Ciência da Computação
Programação

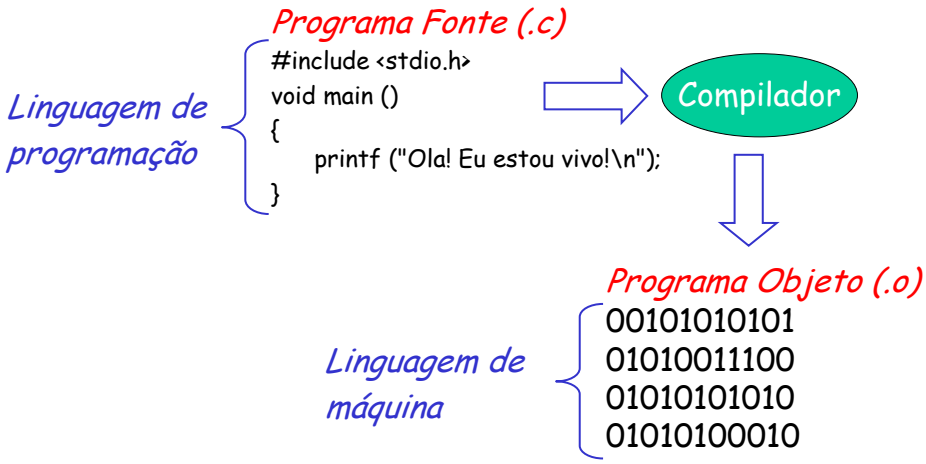
Prof. Sérgio Yunes
syunes@gmail.com

Livro-Texto:

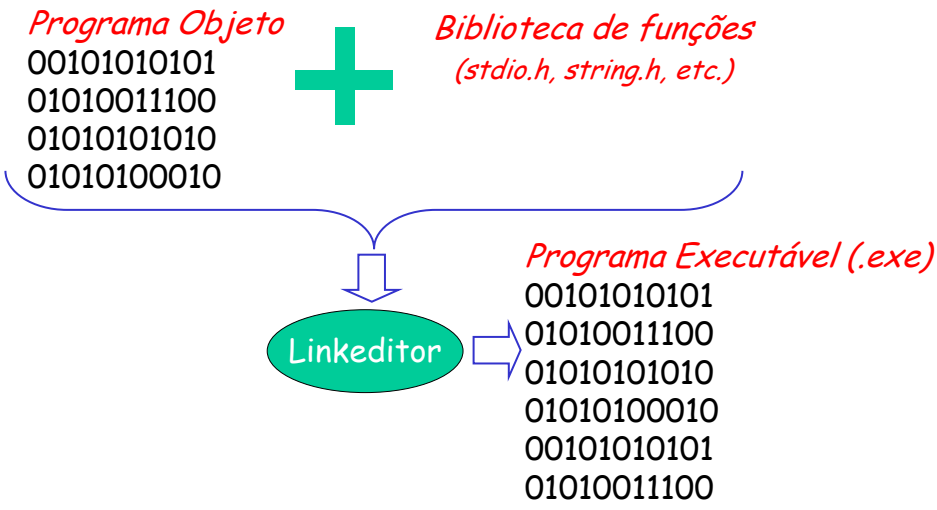
SCHILDT, H. C Completo e Total. 3a. Edição – Makron Books.

Curso de Linguagem C
UFMG
(Material em HTML)

Processo de compilação



Processo de compilação



Processo de compilação

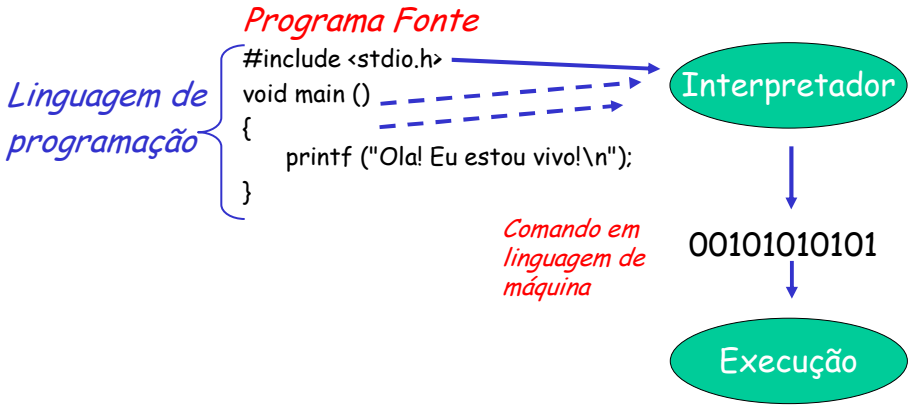
- *Compilador* - recebe um programa escrito numa linguagem de programação (programa fonte) com entrada e fornece como saída um programa escrito em uma linguagem de máquina.
  - É comum referir-se também ao programa fonte como código fonte e ao programa em linguagem de máquina como código executável
  - Exemplos de linguagens compiladas:
  - C, C++, Pascal, Delphi

Processo de compilação

Assim, o editor de textos produz o arquivo fonte .c, que passa pelo compilador, que por sua vez produz arquivos objeto .obj, que passam pelo linkeditor, que produz o arquivo executável .exe.

Você pode então executar os arquivos .exe simplesmente digitando seus nomes no prompt dos ou usando o comando executar do Windows.

Processo de interpretação



Processo de interpretação

Interpretador – traduz para linguagem de máquina os comandos do programa fonte um a um, executando-os em seguida

- A interpretação de um programa fonte não gera programa objeto (programa executável)
- Exemplos de linguagens interpretadas:
- Perl, JavaScript

Compilação X Interpretação

Programa Executável

- C: Gera programa executável
- I: Não gera programa executável

Execução do programa

- C: Programa executável é independente
- I: Depende do programa fonte e do interpretador

Erros de sintaxe

- C: Programa só é executado quando está livre de erros
- I: Erros podem ser encontrados durante a execução

Introdução a Linguagem C

- C nasceu na década de 70. Seu inventor, Dennis Ritchie
- Linguagem de programação genérica que é utilizada para a criação de programas diversos como:
  - processadores de texto, planilhas eletrônicas,
  - sistemas operacionais, programas para a automação industrial,
  - gerenciadores de bancos de dados, programas de projeto assistido por computador, etc ...

Primeiros Passos

- O C é case sensitive: isto é, maiúsculas e minúsculas fazem diferença.
- Assim, Soma, SOMA, SoMa ou sOmA são diferentes

Vejamos um primeiro programa em C:

```
#include <stdio.h>
  void main ()    /* Um Primeiro Programa */
  {
     printf ("Ola! Eu estou vivo!\n");
  }
```

- Compilando e executando este programa você verá que ele coloca a mensagem *Ola! Eu estou vivo!* na tela.

