LINGUAGEM C – UMA INTRODUÇÃO

AULA 1 – Conceitos muito básicos

1 – Introdução

O C nasceu na década de 70. Seu inventor, Dennis Ritchie, implementou-o pela primeira vez usando um DEC PDP-11 rodando o sistema operacional UNIX. O C é derivado de uma outra linguagem: o B, criado por Ken Thompson. O B, por sua vez, veio da linguagem BCPL, inventada por Martin Richards.

O C é uma linguagem de programação genérica que é utilizada para a criação de programas diversos como processadores de texto, planilhas eletrônicas, sistemas operacionais, programas de comunicação, programas para a automação industrial, gerenciadores de bancos de dados, programas de projeto assistido por computador, programas para a solução de problemas da Engenharia, Física, Química e outras Ciências, etc ...

Estudaremos a estrutura do ANSI C, o C padronizado pela ANSI. Veremos ainda algumas funções comuns em compiladores para alguns sistemas operacionais. Quando não houver equivalentes para as funções em outros sistemas, apresentaremos formas alternativas de uso dos comandos.

1.1 - O C é "Case Sensitive"

Vamos começar o nosso curso ressaltando um ponto de suma importância: o C é "Case Sensitive", isto é, maiúsculas e minúsculas fazem diferença. Se você declarar uma variável com o nome *soma* ela será diferente de *Soma*, *SOMA*, *SoMa* ou *sOmA*. Da mesma maneira, os comandos do C *if* e *for*, por exemplo, só podem ser escritos em minúsculas pois senão o compilador não irá interpretá-los como sendo comandos, mas sim como variáveis.

1.2 – O Compilador

Os computadores não entendem nada além de comandos, dados e endereços escritos em linguagem binária. Mas, qualquer ser humano que se disponha a tentar desenvolver um programa complexo programando diretamente em linguagem de máquina simplesmente vai ficar louco muito antes de concluir seu trabalho.

Para resolver este impasse, surgiram as linguagens de programação, que permitem escrever programas usando comandos fáceis de lembrar e funções já prontas. O compilador é programa que permite transformar este código escrito na linguagem de

programação em linguagem de máquina, gerando o binário ou “programa executável”. Existem diversos compiladores disponíveis no mercado, para as mais variadas linguagens de programação. Um exemplo de compilador muito usado atualmente é o GCC da Free Software Fundation, que possui módulos para compilar programas de várias linguagens. Nesse curso será usado o compilador Dev-C++ que na verdade é uma interface de desenvolvimento integrada, que possui um editor de código, um compilador, um debugador e outras coisas mais...

1.3 – Código fonte

O “código fonte” de um programa é o arquivo (ou conjunto de arquivos) que contém os comandos e rotinas que formam um programa. Este código é então compilado, gerando o arquivo binário (ou executável) que será executado. Ao comprar um programa qualquer, recebemos apenas os binários, que permitem instalar e executar o programa, mas não o código fonte, que permitiria alterá-lo ou entender como ele funciona.

Em programas comerciais, o código fonte é cuidadosamente guardado, mas existe um movimento crescente de desenvolvimento de softwares livres, onde o código fonte é distribuído junto com o programa, o que permite a qualquer um com conhecimentos de programação alterá-lo, corrigir bugs ou adicionar novos recursos, desde que sejam mantidos os créditos para o criador original. O movimento de software livre inclui o Linux e a maior parte dos aplicativos desenvolvidos para ele.

1.4 – Esqueleto básico de um programa em C

Todos os programas das primeiras aulas de nosso curso em C possuem um esqueleto, ou seja, um conjunto de instruções básicas. Nesse momento o leitor deverá apenas “decorar” e usar esse esqueleto. No decorrer do curso, todas as linhas de código desse esqueleto serão devidamente explicadas.

O autor desse material acredita que uma explicação dessas linhas nesse momento apenas acrescenta uma complexidade adicional no processo de aprendizado, sem acrescentar nenhum benefício imediato.

Esqueleto básico:

```
#include <stdio.h>

int main() {

	return 0;
}
```

Todos os programas que iremos escrever nessas primeiras aulas serão formados por um conjunto de comandos (ou instruções) entre a linha “int main()” e a linha “return 0;”. Veja um exemplo de um programa (você não precisa entendê-lo agora):

Exemplo 1

```
#include <stdio.h>

int main() {
```

```
	int idade;
	printf(“Qual a sua idade?\n”);
	scanf(“%d”, &idade);
	printf(“\nNossa.. %d aninhos!!\n”, idade);

	return 0;
}
```

1.5 – Para que serve o ponto-e-vírgula?

O ponto-e-vírgula (;) deve ser usado ao final de cada comando em Linguagem C. Isto é uma “regra geral”, mas como todos sabem, toda regra possui exceções. Considere inicialmente essas duas exceções:

- As linhas de #include <> não devem possuir ponto-e-vírgula
- As linhas com { ou } normalmente não devem possuir ponto-e-vírgula

Os dois programas abaixo **estão errados**:

Exemplo 2

```
#include <stdio.h>

int main() {

	int idade;
	printf(“Qual a sua idade?\n”);
	scanf(“%d”, &idade)
	printf(“\nNossa.. %d aninhos!!\n”, idade);

	return 0;
}
```

Exemplo 3

```
#include <stdio.h>;

int main() {
	int idade;
	printf(“Qual a sua idade?\n”);
	scanf(“%d”, &idade);
	printf(“\nNossa.. %d aninhos!!\n”, idade);

	return 0;
};
```

2. Primeiros comandos

2.1 – printf()

O comando *printf()* serve para exibirmos mensagens na tela. Seu uso mais simples é da forma:

```
printf(“seu texto aqui!”);
```

Observe que o uso das aspas duplas é obrigatório, portanto os seguintes comandos estariam errados:

```
printf(‘seu texto aqui!’);
printf(seu texto aqui!);
```

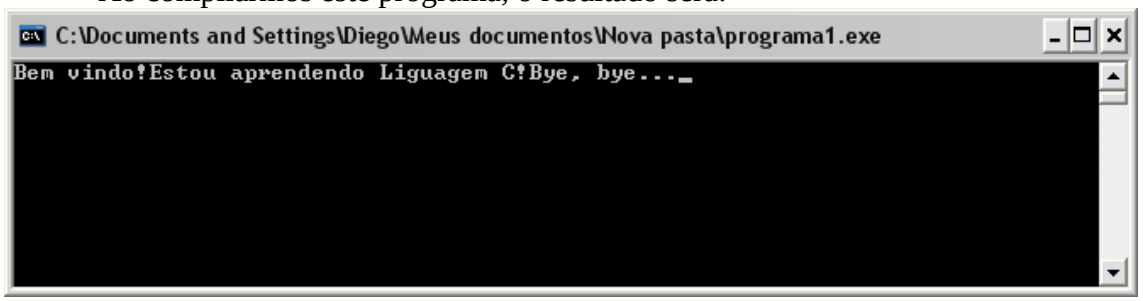
Existem várias combinações de caracteres “especiais” que podem ser usados dentro do comando *printf()*, como por exemplo o *n*. Esse *n* quer dizer “quebra de pinha”, ou “pule uma linha”.

Observe o programa abaixo:

Exemplo 4

```
#include <stdio.h>

int main() {

      printf(“Bem vindo!”);
      printf(“Estou aprendendo Liguagem C!”);
      printf(“Bye, bye...”);

      return 0;
}
```

Ao compilarmos este programa, o resultado será:

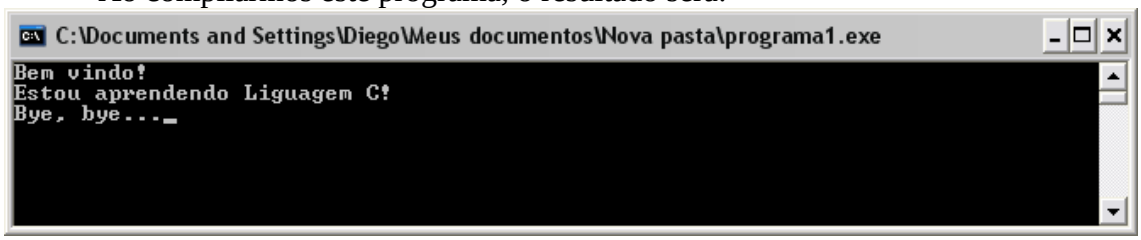


Agora, vamos introduzir quebras de linha (*n*) dentro do segundo e do terceiro *printf()*, de forma que o nosso programa fique da seguinte maneira:

Exemplo 5

```
#include <stdio.h>

int main() {

      printf(“Bem vindo!”);
      printf(“\nEstou aprendendo Liguagem C!”);
      printf(“\nBye, bye...”);

      return 0;
}
```

Ao compilarmos este programa, o resultado será:



2.2 – getchar()

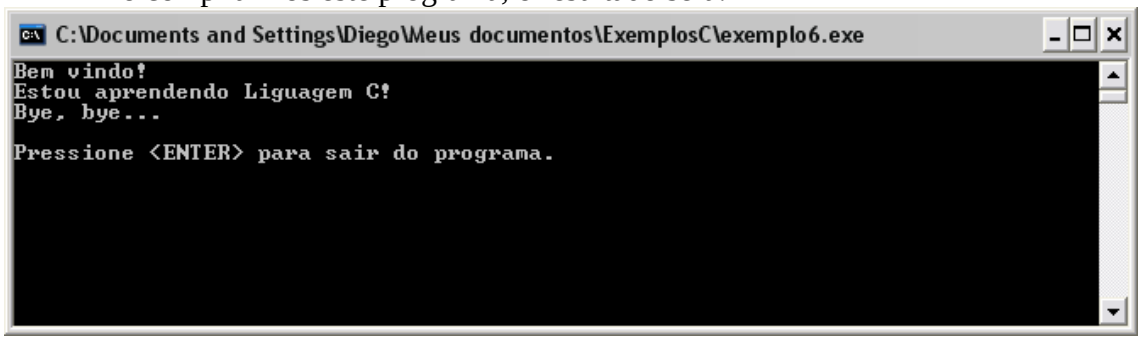
A função *getchar()* espera que o usuário digite um caracter seguido de um <ENTER> ou que ele simplesmente pressione um <ENTER>.

