Dário Darienço

Gabriel Valeze

Edmar Alves

Yuri Chiesa

- Documentação da linguagem

O código apresentado possui a linguagem baseada em C. Portanto, utilizamos a criação de um subconjunto de linguagem já existente para nosso interpretador de calculadora, contemplando as funcionalidades pedidas em https://github.com/wellingtondellamura/compilers/blob/master/final-work/description.txt.

O código a aceita variáveis com somente uma letra. Não pode conter espaço entre os elementos da fórmula inserida. Abaixo é mostrado um erro de sintaxe e seu devido tratamento, indicando para quem o digita que possui um erro.

```
>> 5 - 2

Syntax Error: Erro de sintaxe
5 - 2
 ^
5
```

Portanto, o correto para o nosso caso, seria escrever “5-2” e não “5 – 2”.

- Descrição da implementação

```
#include <iostream>
#include <string>
#include "estruturas.h"
```

Os dois includes acima são necessários, tanto para manipulação de fluxo de dados padrão do sistema (entrada padrão, saída padrão e saída de erros padrão) quanto para uma cadeia de caracteres especializada para o tipo de dado nativo char. O include para as estruturas dizem respeito aos tokens e as funções, que preferimos deixar separado por razões de organização.

```
//tokens utilizados
#define NUMERO 100
#define OPERACAO  100
#define NAO_OPERACAO -1
#define SOMA 1
#define SUBTRACAO 2
#define MULTIPLICACAO 3
```

```
#define DIVISAO 4
#define ATRIBUICAO 5
#define INTEIRO 'int'
#define FLOAT 'float'
#define CHAR 'char'
#define SE 'if'
#define ENQUANTO 'while'
```

Acima estão os tokens, utilizados para que possamos posteriormente embutir as regras que se referirão a esses.

Como a ideia era para ser simples, como uma calculadora, os laços de repetições, e condicionais, if e while, foram colocadas, porém exercerão tarefa estranha; mais apenas para seguir as funcionalidades pedidas do que realmente para funcionar como algo útil em nosso caso.

```
using namespace std;

int valor;
string input;

struct token{
  int t; // para o tipo da variável utilizada
  int a; // para a ação da
};
struct FUNCAO { //Receberá os operadores e os seus valores
  char op;
  int valor;
};
```

Declarando o valor e o input, que não ficou colorido por eu estar usando o replit, que não torna colorido variáveis declaradas quando vindas de uma biblioteca.

As duas structs acima são sobre o token e a função, usada para receber o operador e o seu respectivo valor.

```
int valor;
string input;
void error(string);
void print(char);
int calc(int, int, int);
FUNCAO expr();
FUNCAO term();
FUNCAO rest();
void match(char);
char nextToken();
```

```
char lookahead;
```

As variáveis e as funções utilizadas no programa: valor, input, função para tratamento de erro, função para printar na tela, cálculo a ser realizado, expr, term, rest... o match é para ver se batem o valor do lookahead. O nextoken para pegar o valor token atual e compará-lo com o valor do lookahead.

```
//Lexer
int count = 0;
char nextToken(){
  if (count == input.length())
    return EOF;
  return input[count++];
}


//Parser
FUNCAO expr(){
  FUNCAO
 e, t, r;
  t = term();
  r = rest();
  e.valor = calc(t.valor, r.op, r.valor);
  return e;
}
```

O lexer e o parser, analisador léxico e analisador sintático. O lexer é responsável pela leitura da expressão inserida, é ele quem vai verificar se a expressão terminou ou se ela está correta. O analisador sintático irá calcular a expressão e verificar se os tokens estão na tabela de símbolos.

```
FUNCAO rest(){
  FUNCAO
 e, t, r;
  e.op = NAO_OPERACAO;
  if (lookahead == '-'){
     match('-');
     e.op = SUBTRACAO;
     t = term();
     r = rest();

  }
  else if (lookahead == '+'){
    match('+');
    e.op = SOMA;
```

```
    t = term();
    r = rest();
  }
  else if (lookahead == '*'){
    match('*');
    e.op = MULTIPLICACAO;
    t = term();
    r = rest();
  }
  else if (lookahead == '/'){
    match('/');
    e.op = DIVISAO;
    t = term();
    r = rest();
  }
  else if(lookahead == '=') {
    match('=');
    e.op = ATRIBUICAO;
    t = term();
    r = rest();
  }

   else if (lookahead != EOF) {
    error("Erro de sintaxe");
  }
  e.valor = calc(t.valor, r.op, r.valor);
  return e;
}
```

O rest é quem entende algumas funções nas quais desejamos, como por exemplo o if abaixo, que irá ver se o próximo símbolo é um sinal de menos, para que nos diga que o que queremos é uma subtração.

```
  if (lookahead == '-'){
     match('-');
     e.op = SUBTRACAO;
     t = term();
     r = rest();
```

O term nos irá dizer qual número está inserido na operação.

```
FUNCAO term(){
  FUNCAO r;
  r.op = NAO_OPERACAO;
  switch (lookahead) {
    case '0': match('0'); r.valor = 0; break;
    case '1': match('1'); r.valor = 1; break;
```

```
    case '2': match('2'); r.valor = 2; break;
    case '3': match('3'); r.valor = 3; break;
    case '4': match('4'); r.valor = 4; break;
    case '5': match('5'); r.valor = 5; break;
    case '6': match('6'); r.valor = 6; break;
    case '7': match('7'); r.valor = 7; break;
    case '8': match('8'); r.valor = 8; break;
    case '9': match('9'); r.valor = 9; break;
    case 'x': match('x'); r.valor = valor; break;
    default: error("Erro na escrita .");
  }
  return r;
}
```

As regras utilizadas na calculadora

```
//regras matematicas
int calc(int v1, int op, int v2){
  switch (op){
    case NAO_OPERACAO : return v1;
    case SOMA : return v1+v2;
    case SUBTRACAO : return v1-v2;
    case MULTIPLICACAO : return v1*v2;
    case DIVISAO : return v1/v2;
    case ATRIBUICAO : valor = v2; return v2;
  }
  return 0;
}
```

Um void match para vermos se está batendo.

```
void match(char c){

  if (lookahead == c) {
    lookahead = nextToken();
  } else {
    string s = "Encontrado " + to_string(lookahead) + " mas esperado " +
to_string(c);
    error(s);
  }
}
```

Um tratamento de erro para casos sintáticos, um print para nosso resultado e main para executar tudo.

```
//Tratamento de Erro
```

```
void error(string msg){
  cout << endl << "Erro de sintaxe: "<< msg << endl;
  cout << input << endl;
  abort();
}

//
void print(char c){
  cout << c;
}

// --- MAIN
int main(){
  cout << ">> ";
  getline(cin, input);
  while (!input.empty()){
    lookahead = nextToken();
    FUNCAO e = expr();
    cout << e.valor << endl << ">> ";
    getline(cin, input);
    count = 0;
  }
}
```