Linguagens de Programação I

Tema # 8

Strings e Estruturas

Susana M Iglesias

STRINGS - INTRODUÇÃO

- Strings (cadeia de caracteres): é uma serie de caracteres que podem ser tratados como uma unidade simples,
- Numerosas entidades do dia a dia são representados computacionalmente utilizando strings: nomes, endereços, números de telefones, CPF, ...
- Uma string pode incluir caracteres alfanuméricos (letras ou dígitos), e caracteres especiais (+, -, *, _, $, @, #),

STRINGS - INTRODUÇÃO

- Dados strings são utilizados em quase todas as aplicações computacionais (editores de texto, bancos de dados, redes, internet, ...)
- A diferença da maioria das linguagens de programação, a linguagem C não fornece um tipo string como tipo de dado básico. Limitação?
- Na linguagem C uma string é um ponteiro para seu primeiro caractere, i.e. o valor da string é o endereço do primeiro caractere,

STRINGS - INTRODUÇÃO

- Para manipular strings a linguagem C utiliza vetores de tipo caractere,
- STRING = conjunto de caracteres + caractere nulo
- caractere nulo = '\0', é utilizado para indicar o final da string,
- Strings literais ou constantes de strings são escritas em C utilizando aspas duplas:

```
"Paulo P. Silva"
"Rua Alberto Rangel, s/n, Vila Nova, RJ"
"456.789.534.002"
```

DECLARANDO - STRINGS

- Declaração de uma string:

```
char cor[] = "azul";
char cor[] = {'a', 'z', 'u', 'l', '\0'};
char cor[5];
```

- Em C, `'a'` é diferente de `"a"`?

I/O (E/S) USANDO STRINGS

- **printf**

```
char cor[] = "azul";
printf("%s", cor);
```

- **scanf**

```
char cor[5];
printf("Digite uma cor: ");
scanf("%s", cor);
```

- a função **scanf()** lerá os caracteres até que um espaço ou um indicador de nova linha seja encontrado,

I/O (E/S) USANDO STRINGS

- **`puts(string1)`**, imprime os caracteres contidos em **`string1`** seguidos de um caractere nova linha

```
char cor[] = "azul";
puts(cor);
```

- **`gets(string1)`**, obtém caracteres do dispositivo de entrada e os coloca em **`string1`** até que o caractere nova linha seja encontrado, adiciona o caractere nulo no final de **`string1`**.

```
char cor[5];
printf("Digite uma cor: ");
gets(cor);
```

I/O (E/S) USANDO STRINGS

- como as variáveis strings em C são vetores de caracteres, o nome do vetor e o subscrito correspondente pode ser utilizado para acessar cada caractere por separado,

```
char cor[] = "azul";
char ch;
ch = cor[0];
ch = cor[4];
ch = cor[10];
```

I/O (E/S) USANDO STRINGS

- as funções I/O para caracteres podem ser utilizadas para manipular strings,
- **`getchar()`**, obtém um caractere do dispositivo de entrada e retorna seu valor, lembre se ao criar uma string usando **`getchar()`** colocar o caractere **`NULL`** ao final da string,
- **`putchar(c)`**, envia o caractere c para o dispositivo de saída (video),

EXEMPLO 1

```
#define N 10

int main()
{
  char cor1[N];

  int i=0;

  printf("Digite uma cor: ");
  scanf("%s", cor1);
  printf("Cor: %s\n", cor1);

  while(cor1[i]!='\0')
    printf("%c\n", cor1[i++]);

  system("PAUSE");
  return 0;
}
```

```
Digite uma cor: verde
Cor: verde
v
e
r
d
e
Press any key to continue . . .
```

EXEMPLO 2

```
#define N 50

int main()
{
  char frase[N];
  int i=0;

  printf("Digite uma frase: ");
  gets(frase);

  printf("Vc digitou:\n");
  while(frase[i]!='\0')
    if (frase[i++]!=' ')
      putchar(frase[i-1]);
    else
      putchar('\n');
  printf("\n");
  return 0;
}
```

```
Digite uma frase: A vida e bela
Vc digitou:
A
vida
e
bela
Press any key to continue . . .
```

BIBLIOTECA DE MANIPULAÇÃO DE CARACTERES (`ctype.h`)

- **`int isdigit(int c)`**, retorna verdadeiro se c for um dígito,
- **`int isalpha(int c)`**, retorna verdadeiro se c for uma letra,
- **`int isalnum(int c)`**, retorna verdadeiro se c for um caractere alfanúmerico (letra ou dígito),
- **`int isspace(int c)`**, retorna verdadeiro se c for um espaço em branco (" ", "\n", "\t")
- **`int islower(int c)`**, retorna verdadeiro se c for uma letra minúscula,