Essa função será muito útil nos primeiros programas do curso, pois se não colocarmos esse comando no final dos programas, ao compilarmos um determinado código fonte com o DEV-C++, nosso programa irá apenas piscar na tela sair, pois o programa será executado e encerrado. Você provavelmente não verá o resultado de seu programa. Esse "problema" ocorre em todos os exemplos apresentados até esse momento nesse material, portanto o ideal seria corrigir o programa anterior da seguinte forma:

Exemplo 6

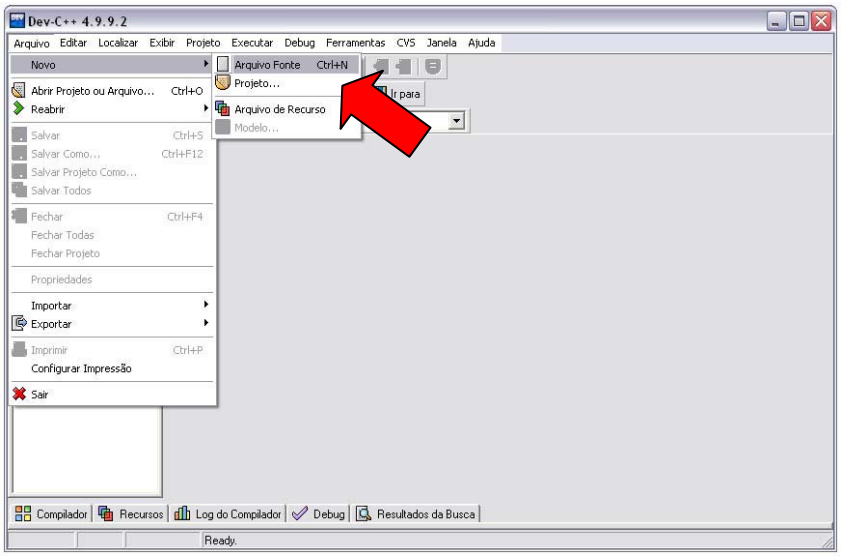
```
#include <stdio.h>

int main() {

      printf("Bem vindo!");
      printf("\nEstou aprendendo Liguagem C!");
      printf("\nBye, bye...");
      printf("\n\nPressione <ENTER> para sair do programa.");
      getchar();

      return 0;
}
```

Ao compilarmos este programa, o resultado será:

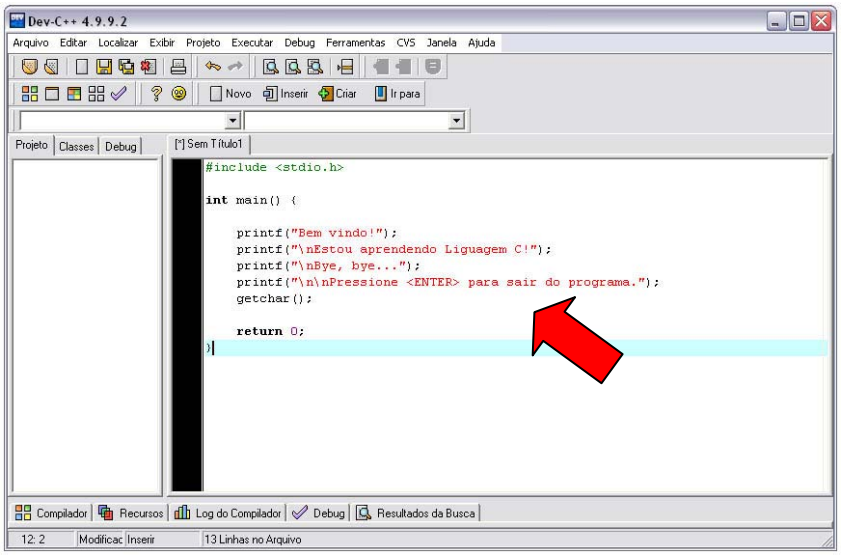


3. Compilando seu primeiro programa no DEV-C++

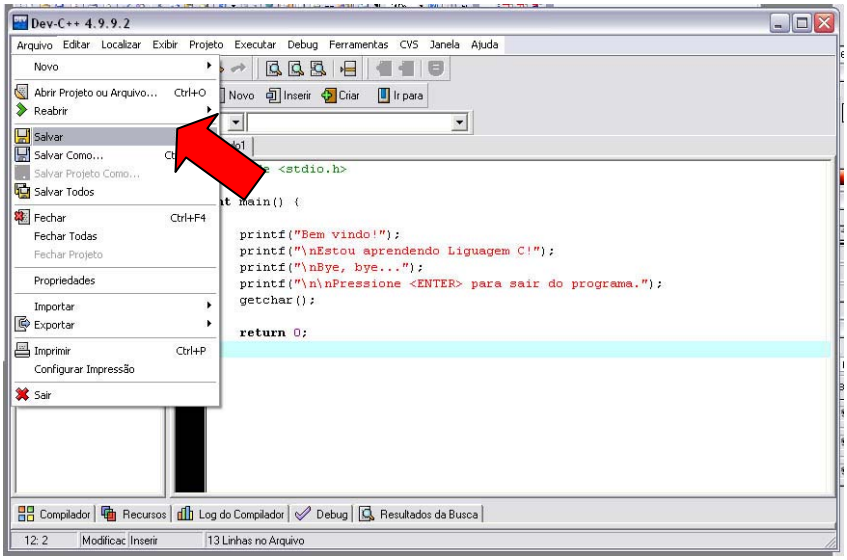
1º Passo: Abra o DEV-C++, vá ao menu “Arquivo”, depois em “Novo” e finalmente em “Arquivo Fonte”.



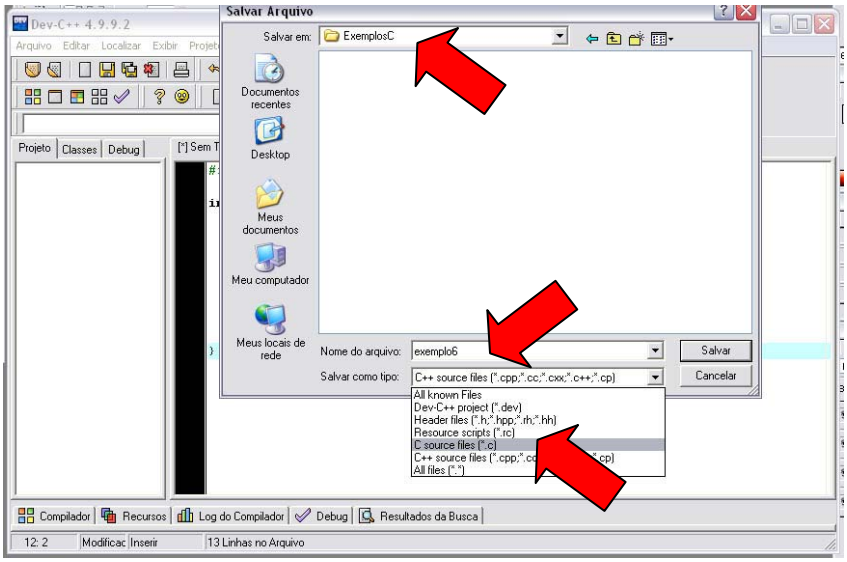
2º Passo: Digite o seu código fonte (Exemplo 6) dentro do DEV-C++.



3º Passo: Salve o seu arquivo como um arquivo C. Parra isso primeiro vá no menu "Arquivo", "Salvar"...

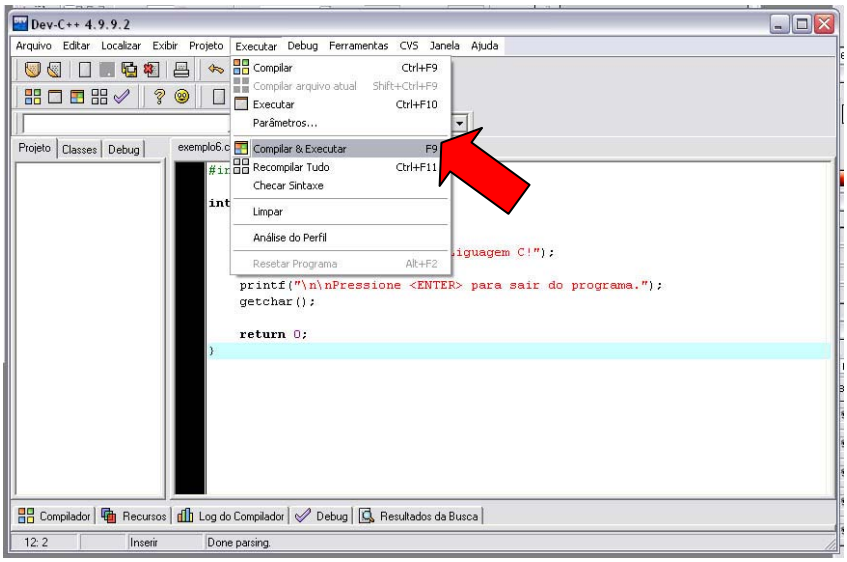


Na tela que irá abrir, escolha uma pasta para salvar o arquivo, depois escolha um nome para o arquivo e finalmente na opção **"Salvar como tipo:" escolha a opção "C source files (*.c)"**, conforme a figura abaixo:

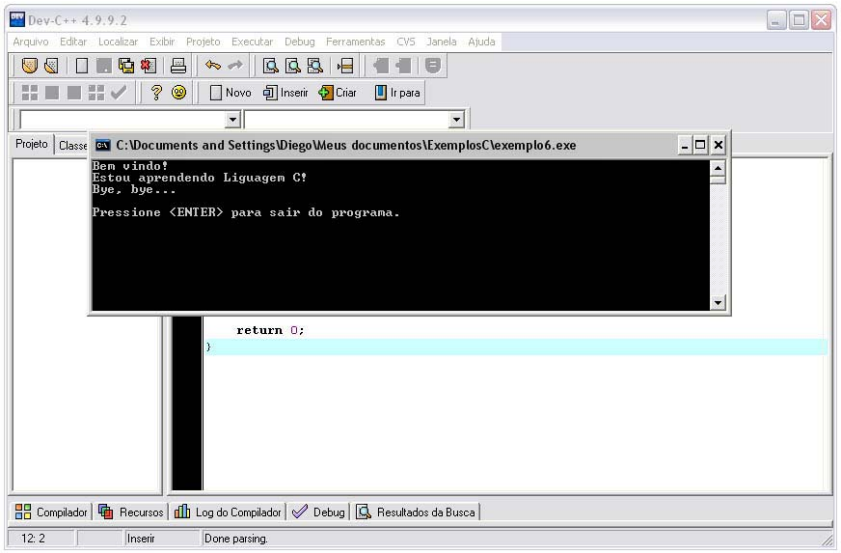


Observação: É recomendado que você crie uma pasta para salvar seus programas. Na tela acima, eu criei uma pasta chamada "ExemplosC" e irei salvar todos os meus programas dentro dela.

4º Passo: Compile e execute o seu programa através do menu “Executar” e “Compilar e Executar”.

