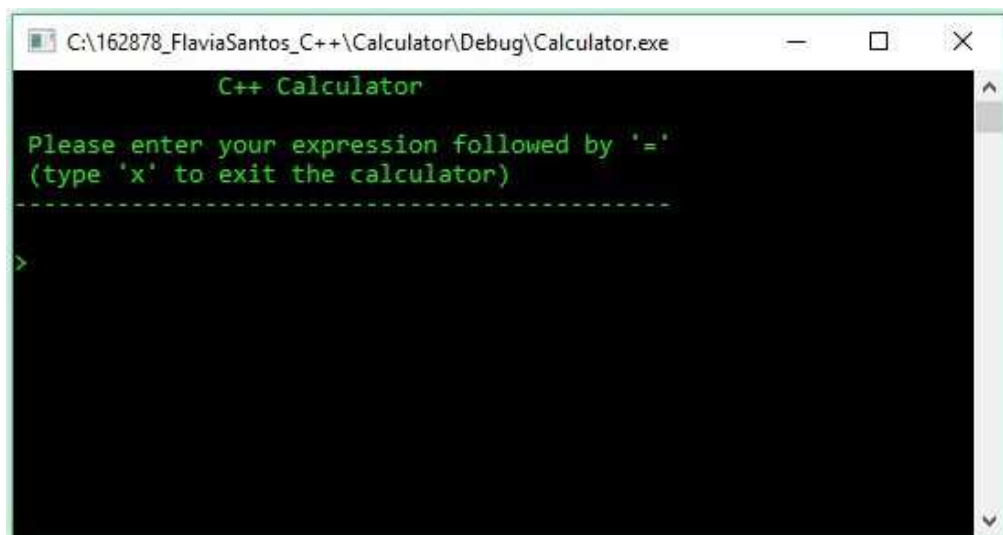


## Simple Calculator in C++



The image shows a screenshot of a Windows application window titled "C++ Calculator". The window's title bar includes the file path "C:\162878\_FlaviaSantos\_C++\Calculator\Debug\Calculator.exe" and standard minimize, maximize, and close buttons. The main content area has a black background with green text. It displays the title "C++ Calculator" at the top, followed by the instruction "Please enter your expression followed by '=' (type 'x' to exit the calculator)". A dashed green line separates the instruction from the input area. Below the line, a green prompt character ">" is visible on the left side of the black area.

```
C++ Calculator

Please enter your expression followed by '='
(type 'x' to exit the calculator)
-----
>
```

---

# Table of Contents

1	Introduction.....	4
2	Problem Analysis.....	5
	2.1 Problem Statement.....	5
	2.2 Solution Design.....	5
3	Implementation and Results.....	7
	3.1 Code Structure.....	7
	3.1.1 Error Handling and Testing .....	8
	3.2 Results.....	10
4	Conclusion .....	12

# 1 Introduction

This document presents a simple calculator in C++ code that reads a string containing multiple numbers and operators given as input from the keyboard, and prints out the result. The calculator ensures that the complete expression is understood correctly, because it evaluates which operator has the strongest binding before giving the final result on screen. In addition, the program handles a wide range of errors.

This report consists in two parts:

- Problem Analysis covers the examination of the problem and the designing of the solution.
- Implementation and Results: gives an overview of the code and results obtained.

---

## 2 Problem Analysis

This chapter describes the first stages of development of the C++ Calculator: analysis of the problem and solution design.

### 2.1 Problem Statement

Create a program that reads a list of strings containing multiple numbers and operators and prints out the result. It should be possible to use '(') to control how the calculation is performed and the program should be able to handle an infinite number of parameters.

The calculator should handle a wide range of error conditions. E.g. that an illegal operator (% , ^), letter, division by zero, or no number is entered.

### 2.2 Solution Design

The goal is to implement a standard calculator with simple functionality. In order to evaluate correctly an expression from left to right, the program should evaluate the order of execution of the operators. The precedence rules are:

1. Expressions inside the parentheses
2. Multiplication operator '\*' and division operator '/'
3. Additive operator '+' and subtraction operator '-'

To observe which operator has the strongest binding, the calculator should read the numbers and operators from input and store them to be used according to precedence rules.

The conventional way to implement this is by assembling tokens from expressions the user type in. Token is a sequence of characters that represents a number (type double) or char units, such as the operands ( + , - , / , \* ) and parenthesis.

The approach to compute ordinary arithmetic used in this calculator is based in one example from the book "Programming Principles and Practice Using C++", by the creator of C++, Bjarne Stroustrup [1]. The rules are described below (Figure 2-2) and are implemented as functions in the code.

