9002 — Aula 16
Algoritmos e Programação de Computadores

Instituto de Engenharia – UFMT

Segundo Semestre de 2014

18 de novembro de 2014

Roteiro

Três notas

Como armazenar 3 notas?

```
float nota1, nota2, nota3;

printf("Nota do aluno 1: ");
scanf("%f", &nota1);

printf("Nota do aluno 2: ");
scanf("%f", &nota2);

printf("Nota do aluno 3: ");
scanf("%f", &nota3);
```

Três notas

Como armazenar 3 notas?

```
float nota1, nota2, nota3;

printf("Nota do aluno 1: ");
scanf("%f", &nota1);

printf("Nota do aluno 2: ");
scanf("%f", &nota2);

printf("Nota do aluno 3: ");
scanf("%f", &nota3);
```

Três notas

Como armazenar 3 notas?

```
float nota1, nota2, nota3;

printf("Nota do aluno 1: ");
scanf("%f", &nota1);

printf("Nota do aluno 2: ");
scanf("%f", &nota2);

printf("Nota do aluno 3: ");
scanf("%f", &nota3);
```

Três notas

Como armazenar 3 notas?

```
float nota1, nota2, nota3;

printf("Nota do aluno 1: ");
scanf("%f", &nota1);

printf("Nota do aluno 2: ");
scanf("%f", &nota2);

printf("Nota do aluno 3: ");
scanf("%f", &nota3);
```

Três notas

Como armazenar 3 notas?

```
float nota1, nota2, nota3;

printf("Nota do aluno 1: ");
scanf("%f", &nota1);

printf("Nota do aluno 2: ");
scanf("%f", &nota2);

printf("Nota do aluno 3: ");
scanf("%f", &nota3);
```

Cem notas

Como armazenar exatamente 100 notas?

```
float nota1, nota2, nota3, ..., nota100;

printf("Nota do aluno 1: ");
scanf("%f", &nota1);

printf("Nota do aluno 2: ");
scanf("%f", &nota2);

...

printf("Nota do aluno 100: ");
scanf("%f", &nota100);
```

Cem notas

Como armazenar exatamente 100 notas?

```
float nota1, nota2, nota3, ..., nota100;

printf("Nota do aluno 1: ");
scanf("%f", &nota1);

printf("Nota do aluno 2: ");
scanf("%f", &nota2);

...

printf("Nota do aluno 100: ");
scanf("%f", &nota100);
```

Cem notas

Como armazenar exatamente 100 notas?

```
float nota1, nota2, nota3, ..., nota100;

printf("Nota do aluno 1: ");
scanf("%f", &nota1);

printf("Nota do aluno 2: ");
scanf("%f", &nota2);

...

printf("Nota do aluno 100: ");
scanf("%f", &nota100);
```

Cem notas

Como armazenar exatamente 100 notas?

```
float nota1, nota2, nota3, ..., nota100;

printf("Nota do aluno 1: ");
scanf("%f", &nota1);

printf("Nota do aluno 2: ");
scanf("%f", &nota2);

...

printf("Nota do aluno 100: ");
scanf("%f", &nota100);
```

Cem notas

Como armazenar exatamente 100 notas?

```
float nota1, nota2, nota3, ..., nota100;

printf("Nota do aluno 1: ");
scanf("%f", &nota1);

printf("Nota do aluno 2: ");
scanf("%f", &nota2);

...

printf("Nota do aluno 100: ");
scanf("%f", &nota100);
```

Cem notas

Como armazenar exatamente 100 notas?

```
float nota1, nota2, nota3, ..., nota100;

printf("Nota do aluno 1: ");
scanf("%f", &nota1);

printf("Nota do aluno 2: ");
scanf("%f", &nota2);

...

printf("Nota do aluno 100: ");
scanf("%f", &nota100);
```

Até cem notas

Como armazenar 100 notas, no máximo?

```
int n;
float nota1, nota2, nota3, ..., nota100;

printf("Número de notas: ");
scanf("%d", &n);

printf("Notas dos alunos: ");

if (n >= 1)
  scanf("%f", &nota1);
if (n >= 2)
  scanf("%f", &nota2);
...
if (n >= 100)
  scanf("%f", &nota100);
```

