

# Estruturas de Dados

Revisão sobre registros e alocação  
dinâmica de memória

# Estruturas Homogêneas

- Na **Linguagem C**, quando desejamos trabalhar com uma **coleção de dados** de mesmo tipo utilizamos vetores e matrizes.
- Como definir e como utilizar um vetor/matriz???

# Estruturas Heterogêneas (struct)

- Na **Linguagem C**, quando desejamos trabalhar com uma **coleção de dados** que podem ser **de diferentes tipos**, é comum utilizarmos **struct**.

# Exemplo de STRUCT

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

typedef struct TipoAluno {
    int matricula;
    char nome[40];
    float coeficiente;
} TAluno;

int main() {

    TAluno aluno;

    printf("\n\nPreenchimento de Dados de Aluno:\n\n");
    printf("\tMATRICULA: ");
    scanf("%d", &aluno.matricula);

    printf("\n\n\tNOME: ");
    fflush(stdin);
    gets(aluno.nome);
```

# Exemplo de STRUCT

- Um **struct** foi definido como uma estrutura de três campos (**matricula**, **nome** e **coeficiente**) que são respectivamente de três diferentes tipos (**integer**, **string** e **float**).

```
typedef struct TipoAluno {  
    int matricula;  
    char nome[40];  
    float coeficiente;  
} TAluno;
```

# Exemplo de STRUCT

- Uma variável de memória de nome “**aluno**” foi declarada como sendo do tipo **TAluno**.

```
typedef struct TipoAluno {  
    int matricula;  
    char nome[40];  
    float coeficiente;  
} TAluno;
```

```
TAluno aluno;
```

# Exemplo de STRUCT

- Eis um esquema de representação de nosso struct TipoAluno (TAluno):

TAluno		
<u>matricula</u>	<u>nome</u>	<u>coeficiente</u>
1090	José da Silva	89.27

# Exemplo de STRUCT

- Note que **TAluno** é um apelido do **struct TipoAluno**.
  - Para declarar uma variável desse tipo podemos usar o **TAluno** ou **struct TipoAluno**
- Para referenciar o **campo nome** dessa **estrutura** é necessário indicar o nome da variável, seguido de ponto e do nome do campo (exemplo: **aluno.nome**).



# Um Novo Exemplo de STRUCT

- Vamos alterar o programa utilizado no exemplo anterior, definindo um vetor da nossa struct

```
#include <stdlib.h>

#define tamanho 4

typedef struct TipoAluno {
    int matricula;
    char nome[40];
    float coeficiente;
} TAluno;

TAluno turma[tamanho];
```

# Novo Exemplo de STRUCT

- Observe que agora foi declarado um **vetor** de **TAluno** de **04** (quatro) **posições**, com o nome **turma**
- Significa que podemos armazenar os dados de até quatro diferentes alunos nesse **vetor**.
- Serão 04 matrículas, 04 nomes e 04 coeficientes.

# Esquema do Vetor TURMA

<u>turma</u>		
Posição	Conteúdo	
0	TAluno	
	matricula	nome                      coeficiente
	1090	José da Silva                      89.27
1	TAluno	
	matricula	nome                      coeficiente
	1099	Marina Ximene                      81.00
2	TAluno	
	matricula	nome                      coeficiente
	1200	Paula Castro                      78.20
3	TAluno	
	matricula	nome                      coeficiente

# Novo Exemplo de STRUCT

- Então um laço de repetição (for) é criado para a alimentação dos dados no **vetor**.

```
printf("\n\nPreenchimento de Dados dos Alunos da TURMA:\n\n");

for (posicao = 0; posicao < tamanho; posicao++){

    printf("\tPOSICAO do VETOR: %d\n\tMATRICULA: ", posicao);
    scanf("%d", &turma[posicao].matricula);

    printf("\n\n\tNOME: ");
    fflush(stdin);
    gets(turma[posicao].nome);

    printf("\n\n\tCOEFICIENTE: ");
    scanf("%f", &turma[posicao].coeficiente);
} //for
```

# Novo Exemplo de STRUCT

- Note que agora, para referenciar um campo específico também deve-se indicar sua posição no vetor: nome do vetor seguido da posição (entre colchetes), um ponto e o nome do campo.
- Exemplo:  
`turma[2].matricula = 357;`

# Novo Exemplo de STRUCT

- De maneira análoga, para exibir os dados armazenados no vetor ...

