

An illustration of a person with a beard and glasses sitting at a desk, working on a computer. The person is wearing a dark t-shirt and pants. The desk is dark wood, and the chair is a modern, dark-colored one. On the desk, there is a large monitor displaying code, a keyboard, and a mouse. A small potted plant with green leaves is on the floor next to the desk. In the background, there is a large, dark, wavy shape that looks like a thought bubble or a cloud, containing several lines of colorful code (green, blue, red, yellow). The overall color scheme is dark with purple and blue tones.

# Programação em C

Henrique Ferreira

# 1

## Mecanismos de Controlo de Execução

### Objetivos

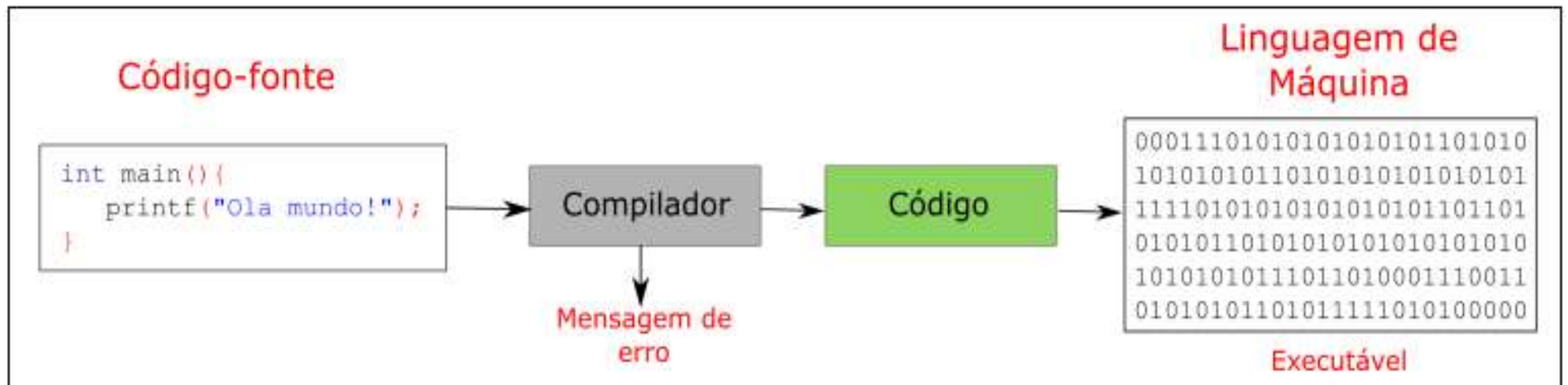
1. Exemplos em linguagem natural envolvendo mecanismos intuitivos de Decisão Binária e Decisão Múltipla
2. Exemplos em linguagem natural envolvendo mecanismos de repetição condicionada por uma expressão lógica
3. Desenvolvimento de algoritmos, fazendo uso de uma linguagem gráfica com o objetivo de analisar o seu fluxo de execução sequencial
4. Estrutura de um programa
5. Tipos de variáveis. Tipos simples
6. Instruções: Afetação, Input e Output de informação
7. Mecanismos de controlo de programa
8. Seleção simples
9. Seleção múltipla
10. Repetição condicional
11. Repetição incondicional

# Introdução

Para iniciar no mundo da [Linguagem C](#), vamos primeiro entender um pouco de como funciona o processo de construção de um programa.

Geralmente, este processo é similar para qualquer linguagem de programação. Em geral, o ciclo de construção de uma aplicação engloba quatro etapas.

# Introdução



# Introdução

## Etapa 1: Escrita do código-fonte

É nesta etapa que o programador realiza o esforço de **escrever o código** que dará origem ao **programa final**.

Este código deve seguir **regras rígidas** para que o **compilador** tenha sucesso ao construir o programa. Este conjunto de regras ou formato que deve ser seguido é denominado como "**sintaxe**" da linguagem.

# Introdução

## Etapa 2: Compilação do programa

Após escrever o código-fonte, o programador deve **acionar o compilador** para que o mesmo possa verificar a **consistência** do código escrito e construir o **programa executável**.

