

## **Relatório Trabalho 2 - PIHS**

**Guilherme Zamberlam Pomini RA: 99345<sup>1</sup> Diogo Fernando de Melo Sales RA: 93814<sup>1</sup>**

<sup>1</sup>Professor: Ronaldo Augusto de Lara Gonçalves  
Disciplina: Programação para Interfaceamento de Hardware e Software  
Ciência da Computação  
Universidade Estadual de Maringá (UEM)  
Sede Maringá – Paraná – Brasil

ra99345@uem.br, ra93814@uem.br



### **Introdução e Objetivo**

O presente documento contém relatório do segundo trabalho da matéria ministrada pelo Ronaldo Augusto de Lara Gonçalves na disciplina de Programação para Interfaceamento de Hardware e Software para a turma de Bacharelado em Ciência da Computação no primeiro semestre de 2018.

O objetivo do segundo trabalho era implementar um programa em linguagem Assembly padrão 32 bits que lesse uma expressão matemática digitada pelo usuário, a interpretasse e a resolvesse, similar ao funcionamento de uma calculadora científica padrão. Similarmente a uma calculadora o programa deve fazer o tratamento de prioridade de operações, funções e sub-expressões delimitadas por parênteses.

A expressão matemática pode conter tanto operações básicas como soma, subtração, multiplicação e divisão, quanto operações matemáticas mais complexas como potências, raiz quadrada, logaritmos na base 10 e funções trigonométricas como seno, cosseno e tangente. Pode-se colocar parênteses como forma de controlar as prioridades das operações mas as funções não aceitam argumentos complexos. A expressão trabalha com números reais, dando o resultado portanto em um número real com ponto flutuante, mas também aceita números inteiros normalmente.

### **Principais Componentes**

O programa desenvolvido possui dois arquivos de códigos principais, um arquivo em linguagem C e Assembly 32 bits chamado principal, e um arquivo em linguagem Assembly 32 bits chamado calcexpress.

A funcionalidade do arquivo principal.c é realizar a leitura de uma expressão utilizando Assembly Inline e chamadas ao sistema, sem chamadas a bibliotecas e resolver

a expressão utilizando Assembly Outline. A leitura da expressão é feita com a função `le_expressao` como mostra a Figura 1

```
void le_expressao(char expressao[], char fim_string[]){
    asm(
        "movl 8(%esp), %ecx;" // endereço da expressao para %ecx
        "movl 12(%esp), %edx;" // endereço do fim de string %edx
        "movl (%edx), %edx;" // move o valor do fim de string para o %edx
        "pushl %edx;" // salva %edx
        "pushl %ecx;" // salva %ecx
        "movl $3, %eax;" // função de leitura
        "movl $2, %ebx;" // leitura do teclado
        "movl $200, %edx;" // tamanho da expressao
        "int $0x80;" // chama a função de leitura para ler para a variavel expressao que está com seu endereço em %ecx
        // Foi feita a leitura
        // Agora é preciso acrescentar o fim de string na string lida
        "popl %ecx;" // Recupera o valor do endereço da expressao caso ele foi alterado
        "determinastrutil:;" // Começa a percorrer a expressao até encontrar o termino dela (espaço ou pula linha)
        "movl %ecx, %edi;"
        "movl $-1, %ebx;"
        "volta:;"
        "addl $1, %ebx;"
        "movb (%edi), %al;"
        "cmpb $'\n', %al;"
        "jz conclui;"
        "cmpb $' ', %al;"
        "jz conclui;"
        "addl $1, %edi;"
        "jmp volta;"
        "conclui:;" // Quando encontrar o fim precisa inserir o fim de string na mesma
        "popl %edx;" // recupera o valor do %edx que contem "\0" que representa o fim de string
        "movl %edx, (%edi);" // move o fim de string para a posição correta
    );
}
```

Figura 1. Função `le_expressao`

A função é implementada com Assembly Inline e somente chamadas ao sistema, consiste primeiramente em ler uma expressão digitada pelo usuário para o endereço da string passada como parâmetro. Feita a leitura é necessário encontrar o fim da expressão e colocar um caractere simbolizando o fim de de string, finalizando a função.

