Aula 25: Alocação Dinâmica

Introdução a Programação

Túlio Toffolo & Puca Huachi
http://www.toffolo.com.br

BCC201 – 2019/1
Departamento de Computação – UFOP

Aulas anteriores

- Estruturas de memórias heterogêneas

Aula de hoje

1. Estruturas heterogêneas (breve revisão)
2. Exercícios da aula prática
3. Alocação dinâmica
4. Exercícios
5. Próxima aula

Aula de hoje

1. Estruturas heterogêneas (breve revisão)
2. Exercícios da aula prática
3. Alocação dinâmica
4. Exercícios
5. Próxima aula

Struct

- `struct`: palavra reservada que cria um novo tipo de dados.
- Tipos conhecidos: `char`, `int`, `float`, `double` e `void`.
- Estrutura: é um tipo de estrutura de dados heterogênea; agrupa itens de dados de diferentes tipos.
- Cada item de dado é denominado membro (ou campo);
- `struct` define um tipo de dados (estrutura): informa ao compilador o nome, o tamanho em bytes e a maneira como ela deve ser armazenada e recuperada da memória.
- Ao ser definido, o tipo passa a existir e pode ser utilizado para criar variáveis.

Exemplo: armazenando dados de um aluno

```c
#include <stdio.h>

struct Aluno {
    int nMat;       // número de matrícula
    float nota[3]; // três notas
    float media;   // média aritmética
};                 // fim da definição da estrutura (com ;)

int main()
{
    struct Aluno bart;   // declara a variável do tipo 'struct Aluno'
    bart.nMat = 1521001;
    bart.nota[0] = 8.5;
    bart.nota[1] = 9.5;
    bart.nota[2] = 10.0;
    bart.media = ( bart.nota[0]+ bart.nota[1] + bart.nota[2] ) / 3.0;
    printf("Matrícula: %d\n", bart.nMat);
    printf("Média    : %.1f\n", bart.media);
    return 0;
}
```

Inicializando estruturas

A inicialização é semelhante a inicialização das matrizes.

```
1  struct Data {
2      int  dia;
3      char mes[10];
4      int  ano;
5  };
6
7  struct Data natal = { 25, "Dezembro", 2016 };
8  struct Data niver = { 20, "Outubro",  1986 };
```

Obs.: as variáveis são inicializadas juntamente com suas declarações. Os valores atribuídos aos membros devem ser colocados **na ordem em que foram definidos na estrutura**, separados por vírgula e entre chaves.

Atribuição entre estruturas

O uso de variáveis de estruturas é similar ao uso das variáveis que estamos acostumados a utilizar...

- Uma variável estrutura pode ser atribuída à outra variável do mesmo tipo por meio de uma atribuição simples.

```
1  struct Data natal = { 25, "Dezembro", 2016 };
2
3  struct Data natalDesteAno;
4  natalDesteAno = natal;
```

- **Importante**:
  - valores dos membros da estrutura são atribuídos de uma única vez;
  - a atribuição entre vetores/matrizes deve ser feita elemento por elemento.

O comando *typedef*

- O comando `typedef` define um apelido (*alias*) para um tipo.
- Em geral, apelidos simplificam o uso de estruturas em C.
- Exemplo:

```
1  typedef struct { // não precisamos definir o nome aqui
2      int  dia;
3      char mes[10];
4      int  ano;
5  } Data;          // 'apelido' (novo nome) para a estrutura: Data
```

- Uso simplificado (omitimos a palavra *struct* ao declarar variáveis):

```
1  Data natal = { 25, "Dezembro", 2016 };
2
3  Data natalDesteAno;
4  natalDesteAno = natal;
```

Exemplo de uso e operações em structs:

```c
typedef struct {
    int   pecas;
    float preco;
} Venda;

int main()
{
    Venda A = {20, 110.0};
    Venda B = {3, 258.0};
    Venda total;

    // soma membro a membro
    total.pecas = A.pecas + B.pecas;
    total.preco = A.preco + B.preco;
}
```

Erro comum

```c
// ERRO!
total = A + B;
```

Estruturas aninhadas

```c
typedef struct {
    int  dia;
    char mes[10];
    int  ano;
} Data;

typedef struct {
    int   pecas;
    float preco;
    Data  diaVenda;
} Venda;

int main()
{
    // exemplo de declaração
    Venda v = {20, 110.0, {7, "Novembro", 2015} };

    // exemplo de uso:
    printf("Ano da venda: %d", v.diaVenda.ano);

    return 0;
}
```

Estruturas em funções

As estruturas podem ser passadas como argumentos de funções da mesma maneira que as variáveis simples.