Esta etapa é denominada "**Compilação**", e neste momento podem ocorrer duas situações:

- Na primeira situação, o compilador **não encontra erros no código-fonte** e avança para etapa seguinte.
- Na segunda situação o compilador **encontra erros**, neste caso, a operação é abortada e são exibidas mensagens com os erros que foram encontrados.

# Introdução

O programador deve então **analisar** as mensagens para **corrigir** os erros no **código-fonte** e **refazer** o processo de **compilação**.

O **compilador** pode identificar também **código-fonte** que, apesar de **não possuir erros na sintaxe**, mas que levantam **suspeita** de que, ao executar o programa, o mesmo irá se comportar de forma **inesperada** em **determinadas situações**.

Neste caso, o compilador avança para a **próxima etapa**, contudo, ele exibe avisos dos possíveis problemas, esses avisos são apresentados com o indicativo: **"Warning"**.

# Introdução

Caso **não tenha detetado erros na sintaxe** (mesmo tendo emitido Warnings), o compilador **avança** para a próxima etapa e gera então, um **arquivo objeto**, com o nome igual ao do programa e com extensão ".o" (a extensão pode variar de um sistema operativo para outro). Para **compilar** um programa em **linguagem C**, pode utilizar uma IDE de desenvolvimento, como: CodeBlocks, DevC++, Netbeans, dentre outras.

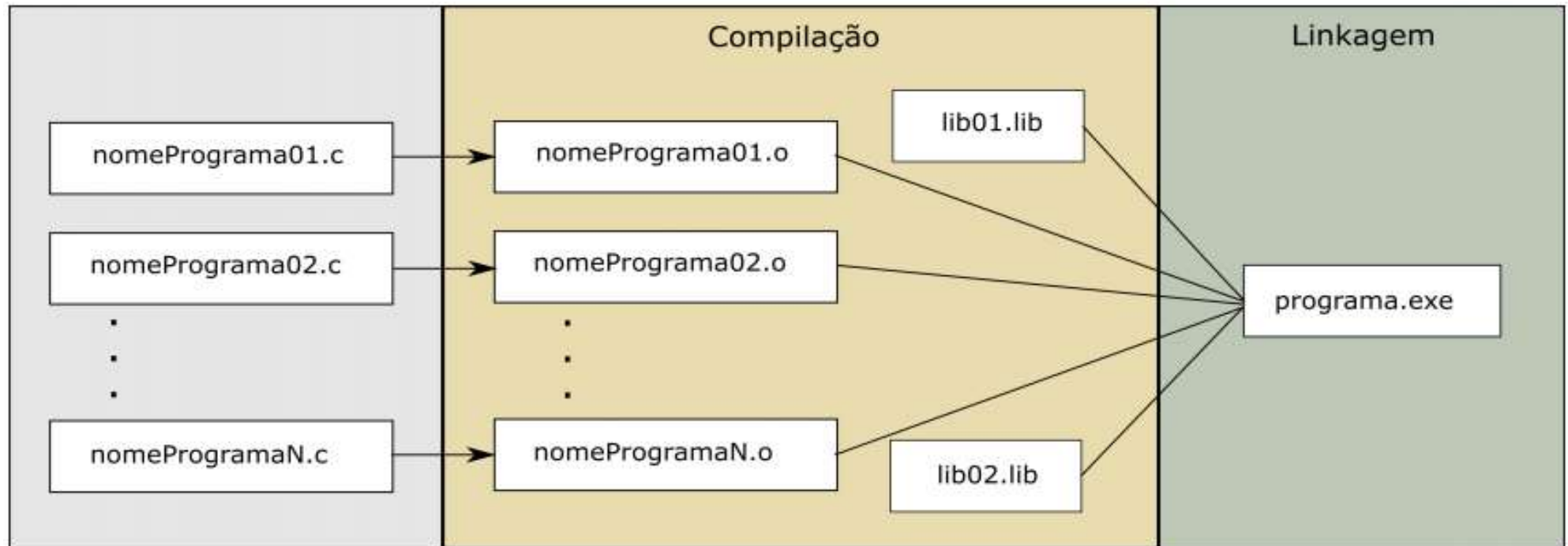


# Introdução

## Etapa 3: "Linkagem" dos objetos

A **compilação** faz então a verificação **sintática** e cria o **arquivo com código-objeto**. Mas o arquivo **executável** é criado na **terceira etapa** em que ocorre a **"linkagem"** dos objetos. O responsável por este processo é o linker que utiliza o arquivo objeto e as bibliotecas nativas do C, que contêm as funções já disponibilizadas pelo C, como: `printf()`, `scanf()`, `fopen()`, etc. Basicamente o linker faz a junção dos arquivos objetos gerados e originados das bibliotecas nativas do C, num único arquivo, o executável.

