



INSTITUTO FEDERAL DE
EDUCAÇÃO, CIÊNCIA E TECNOLOGIA
SUL-RIO-GRANDENSE

Estrutura de Dados

Ponteiros, structs e alocação dinâmica com listas estáticas

Prof. Silvana Teodoro

silvanateodoro@charqueadas.ifsul.edu.br

Sumário

- Objetivos da aula
- Ponteiros
- Tipos estruturados (Structs)
- Alocação dinâmica
- Exercícios
- Referências



Objetivos

- Fazer uma revisão sobre a utilização dos **ponteiros** e dos tipos estruturados (**structs**), bem como sobre o processo de **alocação dinâmica de memória** dentro da linguagem C, observando sua aplicação sobre listas estáticas.

Endereços e ponteiros?

- Onde fica armazenado o valor da variável?
 - Memória!
- Como é a memória do computador?
 - Um monte de “gavetas”: posição de memória
 - Cada gaveta tem um número único: endereço
- Cada nome de variável → endereço
 - Declarar variável
 - Armazenar valor em variável



Endereços e ponteiros?

- Cada nome de variável → endereço

Endereço	0	1	2	3	4	5	6	7
Nome								
Valor	??	??	??	??	??	??	??	??

- char letra;

Endereços e ponteiros?

- Cada nome de variável → endereço

Endereço	0	1	2	3	4	5	6	7
Nome	letra							
Valor	??	??	??	??	??	??	??	??

- char letra;
- int idade;

Endereços e ponteiros?

- Cada nome de variável → endereço

Endereço	0	1	2	3	4	5	6	7
Nome	letra	idade						
Valor	??	??	??	??	??	??	??	??

- char letra;
- int idade;
- char a[3];

Endereços e ponteiros?

- Cada nome de variável → endereço

Endereço	0	1	2	3	4	5	6	7
Nome	letra	idade				a		
Valor	??	??	??	??	??	??	??	??

- char letra;
- int idade;
- char a[3];
- letra = 'c';

Endereços e ponteiros?

- Cada nome de variável → endereço

Endereço	0	1	2	3	4	5	6	7
Nome	letra	idade				a		
Valor	99	??	??	??	??	??	??	??

- char letra;
- int idade;
- char a[3];
- letra = 'c';
- a[1] = 'd';



Endereços e ponteiros?

- Cada nome de variável → endereço

Endereço	0	1	2	3	4	5	6	7
Nome	letra	idade				a		
Valor	99	??	??	??	??	??	100	??

- char letra;
- int idade;
- char a[3];
- letra = 'c';
- a[1] = 'd';
- idade = 256;



Endereços e ponteiros?

- Cada nome de variável → endereço

Endereço	0	1	2	3	4	5	6	7
Nome	letra	idade				a		
Valor	99	0	1	0	0	??	100	??

- char letra;
- int idade;
- char a[3];
- letra = 'c';
- a[1] = 'd';
- idade = 256;

Como saber o endereço de uma variável?

Usar o prefixo &



Endereços e ponteiros?

- Ponteiro: serve para armazenar um endereço
 - Indicado por um * na frente do nome da variável na sua declaração e deve ser do mesmo tipo do dado referenciado.

```
int x = 10;
int *p;
p=&x;
printf("x = %d\n",x); // x = 10
printf("&x = %d\n",&x); // &x=2293572
printf("p = %d\n",p); // p=2293572
```

- E se quisermos saber o valor da variável “apontada”?
 - Usamos o * na frente do nome do ponteiro no decorrer de nosso código.



Endereços e ponteiros?

