Universidade Positivo

Algoritmos de Programação

Vetores e Strings

Prof.ª Mariane Cassenote

mariane.cassenote@up.edu.br

Conteúdo

- Retomada de vetores
- Uso de vetores com funções
- Uso de strings em C

Ao final do componente curricular será possível aplicar estruturas homogêneas de armazenamento com uma dimensão em problemas algorítmicos.

O que são vetores?

- Vetor é uma coleção de elementos de um **mesmo tipo** de compartilham o **mesmo nome** e ocupam posições consecutivas na memória do computador
- Ao invés de armazenar apenas um valor, as variáveis do tipo vetor armazenam vários valores simultaneamente
- Vetor é uma estrutura de dados homogênea

O que são vetores?

- Um vetor precisa ter bem definidos:
  - o tipo dos valores armazenados
  - seu nome
  - seu tamanho (ou sua capacidade)

int vet[12];

índices	0	1	2	3	4	5	6	7	8	9	10	11
valores	57	28	-73	12	0	-14	-21	95	37	92	14	22

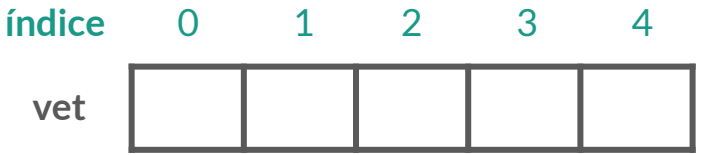
O que são vetores?

- Para acessar o valor armazenado em uma determinada posição de um vetor, precisamos saber exatamente em que posição esse valor se encontra
- A contagem do índice de um vetor começa em zero na maior parte das linguagens de programação
- Nesse caso, o último índice do vetor será: [`<tamanho_vetor> - 1`]

int vet[12];

índices	0	1	2	3	4	5	6	7	8	9	10	11
valores	57	28	-73	12	0	-14	-21	95	37	92	14	22

Vetores em C



- Declaração:
  - `int vet[5];`
  - `float notas[80];`
  - `double calorias[200];`
  - `char nome[158];`

Vetores em C

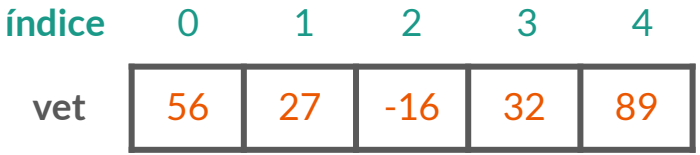
índice	0	1	2	3	4
vet	56	27	-16	32	89

- Atribuição de valores:
  - `vet[0] = 56;` *// atribuição de valor para uma única posição*
  - `int vet[5] = {56, 27, -16, 32, 89};` *// atribuição de valores para todo o vetor*
  - `int vet[] = {56, 27, -16, 32, 89};` *// sem tamanho também funciona*
  - `int vet[5] = {56, 27, -16, 32};` *// o último elemento não foi inicializado*
  - `int vet[5];`

    `vet[5] = {56, 27, -16, 32, 89};` *// declaração e atribuição separadas não funciona!*

    `vet[] = {56, 27, -16, 32, 89};` *// declaração e atribuição separadas não funciona!*

Vetores em C



- Atribuição de valores:
  - `scanf("%d", &vetor[0]);` *// leitura de valor para uma única posição*
  - `for (int i = 0; i < 5; i++) {` *// leitura de valores para todo o vetor*

```
        scanf("%d", &vet[i]);
    }
```

Vetores em C

índice	0	1	2	3	4
vet	56	27	-16	32	89

- Acessar valor:
  - `int varA = vet[0];` *// acesso ao valor de uma única posição*
  - `printf("%d", vet[0]);` *// impressão de valor de uma única posição*
  - `for (int i = 0; i < 5; i++) {` *// impressão de valores de todo o vetor*

```
        printf("%d", vet[i]);
    }
```

Vetores em C

- Se vetores (ou algum elemento específico) não forem inicializados, eles podem conter lixo de memória
- Um tentativa de acesso (leitura ou escrita) fora dos limites do vetor não gera erro de compilação, mas pode retornar lixo de memória
- A linguagem C não tem verificação de limites em vetores. É de responsabilidade do programador fazer essa verificação