# Introdução



# Introdução

Normalmente, quando o **compilador** é **acionado**, este já se encarrega de acionar o processo de **linkagem** e não havendo erros o programa **executável** é construído.

É comum o compilador oferecer parâmetros para serem informados, caso o programador não queira que o mesmo acione o linker automaticamente.

# Introdução

## Etapa 4: Execução do Programa

Se as etapas de 1 a 3 ocorrerem sem erros, então ao final deste processo tem-se o arquivo executável do programa. Para testar então a aplicação criada, basta que execute o programa da mesma maneira que se executa outros programas no computador. No Windows, por exemplo, é só abrir a pasta em que o arquivo foi criado e executar um duplo clique no arquivo.

# O Meu primeiro Programa em C

Para que o **compilador** tenha sucesso em **construir o executável**, é preciso escrever uma sequência de código **organizado** e **coeso** de tal forma que seja capaz de **resolver** um problema **logicamente**.

Outra questão importante que deve ser levada em consideração, é que um programa pode ser desenvolvido em **módulos** distintos e/ou **subprogramas**, assim, é necessário indicar qual o **ponto de partida** do programa, ou seja, em que ponto do código-fonte o programa deve **iniciar** as suas instruções.

Ao iniciar o desenvolvimento de uma aplicação **em linguagem C**, comumente utiliza-se o código-fonte apresentado a seguir como ponto de partida.

# O Meu primeiro Programa em C

```
1 #include <stdio.h>
2 #include <stdlib.h>
3 void main() {
4
5 }
```

As **linhas 1 e 2**, são diretivas responsáveis pela inclusão das duas principais **bibliotecas** da **linguagem C**. Essas linhas são importantes, pois são responsáveis por duas bibliotecas necessárias a um programa C, mesmo que este seja básico.

# O Meu primeiro Programa em C

A **linha 3** é a definição do método **main** da aplicação, que é responsável pela definição de quais serão as primeiras instruções do programa a serem executadas, desta forma, esta é a função que se encarrega de indicar aonde o programa deve começar.

Toda aplicação em **linguagem C** deverá possuir ao menos um método **main** para desempenhar este papel

Note que, as instruções iniciais deverão ser colocadas na **linha 4**, ou seja, entre a **linha 3** e a **linha 5**. Essas duas chaves “**{ }**” encontradas nessas duas linhas são as responsáveis por indicar aonde começa e aonde termina o método **main**. Desta forma, várias instruções podem ser adicionadas no método **main** desde que estejam entre estas duas chaves.

# O Meu primeiro Programa em C

Na **linguagem C**, as chavetas sempre irão indicar um **bloco** de códigos, ou seja, sempre que desejar indicar o local de início e término de um bloco de códigos, utiliza-se chaves.

O nosso primeiro programa vai ter um objetivo muito simples, o de **imprimir** a mensagem: "Olá mundo!".

Para realizar essa operação precisa-se de uma **função** que tenha essa capacidade e a **linguagem C** disponibiliza-nos a função **printf()**, que é, provavelmente umas das funções que mais será utilizadas ao longo do aprendizado. Para utilizar a função **printf()** corretamente, é preciso seguir a sintaxe a seguir:

```
1 //Sintaxe:  
2 printf("formato", argumentos);
```



# O Meu primeiro Programa em C

O primeiro parâmetro da função `printf()` é onde deve ser informado o texto que se pretende apresentar na janela. Desta forma, no nosso exercício o texto "Olá mundo!" será informado neste parâmetro.

O segundo parâmetro é `opcional` e, além disso, podem ser informados vários argumentos, tantos quantos forem necessários.