- Exemplo: lendo o valor apontado

```
int x = 10;
int *p;
p=&x;
printf("x = %d\n",x); // x = 10
printf("&x = %d\n",&x); // &x=2293572
printf("p = %d\n",p); // p=2293572
printf("*p = %d\n",*p); // *p=10
```

Tipos estruturados (Structs)

- Motivação:
 - manipulação de dados compostos ou estruturados
 - Exemplos:
 - ponto no espaço bidimensional
 - representado por duas coordenadas (x e y), mas tratado como um único objeto (ou tipo)
 - dados associados a aluno:
 - aluno representado pelo seu nome, número de matrícula, endereço, etc., estruturados em um único objeto (ou tipo)

Ponto

X
Y

Aluno

Nome	
Matr	
End	Rua
	No
	Compl



Tipos estruturados (Structs)

- Tipo estrutura:
 - tipo de dado com campos compostos de tipos mais simples
 - elementos acessados através do operador de acesso “ponto” (.)

```
struct ponto                /* declara ponto do tipo struct */
{ float x;
  float y;
};
...
struct ponto p;             /* declara p como variável do tipo struct ponto */
...
p.x = 10.0;                 /* acessa os elementos de ponto */
p.y = 5.0;
```



Tipos estruturados (Structs)

```
/* Captura e imprime as coordenadas de um ponto qualquer */
#include <stdio.h>
struct ponto {
    float x;
    float y;
};
int main (void)
{
    struct ponto p;
    printf("Digite as coordenadas do ponto(x y): ");
    scanf("%f %f", &p.x, &p.y);
    printf("O ponto fornecido foi: (%.2f,%.2f)\n", p.x, p.y);
    return 0;
}
```

Basta escrever `&p.x` em lugar de `&(p.x)`.

O operador de acesso ao campo da estrutura tem precedência sobre o operador "endereço de"



Tipos estruturados (Structs)

- Ponteiros para estruturas:
 - acesso ao valor de um campo x de uma variável estrutura p: `p.x`
 - acesso ao *valor* de um campo x de uma variável ponteiro pp: `pp->x`
 - acesso ao *endereço* do campo x de uma variável ponteiro pp: `&pp->x`

```
struct ponto *pp;  
  
(*pp).x = 12.0;      /* formas equivalentes de acessar o valor de um campo x */  
pp->x = 12.0;
```



Tipos estruturados (Structs)

- Passagem de estruturas para funções – por valor:
 - análoga à passagem de variáveis simples
 - função recebe toda a estrutura como parâmetro:
 - função acessa a cópia da estrutura na pilha
 - função não altera os valores dos campos da estrutura original
 - operação pode ser custosa se a estrutura for muito grande

```
/* função que imprima as coordenadas do ponto */  
void imprime (struct ponto p)  
{  
    printf("O ponto fornecido foi: (%.2f,%.2f)\n", p.x, p.y);  
}
```



Tipos estruturados (Structs)

- Passagem de estruturas para funções – por referência:
 - apenas o ponteiro da estrutura é passado, mesmo que não seja necessário alterar os valores dos campos dentro da função

```
/* função que imprima as coordenadas do ponto */  
void imprime (struct ponto* pp)  
{ printf("O ponto fornecido foi: (%.2f,%.2f)\n", pp->x, pp->y); }  
  
void captura (struct ponto* pp)  
{ printf("Digite as coordenadas do ponto(x y): ");  
  scanf("%f %f", &p->x, &p->y);  
}  
  
int main (void)  
{ struct ponto p; captura(&p); imprime(&p); return 0; }
```



Definição de novos tipos (typedef)

- `typedef`
 - permite criar nomes de tipos
 - útil para abreviar nomes de tipos e para tratar tipos complexos
 - Exemplo: definição de nomes de tipos para as estruturas

```
struct ponto {  
    float x;  
    float y;  
};  
  
typedef struct ponto Ponto;  
typedef struct ponto *PPonto;
```

- `ponto` representa uma estrutura com 2 campos do tipo `float`
- `Ponto` representa a estrutura `ponto`
- `PPonto` representa o tipo ponteiro para a estrutura `Ponto`



Definição de novos tipos (typedef)

- typedef
 - Exemplo: (definição em um comando só)

```
typedef struct ponto {  
    float x;  
    float y;  
} Ponto;
```

- ponto representa uma estrutura com 2 campos do tipo float
- Ponto representa a estrutura ponto



Exemplos (structs e ponteiros)

