

Calculo de Integrais

Joao Felipe Bianchi Curcio

Jonas Edward Tashiro

Luan Lopes Barbosa de Almeida

Rafael Melloni Chacon Arnone

Contents

1	Organização de Pastas	3
2	Conteudo dos Arquivos	3
2.1	Pasta src	3
2.2	Pasta exp	8
2.3	Pasta integral	11

1 Organização de Pastas

```
src
├── exp
│   ├── definitions.h
│   ├── expression.h
│   ├── lexicon.h
│   ├── token.h
│   ├── lexicon.cpp
│   ├── expression.cpp
│   └── token.cpp
├── integral
│   ├── integral.h
│   └── integral.cpp
├── Makefile
└── main.cpp
```

2 Conteudo dos Arquivos

2.1 Pasta src

Listing 1: main.cpp

```
1 #include "exp/expression.h"
2 #include "exp/token.h"
3 #include "integral/integral.h"
4 #include <ios>
5 #include <ostream>
6 #include <string>
7 #include <iostream>
8 #include <utility>
9 #include <vector>
10 #include <limits>
11 #include <iomanip>
12
13 int main() {
14     bool quit = false;
```

```

15     bool cannot_be_conv = false;
16     std::string user_expr;
17     std::string user_info1, user_info2;
18     int numTrapz, precision;
19     double start, end;
20     std::vector<std::pair<double, double>> fnTable;
21
22     Expression *expression;
23     do
24     {
25         std::cout << "\nEscreva uma expressao:\n";
26         std::cout << "f(x) = ";
27         std::getline(std::cin, user_expr);
28         user_expr += ',';
29         std::endl(std::cout);
30
31         if(!addSpaces(user_expr))
32         {
33             expression = new Expression(user_expr);
34             if(expression->isValid())
35             {
36                 expression->infixToPostfix();
37                 do{
38                     std::cout << "\nEscreva o ponto extremo da esquerda:\n";
39                     std::cin >> user_info1;
40                     std::cout << "Escreva o ponto extremo da direita:\n";
41                     std::cin >> user_info2;
42                     try{
43                         start = std::stold(user_info1.data());
44                         end = std::stold(user_info2.data());
45                         cannot_be_conv = false;
46                         user_info1.clear(), user_info2.clear();
47                     }catch (std::invalid_argument){
48                         std::cout << "Uma variavel foi escrita de
49                                     forma indevida";
50                         cannot_be_conv = true;
51                     }
52                 }while (cannot_be_conv);
53
54                 do {

```

```

55         std::cout << "\nEscreva o numero de casas decimais\n";
56         std::cin >> user_info1;
57         std::cout << "Escreva o numero de trapezios:\n";
58         std::cin >> user_info2;
59         try {
60             precision = std::stoi(user_info1.data());
61             numTrapz = std::stoi(user_info2.data());
62             cannot_be_conv = false;
63             user_info1.clear(), user_info2.clear();
64         } catch (std::invalid_argument) {
65             std::cout << "Uma variavel foi escrita de
66                 forma indevida";
67             cannot_be_conv = true;
68         }
69     } while (cannot_be_conv);
70
71     std::cout << std::fixed;
72     std::cout << std::setprecision(precision + 1);
73
74     TrapezoidIntegral integralCalc(start,end,numTrapz,precision);
75     integralCalc.calculateIntegral(*expression, fnTable);
76
77     std::cout << "\n\nSaida:\n\n";
78     std::cout << "Erro de arredondamento = "
79         << integralCalc.errorRounding;
80
81     std::cout << std::fixed;
82     std::cout << std::setprecision(precision);
83
84     std::cout << "\nValor da Integral = "
85         << integralCalc.sumTraps;
86     std::cout << "\nTabela de Valores:\n";
87     std::cout << 'x';
88
89     for (int i = -1; i <= precision; i++)
90         std::cout << ' ';
91
92     std::cout << " | " << "f(x)" << '\n';
93     for (auto f : fnTable)
94         std::cout << f.first << " | " << f.second << '\n';

```

```

95         delete expression;
96     }
97     else
98         std::cout << "\nA expressao fornecida contem erro\n";
99     }
100    else
101        std::cout << "\nErro Lexico detectado na expressao fornecida\n";
102
103    std::cout << "Deseja sair do programa?\n";
104    std::cout << "Sim (Escreva q)\t Nao (Escreva qualquer coisa)\n";
105    std::cin >> user_info1;
106    if(user_info1 == "q")
107        quit = true;
108
109    std::cin.ignore(std::numeric_limits<std::streamsize>::max(), '\n');
110    user_expr.clear();
111    user_info1.clear();
112    std::cout << "\033[2J\033[1;1H";
113
114    }while (!quit);
115    return 0;
116 }

```

Este proximo arquivo tem como intuito facilitar o processo de compilação.

