

LISTA 2 DA TERCEIRA UNIDADE (L32)

CONTEÚDO: PONTEIROS

Sistemas de Informação Ano 2022 – UEMS

Professora: Mercedes Gonzales Márquez

1. Explique o significado de cada ocorrência de * no fragmento de código a seguir e indique qual a saída exibida na tela.

```
int *p, x=5;
*p *= 2**p;
printf("%d", x);
```

2. Analise o seguinte programa e sua saída.

```
#include "stdio.h"
void main(void){
    int x = 5 , y = 6;
    int *px,*py;
    px = &x;
    py=&y;
    if(px<py)
        printf("py-px = %u\n",py-px);
    else
        printf("px-py = %u\n",px-py);
    printf("px = %u, *px = %d, &px = %u\n",px,*px,&px);
    printf("py = %u, *py = %d, &py = %u\n",py,*py,&py);
    px++;
    printf("px = %u, *px = %d, &px = %u\n",px,*px,&px);
    py=px+3;
    printf("py = %u, *py = %d, &py = %u\n",py,*py,&py);
    printf("py-px = %u\n",py-px);
}
```

3. Teste o programa a seguir para um nome da lista e para outro que não esteja na lista.

```
#include "stdio.h"
#include "string.h"
#define MAX 5
void main(void){
    int i, existe=0;
    char nome[40], *list[MAX]={"Maria","Gesiel","Lucas","Poliana","Juliana"};
    printf("Digite seu nome : ");
    scanf ("%s", nome);
    for(i=0;i<MAX;i++)
        if(strcmp(list[i],nome) == 0)
            existe = 1;
    if(existe)
        printf("Voce esta na lista, tem permissao para entrar !");
    else
        printf("Vc nao faz parte da lista ! ");
}
```

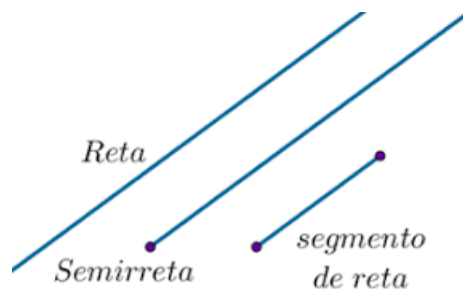
4. Mostre em uma tabela todos os passos (teste de mesa) e identifique qual será a saída do programa para três diferentes pares de valores lidos (x e y).

```
void teste(int *px, int *py) {
    px = py;
    *py = (*py) * (*px);
    *px = *px + 2;
}
int main(void) {
    int x, y;
    scanf("%d",&x);
    scanf("%d",&y);
    teste(&x,&y);
    printf("x = %d, y = %d", x, y);
}
```

5. Escreva uma função `length(s)` que receba como parâmetro uma string `s` e retorne seu tamanho (equivalente a função `strlen(s)` da biblioteca `string.h`).
6. Escreva uma função `copy(s,t)` que receba como parâmetro duas strings e copie o conteúdo da string `t` na string `s` (equivalente a função `strcpy(s,t)` da biblioteca `string.h`).
7. Escreva uma função `compare(s,t)` que receba como parâmetro duas strings e compare `s` e `t`, retornando um valor negativo, zero ou positivo se `s` for, respectivamente, lexicograficamente menor, igual ou maior que `t` (equivalente a função `strcmp(s,t)` da biblioteca `string.h`).
8. Escreva uma função `concatenate(s,t)` que receba como parâmetro duas strings e concatene `t` em `s` (equivalente a função `strcat(s,t)` da biblioteca `string.h`).
9. Suponha que você precise de uma variável para armazenar `n` strings lidas do teclado. É melhor declará-la como uma matriz de caracteres ou como um vetor de ponteiros? Justifique sua resposta.

ALOCAÇÃO DINÂMICA

10. 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.

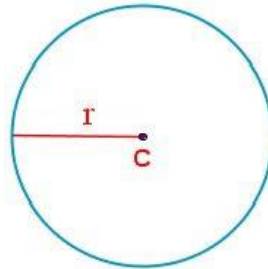


Assumindo a estrutura `Ponto2d` vista em aula, a estrutura para o segmento de reta seria a seguinte:

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

Faça um programa que permita a leitura dos pontos extremos de um número indeterminado de segmentos de reta e imprima o ponto médio de cada segmento de reta. Considere a alocação dinâmica de segmentos de reta até que um flag de finalização seja considerado.

11. 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.