Até cem notas

Como armazenar 100 notas, no máximo?

```
int n;
float nota1, nota2, nota3, ..., nota100;

printf("Número de notas: ");
scanf("%d", &n);

printf("Notas dos alunos: ");

if (n >= 1)
  scanf("%f", &nota1);
if (n >= 2)
  scanf("%f", &nota2);
...
if (n >= 100)
  scanf("%f", &nota100);
```

Até cem notas

Como armazenar 100 notas, no máximo?

```
int n;
float nota1, nota2, nota3, ..., nota100;

printf("Número de notas: ");
scanf("%d", &n);

printf("Notas dos alunos: ");

if (n >= 1)
  scanf("%f", &nota1);
if (n >= 2)
  scanf("%f", &nota2);
...
if (n >= 100)
  scanf("%f", &nota100);
```

Até cem notas

Como armazenar 100 notas, no máximo?

```
int n;
float nota1, nota2, nota3, ..., nota100;

printf("Número de notas: ");
scanf("%d", &n);

printf("Notas dos alunos: ");

if (n >= 1)
  scanf("%f", &nota1);
if (n >= 2)
  scanf("%f", &nota2);
...
if (n >= 100)
  scanf("%f", &nota100);
```

Até cem notas

Como armazenar 100 notas, no máximo?

```
int n;
float nota1, nota2, nota3, ..., nota100;

printf("Número de notas: ");
scanf("%d", &n);

printf("Notas dos alunos: ");

if (n >= 1)
  scanf("%f", &nota1);
if (n >= 2)
  scanf("%f", &nota2);
...
if (n >= 100)
  scanf("%f", &nota100);
```

Até cem notas

Como armazenar 100 notas, no máximo?

```
int n;
float nota1, nota2, nota3, ..., nota100;

printf("Número de notas: ");
scanf("%d", &n);

printf("Notas dos alunos: ");

if (n >= 1)
  scanf("%f", &nota1);
if (n >= 2)
  scanf("%f", &nota2);
...
if (n >= 100)
  scanf("%f", &nota100);
```

Até cem notas

Como armazenar 100 notas, no máximo?

```
int n;
float nota1, nota2, nota3, ..., nota100;

printf("Número de notas: ");
scanf("%d", &n);

printf("Notas dos alunos: ");

if (n >= 1)
  scanf("%f", &nota1);
if (n >= 2)
  scanf("%f", &nota2);
...
if (n >= 100)
  scanf("%f", &nota100);
```

Até cem notas

Como armazenar 100 notas, no máximo?

```
int n;
float nota1, nota2, nota3, ..., nota100;

printf("Número de notas: ");
scanf("%d", &n);

printf("Notas dos alunos: ");

if (n >= 1)
  scanf("%f", &nota1);
if (n >= 2)
  scanf("%f", &nota2);
...
if (n >= 100)
  scanf("%f", &nota100);
```

Vetores — Definição

Vetor

Vetor é uma coleção de variáveis de um mesmo tipo referenciada por um nome comum.

Características

- Cada variável é acessada por por meio de índice
- O tamanho máximo n da coleção é pré-definido (deve ser constante)
- Os índices devem estar em um intervalo definido, normalmente de 0 a $n-1$ ou de 1 a n.

Vetores — Definição

Vetor

Vetor é uma coleção de variáveis de um mesmo tipo referenciada por um nome comum.

Características

- Cada variável é acessada por por meio de índice
- O tamanho máximo n da coleção é pré-definido (deve ser constante)
- Os índices devem estar em um intervalo definido, normalmente de 0 a $n-1$ ou de 1 a n.

Vetores — Definição

Vetor

Vetor é uma coleção de variáveis de um mesmo tipo referenciada por um nome comum.

Características

- Cada variável é acessada por por meio de índice
- O tamanho máximo n da coleção é pré-definido (deve ser constante)
- Os índices devem estar em um intervalo definido, normalmente de 0 a $n-1$ ou de 1 a n.

Vetores — Definição

Vetor

Vetor é uma coleção de variáveis de um mesmo tipo referenciada por um nome comum.