O corpo main do arquivo principal.c como mostra a Figura 2 consiste da declaração das variáveis para armazenar a expressão e a resposta, de prints para o usuário do funcionamento do programa, chamada da função para ler a expressão e chamada outline para resolver a expressão implementada no arquivo `calcexpress.s`

```
int main(){
    char expressao[200];
    char fim_string[10] = "\0";
    double resposta;
    printf("Programa para interpretar expressões matemáticas aceitando parênteses\n");
    printf("Operações suportadas: +,-,*,./\n");
    printf("Funções suportadas: seno(x), cosseno(x), tangente(x), pow(x,y), raiz(x), log(x)\n");
    printf("Digite a expressão: \n");

    le_expressao(expressao, fim_string); // chamada inline para ler a expressao
    calcexpress(expressao, &resposta); // chamada outline para resolver a expressao
    printf ("Resposta = %.2lf\n", resposta);
    //printa_expressao(&resposta);
    return 0;
}
```

Figura 2. Main arquivo principal.c

Funcionalidades do arquivo `calcexpress.s` :

- **calcexpress** é o procedimento que o programa principal chama para resolver a expressão, é passado como parâmetro uma string que é a expressão a ser resolvida e um endereço de um double onde vai ser escrito o resultado. É este o procedimento

que chama os demais procedimentos de criar uma lista duplamente encadeada a partir da expressão e o de resolve-la. A Figura 3 mostra as funcionalidades do procedimento.

```
calcexpress:
    movl 4(%esp), %edi          # pega o endereço da expressao passado
    movl %edi, endereco

    movl 8(%esp), %edi          # pega o endereço do float em que o resultado será escrito
    movl %edi, endereco_retorno

    call cria_lista              # cria a lista a partir da string endereço
    movl listatoken, %edi        # move a lista criada para o %edi para poder iniciar a resolução da lista
    call resolve_parenteses      # chama a resolução dos parenteses que é a função responsável por resolver a lista

    ret
```

**Figura 3. Procedimento calcexpress**

- **cria\_lista** procedimento para transformar a expressão em forma de string para uma lista duplamente encadeada, recebe o endereço da string lida pelo procedimento ler\_expressao no programa principal e passada para o procedimento calcexpress. Realiza a leitura caractere por caractere da expressão, faz os tratamentos necessários como resolver as funções, ler os números e operadores matemáticos. É o procedimento que converte os números em forma de String para ponto flutuante antes de fazer a inserção dos mesmos na lista. Ao final desse procedimento todas as funções foram resolvidas e seus resultados já estão em formato ponto flutuante como um elemento da lista, restando uma lista duplamente encadeada onde seus elementos são somente números reais, operadores matemáticos (+,-,/,\*) e parênteses.
- **resolve\_parenteses** procedimento que faz a varredura da lista procurando sub-expressões delimitadas por abre e fecha parênteses para serem resolvidas. A cada sub-expressão encontrada o procedimento chama o procedimento reduz para resolver somente o pedaço da lista correspondente a sub-expressão. Terminado a resolução de todos os parênteses é chamado novamente o procedimento reduz para resolver o restante da lista desde o começo.
- **reduz** procedimento principal para resolução da expressão, recebe um endereço da lista que é passado pelo procedimento resolve\_parenteses e a resolve até chegar em um final de lista. A resolução é feita sequencialmente em três passadas pela lista, primeiramente tratando os números, em seguida as multiplicações e divisões e por fim as somas e subtrações. A cada tratamento de operações a lista vai sendo reduzida até sobrar somente um único elemento que é o resultado da expressão.

## Funcionamento

A primeira parte do programa se dá no arquivo principal.c. É nele que é chamado a função ler\_expressao que utiliza de Assembly Inline para ler a expressão digitada pelo usuário utilizando somente chamadas ao sistema. Após a leitura o programa chama a função calcexpress para resolver a expressão, por ser um procedimento onde sua definição está no arquivo calcexpress.s, esta é uma chamada utilizando Assembly Outline.

