eFólio B - resolução (2016-2017)

Assuma no entanto que a entrada de dados não é superior a 256 caracteres.

Alínea A

```
#include <stdio.h>
#include <string.h>
#include <stdlib.h>

#define MAXSTR 256

int main() {
    int i;
    char str[MAXSTR], *operadores="+-*/", *pt;
    printf("Expressao:");
    gets(str);
    for(i=0;operadores[i]!=0;i++)
        if(strchr(str,operadores[i])!=NULL) {
            pt=strchr(str,operadores[i]);
            *pt=0;
            switch(i) {
                case 0:
                printf("%d",atoi(str)+atoi(pt+1));
                break;
                case 1:
                printf("%d",atoi(str)-atoi(pt+1));
                break;
                case 2:
                printf("%d",atoi(str)*atoi(pt+1));
                break;
                case 3:
                printf("%d",atoi(str)/atoi(pt+1));
                break;
            }
            break;
        }
}
```

Alínea B

```
#include <stdio.h>
#include <string.h>
#include <stdlib.h>

#define MAXSTR 256

int LerExpressao(char *str) {
    char *operadores="+-*/", *pt, operador;
    for(pt=str+strlen(str)-2;pt>str;pt--)
        if(strchr(operadores,*pt)!=NULL) {
            operador=*pt;
            *pt=0;
            switch(operador) {
                case '+':
                return LerExpressao(str)+atoi(pt+1);
                case '-':
                return LerExpressao(str)-atoi(pt+1);
```

```
                case '*':
                return LerExpressao(str)*atoi(pt+1);
                case '/':
                return LerExpressao(str)/atoi(pt+1);
            }
        }
    return atoi(str);
}

int main() {
    int i;
    char str[MAXSTR], *operadores="+-*/", *pt;
    printf("Expressao:");
    gets(str);
    printf("%d",LerExpressao(str));
}
```

Alínea C

```
#include <stdio.h>
#include <string.h>
#include <stdlib.h>

#define MAXSTR 256

typedef struct SExpressao {
    struct SExpressao *esquerda; // expressão à esquerda do operador
    char operador; // operador (se espaço vale apenas o valor)
    int valor; // valor da expressão caso seja um só número
    struct SExpressao *direita; // expressão à direita
} TExpressao;

// retorna uma nova expressão apenas com um valor
TExpressao *NovaExpressaoValor(int valor) {
    TExpressao *resultado=(TExpressao *)malloc(sizeof(TExpressao));
    if(resultado!=NULL) {
        resultado->esquerda=NULL;
        resultado->direita=NULL;
        resultado->operador=' ';
        resultado->valor=valor;
    } else
        printf("memory error");
    return resultado;
}

// retorna uma nova expressão com base em duas outras
TExpressao *NovaExpressao(char operador, TExpressao *esquerda, TExpressao *direita)
{
    TExpressao *resultado=(TExpressao *)malloc(sizeof(TExpressao));
    if(resultado!=NULL) {
        resultado->esquerda=esquerda;
        resultado->direita=direita;
        resultado->operador=operador;
        resultado->valor=0;
    } else
        printf("memory error");
    return resultado;
}

// liberta a expressão e sub-expressões
```

```
void LibertarExpressao(TExpressao *expressao) {
    if(expressao!=NULL) {
        if(expressao->esquerda!=NULL)
            LibertarExpressao(expressao->esquerda);
        if(expressao->direita!=NULL)
            LibertarExpressao(expressao->direita);
        free(expressao);
    }
}

TExpressao *LerExpressao(char *str) {
    int i;
    char *pt=NULL, *operadores="+-*/", operador;

    // localizar o primeiro operador fora de parêntesis
    for(i=0,pt=str; i>=0 && *pt!=0; pt++)
        if(*pt=='(')
            i++;
        else if(*pt==')')
            i--;
        else if(i==0 && strchr(operadores,*pt)!=NULL)
            break; // operador localizado, nível 0

    // ver se existe operador
    if(*pt==0 || i<0) { // não existe
        // se existirem parêntesis, avançar no parêntesis, c.c. é um valor atómico
        pt=strchr(str,'(');
        if(pt==NULL)
            return NovaExpressaoValor(atoi(str));
        else
            return LerExpressao(pt+1);
    }

    operador=*pt;
    *pt=0;
    return NovaExpressao(operador, LerExpressao(str), LerExpressao(pt+1));
}

void MostrarExpressao(TExpressao * expressao) {
    if(expressao->operador==' ')
        printf("%d",expressao->valor);
    else {
        if(expressao->esquerda->operador!=' ') { // tem parêntesis
            printf("(");
            MostrarExpressao(expressao->esquerda);
            printf(")");
        } else
            MostrarExpressao(expressao->esquerda);
        printf(" %c ",expressao->operador);
        if(expressao->direita->operador!=' ') { // tem parêntesis
            printf("(");
            MostrarExpressao(expressao->direita);
            printf(")");
        } else
            MostrarExpressao(expressao->direita);
    }
}

int main() {
    int i;
    TExpressao *expressao;
    char str[MAXSTR], *operadores="+-*/", *pt;
    printf("Expressao:");
```

```
    gets(str);
    expressao=LerExpressao(str);
    MostrarExpressao(expressao);
    LibertarExpressao(expressao);
}
```