Vamos analisar o programa

#include <stdio.h>

- inclui o arquivo-cabeçalho **stdio.h**.
- Neste arquivo existem declarações de funções úteis para entrada e saída de dados (stdio = Entrada e saída padronizadas).

/* Um Primeiro Programa */

- comentários que ajudam a elucidar o funcionamento do programa
- O compilador C desconsidera qualquer coisa que esteja começando com **/*** e terminando com ***/**.

Vamos analisar o programa

void main()

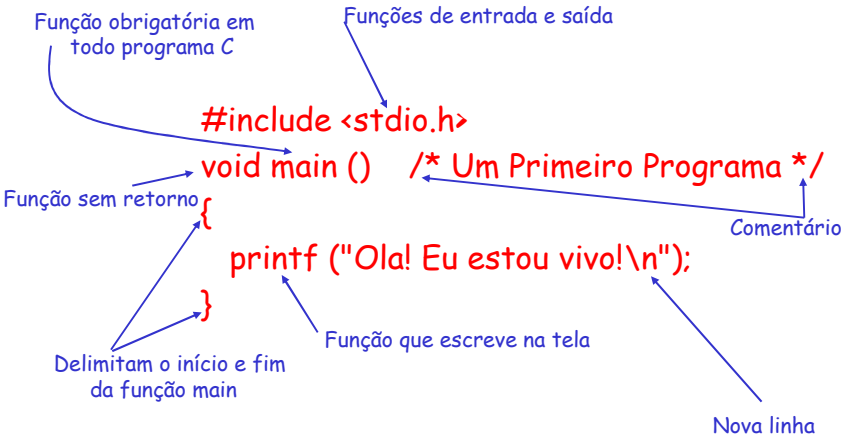
- Função de nome **main**. Todos os programas em C têm que ter uma função **main**, pois é esta função que será chamada quando o programa for executado.
- O conteúdo da função é delimitado por chaves **{ }**.
- A palavra *void* indica que esta função não retorna nada. Posteriormente, veremos que as funções em C podem retornar valores.

Vamos analisar o programa

printf ("Ola! Eu estou vivo!\n");

- O programa chama a função **printf()**, passando a string **"Ola! Eu estou vivo!\n"** como argumento.
- É por causa do uso da função **printf()** que devemos incluir o arquivo- cabeçalho **stdio.h** . A função **printf()** neste caso irá apenas colocar a string na tela do computador.
- **\n** é a constante barra invertida de "new line" e ele é interpretado como um comando de mudança de linha, isto é, após imprimir *Ola! Eu estou vivo!* o cursor passará para a próxima linha. É importante observar também que os *comandos* do C terminam com **;** .

Vamos analisar o programa



Um outro programa

```
#include <stdio.h>
void main ()
{
     int Dias;      /* Declaracao de Variaveis */
     float Anos;
     printf ("Entre com o número de dias: "); /* Entrada de Dados */
     scanf("%d",&Dias);
     Anos=Dias/365.25;                  /* Conversao Dias->Anos */
     printf ("\n\n%d dias equivalem a %f anos.\n",Dias,Anos);
}
```

Um outro programa

```
int Dias;       /* Declaracao de Variaveis */
float Anos;
printf ("Entre com o número de dias: "); /* Entrada de
   Dados */
```

- São declaradas duas variáveis chamadas **Dias** e **Anos**. A primeira é um **int** (inteiro) e a segunda um **float** (ponto flutuante).
- É feita então uma chamada à função **printf()**, que coloca uma mensagem na tela.

Um outro programa

scanf("%d",&Dias);

- Queremos agora ler um dado que será fornecido pelo usuário e colocá-lo na variável **Dias**. Para tanto usamos a função **scanf()**.
- A string **"%d"** diz à função que iremos ler um inteiro. O segundo parâmetro passado à função diz que o dado lido deverá ser armazenado na variável **Dias**.
- É importante ressaltar a necessidade de se colocar um **&** antes do nome da variável a ser lida quando se usa a função **scanf()**.
- Observe que, no C, quando temos mais de um parâmetro para uma função, eles serão separados por vírgula.

Um outro programa

```
Anos=Dias/365.25;     /* Conversao Dias->Ano*/
printf ("\n\n%d dias equivalem a %f anos.\n",Dias,Anos);
```

- Temos então uma expressão matemática simples que atribui a **Anos** o valor de **Dias** dividido por 365.25.

- A função **printf()** tem três argumentos:
- A string **"\n\n%d dias equivalem a %f anos.\n"** diz à função para passar para a próxima linha e depois para a subseqüente, colocar um inteiro na tela, colocar a mensagem **" dias equivalem a "**, colocar um valor **float** na tela, colocar a mensagem **" anos."** e pular mais uma linha.

- Os outros parâmetros são as variáveis das quais devem ser lidos os valores do **inteiro** e do **float**, respectivamente.

Auto Avaliação

Veja como você está. O que faz o seguinte programa?

```
#include <stdio.h>
 main()
 {
     int x;
      scanf("%d",&x);
      printf("%d",x);
 }
```

Introdução a Funções

- Uma função é um bloco de código de programa que pode ser usado diversas vezes em sua execução
- Um programa em C consiste, no fundo, de várias funções colocadas juntas
- Abaixo um exemplo simples de função:

```
#include <stdio.h>
void mensagem ( ) /* Funcao simples: so imprime Ola!*/
{
 printf ("Ola! ");
}
void main ( )
{
   mensagem( );
   printf ("Eu estou vivo!\n");
 }
```


Introdução a Funções

- Este programa terá o mesmo resultado que o primeiro exemplo. O que ele faz é definir uma função mensagem( ) que coloca uma string na tela e não retorna nada (void).
- Depois esta função é chamada a partir de main() (que também é uma função).

Argumentos de Funções

- Argumentos são as entradas que a função recebe. É através dos argumentos que passamos *parâmetros* para a função.
- As funções printf() e scanf() são funções que recebem argumentos.

```
#include <stdio.h>
void square (int x) /* Calcula o quadrado de x */
{
  printf ("O quadrado e %d",(x*x));
}
void main ()
{
  int num;
  printf ("Entre com um numero: ");
  scanf ("%d",&num);
  printf ("\n\n");
  square(num);
}
```


Argumentos de Funções

- Na definição de square() a função receberá um argumento inteiro x. Quando fazemos a chamada à função, o inteiro num é passado como argumento.
- Há alguns pontos a observar:
- Temos de satisfazer aos requisitos da função quanto ao tipo e à quantidade de argumentos quando a chamamos.
- A variável num, ao ser passada como argumento para square() é copiada para a variável x. Dentro de square() trabalha-se apenas com x. Se mudarmos o valor de x dentro de square() o valor de num na função main() permanece inalterado.