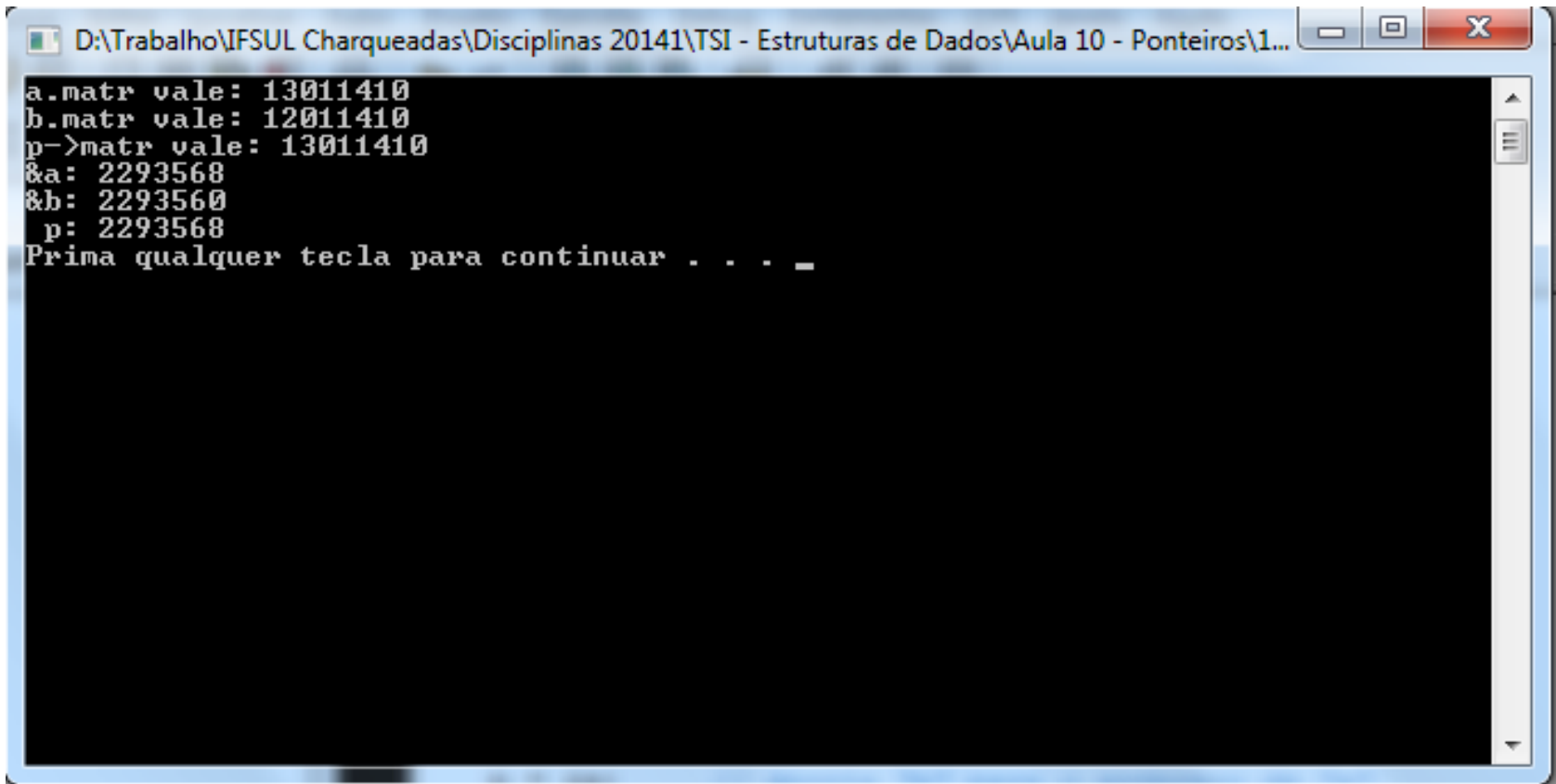
```
struct aluno {
    int matricula;
    float nota;
};

int main(void) {
    aluno a, b; // Declara dois alunos "a" e "b"
    aluno *p; // Declara um ponteiro para um aluno

    a.matricula = 12011410;
    p = &a;      // Aponta "p" para o endereço de "a"
    b = a;       // Copia "a" para "b"
    // A seta -> é alternativo a ter de usar o (*p) !
    // (*ptr).campo é o mesmo que p->campo
    p->matricula = 13011410; // Modifica matricula do aluno apontado por "p"
    p->nota = 9.5;          // Modifica nota do aluno apontado por "p"
    // Observe os valores impressos
    printf("a.matr vale: %d\n", a.matricula);
    printf("b.matr vale: %d\n", b.matricula);
    printf("p->matr vale: %d\n", p->matricula);
    // Observe os endereços
    printf("&a: %d\n", &a);
    printf("&b: %d\n", &b);
    printf(" p: %d\n", p);
    system("PAUSE");
}
```