```
        .....  
    }//for  
  
    printf("\n\n\n\tVoce Informou os seguintes DADOS de ALUNO:\n\n");  
  
    for(posicao = 0; posicao < tamanho; posicao++){  
        printf("\t\tPOSICAO [%d] - MATRICULA: %d - NOME: ", posicao, turma[posicao].matricula);  
        printf("%s - COEF: %.2f\n", turma[posicao].nome, turma[posicao].coeficiente);  
    }//for  
    printf("\n\n");  
    system("PAUSE");  
}
```

# Alocação de Memória

- Do ponto de vista do programador, a **alocação de memória** para executar um programa pode ser realizada de duas formas: **estática** ou **dinâmica**.
- Na **alocação de memória estática**, o espaço de memória ocupado pelas **variáveis é determinado no momento da compilação**.

# Alocação de Memória

- Se o **espaço de memória** é determinado **durante a execução do programa**, a **alocação de memória** é realizada de forma **dinâmica**.
- É nesse caso que surgem as chamadas **variáveis anônimas**, pois não conhecemos o seu nome.
- Sabemos apenas seu **endereço de memória**.



# Alocação de Memória

- Exemplo:

```
#include <stdlib.h>

int main( ) {
    int *ptr, i;

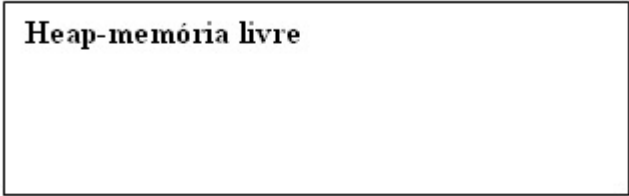
    ptr = (int *)malloc(sizeof(int));
    *ptr = 5800;
    i = 4200;
}
```

# Alocação de Memória

- Suponhamos que o tamanho do tipo inteiro é de 2 bytes.
- Inicialmente o conteúdo da variável **i** é indefinido e, posteriormente, armazena o valor de 4.200.
- A variável ponteiro **ptr** posteriormente armazena o valor do ponteiro do endereço (19.000) da área de heap que contém o valor de 5.800.
- Lembre que esses endereços são fictícios.

# Alocação de Memória

- Antes da execução de nosso programa exemplo, temos a seguinte situação:



Heap-memória livre

A rectangular box with a black border, representing a memory segment. The text "Heap-memória livre" is written in the top-left corner of the box.

# Alocação de Memória

- Então o programa exemplo é executado, sendo que uma área da memória é reservada para uso por esse programa.
- Nessa área reservada da memória principal são alocados espaços de memória para as variáveis declaradas (**ptr** e **i**).
- A variável de memória **i** é do tipo primitivo int (integer = inteiro).

# Alocação de Memória

- A variável de memória **ptr** foi declarada como um ponteiro para um inteiro.

```
int main( ) {  
    int *ptr, i;
```

Memória Usada pelo Programa	Heap-Memória Livre
10000 i ==	
10002 ptr ==	

# Alocação de Memória

- Em nosso exemplo, o endereço de memória 10.000 é reservado para armazenar o conteúdo da variável de memória **i**.
- Na verdade, **10.000** é o endereço inicial para armazenar o conteúdo de **i**. E termina em 10.001.
- Afinal, assumimos a suposição de que **variáveis do tipo inteiro ocupariam 2 bytes** da memória. Portanto, a **variável i** ocupa a área de 10.000 a 10.001 (inclusive).

# Alocação de Memória

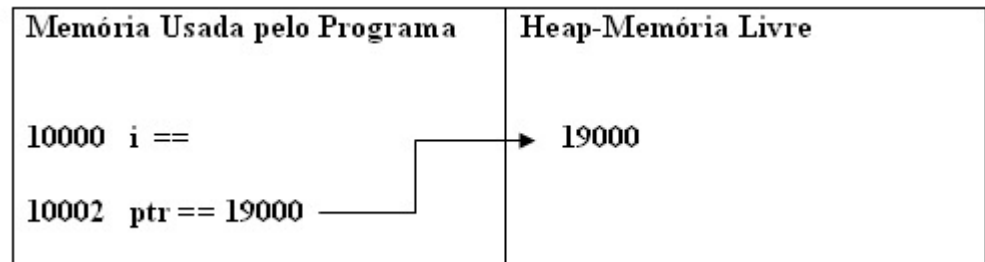
- A área reservada de memória para o conteúdo da variável **ptr** começa no endereço **10.002** (logo após a área reservada para a variável **i**).

Memória Usada pelo Programa	Heap-Memória Livre
10000 i ==	
10002 ptr ==	

# Alocação de Memória

- Até aqui os conteúdos das variáveis de memória **i** e **ptr** são indefinidos (podem conter até mesmo “**sujeira de memória**”).

