Calculadora Simples

Lucas do Vale Santos

16 de Setembro de 2013

1. Introdução

Este trabalho consiste na elaboração de um programa que simule uma calculadora simples e que execute algumas operações básicas. Espera-se que a mesma seja acessível a qualquer usuário leigo, que por meio da manipulação fácil e flexível, seja capaz de executar determinadas tarefas para qual foi projetada.

2. Implementação do Programa

O programa desenvolvido foi escrito utilizando a linguagem de programação **Pascal** e sua biblioteca **Crt**, que é de fundamental importância para o funcionamento dos comandos **writeln** e **readln** (entrada e saída de dados), além do **clrsrc** que serve para apagar os dados impressos na tela que antecedam este mesmo comando.

Dentre os vários recursos que a linguagem proporciona, além dos citados anteriormente, tem-se o uso de estruturas de iteração indefinida (**do** ... **while**), estruturas condicionais (**if ... then ... else**, **case ... of**), funções/procedimentos(**functions/procedure**). Algumas funções específicas também foram aplicadas, como a **sqrt** (cálculo da raiz quadrada) e a power(cálculo da potenciação).

Um dos problemas não solucionados consiste na impossibilidade de comparar entre si, variáveis do tipo **char**, utilizando-se o operador lógico **<>** combinado com outros do mesmo gênero, dentro de uma mesma condição. A condição simplesmente não é respeitada. Por exemplo:

If ((cmd **<>** ‘a’) **OR** (cmd **<>** ’s’)) **then**

//esta expressão não funciona, e para tanto, não possível utilizar algo semelhante

Além desta, não foi encontrado nenhum outro problema. Para contornar o anterior, utilizou-se de manipulações lógicas. Algumas restrições também foram implementadas, como será mostrado a seguir.

3. Casos de Uso

Aqui serão apresentados os casos de usos referentes a entrada, saída e o tratamentos de alguns dados.

O programa disponibiliza na tela de execução, um **menu** fixo, contendo uma breve explicação de como utilizar a calculadora e também, uma lista com os comandos indispensáveis, como pode ser visto na Figura 1.

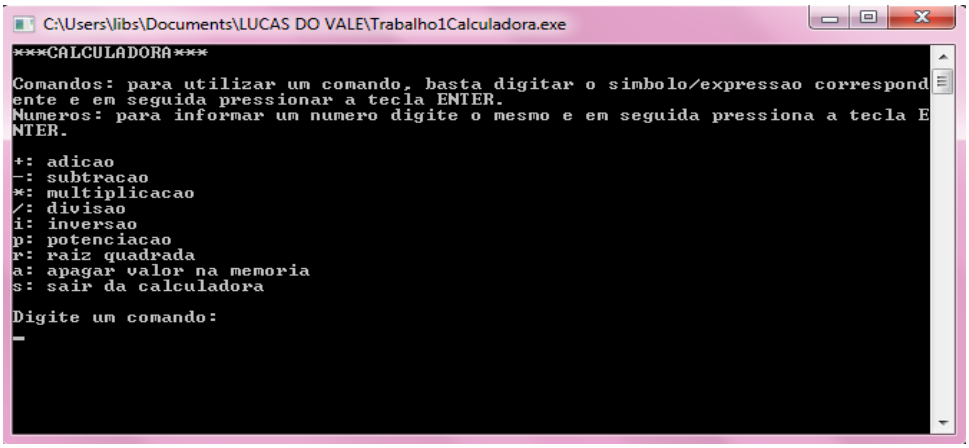

Figura 1: menu da calculadora

A princípio, toda vez que a memória está limpa, é exigido que se entre com uma operação dentre as listadas no menu. Ao digitar o comando 'a', a memória é limpa, e calculadora volta ao seu estado inicial, como está representado nas Figuras 2 e 3. Ao digitar o comando 's', a calculadora é fechada, e apresenta-se uma mensagem de uma mensagem de finalização, como mostram as Figuras 3 e 4.

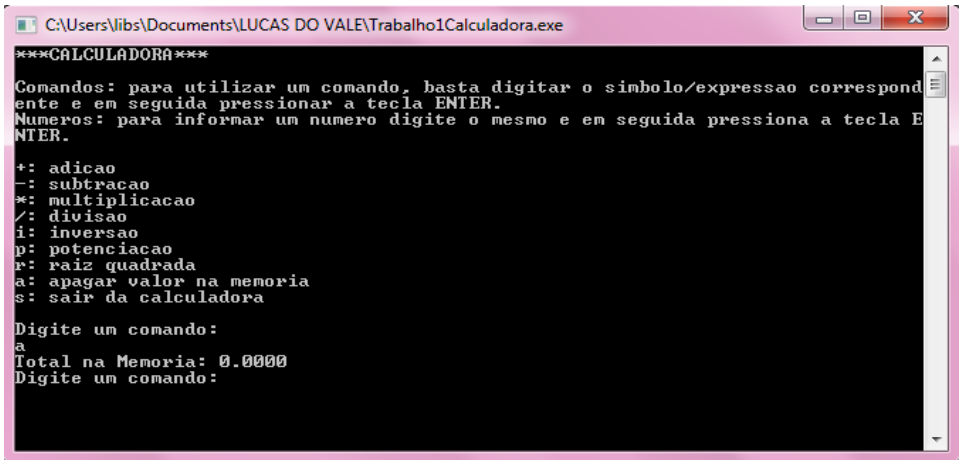

Figura 2: estado da calculadora ao apagar a memória.