Características

- Cada variável é acessada por por meio de índice
- O tamanho máximo n da coleção é pré-definido (deve ser constante)
- Os índices devem estar em um intervalo definido, normalmente de 0 a $n-1$ ou de 1 a n.

Declaração de um vetor

Vetor de tamanho N

<tipo> identificador[N];

Em C:

- A primeira variável do vetor tem índice 0
- e a última variável do vetor tem índice $N - 1$.

Exemplo

```
float notas[100];
int medias[100];
char nome[200];
```

Declaração de um vetor

Vetor de tamanho N

<tipo> identificador[N];

Em C:

- A primeira variável do vetor tem índice 0
- e a última variável do vetor tem índice $N - 1$.

Exemplo

```
float notas[100];
int medias[100];
char nome[200];
```

Declaração de um vetor

Vetor de tamanho N

<tipo> identificador[N];

Em C:

- A primeira variável do vetor tem índice 0
- e a última variável do vetor tem índice $N - 1$.

Exemplo

```
float notas[100];
int medias[100];
char nome[200];
```

Usando um vetor

Acessando a variável na posição *i*

a = vetor[*i*];

- A posição *i* pode ser qualquer valor inteiro
- Podemos usar constantes ou variáveis para a posição

Exemplo

```
float notas[10], ultima_nota;
int i;

// todos tiraram nota 8.5
for(i = 0; i < 10; i++) {
  notas[i] = 8.5;
}

// o último aluno ganhou um ponto de bônus
notas[9] = notas[9] + 1.0;
ultima_nota = notas[9];
```

Usando um vetor

Acessando a variável na posição *i*

a = vetor[*i*];

- A posição *i* pode ser qualquer valor inteiro
- Podemos usar constantes ou variáveis para a posição

Exemplo

```
float notas[10], ultima_nota;
int i;


// todos tiraram nota 8.5
for(i = 0; i < 10; i++) {
  notas[i] = 8.5;
}


// o último aluno ganhou um ponto de bônus
notas[9] = notas[9] + 1.0;
ultima_nota = notas[9];
```

Usando um vetor

Acessando a variável na posição *i*

a = vetor[*i*];

- A posição *i* pode ser qualquer valor inteiro
- Podemos usar constantes ou variáveis para a posição

Exemplo

```
float notas[10], ultima_nota;
int i;

// todos tiraram nota 8.5
for(i = 0; i < 10; i++) {
  notas[i] = 8.5;
}

// o último aluno ganhou um ponto de bônus
notas[9] = notas[9] + 1.0;
ultima_nota = notas[9];
```

Vetores - Melhores práticas

Cuidados

- O tamanho do vetor é constante (não pode ser mudado durante a execução)
- **Então**, escolher sempre um bom tamanho para o vetor
  - Se for muito grande, haverá desperdício de memória
  - Se for muito pequeno, faltará espaço para os dados
- Sempre verificar duas vezes os índices dos vetores
- **Porque** índices fora dos limites causam efeitos indesejados
  - Alteram valores de outras variáveis
  - Causam segmentation fault (tentativa de acesso a um endereço de memória que não existe)

Vetores - Melhores práticas

Cuidados

- O tamanho do vetor é constante (não pode ser mudado durante a execução)
- **Então**, escolher sempre um bom tamanho para o vetor
  - Se for muito grande, haverá desperdício de memória
  - Se for muito pequeno, faltará espaço para os dados
- Sempre verificar duas vezes os índices dos vetores
- **Porque** índices fora dos limites causam efeitos indesejados
  - Alteram valores de outras variáveis
  - Causam segmentation fault (tentativa de acesso a um endereço de memória que não existe)

Vetores - Melhores práticas

Cuidados

- O tamanho do vetor é constante (não pode ser mudado durante a execução)
- **Então**, escolher sempre um bom tamanho para o vetor
  - Se for muito grande, haverá desperdício de memória
  - Se for muito pequeno, faltará espaço para os dados
- Sempre verificar duas vezes os índices dos vetores
- **Porque** índices fora dos limites causam efeitos indesejados
  - Alteram valores de outras variáveis
  - Causam segmentation fault (tentativa de acesso a um endereço de memória que não existe)