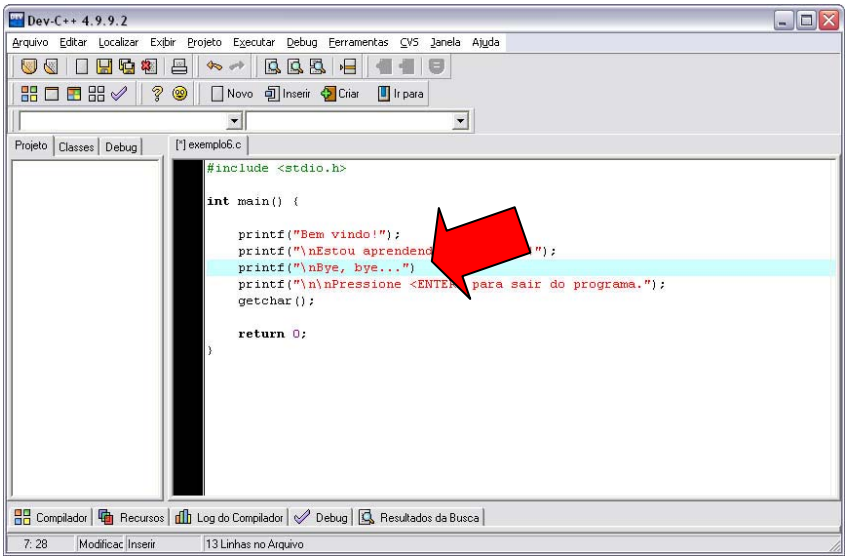


Caso você tenha seguido todos os passos de forma correta, o seu programa será compilado e executado em uma janela de MS-DOS conforme a figura abaixo:

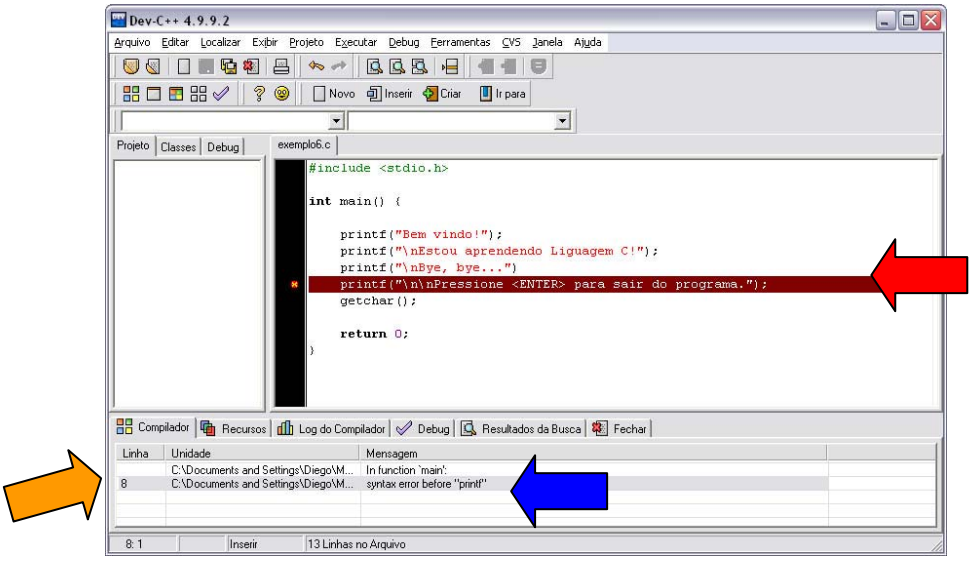


3.2 – Erros de compilação

Quando você cometer algum erro de digitação, ou erro de sintaxe no código fonte de seu programa, o DEV-C++ não irá finalizar a compilação do programa. No exemplo abaixo, um ponto-e-vírgula foi retirado para que fosse gerado um erro:



Ao compilarmos o programa, a seguinte tela é gerada:



Entendendo essa tela:

- O compilador está nos avisando que temos um erro na **linha 8** do nosso programa (seta laranja).
- O erro que ele está apontando é: **sintax error before printf** (seta azul), ou seja, existe algum erro de sintaxe antes do printf() da linha 8. Esse erro é exatamente o erro que esperávamos, pois foi removido o ponto-e-vírgula do final da linha 7, então a linha 7 não foi "encerrada", logo temos um erro antes do printf() da linha 8!
- O DEV-C++ aponta no programa onde está o erro (seta vermelha).

LINGUAGEM C – UMA INTRODUÇÃO

AULA 2 – Conceitos básicos

1 – Identação

Um ponto fundamental organização de escrita de códigos fonte é a identação. Identar é organizar horizontalmente de forma hierárquica as linhas de código, de acordo com o escopo onde elas se encontram. O pseudo programa abaixo exemplifica isto.

```
Variavel A = 0
SE (Variavel A < 1) ENTÃO
    ESCREVA "Variável A é menor que 1."
FIM SE
```

Observe que o comando ESCREVA está adiantado em relação aos outros comandos, que fazem parte do escopo principal do algoritmo e estão alinhados à margem. Isto ocorre porque o comando ESCREVA não pertence diretamente ao escopo do principal do programa, e sim ao escopo condicional SE..ENTÃO...FIM SE.

A identação independe da linguagem de programação que usamos. Mas ela varia um pouco de acordo com a linguagem e tem comportamentos que são convencionados de acordo com o uso dessas linguagens. Não existe uma regra definida para identação. Um exemplo prático pode ser dado se quisermos implementar o pseudo programa acima em Linguagem C:

Exemplo 7

```
#include <stdio.h>

int main() {

      int A=0;
      if ( A < 1 ) {
            printf(“Variável A é menor que 1.”);
      }

      getchar();
      return 0;
}
```

(OBS: Esse “if” será explicado em aulas posteriores… aqui você só deve observar a identação)

Observe o mesmo exemplo sem o uso de identação ficaria muito mais difícil de ser lido e analisado:

Exemplo 8

```
#include <stdio.h>
int main() {
int A=0;
if ( A < 1 ) {
```

```
printf(“Variável A é menor que 1.”);
}
getchar();
return 0;
}
```

2. Comentários

Comentários são blocos de programa que servem apenas para ajudar o programador e seus colaboradores no processo de documentação do código fonte. Em programas de poucas linhas a inclusão de comentários pode parecer irrelevante, mas quando trabalhamos com programas de centenas ou milhares de linhas, a falta de comentários pode prejudicar muito o entendimento do programa alguns dias depois que ele foi escrito.

2.1 - Usando a barra dupla //

A primeira forma de incluirmos comentários em um código fonte é através de duas barras //. Esse tipo de comentário serve apenas quando formos incluir um comentário de uma única linha. Observe as linhas em negrito do exemplo abaixo:

Exemplo 9

```
#include <stdio.h>

int main() {

      //Mensagem de saudação
      printf(“Bem vindo!”);
      printf(“\nEstou aprendendo Linguagem C!”);
      printf(“\nBye, bye...”);

      //Esperando o usuário pressionar <ENTER> para sair do programa
      printf(“\n\nPressione <ENTER> para sair do programa.”);
      getchar();

      return 0;
}
```

O comentário com barra dupla pode também vir no final de uma linha que contenha um comando, como no exemplo abaixo:

Exemplo 10

```
#include <stdio.h>

int main() {

      printf(“Bem vindo!”);
      printf(“\nEstou aprendendo Linguagem C!”);
      printf(“\nBye, bye...”);
      printf(“\n\nPressione <ENTER> para sair do programa.”);

      getchar(); //Esperando o <ENTER>
      return 0;
}
```

2.2 – Comentários com /* */

Qualquer trecho de programa entre /* e */ é considerado um comentário. A grande diferença entre esse método e o método das duas barras é que dessa forma podemos criar comentários de várias linhas. Um grande uso desse tipo de comentário está nos "cabeçalhos de programa", que normalmente são colocados no início do código fonte para identificarmos o programa, conforme o exemplo abaixo:

Exemplo 11

```
/*
 Programa exemplo
 Linguagem C – Uma Introdução
 --
 Diego M. Rodrigues
*/

#include <stdio.h>

int main() {

      printf("Bem vindo!");
      printf("\nEstou aprendendo Linguagem C!");
      printf("\nBye, bye...");
      printf("\n\nPressione <ENTER> para sair do programa.");
      getchar();

      return 0;
}
```

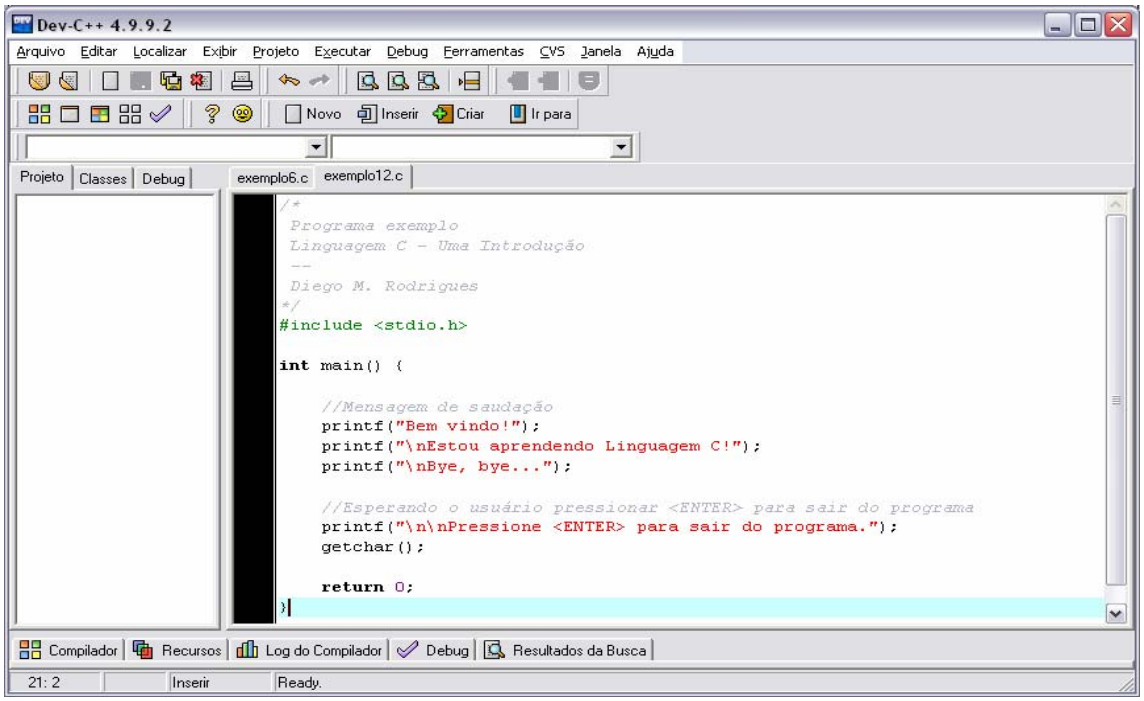
Normalmente usamos os dois tipos de comentários para uma melhor organização dos códigos fonte:

Exemplo 12

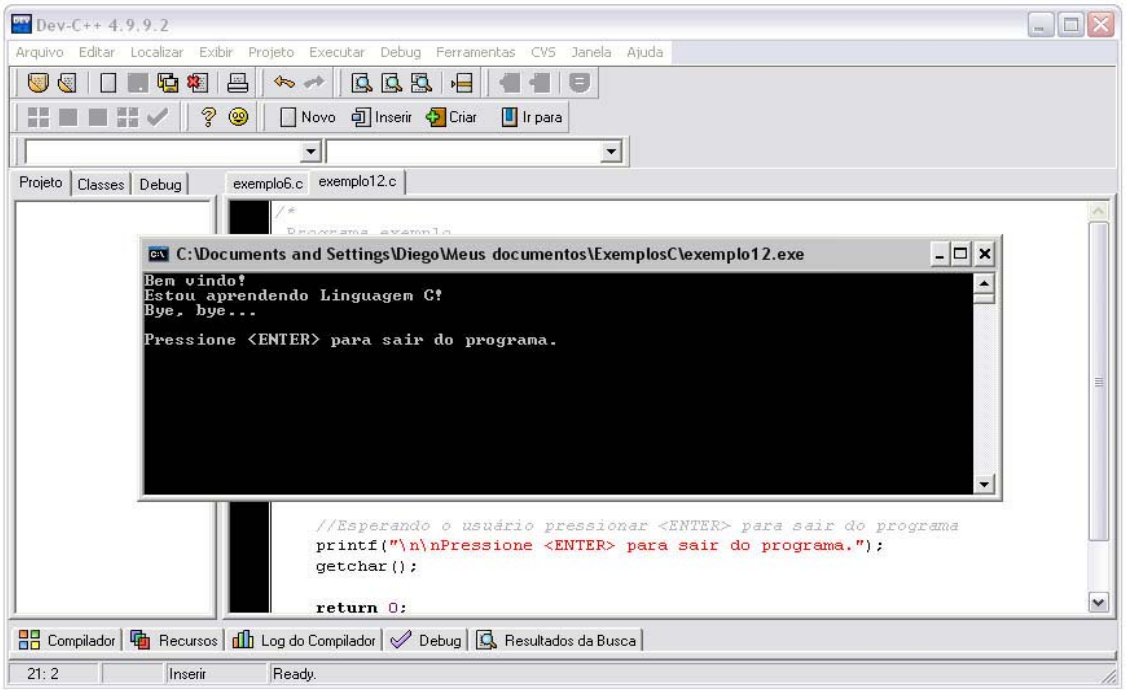
```
/*
 Programa exemplo
 Linguagem C – Uma Introdução
 --
 Diego M. Rodrigues
*/

#include <stdio.h>

int main() {

      //Mensagem de saudação
      printf("Bem vindo!");
      printf("\nEstou aprendendo Linguagem C!");
      printf("\nBye, bye...");

      //Esperando o usuário pressionar <ENTER> para sair do programa
      printf("\n\nPressione <ENTER> para sair do programa.");
      getchar();

      return 0;
}
```