Alínea D

```
#include <stdio.h>
#include <string.h>
#include <stdlib.h>

#define MAXSTR 256

typedef struct SExpressao {
    struct SExpressao *esquerda; // expressão à esquerda do operador
    char operador; // operador (se espaço vale apenas o valor)
    int valor; // valor da expressão caso seja um só número
    struct SExpressao *direita; // expressão à direita
} TExpressao;

// retorna uma nova expressão apenas com um valor
TExpressao *NovaExpressaoValor(int valor) {
    TExpressao *resultado=(TExpressao *)malloc(sizeof(TExpressao));
    if(resultado!=NULL) {
        resultado->esquerda=NULL;
        resultado->direita=NULL;
        resultado->operador=' ';
        resultado->valor=valor;
    } else
        printf("memory error");
    return resultado;
}

// retorna uma nova expressão com base em duas outras
TExpressao *NovaExpressao(char operador, TExpressao *esquerda, TExpressao *direita)
{
    TExpressao *resultado=(TExpressao *)malloc(sizeof(TExpressao));
    if(resultado!=NULL) {
        resultado->esquerda=esquerda;
        resultado->direita=direita;
        resultado->operador=operador;
        resultado->valor=0;
    } else
        printf("memory error");
    return resultado;
}

// liberta a expressão e sub-expressões
void LibertarExpressao(TExpressao *expressao) {
    if(expressao!=NULL) {
        if(expressao->esquerda!=NULL)
            LibertarExpressao(expressao->esquerda);
        if(expressao->direita!=NULL)
            LibertarExpressao(expressao->direita);
        free(expressao);
    }
}

TExpressao *LerExpressao(char *str) {
```

```
    int i,j;
    char *pt=NULL, *operadores[2]={"+-","*/"}, operador;

    // localizar o primeiro operador fora de parêntesis, SS
    for(j=0;j<2;j++) {
        for(i=0,pt=str+strlen(str)-1; i>=0 && pt>=str; pt--)
            if(*pt==')')
                i++;
            else if(*pt=='(') {
                i--;
            } else if(i==0 && strchr(operadores[j],*pt)!=NULL)
                break; // operador localizado, nível 0
        if(pt>=str)
            // operador encontrado, processar este e não ligar ao nível seguinte
            break;
    }

    // ver se existe operador
    if(pt<str || i<0) { // não existe
        // se existirem parêntesis, avançar no parêntesis, c.c. é um valor atómico
        if(pt<str && i>0) { // parêntesis a fechar a mais
            // cortar o último parêntesis
            pt=str+strlen(str)-1;
            while(pt>str && *pt!=')')
                pt--;
            *pt=0;
            return LerExpressao(str);

        } else if(i<0) { // parêntesis a abrir a mais
            pt=strchr(str,'(');
            return LerExpressao(pt+1);
        }
        // não existe parêntesis
        if(strchr(str,'(')==NULL && strchr(str,')')==NULL)
            return NovaExpressaoValor(atoi(str));
        // parêntesis a abrir e a fechar a mais
        str=strchr(str,'(');
        pt=str+strlen(str)-1;
        while(pt>str && *pt!=')')
            pt--;
        *pt=0;
        return LerExpressao(str+1);
    }