Vetores - Melhores práticas

Cuidados

- O tamanho do vetor é constante (não pode ser mudado durante a execução)
- **Então**, escolher sempre um bom tamanho para o vetor
  - Se for muito grande, haverá desperdício de memória
  - Se for muito pequeno, faltará espaço para os dados
- Sempre verificar duas vezes os índices dos vetores
- **Porque** índices fora dos limites causam efeitos indesejados
  - Alteram valores de outras variáveis
  - Causam segmentation fault (tentativa de acesso a um endereço de memória que não existe)

Vetores - Melhores práticas

Cuidados

- O tamanho do vetor é constante (não pode ser mudado durante a execução)
- **Então**, escolher sempre um bom tamanho para o vetor
  - Se for muito grande, haverá desperdício de memória
  - Se for muito pequeno, faltará espaço para os dados
- Sempre verificar duas vezes os índices dos vetores
- **Porque** índices fora dos limites causam efeitos indesejados
  - Alteram valores de outras variáveis
  - Causam segmentation fault (tentativa de acesso a um endereço de memória que não existe)

Vetores - Melhores práticas

Cuidados

- O tamanho do vetor é constante (não pode ser mudado durante a execução)
- **Então**, escolher sempre um bom tamanho para o vetor
  - Se for muito grande, haverá desperdício de memória
  - Se for muito pequeno, faltará espaço para os dados
- Sempre verificar duas vezes os índices dos vetores
- **Porque** índices fora dos limites causam efeitos indesejados
  - Alteram valores de outras variáveis
  - Causam segmentation fault (tentativa de acesso a um endereço de memória que não existe)

Vetores - Representação na memória

Nome	d	vetor					f
Índice	-	0	1	2	3	4	-
	0				9		5

Vamos executar o seguinte código:

```
int d, vetor[5], f;
d = 0;
vetor[3]  = 9;
vetor[5]  = 5;
vetor[-1] = 1;
printf("%d", d);
```

- O programa irá imprimir 1!
- Dependendo de como o compilador organiza a memória, o programa pode sair com *segmentation fault*

Vetores - Representação na memória

Nome	d	vetor					f
Índice	-	0	1	2	3	4	-
	0				9		5

Vamos executar o seguinte código:

```
int d, vetor[5], f;
d = 0;
vetor[3]  = 9;
vetor[5]  = 5;
vetor[-1] = 1;
printf("%d", d);
```

- O programa irá imprimir 1!
- Dependendo de como o compilador organiza a memória, o programa pode sair com *segmentation fault*

Vetores - Representação na memória

Nome	d	vetor					f
Índice	-	0	1	2	3	4	-
	0				9		5

Vamos executar o seguinte código:

```
int d, vetor[5], f;
d = 0;
vetor[3]  = 9;
vetor[5]  = 5;
vetor[-1] = 1;
printf("%d", d);
```

- O programa irá imprimir 1!
- Dependendo de como o compilador organiza a memória, o programa pode sair com *segmentation fault*

Vetores - Representação na memória

Nome	d	vetor					f
Índice	-	0	1	2	3	4	-
	0				9		5

Vamos executar o seguinte código:

```
int d, vetor[5], f;
d = 0;
vetor[3]  = 9;
vetor[5]  = 5;
vetor[-1] = 1;
printf("%d", d);
```

- O programa irá imprimir 1!
- Dependendo de como o compilador organiza a memória, o programa pode sair com *segmentation fault*

Vetores - Representação na memória

Nome	d	vetor					f
Índice	-	0	1	2	3	4	-
	0				9		5

Vamos executar o seguinte código:

```
int d, vetor[5], f;
d = 0;
vetor[3]  = 9;
vetor[5]  = 5;
vetor[-1] = 1;
printf("%d", d);
```

- O programa irá imprimir 1!
- Dependendo de como o compilador organiza a memória, o programa pode sair com *segmentation fault*

Vetores - Representação na memória

Nome	d	vetor					f
Índice	-	0	1	2	3	4	-
	1				9		5

Vamos executar o seguinte código:

```
int d, vetor[5], f;
d = 0;
vetor[3]  = 9;
vetor[5]  = 5;
vetor[-1] = 1;
printf("%d", d);
```