Listing 2: Makefile

```
1  EXPRESSION = ./exp
2  INTEGRAL = ./integral
3  SOURCES = main.cpp
4  SOURCES += $(EXPRESSION)/expression.cpp $(EXPRESSION)/lexicon.cpp
5  SOURCES += $(EXPRESSION)/token.cpp
6  SOURCES += $(INTEGRAL)/integral.cpp
7  OBJECTS = $(addsuffix .o, $(basename $(notdir $(SOURCES))))
8
9
10 COMPILE : LINK
11     g++ -o main $(OBJECTS)
12
13 LINK:
14     g++ -c $(SOURCES)
15
16 clean:
17     rm $(OBJECTS)
```

2.2 Pasta exp

Listing 3: definition.h

```
1 #pragma once
2 #include <utility>
3 #define FAILURE false
4 #define SUCCESS true
5
6 using AttributeValue = int;
7 using Priority = int;
8 using Token_name = int;
9 using Token_type =
10     enum : int
11     {
12         endExpression,
13         leftParen,
14         rightParen,
15         unaryOp,
16         binaryOp,
17         operand,
18         number
19     };
20
21 using Token = std::pair<Token_name,AttributeValue>;
```

Listing 4: expression.h

```
1 #pragma once
2 #include <queue>
3 #include <list>
4 #include <map>
5 #include <string>
6 #include <utility>
7 #include "definitions.h"
8 #include "lexicon.h"
9
```



```

10 using ErrorCode = bool;
11
12 class Expression
13 {
14     std::list<Token> tokenized_expr;
15     Lexicon symbol_table;
16     public:
17         Expression();
18         Expression(std::string &expression); //Tokenize the expression
19         ErrorCode infixToPostfix(); //certify that expression is valid first
20         ErrorCode evaluateAt(double x, double &f_of_x);
21         void tokenizeExpression(std::string &expression);
22         void getIteratorRange(std::list<Token>::iterator &start,
23                               std::list<Token>::iterator &end);
24         ErrorCode isValid(); //check for infix
25     private:
26         void removeFirstToken(); //move back to private
27         void addToken(Token &new_token);
28         float do_unary(double x, Token_name type);
29         float do_binary(double x, double y, Token_name type);
30 };
31
32 /*Authenticate lexical correctness of expression before sending to the class*/
33 /*Also add whitespace to better identify lexemes*/
34 ErrorCode addSpaces(std::string &expression);

```

Listing 5: token.h

```

1 #pragma once
2 #include "definitions.h"
3 #include <string>
4
5 struct Token_data
6 {
7     std::string name;
8     double value;
9     Priority priority;
10     Token_data(std::string token_name, double value, Priority priority);

```

```
11 };
```

Listing 6: lexicon.h

```
1 #include <map>
2 #include <string>
3 #include <vector>
4 #include "definitions.h"
5 #include "token.h"
6 #define NON_EXISTENT -1
7
8 class Lexicon
9 {
10     //shall only be used to setup expression
11     std::map<std::string, AttributeValue> lexeme_map;
12     std::vector<Token_data*> symbol_table;
13     public:
14         Lexicon() = default;
15         void setStandardTokens(); //sets up the map to lexemes
16         Token_data* getTokenInfo(AttributeValue token_id);
17         AttributeValue newToken(Token_data *new_token);
18         AttributeValue findAttribute(std::string &lexeme);
19 };
```

2.3 Pasta integral

Listing 7: integral.h

```
1 #include "../exp/expression.h"
2 #include <utility>
3 #include <vector>
4
5 struct TrapezoidIntegral
6 {
7     int nOfTrapz;
8     int precision;
9     double errorRounding;
10    double sumTraps;
11    double x_start, x_end;
12    void calculateIntegral(Expression &expr,
13        std::vector<std::pair<double, double>> &fnTable);
14    TrapezoidIntegral() = default;
15    TrapezoidIntegral(double start, double end, int num, int precision);
16 };
```

Listing 8: integral.cpp

```
1 #include "integral.h"
2 #include <cmath>
3 #include <math.h>
4 #include <utility>
5 #include <vector>
6
7 void setNumber(int precision, double &value);
8
9 TrapezoidIntegral::TrapezoidIntegral(double start, double end,
10                                     int num, int precision)
11 {
12     x_start = start;
13     x_end = end;
14     nOfTrapz = num;
```

```

15     this->precision = precision;
16 }
17
18 void TrapezoidIntegral::calculateIntegral(Expression &expr,
19     std::vector<std::pair<double, double>> &fnTable)
20 {
21     double increment = std::abs(x_end - x_start);
22     increment /= static_cast<double>(nOfTrapz);
23     double x = x_start;
24     double f_late, f_early;
25     fnTable.clear();
26     sumTraps = 0.0F;
27
28     expr.evaluateAt(x, f_late);
29     setNumber(precision, f_late);
30     fnTable.push_back(std::pair<double, double>(x, f_late));
31     x += increment;
32     expr.evaluateAt(x, f_early);
33     setNumber(precision, f_early);
34
35     for (int i = 1; i <= nOfTrapz; i++)
36     {
37         fnTable.push_back(std::pair<double, double>(x, f_early));
38         x += increment;
39         sumTraps += increment * (f_early + f_late) / 2.0F;
40         f_late = f_early;
41         expr.evaluateAt(x, f_early);
42         setNumber(precision, f_early);
43     }
44
45     errorRounding = nOfTrapz *
46         (5.0F / std::pow(10.0F, precision + 1)) * increment;
47 }
48
49 void setNumber(int precision, double &value)
50 {
51     value *= pow(10.0, precision);
52     value = std::round(value);
53     value /= pow(10.0, precision);
54 }

```