Utilizando então a função `printf()` para fazer o nosso primeiro programa apresentar a mensagem "Olá mundo!" tem-se o código-fonte apresentado a seguir. Observe também que o texto está entre aspas duplas, sempre que desejar utilizar um texto em linguagem natural no código-fonte, então é necessário colocar esse texto entre aspas duplas. Por fim, observe que no final da instrução foi colocado um ponto e vírgula `;`. Deve-se fazer isso sempre no final da instrução.

# O Meu primeiro Programa em C

```
1 #include <stdio.h>
2 #include <stdlib.h>
3 void main() {
4     printf("Ola Mundo!");
5 }
```

# Variáveis

Utiliza-se as variáveis num programa de computador, para armazenar dados.

Tipo	Descrição
int	Utilizado para definir uma variável inteira que comporta valores pertencentes ao conjunto dos números inteiros.
long int	Utilizado para definir uma variável inteira que comporta valores maiores que o int.
short int	Utilizado para definir uma variável inteira que comporta valores menores que o int.
float	Utilizado para definir uma variável real que comporta valores pertencentes ao conjunto dos números reais
double	O tipo double é similar ao float, a diferença é que o este tipo comporta valores reais com um número de dígitos maior, assim, a precisão nos cálculos com casas decimais aumenta
char	Utilizado para armazenar um caractere. Comporta apenas um caractere.

# Variáveis

Agora que os tipos que foram apresentados, falta entender como definir uma **variável**. Em **linguagem C**, essa tarefa é muito simples, veja a seguir a sintaxe:

```
1 //Sintaxe:  
2 tipoVariavel nomeVariavel;
```

Como é apresentado na sintaxe, primeiro informa-se o tipo da variável

```
1 int idade;  
2 float peso;  
3 char genero;  
4 double rendimento;
```

# Variáveis

Quando é preciso **definir** mais de uma **variável** do **mesmo tipo**, pode-se fazer isso numa **mesma** instrução, apenas separando o nome das variáveis por virgula, como os exemplos a seguir:

```
1 int idade, matricula;  
2 float vCelcius, vKelvin;  
3 char genero, sexo;
```

# Variáveis

Assim, como noutras linguagens, o C possui várias **regras** para a definição de **variáveis**. Além disso, existem **palavras reservadas** que não podem ser utilizadas na nomenclatura das variáveis.

Outro ponto que deve ser levado em consideração ao definir variáveis e que a linguagem C é "**case sensitive**", ou seja, letras maiúsculas e minúsculas correspondentes são consideradas diferentes, desta forma, ao utilizar a variável, à mesma deverá ser escrita exatamente como foi escrita na definição, pois caso contrário o compilador irá entender que se trata de outra variável.

# Variáveis

Forma incorreta	Motivo	Sugestão de correção
<code>float 4nota;</code>	Não é permitido iniciar o nome da variável com números.	<code>float nota4;</code>
<code>char float;</code>	Não é permitido utilizar palavra reservada como nome de uma variável.	<code>char vFloat;</code>
<code>double vinte%;</code>	Não é permitido utilizar os seguintes caracteres especiais como %, @, #, \$, &, etc.	<code>double vintePercent;</code>
<code>int idade pes;</code>	Não é permitido separar os nomes compostos em variáveis.	<code>int idade_pes;</code>

# Variáveis

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



# Atribuição

Para **armazenar** dados em **variáveis**, é preciso fazer **atribuição**.

Para isso, deve-se utilizar o operador de atribuição, que na linguagem C é o sinal de igualdade, "**=**".

```
1 int idade, matricula;  
2 idade = 30;  
3 matricula = 123659;  
4 float preco = 42.9;
```

# Atribuição

Ao escrever uma **expressão matemática em C** é preciso considerar a **ordem** das operações em matemática, exemplo, na inexistência de parênteses, a operação de multiplicação e divisão será realizada antes das operações de adição e subtração.

Quando houver a necessidade de alterar a precedência das operações devem-se utilizar os parênteses.

```
1 int contador = 2;  
2 float valor1 = 300;  
3 float valor2 = 400;  
4 float totalSom = valor1 + valor2;  
5 float totalMult = valor1 * valor2;  
6 contador++;  
7 resul = (totalMult + totalSom) * contador;
```

# Atribuição

A Figura apresenta os operadores de atribuições disponíveis.

