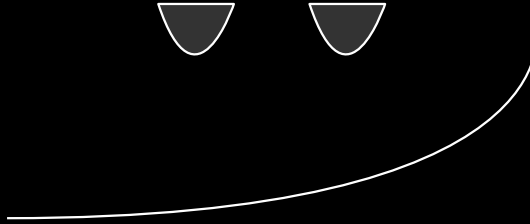


Alocação dinâmica, ponteiros e structs em C



Definição de ponteiro

- Ponteiro é uma variável que armazena o ENDEREÇO de outra variável. Em C, o ponteiro é usado para alocar memória dinamicamente. O ponteiro pode ter diversos tipos, como char*, int*, float* etc.
- A inicialização de um ponteiro se dá atribuindo ao mesmo o ENDEREÇO de uma variável do mesmo TIPO.

```
#include <stdio.h>
#define SUCESSO      (0)

int main(int argc, char** argv) {
    int* p; // Declaração de um ponteiro de tipo 'int'
    int n; // Declaração de um inteiro

    n = 106; // Inicializa 'int n' como 106
    p = &n; // Inicializa 'int* p' com o endereço de 'n'

    printf("%d %d\n", n, *p);
    // Imprime o valor de 'n' e o valor para onde aponta 'p'

    return SUCESSO;
}
```

Saída:

106 106

Lógica

- Valor de n: (106)
- Endereço de n: (00FBFA0C) (endereço de memória genérico)
- Valor para onde aponta p: (106)
- Valor de p: (00FBFA0C)

ATENÇÃO!!

Um ponteiro é dinamicamente ligado ao endereço da variável. Caso o valor apontado pelo endereço mude, o valor ‘*p’ do ponteiro também irá mudar. O ponteiro ‘p’ em si armazena o endereço de uma variável, não o seu valor.

Lógica

Declaração de ponteiro:

```
int* p;
```

Inicialização de um ponteiro:

```
p = &variavel;
```

Acesso ao valor para onde o ponteiro aponta:

```
printf("%d", *p);
```

`tipo*` indica o tipo do valor para onde o ponteiro deve apontar (portanto seu tamanho)

`&variavel` retorna o endereço de `variavel`

`*p` retorna o valor para onde `p` aponta

*Um ponteiro, por convenção, sempre terá tamanho fixo. Ele será de **4 bytes** em arquiteturas de 32-bits e **8 bytes** em arquiteturas de 64-bits, computadores modernos.*

Definição de alocação dinâmica

- Alocação dinâmica se dá por alocar a memória durante a execução do programa;
- Se você não sabe quanta memória usar, é sempre um bom momento para aplicá-la.

Para usar alocação dinâmica, você necessitará das funções:

<code>malloc(tamanho)</code>	Aloca memória
<code>calloc(nElementos, tamanho)</code>	Aloca memória e inicializa cada elemento com 0
<code>realloc(ponteiro, tamanho)</code>	Realoca memória já alocada ('tamanho' deve aumentar sempre)
<code>free(ponteiro)</code>	Libera memória alocada de um ponteiro

Para acessá-las, inclua `<stdlib.h>`

```
#include <stdlib.h>
#define SUCESSO          (0)

int main(int argc, char** argv) {
    int estatico[50];
    int* dinamico = malloc(sizeof(int) * 50);           // 50x tamanho de int
    dinamico = realloc(dinamico, sizeof(int) * 100);    // 50x tamanho de int
    free(dinamico);                                     // 100x tamanho de int
    return SUCESSO;                                     // foi pro saco kk
}
// podes aumentar 'dinamico' e descartá-lo quando não precisar mais
// estatico = paia
```

Problema

- Você terá de salvar n números, quantos der na telha do cliente. O que fazer?

```
#include <stdio.h>
#define SUCESSO      (0)

int main(int argc, char** argv) {
    int i = 0, vetor_estatico[250]; // desperdício incompetente de 1KB!

    do {
        printf("\nPreencha o registro [%d] de seu vetor: ", i);
        scanf("%d", &vetor_estatico[i]);
        i++;
    } while (vetor_estatico[i-1] != 0); // interromper se receber '0'

    return SUCESSO;
}
```

PÉSSIMA PRÁTICA. E se for necessário mais que 100 números inteiros? E se for necessário menos? Nesse caso, pode-se julgar necessário usar alocação dinâmica.

Para te ajudar a distinguir, use constantes quando for usar alocação estática (ex. `#define TAMANHO_VETOR 100`). Se você precisar de muitas constantes, talvez seja hora de tentar alocação dinâmica!