```
int main( ) {  
    int *ptr, i;  
  
    ptr = (int *)malloc(sizeof(int));
```





# Alocação de Memória

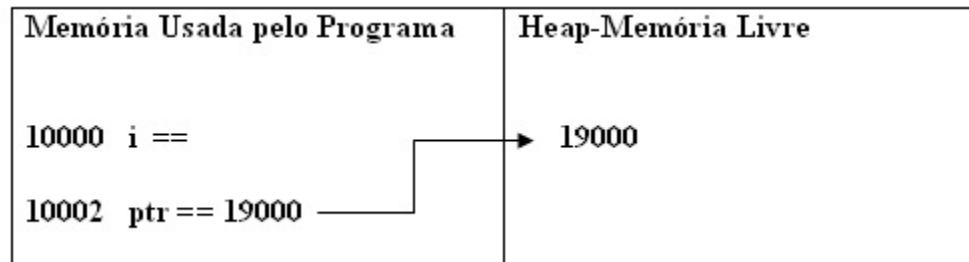
- O comando **malloc** (**memory allocation**) constitui uma solicitação por parte do programa ao sistema operacional.
- **sizeof(int)** – que aparece à direita de **malloc** – indica que o **espaço de memória requisitado** deve ser de **tamanho suficiente para armazenar um valor do tipo inteiro**.

# Alocação de Memória

- A parte **(int \*)** – à esquerda de **malloc** – indica que o **endereço de memória** retornado deve ser entendido como um **ponteiro para um inteiro (int)**.
- No canto mais à esquerda do comando, **ptr =** indica que o endereço retornado deverá ser armazenado na variável de memória **ptr**.

# Alocação de Memória

- Em nosso exemplo o **sistema operacional** disponibiliza o endereço de memória inicial **19.000** (fora da área usada pelo programa) para armazenar um valor inteiro a ser apontado por **ptr**.



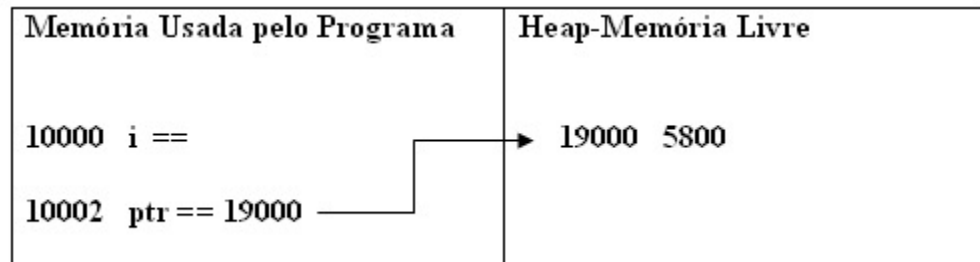
# Alocação de Memória

- Note que a variável **ptr** (nosso ponteiro para um inteiro) encontra-se armazenada a partir do endereço de memória 10.002.
- Seu conteúdo é um **endereço de memória (19.000)** onde deve ser armazenado um **valor inteiro**.

# Alocação de Memória

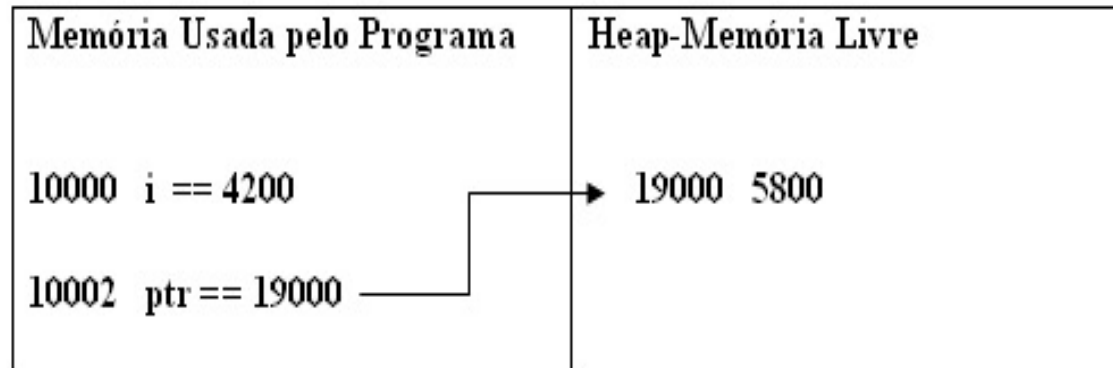
- Na seqüência, nosso **programa** de exemplo armazena o valor 5.800 na posição de memória apontada por **ptr**.

