Programação de Computadores I

Prof. Rafael Nunes

A Linguagem C

Parte 3

Vamos *Recordar*...

Argumentos e
Retornando Valores de funções

```c
#include <stdio.h>
int prod (int x,int y)
{
    return (x*y);
}

int main ()
{
    int saida;
    saida = prod(12,7);
    printf ("A saida e: %d\n",saida);
    return 0;
}
```

Introdução às Entradas e Saídas

Caracteres

- Os caracteres são um *tipo* de dado:
  - o ***char***

- O C trata os caracteres como sendo ***variáveis de um byte*** (8 bits).

- Um ***bit*** é a **menor unidade** de ***armazenamento de informações*** em um computador.

Inteiros

- Os inteiros (ints) têm um número maior de bytes
- Dependendo da implementação do compilador, eles podem ter ***2 bytes*** (16 bits) ou ***4 bytes*** (32 bits)
- Isto será melhor explicado **mais tarde**

Caracteres – cont...

- Na linguagem C, também podemos ***usar um char*** para ***armazenar valores numéricos inteiros***
  - além de usá-lo para armazenar caracteres de texto.
- Para indicar um caractere de texto usamos ***apóstrofes***.
- Veja um exemplo de programa que usa caracteres...

Caracteres - Exemplo

```
#include <stdio.h>
int main (  )
{
   char Ch;
   Ch = 'D';
   printf ("%c",Ch);
   return 0;
}
```

Caracteres

- No programa anterior, **%c** indica que printf() deve colocar um ***caractere*** na tela.
- Como vimos, um char também pode usado para armazenar um ***número inteiro***.
- Este número é conhecido como o ***código ASCII*** correspondente ao caractere.
- Veja o próximo programa...

Execute o programa na ferramenta e veja o resultado

```
#include <stdio.h>
int main ()
{
    char Ch;
    Ch='D';
    printf ("%d",Ch);
    return(0);
}
```

Caracteres

- Muitas vezes queremos ***ler um caractere*** fornecido pelo **usuário**.

- Para isto as funções mais usadas, quando se está trabalhando em ambiente DOS ou Windows, são ***getch()*** e ***getche()***

Caracteres

- Ambas retornam o caractere pressionado.
  - ***getche( )*** imprime o caractere na tela antes de retorná-lo
  - ***getch( )*** apenas retorna o caractere pressionado sem imprimí-lo na tela.
- Ambas as funções podem ser encontradas no arquivo de cabeçalho ***conio.h***

Caracteres

- Geralmente estas funções ***não*** estão disponíveis em ambiente **Unix** (compiladores cc e gcc)

- Elas podem ser substituídas pela função ***scanf(),*** porém sem as mesmas funcionalidades.

Vejamos um exemplo

Utilização da função getch(), e de seu correspondente no

ambiente Unix

Windows

```
#include <stdio.h>
#include <conio.h>
int main ()
{
    char Ch;
    Ch=getch();
    printf ("Voce pressionou
    a tecla %c",Ch);
    return(0);
}
```

UNIX

```
#include <stdio.h>
int main ()
{
    char Ch;
    scanf("%c", &Ch);
    printf ("Voce pressionou
    a tecla %c",Ch);
    return(0);
}
```

Caracteres

- A principal diferença da versão que ***utiliza*** getch() para a versão que ***não utiliza*** getch() é que no primeiro caso o usuário simplesmente aperta a tecla e o sistema ***lê diretamente a tecla pressionada***.

- No segundo caso, é ***necessário*** apertar também a tecla <ENTER>.

String

- No C uma string é um *vetor de caracteres* terminado com um ***caractere nulo***.
- O caracter nulo é um caractere com valor inteiro ***igual a zero***
  - (código ASCII igual a 0).
- O terminador nulo também pode ser escrito usando a convenção de barra invertida do C como sendo **'\0'**

String

- Discutiremos ***Vetores*** **mais tarde**
- Veremos aqui os fundamentos necessários para que ***possamos utilizar*** as strings.
- Para ***declarar*** uma string podemos usar o seguinte formato geral:

```
char nome_da_string[tamanho];
```

String

char *nome_da_string*[**tamanho**];

- Isto declara um vetor de caracteres (uma string) com número de ***posições*** igual a ***tamanho***.

- Note que, como ***temos que reservar um caractere*** para ser o ***terminador nulo***, temos que declarar o comprimento da string como sendo, no mínimo, um ***caractere maior*** que a maior string que pretendemos armazenar.

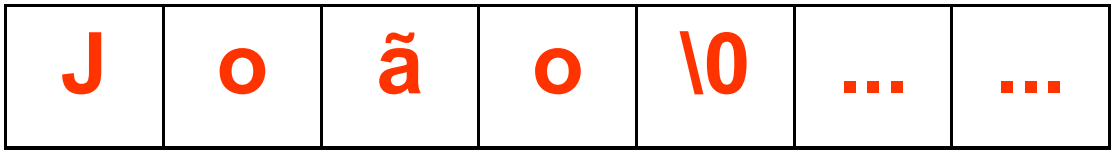
String

- Vamos supor que declaremos uma string de 7 posições e coloquemos a palavra ***João*** nela.

- Teremos:

J	**o**	**ã**	**o**	**\0**	**...**	**...**

String



- No caso acima, as duas células não usadas têm valores indeterminados.

