

Lab 2 – Ponteiros

Ricardo e Myriam

Conceitos básicos:

1. Um ponteiro é um *endereço* para uma variável;
2. Assim como uma variável precisa ter seu tipo (int, char, float) declarado, uma variável que armazena um ponteiro também: **int *** (ponteiro para uma variável do tipo int); **char *** (ponteiro para uma variável do tipo char), **float *** (ponteiro para uma variável do tipo float);
3. Ponteiros não podem ser criados "do nada", ou seja, devem ser inicializados a partir do *endereço* de uma variável já declarada, através do **operador &**.
Exemplo: int c, *p; p=&c; (a variável p, que é do tipo ponteiro para int, recebe o ponteiro da variável int c);
4. O conteúdo de um ponteiro, ou seja, o valor da variável para a qual ele aponta é obtido pelo **operador ***.
Exemplo: printf("%d", *p); (se p aponta para a variável c, conforme o exemplo anterior, então será impresso na tela o valor da variável c).
5. Ponteiros podem ser incrementados ou decrementados através dos **operadores ++ e --**, respectivamente. A operação de incremento (++) move o ponteiro para o próximo elemento de um vetor (e analogamente para --).
Exemplo: float v[10], *p; p=&v[0]; p++; printf("%d", *p); (imprime o valor do segundo elemento do vetor v, ou seja, v[1]);

1. Deseja-se criar uma função para contar o número de vezes que ela é chamada pelo programa principal. Por exemplo, poderia ser algo como:

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

void incrementa(int counter);

int main(int argc, char *argv[])
{
    int n, c=0, i;
    printf("Forneca o numero de vezes que o laco deve ser executado:");
    scanf("%d", &n);
    for(i=0;i<n;i++)
        incrementa(c);
    printf("O laco executou %d vezes e a funcao foi chamada %d vezes\n", n, c);
    system("PAUSE");
    return 0;
}

void incrementa(int counter)
{
    counter +=1;
}
```

Explique porque esta implementação não funciona como o desejado. Refaça o código:

- a) Usando uma variável indexada como argumento: void incrementa(int counter[]);
- b) Usando um ponteiro como argumento: void incrementa(int *counter);

2. Implemente em C uma função de nome **ordena** que ordena em ordem crescente um conjunto de notas. O número de notas é passado à função no seu primeiro argumento (**int n**) e o conjunto de notas no segundo argumento (**float *nota**). Após o retorno da função, a ordenação original do vetor de notas será perdida. A função não possui VALOR DE RETORNO.

Use o seguinte protótipo para a função:

void ordena(int n, float *nota);

Use apenas ponteiros para acessar o vetor de notas. Implemente o programa principal para testar a função, onde as notas serão fornecidas pelo usuário a partir do teclado e as notas ordenadas serão impressas na tela.

3. Implemente em C uma função de nome **ordena_lista** que ordena uma lista de nomes. O número de nomes a serem ordenados é passado à função no seu primeiro argumento (**int n**) e a lista de nomes no segundo argumento (**char *lista[]**). Após o retorno da função, a ordenação original dos nomes na lista será perdida. A função não possui VALOR DE RETORNO.

Use o seguinte protótipo para a função:

void ordena_lista(int n, char *lista[]);

Implemente o programa principal para testar a função, onde os nomes serão fornecidos pelo usuário a partir do teclado e a lista ordenada será impressa na tela.