Funções com mais de 1 argumento

- Neste caso, os argumentos são separados por vírgula e deve-se explicitar o tipo de cada um dos argumentos, um a um.
- Os argumentos passados para a função não necessitam ser todos variáveis porque mesmo sendo constantes serão copiados para a variável de entrada da função.

```
#include <stdio.h>
void mult (float a, float b,float c) /* Multiplica 3 números*/
{
   printf ("%f",a*b*c);
}
void main ()
{
  float x,y;
  x=23.5;
  y=12.9;
  mult (x,y,3.87);
}
```


Retornando valores

- As funções que vimos até aqui não retornam nada, pois especificamos um retorno void.
- Podemos especificar um tipo de retorno indicando-o antes do nome da função.
- Mas para dizer ao C que vamos retornar precisamos da palavra reservada return.

```
#include <stdio.h>
int prod (int x,int y)
{       return (x*y);
}
void main ()
{       int saida;
saida=prod (12,7);
printf ("A saida e: %d\n",saida);
}
```

Forma geral de função

Apresentamos aqui a forma geral de uma função:

```
tipo_de_retorno nome_da_função (lista_de_argumentos)
  {
     código_da_função
  }
```

AUTO AVALIAÇÃO

Veja como você está. Escreva uma função que some dois inteiros e retorne o valor da soma.

Introdução Básica às Entradas e Saídas

- Caracteres
- Strings
- printf
- scanf

Caracteres

- Os caracteres são um tipo de dado: o char. O C trata os caracteres como sendo variáveis de um *byte* (8 *bit*s).
- Para indicar um caractere de texto usamos apóstrofes. Veja um exemplo de programa que usa caracteres:

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

- No programa acima, %c indica que printf() deve colocar um caractere na tela.

Getch() e Getche()

- Permite ler um caractere fornecido pelo usuário
- getche(): imprime o caractere na tela antes de retorná-lo
- getch(): retorna o caractere pressionado sem imprimí-lo na tela.
- Funções encontradas no arquivo de cabeçalho conio.h.

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

Strings

- No C uma string é um vetor (conjunto) de caracteres terminado com um caractere nulo representado por \0'.
- Para declarar uma string podemos usar o seguinte formato geral:

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 caracter nulo, temos que declarar o tamanho da string como sendo, no mínimo, um caractere maior que a maior string que pretendemos armazenar.

Strings

- *char nome[7];*
- Vamos supor que declaremos uma string de 7 posições e coloquemos a palavra João nela. Teremos:

J	O	A	O	\0	...	...

- No caso acima, as duas células não usadas têm valores indeterminados. Isto acontece porque o C *não* inicializa variáveis, cabendo ao programador esta tarefa.

Strings

- Para ler uma string fornecida pelo usuário podemos usar a função gets().
- A função gets() coloca o caracter nulo na string, quando você aperta a tecla "Enter".

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


Strings

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

- No programa acima, 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 posteriormente.

Strings

- Como as strings são vetores de caracteres para se acessar um determinado caracter basta usarmos um índice para acessarmos o caracter desejado dentro da string.

- Suponha uma string chamada *str*. Podemos acessar a **segunda** letra de *str* da seguinte forma:

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


Strings

- Segue um exemplo que imprimirá a segunda letra da string "Joao", apresentada acima. Em seguida, ele mudará esta letra e apresentará a string no final.

```
#include <stdio.h>
   int main()
   {
      char str[10] = "Joao";
      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);
   }
```

Strings

- Na string str, o terminador nulo está na posição 4. Das posições 0 a 4, sabemos que temos caracteres válidos, e portanto podemos escrevê-los
- Note a forma como inicializamos a string **str** com os caracteres 'J' 'o' 'a' 'o' e '\0' simplesmente declarando char str[10] = "Joao"
- No programa acima, **%s** indica que **printf()** deve colocar uma string na tela

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.
- 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 **%**. argumentos.

printf

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

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

A3

printf

- Vamos ver alguns exemplos de printf() e o que eles exibem:

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%"

A3

Código
Formato

%c Um caracter (char)
%d Um número inteiro decimal (int)
%i O mesmo que %d
%e Número em notação científica com o "e"minúsculo
%E Número em notação científica com o "e"maiúsculo
%f Ponto flutuante decimal
%g Escolhe automaticamente o melhor entre %f e %e
%G Escolhe automaticamente o melhor entre %f e %E
%o Número octal
%s String
%u Decimal "unsigned" (sem sinal)
%x Hexadecimal com letras minúsculas
%X Hexadecimal com letras maiúsculas
%% Imprime um %
%p Ponteiro

Vamos ver alguns exemplos:

Administrator; 21/2/2005

A4

scanf

- O formato geral da função **scanf()** é:

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

- Usando a função **scanf()** podemos pedir dados ao usuário
- Mais uma vez, 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

A5

scanf

AUTO AVALIAÇÃO

Veja como você está:

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

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

Slide 41

A4

Código
Formato

%c Um caracter (char)
%d Um número inteiro decimal (int)
%i O mesmo que %d
%e Número em notação científica com o "e"minúsculo
%E Número em notação científica com o "e"maiúsculo
%f Ponto flutuante decimal
%g Escolhe automaticamente o melhor entre %f e %e
%G Escolhe automaticamente o melhor entre %f e %E
%o Número octal
%s String
%u Decimal "unsigned" (sem sinal)
%x Hexadecimal com letras minúsculas
%X Hexadecimal com letras maiúsculas
%% Imprime um %
%p Ponteiro

Vamos ver alguns exemplos:

Administrator; 21/2/2005

Slide 42

A5

Código
Formato

%c Um caracter (char)
%d Um número inteiro decimal (int)
%i O mesmo que %d
%e Número em notação científica com o "e"minúsculo
%E Número em notação científica com o "e"maiúsculo
%f Ponto flutuante decimal
%g Escolhe automaticamente o melhor entre %f e %e
%G Escolhe automaticamente o melhor entre %f e %E
%o Número octal
%s String
%u Decimal "unsigned" (sem sinal)
%x Hexadecimal com letras minúsculas
%X Hexadecimal com letras maiúsculas
%% Imprime um %
%p Ponteiro

Vamos ver alguns exemplos:

Slide 42 (Continuado)

Administrator; 21/2/2005

Estruturas de Controle de Fluxo

A6

Comandos de Controle de Fluxo

- Os comandos de controle de fluxo são aqueles que permitem ao programador alterar a sequência de execução do programa
- Vamos dar uma breve introdução a dois comandos de controle de fluxo
- IF
- FOR

A6

Código
Formato

%c Um caracter (char)
%d Um número inteiro decimal (int)
%i O mesmo que %d
%e Número em notação científica com o "e"minúsculo
%E Número em notação científica com o "e"maiúsculo
%f Ponto flutuante decimal
%g Escolhe automaticamente o melhor entre %f e %e
%G Escolhe automaticamente o melhor entre %f e %E
%o Número octal
%s String
%u Decimal "unsigned" (sem sinal)
%x Hexadecimal com letras minúsculas
%X Hexadecimal com letras maiúsculas
%% Imprime um %
%p Ponteiro

Vamos ver alguns exemplos:

Administrator; 21/2/2005

IF

- O comando if representa uma tomada de decisão do tipo "SE isto ENTÃO aquilo".
- A sua forma geral é:

  if (condição) declaração;

- A condição é uma expressão que será avaliada. Se o resultado for zero a declaração não será executada. Se o resultado for qualquer coisa diferente de zero a declaração será executada.
- A declaração pode ser um bloco de código ou apenas um comando.