- O nome de uma estrutura em C não é um endereço, portanto ela pode ser passada por **valor**.
- Exemplo: função que recebe duas estruturas como argumento e imprime os valores da soma de seus membros.

```
1  typedef struct {
2      int   pecas;
3      float preco;
4  } Venda;
5
6  // protótipo (com passagem por valor)
7  void imprimeTotal(Venda v1, Venda v2);
```

Estruturas em funções

Exemplo utilizando passagem por **valor**:

```c
typedef struct {
    int   pecas;
    float preco;
} Venda;

void imprimeTotal(Venda v1, Venda v2)
{
    Venda total = {0, 0.0};
    total.pecas = v1.pecas + v2.pecas;
    total.preco = v1.preco + v2.preco;
    printf("Nro peças:   %d\n", total.pecas);
    printf("Preço total: %.2f\n", total.preco);
}

int main()
{
    Venda v1 = {1, 20}, v2 = {3, 10};
    imprimeTotal(v1, v2);
    return 0;
}
```

Estruturas em funções

- Podemos usar ponteiros para fazer passagem por **referência**:

```
1  typedef struct {
2      int   pecas;
3      float preco;
4  } Venda;
5
6  // protótipo (com passagem por referência)
7  void imprimeTotal(Venda *v1, Venda *v2);
```

Estruturas em funções

Exemplo utilizando **ponteiros**:

```
1  typedef struct {
2      int   pecas;
3      float preco;
4  } Venda;
5
6  void imprimeTotal(Venda *v1, Venda *v2)
7  {
8      Venda total = {0, 0.0};
9      total.pecas = (*v1).pecas + (*v2).pecas;
10     total.preco = (*v1).preco + (*v2).preco;
11     printf("Nro peças:   %d\n", total.pecas);
12     printf("Preço total: %.2f\n", total.preco);
13 }
14
15 int main()
16 {
17     Venda v1 = {1, 20}, v2 = {3, 10};
18     imprimeTotal(&v1, &v2);
19     return 0;
20 }
```

Estruturas em funções

Exemplo utilizando **ponteiros** (alternativa):

```
1  typedef struct {
2      int   pecas;
3      float preco;
4  } Venda;
5
6  void imprimeTotal(Venda *v1, Venda *v2)
7  {
8      Venda total = {0, 0.0};
9      total.pecas = v1->pecas + v2->pecas;
10     total.preco = v1->preco + v2->preco;
11     printf("Nro peças:   %d\n", total.pecas);
12     printf("Preço total: %.2f\n", total.preco);
13 }
14
15 int main()
16 {
17     Venda v1 = {1, 20}, v2 = {3, 10};
18     imprimeTotal(&v1, &v2);
19     return 0;
20 }
```

v1->pecas ou (*v1).pecas

Aula de hoje

1. Estruturas heterogêneas (breve revisão)
2. Exercícios da aula prática
3. Alocação dinâmica
4. Exercícios
5. Próxima aula

Questão 01

Escreva um programa que lê dois números: n ($n \leq 100$) e m ($m \leq 10$). Em seguida, o programa deve ler dados de n alunos: *nome completo*, *número de matricula* e m *notas*.
Após ler os dados dos alunos, seu programa deve:

- imprimir a média de todas notas;
- solicitar que o usuário digite um número de matrícula;
- imprimir o nome completo e a média das notas do aluno referente ao número de matrícula digitado.

Importante: você deve criar uma estrutura `Aluno`, além de uma função que retorna o `Aluno` que possui o número de matrícula passado por parâmetro. Utilize o seguinte protótipo:

```
1  Aluno encontraAluno(Aluno alunos[], int nAlunos, int matricula);
```

Exercícios da aula prática

Questão 02

Implemente uma função `equal` que retorna o inteiro 1 se dois números racionais, r_1 e r_2, são iguais e 0 caso contrário. Dica: reduza r_1 e r_2 a seus termos mínimos (lembra do MDC?) e verifique em seguida se os termos são iguais.

Implemente o método `main` para ler e comparar os números racionais r_1 e r_2, representados pela estrutura a seguir:

```
1  struct Racional {
2      int numerador;
3      int denominador;
4  };
```

Aula de hoje

1. Estruturas heterogêneas (breve revisão)
2. Exercícios da aula prática
3. Alocação dinâmica
4. Exercícios
5. Próxima aula

Alocação dinâmica

Comando `malloc`:

- Faz parte da biblioteca `<stdlib.h>`.
- Aloca dinamicamente um bloco consecutivo de *bytes* na memória e retorna o endereço deste bloco.
- Isto permite escrever programas mais flexíveis.
- Exemplo de uso: alocar um vetor de tamanho definido pelo usuário...

