UNIVERSIDADE ESTADUAL PAULISTA
FACULDADE DE CIÊNCIAS E TECNOLOGIA

BACHARELADO EM CIÊNCIA DA COMPUTAÇÃO

TRABALHO PRÁTICO 2 - CALCULADORA

MICROPROCESSADORES

PROF. DR. ALMIR OLIVETTE ARTERO

LUIZ HENRIQUE SERAFIM DA SILVA

PRESIDENTE PRUDENTE

2022

1 - AssemblyFunctions.h

```
#pragma once
class AssemblyFunctions
{
public:

	AssemblyFunctions()
	{
	}

	~AssemblyFunctions()
	{
	}

	float soma(float a, float b) {
		__asm {
			finit
			fld a
			fld b
			faddp st(1),st(0)
			fstp a
		}
		return a;
	}

	float subtracao(float a, float b) {
		__asm {
			finit
			fld a
			fld b
			fsubp st(1), st(0)
			fstp a
		}
		return a;
	}

	float divisao(float a, float b) {
		__asm {
			finit
			fld a
			fld b
			fdivp st(1), st(0)
			fstp a
		}
		return a;
	}

	float multiplicacao(float a, float b) {
		__asm {
			finit
			fld a
			fld b
			fmulp st(1), st(0)
			fstp a
		}
		return a;
	}
```

```
float sin(float a,int isRadiano) {
	__asm {
		finit
		fld a
		mov eax, isRadiano
		sub eax, 1
		jz _GRAU
		JMP _RADIANO

		_GRAU:
			call AssemblyGrauRaidano
			JMP _RADIANO

		_RADIANO:
			fsin
			fstp a
	}
	return a;
}

float cos(float a,int isRadiano) {
	__asm {
		finit
		fld a
		mov eax, isRadiano
		sub eax, 1
		jz _GRAU
		JMP _RADIANO

		_GRAU :
			call AssemblyGrauRaidano
			JMP _RADIANO

		_RADIANO :

			fcos
			fstp a
	}
	return a;
}

float tg(float a,int isRadiano){
	__asm {
		finit
		fld a
		mov eax, isRadiano
		sub eax, 1
		jz _GRAU
		JMP _RADIANO

		_GRAU :
			call AssemblyGrauRaidano
			JMP _RADIANO

		_RADIANO :

			fsincos
			fdivp st(1), st(0)
			fstp a
	}
	return a;
}
```

```
float sqrt(float a) {
	__asm {
		finit
		fld a
		fsqrt
		fstp a
	}
	return a;
}

float xElevadoAoQuadrado(float a) {
	__asm {
		finit
		fld a
		fld a
		fmulp st(1), st(0)
		fstp a
	}
	return a;
}


float RaizNdeX(float x, float n) {
	__asm {
		finit
			fld1
			fld n
			fdiv
			fstp n
	}

	x = xElevadoAy(x, n);
	return x;
}

float log(float x,float a) {
	__asm {
		finit
			fld1
			fld a
			fyl2x
			fld1
			fdiv st, st(1)
			fld1
			fld x
			fyl2x
			fmul
			fstp x
	}

	return x;
}
```

```
float xElevadoAy(float x, float y) {
	__asm {
		finit
			fld y
			fld1
			fld x
			fyl2x
			fmul
			fld st
			frndint
			fsub st(1), st
			fxch
			f2xm1
			fld1
			fadd
			fscale
			fstp x
	}
	return x;
}

float fatorial(int x) {
	float result;
	__asm {
			finit
				fld1
				fldz
				mov ecx, x

				_loop :
				fld1
				faddp st(1), st(0)
				fmul st(1), st(0)

				dec ecx
				jnz _loop

				fstp x
				fstp result
		}
		return result;
}

float arctg(float x, int isRadiano) {
	__asm {
		finit
			fld x
			fld1
			fpatan
			mov eax, isRadiano
			sub eax, 1
			jz _GRAU
			jmp _RADIANO

			_GRAU:
				call AssemblyRadianoGrau
				JMP _RADIANO

			_RADIANO:
				fst x
	}
	return x;
```

```
}

float arcsin(float x, int isRadiano) {
	__asm {
		finit
			fld x
			fld x
			fmulp st(1), st(0)
			fld1
			fld st(1)
			fsubp st(1), st(0)
			fdivp st(1), st(0)

			fsqrt
			fld1
			fpatan

			mov eax, isRadiano
			sub eax, 1
			jz _GRAU
			jmp _RADIANO

			_GRAU :
		call AssemblyRadianoGrau
			JMP _RADIANO

			_RADIANO :
		fst x
	}
	return x;
}

float arccos(float x, int isRadiano) {
	int y;
	__asm {

		finit
			fld1
			fld x
			fld x
			fmulp st(1), st(0)
			fst y
			fsubp st(1), st(0)
			fld y
			fdivp st(1), st(0)
			fsqrt
			fld1
			fpatan

			mov eax, isRadiano
			sub eax, 1
			jz _GRAU
			jmp _RADIANO

			_GRAU :
		call AssemblyRadianoGrau
			JMP _RADIANO

			_RADIANO :
		fst x
	}
	return x;
```

```
	}

	void AssemblyGrauRaidano() {
		float constante = 180;
		__asm {
			fldpi
			fmulp st(1), st(0)
			fld constante
			fdivp st(1), st(0)
		}
		return;
	}

	void AssemblyRadianoGrau() {
		float constante = 180;
		__asm {
			fld constante
			fmulp st(1), st(0)
			fldpi
			fdivp st(1), st(0)
		}
	}
};
```

