

Vetores e Strings

Prof. Denio Duarte

duarte@uffs.edu.br

Prof.

Vetores

- Um vetor é uma área de alocação contígua de memória
 - Ou seja, quando declaramos um vetor o programa vai alocar uma área contígua de memória baseada no tamanho do vetor
- Em C o vetor possui apenas um tipo de dado, todos os elementos ou posições do vetor armazenam o mesmo tipo de dado (**homogêneo**)
- Como declarar um vetor: **tipo nome[tamanho]**

```
3  int main(){
4
5      int vetor[10];
6      float vet[5000];
7
8      return 0;
9  }
```

Vetores

- Um vetor é uma área de alocação contígua de memória
 - Ou seja, quando declaramos um vetor o programa vai alocar uma área contígua de memória baseada no tamanho do vetor
- Em C o vetor possui apenas um tipo de dado, todos os elementos ou posições do vetor armazenam o mesmo tipo de dado (**homogêneo**)
- Como declarar um vetor: **tipo nome[tamanho]**

```
3  int m
4
5  int vetor[10];
6  float vet[5000];
7
8  return 0;
9  }
```



Vetores

- Um vetor é uma área de alocação contígua de memória
 - Ou seja, quando declaramos um vetor o programa vai alocar uma área contígua de memória baseada no tamanho do vetor
- Em C o vetor possui apenas um tipo de dado, todos os elementos ou posições do vetor armazenam o mesmo tipo de dado (**homogêneo**)
- Como declarar um vetor: **tipo nome[tamanho]**

```
3  int main()  
4  
5  int vetor[10];  
6  float vet[5000];  
7  
8  return 0;  
9  }
```

Nome do
vetor

Vetores

- Um vetor é uma área de alocação contígua de memória
 - Ou seja, quando declaramos um vetor o programa vai alocar uma área contígua de memória baseada no tamanho do vetor
- Em C o vetor possui apenas um tipo de dado, todos os elementos ou posições do vetor armazenam o mesmo tipo de dado (**homogêneo**)
- Como declarar um vetor: **tipo nome[tamanho]**

```
3  int main(){
4
5      int vetor[10];
6      float vet[5000];
7
8      return 0;
9  }
```

Tamanho, ou
número de
posições

Vetores

- Em C a contagem do índice inicia em 0 (zero)
 - O primeiro elemento do vetor é sempre o elemento na posição 0 (zero)
- Acesso é feito da mesma forma que em Python

```
3  int main(){  
4      int vetor[4];  
5  
6      vetor[0] = 1;  
7      vetor[1] = 1;  
8      vetor[2] = 1;  
9      vetor[3] = 1;  
10  
11     return 0;  
12 }
```

Vetores

- Em C a contagem do índice inicia de 0
 - O primeiro elemento do vetor é sempre o elemento na posição 0
- Acesso é feito da mesma forma que em Python

```
3  int main(  
4      int vet  
5  
6      vetor[0] = 1;  
7      vetor[1] = 1;  
8      vetor[2] = 1;  
9      vetor[3] = 1;  
10  
11     return 0;  
12 }
```

Acesso a uma
posição do
vetor.

Vetores

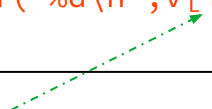
- Um vetor pode ser inicializado das seguintes formas

```
3  int main(){  
4      int vetor[] = {1,2,3,4};  
5  
6      int vetor[4] = {1,2,3,4};  
7  
8      return 0;  
9  }
```


Vetores

- Curiosidade sobre vetores:
 - São “ponteiros” disfarçados, ou seja, as posições são contíguas (contínuas) no espaço de endereçamento da RAM

```
int i, v[5]={2,4,6,8,10};
for (i=0;i<100;i++)
{
    printf(“%d\n”,v[i]);
}
```



Foram reservadas 5 posições na memória (0 a 4). O acesso para leitura das outras posições é permitido, porém conterá lixo. Para escrever, ocorrerá um erro

```
int i, v[5];
scanf(“%d”,&v[10]);
```

```
void printV(int *v, int l)
{
    int i;
    for (i=0;i<l;i++)
    {
        printf(“%d\n”,v[i]);
    }
}
//
int main()
{
    int v[5]={1,3,5,7,9};
    printV(v,5);
    return 0;
}
```

```
void printV(int v[], int l)
{
    int i;
    for (i=0;i<l;i++)
    {
        printf(“%d\n”,v[i]);
    }
}
//
int main()
{
    int v[5]={1,3,5,7,9};
    printV(v,5);
    return 0;
}
```

Vetores

O `v` é um ponteiro “constante”, não pode receber outro endereço de memória. Ou seja, não pode mudar o apontamento.