Alocação dinâmica

Uso do método `malloc` para criar um `double`:

```c
// aloca memória de forma dinâmica
double *nro = malloc(sizeof(double));

// altera o conteúdo da memória apontada por nro para 3.5
*nro = 3.5;

printf("Endereço de memória: %p\n", nro);
printf("Valor na memória: %lf\n", *nro);
```

- Este código imprimirá, por exemplo (arquitetura 64 bits):

```
Endereço de memória: 0x7feaf4400690
Valor na memória: 3.500000
```

Alocação dinâmica

Uso do método `malloc` para criar um bloco com 3 doubles:

```c
// aloca memória de forma dinâmica e inicializa com valor 10.5
double *nro = malloc(3 * sizeof(double));

// alterando valores
nro[0] = 1.1;
nro[1] = 1.5;
nro[2] = 2.2;

for (int i = 0; i < 3; i++)
    printf("%.1lf ", nro[i]);
```

- Este código imprimirá:

```
1.1 1.5 2.2
```

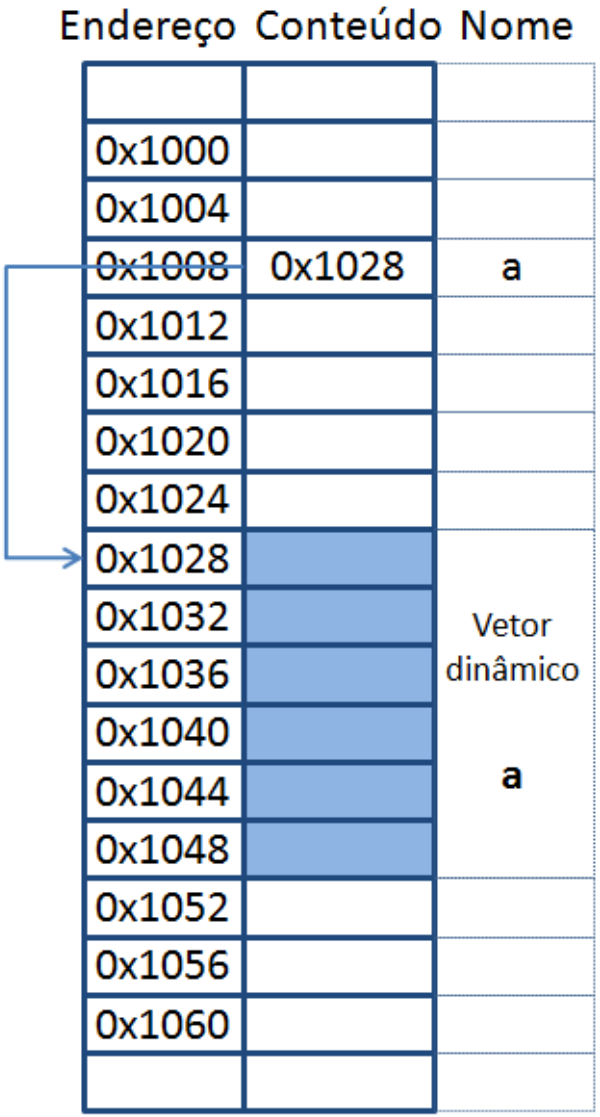
Exemplo de alocação dinâmica

Endereço	Conteúdo	Nome
0x1000		
0x1004		
0x1008	0x0000	a
0x1012		
0x1016		
0x1020		
0x1024		
0x1028		
0x1032		
0x1036		
0x1040		
0x1044		
0x1048		
0x1052		
0x1056		
0x1060		

```
1  int main() {
2   → int *a = NULL;
3      a = malloc(6 * sizeof(int));
4      for (int i = 0; i < 6; i++)
5          a[i] = i;
6      imprimeVetor3(a, 6);
7      ...
8  }
```

- Cria o ponteiro `a` com valor inicial `NULL` (0).

