool has\_decimal\_number = false;

int i = 0;

for (i; i < number.length(); i++) {

if (!has\_decimal\_number) {

if ((number[0] == PLUS\_SYMBOL) || (number[0] == MINUS\_SYMBOL)) {

continue;

}

if ((number[i] < ASCII\_NUMBER\_0 && number[i] != FLOATING\_POINT\_SYMBOL) || number[i] > ASCII\_NUMBER\_9)

return false;

else if (number[i] == FLOATING\_POINT\_SYMBOL) {

has\_decimal\_number = true;

continue;

}

}

else {

if (number[i] != ASCII\_NUMBER\_0) {

return false;

}

}

}

return true;

}

bool isValidOp(std::string str) {

std::string listOp[] = { "+", "-", "\*", "/", "%" };

for (std::string x : listOp) {

if (x == str) {

return true;

}

}

return false;

}

int convert\_to\_Integer(std::string str) {

int sum = 0;

for (int i = 0; i < str.size(); i++) {

if ((str[i] == MINUS\_SYMBOL) || (str[i] == PLUS\_SYMBOL)) {

continue;

}

if (str[i] != FLOATING\_POINT\_SYMBOL)

sum = sum \* 10 + (str[i] - GET\_INTEGER\_VALUE\_FROM\_CHARACTER);

else

break;

}

if (str[0] == MINUS\_SYMBOL) {

return -sum;

}

else

return sum;

}

bool isInRange(int num) {

return !((num < MINIMUM\_RANGE) || (num > MAXIMUM\_RANGE));

}

bool divisionByZero(std::string op, int num2) {

return ((op == "/") && (num2 == 0));

}

//bool isDummyVar(std::string str) {

// int numberSpace = 0;

// for (auto x : str) {

// if (x == ' ')

// numberSpace++;

// }

//

// if (numberSpace != 2) return true;

// else return false;

//}

std::string removeEndingSpace(std::string str) {

int initial\_string\_size = str.size();

int x = str.size() - 1;

for (x; x > 0; x--) {

if (str[x] != ' ') {

//cout << "Valus of x is :" << x << endl;

str.erase(x + 1, initial\_string\_size);

break;

}

}

return str;

}

std::string remove\_beginning\_white\_space(std::string str) {

for (int i = 0; i < sizeof(str); i++) {

if (str[i] != WHITE\_SPACE) {

str.erase(0, i);

break;

}

}

return str;

}

std::string\* spliteStringbyWhiteSpace(std::string str) {

std::string\* myArr = new std::string[MAXIMUM\_PARTS\_DIVIDE\_BY\_SPACE];

int index\_divide\_by\_space = 0;

for (char x : str) {

if (x == WHITE\_SPACE) {

index\_divide\_by\_space++;

}

else

myArr[index\_divide\_by\_space] += x;

}

return myArr;

}

int count\_space(std::string str) {

int count\_space = 0;

for (int i = 0; i < str.length(); i++) {

if (str[i] == WHITE\_SPACE) {

count\_space++;

}

}

return count\_space;

}

bool check\_exit\_command(std::string str) {

std::string user\_exit = "Exit";

int index = 0;

//std::cout << str[0] << std::endl;

for (index; index <= 3; index++) {

if (str[index] != user\_exit[index]) {

return false;

break;

}

}

if (str[4] == NULL)

return true;

}

int performingCalculation(int num1, int num2, char op) {

switch (op) {

case PLUS\_SYMBOL: return num1 + num2;

case MINUS\_SYMBOL: return num1 - num2;

case MULTIPLY\_SYMBOL: return num1 \* num2;

case DIVIDE\_SYMBOL: return num1 / num2;

case MOD\_SYMBOL: return num1 % num2;

default:

return 0;

}

}