Vetores em C

```c
// violação de limites de um vetor
#include <stdio.h>

int main() {

  int vet[5], i;

  // acesso fora dos limites quando i == 5
  for (i = 0; i <= 5; i++) {
      vet[i] = 1;
  }

  return 0;
}
```

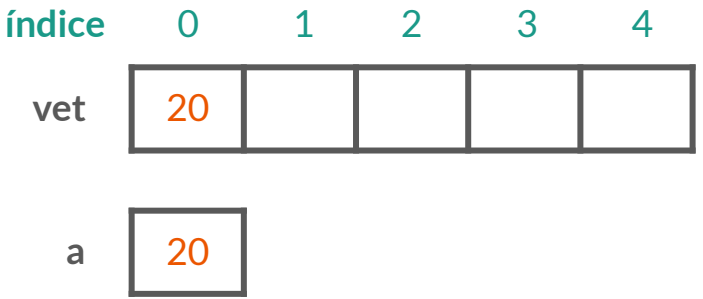
Não gera erro de compilação, mas pode destruir alguma informação importante que estava armazenada no endereço de memória invadido

Vetores em C

```c
// operações entre vetores
#include <stdio.h>

int main() {

  int i;
  int vetY[5] = {7, 1, 4, 9, 3};

  // vetX = vetY;  // incorreto!!!

  for (i = 0; i < 5; i++) {
     vetX[i] = vetY[i]; // correto!!!
  }

  for (i = 0; i < 5; i++) {
     printf("VetorX [%d] : %d \n", i, vetX[i]);
  }

  return 0;
}
```

Vetores em pseudocódigo

- Declaração:
  - `inteiro vet[5]`
- Atribuição de valor:
  - `vet[0] = 20`
- Acessar valor:
  - `a = vet[0]`



Strings em C

- Em linguagem C não existe um tipo de dados string nativo
- Strings são implementadas como vetores de caracteres (tipo `char`)
- Toda string é terminada pelo caractere especial `'\0'`, que é considerado no tamanho do vetor
    - Strings devem ser definidas com um caractere a mais do que o necessário
- Declaração:
    - `char nome[151];`

Adaptado de http://linguagemc.com.br/string-em-c-vetor-de-caracteres/. Acesso em 24/09/2023.

Strings em C – Inicialização direta

- Inicialização:
  - `char nome[101] = "Irmao do Jorel";`
  - `char nome[101] = {'I', 'r', 'm', 'a', 'o', ' ', 'd', 'o', ' ', 'J', 'o', 'r', 'e', 'l'};`
  - `char nome[] = "Irmao do Jorel";`
  - `char nome[15] = "Irmao do Jorel";`

Equivalentes

Neste caso, quantidade de caracteres é calculada automaticamente (14 + 1 para o finalizador `'\0'`)

Strings em C – Impressão

```c
// percorrendo um vetor de char e imprimindo cada caractere
#include <stdio.h>

int main() {
  char nome[15] = "Irmao do Jorel";

  // imprimindo a string inteira
  printf("O nome armazenado foi: %s \n", nome);

  // imprimindo cada caractere
  for(int i = 0; i < 14; i++) {
      printf("Elemento %d: %c \n", i, nome[i]);
  }

  return 0;
}
```

```
> ./main
O nome armazenado foi: Irmao do Jorel
Elemento 0: I
Elemento 1: r
Elemento 2: m
Elemento 3: a
Elemento 4: o
Elemento 5:
Elemento 6: d
Elemento 7: o
Elemento 8:
Elemento 9: J
Elemento 10: o
Elemento 11: r
Elemento 12: e
Elemento 13: l
>
```

Strings em C – Impressão

- Strings podem ser impressas usando **`printf()`** e o especificador de formato **`%s`**
- Strings também podem ser impressas com a função **`puts()`**, que as **imprime e dá uma quebra de linha**. Nesse caso, **nenhuma opção de formatação pode ser definida**. A função somente pega uma string como argumento e a imprime

```c
#include <stdio.h>

int main() {
  char nome[] = "Irmao do Jorel";

  printf("O nome armazenado foi: %s \n", nome);

  puts("O nome armazenado foi:");
  puts(nome);

  return 0;
}
```

```
> ./main
O nome armazenado foi: Irmao do Jorel
O nome armazenado foi:
Irmao do Jorel
>
```