Operador	Operação Matemática
=	Atribuição simples
+=	Atribuição acumulando por soma.
-=	Atribuição acumulando por subtração
*=	Atribuição acumulando por multiplicação
/=	Atribuição acumulando por divisão
%=	Atribuição acumulando por módulo

# Entrada e Saída

## Função printf()

A função `printf()` permite realizar a **impressão** de **textos** no monitor, ou seja, é responsável pela **saída** de informações. Esta **função** possui um número variado de parâmetros, tantos quantos forem necessários.

```
1 //Sintaxe:  
2 printf("formato", argumentos);
```

Como já mencionado antes, o primeiro argumento da função é **obrigatório**, ou seja, no mínimo deve-se informar um **texto** para ser **impresso**. Os próximos argumentos são opcionais. O uso da função com mais de um argumento requer também o uso de **formatadores de tipo**.

```
1 float total = 300 + 400;  
2 printf("Total da conta: %f", total);
```

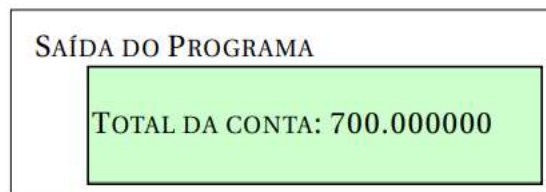
# Entrada e Saída

Formato	Tipo da variável	Conversão realizada
%c	Caracteres	char, short int, int, long int
%d	Inteiros	int, short int, long int
%e	Ponto flutuante, notação científica	float, double
%f	Ponto flutuante, notação decimal	float, double
%lf	Ponto flutuante, notação decimal	double
%g	O mais curto de %e ou %f	float, double
%o	Saída em octal	int, short int, long int, char
%s	String	char *, char[]
%u	Inteiro sem sinal	unsigned int, unsigned short int, unsigned long int
%x	Saída em hexadecimal (0 a f )	int, short int, long int, char
%X	Saída em hexadecimal (0 a F)	int, short int, long int, char
%ld	Saída em decimal longo	Usado quando long int e int possuem tamanhos diferentes.

# Entrada e Saída

Observe que na **segunda linha** do exemplo a função **printf()** foi acionada com dois argumentos, o primeiro argumento é o texto combinado com o formatador de tipo e o segundo é a variável que foi definida na linha anterior.

O objetivo é que seja impresso o texto combinado com o dado **armazenado** na **variável**. O dado armazenado na variável é o resultado da expressão  $300 + 400$ , portanto o dado armazenado é 700. Assim sendo, a saída deste programa deve ser conforme apresentado na Figura.



SAÍDA DO PROGRAMA

TOTAL DA CONTA: 700.000000

# Entrada e Saída

Ou seja, o dado da variável é substituído pelo formatador de tipo `%f`. Observe que foi utilizado o formatador para `ponto flutuante` com notação decimal, pois a variável a ser impressa é do tipo `float` e deseja-se imprimir o dado no formato `decimal`. Se a variável fosse do tipo `int`, então pode-se utilizar o formatador `%d`. Nota-se também que o dado 700 foi impresso com vários zeros na sua fração, isso aconteceu porque não foi realizada uma formatação do número para indicar a quantidade de casas decimais que se deseja.

# Entrada e Saída

Para formatar um float, basta seguir a definição:

```
1 %[tam].[casa_dec]f
```

Em que:

- **tam** – indica o tamanho mínimo que deve ser impresso na saída. Se o número possuir um tamanho superior ao informado neste parâmetro, o número não será truncado.
- **casa\_dec** – número de casas decimais que devem ser impressas. Neste caso, se o número possuir uma quantidade de casas decimais superior ao indicado, então as casas decimais serão truncadas.



