UNIVERSIDADE DE ÉVORA

Escola de Ciências e Tecnologias

Engenharia Informática

Arquitetura de Sistemas e Computadores I

2017/2018

1ºVersão – Arquitetura geral do programa

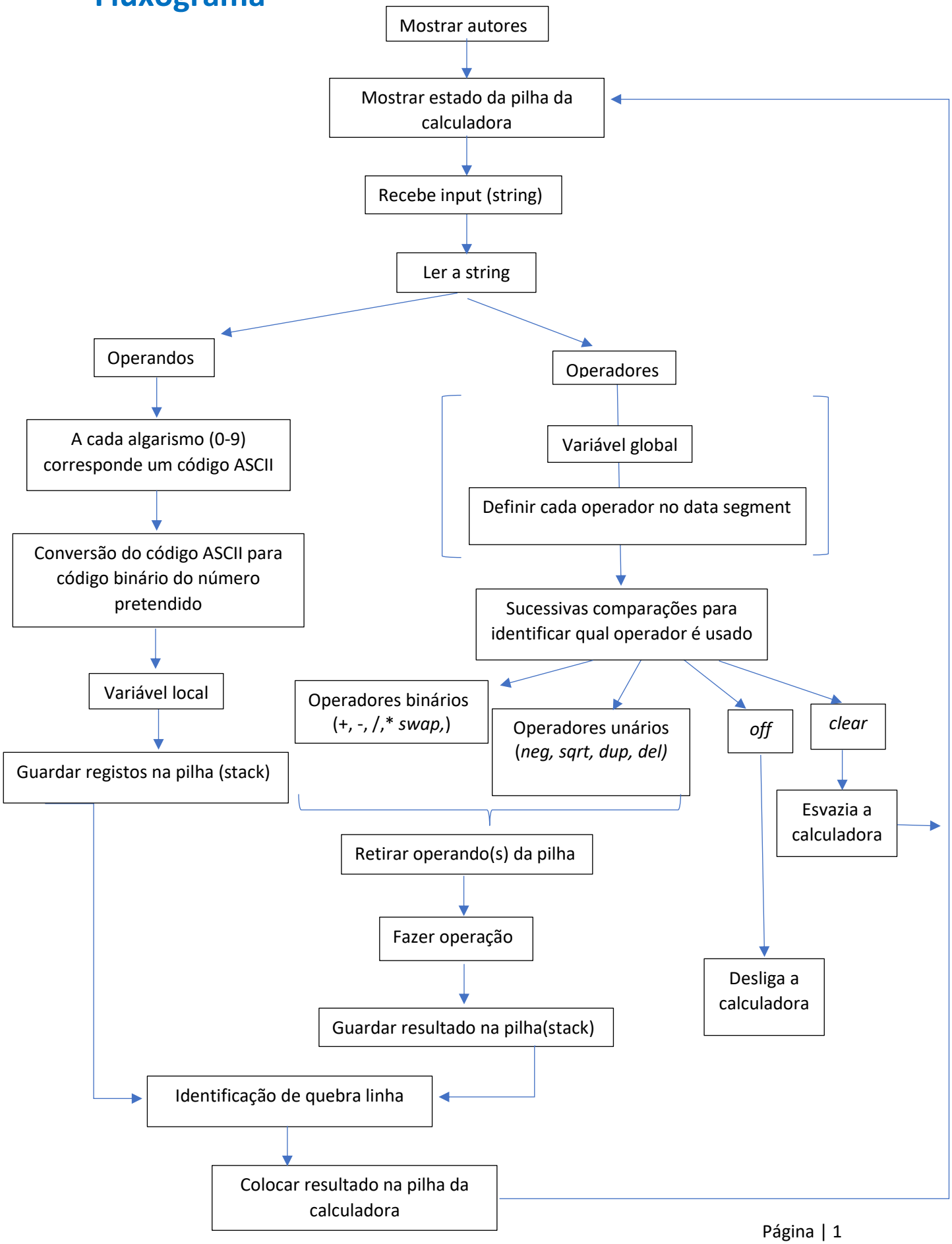
- Calculadora em Notação Polaca Inversa -

Docente: Miguel Barão

Discentes: Cláudia Dias – 35308

Ana Ferro – 39872

Fluxograma


Mostrar autores
Mostrar estado da pilha da calculadora
Recebe input (string)
Ler a string
Operandos
Operadores
A cada algarismo (0-9) corresponde um código ASCII
Variável global
Definir cada operador no data segment
Conversão do código ASCII para código binário do número pretendido
Sucessivas comparações para identificar qual operador é usado
Variável local
Operadores binários (+, -, /,* swap,)
Operadores unários (neg, sqrt, dup, del)
off
clear
Guardar registos na pilha (stack)
Esvazia a calculadora
Retirar operando(s) da pilha
Fazer operação
Desliga a calculadora
Guardar resultado na pilha(stack)
Identificação de quebra linha
Colocar resultado na pilha da calculadora