```
*ptr = 5800;
```



# Alocação de Memória

- Observe que **ptr** “aponta para” (armazena o) endereço de memória (**19.000**) de onde está o valor inteiro **5.800**.



# Alocação de Memória

- A **variável de memória i**, diferente de **ptr**, não é um **ponteiro** para inteiro, mas uma **variável de tipo inteiro**.
- Então, o conteúdo **4.200** é armazenado diretamente no endereço de memória **10.000** (**reservado estaticamente** para a variável **i**).
- A variável **ptr** faz uso da **alocação dinâmica**.

# Alocação de Memória

- Enquanto o \* é utilizado para acessar a posição de memória apontada por um ponteiro, o & é utilizado para obter o endereço de memória de uma variável.



# Alocação de Memória - Exercícios

- Faça um programa que aloque dois ponteiros para inteiros e imprima o endereço de memória e o valor armazenado nos mesmos.
- Escreva um procedimento que receba duas variáveis inteiras e inverta seus valores (coloque o valor da primeira na segunda e o valor da segunda na primeira)

# Erros Frequentes

# Alocação de Ponteiros

**Erro Frequente: não alocar o ponteiro.**

```
int main(){  
    int *p;  
    p=516;  
    *p=32;  
}
```

O código acima compila? Caso compile, o que acontecerá se eu rodá-lo? Por que?

# Alocação de Ponteiros

## Erro Frequente: alocar ponteiro sem necessidade

```
int main(){  
    int *p = (int *) malloc (sizeof(int));  
    int *q = (int *) malloc (sizeof(int));  
    *p=516;  
    q=p;  
    *q=32;  
    printf("%d %d", *p, *q);  
}
```

# Alocação de Ponteiros

```
int main(){  
    int *p = (int *) malloc (sizeof(int));  
    int *q;  
    *p=516;  
    q=p;  
    *q=32;  
    printf("%d %d", *p, *q);  
}
```

**Dica:** Podemos ter vários ponteiros referenciando a mesma posição de memória

# Estruturas de dados

```
typedef struct aluno{  
    int matricula;  
    char nome[30];  
} TAluno;
```

```
main{
```

```
    aluno a;
```

```
    .  
    .  
    .  
}
```

**Erro Frequente: referenciar o nome da struct sem colocar a palavra “struct”**

# Indexação de vetores

**Erro Frequentes: esquecer que strings tem que ter tamanho um caracter maior para guardar o caracter de fim de string.**

```
int main(int argc, const char * argv[]) {  
    char nome[4];  
    strcpy(nome, "Toin");  
    printf("%s", nome);  
}
```

# Outras dicas

- *EVITEM UTILIZAR VARIÁVEIS GLOBAIS*
- *COMENTEM OS CÓDIGOS DOCUMENTANDO O RACIOCÍNIO SEGUIDO.*
- *UTILIZEM BIBLIOTECAS PARA MODULARIZAR O CÓDIGO*