IF

```
#include <stdio.h>
int main () {
int num;
printf ("Digite um numero: ");
scanf ("%d",&num);
if (num>10) printf ("\n\nO numero e maior que 10");
if (num==10) {
   printf ("\n\nVoce acertou!\n");
   printf ("O numero e igual a 10.");
}
if (num<10) printf ("\n\nO numero e menor que 10");
return (0);
}
```

IF

- Para testar igualdades usamos o operador == e não =
- O operador = representa *apenas* uma atribuição.Assim,

```
if (num=10) ...      /* Isto esta errado */
```

- O compilador iria *atribuir* o valor 10 à variável num e a expressão num=10 iria retornar 10, fazendo com que o nosso valor de num fosse adulterado e fazendo com que a declaração fosse executada sempre.
- Os operadores de comparação são: == (igual), != (diferente de), > (maior que), < (menor que), >= (maior ou igual), <= (menor ou igual).

FOR

- O loop (laço) for é usado para repetir um comando, ou bloco de comandos, diversas vezes, de maneira que se possa ter um bom controle sobre o loop.
- Sua forma geral é:

for (inicialização;condição;incremento) declaração;

- A declaração no comando for também pode ser um bloco ({ }) e neste caso o ; é omitido.

FOR

- O melhor modo de se entender o loop for é ver de que maneira ele funciona "por dentro". O loop for é equivalente a se fazer o seguinte:

```
inicialização;
if (condição)
    {
    declaração;
    incremento;
    "Volte para o comando if"
    }
```

- **for** executa a inicialização incondicionalmente e testa a condição. Se a condição for F ele não faz mais nada. Se a condição for V ele executa a declaração, o incremento e volta a testar a condição. **For** fica repetindo estas operações até que a condição seja falsa.

FOR

- Abaixo vemos um programa que coloca os primeiros 100 números na tela:

```
#include <stdio.h>
int main () {
  int count;
  for (count=1;count<=100;count=count+1) printf
  ("%d",count);
  return(0);
}
```

FOR

- O programa lê uma string e conta quantos dos caracteres desta string são iguais à letra 'c'

```
#include <stdio.h>
int main () {
      char string[100];    /* String, ate' 99 caracteres */
      int i, cont;
      printf("\n\nDigite uma frase: ");
      gets(string); /* Le a string */
      printf("\n\nFrase digitada: \n%s", string);
      cont = 0;
      for (i=0; string[i] != '\0'; i=i+1)
      {
         if ( string[i] == 'c' )     /* Se for a letra 'c' */
            cont = cont +1;       /* Incrementa o contador de caracteres */
      }
      printf("\nNumero de caracteres c = %d", cont);
      return(0);
   }
```


A9

FOR

AUTO AVALIAÇÃO

Veja como você está.

a) Explique porque está **errado** fazer

if (num=10) ... O que irá acontecer?

b) Escreva um programa que coloque os números de 1 a 100 na tela na ordem inversa (começando em 100 e terminando em 1)

c) Escreva um programa que leia uma string, conte quantos caracteres desta string são iguais a 'a' e substitua os que forem iguais a 'a' por 'b'. O programa deve imprimir o número de caracteres modificados e a string modificada.

A9

Código
Formato

%c Um caracter (char)
%d Um número inteiro decimal (int)
%i O mesmo que %d
%e Número em notação científica com o "e"minúsculo
%E Número em notação científica com o "e"maiúsculo
%f Ponto flutuante decimal
%g Escolhe automaticamente o melhor entre %f e %e
%G Escolhe automaticamente o melhor entre %f e %E
%o Número octal
%s String
%u Decimal "unsigned" (sem sinal)
%x Hexadecimal com letras minúsculas
%X Hexadecimal com letras maiúsculas
%% Imprime um %
%p Ponteiro

Vamos ver alguns exemplos:

Administrator; 21/2/2005

Comentários

- Uso de comentários torna o código do programa mais fácil de se entender.
- Os comentários do C devem começar com **/*** e terminar com ***/**.
- O C padrão não permite comentários aninhados (um dentro do outro), mas alguns compiladores os aceitam.

AUTO AVALIAÇÃO

Veja como você está:

Escreva comentários para os programas dos exercícios já realizados.

Palavras Reservadas do C

- Só podem ser usadas nos seus propósitos originais, isto é, não podemos declarar funções ou variáveis com os mesmos nomes.
- C é "case sensitive": For é diferente de for
- Podemos declarar uma variável For, mas isto não é recomendável pois pode gerar confusão.
- Palavras reservadas do ANSI C:

auto	*double*	*struct*	*int*
break	*else*	*switch*	*long*
case	*enum*	*typedef*	*register*
char	*extern*	*union*	*return*
const	*float*	*unsigned*	*short*
continue	*for*	*void*	*signed*
default	*goto*	*volatile*	*sizeof*
do	*if*	*while*	*static*

Nomes de Variáveis

- Podem ter qualquer nome, entretanto:
  - deve começar com uma letra ou sublinhado (_) e os caracteres subseqüentes devem ser letras, números ou sublinhado (_);
  - não pode ser igual a uma palavra reservada, nem igual ao nome de uma função declarada pelo programador, ou pelas bibliotecas do C.
- Variáveis de até 32 caracteres são aceitas.
- C é "case sensitive" e portanto deve-se prestar atenção às maiúsculas e minúsculas.
- É uma prática tradicional do C, usar letras minúsculas para nomes de variáveis e maiúsculas para nomes de constantes. Isto facilita na hora da leitura do código
- Usando nomes de variáveis em português, evita-se possíveis conflitos com nomes de rotinas encontrados nas diversas bibliotecas, que são em sua maioria absoluta, palavras em inglês

Os tipos do C

- O C tem 5 tipos básicos: **char**, **int**, **float**, **void**, **double**.
- **double:** ponto flutuante com muito mais precisão.
- **Modificadores de tipo** do C: **signed**, **unsigned**, **long** e **short**.
- **float** não se pode aplicar nenhum. **Double** pode-se aplicar apenas o **long**. Os quatro podem ser aplicados a inteiros. A intenção é que **short** e **long** devam prover tamanhos diferentes de inteiros onde isto for prático.
- Numa máquina de 16 bits, **int** provavelmente terá 16 bits. Numa máquina de 32, **int** deverá ter 32 bits.
- Na verdade, cada compilador é livre para escolher tamanhos adequados para o seu próprio hardware
- A única restrição de que **shorts** e **ints** devem ocupar pelo menos 16 bits, **longs** pelo menos 32 bits, e **short** não pode ser maior que **int**, que não pode ser maior que **long**.

