PROVA DE PROGRAMAÇÃO DE COMPUTADORES I

TERCEIRA UNIDADE (PP3)

CONTEÚDO: REGISTROS E PONTEIROS

Sistemas de Informação Ano 2019 – UEMS

Professora: Mercedes Gonzales Márquez

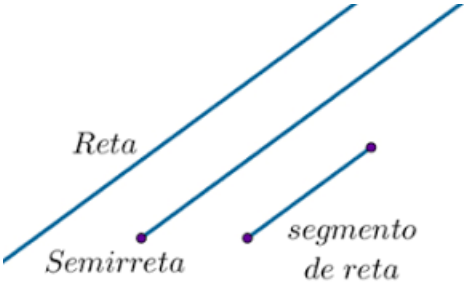
Data de Entrega: Até 19 horas do 23/09/2019.

Enviar no email mercedes@comp.uems.br com Assunto: Prova PP3 de Programação.

Data da Avaliação Oral PO3: 23/09/2019

ESTRUTURAS

1. Um segmento de reta S é uma porção de uma reta que se encontra entre dois pontos. Assim, segmentos de reta são mais naturalmente representados por pares de pontos extremos. Observe na figura abaixo a diferença entre reta, semirreta e segmento de reta.

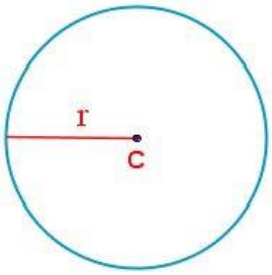


   Assumindo a estrutura Ponto2d vista em aula, a estrutura para o segmento de reta seria a seguinte:

```
typedef struct {
        Ponto2d p1,p2;          /* pontos extremos do segmento de reta */
} segmento;
```

   Faça um programa que permita a leitura dos pontos extremos de n segmentos de reta e imprima o ponto médio de cada segmento de reta.

2. Uma circunferência é formada por um conjunto de pontos que são equidistantes ao centro C desta circunferência. Essa distância é chamada de raio r da circunferência.
   Assim, uma circunferência é naturalmente representada pelo seu centro e raio.



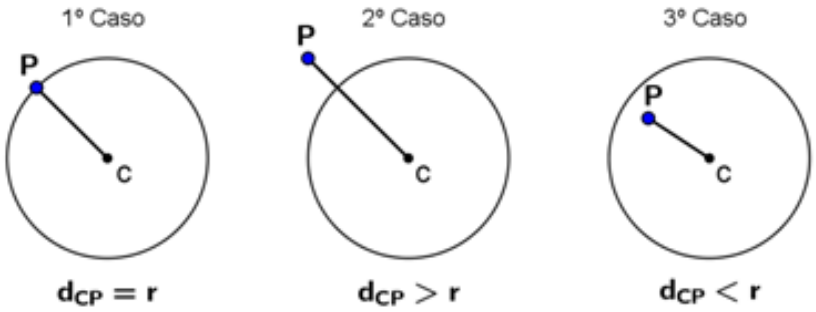
   Assumindo a estrutura Ponto2d vista em aula, a estrutura para uma circunferência seria a seguinte:

```
typedef struct {
      Ponto2d c;     /* centro da circunferência */
      float r;       /* raio da circunferencia */
} circunferencia;
```

Faça um programa que permita a leitura de
(a) raio e o centro de uma circunferência e
(b) n pontos Pi.
e determine se os pontos estão dentro o fora da área delimitada pela circunferência.

Sugestão:

Considere a distancia entre o ponto P=(Px,Py) e o centro C=(Cx,Cy) usando a fórmula da distância: $d_{CP} = \sqrt{(Px - Cx)^2 + (Py - Cy)^2}$. Quando esta distância for maior do que r temos que o ponto está fora da circunferência (2º caso) e quando esta distância for menor ou igual do que o raio, temos que o ponto está dentro da circunferência (casos 1 e 3).