# Entrada e Saída

Assim, se quiser imprimir a informação com apenas duas casas decimais, basta ajustar o código da seguinte forma:

```
1 float total = 300 + 400;  
2 printf("Total da conta: %3.2f", total);
```

SAÍDA DO PROGRAMA

TOTAL DA CONTA: 700.00

# Entrada e Saída

Observe agora um exemplo de uso da função `printf()` com 3 parâmetros.

```
1 int mat = 335642;  
2 float medF = 7;  
3 printf("Matricula: %d, Med Final: %2.2f ", mat, medF);
```

SAÍDA DO PROGRAMA

MATRICULA: 335642, MED FINAL: 7.00

# Entrada e Saída

No exemplo com três parâmetros, observa-se que foram adicionados dois formatadores no texto, o primeiro `%d`, pois o segundo parâmetro é uma variável do tipo `int` e o segundo formatador `%2.2f` visto que o terceiro parâmetro é uma variável do tipo `float`.

Conclui-se então que, o texto pode ser combinado com tantas quantas forem às informações que se deseja apresentar, e que para isso, basta adicionar os formatadores de tipo na mesma ordem em que os parâmetros seguintes serão informados.

# Entrada e Saída

agora outro exemplo de uso da função `printf()`.

```
1 int mat = 335642;  
2 float medF = 7;  
3 printf("Matricula: %d", mat);  
4 printf("Media Final: %2.2f", medF);
```

SAÍDA DO PROGRAMA

MATRICULA: 335642MEDIA FINAL: 7.00

# Entrada e Saída

Note na Figura acima como a saída do programa foi impressa. Veja que o texto impresso pela primeira função `printf()` está **concatenado** com o texto impresso pela segunda função. Isso ocorreu porque não foi informado para a primeira função a intenção de adicionar espaço, tabulação ou quebra de linha.

Como é possível resolver isso então?

A linguagem C tem uma forma bem elegante de resolver isso, ela trata o caractere `"\` como sendo especial para uso combinado com outros caracteres que irão permitir operações especiais no texto. Para resolver o problema do exemplo anterior, basta adicionar uma quebra de linha.

# Entrada e Saída

Note que no exemplo foram adicionados os caracteres `"\n"` na primeira função `printf()`. Ao fazer isso, a função entenderá que deve "mudar de linha" após a impressão do texto anterior aos caracteres `"\n"`, com isso, o texto da segunda função foi impresso na linha seguinte.

```
1 int mat =335642;
2 float medF = 7;
3 printf("Matricula: %d\n", mat);
4 printf("Media Final: %2.2f", medF);
```

## SAÍDA DO PROGRAMA

```
MATRICULA: 335642
MEDIA FINAL: 7.00
```

# Entrada e Saída

## Função scanf()

Como visto a função `printf()` é responsável pela **saída** das informações do programa. Como fazer então para **entrar** com dados na fronteira do programa?

Neste caso, deve-se utilizar a função `scanf()`. Similar à função `printf()`, a função `scanf()` também suporta uma quantidade “n” de argumentos e permite que os dados digitados pelo utilizador do programa sejam armazenados nas variáveis do programa.

```
1 //Sintaxe:  
2 scanf("formato", enderecosArgumentos);
```

# Entrada e Saída

Caracteres	Ação
\b	Retrocesso (back)
\f	Alimentação de formulário (form feed)
\n	Nova linha (new line)
\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)



# Entrada e Saída

No caso da função `scanf()`, no mínimo devem ser informados dois argumentos, o primeiro é responsável pelo formato dos dados de entrada, e o segundo argumento é o endereço da variável, para armazenar o dado digitado pelo utilizador.

```
1 int mat;  
2 scanf("%d", &mat);
```

Na linha 1 do exemplo, foi feita a declaração da **variável** que será utilizada para **armazenar** o dado. Na linha 2 foi realizado o acionamento da função `scanf()`, note que no primeiro argumento foi informado o formato entre aspas, `"%d"`. E o segundo argumento é o caractere `&` acompanhado do nome da **variável**. A partir do segundo argumento deve-se informar o **endereço de memória** no qual o dado será armazenado, por isso, foi utilizado o caractere `&` acompanhado do **nome da variável**, pois em C, o caractere `&` é responsável por indicar que o **retorno** será o **endereço de memória** de uma determinada **variável**.