2.3 – Comentários no DEV-C++

Dentro do DEV-C++ todos os comentários ficam em cor cinza, conforme pode ser visto na tela abaixo:

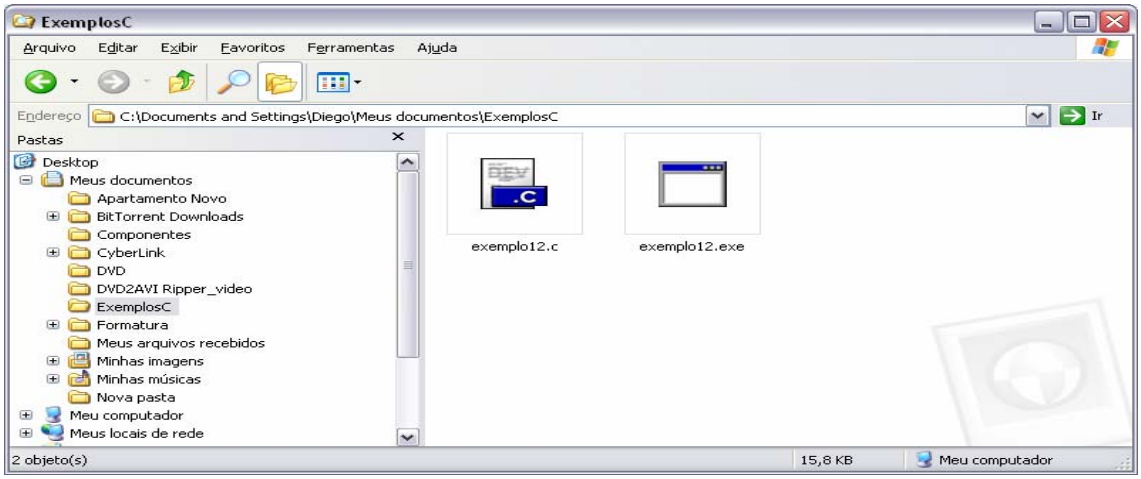


Conforme já foi dito, comentários servem apenas para ajudar o programador e seus colaboradores no processo de documentação do código fonte e não produzem nenhuma alteração no programa compilado. Observe a tela gerada na compilação do exemplo anterior:

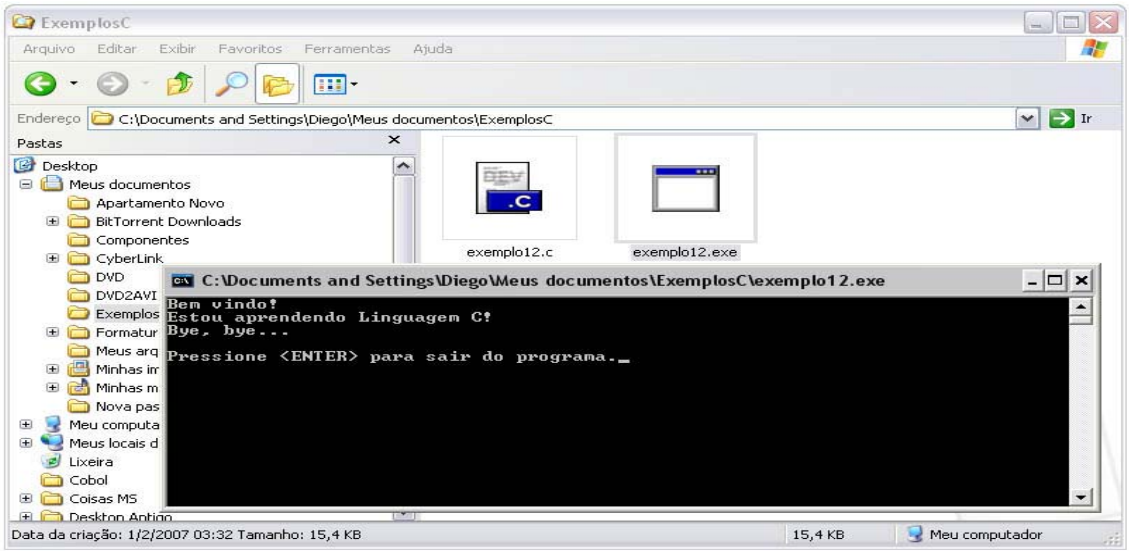


3. Código fonte X programa executável

Quando salvamos um código fonte no DEV-C++ com o tipo *"C source files (*.c)"* um arquivo com extensão *.c* é armazenado no nosso computador. Quando compilamos o nosso programa, um arquivo com extensão *.exe* é gerado na mesma pasta em que o arquivo *.c* foi salvo, este arquivo *.exe* é o nosso "programa executável", em outras palavras, ele é o resultado do processo de compilação do código fonte com extensão *.c*.



O arquivo *.exe* depois de gerado é completamente independente do código fonte. Apenas um duplo clique nesse arquivo já executa o programa.



Essa independência do programa gerado implica que quando quisermos enviar nosso programa para outra pessoa ou copiarmos o nosso programa em outro computador, basta enviarmos (ou copiarmos) o arquivo *.exe*, ou seja, não existe a necessidade de copiarmos o código fonte.

Dessa forma a pessoa que recebe o programa pronto pode apenas executá-lo, sem poder alterá-lo. Para que essa outra pessoa pudesse alterar o programa ela precisaria possuir o código fonte (arquivo .c), alterar o código fonte e depois recompilar o programa.

4. O que são Algoritmos?

“Um Algoritmo é uma seqüência de instruções ordenadas de forma lógica para a resolução de uma determinada tarefa ou problema.”

Um algoritmo é formalmente uma seqüência finita de passos que levam a execução de uma tarefa. Podemos pensar em algoritmo como uma receita, uma seqüência de instruções que dão cabo de uma meta específica. Estas tarefas não podem ser redundantes nem subjetivas na sua definição, devem ser claras e precisas.

Como exemplos de algoritmos podemos citar os algoritmos das operações básicas (adição, multiplicação, divisão e subtração) de números reais decimais. Outros exemplos seriam os manuais de aparelhos eletrônicos, como um videocassete, que explicam passo-a-passo como, por exemplo, gravar um evento.

Até mesmo as coisas mais simples, podem ser descritas por seqüências lógicas. Por exemplo, podemos descrever o algoritmo de como chupar uma bala:

“Chupar uma bala”

- Pegar a bala
- Retirar o papel
- Chupar a bala
- Jogar o papel no lixo

O algoritmo acima não pode ser implementado computacionalmente através de uma linguagem de programação. Vamos então partir para um problema real, que podemos implementar com Linguagem C. Um exemplo de algoritmo implementável é o cálculo da média de um aluno.

“Calcular média”

- Receba a nota da prova mensal
- Receba a nota da prova bimestral
- Calcule a média com a fórmula media=(mensal+bimestral)/2
- Exiba a média na tela

O programa que realiza a tarefa acima poderia ser escrito em uma pseudo linguagem da seguinte forma:

```
Variavel mensal=0
Variavel bimestral=0
Variavel media=0

ESCREVA “Digite a nota mensal:”
RECEBA mensal
ESCREVA “Digite a nota bimestral:”
RECEBA bimestral
CALCULE media=(mensal+bimestral)/2
ESCREVA “A média do aluno é:”
ESCREVA media
```

Podemos agora escrever o nosso programa em Linguagem C baseado no algoritmo acima (**esse programa será explicado detalhadamente durante as aulas 3 e 4**):

Exemplo 13

```
/*
 Exemplo de algoritmo para cálculo de média
 Linguagem C - Uma introdução
 --
 Diego M. Rodrigues
*/
#include <stdio.h>

int main() {
    //Declarando as variáveis
    float mensal=0;
    float bimestral=0;
    float media=0;

    //Recebendo a nota mensal
    printf("\nDigite a nota mensal: ");
    scanf("%f",&mensal);

    //Recebendo a nota bimestral
    printf("\nDigite a nota bimestral: ");
    scanf("%f",&bimestral);

    //Calculando a média
    media = (mensal+bimestral)/2;

    //Mostrando a média
    printf("\n\nA média do aluno é %f",media);

    //Esperando o <ENTER> para sair com getch() ao invés de getchar()
    getch();
    return 0;
}
```

Exemplo de execução do programa acima:

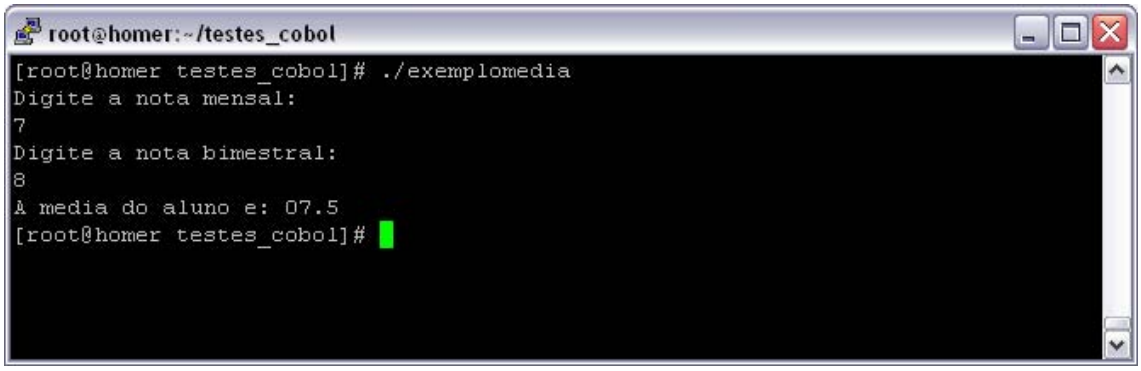
Vale ressaltar que um algoritmo é uma definição formal, completamente independente de qualquer linguagem de programação específica. Apenas como ilustração segue a implementação desse mesmo algoritmo “Calcular média” na linguagem de programação COBOL:

exemplomedia.cob

```
      * Exemplo do algoritmo CALCULAR MEDIA em COBOL
      * Liguagem C - Uma Introducao
      * Diego M. Rodrigues
            IDENTIFICATION DIVISION.
            PROGRAM-ID.    exemplomedia.
            ENVIRONMENT DIVISION.

            DATA DIVISION.
            WORKING-STORAGE SECTION.
            77  mensal  PIC 99V9  VALUE ZERO .
            77  bimestral  PIC 99V9    VALUE ZERO .
            77  media  PIC 99v9    VALUE ZERO .

            PROCEDURE DIVISION.
                DISPLAY "Digite a nota mensal:"
                ACCEPT mensal
                DISPLAY "Digite a nota bimestral:"
                ACCEPT bimestral
                COMPUTE media = (mensal+bimestral)/2.
                DISPLAY "A media do aluno e: ", media
                STOP RUN.
```

(Não tente digitar esse código no DEV-C++ que ele não sabe compilar programas na Linguagem COBOL!)

