# Computação Eletrônica

## Strings

Prof: Luciano Barbosa

Centro de Informática U·F·P·E

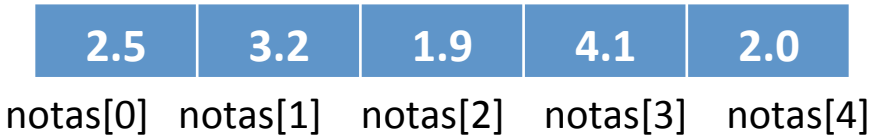
Universidade Federal de Pernambuco

# Recapitulando: Vetores

- Representar uma coleção de variáveis de um mesmo tipo em uma dimensão
- Ex: float notas[5];

  ou

  float notas[5] = {2.5,3.2,1.9,4.1,2.0};



# Recapitulando: Matrizes

- Representar uma coleção de variáveis de um mesmo tipo em duas dimensões
- Ex: float mat[2][3];

  ou

  float mat[2][3] = {{2.5,3.2,1.9},{4.1,2.0,5.4}};

| mat[0,0] | mat[0,1] | mat[0,2] |
|---|---|---|
| **2.5** | **3.2** | **1.9** |
| **4.1** | **2.0** | **5.4** |
| mat[1,0] | mat[1,1] | mat[1,2] |

# Manipular Vetores de Caracteres (Strings)

- Caracteres em C
  - Entrada/Saída de caracteres
  - Funções que manipulam caracteres
- Vetores de caracteres (Strings)
  - Inicialização
  - Entrada/Saída de Strings
  - Funções de Manipulação de Strings

# Caracteres: Tipo char

- Usado para representar caracteres
- Armazena valores inteiros (em 1 byte)
- Um *literal* **char** *é escrito entre aspas simples:*

```c
#include <stdio.h>

int main()
{
    char letraA = 'A';
    char letraC;
    letraC = 'C';
    printf ( " %c %c ", letraA , letraC);
    return 0;
}
```

# Caracteres: Representação Interna

- Representados internamente na memória do computador por códigos numéricos

```c
#include <stdio.h>

int main() {
    char letraA = 65;//Código da letra A
    char letraC;
    letraC = 67;//Código da letra C
    printf ( "%c %c ", letraA , letraC) ;
}
```

- Tabela ASCII: mapeamento entre caractere e código numérico
- Na tabela ASCII:
  - os dígitos são codificados em sequência
  - as letras minúsculas e maiúsculas também

# Tabela ASCII do 30 ao 126

| | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 |
|---|---|---|---|---|---|---|---|---|---|---|
| 30 | | | sp | ! | “ | # | $ | % | & | ‘ |
| 40 | ( | ) | * | + | , | - | . | / | 0 | 1 |
| 50 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | : | ; |
| 60 | < | = | > | ? | @ | A | B | C | D | E |
| 70 | F | G | H | I | J | K | L | M | N | O |
| 80 | P | Q | R | S | T | U | V | W | X | Y |
| 90 | Z | [ | \ | ] | ^ | _ | ` | a | b | c |
| 100 | d | e | f | g | h | i | j | k | l | m |
| 110 | n | o | p | q | r | s | t | u | v | w |
| 120 | x | y | z | { | \| | } | ~ | | | |

Observe que as letras maiúsculas (65-90), minúsculas (97-122) e os dígitos (48-57) estão dispostos em sequência!