Exemplos (structs e ponteiros)



```
D:\Trabalho\IFSUL Charqueadas\Disciplinas 20141\TSI - Estruturas de Dados\Aula 10 - Ponteiros\1...
a.matr vale: 13011410
b.matr vale: 12011410
p->matr vale: 13011410
&a: 2293568
&b: 2293560
p: 2293568
Prima qualquer tecla para continuar . . . _
```

Alocação dinâmica

- Uso da memória:
 - uso de variáveis globais (e estáticas):
 - espaço reservado para uma variável global existe enquanto o programa estiver sendo executado
 - uso de variáveis locais:
 - espaço existe apenas enquanto a função que declarou a variável está sendo executada
 - liberado para outros usos quando a execução da função termina
 - variáveis globais ou locais podem ser simples ou vetores:
 - para vetor, é necessário informar o número máximo de elementos pois o compilador precisa calcular o espaço a ser reservado



Alocação dinâmica

- Uso da memória:
 - alocação dinâmica:
 - espaço de memória é requisitado em tempo de execução
 - espaço permanece reservado até que seja explicitamente liberado
 - depois de liberado, espaço estará disponibilizado para outros usos e não pode mais ser acessado
 - espaço alocado e não liberado explicitamente, será automaticamente liberado ao final da execução



Alocação dinâmica

- Uso da memória:

- memória estática:

- código do programa
 - variáveis globais
 - variáveis estáticas

- memória dinâmica:

- variáveis alocadas dinamicamente
 - memória livre
 - variáveis locais

memória estática	Código do programa
	Variáveis globais e Variáveis estáticas
memória dinâmica	Variáveis alocadas dinamicamente
	Memória livre
	Variáveis locais (Pilha de execução)

Alocação dinâmica

- Uso da memória:
 - alocação dinâmica de memória:
 - usa a memória livre
 - se o espaço de memória livre for menor que o espaço requisitado, a alocação não é feita e o programa pode prever tratamento de erro
 - pilha de execução:
 - utilizada para alocar memória quando ocorre chamada de função:
 - sistema reserva o espaço para as variáveis locais da função
 - quando a função termina, espaço é liberado (desempilhado)
 - se a pilha tentar crescer mais do que o espaço disponível existente, programa é abortado com erro

memória estática	Código do programa
	Variáveis globais e Variáveis estáticas
memória dinâmica	Variáveis alocadas dinamicamente
	Memória livre
	Variáveis locais (Pilha de execução)