A execução desse programa compilado em COBOL apresenta a seguinte saída:



Apenas como um exemplo final, vamos implementar o mesmo algoritmo “Calcular média” em um website utilizando HTML e JavaScript. Abaixo o código fonte:

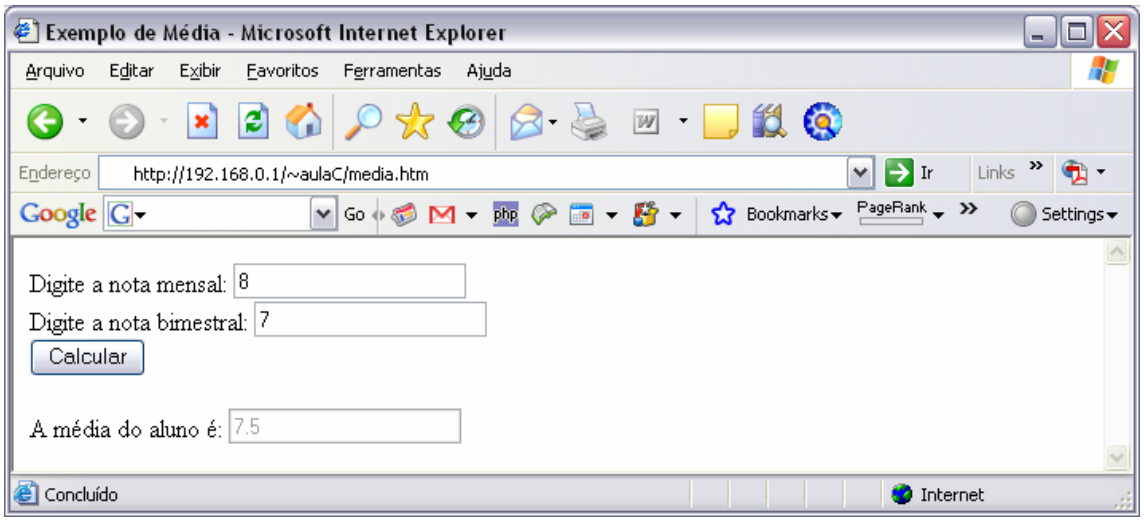
media.htm

```
<!--  Exemplo de algoritmo para cálculo de média em um website
      Linguagem C - Uma introdução
      --
      Diego M. Rodrigues
-->
<html>
<head>
       <title>Exemplo de Média</title>
       <script language="JavaScript">
              function calcular() {
                     var xMensal = parseFloat(document.aluno.mensal.value);
                     var xBimentral = parseFloat(document.aluno.bimestral.value);
                     document.aluno.media.value = (xMensal+xBimentral)/2;
              }
       </script>
```

```
</head>

<body>
	<form name="aluno" id="aluno">
		Digite a nota mensal:
		<input name="mensal" type="text" id="mensal" value="0"><br>
		Digite a nota bimestral:
		<input name="bimestral" type="text" id="bimestral" value="0"><br>
		<input type="button" name="Submit" value="Calcular"
		onClick="javascript:calcular();">
		<br><br>
		A média do aluno é:
		<input name="media" type="text" id="media" value="0" disabled>
	</form>
</body>
</html>
```

(Não tente digitar esse código no DEV-C++ por que HTML não tem nada a ver com Linguagem C!)

A execução desse arquivo HTML em um Navegador resulta na seguinte tela:



LINGUAGEM C – UMA INTRODUÇÃO

AULA 3 – Variáveis

1 – O que são variáveis

Uma variável nada mais é do que um espaço na memória do computador para armazenarmos algum tipo de informação. Todas as variáveis de um programa em Linguagem C devem ser declaradas antes de serem usadas. Isto é necessário para que seja alocada memória para as mesmas. Existem diferentes tipos de variáveis em C e o tamanho destes tipos podem variar de acordo com o processador e a implementação do compilador.

1.1 – Tipos básicos de variáveis na Linguagem C

- **int:** Esse tipo de variável armazena valores numéricos inteiro. Normalmente podem ser armazenados valores entre -32.768 e +32.767 nesse tipo de variável.
    - Exemplos de valores que podem ser armazenados em uma variável int: 100, -28, 0, 19, 28965, -19852...
    - Geralmente é armazenado na memória com a tamanho de 2 bytes (ou 16bits).

- **float:** O tipo *float* permite representar valores numéricos pertencentes ao conjunto dos números reais. Normalmente podem ser armazenados valores entre $3.4*10^{-38}$ e $3.4*10^{+38}$.
    - Exemplos de valores que podem ser armazenados em uma variável float: 75.87, -28.9985, 0.741, 199852.58, -852.87...
    - Geralmente é armazenado na memória com a tamanho de 4 bytes (ou 32 bits).

- **double:** O tipo *double* também é usado para representar valores numéricos pertencentes ao conjunto dos números reais. A diferença entre uma variável *float* e uma variável *double* é que esta última possui o dobro da precisão, eu seja, pode armazenar números muito maiores.
    - Normalmente podem ser armazenados valores entre $1.7*10^{-308}$ e $1.7*10^{+308}$
    - Geralmente é armazenado na memória com a tamanho de 8 bytes (ou 64 bits).

- **char:** Usado para armazenar um único caractere. Em outras palavras, este tipo é utilizado para se guardar valores definidos dentro da tabela ASCII (-127 a 127).
    - Exemplos de valores que podem ser armazenados em uma variável *char*: ‘v’ , ‘t’, 25...
    - Geralmente é armazenado na memória com a tamanho de 1 byte (ou 8 bits).

- **void:** Esse tipo de variável não armazena nenhum valor e é usado normalmente junto com ponteiros e funções, tópicos que serão abordados em outras aulas deste curso.

1.2 - Declarando variáveis

Toda variável que será usada em um programa em Linguagem C precisa ser declarada. O processo de declaração de variáveis é fundamental para que o compilador saiba quantos lugares (e de que tamanhos) ele dele reservar na memória do computador para que o programa que está sendo compilado possa manipular seus dados.

O formato básico de declaração de uma variável em Linguagem C é:

```
<tipo_da_variável> <nome_da_variável>;
```

Portanto, para declararmos uma variável inteira chamada *idade*, escrevemos o seguinte trecho de programa:

```
int idade;
```

Para declararmos uma variável inteira chamada idade e uma variável *float* (número real) chamada *peso*, escrevemos o seguinte trecho de programa:

```
int idade;
float peso;
```

Podemos declarar várias variáveis do mesmo tipo em uma única linha, separando seus nomes por uma vírgula, como abaixo:

```
float mensal, bimestral, media;
```

Podemos escrever um programa bem inútil, que apenas declara as variáveis *idade*, *peso*, *mensal*, *bimestral* e *media* e depois retorna. Esse exemplo não irá exibir nada na tela, nem realizar nenhuma conta, nem armazenar nenhum valor nessas variáveis, ele irá apenas declarar as variáveis.

Exemplo 14

```
#include <stdio.h>

int main() {

	int idade;
	float peso;
	float mensal, bimestral, media;

	return 0;
}
```

1.3 – Nomes de variáveis

Existem algumas regras básicas que devem ser lembradas sempre que formos nomear nossas variáveis:

- Escolha de nomes significativos para suas variáveis, isso pode ajudar a entender o que o programa faz e prevenir erros.
- Você pode usar quantos caracteres quiser para um nome de variável com o primeiro sendo obrigatoriamente uma letra ou sublinhado (_) e os demais podendo ser letras, números ou outros sublinhados.
- Nunca use espaços em branco no nome de variáveis.
- Nunca use acentos ou caracteres especiais no nome de variáveis.
- Uma variável não pode ter o mesmo nome de uma palavra reservada do C e não deverá ter o mesmo nome de uma função. Abaixo a lista de palavras reservadas da Linguagem C:

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

- Em C maiúsculas e minúsculas são tratadas como diferentes e distintas umas das outras. Por isso, cont, Cont e CONT são três variáveis distintas. Isso pode ser usado para criarmos variáveis com nomes significativos, como por exemplo *mediaDoAluno*, ou *cotacaoDoDolar* e etc...

Abaixo um exemplo com declarações de nomes de variáveis válidos:

```
int mediaDoAnulo;
double _contador;
char opcao_menu;
float a6y9u2;
```

Abaixo um exemplo com declarações de nomes de variáveis **INVÁLIDOS**:

```
float 9idade;
char opcao menu;
double cotação;
int default;
```

1.4 – Inicialização de variáveis

É possível combinar uma declaração de variável com o operador de atribuição (sina de igual) para que a variável tenha um valor no instante de sua declaração. A forma geral de inicialização é:

```
<tipo_da_variável> <nome_da_variável> = <valor>;
```

Exemplos:

```
int numero = 2;
char letra = ‘a’;
```

1.5 – Mostrando variáveis na tela

A função *printf()* foi apresentada na primeira aula do curso de uma forma simplista. Sua forma mais geral é:

```
printf(“expressão de controle”, lista de argumentos);
```

A *expressão de controle* pode conter caracteres que serão exibidos na tela e códigos de formatação que indicam o formato em que os argumentos devem ser impressos. A tabela abaixo mostra alguns códigos de formatação que podem ser utilizados pela função *printf()*:

Código	Descrição
%d	Valor *int* (ou decimal)
%f	Valor *float*
%c	Caractere simples
%s	Cadeia de caracteres
%e	Notação científica
%o	Valor octal
%u	Valor *int* sem sinal
%x	Valor hexadecimal
%ld	Valor *int* longo
%lf	Valor *float* longo

A lista de argumentos pode estar vazia, pode conter valores ou pode conter variáveis. Observe os exemplos abaixo:

```
//printf() com a lista de argumentos vazia
printf(“Estou aprendendo Linguagem C”);
```

```
//printf() com valores numéricos na lista de argumentos
printf(“Eu tenho %d anos e minha altura é %f metros”, 24, 1.7);
```

```
//printf() com variáveis na lista de argumentos
int idade = 24;
float altura = 1.7;
printf(“Eu tenho %d anos e minha altura é %f metros”, idade, altura);
```

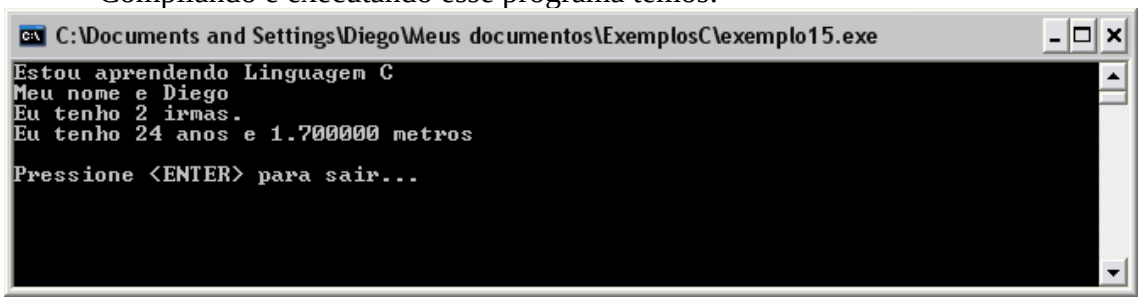
Abaixo um exemplo de programa completo:

Exemplo 15

```
/*
 Exemplo do uso da função printf()
 Linguagem C – Uma introdução
 -
 Diego M. Rodrigues
*/
#include <stdio.h>

int main() {

      //Declarando e inicializando as variáveis
      int idade = 24;
      float altura = 1.7;

      //printf() com a lista de argumentos vazia
      printf(“Estou aprendendo Linguagem C”);
```

```
      //printf() com valores na lista de argumentos
      printf("\nMeu nome e %s", "Diego");
      printf("\nEu tenho %d irmas.", 2);

      //printf() com variáveis na lista de argumentos
      printf("\nEu tenho %d anos e %f metros", idade, altura);

      //Esperando o <ENTER> para sair do programa
      printf("\n\nPressione <ENTER> para sair...");
      getchar();

      return 0;
}
```

Compilando e executando esse programa temos:



2. Operadores aritméticos

Os operadores aritméticos são usados para calcular expressões matemáticas. Sendo classificados em duas categorias: os binários ou unários. Os operadores unários atuam na inversão de valores. Veja a tabela abaixo.