Caracteres de controle

- 000 – ‘\0’ (fim de string)
- 009 – ‘\t’ (tabulação)
- 010 – ‘\n’ (fim de linha)
- 013 – ‘\r’ (retorno de linha)

# Codificação Sequencial

- Ajuda a identificar o tipo de caractere
- A função abaixo verifica se um dado caractere é um dígito entre ‘ 0 ’ e ‘ 9 ’:

```
int digito (char c) {
    int ehDigito;
    if(( c >= '0')&&(c <= '9')) {
        ehDigito = 1;
    }
    else {
        ehDigito = 0;
    }
    return ehDigito;
}
```

# Conversão para Maiúsculoa

```
char maiuscula(char c)
{
    char maiusc = c;
    if((c >= 'a')&&(c <= 'z'))
    {
        maiusc = c - ('a' - 'A');
    }
    return maiusc ;
}
```

Testa se é minúscula

A diferença entre qualquer caractere minúsculo e seu respectivo maiúsculo é a mesma da letra 'a' minúscula para a letra 'A' maiúscula.

| | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 |
|---|---|---|---|---|---|---|---|---|---|---|
| 30 | | | sp | ! | “ | # | $ | % | & | ‘ |
| 40 | ( | ) | * | + | , | - | . | / | 0 | 1 |
| 50 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | : | ; |
| 60 | < | = | > | ? | @ | A | B | C | D | E |
| 70 | F | G | H | I | J | K | L | M | N | O |
| 80 | P | Q | R | S | T | U | V | W | X | Y |
| 90 | Z | [ | \ | ] | ^ | _ | ` | a | b | c |
| 100 | d | e | f | g | h | i | j | k | l | m |
| 110 | n | o | p | q | r | s | t | u | v | w |
| 120 | x | y | z | { | \| | } | ~ | | | |

# Impressão de Caracteres

- Duas formas diferentes usando o printf (%d: como valor ASCII, %c: como caractere):

```
char lc = 97 ;
printf("%d %c",lc,lc);
```

Saída:
97 a

```
char la = 'a' ;
printf("%d %c",la,la );
```

Saída:
97 a

- Função putchar (stdio.h) permite a impressão de um caractere:

```
char la = 'a'; //ou: la = 97;
putchar(la);
```

Saída:
a

# Leitura de Caracteres

- Função scanf

```
char a ;
scanf("%c",&a);
```

ou

```
char a ;
scanf(" %c",&a);
```

Este espaço diz ao scanf para ignorar espaços, tabulações, ou outros caracteres de controle que estejam no buffer do teclado

- Função getchar (stdio.h) permite a leitura de um caractere

```
char a ;
a = getchar();
```

# Leitura de Caracteres

- As funções scanf e getchar obrigam que a tecla *enter* '\n' seja pressionada após a entrada dos dados.
- Existem funções para ler dados sem esperar pelo *enter* em C **para ambientes Windows:**
  - Função getche – definida em conio.h:

```
char letra;
letra = getche();
```

Lê um caractere e o exibe na tela

  - Função getch – definida em conio.h:

```
char letra;
letra = getch();
```

Lê um caractere e não exibe na tela

# Strings: vetores de caracteres

- Vetor do tipo **char**
- Terminadas pelo caractere nulo: ‘ \0 ’;

| R | i | o | \0 |
|---|---|---|---|

- Para imprimir printf: %s
- Muitas funções que manipulam strings o fazem caractere a caractere, a partir do endereço do primeiro até que ‘\0’ seja encontrado.

```
int main() {
    char cidade[4];
    cidade[0]='R';
    cidade[1]='I';
    cidade[2]='O';
    cidade[3]='\0';
    printf("%s",cidade);
}
```

O identificador “cidade” (sem colchetes) referencia o endereço do primeiro caractere da string.

# Inicialização de Strings

- Na declaração:

```
int main(){
    char cidade[]={'R','I','O','\0'} ;
    printf ("%s\n",cidade );
}
```

- Através da escrita dos caracteres entre aspas duplas:

```
int main(){
    char cidade[]= "RIO";
    printf("%s\n",cidade);
}
```

O caractere nulo é representado implicitamente e o vetor é declarado com tamanho 4

# Declaração de Strings

- Inicialização do vetor de caracteres na declaração:

```
char s2[] = "Rio de Janeiro";

char s3[81];

char s4[81] = "Rio";
```