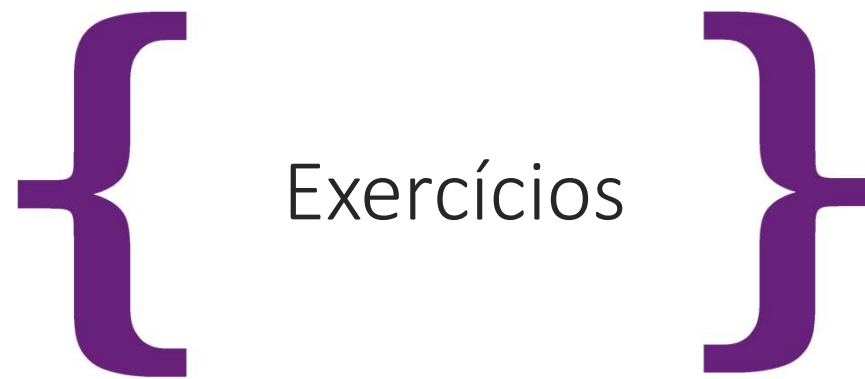
# Entrada e Saída

agora um exemplo do uso da função para três argumentos.

```
1 float nota1, nota2;  
2 scanf("%f %f", &nota1, &nota2);
```

A primeira linha do exemplo é a **declaração** de duas **variáveis** do tipo **float** e a segunda linha é o acionamento da função **scanf()**, neste caso, para a **leitura** e **armazenamento** em duas variáveis, nota1 e nota2. Note que, no formato foi escrito o **"%f"** duas vezes e separado por um espaço, assim, o **scanf()** saberá tratar a formatação dos dois próximos argumentos. E da mesma forma pode-se combinar o uso não só de variáveis diferentes, mas também de tipos diferentes.

# Entrada e Saída



# Entrada e Saída

Faça um programa em C que receba dois números inteiros e ao final imprima a soma deles.

# Entrada e Saída

```
1 #include <stdio.h>
2 #include <stdlib.h>
3 void main() {
4     int soma, num1, num2;
5     printf("Informe o primeiro numero:");
6     scanf("%d", &num1);
7     printf("Informe o segundo numero:");
8     scanf("%d", &num2);
9
10    soma = num1 + num2;
11
12    printf("Resultado da soma: %d", soma);
13 }
```

# Entrada e Saída

## Lendo texto com a função scanf()

A função `scanf()` funciona muito bem para os tipos `int`, `float`, `double`, entre outros, contudo, quando se trata do armazenamento de um `texto`, caso em que, deve-se utilizar o `char` com definição de tamanho, esta função apresenta um comportamento indesejável em uma situação específica.

```
1  #include <stdio.h>
2  #include <stdlib.h>
3
4  void main()
5  {
6      char produto[30];
7      printf("Informe o nome do produto: \n");
8      scanf("%s", &produto);
9
10     printf("Produto: %s \n", produto);
11 }
```

# Entrada e Saída

Se fez exatamente como solicitado, então obterá a saída para a execução do programa apresentada na Figura.



Agora, execute novamente o programa e digite para o nome do produto: "Arroz integral", após digitar, pressione ENTER. Notou que o resultado da execução é exatamente igual ao apresentado na Figura acima?

# Entrada e Saída

Isso ocorre, porque o `scanf()` não sabe lidar muito bem com nomes compostos.

Para resolver isso, podemos utilizar uma formatação diferente que se encarregará de resolver este problema.

```
1 #include <stdio.h>
2 #include <stdlib.h>
3
4 void main()
5 {
6     char produto[30];
7     printf("Informe o nome do produto: \n");
8     scanf("\n%[^\\n]s", &produto);
9
10    printf("Produto: %s \n", produto);
11 }
```



# Entrada e Saída

Veja a função `scanf()` na linha 8, notou a diferença no formato utilizado?

Agora repita o teste anterior e note que o resultado será similar ao apresentado na Figura acima. Contudo, esse formato não é recomendado, pois o `scanf()` poderá produzir outro efeito indesejado, no caso, tentar armazenar um texto com tamanho superior ao suportado pela variável.

O método recomendado para resolver este problema é utilizar a função `fgets()`, veja na sequência, como essa função poderá ser utilizada.