Operador binário	Descrição
=	Atribuição
+	Soma
-	Subtração
*	Multiplicação
/	Divisão
%	Resto da divisão

Operador unário	Descrição
-	Sinal negativo
+	Sinal positivo

A forma geral de uso dos operadores binários é:

```
<variável_que_recebe> = <operando1> <operador> <operando2>;
```

Exemplos:

```
somaDasNotas = mensal + bimestral;
diasDoAno = meses * diasDoMes;
metade = notaTotal / 2;
restoDaDivisao = notaTotal % 2;
```

A forma geral de uso dos operadores unários é:

```
<variável_que_recebe> = <operador><operando1>;
```

Exemplo:

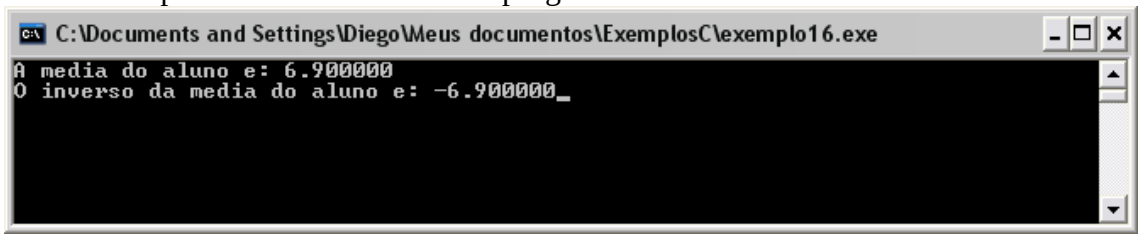
```
inversoDaNota = -mensal;
```

Segue abaixo um programa completo que calcula a média de um aluno com base nas notas atribuídas às variáveis *mensal* e *bimestral* dentro do código fonte.

Exemplo 16

```
/*
 Exemplo do uso de operadores aritméticos
 Linguagem C – Uma introdução
 -
 Diego M. Rodrigues
*/
#include <stdio.h>

int main() {

      //Declarando as variáveis
      float mensal, bimestral;
      float soma, media, inverso;

      //Usando o operador de atribuição
      mensal = 6.3;
      bimestral = 7.5;

      //Usando o operador de soma
      soma = mensal + bimestral;

      //Usando o operador de divisão
      media = soma/2;

      //Exibindo a média na tela
      printf("A media do aluno e: %f", media);

      //Usando o operador unário negativo
      inverso = -media;
      printf("\nO inverso da media do aluno e: %f", inverso);

      //Esperando o <ENTER> para sair do programa
      getchar();

      return 0;
}
```

Compilando e executando esse programa temos:



2.1 – Operadores de incremento e decremento

São dois operadores bastante úteis para simplificar expressões: ++ (incremento de 1) e -- (decremento de 1).

Veja o exemplo abaixo escrito sem usar os operadores de incremento e decremento:

```
int contador = 10;
int numero = 2;
//numero tem o valor 2 e contador tem o valor 10...
numero = numero + 1;
contador = contador – 1;
//...agora numero tem o valor 3 e contador tem o valor 9
```

Agora o mesmo programa usando operadores de incremento e decremento:

```
int contador = 10;
int numero = 2;
//numero tem o valor 2 e contador tem o valor 10...
numero++;
contador--;
//...agora numero tem o valor 3 e contador tem o valor 9
```

Os operadores de incremento e decremento podem ser usados junto com o operador de atribuição, conforme os exemplos abaixo:

```
int x, var;
x = 2;
var = ++x;
//Agora o valor de var será 3 e x será 3
```

Neste caso, colocamos o operador de incremento ++ antes da variável *x*, então o valor de *x* será incrementado antes de ser atribuído à variável *var*.

```
int y, var;
y = 2;
var = y++;
//Agora o valor de var será 2 e y será 3
```

Neste caso, colocamos o operador de incremento ++ depois da variável *y*, então o valor de *y* será primeiro atribuído à variável *var* e depois será incrementado.

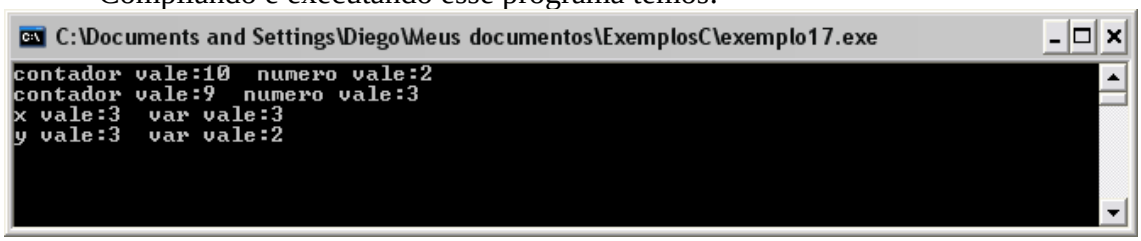
Segue abaixo um programa completo que ilustra o uso dos operadores de incremento e decremento:

Exemplo 17

```
/*
 Exemplo do uso de operadores de incremento e decremento
 Linguagem C – Uma introdução
 -
 Diego M. Rodrigues
*/
#include <stdio.h>

int main() {

      //Declarando as variáveis
      int contador = 10;
      int numero = 2;
      int x, y, var;

      //Exibindo valores iniciais de contador e numero
```

```
      printf("contador vale:%d  numero vale:%d", contador, numero);

      //Usando operadores de incremento e decremento
      numero++;
      contador--;

      //Exibindo novos valores de contador e numero
      printf("\ncontador vale:%d  numero vale:%d", contador, numero);

      //Colocando valores iniciais em x e var
      x = 2;
      var = 0;

      //Usando o operador de incremento junto com o de atribuição
      var = ++x;

      //Exibindo os valores de x e var
      printf("\nx vale:%d  var vale:%d", x, var);

      //Colocando valores iniciais em x e var
      y = 2;
      var = 0;

      //Usando o operador de incremento junto com o de atribuição
      var = y++;

      //Exibindo os valores de x e var
      printf("\ny vale:%d  var vale:%d", y, var);

      //Esperando o <ENTER> para sair do programa
      getchar();

      return 0;
}
```

Compilando e executando esse programa temos:



2.2 - Operadores aritméticos de atribuições

São combinações de operadores que simplificam (reduzem) as instruções e que geralmente são usados por programadores experientes. As formas básicas de uso são:

Instrução normal	Instrução reduzida
var = var + expr	var += expr
var = var - expr	var -= expr
var = var * expr	var *= expr
var = var / expr	var /= expr

Acompanhe os exemplos abaixo:

```
int quantidade = 10;
quantidade = quantidade + 3;
```

pode ser reduzida para:

```
int quantidade = 10;
quantidade += 3;
```

```
int quantidade = 10;
quantidade = quantidade * 2;
```

pode ser reduzida para:

```
int quantidade = 10;
quantidade *= 2;
```

```
int quantidade = 10;
int vendas = 2;
quantidade = quantidade - vendas;
```

pode ser reduzida para:

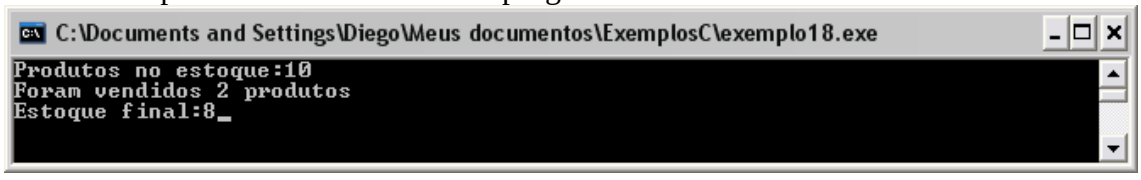
```
int quantidade = 10;
int vendas = 2;
quantidade -= vendas;
```

Segue abaixo um programa completo que ilustra os conceitos acima:

Exemplo 18

```
/*
 Exemplo do uso de operadores compostos
 Linguagem C – Uma introdução
 -
 Diego M. Rodrigues
*/
#include <stdio.h>

int main() {

      //Declarando as variáveis
      int quantidade = 10;
      int vendas = 2;

      //Exibindo o estoque inicial
      printf("Produtos no estoque:%d", quantidade);

      //Tirando os itens vendidos do estoque
      printf("\nForam vendidos %d produtos", vendas);
      quantidade -= vendas;

      //Exibindo o estoque final
      printf("\nEstoque final:%d", quantidade);

      //Esperando o <ENTER> para sair do programa
      getchar();
      return 0;
}
```

Compilando e executando esse programa temos:



LINGUAGEM C – UMA INTRODUÇÃO

AULA 4 – Entrada e saída

1 – Entrada, saída, hãmm?

O conceito de entrada e saída já foi utilizado de forma intuitiva nos exemplos das aulas anteriores. Entendemos por "entrada" qualquer forma de passarmos dados para o programa e por "saída" qualquer forma do programa enviar dados para o mundo.

Tipos mais comuns:

- **Entrada:** usuário respondendo uma pergunta, programa lendo um arquivo, programa capturando informações de um sensor eletrônico ligado ao computador, parâmetros passados ao programa na linha de comando...
- **Saída:** programa mostrando informações na tela, programa enviando dados para uma impressora, programa acionando um equipamento eletrônico ligado ao computador...

2 – Entrada com a função scanf()

A função *scanf()* é usada para recebermos dados do teclado de uma forma "formatada". Ela foi usada intuitivamente em alguns exemplos passados do curso (exemplo 1, exemplo 3, exemplo 13, etc..) e agora chegou a hora de entendermos melhor como ela funciona. Sua sintaxe é:

```
scanf("especificador de formato",&variável)
```

O primeiro ponto importante a ser observado é que a função *scanf()* sempre armazena o valor digitado no teclado em uma variável.

O segundo ponto importante é que antes do nome da variável existe um **&** e esse **& é OBRIGATÓRIO**. O programa não irá funcionar sem ele.