O laço de repetição mais interno é o responsável pela contínua entrada de dados até que se deseja limpar a memória. O laço mais externo se mantém até que se deseje sair.

```
begin
    menu;

    writeln('Digite um comando: ');
    readln(cmd);

while (cmd <> 's')do    //mantém o laço
   begin
        n1 := 0;
        n2 := 0;
        res := 0;

        if(cmd <> 'a')then
        begin
            writeln('Digite um numero: ');
            readln(n1);
        end;

        while (cmd <> 'a')do      //mantém
        begin
```

```
        begin
          if(cmd = 'a')then
          begin
               res := 0;
               writeln('Total na Memoria: ',res:3:4);
          end;
          writeln('Digite um comando: ');
          readln(cmd);
        end;

   end;
clrscr;
writeln('Voce saiu! Obrigado por utilizar a Calculadora.');
readln;
end.
```

Figura 3: à esquerda temos destacados os dois laços e à direita, a mensagem final.

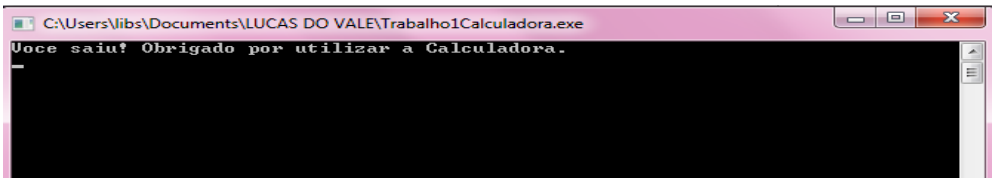

Figura 4: mensagem final na tela de execução

Ao digitar cada operação, devem-se levar em consideração dois grupos: o que exige dois números para realização do cálculo (‘+’, ’-’, ’*’, ‘/’, ‘p’) e o que exige apenas um número (‘i’, ‘r’). Para ambos os casos, temos a estrutura condicional **if...then**(que faz a diferenciação) e a **case...of** (que faz seleção), como mostra a Figura 5.

```
if((cmd = '+') OR (cmd = '-') OR (cmd = '*') OR (cmd = '/') OR (cmd = 'p'))then
begin
             writeln('Digite um numero: ');
             readln(n2);
end;
case cmd of

    '+': begin
            clrscr;
            menu;
            res := adicao(n1,n2);
         end;

    '-': begin
             clrscr;
             menu;
             res := subtracao(n1,n2);
         end;

    '*': begin
             clrscr;
             menu;
             res := multiplicacao(n1,n2);
```

Figura 5: diferenciação e alguns casos da respectiva seleção

Dependendo da condição, o programa faz chamadas à funções secundárias que tratam de cada comando digitado, fazendo o cálculo entre os valores como mostra a Figura 6. Caso seja verificada invalidez no comando, o usuário é alertado com uma mensagem, e a calculadora retorna a seu estado inicial como apresenta a Figura 7.

```
function adicao(x,y: real):real;
    begin
         adicao := x + y;
    end;
```

```
Digite um comando:
+
Digite um numero:
5
Digite um numero:
6
```

```
Total na Memoria: 11.0000
Digite um comando:
```

```
function raizQuadrada(x:real; var c: char):real;
    begin
        if(x < 0)then
            begin
                writeln('Número Imaginario!');
                c := 'a';
            end
        else
            begin
                raizQuadrada := sqrt(x);
            end;
    end;
```

```
Digite um comando:
r
Digite um numero:
81
```

```
Total na Memoria: 9.0000
Digite um comando:
```

Figura 6: algumas funções e seus respectivos efeitos na tela de execução

```
else
    begin
                if(cmd <> 'a')then
                begin
                     clrscr;
                     menu;
                     writeln('Comando Invalido!');
                     cmd := 'a';
                end;
    end;
```

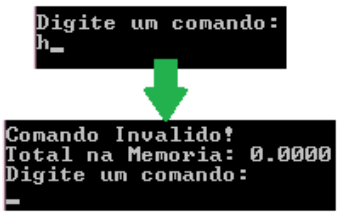

Figura 7: quando um comando inválido é digitado

Vale ressaltar que, para algumas operações (‘/’, ‘i’, ‘p’, ‘r’), faz-se necessário a aplicação de restrições, tomando como incoerentes determinadas relações matemáticas que o usuário tente estabelecer. Como exemplo disso, temos a divisão por **zero** que impossível de ser realizada. A incoerência é verificada dentro das respectivas funções, e dependendo da operação, o usuário recebe um alerta, como está representado na Figura 8. Em seguida, a memória é esvaziada.

```
function divisao(x,y: real; var c: char):real;       //ca
    begin
        if (y = 0) then
            begin
                writeln('Impossivel dividir por ZERO!'),
                c := 'a';        //c permite alteração n
            end
        else
            begin
                divisao := x/y;
            end;
    end;
```

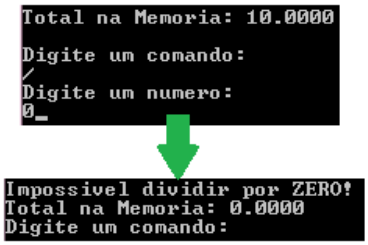

Figura 8: quando uma incoerência no cálculo é detectada

Como observado ao longo dos casos, nota-se que esta calculadora permite armazenar o valor de uma operação anterior, caso o usuário deseje utilizá-lo no cálculo. Para que seja descartado, como já visto, utiliza-se o comando ‘a’. Temos por fim, que tal programa executa a tarefa de memorização.

4. Conclusão

O desenvolvimento do código em si, foi simples, mas trabalhoso. Quanto à lógica de armazenamento de valores, houve certa dificuldade, pois foi preciso desenvolver um trecho que atendesse a diversos requisitos entrelaçados. As restrições devem ser consideradas nos momentos em que sejam identificadas incoerências, tomadas as entradas de dados. Por fim,

pode-se concluir que, visto esses e outros aspectos, a calculadora conseguiu cumprir com sua finalidade.