BIBLIOTECA DE MANIPULAÇÃO DE CARACTERES (`ctype.h`)

- **`int isupper(int c)`**, retorna verdadeiro se c for uma letra maiúscula,
- **`int tolower(int c)`**, se c for uma letra maiúscula retorna a minúscula correspondente, senão retorna o mesmo caractere inalterado,
- **`int tolower(int c)`**, se c for uma letra minúscula retorna a maiúscula correspondente, senão retorna o mesmo caractere inalterado,

FUNÇÕES DE CONVERSÃO DE STRINGS

- `int atoi(char s[])`, retorna o valor inteiro representado pela string `s`, se s não representar um valor inteiro gerara um erro,
- `double atof(char s[])`, retorna o valor de ponto flutuante representado pela string `s`, se s não representar um valor de ponto flutuante gerara um erro,

BIBLIOTECA DE MANIPULAÇÃO DE STRINGS (`string.h`)

- **`char * strcpy(char s1[], char s2[])`**, copia a string **`s2`** para o array **`s1`** e retorna um ponteiro a **`s1`**,
- **`char * strcat(char s1[], char s2[])`**, anexa a string **`s2`** ao vetor **`s1`**. O primeiro caractere de **`s2`** é sobrescrito ao caractere **`NULL`** de **`s1`**. O valor de **`s1`** é retornado,
- **`int strcmp(char s1[], char s2[])`**, retorna 0 se a string **`s2`** é igual a string **`s1`**; menor que 0 se **`s1<s2`** e maior que 0 se **`s1>s2`**,

BIBLIOTECA DE MANIPULAÇÃO DE STRINGS (`string.h`)

- `int strlen(char s1[])`, determina o comprimento da string `s1`. Retorna o número de caracteres que antecedem ao caractere `NULL`.
- Existem muitas outras funções de manipulação de strings, pesquise sobre elas.

INTRODUÇÃO

- Estruturas são grupos de variáveis relacionadas entre si sob um nome,
- As estruturas podem conter variáveis de muitos tipos diferentes de dados relacionadas entre si,
- A estrutura é um tipo de dado definido pelo usuário,
- A estrutura é um tipo de dado derivado, elas são construídas utilizando tipos de dados básicos,

INTRODUÇÃO

- As estruturas ou registros são utilizadas para representar entidades que não podem ser representadas por um tipo de dados simples,
- Exemplos:

Aluno (nro. matrícula, idade, nome, sexo, tel, email, CR)

Carro (fabricante, modelo, ano, motor)

DEFINDO UMA ESTRUTURA

```
struct nome_da_estrutura{
   tipo_membro1 nome_membro1;
   tipo_membro2 nome_membro2;
   ...
   tipo_membroN nome_membroN;
};
```

- Estrutura aluno:

```
struct aluno{
   int matricula;
   double CR;
   char sexo;
};
```

DEFININDO UMA ESTRUTURA

- Definir uma estrutura, cria um novo tipo de dados,
- O fato de definir uma estrutura não cria nenhuma variável e nenhum espaço é reservado na memória,
- Geralmente as estruturas são definidas fora de todas as funções, no começo do programa, e utilizadas ao longo de todo o código,
- Uma estrutura é formada pelo rotulo ou nome da estrutura e seus membros.

DEFININDO UMA ESTRUTURA

- Um membro de uma estrutura pode ser um tipo estrutura criado anteriormente,
- um membro de uma estrutura pode ser um ponteiro para o uma variável da mesma estrutura, (estruturas autoreferenciadas, pilhas, filas, listas).

DECLARANDO VARIÁVEIS DE TIPO ESTRUTURA

- Sintaxe:

```
struct nome_estrutura variavel1, variavel2;
```

- Exemplo:

```
struct aluno{
   int matricula;
   double CR;
   char sexo;
};

int main(){
   struct aluno aluno1, alunos[10];
   ...
   return 0;
}
```

DECLARANDO VARIÁVEIS DE TIPO ESTRUTURA

- Outras alternativas:

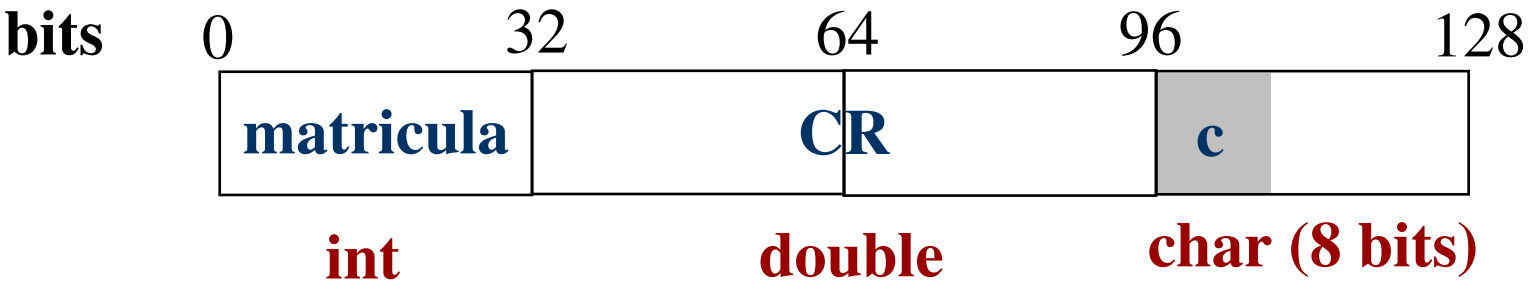
```
struct aluno{
   int matricula;
   double CR;
   char sexo;
}aluno1, alunos[10];
```

```
struct{
   int matricula;
   double CR;
   char sexo;
}aluno1, alunos[10];
```

- **Cuidado!!!**, ao declarar variáveis na própria definição da estrutura poderíamos estar criando variáveis globais.

ESTRUTURAS NA MEMÓRIA

- Representação na memória da estrutura `aluno`.



- Em dependência do sistema de memória (tamanho da palavra, mecanismo de endereçamento) e os tipos membros de nossa estrutura podem existir "buracos" na memória.

ESTRUTURAS NA MEMÓRIA

- As variáveis estrutura não podem ser comparadas utilizando operadores relacionais. Porque?
- Ao comparar uma estrutura devemos utilizar os membros da estrutura. Isto é, duas estruturas são iguais se cada um de seus membros são iguais.

INICIALIZANDO ESTRUTURAS

- As estruturas podem ser inicializadas utilizando uma lista de inicializadores,
- Exemplo:

```
struct aluno alunos[3]={{200501317,8.3,'M'},
                        {200501234,7.1,'F'},
                        {200501453,9.0,'M'}};
```

ACESSO A MEMBROS DE ESTRUTURAS

- Sintaxe:

  nome_estrutura.nome_membro;

- Exemplo:

```
struct aluno aluno1;
scanf("%d", aluno1.matricula);
printf("%f", aluno1.CR)
```

CRIAÇÃO DE SINONIMOS (`typedef`)

- A palavra-chave `typedef` fornece um mecanismo para a criação de sinônimos ou alias para tipos de dados definidos previamente,
- O uso de `typedef`, facilita a compreensão dos tipos de dados criados pelo programador,
- As estruturas são freqüentemente definidas usando `typedef` para criar nomes mais curtos e intuitivos,

CRIAÇÃO DE SINONIMOS (`typedef`)

- Sintaxe: ***`typedef Tipo_de_Dado alias;`***
- Exemplo:

```
struct aluno{
   int matricula;
   double CR;
   char sexo;
};

typedef struct aluno Aluno;

int main(){
   Aluno aluno1, alunos[10];
   ...
   return 0;}
```

CRIAÇÃO DE SINONIMOS (`typedef`)

- Geralmente ao criarmos estruturas usando typedef o nome da estrutura é omitido,
- Exemplo:

```
typedef struct {
   int matricula;
   double CR;
   char sexo;
} Aluno;

int main(){
   Aluno aluno1, alunos[10];
   ...
   return 0;}
```

EXEMPLO 1

- Que faz o seguinte programa?

```
#define N 3

typedef struct{
  int matricula;
  double CR;
} Aluno;

int main()
{
  Aluno alunos[N]={{200501317,8.3},
                   {200501234,7.1},
                   {200501453, 9}};
  int i;

  for(i=0;i<N;i++)
    printf("Aluno: %d Matricula: %10d Rendimento: %3.2f\n",
            i,
            alunos[i].matricula,
            alunos[i].CR);

  system("PAUSE");
  return 0;
}
```

```
Aluno: 0 Matricula:  200501317 Rendimento: 8.33
Aluno: 1 Matricula:  200501234 Rendimento: 7.10
Aluno: 2 Matricula:  200501453 Rendimento: 9.00
```

EXEMPLO 2

- Que faz o seguinte programa?

```
typedef struct{
  char  matricula[10];
  float CR;
  char sexo;
} Aluno;

int main()
{
  Aluno aluno;

  printf("Digite matricula: ");
  gets(aluno.matricula);
  printf("Digite sexo (M ou F): ");
  scanf("%c", &aluno.sexo);
  printf("Digite CR: ");
  scanf("%f", &aluno.CR);
  printf("\nDados\nMatricula: %s\nRendimento: %3.2f\nSexo: %c\n",
         aluno.matricula,
         aluno.CR,
         aluno.sexo);

  system("PAUSE");
  return 0;}
```

```
Digite matricula: 200502415
Digite sexo (M ou F): M
Digite CR: 7.3

Dados
Matricula: 200502415
Rendimento: 7.30
Sexo: M
```