2 – Polonesa.h

```
#pragma once
#include "tabela.h"
#include <stack>
#include <queue>
#include <vector>
#include "AssemblyFunctions.h"
#include <string>

using namespace std;

class Polonesa
{

private:

public:

	Polonesa()
	{
	}

	~Polonesa()
	{
	}

	vector<string, allocator<string>> conversaoNotacao(vector<string,
allocator<string>> *expressaoOriginal) {
		int tipoToken;
		vector<string, allocator<string>> pilhaAuxiliar;
		vector<string, allocator<string>> filaSaida;
		string tokenAtual,topoPilha;
		unsigned int posiAtual = 0;

		while (posiAtual < expressaoOriginal->size()) {
			tokenAtual = expressaoOriginal->at(posiAtual);
			posiAtual++;
			if (tokenAtual.empty() == true)continue;
			tipoToken = getTipoToken(tokenAtual);

			switch (tipoToken) {
			case _NUMERO:
				if (tokenAtual.at(0) == _PI)tokenAtual =
to_string(CONSTPI);
				filaSaida.push_back(tokenAtual);
				break;
			case _NUMEROSINAL:
				topoPilha = _OP;
				pilhaAuxiliar.push_back(topoPilha);
				pilhaAuxiliar.push_back(tokenAtual.substr(1));
				break;
			case _OPERADOR:
				if (pilhaAuxiliar.empty() ==
true)pilhaAuxiliar.push_back(tokenAtual);
				else {
					while (pilhaAuxiliar.empty() == false) {
						topoPilha = pilhaAuxiliar.back();
						if (topoPilha.front() == '(')break;
						if (tokenAtual.front() == '^' &&
precedencia(tokenAtual, topoPilha) > 0) {
```

```
                                                filaSaida.push_back(topoPilha);
                                                pilhaAuxiliar.pop_back();
                                          }else if (precedencia(tokenAtual,
topoPilha) >= 0) {
                                                filaSaida.push_back(topoPilha);
                                                pilhaAuxiliar.pop_back();
                                          }
                                          else break;
                                    }
                                    pilhaAuxiliar.push_back(tokenAtual);
                              }
                              break;
                        case _FUNCAO:
                              pilhaAuxiliar.push_back(tokenAtual);
                              tokenAtual = _OP;
                              pilhaAuxiliar.push_back(tokenAtual);
                              break;
                        case _ABRIR_PARENTESES:
                              pilhaAuxiliar.push_back(tokenAtual);
                              break;
                        case _FECHAR_PARENTESES:
                              topoPilha = pilhaAuxiliar.back();
                              while (topoPilha.front() != _OP) {
                                    filaSaida.push_back(topoPilha);
                                    pilhaAuxiliar.pop_back();
                                    topoPilha = pilhaAuxiliar.back();
                              }
                              pilhaAuxiliar.pop_back();
                              if (pilhaAuxiliar.empty() == true)break;
                              topoPilha = pilhaAuxiliar.back();
                              if (getTipoToken(topoPilha) == _FUNCAO) {
                                    filaSaida.push_back(topoPilha);
                                    pilhaAuxiliar.pop_back();
                              }
                              break;

                        }
                  }

                  while (pilhaAuxiliar.empty() == false) {
                        filaSaida.push_back(pilhaAuxiliar.back());
                        pilhaAuxiliar.pop_back();
                  }

                  return filaSaida;
            }

            //menor que zero 'a' possui maior precedencia
            //igual a zero  'a' e 'b' possuem a mesma precedencia
            //maior que zero 'b' possui maior precedencia
            int precedencia(string a, string b) {
                  int precedenciaA, precedenciaB;
                  char operador;

                  operador = a.front();
                  if (operador == _ADD || operador == _SUB)precedenciaA = 1;
                  else if (operador == _MUL || operador == _DIV)precedenciaA = 2;
                  else if (operador == _XPOWY || operador == _POW2) precedenciaA = 3;
                  else if (operador == _FAT)precedenciaA = 4;

                  operador = b.front();
                  if (operador == _ADD || operador == _SUB)precedenciaB = 1;
```

```
		else if (operador == _MUL || operador == _DIV)precedenciaB = 2;
		else if (operador == _XPOWY) precedenciaB = 3;
		else if (operador == _FAT)precedenciaB = 4;

		return precedenciaB - precedenciaA;
	}

	int getTipoToken(string tokenAtual) {
		char primeiroCharString = tokenAtual.at(0);

		if (primeiroCharString == '(')return _NUMEROSINAL;

		if ( (primeiroCharString >= '0' && primeiroCharString <= '9')
			|| primeiroCharString == _PI || primeiroCharString == '-
')return _NUMERO;

		if (primeiroCharString == _OP)return _ABRIR_PARENTESES;

		if (primeiroCharString == _CP)return _FECHAR_PARENTESES;

		if ((primeiroCharString >= 'a' && primeiroCharString <= 'i'))return
_OPERADOR;

		return _FUNCAO;
	}

	string calcularPolonesa(vector<string, allocator<string>> filaExp,int
isRadiano) {
		vector<string, allocator<string>> pilhaDeCalculo;
		string tokenAtual,operadorA,operadorB;
		AssemblyFunctions operacoes;
		int tipoToken;

		while (filaExp.empty() == false) {
			tokenAtual = filaExp.front();
			filaExp.erase(filaExp.begin());
			tipoToken = getTipoToken(tokenAtual);
			switch (tipoToken) {
			case _NUMERO:
				pilhaDeCalculo.push_back(tokenAtual);
				break;
			case _OPERADOR:
				if (tokenAtual.front() != _FAT && tokenAtual.front()
!= _POW2){
					operadorB = pilhaDeCalculo.back();
					pilhaDeCalculo.pop_back();
				}
			case _FUNCAO:
				if ((tokenAtual.front() == _NROOT ||
tokenAtual.front() == _XPOWY) && tipoToken == _FUNCAO) {
					operadorB = pilhaDeCalculo.back();
					pilhaDeCalculo.pop_back();
				}
				operadorA = pilhaDeCalculo.back();
				pilhaDeCalculo.pop_back();
```

```
                        switch (tokenAtual.front()) {

                                case _ADD:

        pilhaDeCalculo.push_back(to_string(operacoes.soma(stof(operadorA),
stof(operadorB))));
                                        break;
                                case _MUL:

        pilhaDeCalculo.push_back(to_string(operacoes.multiplicacao(stof(operadorA)
, stof(operadorB))));
                                        break;
                                case _SUB:

        pilhaDeCalculo.push_back(to_string(operacoes.subtracao(stof(operadorA),
stof(operadorB))));
                                        break;
                                case _DIV:

        pilhaDeCalculo.push_back(to_string(operacoes.divisao(stof(operadorA),
stof(operadorB))));
                                        break;
                                case _POW2:

        pilhaDeCalculo.push_back(to_string(operacoes.xElevadoAoQuadrado(stof(opera
dorA))));
                                        break;
                                case _XPOWY:

        pilhaDeCalculo.push_back(to_string(operacoes.xElevadoAy(stof(operadorA),
stof(operadorB))));
                                        break;
                                case _FAT:

        pilhaDeCalculo.push_back(to_string(operacoes.fatorial(stof(operadorA))));
                                        break;
                                case _EXP:

        pilhaDeCalculo.push_back(to_string(operacoes.xElevadoAy(euler,
stof(operadorA))));
                                        break;
                                case _TG:

        pilhaDeCalculo.push_back(to_string(operacoes.tg(stof(operadorA),
isRadiano)));
                                        break;
                                case _LN:

        pilhaDeCalculo.push_back(to_string(operacoes.log(stof(operadorA),
euler)));
                                        break;
                                case _SIN:

        pilhaDeCalculo.push_back(to_string(operacoes.sin(stof(operadorA),isRadiano
)));
                                        break;
                                case _COS:

        pilhaDeCalculo.push_back(to_string(operacoes.cos(stof(operadorA),
isRadiano)));
                                        break;
```

```
                                    case _LOG:

        pilhaDeCalculo.push_back(to_string(operacoes.log(stof(operadorA),10)));
                                        break;
                                    case _NROOT:

        pilhaDeCalculo.push_back(to_string(operacoes.RaizNdeX(stof(operadorA),stof
(operadorB))));
                                        break;
                                    case _SQRT:

        pilhaDeCalculo.push_back(to_string(operacoes.sqrt(stof(operadorA))));
                                        break;
                                    case _ARCTG:

        pilhaDeCalculo.push_back(to_string(operacoes.arctg(stof(operadorA),
isRadiano)));
                                        break;
                                    case _ARCCOS:

        pilhaDeCalculo.push_back(to_string(operacoes.arccos(stof(operadorA),
isRadiano)));
                                        break;
                                    case _ARCSIN:

        pilhaDeCalculo.push_back(to_string(operacoes.arcsin(stof(operadorA),
isRadiano)));
                                        break;
                                }
                                break;
                            }
                        }
                        return pilhaDeCalculo.back();

        }

};
```