Strings em C – Leitura com scanf

- A função scanf permite fazer leitura de strings usando **%s**
- scanf realiza a leitura **até encontrar um espaço**, depois encerra a leitura e coloca o caractere terminal '\0'
- A variável que vai armazenar a string **não deve ser precedida por &**

```c
#include <stdio.h>

int main() {
  char nome[101];

  printf("Digite seu nome: ");
  scanf("%s", nome);

  printf("O nome armazenado foi: %s", nome);

  return 0;
}
```

Não usa &

```
> ./main
Digite seu nome: Irmao do Jorel
O nome armazenado foi: Irmao>
```

Strings em C – Leitura com scanf

```c
#include <stdio.h>

int main() {
  char nome[101];
  char outronome[101];

  printf("Digite seu nome: ");
  scanf("%20s", nome); // "%20s" limita a 20 caracteres

  printf("Digite outro nome: ");
  scanf("%s", outronome);

  printf("Os nomes armazenados foram: %s e %s", nome, outronome);

  return 0;
}
```

```
> ./main
Digite seu nome: IrmaoDoJorel
Digite outro nome: VovoJuju
Os nomes armazenados foram: IrmaoDoJorel e VovoJuju>
```

Strings em C – Leitura com `fgets`

- Armazenam tudo o que foi digitado, inclusive espaços, **até que a tecla ENTER seja pressionada ou o tamanho máximo seja atingido**
- Com `fgets()` **limita-se** o tamanho da string aceita

```c
#include <stdio.h>

int main() {
  char nome[101];

  printf("Digite seu nome: ");
  fgets(nome, 101, stdin);

  printf("O nome armazenado foi: %s", nome);

  return 0;
}
```

aceita 100 caracteres e reserva uma posição para '\0'

```
> ./main
Digite seu nome: Irmao do Jorel
O nome armazenado foi: Irmao do Jorel
>
```

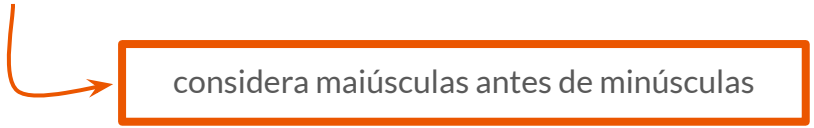
Strings em C

Leituras **fortemente** recomendadas sobre funções da biblioteca string.h:

- `http://linguagemc.com.br/a-biblioteca-string-h/`
- `https://man7.org/linux/man-pages/man3/string.3.html`

Strings em C

- A biblioteca string.h contém uma série de funções para manipular strings. Por exemplo:
  - Copiar string `origem` para a string `destino` com **`strcpy`**`(destino, origem)`
  - Concatenar `string2` no fim da `string1` com **`strcat`**`(string1, string2)`
  - Descobrir o tamanho de uma string com **`strlen`**`(stringX)`
  - Comparar strings alfabeticamente com **`strcmp`**`(string1, string2)`



Adaptado de http://linguagemc.com.br/string-em-c-vetor-de-caracteres/. Acesso em 24/09/2023.

Vetores com funções em C – outra forma de escrever

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

int main() {

  char nome[50] = "Irmao";
  char outronome[50] = "do Jorel";

  int compara = strcmp(nome, outronome);

  if (compara < 0) {
     printf("Primeiro nome vem primeiro \n");
  } else if (compara > 0) {
     printf("Segundo nome vem primeiro \n");
  } else {
     printf("Nomes iguais \n");
  }
```

considera maiúsculas antes de minúsculas

→ *(segue no quadro ao lado)*