Representa um vetor com 15 elementos

Representa um vetor de no máximo, 80 caracteres válidos

Representa um vetor de no máximo 80 caracteres válidos, mas com um valor já inicializado

# Literais do tipo String

- Devem ser declarados entre aspas duplas.
- Ex:

```
printf("Um literal string!\n");
printf("Eu moro em %s ","Recife");
```

- Para cada literal presente no código, é criada e inicializada uma região de memória que cabe todos os caracteres e o '\0':

| Endereço: | 100 | 101 | 102 | 103 | 104 | 105 | 106 |
|---|---|---|---|---|---|---|---|
| Dados: | R | e | c | i | f | e | \0 |

# Literais do tipo String

- A atribuição de literais para strings não é permitida:

```
int main(){
    char cidade[4];
    cidade = "Rio";
}
```

- A não ser na declaração:

```
int main(){
    char cidade[]= "RIO";
    printf("%s\n",cidade);
}
```

- Ou pode ser utilizada a função strcpy em string.h

```
int main(){
    char cidade[4];
    strcpy(cidade, "Rio");
}
```

# Leitura de Strings

- Pode ser utilizada a função scanf com o especificador %s.

```
char cidade [81];
scanf ("%s", cidade );
```

- scanf precisa do endereço de memória onde os dados lidos serão armazenados (identificador cidade, já se refere ao endereço de memória do primeiro elemento do vetor)
- "%s" para a captura com qualquer caractere de controle (espaço, tabulação, etc)

# Buffer Overflow

- Quando o usuário digitar mais caracteres do que cabem na string
- Solução: função scanf com outros especificadores:

```
int main(){
    char cidade[10];
    scanf("%[^\n]s", cidade);
    printf("%s", cidade);
    return 0;
}
```

- [^\n] diz para a função scanf parar apenas quando encontrar um caractere '\n' (enter).
- Podem ser utilizados outros caracteres: [^abc\n] diz para scanf parar apenas ao encontrar um dos caracteres: 'a', 'b', 'c', ou '\n'.

# Leitura de Strings

- Para evitar *buffer overflow,* pode-se utilizar um número após % para indicar o máximo de caracteres a serem lidos:

```
int main(){
    char cidade[10];
    scanf("%9[^\n]s", cidade);
    printf("%s", cidade);
    return 0;
}
```

Lê no máximo 9 caracteres digitados, pois deve deixar espaço para o '\0'.

- %9[^\n] diz para a função scanf parar apenas quando encontrar um caractere '\n' (enter) ou quando o limite de 9 caracteres for atingido. Adiciona o '\0' automaticamente ao final da string.
- Outra opção é utilizar a função fgets:

```
int main(){
    char cidade[10];
    fgets(cidade, 9, stdin);
    printf("%s", cidade);
    return 0;
}
```

Diferentemente do scanf, a função fgets inclui o caractere '\n' (enter) digitado pelo usuário ao final da string.

# Leitura de Strings: fflush

- Caracteres digitados pelo usuário e não lidos ficam no buffer do teclado e serão entregues para a próxima chamada de leitura.
- Utilize fflush(stdin) para descartar o que ficou no buffer do teclado antes de fazer a próxima leitura. Para garantir que os novos dados digitados pelo usuário é que serão lidos:

```
int main()
{
    char cidade1[10], cidade2[10];

    printf("Cidade1: ");
    scanf("%9[^\n]", cidade1);
    printf("%s\n", cidade1);
    fflush(stdin);
    printf("Cidade1: ");
    scanf("%9[^\n]", cidade2);
    printf("%s\n", cidade2);
    return 0;
}
```

Descarta o que não foi lido pelo primeiro scanf

# Impressão de Strings

- Exemplo de uma função que imprime uma string, caractere a caractere até encontrar o '\0':

```
void imprime(char s[]) {
    int i;
    for (i = 0; s[i] != '\0'; i++){
        printf("%c",s[i]) ;
    }
    printf("\n");
}
```