3 – Memory.h

```
#pragma once
#include <vector>
#include <string>
#include <allocators>
#include <queue>
#include "Polonesa.h"

using namespace std;
class Memory{
private: vector<string, allocator<string>> *expressao;

public:
	Memory() {
		expressao = new vector<string, allocator<string>>(0);
	}
	~Memory();


	int signal() {
		if (expressao->empty() == true) expressao->push_back("");
		string numAtual = expressao->back();
		int tamString;

		if (numAtual.empty() != true) {
			if (numAtual.at(0) == '(') {
				tamString = numAtual.size();
				expressao->back() = numAtual.substr(2,
numAtual.length());
				return -tamString;
			}
		}
		tamString = numAtual.length();
		expressao->back() = "(-" + expressao->back();
		return tamString;

	}

	bool inserirNovoNumero(char novoDig){
		string stringAtual;
		if (expressao->empty() == true)expressao->push_back("");

		stringAtual = expressao->back();
		if (stringAtual.length() == 1 && stringAtual.back() == '0') {
			expressao->back() = novoDig;
			return true;
		}
		else {
			expressao->back() += novoDig;
			return false;
		}
	}

	void novoOperador(char novoOperador) {
		expressao->push_back("");
		expressao->back() = novoOperador;
		expressao->push_back("");
	}

	int decimal() {
```

```
		if (expressao->empty() == true)expressao->push_back("");
		if (expressao->back().empty() == true) {
			expressao->back() = "0.";
			return 1;
		}
		if (expressao->back().find('.', 0) == -1) {
			expressao->back() += ".";
			return 0;
		}
		return 2;

	}

	int remover() {
		string stringAtual;
		char charAtual;

		if (expressao->empty() == true)return 0;

		stringAtual = expressao->back();

		if (stringAtual.empty() == true) {
			expressao->pop_back();
			stringAtual = expressao->back();
		}

		charAtual = stringAtual.back();
		expressao->back().pop_back();

		if (expressao->back().empty() == true)expressao->pop_back();

		if ((charAtual >= '0' && charAtual <= '9') || charAtual ==
','')return 1;
		if (charAtual >= 'a' && charAtual <= 'h')return 1;
		if (charAtual >= 'i' && charAtual <= 'j')return 2;
		if (charAtual >= 'k' && charAtual <= 'm')return 3;
		if (charAtual >= 'n' && charAtual <= 'q')return 4;
		if (charAtual >= 'r' && charAtual <= 'r')return 5;
		if (charAtual >= 's' && charAtual <= 's')return 6;
		if (charAtual >= 't' && charAtual <= 'u')return 7;
		return 0;
	}

	void clearMemory() {
		expressao->resize(1);
		expressao->back() = "";
	}

	//Se isRadiano == 2 então esta sendo usado radiano
	//se isRadiano == 1 então esta sendo usado graus
	string CalcularExpressao(int isRadiano){
		vector<string, allocator<string>> expPolonesa;
		Polonesa converter;
		expPolonesa = converter.conversaoNotacao(expressao);
		this->clearMemory();
		expressao->back() =
converter.calcularPolonesa(expPolonesa,isRadiano);
		string resultado = expressao->back();
		return expressao->back();
	}

};
```