A diagrama de saída de programa. Um retângulo branco com uma borda preta contém o texto "SAÍDA DO PROGRAMA" no topo. Abaixo dele, há um retângulo verde com uma borda preta que contém o texto "PRODUTO: ARROZ INTEGRAL".

SAÍDA DO PROGRAMA

PRODUTO: ARROZ INTEGRAL

# Entrada e Saída

## Função fgets()

Outra forma de resolver o problema, apresentado pelo `scanf()` ao ler textos, é utilizar a função `fgets()`. Esta função irá **armazenar** corretamente o **texto**, mesmo quando este é composto por duas, três ou mais palavras, desde que, seja respeitado o tamanho limite determinado para o **char**.

```
1 #include <stdio.h>
2 #include <stdlib.h>
3
4 void main()
5 {
6     char produto[30];
7     printf("Informe o nome do produto: \n");
8     fgets(produto, 30, stdin);
9
10    printf("Produto: %s \n", produto);
11 }
```

# Entrada e Saída

Veja que a única linha do código-fonte que sofreu mudança é a linha 8, na qual estava a instrução do `scanf()`, que foi substituído pelo `fgets()`. A função `fgets()` possui três argumentos, o primeiro é o **nome da variável**, note que, neste caso, não é necessário indicar o sinal de `&` como é necessário no `scanf()`, pois, o argumento esperado pela função `fgets()` é o nome da variável e não o endereço de memória dela. O segundo argumento é o **tamanho da variável**, que no exemplo, é 30, conforme definido na linha 6, e o terceiro argumento é o **arquivo** no qual o texto deve ser armazenado, como no caso, a intenção é armazenar numa variável na memória, então informamos a constante `stdin` para indicar que a leitura é proveniente do teclado. Se compilar e executar agora, este trecho de código e informar o nome do produto igual a "Arroz integral", obterá a saída conforme a Figura acima.

# Comentários

É importante, principalmente em trechos menos intuitivos, **comentar** (explicar) sobre o que foi escrito, mas como o **compilador** não aceita algo diferente da **sintaxe** da linguagem de programação misturada ao código, em geral, as linguagens de programação disponibilizam "**indicadores**" para de certa forma, dizer ao **compilador**: "despreze esse trecho!", assim, ao indicar qual trecho do código que o compilador deve desprezar, pode-se fazer uso deste para escrever em qualquer formato, incluindo a linguagem formal ou informal, que o compilador não irá gerar erros relacionados àquele trecho. Este recurso é chamado de "**comentário**".

# Comentários

```
1 #include <stdio.h>
2 #include <stdlib.h>
3 void main() {
4     int soma, num1, num2;
5     //trecho responsavel pela entrada dos dados
6     printf("Informe o primeiro numero:");
7     scanf("%d", &num1); //leitura de num1
8     printf("Informe o segundo numero:");
9     scanf("%d", &num2);
10    /* O trecho a seguir e responsavel pela soma dos valores de num1 e num2
11       informados pelo usuario do programa */
12    soma = num1 + num2;
13    printf("Resultado da soma: %d", soma);
14 }
```

# Comentários

Há duas formas de utilizar **comentários** em **linguagem C**, a primeira forma é apresentada na linha 5 do exemplo, trata-se do uso de duas barras `"/"`, este formato é o mais comum e permite comentar o código a partir do ponto em que as barras são incluídas e apenas na linha em que elas foram colocadas. Na linha 7 pode-se ver um exemplo de comentário que foi colocado após um trecho de código que deve ser considerado pelo compilador. Uma **vantagem** do uso das duas barras é que não é necessário indicar aonde **termina** o comentário, uma vez que, pela sua natureza apenas uma linha ou parte dela é comentada.

Contudo se for preciso escrever um comentário com mais de uma linha, então o segundo formato é o mais indicado, pois, neste caso, basta colocar o indicador de onde o comentário deve iniciar e depois colocar o indicador de onde o comentário termina, assim, é possível escrever comentários com várias linhas apenas indicando o ponto de início e término. A linha 10 apresenta o exemplo deste formato de comentário, note que neste caso, o indicador de início é `"/*"` e o indicador de término é `"*/"`.