Relatório de INF1010 - Trabalho 1 (Lista heterogênea)

Gustavo Molina Soares (2020209)
Leo Land Bairros Lomardo (2020201)

Durante o desenvolvimento do laboratório elaboramos todas as funções que foram pedidas e vamos explicar como utilizar cada uma delas na main.

- criaLista(); Essa função não requer parâmetro e vai retornar o valor nulo para inicializar a lista.

- imprimeFigura(Lista *dados, int tipo); Essa função recebe dois parâmetros e vai imprimir os dados das figuras solicitadas, no parâmetro tipo podemos inserir (ALL,TRI,QUA,HEX,CIR,RET) para obter dados da figura requisitada e o primeiro parâmetro é a lista a ser analisada.

- buscaLista(Lista *dados, int index); Essa função recebe dois parâmetros e retorna um ponteiro para o index inserido no segundo parâmetro, lembrando que o primeiro parâmetro é a lista a ser analisada, então caso eu queira a informação na 3 posição é só realizar buscaLista(dados, 3).

- retiraLista(Lista *dados, Lista *retira); Essa função recebe dois parâmetros e retorna a lista com o elemento passado no segundo
parâmetro sendo retirado da lista, ela funciona em conjunto com a buscaLista.

- freeLista(Lista *dados); Essa função recebe um parâmetro e ela vai liberar a memória ocupada pela lista.

- isEmpty(Lista *dados); Essa função recebe um parâmetro e informa se a lista está vazia ou preenchida.

Comentários:

A gente não apresentou muitas dificuldades durante o trabalho já que já temos conhecimento prévio de listas encadeadas e não teve muitas funções complicadas para implementar,a dificuldade apresentada foi como realizar a busca especificamente para o triângulo, mas percebemos que não seria eficiente fazer um programa que funciona apenas para triângulos, então a busca pode ser realizada para qualquer index, e caso queira alterar a figura geométrica que quer analisar é possível também, mas em geral o trabalho correu bem e conseguimos implementar tudo que foi pedido.

Outro detalhe foi que usamos a biblioteca math.h, então para compilar o código tem que usar gcc -Wall -o nomedoarquivo nomedoarquivo.c -lm