EXEMPLO 3

- *Crie um programa que receba os dados de 5 alunos (nro matricula, nome, idade, coeficiente de rendimento). Imprima o nome do melhor aluno e a matricula do aluno mais jovem.*

```
#define N 5

typedef struct{
  int matricula;
  char nome[15];
  int idade;
  float CR;
}Taluno;

int main(){
  Taluno alunos[N]; int melhor=0, jovem=0, i;

  for(i=0;i<N;i++){
    printf("\nDados do aluno %d\n", i+1);
    printf("Digite a matricula: ");
    scanf("%d", &alunos[i].matricula);
    fflush(stdin);
    printf("Digite o nome: ");
    gets(alunos[i].nome);
    printf("Digite a idade: ");
    scanf("%d", &alunos[i].idade);
    printf("Digite o coef. de rendimento: ");
    scanf("%f", &alunos[i].CR);
    if (alunos[melhor].CR<alunos[i].CR) melhor = i;
    if (alunos[jovem].idade>alunos[i].idade) jovem = i;
}
```

```
  printf("O melhor aluno e %s\n", alunos[melhor].nome);
  printf("A matricula do mais jovem e %d\n",
          alunos[jovem].matricula);

  system("PAUSE");
  return 0;
}
```

EXEMPLO 4

- *Crie um programa que leia uma lista de alunos (nro de matricula e coeficiente de rendimento), e imprima a lista de alunos ordenados segundo o rendimento.*

```
#define N 5

typedef struct{
  int matricula;
  float CR;
}Aluno;

int main()
{
  Aluno alunos[N];
  int orden[N], i, j, buff, n;

  printf("Digite a quantidade de alunos");
  scanf("%d", &n);
  if (n>N){
    printf("Erro!!! Estouro de memória\n");
    system("PAUSE");
    return -1;
  }
```

```
  printf("Entrada de Dados\n");
  for(i=0;i<n;i++){
    orden[i]=i;
    printf("Digite matricula: ");
    scanf("%d", &alunos[i].matricula);
    printf("Digite coeficente de rendimento: ");
    scanf("%f", &alunos[i].CR);
  }
  for(i=0;i<n;i++){
    for(j=0;j<n-1;j++)
      if (alunos[orden[j]].CR>alunos[orden[j+1]].CR){
        buff = orden[j];
        orden[j] = orden[j+1];
        orden[j+1] = buff;
       }
  }
  printf("\nSaida de dados\n");
  for(i=0;i<n;i++)
    printf("Matricula %10d   Rendimento %3.1f\n",
            alunos[orden[i]].matricula,
            alunos[orden[i]].CR);
return 0;}
```

```
Entrada de Dados
Digite matricula: 200500001
Digite coeficente de rendimento: 6.3
Digite matricula: 200500002
Digite coeficente de rendimento: 5.2
Digite matricula: 200500003
Digite coeficente de rendimento: 4.1
Digite matricula: 200500004
Digite coeficente de rendimento: 9.2
Digite matricula: 200500005
Digite coeficente de rendimento: 7.1

Saida de dados
Matricula  200500003   Rendimento 4.1
Matricula  200500002   Rendimento 5.2
Matricula  200500001   Rendimento 6.3
Matricula  200500005   Rendimento 7.1
Matricula  200500004   Rendimento 9.2
```

FUNÇÕES E ESTRUTURAS

- Como qualquer outro valor do tipo **`int`** ou **`float`**, valores de estruturas podem ser passados como argumentos para funções, e podem ser retornados de funções.
- Exemplo:

```
typedef struct {
  int membro1;
  float membro2;
} Struct1;
```

FUNÇÕES E ESTRUTURAS

- Exemplo...

```
//retorna um valor estrutura
Struct1 lee_estrutura(void);
//recebe um valor estrutura
void prn_estrutura(Struct1);

int main(){
   Struct1 var1;

   var1 = lee_estrutura();
   prn_estrutura(var1);
   return 0;
}
```

FUNÇÕES E ESTRUTURAS

- Exemplo...

```
//função que retorna um valor
//estrutura
Struct1 lee_estrutura(void){
  Struct1 var2;
  printf("Digite inteiro:");
  scanf("%d", &var2.membro1);
  printf("Digite float:");
  scanf("%f", &var2.membro2);

  return var2;
}
```

FUNÇÕES E ESTRUTURAS

- Exemplo...

```
//função que recebe como
//parâmetro um valor estrutura
void prn_estrutura(Struct1 var3){
  printf("***ESTRUTURA***\n");
  printf("Valor inteiro: %d\n",
          var3.membro1);
  printf("Valor float: %f\n",
          var3.membro2);
}
```