Dentro do procedimento calcexpress será resolvida a expressão com o primeiro passo sendo transformar a mesma para uma lista duplamente encadeada onde cada número e operação é um elemento da lista. Os elementos da lista são da forma de 20 bits, sendo

4 para o tipo do elemento, 8 para o valor, 4 para o endereço do elemento anterior e 4 para o endereço do elemento posterior. A conversão é feita por um método `cria_lista` que vai percorrer a expressão caractere por caractere para reconhecer se é um número, operação ou uma função e realiza o seu respectivo tratamento.

A função `cria_lista` possui um tratamento para cada tipo de elemento encontrado antes de fazer a inserção na lista de fato. Quando encontrado um caractere numérico o mesmo é lido até o fim e convertido para ponto flutuante independente se é um número inteiro ou número real. Quando encontrado uma operação matemática básica a mesma é inserida normalmente na lista sem necessidade de conversão. O tratamento das funções como Logaritmo, Raíz, Potência, Seno, Cosseno e Tangente é o mesmo, é encontrado o seu valor dentro dos respectivos parênteses, convertido para ponto flutuante e calculado a sua função, sendo essa a única parte que realmente difere de fato pois cada função possui uma chamada diferente para o seu cálculo. Como o cálculo das funções é feito na hora de criar a lista, o que é inserido na lista é o seu valor de fato como um número real, portanto as funções são resolvidas no momento da criação da lista.

Após a criação da lista é necessário resolvê-la, se fosse uma expressão normal bastaria chamar a função `reduz`. Contudo, como a expressão pode ou não conter sub-expressões que delimitam prioridades sobre outras operações o primeiro passo para resolver a lista é encontrar tais sub-expressões.

Tal problema é resolvido com o procedimento `resolve_parenteses` que percorre a expressão resolvendo as suas sub-expressões até chegar no momento em que todos os parênteses foram eliminados, e resolvendo a expressão restante para dar o resultado desejado. O procedimento se baseia em percorrer a expressão até encontrar um token simbolizando um fechamento de parênteses, quando encontrado é chamado o método `reduz` que resolve expressões para resolver somente entre o endereço do último abertura de parênteses até o fechamento de parênteses encontrado, retirando todos os elementos utilizados e substituindo pelo resultado. A cada sub-expressão resolvida o procedimento começa a varredura do começo da lista, sempre garantindo pegar as sub-expressões mais aninhadas que possuem maior prioridade sobre as outras.

O procedimento `reduz` tem o objetivo de reduzir a lista a somente um elemento que seria o resultado da expressão. Recebe como endereço inicial o endereço de uma parte da expressão passada pelo método `resolve_parenteses`, resolvendo até encontrar o fim da lista passada. Consiste em três passadas pela lista, a primeira fazendo o tratamento dos números negativos, encontrando um token de menos e um número após e transformando em somente um elemento que seria o número negativado. A segunda passada resolve somente divisões e multiplicações por terem prioridade sobre a soma, resolvendo na ordem que as mesmas vão aparecendo. E por fim a terceira passada sobre a lista termina resolvendo as somas e subtrações que são as únicas operações restantes.

Tanto na segunda passada quanto na terceira o método de resolução é o mesmo, após encontrar uma operação entre dois números os mesmos são enviados para a pilha de ponto flutuante (FPU) e é identificada a operação, chamando seu respectivo tratamento através da FPU. Após computado o resultado o mesmo é sobrescrito no endereço do primeiro operando da operação e os dois elementos posteriores (a operação e o segundo operando) são retirados da lista através da manipulação dos seus ponteiros. Tal repetição

de retirada da lista vai levar a um momento em que só existirá um único elemento na lista, o resultado esperado da expressão.

Portanto, a resolução da lista é baseada nos métodos `resolve_parentheses` que encontra as sub-expressões para serem resolvidas (a expressão sem parênteses é uma sub-expressão própria) utilizando o método `reduz`. Ao final o resultado será o único elemento da lista que será passado para o endereço de retorno do procedimento `calcexpress` para ser printado pelo usuário no programa principal.