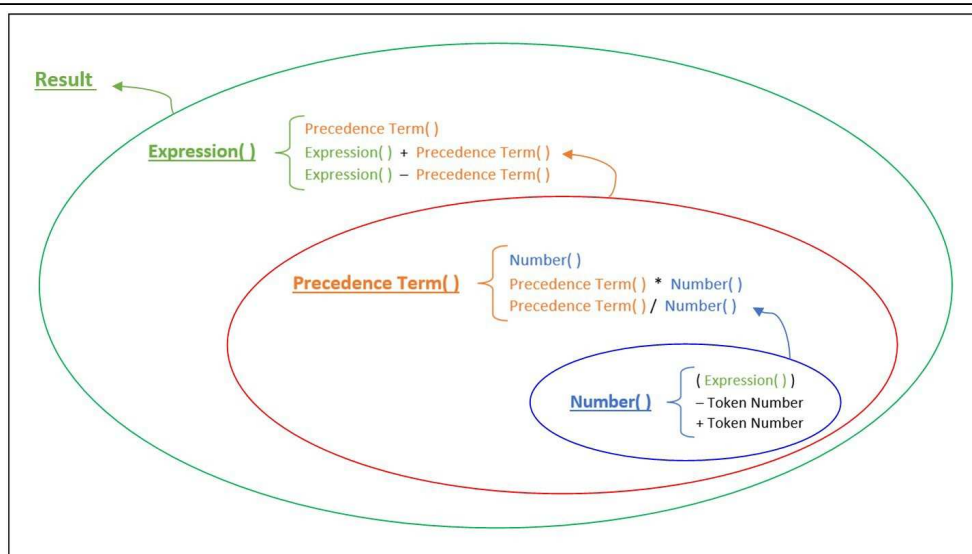


Figure 2-2: Precedence Rules

In this solution, given some input, the program reads a stream of strings and produces the tokens. Then, calling top rule first, it searches for Expression, which consider the result of a Precedence Term function. An Expression can also be followed by the token '+' or '-' and another Precedence Term result. Therefore, the program calls the Precedence Term, see if it is followed by a plus or a minus sign and keep trying to find for tokens that match its conditions until there is no more '+' or '-' tokens.

Any Precedence Term starts with a result from a Number function and such result can be followed by a token '\*' or a token '/'. Thus, looking for a Number result, it analyses if it is followed by a '\*' or a '/', and keep doing that until there are no more multiplication or division signs in the stream.

In the same logic, a Number is a evaluation of unary '-' or '+' and a token number, as well as the result of numbers and operators inside the parenthesis.

As soon as the program finds a '(' it calls the top rule and evaluates the expression until the closing parenthesis appear, what guarantees the precedence rules.

---

## 3 Implementation and Results

A brief description of the code implementation for the Simple calculator is presented in this chapter.

### 3.1 Code Structure

The overview of the calculator code, without any implementation details is:

□ Token.h:

Header file deals with the declarations needed to use Token and Parser classes.

```
#include<iostream>
#include<string>
#include <sstream>

using namespace std;

class Token {...};

class Parser {...};

const char t_number = '1';
```

□ Token.cpp:

Source file contains the definitions of the Parser class.

```
#include "Token.h"

Token Parser::get(){...};

void Parser::copy_buffer(Token t) {...};

void Parser::ignore(char c) {...};
```

□ main.cpp:

The main file implements the uses of the functions defined previously and error handling.

```
#include "Token.h"

double expression(Parser&);           // declaration so that number() can use it
```

---

```

double number(Parser& t_stream) {...}           // deal with numbers and parentheses
double precedence_term(Parser& t_stream) {...} // deal with * and /
double expression(Parser& t_stream) {...}       // deal with + and -

int main(){                                     // main loop and deal with errors try {...}
    catch () {...}
}

```

### 3.1.1 Error Handling and Testing

One of the requirements stated for the Simple Calculator is that it should handle a wide range of error conditions.