Abaixo um exemplo de como ler um valor numérico e inteiro do teclado:

```
int idade;
scanf("%d",&idade);
```

O *"especificador de formato"* da função *scanf()* segue a mesma sintaxe da *"expressão de controle"* da função *printf()*. Os seguintes códigos especiais devem ser usados:

Código	Descrição
%d	Valor *int* (ou decimal)
%f	Valor *float*

%c	Caractere simples
%s	Cadeia de caracteres
%e	Notação científica
%o	Valor octal
%u	Valor *int* sem sinal
%x	Valor hexadecimal
%ld	Valor *int* longo
%lf	Valor *float* longo

Exemplos de uso da função *scanf()*:

```
float peso;
scanf("%f",&peso);
```

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

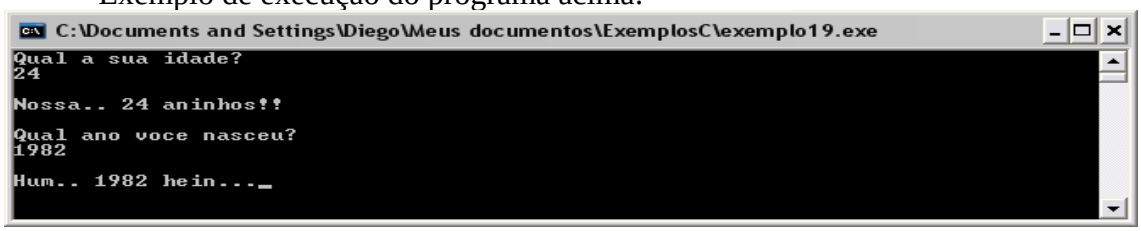
```
int ano;
scanf("%d",&ano);
```

Veja abaixo um exemplo completo usando a função *scanf()*:

Exemplo 19

```
/*
 Exemplo de entrada de dados com a função scanf()
 Linguagem C – Uma introdução
 -
 Diego M. Rodrigues
*/
#include <stdio.h>

int main() {

    int idade, ano;

    printf("Qual a sua idade?\n");
    //Recebendo a variável idade
    scanf("%d", &idade);

    //Mostrando a variável idade na tela
    printf("\nNossa.. %d aninhos!!\n", idade);

    printf("\nQual ano voce nasceu?\n");
    //Recebendo a variável ano
    scanf("%d", &ano);

    //Mostrando na tela a variável letra
    printf("\nHum.. %d hein...",ano);

    //Esperando o <ENTER> para sair com getch() ao invés de getchar()
    getch();
    return 0;
}
```

Exemplo de execução do programa acima:



2.1 – Revendo o algoritmo “Calcular média”

Agora que a função *scanf()* foi devidamente apresentada, podemos rever o algoritmo “Calcular média”, apresentado na Aula 2.

Observe que sempre que encontrarmos coisas do tipo “Receba”, “Leia” e etc em um algoritmo, podemos usar a função *scanf()* para implementar essa instrução e sempre que encontrarmos coisas do tipo “Exiba”, “Escreva”, “Mostre” e etc podemos usar a função *printf()* para realizar essa tarefa.

“Calcular média”

- Receba a nota da prova mensal
- Receba a nota da prova bimestral
- Calcule a média com a fórmula media=(mensal+bimestral)/2
- Exiba a média na tela

O programa que realiza a tarefa acima poderia ser escrito em uma pseudo linguagem da seguinte forma:

```
Variavel mensal=0
Variavel bimestral=0
Variavel media=0

ESCREVA “Digite a nota mensal:”
RECEBA mensal
ESCREVA “Digite a nota bimestral:”
RECEBA bimestral
CALCULE media=(mensal+bimestral)/2
ESCREVA “A média do aluno é:”
ESCREVA media
```

Uma implementação do algoritmo “Calcular média” em Linguagem C pode ser escrita da seguinte forma:

Exemplo 20

```c
/*
 Exemplo de algoritmo para cálculo de média
 Linguagem C - Uma introdução
 --
 Diego M. Rodrigues
*/
#include <stdio.h>

int main() {
    //Declarando as variáveis
    float mensal=0;
    float bimestral=0;
    float media=0;

    //Recebendo a nota mensal
    printf("\nDigite a nota mensal: ");
    scanf("%f",&mensal);

    //Recebendo a nota bimestral
    printf("\nDigite a nota bimestral: ");
    scanf("%f",&bimestral);
```

```
    //Calculando a média
    media = (mensal+bimestral)/2;

    //Mostrando a média
    printf("\n\nA média do aluno é %f",media);

    //Esperando o <ENTER> para sair com getch() ao invés de getchar()
    getch();
    return 0;
}
```

3. Outros exemplos com scanf() e printf()

3.1 – Área do quadrado

Problema: Implemente um programa em Linguagem C para calcular a área de um quadrado. Esse programa deve receber o tamanho do lado do quadrado e imprimir a sua área na tela.

Um algoritmo para resolver o problema proposto poderia ser:

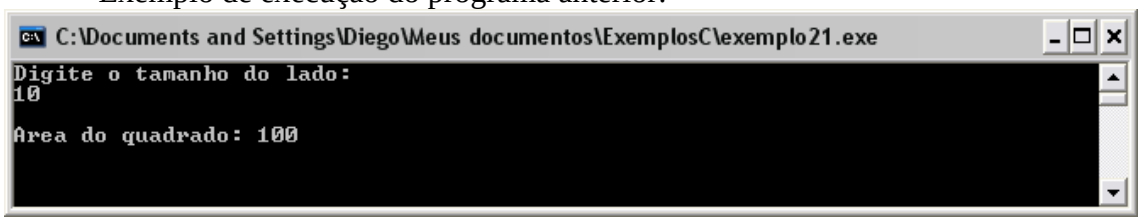
- Receba o lado do quadrado
- Calcule a área com a fórmula area=lado*lado
- Exiba a variável área na tela

Implementando em Linguagem C:

Exemplo 21

```
/*
 Exemplo de algoritmo para cálculo da área de um quadrado
 Linguagem C - Uma introdução
 --
 Diego M. Rodrigues
*/
#include <stdio.h>

int main() {
      //Declarando as variáveis
      int lado=0, area=0;

      //Recebendo o lado
      printf("Digite o tamanho do lado:\n");
      scanf("%d", &lado);

      //Calculando a área
      area = lado * lado;

      //Mostrando a área na tela
      printf("\nArea do quadrado: %d", area);

      //Esperando o <ENTER> para sair com getch() ao invés de getchar()
      getch();

      return 0;
}
```

Exemplo de execução do programa anterior:



3.2 – Índice de massa corporal

O índice de Massa Corporal (IMC) é uma fórmula que indica se um adulto está acima do peso, se está obeso ou abaixo do peso ideal considerado saudável. A fórmula para calcular o Índice de Massa Corporal é:

IMC = peso / $(altura)^2$

A Organização Mundial de Saúde usa a seguinte tabela para determinar a condição de um adulto:

Condição	IMC em adultos
Abaixo do peso	Abaixo de 18.5
No peso normal	Entre 18.5 e 25
Acima do peso	Entre 25 e 30
Obeso	Acima de 30

Problema: Implemente um programa em Linguagem C para calcular o IMC de um adulto. Esse programa deve receber a altura e o peso da pessoa e exibir o IMC na tela.

Um algoritmo para resolver o problema proposto poderia ser:

- Receba a altura da pessoa
- Receba o peso da pessoa
- Calcule a altura ao quadrado com a fórmula quadrado=altura*altura
- Calcule o IMC com a fórmula imc=peso/quadrado
- Exiba a variável imc na tela

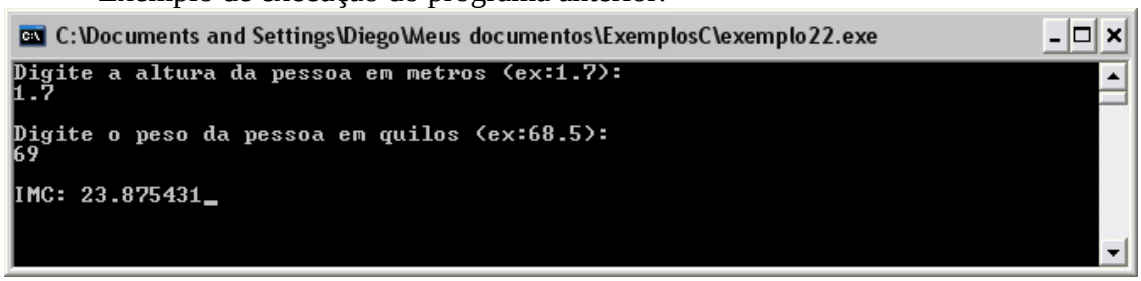
Implementando em Linguagem C:

Exemplo 22

```
/*
 Exemplo de algoritmo para cálculo do IMC de um adulto
 Linguagem C - Uma introdução
 --
 Diego M. Rodrigues
*/
#include <stdio.h>

int main() {
      //Declarando as variáveis
      float peso=0, altura=0;
      float quadrado=0, imc=0;

      //Recebendo a altura
      printf("Digite a altura da pessoa em metros (ex:1.7):\n");
      scanf("%f", &altura);
```

```
      //Recebendo o peso
      printf("\nDigite o peso da pessoa em quilos (ex:68.5):\n");
      scanf("%f", &peso);

      //Calculando o quadrado da altura
      quadrado = altura * altura;

      //Calculando o IMC
      imc = peso / quadrado;

      //Mostrando o IMC na tela
      printf("\nIMC: %f", imc);

      //Esperando o <ENTER> para sair com getch() ao invés de getchar()
      getch();

      return 0;
}
```

Exemplo de execução do programa anterior:



4 - Tamanho de Campos na Impressão

No uso da função *printf()* é possível estabelecer o tamanho mínimo para a impressão de um campo colocando o tamanho depois do %.

Quando usamos %d, estamos dizendo para o compilador que ele deve imprimir um número inteiro. Já quando colocamos %4d dizemos ele deve imprimir um número inteiro com no mínimo 4 posições.

Exemplo 23

```
/*
 Limitando o tamanho da impressão na função printf()
 Linguagem C - Uma introdução
 --
 Diego M. Rodrigues
*/
 #include <stdio.h>

int main()
{
    printf("Os alunos sao %3d \n", 350);
    printf("Os alunos sao %4d \n", 350);
    printf("Os alunos sao %5d \n", 350);

    getchar();
    return 0;
}
```

Exemplo de execução do programa anterior:

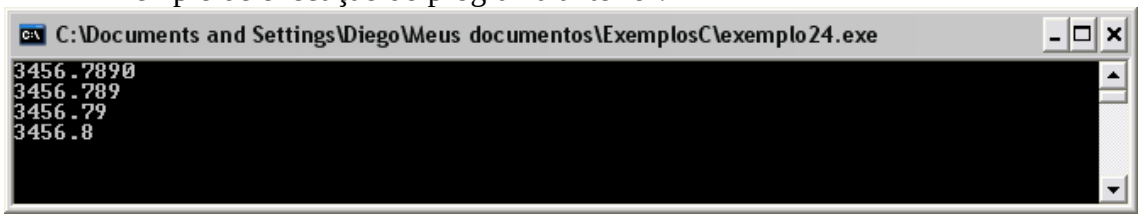
Podemos também especificar a precisão do arredondamento de variáveis *float* ou *double* usando um ponto depois do %. Dessa forma, %.4 determina que será impresso um número real (*float*) com quatro casas decimais.

Observe o exemplo:

Exemplo 24

```
/*
 Especificando o arredondamento na função printf()
 Linguagem C - Uma introdução
 --
 Diego M. Rodrigues
*/
#include <stdio.h>
int main()
{
      printf("%.4f\n", 3456.789);
      printf("%.3f\n", 3456.789);
      printf("%.2f\n", 3456.789);
      printf("%.1f\n", 3456.789);

      getchar();
      return 0;
}
```

Exemplo de execução do programa anterior:



Podemos misturar os dois conceitos anteriores através de coisas do tipo %7.3f ou %10.2f. Este %10.2f deve ser entendido da seguinte forma: variável float com um tamanho de 10 e duas casas depois da vírgula.