- Função análoga a:

```
printf("%s\n",s)
```

# Copiando Strings

- Copiar uma string para outra: origem -> destino

```
void copia (char dest[], char orig[] )
{
    int i;
    for (i = 0; orig[i]!='\0'; i++)
    {
        dest[i] = orig[i];
    }
    /* fecha a cadeia copiada */
    dest[i] = '\0';
}
```

- Função strcpy (definida em string.h):

```
strcpy(char* dest, char* orig);
```

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

int main(){
    char s1[] = "ABC";
    char s2[4];
    strcpy(s2,s1);
    printf(":%s:\n",s1);
    return 0;
}
```

# Comparação de Strings (string.h)

- Função strcmp: comparar strings (não é permitido comparar strings com os operadores '<', '>', '==', '<=', e '>=')
  - `strcmp(char *str1,char *str2);`
- Retorna um inteiro positivo se str1 é lexicamente posterior a str2; zero se as duas são idênticas; e negativo se str1 é lexicamente anterior que str2; Ex.:

```
char nome1[20]="Joao da Silva", nome2[20]="Jose Santos";
if (strcmp(nome1,nome2)!=0)
    printf("Os nomes são diferentes"!);
```

# Outras funções em string.h

- strncat: concatena n caracteres de origem para destino:

```
strncat(char *dest,char *origem, int n)
```

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

int main(){
    char s1[20] = "Rio ";
    char s2[20] = "de Janeiro";
    strncat(s1,s2,10);
    printf("%s\n",s1);
    return 0;
}
```

- strncpy: copia n caracteres da origem no destino:

```
strncpy(char *dest,char *origem, int n)
```

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

int main(){
    char s1[20] = "Rio de Janeiro";
    char s2[10];
    strncpy(s2,s1,3);
    printf("%s\n",s2);
    return 0;
}
```

Certifique-se de que a string de destino possui espaço suficiente para caber o que está sendo colocado nela

# Vetor de Strings

- Vetor de Strings equivale a um vetor de vetores
  - Matriz
  - Cada linha da matriz corresponde a uma string
- Útil quando queremos armazenar uma coleção de strings

```
1  /* Exemplos dos slides */
2
3  #include <stdio.h>
4  #define MAX 50
5
6  int main ()
7  {
8      int i , numAlunos ;
9      char alunos[MAX][121] ;
10     do {
11         printf("Digite o numero de alunos: ") ;
12         scanf ("%d",&numAlunos);
13         for (i = 0; i < numAlunos; i++)
14             scanf("%120[^\n]s", alunos[i]) ;//Lê uma string na linha i
15     }
16     while ( numAlunos > MAX );
17     return 0 ;
18 }
```

Cada linha da matriz guarda uma string

# Atividade 1

- Faça um programa que solicita ao usuário digitar o nome e endereço completo (armazenando em duas strings). Em seguida o programa imprime na tela o que foi digitado.

# Atividade 2

- Faça um programa que solicita ao usuário digitar o nome e sobrenome.
- Em seguida o programa solicita ao usuário digitar rua, numero, bairro, cidade (capturando todos os dados como string).
- Finalmente o programa concatena o nome e sobrenome e mostra na tela.
- Depois o programa concatena os dados do endereço e imprime o endereço de uma só vez.

# Atividade 3

- Faca um programa que solicita ao usuário digitar uma mensagem (string). Em seguida o programa converte todos os caracteres da string para maiúsculo e depois imprime os resultados.

# Atividade 4

- Faca um programa que solicita o usuário digitar o nome de 5 pessoas, leia estes nomes em um vetor de strings e, em seguida, imprima o primeiro e o ultimo deles na ordem alfabética.

# Atividade 5

- Faça um programa que solicita ao usuário digitar o nome e sobrenome, concatena-os e imprime primeiro como digitado e depois com todas as letras minúsculas