3. *Polígonos* são cadeias fechadas de segmentos de reta que não se cruzam. Em lugar de explicitamente listar os segmentos ou arestas do polígono, nós podemos implicitamente representá-lo listando os n vértices que formam a borda do polígono. Assim um segmento existe entre o i-ésimo e o (i+1)-ésimo pontos na cadeia para 0<=i<=n-1.
   Assumindo a estrutura Ponto2d vista em aula, a estrutura para um polígono seria a seguinte:

```
typedef struct {
      int n;              /* numero de pontos do poligono */
      Ponto2d p[MAXPOLY];     /* array de pontos no polygono */
} poligono;
```

   Considerando a fórmula seguinte que calcula a área de um polígono dados seus n pontos

$$A(P) = \frac{1}{2} \sum_{i=0}^{n-1} (x_i \cdot y_{i+1} - x_{i+1} \cdot y_i)$$

   temos a função que determina tal área A.

```
double area(polygon *p){
   double total = 0.0;             /* área calculada gradativamente  */
   int i, j;                       /* contadores */

   for (i=0; i<p->n; i++) {
```

```
        j = (i+1) % p->n;  /* os índices são considerados mod n para garantir que
                              haja um segmento entre o primeiro e o último ponto */

        total += (p->p[i][X]*p->p[j][Y]) - (p->p[j][X]*p->p[i][Y]);
    }

    return(total / 2.0);
}
```

Faça um programa que, dados os n pontos de m polígonos, determine a área dos m polígonos. O programa deverá imprimir os pontos de cada polígono e sua respectiva área.

PONTEIROS

ALOCAÇÃO DINÂMICA

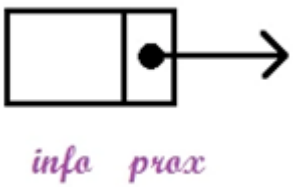
4. Modifique o exercício 1 para que considere a alocação dinâmica de segmentos de reta até que um flag de finalização seja considerado.
5. Modifique o exercício 2 para que considere a alocação dinâmica de circunferências até que um flag de finalização seja considerado.

LISTAS ENCADEADAS

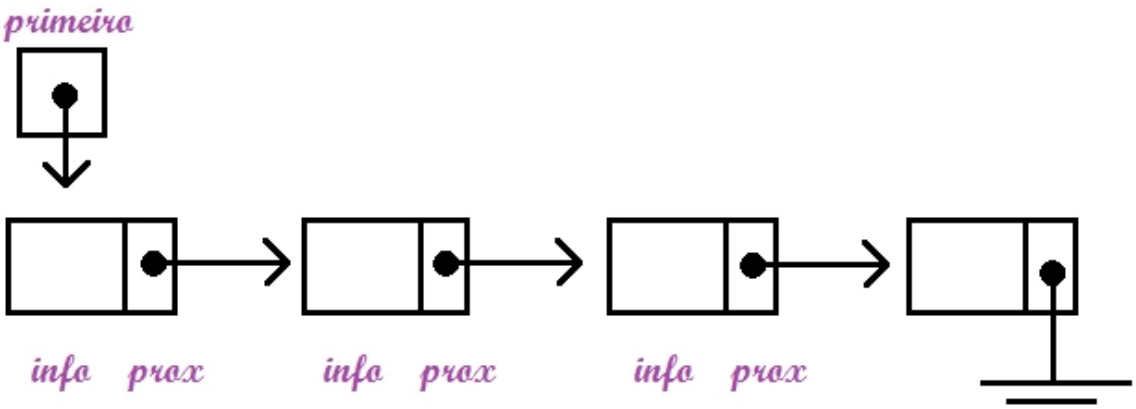
Listas encadeadas são estruturas de dados lineares e dinâmicas, tendo como vantagem, em relação aos vetores, o seu tamanho máximo relativamente infinito (o tamanho máximo é o da memória do computador), e ao mesmo tempo sendo capaz de terem o seu tamanho mínimo de 1 elemento, evitando assim o desperdício de memória.

A sua estrutura consiste numa sequência encadeada de elementos, em geral chamados de nós da lista. A lista é representada por um ponteiro para o primeiro elemento (ou nó). Do primeiro elemento, podemos alcançar o segundo seguindo o encadeamento, e assim por diante. O último elemento da lista aponta para NULL, sinalizando que não existe um próximo elemento.

Nó



Lista encadeada



Vamos considerar um exemplo simples em que queremos armazenar valores inteiros numa lista encadeada. O nó da lista pode ser representado pela estrutura abaixo:

```
struct lista {
   int info;
   struct lista* prox;
};
typedef struct lista Lista;
```