- Curiosidade sobre vetores
 - São “ponteiros” constantes, ou seja, as posições são contíguas (contínuas) no espaço de endereçamento da RAM

```
int i, v[5]={2,4,6,8,10};
for (i=0;i<100;i++)
{
    printf(“%d\n”,v[i])
}
```

Foram reservadas 5 posições na memória (0 a 4). O acesso para leitura das outras posições é permitido, porém conterá lixo. Para escrever, ocorrerá um erro

```
int i, v[5];
scanf(“%d”,&v[10]);
```

```
void printV(int *v, int l)
{
    int i;
    for (i=0;i<l;i++)
    {
        printf(“%d\n”,v[i]);
    }
}
//
int main()
{
    int v[5]={1,3,5,7,9};
    printV(v,5);
    return 0;
}
```

```
void printV(int v[], int l)
{
    int i;
    for (i=0;i<l;i++)
    {
        printf(“%d\n”,v[i]);
    }
}
//
int main()
{
    int v[5]={1,3,5,7,9};
    printV(v,5);
    return 0;
}
```

Vetores

Passar um vetor para uma função o faz perder a dimensão porque é lido como um ponteiro. (não temos como calcular o tamanho do vetor na função através da função `sizeof`)

- Curiosidade sobre vetores:

- São “ponteiros” disfarçados, ou seja, os elementos são contíguos (contínuos) no espaço de endereçamento da RAM

```
int i, v[5]={2,4,6,8,10};
for (i=0;i<100;i++)
{
    printf(“%d\n”,v[i])
}
```

Foram reservadas 5 posições na memória (0 a 4). O acesso para leitura das outras posições é permitido, porém conterá lixo. Para escrever, ocorrerá um erro

```
int i, v[5];
scanf(“%d”,&v[10]);
```

```
void printV(int *v, int l)
{
    int i;
    for (i=0;i<l;i++)
    {
        printf(“%d\n”,v[i]);
    }
}
//
int main()
{
    int v[5]={1,3,5,7,9};
    printV(v,5);
    return 0;
}
```

```
void printV(int v[], int l)
{
    int i;
    for (i=0;i<l;i++)
    {
        printf(“%d\n”,v[i]);
    }
}
//
int main()
{
    int v[5]={1,3,5,7,9};
    printV(v,5);
    return 0;
}
```

Vetores

1. Crie um programa que lê 5 valores inteiros, armazena-os em um vetor e, em seguida, mostre na tela os valores lidos.
2. Faça uma função que receba um vetor e o seu tamanho como parâmetro e imprima o maior e o menor elemento do vetor.
 - a. Esta função é do tipo **void** pois não retorna nenhum valor

Strings

- C não possui um tipo string explícito.
 - Ou seja, não existe a palavra reservada “string” ou “String”
- Importante lembrar:
 - O tipo caracter em C é envolvido por apóstrofes: ‘A’ e ‘c’.
 - O tipo string é envolvido por aspas: “Hoje”, “A”, “x” e “amanhã”.
 - Perceba que é “A” uma string e ‘A’ é um caracter
- Para declarar uma string em C, é feita uma cadeia de caracteres que é implementada por um vetor de caracteres
 - `char string[20];`

Strings

- Apesar de ser declarado como um vetor de caracteres no `scanf` e no `printf` basta utilizar `%s` para informar ao `C` que é uma string.

```
1  #include <stdio.h>
2
3  int main(){
4      char string[20];
5      scanf("%s", string);
6
7      printf ("Você digitou: %s\n", string);
8      return (0);
9  }
```