Alocação dinâmica

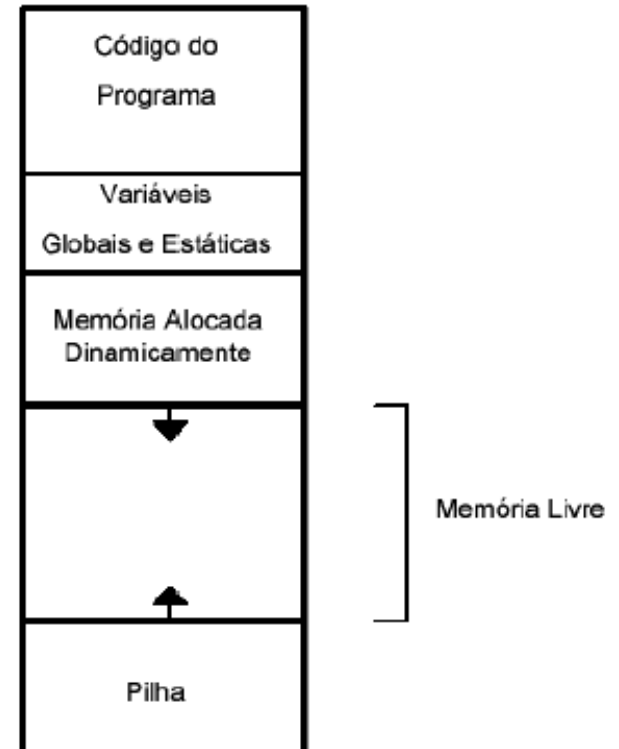
- Funções da biblioteca padrão “stdlib.h”
 - contém uma série de funções pré-definidas:
 - funções para tratar alocação dinâmica de memória
 - constantes pré-definidas
 -



Alocação dinâmica

```
void * malloc(int num_bytes) ;
```

```
void free(void * p) ;
```



Alocação dinâmica

- Função “**malloc**”:
 - recebe como parâmetro o número de bytes que se deseja alocar
 - retorna um ponteiro genérico para o endereço inicial da área de memória alocada, se houver espaço livre:
 - ponteiro genérico é representado por `void*`
 - ponteiro é convertido automaticamente para o tipo apropriado
 - ponteiro pode ser convertido explicitamente
 - retorna um endereço nulo, se não houver espaço livre:
 - representado pelo símbolo `NULL`



Alocação dinâmica

- Função “**sizeof**”:
 - retorna o número de bytes ocupado por um tipo
- Função “**free**”:
 - recebe como parâmetro o ponteiro da memória a ser liberada
 - a função free deve receber um endereço de memória que tenha sido alocado dinamicamente



Alocação dinâmica

- **Teste de tamanho:** execute o código abaixo...

```
typedef struct {  
    int dia, mes, ano;  
} data;  
  
int main( void) {  
    printf( "sizeof (data) = %d\n", sizeof (data));  
    printf( "sizeof (data *) = %d\n", sizeof (data *));  
    return 0;  
}
```



Alocação dinâmica

- Exemplo:
 - alocação dinâmica de um **vetor de inteiros com 10 elementos**
 - malloc retorna o endereço da área alocada para armazenar valores inteiros
 - ponteiro de inteiro recebe endereço inicial do espaço alocado

```
int *v;  
v = (int *) malloc(10*sizeof(int)) ;
```

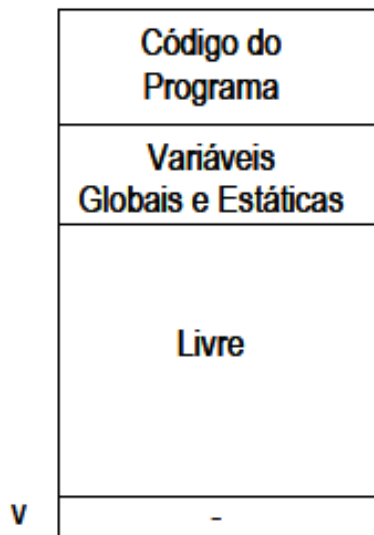
Alocação dinâmica

- Exemplo (cont.):

```
v = (int *) malloc(10*sizeof(int));
```

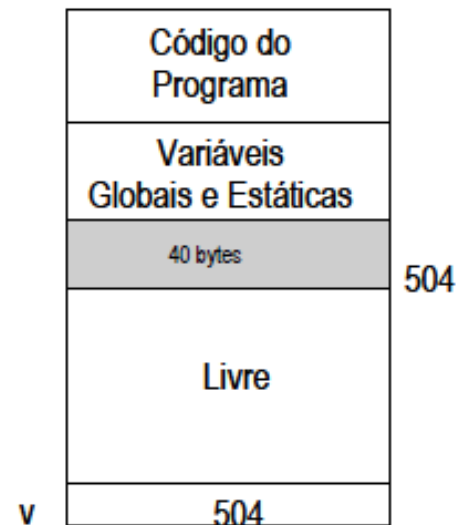
1 - Declaração: `int *v`

Abre-se espaço na pilha para o ponteiro (variável local)