    operador=*pt;
    *pt=0;

    return NovaExpressao(operador, LerExpressao(str), LerExpressao(pt+1));
}

int TemParentesisEsquerda(char operador1, char operador2) {
    int i;
    char *operadores[2]={"+-","*/"};
    if(operador1==' ')
        return 0;
    for(i=0;i<2;i++)
        if(strchr(operadores[i],operador2)!=NULL &&
          strchr(operadores[i],operador1)!=NULL)
            return 0;
    if(strchr(operadores[1],operador1)!=NULL &&
      strchr(operadores[0],operador2)!=NULL)
        return 0;
    return 1;
```

```
}

int TemParentesisDireita(char operador1, char operador2) {
    int i;
    char *operadores[2]={"+-","*/"};
    if(operador2==' ')
        return 0;
    if(operador1=='+' && strchr(operadores[0],operador2)!=NULL ||
      operador1=='*' && operador2=='*')
        return 0;
    // operador2 não pode ser divisão A*(B/C) original não pode passar para A*B/C
por causa dos arredondamentos
    // apenas (A*B)/C é que pode passar para A*B/C
    // casos de teste realizados com a expressão:
    // operador1=='*' && strchr(operadores[1],operador2)!=NULL

    if(strchr(operadores[0],operador1)!=NULL &&
      strchr(operadores[1],operador2)!=NULL)
        return 0;
    return 1;
}

void MostrarExpressao(TExpressao * expressao) {
    if(expressao->operador==' ')
        printf("%d",expressao->valor);
    else {
        if(TemParentesisEsquerda(
            expressao->esquerda->operador,
            expressao->operador))
        {
            // tem parêntesis
            printf("(");
            MostrarExpressao(expressao->esquerda);
            printf(")");
        } else
            MostrarExpressao(expressao->esquerda);
        printf(" %c ",expressao->operador);
        if(TemParentesisDireita(
            expressao->operador,
            expressao->direita->operador))
        {
            // tem parêntesis
            printf("(");
            MostrarExpressao(expressao->direita);
            printf(")");
        } else
            MostrarExpressao(expressao->direita);
    }
}

int AvaliaExpressao(TExpressao *expressao) {
    if(expressao!=NULL) {
        switch(expressao->operador) {
            case ' ':
            return expressao->valor;
            case '+':
            return AvaliaExpressao(expressao->esquerda)+
                AvaliaExpressao(expressao->direita);
            case '-':
            return AvaliaExpressao(expressao->esquerda)-
                AvaliaExpressao(expressao->direita);
            case '*':
            return AvaliaExpressao(expressao->esquerda)*
```

```
                AvaliaExpressao(expressao->direita);
            case '/':
            return AvaliaExpressao(expressao->esquerda)/
                AvaliaExpressao(expressao->direita);
        }
    }
    return 0;
}

int main() {
    int i;
    TExpressao *expressao;
    char str[MAXSTR], *operadores="+-*/", *pt;
    printf("Expressao:");
    gets(str);
    expressao=LerExpressao(str);
    MostrarExpressao(expressao);
    printf("\n%d",AvaliaExpressao(expressao));
    LibertarExpressao(expressao);
}
```