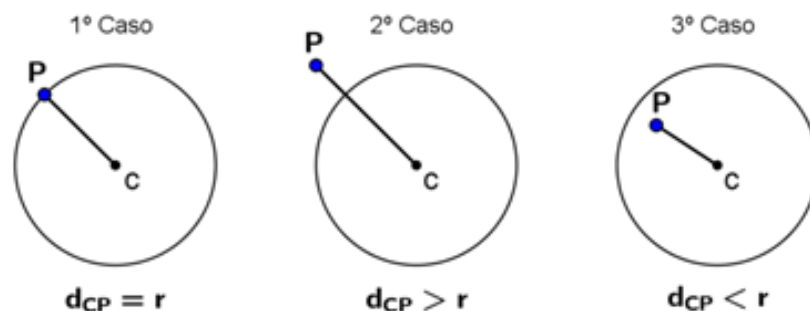
Assumindo a estrutura Ponto2d vista em aula, a estrutura para uma circunferência seria a seguinte:

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

Faça um programa que permita a leitura de
 (a) raio e o centro de uma circunferência e
 (b) um número indeterminado de pontos P_i ,
 e determine se os pontos estão dentro ou fora da área delimitada pela circunferência. Considere a alocação dinâmica de circunferências até que um flag de finalização seja considerado.

Sugestão:

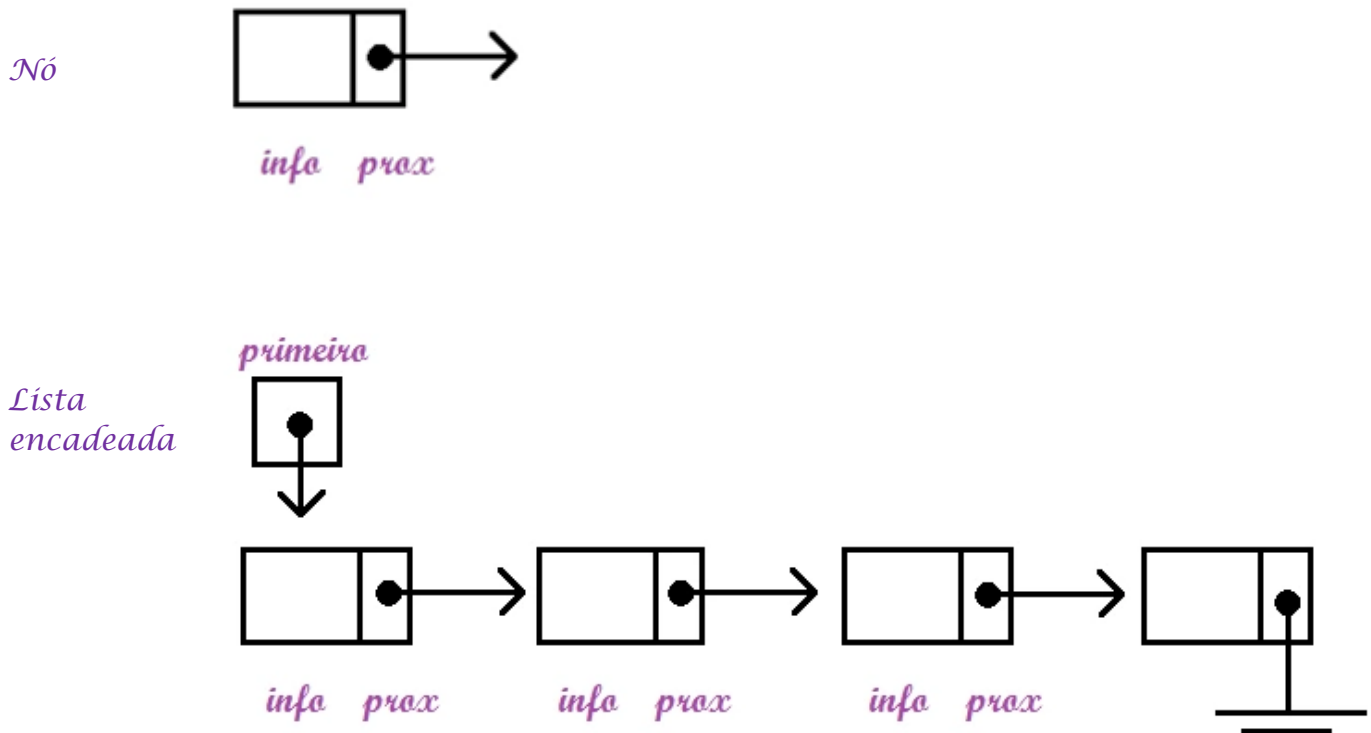
Considere a distância entre o ponto $P=(P_x,P_y)$ e o centro $C=(C_x,C_y)$ usando a fórmula da distância: $d_{CP} = \sqrt{(P_x - C_x)^2 + (P_y - C_y)^2}$. Quando esta distância for maior do que r temos que o ponto está fora da circunferência (2º caso) e quando esta distância for menor ou igual do que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



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.



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 exhibe(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 alocao 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:
        exhibe(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 exhibe(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;
}
}

```

12. 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;
    float 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 ($a = \pi \cdot r^2$) entre os elementos da lista.
- (3) Um procedimento que determine em quais circunferências da lista um ponto P está contido.