4 – Tabela.h

```
#pragma once

//Operações que usam 1 char na string de exibição
#define _DIV 'a' //precedencia 2
#define _MUL 'b' //precedencia 2
#define _SUB 'c' //precedencia 1
#define _ADD 'd' //precedencia 1
#define _OP  'e'
#define _CP  'f'
#define _XPOWY 'g' //precedencia 3
#define _FAT  'h' //precedencia 4

//Operações que usam 2 char na string de exibição
#define _POW2 'i'
#define _PI   'j'

//Operações que usam 3 char na string de exibição
#define _EXP 'k'
#define _TG   'l'
#define _LN   'm'


//Operações que usam 4 char na string de exibição
#define _SIN 'n'
#define _COS 'o'
#define _LOG 'p'
#define _NROOT 'q'

//Operações que usam 5 char na string de exibição
#define _SQRT 'r'

//Operações que usam 6 char na string de exibição
#define _ARCTG 's'
//Operações que usam 7 char na string de exibição
#define _ARCCOS 't'
#define _ARCSIN 'u'

//Tipos de tokens
#define _OPERADOR 1
#define _ABRIR_PARENTESES 2
#define _FECHAR_PARENTESES 3
#define _NUMERO 4
#define _FUNCAO 5
#define _NUMEROSINAL 6


#define euler 2.7182818284
#define CONSTPI     3.14159265359
```