Strings

- Apesar de ser declarado como um vetor de caracteres no `scanf` e no `printf` basta utilizar `%s` para informar ao `C` que é uma string.

```
1  #include <stdio.h>
2
3  int main(){
4      char string[20];
5      scanf("%s", string);
6
7      printf ("Você digitou: %s\n", string);
8      return (0);
9  }
```

Estou limitando o
tamanho a 20

Strings

- Apesar de ser declarado como um vetor de caracteres no `scanf` e no `printf` basta utilizar `%s` para informar ao C que é uma string.

```
1  #include <stdio.h>
2
3  int main(){
4      char string[20];
5      scanf("%s", string);
6
7      printf ("Você digitou: %s\n", string);
8      return (0);
9  }
```

vai ler o valor digitado até o primeiro " " (espaço)

Strings

- Não é possível atribuir valores diretamente para strings pois, afinal, são vetores de caracteres

```
char st[10];  
st="Bingo"; // ERRADO  
st[0]='B';  
st[1]='i';  
st[2]='n';  
st[3]='g';  
st[4]='o'; // Correto
```

Strings

- Não é possível atribuir valores diretamente para strings pois, afinal, são vetores de caracteres

```
char st[10];  
st="Bingo"; // ERRADO  
st[0]='B';  
st[1]='i';  
st[2]='\n';  
st[3]='g';  
st[4]='o'; // Correto
```

```
st[5]='\0'; \\ ou st[5]=0
```

Como avisar ao C que as outras posições não fazem parte da cadeia? No exemplo, as posições **5, 6, 7, 8 e 9** não estão sendo utilizadas. A solução é colocar na próxima posição o caracter de fim de cadeia. Caracter **'\0'** faz o papel. Ou pode-se colocar, simplesmente, **0**

O **scanf** e o **printf** fazem esse controle automaticamente quando colocamos **%s** como formatação.

Strings

- Não é possível comparar strings (lembre-se, string são vetores)

```
if (st=="Ok") // não é possível
{
    :
}
// é possível mas trabalhoso
if (st[0]=='O' && st[1]=='k')
{
    :
}
```

Strings

- Como strings são vetores e vetores são ponteiros, podemos:
 - Iniciar a string como fazemos com ponteiros mas utilizando aspas
 - Não colocar o tamanho do vetor (assume o tamanho da string + 1 - para o \0)
 - Podemos criar como ponteiro diretamente

```
char st[10]="Bingo";  
    ou  
char st[]="Bingo";  
    ou  
char *st="Bingo";  
// nas 2 últimas opções  
// st terá o tamanho 6
```

- Não colocamos **&** no **scanf** pois o nome da string representa um endereço

```
scanf("%s",st);
```

Strings

- Como strings são vetores e vetores são ponteiros, podemos:
 - Iniciar a string como fazemos com ponteiros mas utilizando aspas
 - Não colocar o tamanho do vetor (assume o tamanho da string + 1 - para o \0)

- Podemos

scanf lê até o primeiro espaço, ou enter (\n).

- Usar `scanf("%s[^\n]", st);` para ler uma linha com espaços
- Sugere-se usar a função `fgets(str,<tamanho>, stdin);`

`char`

`char`

ou

`char *st="Bing"`

`// nas 2 últi`

`// st terá o`

- Não colocamos o tamanho no **scanf** pois o nome da string representa um endereço

`scanf("%s",st);`

Strings

- Como resolver os problemas de atribuição e comparação entre strings?
- Biblioteca “string.h”
 - Ela apresenta uma série de funções úteis que permitem manipular as strings
 - Para utilizá-la basta adicionar
 - `#include <string.h>`
 - Funções úteis
 - **strcpy(destino, origem)**: Copia uma string **origem** para uma string **destino**
 - **char st[10];**
 - **strcpy(st,"Denio");** // fica **st[0]='D' st[1]='e' st[2]='n' st[3]='i' st[4]='o' st[5]='\0'**
 - **strcmp** : Compara duas strings, retorna 0 se forem iguais, outros valores, são diferentes
 - if (strcmp(st,"Denio")==0)**
printf("Seu sobrenome é Duarte\n");
 - if (strcmp(st,"Denio")!=0)**
printf("Você não é o senhor Duarte\n");
 - **strlen(str)**: devolve o tamanho de **st**