- O programa irá imprimir 1!
- Dependendo de como o compilador organiza a memória, o programa pode sair com *segmentation fault*

Vetores - Representação na memória

Nome	d	vetor					f
Índice	-	0	1	2	3	4	-
	1				9		5

Vamos executar o seguinte código:

```
int d, vetor[5], f;
d = 0;
vetor[3]  = 9;
vetor[5]  = 5;
vetor[-1] = 1;
printf("%d", d);
```

- O programa irá imprimir 1!
- Dependendo de como o compilador organiza a memória, o programa pode sair com *segmentation fault*

Vetores - Representação na memória

Nome	d	vetor					f
Índice	-	0	1	2	3	4	-
	1				9		5

Vamos executar o seguinte código:

```
int d, vetor[5], f;
d = 0;
vetor[3]  = 9;
vetor[5]  = 5;
vetor[-1] = 1;
printf("%d", d);
```

- O programa irá imprimir 1!
- Dependendo de como o compilador organiza a memória, o programa pode sair com *segmentation fault*

Vetores - Representação na memória

Nome	d	vetor					f
Índice	-	0	1	2	3	4	-
	1				9		5

Vamos executar o seguinte código:

```
int d, vetor[5], f;
d = 0;
vetor[3]  = 9;
vetor[5]  = 5;
vetor[-1] = 1;
printf("%d", d);
```

- O programa irá imprimir 1!
- Dependendo de como o compilador organiza a memória, o programa pode sair com *segmentation fault*

Até cem notas - Com vetor

```
int numero_notas, i;
float notas[100];

printf("Número de notas: ");
scanf("%d", &numero_notas);

for (i = 0; i < numero_notas; i++) {
    printf("Nota do aluno %d: ", i + 1);
    scanf("%f", &notas[i]);
}
```

O programa acima está correto?

Até cem notas - Com vetor

```
int numero_notas, i;
float notas[100];

printf("Número de notas: ");
scanf("%d", &numero_notas);

for (i = 0; i < numero_notas; i++) {
    printf("Nota do aluno %d: ", i + 1);
    scanf("%f", &notas[i]);
}
```

O programa acima está correto?

Até cem notas - Com vetor (evitando erros)

```
int numero_notas, i;
float notas[100];

printf("Número de notas: ");
scanf("%d", &numero_notas);

if (numero_notas > 100) {
    printf("ATENÇÃO: Número de alunos muito grande.\n");
    printf("Lendo apenas os 100 primeiros...\n");
    numero_notas = 100;
}

for (i = 0; i < numero_notas; i++) {
    printf("Nota do aluno %d: ", i + 1);
    scanf("%f", &notas[i]);
}
```

Parênteses: definindo constantes

Quantas vezes aparece o 60? E se o número de alunos mudar?

```
#include <stdio.h>

int main() {
    float provas[60], exercicios[60], media_turma;
    int i;

    printf("Notas das provas e exercícios dos alunos: ");
    for (i = 0; i < 60; i++)
        scanf("%f %f", &provas[i], &exercicios[i]);

    for (media_turma = 0, i = 0; i < 60; i++)
        media_turma += (provas[i] + exercicios[i])/2;
    media_turma = media_turma / 60;

    printf("Media da turma: %f\n", media_turma);
}
```

Parênteses: definindo constantes

Podemos dar um nome uma constante! Basta usar a diretiva `#define`.

```
#include <stdio.h>

#define NUM_ALU 60

int main() {
    float provas[NUM_ALU], exercicios[NUM_ALU], media_turma;
    int i;

    printf("Notas das provas e exercícios dos alunos: ");
    for (i = 0; i < NUM_ALU; i++)
        scanf("%f %f", &provas[i], &exercicios[i]);

    for (media_turma = 0, i = 0; i < NUM_ALU; i++)
        media_turma += (provas[i] + exercicios[i])/2;
    media_turma = media_turma / NUM_ALU;

    printf("Media da turma: %f\n", media_turma);
}
```

Nas próximas aulas...

- Nas próximas aulas veremos outros aspectos da linguagens relacionados ao uso de vetores e importantes algoritmos :)
- FIM!!