**Revisão Estrutura de Dados 1**

1. **Ponteiros:**

Ponteiros são variáveis utilizadas para armazenar um endereço de memória de outra variável.

Exemplo:

#include <stdio.h>

int main()

{

int a=10;

int\* p; 🡪 Inicialização.

p=&a; 🡪 utilizamos & pois ele nos dá acesso ao endereço da memória do seu operando.

printf("%d\n",a);

\*p=5;

printf("%d",a);

return 0;

}

Quando modificamos apenas P, modificamos o endereço de memória do operando.

Quando modificamos \*P, modificamos o valor do operando, no caso A.

int A, B, S;

int\* ptA

int\* ptB;

int \* ptS;

A = 2;

B = 3;

ptA = &A;

ptB = &B; (Colocando endereço nos ponteiros)

ptS = &S;

\*ptS = \*ptA + \*ptB; (acessando os operandos e somando)

printf("Resultado = %d",S);

1. **Malloc**

int\* e;

e =malloc(sizeof(int));

Malloc serve para alocarmos um ponteiro em um endereço não necessariamente de uma variável.

If(e==NULL){

Printf(“memória cheia”)

Exit(1)

}

Free(e) 🡪 limpa e

Se eu aloco memória para 3 ponteiros (p,q,r) e depois falo que q=p e r=p, o que acontece com a memória que aloquei para q e r

R= Isso é um grave erro, você está perdendo o acesso ao bloco que solicitou e não será liberada automaticamente, assim ocorre um vazamento de memória

Ou seja, antese disso use

Free(q)

Free(r)

1. **Passagem de parâmetros para funções (ponteiros)**

Ocorrerá da seguinte forma:

Na hora da passagem usamos:

NameFunction(&a,&b)

Na hora de receber:

Void NameFunction(int \* a, int \*b)

Porém em vetor, temos da seguinte forma:

Passar:

NameFunction(array, tamanho)

Receber (int\* array, int tamanho)

1. **Ponteiros múltiplos**

A grosso modo ocorre assim:

int x =5;

int\*p1;

p=&x;

int \*\*p2=%p1;

Dessa forma temos p1 com o endereço de x e p2 com o endereço de p1.

Printf(“%d”,\*\*p2) == 5.

1. **Aritmética de ponteiros.**

x=10;

int\*p=&x;

Se colocarmos p++, ele passará para o próximo ponteiro.

Serve pro p—também.