Tipos de dados permitidos e seu valores máximos e mínimos em um compilador típico para um hardware de 16 bits.

Tipo	Num de bits	Formato para leitura com scanf	Intervalo	
			Inicio	Fim
char	8	%c	-128	127
unsigned char	8	%c	0	255
signed char	8	%c	-128	127
int	16	%i	-32.768	32.767
unsigned int	16	%u	0	65.535
signed int	16	%i	-32.768	32.767
short int	16	%hi	-32.768	32.767
unsigned short int	16	%hu	0	65.535
signed short int	16	%hi	-32.768	32.767
long int	32	%li	-2.147.483.648	2.147.483.647
signed long int	32	%li	-2.147.483.648	2.147.483.647
unsigned long int	32	%lu	0	4.294.967.295
float	32	%f	3,4E-38	3.4E+38
double	64	%lf	1,7E-308	1,7E+308
long double	80	%Lf	3,4E-4932	3,4E+4932

Programação Sérgio Yunes 57

Declaração e Inicialização de Variáveis

- Variáveis em C devem ser declaradas antes de serem usadas.

  tipo_da_variável lista_de_variáveis;

- As variáveis da lista de variáveis terão todas o mesmo tipo e deverão ser separadas por vírgula.

- Como o tipo default do C é o **int**, quando vamos declarar variáveis **int** com algum dos modificadores de tipo, basta colocar o nome do modificador de tipo. Assim um **long** basta para declarar um **long int**.

Programação Sérgio Yunes 58

Declaração e Inicialização de Variáveis

- Há três lugares nos quais podemos declarar variáveis:
- Fora de todas as funções do programa. São chamadas **variáveis globais** e podem ser usadas a partir de qualquer lugar no programa.
- No início de um bloco de código. Estas variáveis são chamadas **locais** e só têm validade dentro do bloco no qual são declaradas, isto é, só a função à qual ela pertence sabe da existência desta variável, dentro do bloco no qual foram declaradas.
- Na lista de parâmetros de uma função. Estas variáveis são conhecidas apenas pela função onde são declaradas.

Declaração e Inicialização de Variáveis

- Veja o programa abaixo:

```
#include <stdio.h>
int contador;
int func1(int j) {
...
}
int main() {
    char condicao;
    int i;
    for (i=0; ...) {          /* Bloco do for */
        float f2;
            ...
        func1(i);
    }
    ...
    return(0);
}
```

Declaração e Inicialização de Variáveis

- As regras que regem *onde* uma variável é válida chamam-se regras de *escopo* da variável.
- Duas variáveis globais não podem ter o mesmo nome.
- O mesmo vale para duas variáveis locais de uma mesma função.
- Já duas variáveis locais, de funções diferentes, podem ter o mesmo nome sem perigo algum de conflito.
- Podemos inicializar variáveis no momento de sua declaração.

tipo_da_variável nome_da_variável = constante;

Declaração e Inicialização de Variáveis

AUTO AVALIAÇÃO

Veja como você está:

Escreva um programa que declare uma variável inteira global e atribua o valor 10 a ela. Declare outras 5 variáveis inteiras locais ao programa principal e atribua os valores 20, 30, ..., 60 a elas. Declare 6 variáveis caracteres e atribua a elas as letras c, o, e, l, h, o . Finalmente, o programa deverá imprimir, usando todas as variaveis declaradas:

As variáveis inteiras contem os números: 10,20,30,40,50,60

O animal contido nas variáveis caracteres e o coelho

Constantes

- Constantes são valores que são mantidos fixos pelo compilador
- São consideradas constantes, por exemplo, os números e caracteres como 45.65 ou 'n', etc...

Constantes strings

- A string **"Joao"** é na realidade uma constante string. Isto implica, por exemplo, no fato de que **'t'** é diferente de **"t"**, pois **'t'** é um **char** enquanto que **"t"** é uma constante string com dois **chars** onde o primeiro é **'t'** e o segundo é **'\0'**.

Constantes

Constantes de barra invertida

- O C utiliza, para nos facilitar a tarefa de programar, vários códigos chamados códigos de barra invertida.

Código	Significado
\b	Retrocesso ("back")
\f	Alimentação de formulário ("form feed")
\n	Nova linha ("new line")
\r	Retorno de carro ("carriage return")
\t	Tabulação horizontal ("tab")
\"	Aspas
\'	Apóstrofo
\0	Nulo (0 em decimal)
\\	Barra invertida
\v	Tabulação vertical
\a	Sinal sonoro ("beep")
\N	Constante octal (N é o valor da constante)
\xN	Constante hexadecimal (N é o valor da constante)

Operadores Aritméticos e de Atribuição

- A seguir apresentamos a lista dos operadores aritméticos do C:

Operador	Ação
+	Soma (inteira e ponto flutuante)
-	Subtração ou Troca de sinal (inteira e ponto flutuante)
*	Multiplicação (inteira e ponto flutuante)
/	Divisão (inteira e ponto flutuante)
%	Resto de divisão (de inteiros)
++	Incremento (inteiro e ponto flutuante)
--	Decremento (inteiro e ponto flutuante)

Operadores Aritméticos e de Atribuição

O C possui operadores unários e binários.

- Os unários agem sobre uma variável apenas, modificando ou não o seu valor, e retornam o valor final da variável.

+, - (subtração), *, / e %.

- Os binários usam duas variáveis e retornam um terceiro valor, sem alterar as variáveis originais.

- (troca de sinal)

- / (divisão) quando aplicado a variáveis inteiras, nos fornece o resultado da divisão inteira; quando aplicado a variáveis em ponto flutuante nos fornece o resultado da divisão "real".

```
int a = 17, b = 3;
int x, y;
float z = 17. , z1, z2;
x = a / b;
y = a % b;
z1 = z / b;
z2 = a/b;
```

- final da execução destas linhas, os valores calculados seriam x = 5, y = 2, z1 = 5.666666 e z2 = 5.0

Operadores Aritméticos e de Atribuição

- Os operadores de incremento e decremento são unários. O que eles fazem é incrementar ou decrementar, a variável sobre a qual estão aplicados, de 1. Então:

  x++; e x--; são equivalentes a x=x+1; e x=x-1;

- Pré-fixados: incrementam e retornam o valor da variável já incrementada.
- Pós- fixados: retornam o valor da variável sem o incremento e depois incrementam a variável. Então, em:

```
x=23;
y=x++;
```

teremos, no final, **y=23** e **x=24**

```
x=23;
y=++x;
```

teremos, no final, **y=24** e **x=24**.