Solução

```
#include <stdio.h>
#define SUCESSO      (0)

int main(int argc, char** argv) {
    int i = 0, *vetor_dinamico = NULL; // inicializa ponteiro com nada

    do {
        vetor_dinamico = realloc(vetor_dinamico, (sizeof(int) * i));
        // realoca a: número de cadastros x int

        printf("\nPreencha o registro [%d] de seu vetor: ", i);
        scanf("%d", &vetor_dinamico[i]);
        i++;
    } while (vetor_dinamico[i - 1] != 0); // interromper se receber '0'

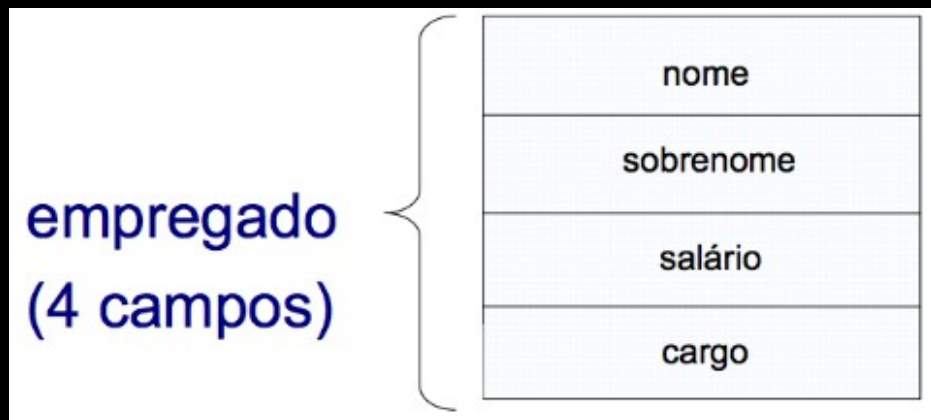
    return SUCESSO;
}
```

Agora sim. Tamanho bem justo, nada mais nada menos.

Como assim vetor? Não era ponteiro? Vetores são ponteiros, que apontam para o endereço onde o vetor começa.

Definição de struct

- Struct é um vetor de variáveis. Cada variável na struct é chamada de *membro* e pode ser de um tipo distinto.



- Protótipo para declarar uma struct com n campos:

```
1 struct nome_estrutura {  
2     tipo1 identificador1;  
3     tipo2 identificador2;  
4     ...  
5     tipon identificadorn;  
6 };
```

Créditos pelas imagens: João Eduardo Montandon e Vírginia Fernandes Mota

Definição de struct

- A declaração de um TIPO de struct deve ser feita **fora** do main() ou qualquer função.
- A declaração de uma VARIÁVEL de tipo struct é feita **dentro** do programa.
- Os membros de uma struct são acessados por meio do ponto (.)
- É usado typedef para facilitar a criação de variáveis.

```
01 #include <stdio.h>
02 #define SUCESSO      (0)
03
04 struct registro_clinico_s {
05     int id_do_paciente;
06     char tipo_sanguineo;
07 };
08
09 int main(int argc, char** argv) {
10     struct registro_clinico_s hospital_local; // Declaração de uma variável da struct
11     hospital_local.id_do_paciente = 1;        // Inicialização
12     hospital_local.tipo_sanguineo = 'A';      // dos valores
13     printf("\n%d:%c", hospital_local.id_do_paciente, hospital_local.tipo_sanguineo);
14     return SUCESSO;
15 }
```

```
typedef struct registro_clinico_s {
    int id_do_paciente;
    char tipo_sanguineo;
} hospital_t;
```

```
hospital_t hospital_local;

// Código com typedef
```

* Caso queira testar este código, muita atenção para não colar as linhas 4-7 e 10 uma sobre a outra e causar conflitos.

Exercícios

1. Faça um programa para leitura, via teclado, dos dados de um aluno. Os dados a serem guardados na estrutura aluno são os seguintes: char nome[], char curso[], int idade. Ao final, imprima estas informações na tela. (STRUCTS)
2. Altere o programa do exercício 2 para que ele leia as informações de N alunos. Imprima a média de cada aluno e a média geral da turma. (STRUCTS) + (PONTEIROS / ALOCAÇÃO DINÂMICA)
3. Faça um programa que leia um vetor de um tamanho escolhido pelo usuário e calcule a média aritmética de seus valores. (PONTEIROS / ALOCAÇÃO DINÂMICA)