- Isto acontece porque o C não inicializa variáveis, cabendo ao programador esta tarefa.

- Se quisermos ler uma string fornecida pelo usuário podemos usar a ***função gets().***

String

- Um exemplo do uso desta função é apresentada a seguir.

- Repare que a ***função gets()*** coloca o ***terminador nulo*** na string quando você aperta a tecla **"Enter"**

String

```
#include <stdio.h>
int main ()
{
    char string[100];
    printf ("Digite uma string: ");
    gets (string);
    printf ("\n\nVoce digitou %s",string);
    return(0);
}
```

String

- Neste programa, o ***tamanho máximo*** da string que você pode entrar é uma string de ***99 caracteres***.
- Se você entrar com uma string de comprimento maior, o programa irá aceitar, mas os resultados podem ser **desastrosos**.
- Veremos porque **mais tarde**

String

- Como as **strings** são ***vetores de caracteres***, para se acessar um determinado caracter de uma string, basta "***indexarmos***", ou seja, usarmos ***um índice*** para acessarmos o caractere desejado dentro da string.

- Como assim professor?...

String

- Suponha uma string chamada

***str[ ]* = “rafael”**

- Podemos acessar a segunda letra de str da seguinte forma:

str[**1**] = 'a';

Por quê estamos acessando a *segunda letra* e não a primeira?

String

- Na linguagem C, o índice ***começa*** em ***zero***.

- Assim, a primeira letra da string sempre estará na ***posição 0***.

- A segunda letra sempre estará na ***posição 1*** e assim sucessivamente.

String

- Segue um exemplo que imprimirá a ***segunda letra*** da string “Rafael” vista anteriormente

- Em seguida, ele ***mudará*** uma letra e apresentará a ***nova string*** no final

String

```
#include <stdio.h>
int main()
{
    char str[10] = “Rafael";
    printf("\n\nString: %s", str);
    printf("\nSegunda letra: %c", str[1]);
    str[1] = 'U';
    printf("\nAgora a segunda letra eh: %c", str[1]);
    printf("\n\nString resultante: %s", str);
    return(0);
}
```

String

- Nesta string, o ***terminador nulo*** está na ***posição 6***.
- Das posições ***0 a 6***, sabemos que temos ***caracteres válidos***, e portanto podemos escrevê-los.
- Note a forma como inicializamos a string str com os caracteres ‘R’ ‘a’ ‘f’ ‘a’ ‘e’ ‘l’ e **‘\0’** simplesmente declarando

char str[10] = “Rafael”

String

- Veremos, posteriormente que “Rafael” (***uma cadeia de caracteres entre aspas***) é o que chamamos de string constante!
- Uma cadeia de caracteres que está pré-carregada com valores que ***não podem ser modificados***.
- Já a string ***str*** é uma ***string variável***, pois ***podemos modificar*** o que nela está armazenado, como de fato fizemos.

No programa anterior, *%s* indica que *printf( )* deve colocar uma *string* na tela.

Vamos agora fazer uma abordagem inicial às duas funções que já utilizamos para fazer a *entrada e saída*

Printf( )

- A função printf() tem a seguinte forma geral:

printf (***string_de_controle***,*lista_de_argumentos*);

- Teremos, na ***string de controle***, uma descrição de tudo que a função vai colocar na tela.

Printf( )

- A ***string de controle*** mostra não apenas os caracteres que devem ser colocados na tela, mas também ***quais as variáveis*** e suas ***respectivas posições***.

- Isto é feito usando-se os ***códigos de controle***, que usam a notação **%**

Printf( )

- Na ***string de controle*** indicamos quais, de ***qual tipo*** e em ***que posição*** estão as ***variáveis*** a serem apresentadas.

- É muito importante que, para cada ***código de controle***, tenhamos um ***argumento*** na lista de argumentos.

Printf( )

- Apresentamos agora alguns dos ***códigos %:***

Código	Significado
%d	Inteiro
%f	Float
%c	Caractere
%s	String
%%	Mostra um % na tela

Alguns exemplos de printf()...

Printf( )

- printf ("Teste ***%% %%***")
  - -> *"Teste % %"*
- printf (" ***%f*** ",40.345)
  - -> *"40.345"*
- printf ("Um caractere ***%c*** e um inteiro ***%d***",'D',120)
  - -> *"Um caractere D e um inteiro 120"*
- printf ("***%s*** e um exemplo","Este")
  - -> *"Este e um exemplo"*
- printf ("***%s%d%%***","Juros de ",10)
  - -> *"Juros de 10%"*

Scanf( )

- O formato geral da função scanf() é:

  scanf (string-de-controle,lista-de-argumentos);

- Usando a função scanf() podemos pedir dados ao usuário (Já visto anteriormente)

Scanf( )

- Devemos ficar atentos a fim de colocar o mesmo número de argumentos que o de códigos de controle na string de controle.
- Outra coisa importante é lembrarmos de colocar o & antes das variáveis da lista de argumentos.
- É impossível justificar isto agora, mas veremos depois a razão para este procedimento.

Exercícios

- 01) Escreva um programa que leia um caractere digitado pelo usuário, imprima o caractere digitado e o código ASCII correspondente a este caractere.

- 02) Escreva um programa que leia duas strings e as coloque na tela. Imprima também a segunda letra de cada string.

Até a próxima...