```c
  int tamanho = strlen(nome);
  printf("O tamanho do primeiro nome eh: %d \n",
  tamanho);

  strcat(nome, outronome);
  printf("Nomes concatenados: %s \n", nome);

  strcpy(nome, outronome);
  printf("Segundo copiado para o lugar do
  primeiro: %s \n", nome);

  return 0;
}
```

```
> ./main
Segundo nome vem primeiro
O tamanho do primeiro nome eh: 5
Nomes concatenados: Irmaodo Jorel
Segundo copiado para o lugar do primeiro: do Jorel
>
```

Adaptado de http://linguagemc.com.br/a-biblioteca-string-h/. Acesso em 24/09/2022.

Strings em C

Função	Operação realizada
`int strlen(s)`	informa o número de caracteres da string `s` (sem o `'\0'`)
`char * strcpy (destino, origem)`	copia a string `origem` para o lugar da string `destino` (deve haver espaço alocado)
`char * strncpy (destino, origem, n)`	copia `n` caracteres da string `origem` para o lugar da string `destino` (deve haver espaço alocado)
`int strcmp (string1, string2)`	compara duas strings considerando a ordem alfabética (retorna 0 para strings iguais, negativo se `string1 > string2` e positivo se `string1 < string2`) considera maiúsculas antes de minúsculas
`int strncmp (string1, string2 ,n)`	idem, mas considera somente os `n` primeiros caracteres

Créditos ao professor Carlos Alberto Mazieiro

Strings em C

Função	Operação realizada
`char * strcat (string1, string2)`	concatena a `string2` no final da `string1` (deve haver espaço alocado)
`char * strncat (string1, string2, n)`	idem, mas concatena somente os `n` primeiros caracteres
`char * strchr (string, caractere)`	retorna um ponteiro para a primeira ocorrência do caractere na string ou `NULL` se não encontrar nenhuma ocorrência
`char * strrchr (string, caractere)`	idem, mas retorna um ponteiro para a última ocorrência do caractere

Strings no Portugol WebStudio

- Strings em Visualg são tratadas como variáveis comuns do tipo `cadeia`
- Operações como comparação e concatenação podem ser realizada da mesma forma que operações numéricas (com os operadores de igualdade “=” e adição “+”)Biblioteca texto
- https://www.youtube.com/watch?v=84TY2C1kKJQ

Strings no Visualg

- Strings em Visualg são tratadas como variáveis comuns do tipo `caractere`
- Operações como comparação e concatenação podem ser realizada da mesma forma que operações numéricas (com os operadores de igualdade “=” e adição “+”)
- Funções para manipulação de strings em Visualg podem ser encontradas em:
  - `https://www.manual.visualg3.com.br/doku.php?id=manual`
  - `http://www.bosontreinamentos.com.br/logica-de-programacao/logica-de-programacao-funcoes-de-manipulacao-de-strings-literais/`

Para praticar

1. Escreva um programa que leia uma string do teclado e a escreva na tela ao contrário (do final para o início).

2. Escreva um programa que leia uma string do teclado (com tamanho máximo de 200 caracteres) e conte quantas ocorrências de cada vogal existem nessa string.

3. Escreva um programa que receba do teclado uma string de até 200 caracteres e um determinado caractere, e devolva a localização da última ocorrência desse caractere na string. Se o caractere não aparecer na string, o programa deve exibir uma mensagem informando isso.

Desafio

Seja um vetor de inteiros de nove elementos. Escreva um programa que realize a seguinte sequência de ações, na ordem indicada:

a) preenchimento do vetor por leitura de valores positivos;

b) impressão dos elementos do vetor em uma linha, usando dois espaços em branco para separar os valores de cada elemento;

c) rotação dos elementos do vetor, com o deslocamento de todos os valores dos elementos para a posição seguinte, exceto o último valor, que deve ser colocado na posição do primeiro elemento;

d) repetição do item b;

e) soma do índice de cada elemento ao conteúdo da posição correspondente;

f) repetição do item b;

g) leitura de um valor inteiro e apresentação do número de ocorrências desse valor no vetor.

EDELWEISS, Nina. Algoritmos e programação com exemplos em Pascal e C. Porto Alegre Bookman 2014 1 recurso on-line (Livros didáticos UFRGS 23).

Universidade Positivo

Algoritmos de Programação

Vetores e Strings

Prof.ª Mariane Cassenote

mariane.cassenote@up.edu.br