## Capacidades e Limitações

Lista das capacidades do programa:

1. Leitura da expressão com `Assembly Inline`;
2. Resolução da expressão com `Assembly Outline`;
3. Tratamento para números em ponto flutuante, aceitando tanto inteiros quanto números reais e tratando os dois da mesma forma (como ponto flutuante) através da mesma conversão;
4. Tratamento prioridade de sub-expressões delimitadas por parênteses;
5. Trata espaços em branco como divisão entre elementos na escrita da expressão;
6. Trata precedência de operações, resolvendo primeiramente as funções enquanto cria a lista, multiplicação e divisão, e por fim soma e subtração.
7. Soma de dois números reais;
8. Subtração de dois números reais;
9. Divisão de dois números reais;
10. Multiplicação de dois números reais;
11. Potência na forma  $x$  elevado a  $y$ ;
12. Raiz quadrada de um número real;
13. Logaritmo na base 10 de um número real;
14. Seno de um número real (passado em forma de radianos);
15. Cosseno de um número real (passado em forma de radianos);
16. Tangente de um número real (passado em forma de radianos);
17. Tratamento para números negativos.

E as limitações do programa são:

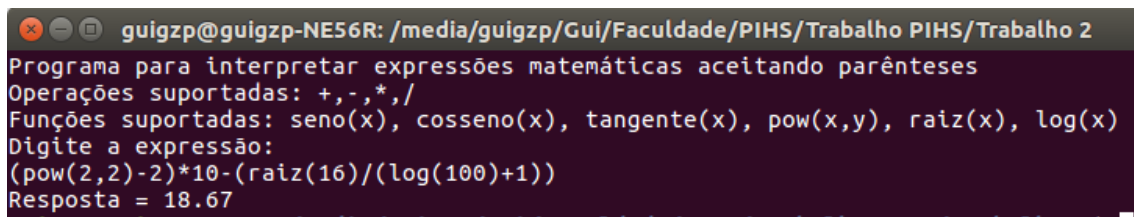
1. Falta de uma função para verificar a ordem dos elementos da lista, para cancelar a execução caso os elementos não estejam da forma esperada para a resolução;
2. A expressão deve possuir um tamanho máximo de 200 caracteres;
3. Não são aceitos argumentos compostos para expressões como da forma `raiz(10+10)`;
4. O print do resultado final não é feito com chamada ao sistema por não ter sido encontrado uma maneira de printar um float sem chamadas de bibliotecas, fica como sugestão para melhoria do trabalho numa futura atualização.

## Conclusão

O objetivo deste trabalho era implementar um interpretador de expressões matemáticas em linguagem `Assembly 32 bits` que trabalhasse tanto com operações básicas quanto algumas mais complexas, realizando também o tratamento de prioridades de sub-expressões

através de parênteses. O programa deveria ser dividido em duas partes, a primeira realizando a leitura da expressão com Assembly Inline e a segunda sendo um procedimento Assembly Outline para resolver a expressão.

Ao final do trabalho foi possível chegar no resultado esperado produzindo um programa nas especificações desejadas, salvo as limitações listadas, que resolve desde expressões simples até as mais complexas com prioridades. A Figura 4 mostra a resolução da expressão  $(2^2 - 2) * 10 - (\sqrt{16}/(\log_{10} 100 + 1))$  como era de se esperar de uma calculadora.



```
guigzp@guigzp-NE56R: /media/guigzp/Gui/Faculdade/PIHS/Trabalho PIHS/Trabalho 2
Programa para interpretar expressões matemáticas aceitando parênteses
Operações suportadas: +,-,*,/
Funções suportadas: seno(x), cosseno(x), tangente(x), pow(x,y), raiz(x), log(x)
Digite a expressão:
(pow(2,2)-2)*10-(raiz(16)/(log(100)+1))
Resposta = 18.67
```

Figura 4. Resolução expressão

## Referências

Blum, R. (2005). *Professional Assembly Language*. Wrox Press Ltd., Birmingham, UK, UK.