Operadores Aritméticos e de Atribuição

- O operador de atribuição do C é o =. O que ele faz é pegar o valor à direita e atribuir à variável da esquerda. Além disto ele retorna o valor que ele atribuiu. Isto faz com que as seguintes expressões sejam válidas:

```
x=y=z=1.5;          /* Expressao 1 */
if (k=w) ...   /* Expressao 2 */
```

- A expressão 1 é válida, pois quando fazemos **z=1.5** ela retorna 1.5, que é passado adiante.
- A expressão 2 será verdadeira se **w** for diferente de zero, pois este será o valor retornado por **k=w**. Pense bem antes de usar a expressão dois, pois ela pode gerar erros de interpretação. Você *não* está comparando **k** e **w**. Você está atribuindo o valor de **w** a **k** e usando este valor para tomar a decisão.

Operadores Aritméticos e de Atribuição

AUTO AVALIAÇÃO

Veja como você está:

Diga o resultado das variáveis x, y e z depois da seguinte sequência de operações:

```
int x,y,z;
x=y=10;
z=++x;
x=-x;
y++;
x=x+y-(z--);
```


Operadores Relacionais e Lógicos

- Os operadores relacionais do C realizam ***comparações*** entre variáveis. São eles:

Operador	**Ação**
>	Maior do que
>=	Maior ou igual a
<	Menor do que
<=	Menor ou igual a
==	Igual a
!=	Diferente de

- Operadores relacionais retornam verdadeiro (1) ou falso (0). Para fazer ***operações com valores lógicos*** (verdadeiro e falso) temos ***os operadores lógicos***:

Operador	**Ação**
&&	AND (E)
||	OR (OU)
!	NOT (NÃO)

Operadores Relacionais e Lógicos

- Usando os operadores relacionais e lógicos podemos realizar uma grande gama de testes. A tabela-verdade destes operadores é dada a seguir:

p	q	p AND q	p OR q
falso	falso	falso	falso
falso	verdadeiro	falso	verdadeiro
verdadeiro	falso	falso	verdadeiro
verdadeiro	verdadeiro	verdadeiro	verdadeiro

- Exemplo: No trecho de programa abaixo o if será executado, pois o resultado da expressão lógica é verdadeiro:

```
int i = 5, j =7;
if ( (i > 3) && ( j <= 7)  && ( i != j) )  j++;
      V    AND     V    AND    V  = V
```


Operadores Aritméticos e de Atribuição

AUTO AVALIAÇÃO

Veja como você está:

Diga se as seguintes expressões serão verdadeiras ou falsas:

-> ((10>5)||(5>10))

-> (!(5==6)&&(5!=6)&&((2>1)||(5<=4)))

Expressões

- Expressões são combinações de variáveis, constantes e operadores.

```
Anos=Dias/365.25;
i = i+3;
c= a*b + d/e;
c= a*(b+d)/e; :
```

- Tabela de precedências da linguagem C.

Modeladores (Casts)

- Um modelador é aplicado a uma expressão. Ele *força* a mesma a ser de um tipo especificado. Sua forma geral é:

(tipo)expressão

```
#include <stdio.h>
 int main ()       {
 int num;
float f1, f2,f3;
num=10;
f1= num/7;
printf ("%f",f1);                  // 1.000000
f2=(float) (num/7);
printf ("%f",f2);                  // 1.000000
f3=(float)num/7;
printf ("%f",f3);          // 1.428571
return(0);
}
```

If-else-if

- **if-else-if** é apenas uma extensão da estrutura **if-else**.

```
if (condição_1) declaração_1;
else if (condição_2) declaração_2;
else if (condição_3) declaração_3;
   .
   .
   .
else if (condição_n) declaração_n;
else declaração_default;
```

- Só uma declaração será executada
- A última declaração (default) é a que será executada se todas as condições forem Falsas e é opcional.

If-else-if

```
#include <stdio.h>
int main () {
int num;
printf ("Digite um numero: ");
scanf ("%d",&num);
if (num>10)      printf ("\n\nO numero e maior que 10");
else if (num==10)   {
   printf ("\n\nVoce acertou!\n");
   printf ("O numero e igual a 10.");
} else if (num<10)   printf ("\n\nO numero e menor que 10");
return(0);
}
```

Ifs Aninhados

- O **if** aninhado é simplesmente um **if** dentro da declaração de um outro **if** externo.

```
#include <stdio.h>
int main () {
int num;
printf ("Digite um numero: ");
scanf ("%d",&num);
if (num==10)  {
   printf ("\n\nVoce acertou!\n");
   printf ("O numero e igual a 10.\n");
} else  {
    if (num>10)  {
          printf ("O numero e maior que 10.");
    }else  {
          printf ("O numero e menor que 10.");
    }
}
return(0);
}
```


Operador ?

- Uma expressão como:

```
if (a>0)  b=-150;
else      b=150;
```

- pode ser simplificada usando-se o operador ?:

```
b=a>0?-150:150;
```

- De uma maneira geral expressões do tipo:

```
if (condição) expressão_1;
else  expressão_2;
```

- podem ser substituídas por:

```
condição?expressão_1:expressão_2;
```

Operador ?

- O operador ? é limitado mas pode ser usado para simplificar expressões complicadas. Veja o exemplo de um contador circular :

```
#include <stdio.h>
int main() {
int index = 0,
contador;
char letras[5] = "Joao";
for (contador=0; contador < 1000; contador++) {
   printf("\n%c",letras[index]);
   index=(index==4)? index=0: ++index;
}
}
```

- O nome Joao é escrito na tela verticalmente até a variável contador determinar o término do programa. Enquanto isto a variável index assume os valores 0, 1, 2, 3, 4, 0, 1, ... progressivamente.

Operadores Aritméticos e de Atribuição

AUTO AVALIAÇÃO

Veja como você está:

Altere o último exemplo para que ele escreva cada letra 5 vezes seguidas. Para isto, use um 'if' para testar se o contador é divisível por cinco (utilize o operador %) e só então realizar a atualização em index.

O comando switch

- O comando **switch** é próprio para se testar uma variável em relação a diversos valores pré-estabelecidos. Sua forma geral é:

```
switch (variável)
    {
       case constante_1:
                declaração_1;
                break;
       case constante_2:
                declaração_2;
                break;
       .
       .
       .
       case constante_n:
                declaração_n;
                break;
       default
                declaração_default;
    }
```


O comando switch

- **switch** *não* aceita expressões. Aceita apenas constantes
- A declaração **default** é opcional e será executada apenas se a variável, que está sendo testada, não for igual a nenhuma das constantes.
- O comando **break**, faz com que o **switch** seja interrompido assim que uma das declarações seja executada
- Mas ele não é essencial ao comando **switch**. Se após a execução da declaração não houver um **break**, o programa continuará executando. Isto pode ser útil em algumas situações, mas eu recomendo cuidado.