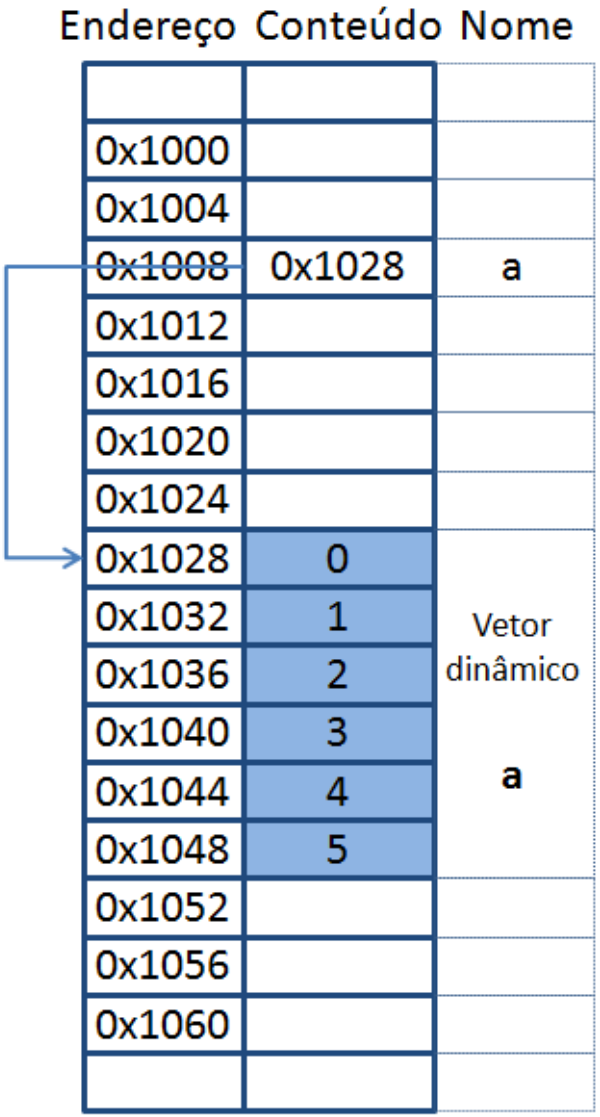
Exemplo de alocação dinâmica




```
1  int main() {
2      int *a = NULL;
3   →  a = malloc(6 * sizeof(int));
4      for (int i = 0; i < 6; i++)
5          a[i] = i;
6      imprimeVetor3(a, 6);
7      ...
8  }
```

- Aloca um bloco de memória com 6 inteiros (usando o comando `malloc`)
- Armazena o endereço de memória no ponteiro `a`.

Exemplo de alocação dinâmica




```
1  int main() {
2      int *a = NULL;
3      a = malloc(6 * sizeof(int));
4   →  for (int i = 0; i < 6; i++)
5   →      a[i] = i;
6      imprimeVetor3(a, 6);
7      ...
8  }
```

- Altera o valor de `a[0]`, `a[1]`, ..., `a[5]`

Muito importante: liberar a memória

- A memória alocada de forma estática pelo compilador é liberada automaticamente.
- Quando fazemos alocação dinâmica, **liberar a memória se torna nossa responsabilidade**.
- Em C usamos o procedimento `free`
- Exemplo (1):

```
1  int *a = malloc(sizeof(int));
2  ...
3  free(a);
```

Muito importante: liberar a memória

- A memória alocada de forma estática pelo compilador é liberada automaticamente.
- Quando fazemos alocação dinâmica, **liberar a memória se torna nossa responsabilidade**.
- Em C usamos o operador `free`
- Exemplo (2):

```
1  int *a = malloc(100 * sizeof(int));
2  ...
3  free(a);
```

Exemplo:

Endereço	Conteúdo	Nome
0x1000		
0x1004		
0x1008		
0x1012		
0x1016		
0x1020		
0x1024		
0x1028		
0x1032		
0x1036		
0x1040		
0x1044		
0x1048		
0x1052		
0x1056		
0x1060		

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

void leVetor(int *v, int n) {
    for (int i = 0; i < n; i++)
        scanf("%d", &v[i]);
}

int *maior(int *v, int n) {
    int *maior = v;
    for (int i = 1; i < n; i++)
        if (v[i] > *maior)
            maior = v + i;
    return maior;
}

int main() {
    int n, *v;
    scanf("%d", &n);
    v = malloc(n * sizeof(int));
    leVetor(v, n);
    int *valor = maior(v, n);
    printf("Maior = %d\n", *valor);
    free(v);
    return 0;
}
```

Aula de hoje

1. Estruturas heterogêneas (breve revisão)
2. Exercícios da aula prática
3. Alocação dinâmica
4. Exercícios
5. Próxima aula

Exercícios

Exercício 1

Crie uma função que:

1. recebe um vetor `v` e seu tamanho `n` por parâmetro;
2. cria um novo vetor por alocação dinâmica, preenchendo-o com o conteúdo de `v` em ordem inversa;
3. retorna este novo vetor.

Dica: utilize o protótipo a seguir:

```
1  int *inverso(int *v, int n);
```

- Crie um exemplo de utilização desta função no método `main()`.
- Não se esqueça de liberar a memória (usando `free`)!

Aula de hoje

1. Estruturas heterogêneas (breve revisão)
2. Exercícios da aula prática
3. Alocação dinâmica
4. Exercícios
5. Próxima aula

Próxima aula

- Leitura de arquivos texto
  - (a **Aula 28** será adiantada para *acelerar* o Trabalho Prático)
- Alocação dinâmica (parte 2)

Perguntas?