Calculadora em notação polaca inversa

- Descrição do problema:

Pretende-se desenvolver um programa em assembly MIPS para correr no simulador MARS. Este programa consiste em recriar uma calculadora que opere na notação polaca inversa, onde os operandos são introduzidos em primeiro, seguidos dos operadores. O programa deverá receber strings com a expressão que se pretende calcular, retornando o seu resultado através da pilha na consola. Os dados podem ser introduzidos um a um ou numa só linha, sendo estes separados por espaços ou por quebras-de-linha. Dependente da maneira como são introduzidos os dados, assim será o funcionamento do programa, caso seja introduzido um a um, cada vez que é dada uma quebra-de-linha deve-se mostrar o estado da pilha na consola, ao contrário de quando é introduzida a expressão, que só deverá mostrar o resultado no final da execução da mesma.

Input: String composta por operandos e/ou operadores (o utilizador pode decidir introduzir um a um ou numa só linha)

Output: Estado da memória da calculadora na forma de uma pilha (a quando da existência de uma quebra-de-linha)

- Funções

Depois de compreendido o que é pedido, chegou-se à conclusão que é necessária a criação de um loop principal e de várias funções.

1. **Função Mensagem();**
   Esta função tem como objetivo imprimir na pilha a seguinte mensagem:

```
*** Calculadora em notação polaca inversa***
***Autores: <aluno> (<nº de aluno>) e <aluno> (<nº de aluno>)***
***ASC1 2017/2018***
```

   Esta função será executada uma única vez no início do programa não fazendo parte do loop principal, uma vez que não é do nosso intuito imprimir esta mensagem na consola cada vez que é introduzido um novo dado na calculadora. Esta função tem como argumento uma string, ou seja, a mensagem descrita anteriormente. Como esta mensagem vai permanecer inalterada desde o inicio ao fim do programa será tratada como uma variável global, sendo definida no data segment onde teremos acesso a ela através da label: “Msg:”.

2. **Função Pedir_input();**
   Esta função possibilitará a introdução do input pelo utilizador. Para tal terá que se alocar espaço na pilha e de seguida executar-se um syscall que irá gerar uma exceção, e que durante a qual se poderá escrever a string. Quando se carregar no *enter* ou quando excedermos o tamanho da pilha, o sistema operativo retorna.

3. **Função Val_input();**
   Esta função tem como argumento o input introduzido pelo utilizador e tem como objetivo avaliar a sua validade. Para tal iremos recorrer a diversos *branches* que nos permitirão mudar o rumo de execução do nosso programa dependendo de qual condição é satisfeita.
   Pode existir três situações diferentes:
   - Não cumprir notação polaca inversa;
   - Erros de sintaxe (operadores mal escritos);
   - Código válido;

   Para identificar a primeira situação teremos que confirmar os seguintes requisitos:
   -Se os operadores são escritos antes dos operandos;
   -Se existem operandos suficientes para a operação pretendida. É de notar que existem dois tipos de operadores, os operadores unários que têm só um argumento e os operadores binários que necessitam de dois argumentos.

   Para identificar a segunda situação, teremos que comparar o primeiro caractere com o código identificador dos operadores (códigos esses já definidos no data segment).
   Se estas situações se confirmarem, serão impressas na consola da pilha da calculadora as seguintes mensagens de erro:

1ºCaso: “Input inválido: Não cumpre as normas da notação polaca inversa” 2ºCaso: “Input Inválido: Operador desconhecido”

   Se não se verificar as situações anteriores, verificamos que o código introduzido é válido. Ao chegarmos a esta conclusão o programa pode continuar para a função seguinte, função Ler_input().

4. **Função Ler_input();**
   Esta função tem como objetivo descodificar o que o input pretende representar, ou seja identificar os operandos e os operadores. Para tal vai criar um ciclo que vai avaliar cada caracter um a um através de diversos *branches.*

O argumento desta função é uma string (input do utilizador), que pode ser constituída por operandos (números constituídos pelos algarismos de 0-9), operadores, espaços e quebras-de-linha.
Uma vez que os operandos, nesta primeira fase, ainda se encontram codificados de acordo com o código ascii, o modo de concluir que se trata de um algarismo é descobrir se esse carater encontra-se entre o intervalo de 30 a 39 (código binário equivalente aos algarismos entre 0 e 9). Se tal se confirmar faz-se um *jump* para a função Conversao().
Para averiguar que se trata de um espaço(Tab), compara-se ao código identificador de espaço que foi anteriormente definido no data segment, uma vez que se trata de uma variável global. Se tal se confirmar o programa passa para o caracter seguinte.
Este mecanismo de identificação é igual para a quebra-de-linha. Quando se deparar com uma quebra-de-linha o programa faz um *jump* para a função Pilha(). Se nenhuma destas condições se confirmar o programa salta para a função Operadores().