O seguinte programa permite a inserção (push) e a eliminação de elementos de uma pilha (valores inteiros) através da estrutura Lista.

```
#include <stdio.h>
#include <stdlib.h>
struct lista{ // definicao da estrutura Lista, a qual eh uma lista simplesmente encadeada com um valor inteiro
        int info;
        struct lista *prox;
};
typedef struct lista Lista; // estrutura "lista" passa a ser o tipo de dados "Lista"
int tam;

// Prototipos de funcoes e procedimentos
int menu(void);
void inicia(Lista*PILHA);
void opcao(Lista*PILHA, int op);
void exibe(Lista *PILHA);
void libera(Lista *PILHA);
void push(Lista *PILHA);
Lista *pop(Lista *PILHA);

//Programa Principal
int main(void) {
        int opt;
        Lista *PILHA = (Lista *) malloc(sizeof(Lista)); // Alocacao de memoria do primeiro elemento da lista
        if(!PILHA){
                printf("Sem memoria disponivel!\n"); exit(1);
        }else{ // Se a alocacao tiver sucesso entraremos em um menu do qual sairemos so quando a opcao for 0
                inicia(PILHA);
      do{
         opt=menu();
         opcao(PILHA,opt);
      }while(opt);
      free(PILHA);
        }
```

```
}

void inicia(Lista *PILHA) {
	PILHA->prox = NULL; tam=0;
}
int menu(void) {
	int opt;
	printf("Escolha a opcao\n");
	printf("0. Sair\n");
	printf("1. Zerar PILHA\n");
	printf("2. Exibir PILHA\n");
	printf("3. PUSH\n");
	printf("4. POP\n");
	printf("Opcao: ");
	scanf("%d", &opt);
	return opt;
}
void opcao(Lista *PILHA, int op) {
	Lista *tmp;
	switch(op){
		case 0:
			libera(PILHA);
			break;
		case 1:
			libera(PILHA);
	 inicia(PILHA);
			break;
		case 2:
			exibe(PILHA);
			break;
		case 3:
			push(PILHA);
			break;
		case 4:
		    tmp= pop(PILHA);
			if(tmp != NULL) printf("Retirado: %3d\n\n", tmp->info);
			break;
		default:
			printf("Comando invalido\n\n");
	}
}
int vazia(Lista *PILHA) {
	if(PILHA->prox == NULL)
		return 1;
	else return 0;
}

Lista *aloca() {
	Lista *novo=(Lista *) malloc(sizeof(Lista));
	if(!novo){ printf("Sem memoria disponivel!\n"); exit(1); }
	else{
		printf("Novo elemento: "); scanf("%d", &novo->info);
		return novo; }
}
void exibe(Lista *PILHA) {
```

```
        if(vazia(PILHA)){ printf("PILHA vazia!\n\n"); return ; }
        Lista *tmp;
        tmp = PILHA->prox;
        printf("PILHA:");
        while( tmp != NULL){
                printf("%5d", tmp->info);
                tmp = tmp->prox;
        }
        printf("\n ");
        int count;
        for(count=0 ; count < tam ; count++)
      printf(" ^ ");
        printf("\nOrdem:");
        for(count=0 ; count < tam ; count++)
                printf("%5d", count+1);
   printf("\n\n");
}

void libera(Lista *PILHA) {
        if(!vazia(PILHA)){
                Lista *proxNo, *atual;
                atual = PILHA->prox;
                while(atual != NULL){
                        proxNo = atual->prox;
                        free(atual);
                        atual = proxNo;
                }
        }
}
void push(Lista *PILHA) {
        Lista *novo=aloca();
        novo->prox = NULL;
        if(vazia(PILHA)) PILHA->prox=novo;
        else{ Lista *tmp = PILHA->prox;
                while(tmp->prox != NULL)
                        tmp = tmp->prox;
                tmp->prox = novo;
        }
        tam++;
}
Lista *pop(Lista *PILHA) {
        if(PILHA->prox == NULL){ printf("PILHA ja vazia\n\n");
                return NULL; }
        else{
                Lista *ultimo = PILHA->prox, *penultimo = PILHA;
                while(ultimo->prox != NULL){
                        penultimo = ultimo;
                        ultimo = ultimo->prox;
                }
                penultimo->prox = NULL; tam--;
      return ultimo;
        }
}
```

6. A informação armazenada na lista acima não precisa ser necessariamente um dado simples. Consideremos, por exemplo, a construção de uma lista para armazenar um conjunto de circunferências. Cada circunferência é definida pelo seu centro c e o seu raio r. Assim, a estrutura do nó pode ser dada por:

```
struct lista {
   Ponto2d c;
   Ponto2d r;
  struct lista *prox;
};
typedef struct lista Lista;
```

Modifique o programa anterior para que considere

(1) a inserção e a eliminação de elementos geométricos (circunferências)

(2) uma função que forneça como valor de retorno a maior área ($a= pi*r^2$) entre os elementos da lista.

(3) Um procedimento que determine em quais circunferências da lista um ponto P está contido.