When a function is called and receives legal inputs, it returns normally. However, when it detects an error, the function throws an exception with an error message. The error detection is shown in the code excerpts of its respective function:

```

Token Parser::get()
{
    // reads input stream
    // switch cases to return tokens

    default:
        if (isdigit(ch)||ch=='.') {
            input.putback(ch); double
            val;
            input >> val;
            return Token{ t_number,val };
        }
        throw ("illegal input");
}

```

```

void Parser::copy_buffer(Token t)
{
    if (full) throw ("copy into a full buffer");
    buffer = t;
    full = true;
}

```

```

double number(Parser& t_stream)
{
    //creates a Token object
    //switch cases

    case '(':
    {
        double n = expression(t_stream);
        t = t_stream.get();
        if (t.kind != ')') throw ("')' expected");
        return n;
        break;
    }
    default:
        throw ("number expected");
}
}

```

---

```
double precedence_term(Parser& t_stream)
{
    //call number()
    //call get()
    //switch cases

    case '/':
    {
        double n = number(t_stream);
        t = t_stream.get();
        if (n == 0) throw ("divided by zero");
        left_operand /= n;
        break;
    }
}
```

The handling of that exception is implemented in the main function, which catches that exception and deal with it keeping the window open, so the user can see what went wrong.

```
int main()
{
    //creates a Parser object
    //prompt the user

    try {
        //get()
        //copy_buffer()
        //expression()
    }

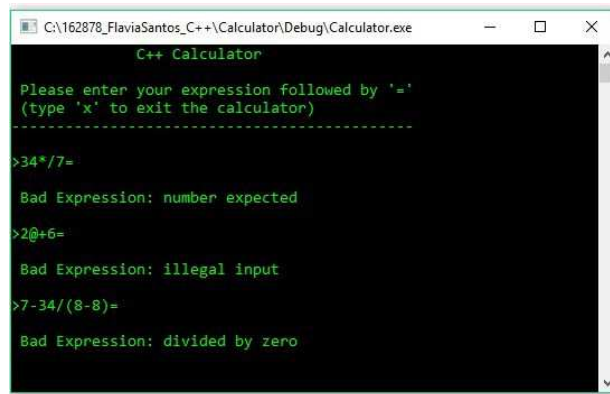
    catch (const char *e) {
        cout << "\n Bad Expression: " << e << "\n" << endl;
        cin.clear();
        cin.ignore();
        cin.get();
    }
}
```

The program should give error messages for illegal inputs, such as:

- ☐ illegal operator (%, ^)
- ☐ division by zero
- ☐ no number is entered
- ☐ other characters are entered (e.g. letters, «, @)

Some outputs from the testing are reproduced in Figure 3-1:





```
C:\162878_FlaviaSantos_C++\Calculator\Debug\Calculator.exe
C++ Calculator

Please enter your expression followed by '='
(type 'x' to exit the calculator)
-----
>34*/7=
Bad Expression: number expected

>2@+6=
Bad Expression: illegal input

>7-34/(8-8)=
Bad Expression: divided by zero
```

Figure3-1: Testing Illegal Inputs

## 3.2 Results

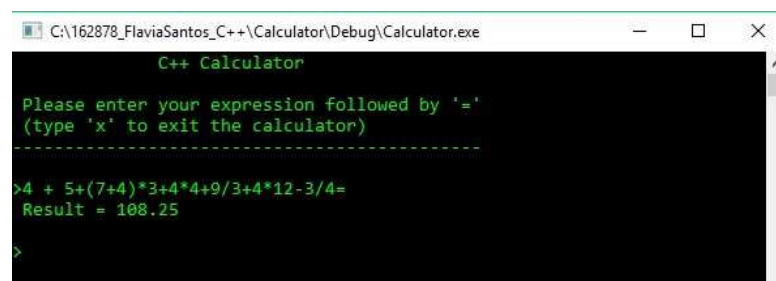
The user interaction with the calculator is by using the computer keyboard and its screen, it is a very simple solution implemented with cin and cout from C++'s standard library.

When the program starts running the user is asked to enter an expression. The equality sign is required to indicate when the expression to be calculated stops. After that, the program created performs the calculation and prints out the result on the console window.

Consider the following expression as an example:

$$4 + 5 + (7 + 4) * 3 + 4 * 4 + 9 / 3 + 4 * 12 - 3 / 4 =$$

(1) The result should be: 108.25, as displayed in Figure 3-2:



```
C:\162878_FlaviaSantos_C++\Calculator\Debug\Calculator.exe
C++ Calculator

Please enter your expression followed by '='
(type 'x' to exit the calculator)
-----
>4 + 5+(7+4)*3+4*4+9/3+4*12-3/4=
Result = 108.25
>
```

Figure3-2:  
Results

---

This version of the calculator is working fine, it is evaluating parenthesis before any other operator and it considers multiplication and division before addition and subtraction operations.

When the user wishes to exit the program the letter 'x' should be entered

## 4 Conclusion

The Simple C++ Calculator works perfectly solving expressions typed by the user and prints out the desired result on the screen when legal inputs are typed in. It is a very useful program that gives error messages to the user when it detects illegal inputs.

# Reference List

[1] B. Stroustrup, Programming Principles and Practice Using C++. Addison-Wesley Professional, 2014.