Código Fonte:

```c
#include <stdio.h>
#include <stdlib.h>
#include <math.h>

#define ALL -1
#define QUA 0
#define TRI 1
#define HEX 2
#define RET 3
#define CIR 4

/* structs das formas */

typedef struct triangulo{
    float b;
}Triangulo;

typedef struct hexagono{
    float b;
}Hexagono;

typedef struct retangulo{
    float b;
    float h;
}Retangulo;

struct circulo{
    float r;
};
typedef struct circulo Circulo;

struct quadrado{
    float b;
};
typedef struct quadrado Quadrado;

/* struct lista */
```

```c
struct lista{
  float area;
  float perimetro;
  int tipo;
  void *info;
  struct lista *prox;

};
typedef struct lista Lista;

/* calcular as áreas */

float areaRetang(Retangulo *r);
float areaTriang(Triangulo *t);
float areaCirc(Circulo *c);
float areaHexa(Hexagono *h);
float areaQuad(Quadrado *q);
Lista *calculaDados(Lista *dados);

/* calcular perimetro */

float perimRetang(Retangulo *r);
float perimTriang(Triangulo *t);
float perimCirc(Circulo *c);
float perimHexa(Hexagono *h);
float perimQuad(Quadrado *q);

/* cria as formas nas listas */

Lista* insereRetangulo(float b, float h, Lista* dados);
Lista* insereCirculo(float r, Lista* dados);
Lista* insereTriangulo(float b, Lista *dados);
Lista* insereHexagono(float b, Lista* dados);
Lista* insereQuadrado(float b, Lista *dados);

/* manipulacao de lista */

void imprimeDados(Lista* dado);

/* metodos para serem utilizados */
```

```c
Lista *criaLista(); // cria a lista
void imprimeFigura (Lista* dados, int tipo); // imprime a figura
especificada, caso queira a lista inteira insira ALL
Lista* buscaLista (Lista* dados, int index); // busca elemento na lista e
retorna de acordo com o index
Lista *retiraLista(Lista* dados, Lista *retira); // retira um elemento da
lista
void freeLista(Lista* dados); // libera memoria da lista
void isEmpty(Lista *dados); // verifica se a lista esta vazia

/* main */

int main (int argc, char **argv){
    Lista *dados = criaLista();

    dados = insereTriangulo(3.0, dados);
    dados = insereQuadrado(2.0, dados);
    dados = insereRetangulo(3.0, 5.0, dados);
    dados = insereTriangulo(5.0, dados);
    dados = insereCirculo(4.0, dados);
    dados = insereHexagono(3.0, dados);
    dados = insereTriangulo(4.0, dados);

    puts("Lista inicial:\n");

    imprimeFigura(dados, TRI); // Primeira impressão

    puts("");

    Lista * retira = buscaLista(dados, 3);
    Lista * retira1 = buscaLista(dados, 0);
    Lista * retira2 = buscaLista(dados, 6);

    dados = retiraLista(dados, retira);
    puts("Primeira Exclusão");
    imprimeFigura(dados, TRI); // imprimindo

    dados = retiraLista(dados, retira1);
    puts("Segunda Exclusão");
```

```
    imprimeFigura(dados, TRI); // imprimindo

    dados = retiraLista(dados, retira2);
    puts("Terceira Exclusão");
    imprimeFigura(dados, TRI); // imprimindo

    freeLista(dados);


    return 0;
}

// FUNÇÕES

float areaRetang(Retangulo *r){
    return(r->b*r->h);
}
float areaTriang(Triangulo *t){
    float area = (sqrt(3)/4)*pow(t->b,2);
    return (area);
}
float areaCirc(Circulo *c){
    float area = M_PI*pow(c->r,2);
    return area;
}
float areaHexa(Hexagono *h){
    float area = (3*sqrt(3)*pow(h->b,2))/2;
    return area;
}
float areaQuad(Quadrado *q){
    return (q->b * q->b);
}

Lista *calculaDados(Lista *dados){
        Lista *aux = dados;

        switch (dados->tipo){
            case QUA:
                aux->area  = areaQuad((Quadrado*)dados->info);
                aux->perimetro = perimQuad((Quadrado*)dados->info);
```

```
                dados = aux;
                break;

            case TRI:
                aux->area  = areaTriang((Triangulo*)dados->info);
                aux->perimetro = perimTriang((Triangulo*)dados->info);
                dados = aux;
                break;

            case HEX:
                aux->area  = areaHexa((Hexagono*)dados->info);
                aux->perimetro = perimHexa((Hexagono*)dados->info);
                dados = aux;
                break;

            case RET:
                aux->area  = areaRetang((Retangulo*)dados->info);
                aux->perimetro = perimRetang((Retangulo*)dados->info);
                dados = aux;
                break;

            case CIR:
                aux->area  = areaCirc((Circulo*)dados->info);
                aux->perimetro = perimCirc((Circulo*)dados->info);
                dados = aux;
                break;

            default:
                printf("Tipo inexistente");
                break;
        }

    return dados;
}


float perimRetang(Retangulo *r){
    return (r->b *2 + r->h *2);
}
float perimTriang(Triangulo *t){
```

```
    return (t->b * 3);
}
float perimCirc(Circulo *c){
    return (c->r * 2 * M_PI);
}
float perimHexa(Hexagono *h){
    return (h->b * 6);
}
float perimQuad(Quadrado *q){
    return (q->b * 4);
}

Lista* insereRetangulo(float b, float h, Lista* dados){
    Retangulo* ret;
    Lista* curr_data = NULL;


    if (!(dados)) {

        ret = (Retangulo*)malloc(sizeof(Retangulo));
        ret->b = b;
        ret->h = h;

        curr_data = (Lista*)malloc(sizeof(Lista));
        curr_data->tipo = RET;
        curr_data->info = ret;
        curr_data->prox = NULL;
        dados = curr_data;

        dados = calculaDados(dados);