5. **Função Conversao();**
   Esta função recebe como argumento o código ascii correspondente a um algarismo. O seu objetivo é converter esse código para o código binário do verdadeiro valor do algarismo.
   Para chegar a esse resultado temos que subtrair 30 a esse código. Por exemplo, o algarismo 1 está codificado como sendo 31, ao subtrairmos 30 ficamos com o código a 1. Este mecanismo repete-se para os restantes algarismos.
   É de notar que para numeros com dois ou mais algarismos é necessário converte-lo num só código, representando assim um único número. Para tal será necessário fazer multiplicações por 10 e somas para chegar ao número pretendido.
   Depois de executada a conversão guarda o número na pilha(stack). Retorna à função Ler_input().

6. **Função Operadores();**
   A função recebe como argumento o código ascii de um operador. O objetivo desta função é identificar qual operador representa.
   Como os códigos identificadores dos operadores se manterão inalterados desde o inicio até ao fim do programa, estes são tratados como sendo variáveis globais que já estão definidas no data segment.
   Para descobrir qual operador se trata recorrer-se-á a diversas comparações entre o argumento e as variáveis globais. Dependente de qual condição é satisfeita, assim será o jump para a função seguinte. Por exemplo, se o programa identificar o operador como sendo o da adição, o programa salta para a função da operação correspondente, função Add().
   É de notar que esta função faz um *jump* para a função Dup() sempre que se identifica o operador *dup* ou quando estamos na presença de uma linha vazia.

7. **Função Add();**
   Esta função recebe como argumentos dois operandos e tem como objetivo executar a soma entre eles. Esta função vai buscar os seus argumentos à pilha(últimos dois valores a serem introduzidos) e guarda o resultado da operação também na pilha. Retorna à função Ler_input().

8. **Função Sub();**
   A função recebe dois operandos como argumentos e tem como objetivo realizar a operação de subtração entre eles. Esta função descreve o mesmo comportamento que a função anterior, indo buscar os seus argumentos e guardando o resultado na pilha. Retorna à função Ler_Input().

9. **Função Div();**
   Esta função recebe dois operandos como argumentos provenientes da pilha(stack) e tem como objetivo realizar a divisão dos mesmos, guardando o resultado da operação na pilha. No final, retorna à função Ler_input().

10. **Função Mul();**
    Por se tratar de um operador binário, esta função recebe dois operandos como argumentos (provenientes da pilha) e pretende realizar a multiplicação dos mesmos, guardando o resultado na pilha. Retorna à função Ler_input().

11. **Função Swap();**
    Esta função recebe os últimos dois operandos provenientes da pilha como argumentos e tem como objetivo trocar a posição dos mesmos no topo da pilha. Essa alteração é guardada na pilha e retorna à função Ler_input().

12. **Função Neg();**
    O objetivo desta função é calcular o simétrico. Para tal vai buscar o último valor introduzido na pilha como argumento e guarda o resultado na mesma. No final, retorna à função Ler_input().

13. **Função Dup();**
    Por tratar-se de um operador unário, esta função possui apenas um argumento (operando do topo da pilha). Com esta função pretende-se duplicar o argumento, adicionando à pilha o seu duplicado. Retorna à função Ler_input().

14. **Função Sqrt();**
    Tal como a anterior esta função apenas possui um argumento (último operando inserido na pilha). E tem como objetivo fazer a raiz quadrada do valor do operando, guardando o seu resultado na pilha. Por último, retorna à função Ler_input().

15. **Função Clear();**
    Esta função limpa toda a pilha, ou seja, elimina todos os registos da pilha, ficando esta vazia. Após a sua execução, o programa salta para a função Pilha().

16. **Função Del();**
    Esta função tem como objetivo remover um operando do topo da pilha (em caso de engano), recebendo como argumento esse mesmo operando. Retorna à função Pedir_input().

17. **Função Off();**
    O objetivo desta função é desligar a calculadora. Para tal se fará um jump para a *label* "FIM:", onde se dará por terminado o programa.

18. **Função Pilha();**
    Esta função tem como objetivo imprimir o estado atual da pilha na consola da calculadora, para tal vai buscar o operando do topo da pilha(stacK). É executada no início do programa, mostrando que a pilha está vazia; sempre que é identificado uma quebra de linha e quando se executa a função Clear().

19. **Função Main();**
    Esta função será a responsável pela execução do loop principal. Nela estará contida todas as outras funções descritas anteriormente (à exceção da função Mensagem()), sendo a "chave" para execução de um programa contínuo.