O comando switch

```
#include <stdio.h>
int main () {
int num;
printf ("Digite um numero: ");
scanf ("%d",&num);
switch (num)        {
    case 9:
          printf ("\n\nO numero e igual a 9.\n");
          break;
    case 10:
          printf ("\n\nO numero e igual a 10.\n");
          break;
    case 11:
          printf ("\n\nO numero e igual a 11.\n");
          break;
    default:
          printf ("\n\nO numero nao e nem 9 nem 10 nem 11.\n");
}
return(0);
}
```


Operadores Aritméticos e de Atribuição

AUTO AVALIAÇÃO

Veja como você está:

Escreva um programa, utilizando os comandos *for* e *switch*, que leia uma string (use gets()) e substitua todos os espaços e tabulações ('\t') por caracteres de nova linha ('\n'). O for deve ser encerrado quando o caracter de final da string ('\0') for encontrado.

For Loop Infinito

- O loop infinito tem a forma:

  for (inicialização; ;incremento) declaração;

- Para interromper um loop como este usamos o comando **break**.

```
#include <stdio.h>
#include <conio.h>
int main () {
int Count;
char ch;
for (Count=1;;Count++)  {
    ch = getch();
    if (ch == 'X') break;
    printf("\nLetra: %c",ch);
}
return(0); }
```


While

- O comando while tem a seguinte forma geral:

while (condição) declaração;

- O while seria equivalente a:

```
if (condição)  {
        declaração;
        "Volte para o comando if"
}
```

```
#include <stdio.h>
int main () {
char Ch;
Ch='\0';
while (Ch!='q') {
    Ch = getch();
}
return(0);
}
```

For e While

AUTO AVALIAÇÃO

Veja como você está:

Faça um programa que inverta uma string: leia a string com gets e armazene-a invertida em outra string. Use o comando for para varrer a string até o seu final.

Refaça o programa acima. Use o comando *while* para fechar o loop.

While

- O comando while tem a seguinte forma geral:

```
while (condição) declaração;
```

- O while é equivalente a:

```
if (condição) {
        declaração;
        "Volte para o comando if"
}
```

```
#include <stdio.h>
int main () {
char Ch;
Ch='\0';
while (Ch!='q') {
   Ch = getch();
}
return(0);
}
```

Do While

- Forma geral:

```
do {
      declaração;
   } while (condição);  // O ponto-e- vírgula final é
     obrigatório
```

- Do While é equivalente a:

```
declaração;
if (condição) "Volta para a declaração"
```

- Ao contrário do **for** e do **while**, garante que a declaração será executada pelo menos uma vez.

Do While

```
#include <stdio.h>
int main () {
int i;
do  {
    printf ("\n\nEscolha a fruta pelo numero:\n\n");
    printf ("\t(1)...Mamao\n");
    printf ("\t(2)...Abacaxi\n");
    printf ("\t(3)...Laranja\n\n");
    scanf("%d", &i);
    } while ( (i<1) || (i>3) );  // garante que o valor digitado pelo usuário seja válido
    switch (i)  {
       case 1:
           printf ("\t\tVoce escolheu Mamao. \n");
           break;
       case 2:
           printf ("\t\tVoce escolheu Abacaxi. \n");
           break;
       case 3:
           printf ("\t\tVoce escolheu Laranja. \n");
           break;
    }
    return(0);
}
```

Break

- Pode quebrar a execução de um comando: **switch**
- Ou interromper a execução de *qualquer* loop: **for**, **while** ou **do while**
- O **break** faz com que a execução do programa continue na primeira linha seguinte ao loop ou bloco que está sendo interrompido.

Continue

- Oposto do **break**. Ele só funciona dentro de um loop.
- **continue** faz o loop pular para a próxima iteração, sem o abandono do loop, ao contrário do que acontecia no comando **break**.

```
#include <stdio.h>
int main() {
int opcao=1;
while (opcao != 3)      {
    printf("\n\n Escolha uma opcao entre 1 e 3: ");
    scanf("%d", &opcao);
    if ( (opcao > 3) || (opcao <1) ) continue;/*Opcao invalida: volta ao inicio do loop */
    switch (opcao)          {
          case 1:  printf("\n --> Primeira opcao..");
                   break;
          case 2: printf("\n --> Segunda opcao..");
                   break;
          case 3: printf("\n --> Abandonando..");
                   break;
    }
 } return(0);
 }
```

Vetores, Matrizes e Strings

Vetores

- Vetores nada mais são que matrizes unidimensionais.
- Vetores e matrizes são caracterizadas por terem todos os elementos pertencentes ao mesmo tipo de dado.

- Para se declarar, forma geral:

tipo_da_variável nome_da_variável [tamanho];

- Quando o C vê uma declaração como esta ele reserva um espaço na memória suficientemente grande para armazenar o número de células especificadas em tamanho. Por exemplo, se declararmos:

```
float exemplo [20];
```

- O C irá reservar 4x20=80 bytes. Estes bytes são reservados de maneira contígua.

Vetores

- Em C a numeração começa sempre em zero
- float exemplo [20]; Para acessá-los vamos escrever:

exemplo[0] exemplo[1] . . .exemplo[19]

- Mas ninguém o impede de escrever:

exemplo[30] exemplo[103] Por quê?

- Porque o C não verifica se o índice que você usou está dentro dos limites válidos. Este é um cuidado que *você* deve tomar.
- Se o programador não tiver atenção com os limites de validade para os índices ele corre o risco de ter variáveis sobreescritas ou de ver o computador travar. Bugs terríveis podem surgir

Vetores

```
#include <stdio.h>
int main () {
int num[100];  /* Declara um vetor de inteiros de 100 posicoes */
int count=0;
int totalnums;
do  {
   printf ("\nEntre com um numero (-999 p/ terminar): ");
   scanf ("%d",&num[count]);
   count++;
} while (num[count-1]!=-999);
totalnums=count-1;
printf ("\n\n\n\t Os números que você digitou foram:\n\n");
for (count=0;count<totalnums;count++)  printf (" %d",num[count]);
return(0);
}
```

Vetores

AUTO AVALIAÇÃO

Veja como você está:

Reescreva o exemplo acima, realizando a cada leitura um teste para ver se a dimensão do vetor não foi ultrapassada. Caso o usuário entre com 100 números, o programa deverá abortar o loop de leitura automaticamente. O uso do Flag (-999) não deve ser retirado.