        return dados;
    }

    dados->prox = insereRetangulo(b, h, dados->prox);


    return dados;
}

Lista* insereCirculo(float r, Lista* dados){
```

```
    Circulo* circ;
    Lista* curr_data = NULL;


    if (!(dados)) {

        circ = (Circulo*)malloc(sizeof(Circulo));
        circ->r = r;

        curr_data = (Lista*)malloc(sizeof(Lista));
        curr_data->tipo = CIR;
        curr_data->info = circ;
        curr_data->prox = NULL;
        dados = curr_data;

        dados = calculaDados(dados);
        return dados;
    }

    dados->prox = insereCirculo(r, dados->prox);


    return dados;
}

Lista* insereTriangulo(float b, Lista *dados){
    Triangulo* trig;
    Lista* curr_data = NULL;


    if (!(dados)) {

        trig = (Triangulo*)malloc(sizeof(Triangulo));
        trig->b = b;

        curr_data = (Lista*)malloc(sizeof(Lista));
        curr_data->tipo = TRI;
        curr_data->info = trig;
        curr_data->prox = NULL;
        dados = curr_data;
```

```
        dados = calculaDados(dados);
        return dados;
    }

    dados->prox = insereTriangulo(b, dados->prox);


    return dados;
}

Lista* insereHexagono(float b, Lista* dados){
    Hexagono* hex;
    Lista* curr_data = NULL;


    if (!(dados)) {

        hex = (Hexagono*)malloc(sizeof(Hexagono));
        hex->b = b;

        curr_data = (Lista*)malloc(sizeof(Lista));
        curr_data->tipo = HEX;
        curr_data->info = hex;
        curr_data->prox = NULL;
        dados = curr_data;

        dados = calculaDados(dados);
        return dados;
    }

    dados->prox = insereHexagono(b, dados->prox);


    return dados;
}

Lista* insereQuadrado(float b, Lista *dados){
    Quadrado* quad;
    Lista* curr_data = NULL;
```

```
    if (!(dados)) {

        quad = (Quadrado*)malloc(sizeof(Quadrado));
        quad->b = b;

        curr_data = (Lista*)malloc(sizeof(Lista));
        curr_data->tipo = QUA;
        curr_data->info = quad;
        curr_data->prox = NULL;
        dados = curr_data;

        dados = calculaDados(dados);
        return dados;
    }

    dados->prox = insereQuadrado(b, dados->prox);


    return dados;
}

void imprimeDados(Lista* dado){
    Lista *aux = dado;
    Triangulo *t;
    Quadrado *q;
    Retangulo *r;
    Circulo *c;
    Hexagono *h;

    switch (aux->tipo){
        case QUA:
            q = dado->info;
            printf("Quadrado de área %.2f, com perímetro %.2f e seu lado
%.2f  \n", dado->area, dado->perimetro, q->b);
            break;
        case TRI:
            t = dado->info;
```

```
            printf("Triângulo de área %.2f, com perímetro %.2f e sua base
%.2f  \n", dado->area, dado->perimetro, t->b);
            break;
        case HEX:
            h = dado->info;
            printf("Hexagono de área %.2f, com perímetro %.2f e sua base
%.2f  \n", dado->area, dado->perimetro, h->b);
            break;
        case RET:
            r = dado->info;
            printf("Retangulo de área %.2f, com perímetro %.2f, sua base
%.2f e sua altura %.2f\n", dado->area, dado->perimetro, r->b, r->h);
            break;
        case CIR:
            c = dado->info;
            printf("Circulo de área %.2f, com perímetro %.2f e seu raio
%.2f  \n", dado->area, dado->perimetro , c->r);
            break;
        default:
            printf("Tipo inexistente");
            break;
        }
}

Lista* buscaLista (Lista* dados, int index){
    int count = 0;
    for (Lista *aux = dados; aux != NULL; aux = aux->prox) {
        if (index == count){
            return aux;
        }
        else{
            count++;
        }


    }
    return NULL; /* não achou o elemento */
}

void imprimeFigura (Lista* dados, int tipo){
```

```
    for (Lista *aux = dados; aux != NULL; aux = aux->prox) {
        if (tipo == aux->tipo){
            imprimeDados(aux);
        }
        else if (tipo == ALL){
            imprimeDados(aux);
        }
    }
}

void isEmpty(Lista *dados){
    if (!(dados)){
        printf("Lista vazia");
    }
    else {
        printf("Lista preenchida");
    }
}

Lista *retiraLista(Lista* dados, Lista *retira) {
    Lista * anterior = dados;

    if (dados == retira){
        dados = retira->prox;
    }
    else {
        for(anterior=dados; anterior!=NULL; anterior = anterior->prox) {
            if (anterior->prox == retira){
                break;
            }
        }
        anterior->prox = retira->prox;
    }

    free(retira->info);
    free(retira);

    return dados;

}
```

```c
void freeLista(Lista* dados){
    Lista* aux = dados;
    while (aux != NULL) {
        Lista* next = aux->prox;
        free(aux->info); // liberando memória da forma
        free(aux);  // liberando posição da lista
        aux = next; // resetando auxiliar
    }
}

Lista *criaLista(){
    return NULL;
}
```

Saída apresentada:

Lista inicial:

Triângulo de área 3.90, com perímetro 9.00 e sua base 3.00
Triângulo de área 10.83, com perímetro 15.00 e sua base 5.00
Triângulo de área 6.93, com perímetro 12.00 e sua base 4.00

Primeira Exclusão
Triângulo de área 3.90, com perímetro 9.00 e sua base 3.00
Triângulo de área 6.93, com perímetro 12.00 e sua base 4.00
Segunda Exclusão
Triângulo de área 6.93, com perímetro 12.00 e sua base 4.00
Terceira Exclusão