2 - Comando: `v = (int *) malloc(10*sizeof(int))`

Reserva espaço de memória da área livre e atribui endereço à variável



Alocação dinâmica

- Exemplo (cont.):
 - v armazena endereço inicial de uma área contínua de memória suficiente para armazenar 10 valores inteiros
 - v pode ser tratado como um vetor declarado estaticamente
 - v aponta para o início da área alocada
 - v[0] acessa o espaço para o primeiro elemento
 - v[1] acessa o segundo
 - até v[9]



Alocação dinâmica

- Exemplo (cont.):
 - tratamento de erro após chamada a `malloc`
 - imprime mensagem de erro
 - aborta o programa (com a função `exit`)

```
...  
v = (int*) malloc(10*sizeof(int));  
if (v==NULL)  
{  
    printf("Memoria insuficiente.\n");  
    exit(1); /* aborta o programa e retorna 1 para o sist. operacional */  
}  
...  
  
free(v);
```



Alocação dinâmica

```
#include <stdlib.h>

int main ( void )
{
    float *v;
    float med, var;
    int i,n;

    printf("Entre n e depois os valores\n");
    scanf("%d",&n);
    v = (float *) malloc(n*sizeof(float));
    if (v==NULL) { printf("Falta memoria\n"); exit(1); }

    for ( i = 0; i < n; i++ )
        scanf("%f", &v[i]);

    med = media(n,v);
    var = variancia(n,v,med);

    printf ( "Media = %f   Variancia = %f   \n", med, var);
    free(v);
    return 0;
}
```



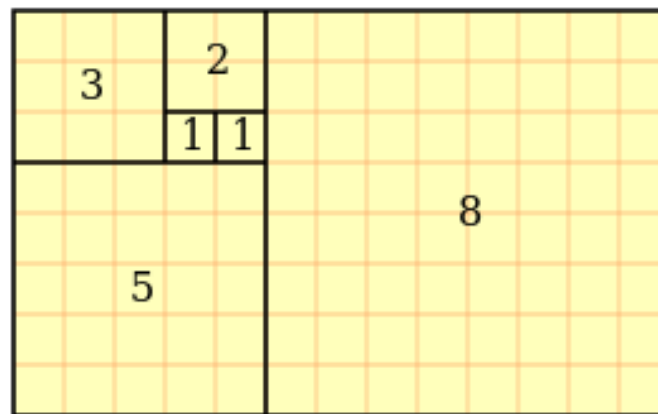
Atividade 1

- Escrever um programa que leia e ordene um vetor de estruturas contendo N valores reais e N nomes (N máximo é 1000). Os nomes não ultrapassam 50 caracteres.
- O vetor resultante deve ser ordenado pelos valores numéricos, de forma crescente, a partir da primeira posição do vetor. Para ordenar o vetor compare cada elemento com o seguinte e troque-os de posição se necessário. Faça isto da primeira a última posição do vetor. Se, nesta varredura, houver ao menos uma troca de posições, será necessário repetir o procedimento. Quando realizar uma troca, não esqueça de trocar números e nomes.



Atividade 2

- Implemente uma função recursiva que encontre o elemento apresentado na série de Fibonacci em uma dada posição.



- Série de Fibonacci:
 - $F_0 = F_1 = 1$
 - $F_n = F_{n-1} + F_{n-2}$ para $n > 1$,
 - 1, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89...

Referências

- CELES, Waldemar; CERQUEIRA, Renato; RANGEL, José Lucas. **Introdução a Estrutura de Dados**. Editora Campus, 2004.
- MIZRAHI, V. V. **Treinamento em linguagem C**. São Paulo: Makron Books, 1990.
- Material didático do Departamento de Informática da PUC-Rio (2014).
- Material didático do prof. Dr. Daniel Caetano (2012).