Manipulação de Strings

```
string1=string2;      /* NAO faca isto */
```

- **Gets**
- A função **gets()** lê uma string do teclado. Sua forma geral é:

```
gets (nome_da_string);
```

- O programa abaixo demonstra o funcionamento da função **gets()**:

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

Strcpy

- Forma geral é: *strcpy (string_destino,string_origem); // string.h*
- Copia a string-origem para a string- destino

```
#include <stdio.h>
#include <string.h>
int main () {
char str1[100], str2[100], str3[100];
printf ("Entre com uma string: ");
gets (str1);
strcpy (str2,str1); /* Copia str1 em str2 */
strcpy (str3,"Voce digitou a string "); /* Copia "Voce digitou a string"
   em str3 */
printf ("\n\n%s%s",str3,str2);
return(0);
}
```


Strcat

- Forma geral é:*strcat (string_destino,string_origem); // string.h*
- *A string de origem permanecerá inalterada e será anexada ao fim da string de destino*

```
#include <stdio.h>
#include <string.h>
int main () {
char str1[100],str2[100];
printf ("Entre com uma string: ");
gets (str1);
strcpy (str2,"Voce digitou a string ");
strcat (str2,str1); /* str2 armazenara' Voce digitou a string + o
   conteudo de str1 */
printf ("\n\n%s",str2);
return(0);
}
```

Strlen

- Forma geral é: *strlen (string); // string.h*

- *Retorna o comprimento da string fornecida. O terminador nulo não é contado.*

```
#include <stdio.h>
#include <string.h>
int main () {
int size;
char str[100];
printf ("Entre com uma string: ");
gets (str);
size=strlen (str);
printf ("\n\nA string que voce digitou tem tamanho %d",size);
return(0);
}
```


Strcmp

- Forma geral é: *strcmp (string1,string2); // string.h*

- Compara a string 1 com a string 2. Se as duas forem idênticas a função retorna zero (V). Se elas forem diferentes a função retorna não-zero (F)

```
#include <stdio.h>
#include <string.h>
int main () {
char str1[100],str2[100];
printf ("Entre com uma string: ");
gets (str1);
printf ("\n\nEntre com outra string: ");
gets (str2);
if  ( strcmp(str1,str2) )
printf ("\n\nAs duas strings são diferentes.");
else printf ("\n\nAs duas strings são iguais.");
return(0);
}
```

Manipulação de Strings

AUTO AVALIAÇÃO

Veja como você está:

Faça um programa que leia quatro palavras pelo teclado, e armazene cada palavra em uma string. Depois, concatene todas as strings lidas numa única string. Por fim apresente esta como resultado ao final do programa.

Matrizes bidimensional

- *A forma geral de declaração:*

tipo_da_variável nome_da_variável [altura][largura];

- *O índice da esquerda indexa as linhas e o da direita indexa as colunas.*

```
#include <stdio.h>
int main () {
int mtrx [20][10];
int i,,j,count; count=1;
for (i=0;i<20;i++)  {
    for (j=0;j<10;j++)    {
          mtrx[i][j]=count;
          count++;
    }
}
return(0);
}
```

- *No exemplo acima, a matriz **mtrx** é preenchida, sequencialmente por linhas, com os números de 1 a 200.*

Matrizes de Strings

- *A forma geral de declaração:*

```
char nome_da_variável
  [num_de_strings][compr_das_strings];
```

- *Matrizes de strings são matrizes bidimensionais. Imagine uma string. Ela é um vetor. Se fizermos um vetor de strings estaremos fazendo uma lista de vetores. Esta estrutura é uma matriz bidimensional de* ***chars****.*

- *Como acessar uma string individual? É só usar apenas o primeiro índice. Então, para acessar uma determinada string faça:*

```
nome_da_variável [índice]
```


Matrizes de Strings

- *Exemplo de um programa que lê 5 strings e as exibe na tela:*

```
#include <stdio.h>
int main () {
char strings [5][100];
int count;
for (count=0;count<5;count++)  {
   printf ("\n\nDigite uma string: ");
   gets (strings[count]);
}
printf ("\n\n\nAs strings que voce digitou foram:\n\n");
for (count=0;count<5;count++)  printf
   ("%s\n",strings[count]);
return(0);
}
```

Matrizes Multidimensionais

- *A forma geral de declaração:*

  tipo_da_variável nome_da_variável [tam1][tam2] ... [tamN];

- *Uma matriz N-dimensional funciona basicamente como outros tipos de matrizes. Basta lembrar que o índice que varia mais rapidamente é o índice mais à direita.*

Inicialização

- *A forma geral de inicialização de uma matriz:*

  tipo_da_variável nome_da_variável [tam1][tam2] ... [tamN] = {lista_de_valores};

- A lista de valores é composta por valores separados por vírgula. Os valores devem ser dados na ordem em que serão colocados na matriz.

float vect [5] = { 1.3, 4.5, 2.7, 4.1, 0.0}; inicialização de vetores.

int matrx [3][4] = { 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12 }; **matrx** será inicializada com 1, 2, 3 e 4 em sua 1a linha, 5, 6, 7 e 8 na 2a. linha e 9, 10, 11 e 12 na última linha

char str [10] = { 'J', 'o', 'a', 'o', '\0' }; como inicializar uma string

char str [10] = "Joao"; um modo mais simples de inicializar uma string

char str_vect [3][10] = { "Joao", "Maria", "Jose" };

Combina as duas técnicas para inicializar um vetor de strings.

Inicialização Sem Especificação de Tamanho

- Podemos inicializar matrizes das quais não sabemos o tamanho *a priori*.
- O compilador C vai, neste caso verificar o tamanho do que você declarou e considerar como sendo o tamanho da matriz.
- Isto ocorre na hora da compilação e não poderá mais ser mudado durante o programa,
- Muito útil, por exemplo, quando vamos inicializar uma string e não queremos contar quantos caracteres serão necessários.

```
char mess [] = "Linguagem C: flexibilidade e poder.";
```

- A string mess terá tamanho 36. Repare que o artifício para realizar a inicialização sem especificação de tamanho é não especificar o tamanho!

```
int matrx [][2] = { 1,2,2,4,3,6,4,8,5,10 };
```

- Neste caso o valor não especificado será 5.

Matrizes

AUTO AVALIAÇÃO
Veja como você está:

O que imprime o programa a seguir? Tente entendê-lo e responder. A seguir, execute-o e comprove o resultado.

```
# include <stdio.h>
int main()
{
   int t, i, M[3][4];
   for (t=0; t<3; ++t)
      for (i=0; i<4; ++i)
         M[t][i] = (t*4)+i+1;

   for (t=0; t<3; ++t)
   {
      for (i=0; i<4; ++i)
         printf ("%3d ", M[t][i]);
      printf ("\n");
   }
   return(0);
}
```