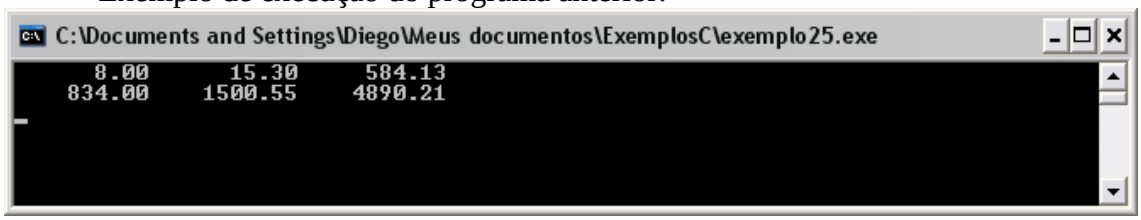
Observe o exemplo:

Exemplo 25

```
/*
 Especificando o arredondamento e o tamanho na função printf()
 Linguagem C - Uma introdução
 --
 Diego M. Rodrigues
*/
#include <stdio.h>
int main()
{
      printf("%10.2f %10.2f %10.2f\n", 8.0, 15.3, 584.13);
      printf("%10.2f %10.2f %10.2f\n", 834.0, 1500.55, 4890.21);
```

```
	getchar();
	return 0;
}
```

Exemplo de execução do programa anterior:

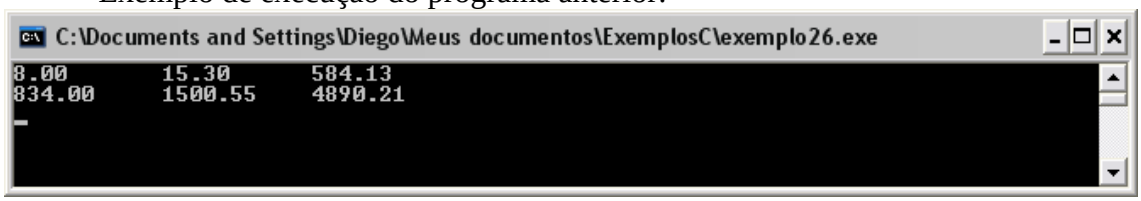


O sinal de menos (-) precedendo a especificação do tamanho do campo justifica os campos à esquerda. Exemplo:

Exemplo 26

```
/*
 Especificando o arredondamento, o tamanho e alinhando à esquerda
 na função printf()
 Linguagem C - Uma introdução
 --
 Diego M. Rodrigues
*/
#include <stdio.h>
int main()
{
	printf("%-10.2f %-10.2f %-10.2f\n", 8.0, 15.3, 584.13);
	printf("%-10.2f %-10.2f %-10.2f\n", 834.0, 1500.55, 4890.21);

	getchar();
	return 0;
}
```

Exemplo de execução do programa anterior:



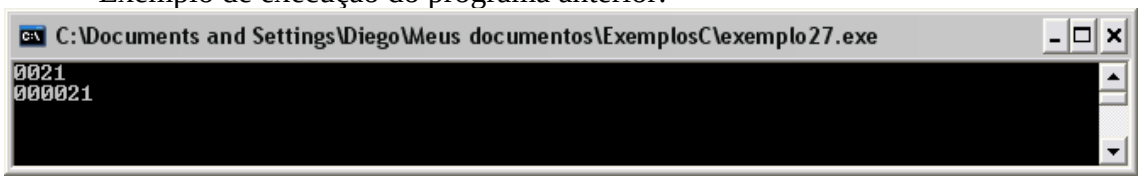
Além de especificar o tamanho do campo, podemos complementar o campo todo ou parte dele com zeros à esquerda, para isso colocamos um zero depois do %. Exemplo:

Exemplo 27

```
/*
 Completando o tamanho do campo com zeros na função printf()
 Linguagem C - Uma introdução
 --
 Diego M. Rodrigues
*/
#include <stdio.h>
int main()
{
	printf("%04d\n",21);
	printf("%06d\n",21);

	getchar();
```

```
	return 0;
}
```

Exemplo de execução do programa anterior:



5. Imprimindo Caracteres

Em Linguagem C um caractere pode ser representado de diversas maneiras: o próprio caracter entre aspas simples ou sua representação decimal, hexadecimal ou octal segundo a tabela ASCII.

A tabela ASCII tem 256 códigos decimais numerados de 0 a 255. Cada código decimal corresponde à um caracter distinto, por exemplo, o decimal 65 corresponde ao caracter A, decimal 70 corresponde ao caracter F, o decimal 33 corresponde ao ponto de exclamação...

TABELA ASCII – Parte 1

código	caracter	código	caracter	código	caracter	código	caracter
0	CTRL-@	32	(BRANCO)	64	@	96	`
1	CTRL-A	33	!	65	A	97	a
2	CTRL-B	34	"	66	B	98	b
3	CTRL-C	35	#	67	C	99	c
4	CTRL-D	36	$	68	D	100	d
5	CTRL-E	37	%	69	E	101	e
6	CTRL-F	38	&	70	F	102	f
7	CTRL-G	39	'	71	G	103	g
8	CTRL-H	40	(	72	H	104	h
9	CTRL-I	41	)	73	I	105	i
10	CTRL-J	42	*	74	J	106	j
11	CTRL-K	43	+	75	K	107	k
12	CTRL-L	44	,	76	L	108	l
13	CTRL-M	45	-	77	M	109	m
14	CTRL-N	46	.	78	N	110	n
15	CTRL-O	47	/	79	O	111	o
16	CTRL-P	48	0	80	P	112	p
17	CTRL-Q	49	1	81	Q	113	q
18	CTRL-R	50	2	82	R	114	r
19	CTRL-S	51	3	83	S	115	s
20	CTRL-T	52	4	84	T	116	t
21	CTRL-U	53	5	85	U	117	u
22	CTRL-V	54	6	86	V	118	v
23	CTRL-W	55	7	87	W	119	w
24	CTRL-X	56	8	88	X	120	x
25	CTRL-Y	57	9	89	Y	121	y
26	CTRL-Z	58	:	90	Z	122	z
27	CTRL-[	59	;	91	[	123	{

28	CTRL-\	60	<	92	\	124	|
29	CTRL-]	61	=	93	]	125	}
30	CTRL-^	62	>	94	^	126	~
31	CTRL-_	63	?	95	_	127	DEL

TABELA ASCII – Parte 2							
código	**caracter**	**código**	**caracter**	**código**	**caracter**	**código**	**caracter**
128	Ç	160	á	192	+	224	Ó
129	ü	161	í	193	-	225	ß
130	é	162	ó	194	-	226	Ô
131	â	163	ú	195	+	227	Ò
132	ä	164	ñ	196	-	228	õ
133	à	165	Ñ	197	+	229	Õ
134	å	166	ª	198	ã	230	µ
135	ç	167	º	199	Ã	231	þ
136	ê	168	¿	200	+	232	Þ
137	ë	169	®	201	+	233	Ú
138	è	170	¬	202	-	234	Û
139	ï	171	½	203	-	235	Ù
140	î	172	¼	204	¦	236	ý
141	ì	173	¡	205	-	237	Ý
142	Ä	174	«	206	+	238	¯
143	Å	175	»	207	¤	239	´
144	É	176	_	208	ð	240	
145	æ	177	_	209	Ð	241	±
146	Æ	178	_	210	Ê	242	_
147	ô	179	¦	211	Ë	243	¾
148	ö	180	¦	212	È	244	¶
149	ò	181	Á	213	i	245	§
150	û	182	Â	214	Í	246	÷
151	ù	183	À	215	Î	247	¸
152	ÿ	184	©	216	Ï	248	°
153	Ö	185	¦	217	+	249	¨
154	Ü	186	¦	218	+	250	·
155	ø	187	+	219	_	251	¹
156	£	188	+	220	_	252	³
157	Ø	189	¢	221	¦	253	²
158	×	190	¥	222	Ì	254	_
159	ƒ	191	+	223	_	255	

O seguinte trecho de programa imprime o valor ASCII da letra ‘d’:

```
printf(“%d”, ‘d’);
```

Já o trecho abaixo faz o inverso, imprime a letra ‘d’ a partir de seu código ASCII:

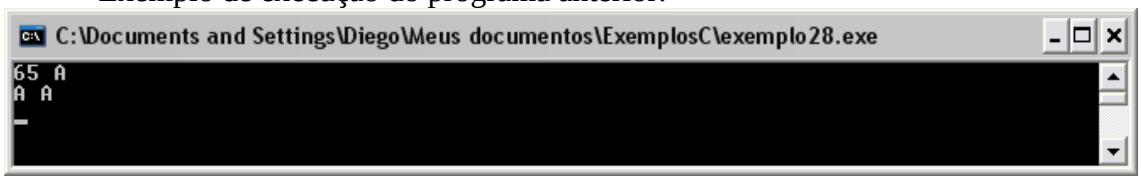
```
printf(“%c”, 100);
```

Exemplo 28

```
/*
 Imprimindo Caracteres
 Linguagem C - Uma introdução
 --
 Diego M. Rodrigues
```

```
*/
#include <stdio.h>
int main()
{
      //Mostrando o valor decimal da letra A e depois a letra A
      printf("%d %c \n", 'A', 'A');

      //Mostrando a letra A usando o seu valor decimal
      printf("%c %c \n", 'A', 65);

      getchar();
      return 0;
}
```

Exemplo de execução do programa anterior:



5.1 – Códigos especiais

Em quase todos os exemplos apresentados até agora usamos o código especial \n na função *printf()* para pular linhas. Existem outros códigos especiais que podem ser usados:

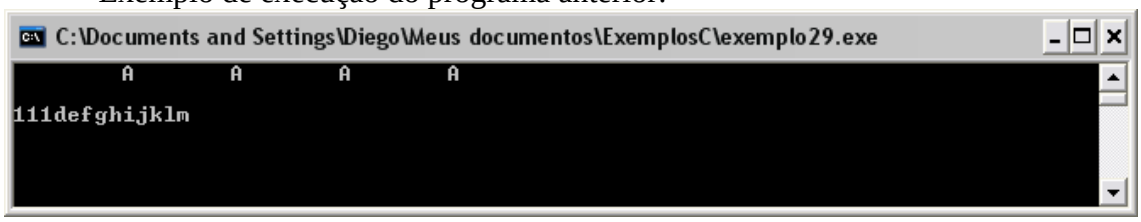
Código	Significado
\n	Linha Nova (Line Feed)
\b	Retrocesso (BackSpace)
\f	Salto de Página (Form Feed)
\r	Retorno do Carro (cr)
\t	Tabulação Horizontal (TAB)
\'	Caracter com apóstrofo
\0	Caracter Nulo ou Fim de String
\x	Representação hexadecimal

Exemplo 29

```
/*
 Usando caracteres especiais
 Linguagem C - Uma introdução
 Diego M. Rodrigues
*/
#include <stdio.h>
int main()
{
      //Usando a tabulação
      printf("\tA\tA\tA\tA");

      //Pulando duas linhas
      printf("\n\n");

      //Usando o retorno
      printf("abcdefghijklm");
      printf("\r111");

      getchar();
```

```
	return 0;
}
```

Exemplo de execução do programa anterior:



A tela acima podemos notar as quatro letras "*A*" separadas por tabulações na primeira linha. Depois foram puladas duas linhas e a seqüência "*abcdefghijklm*" foi impressa na tela. Finalmente foi usando um retrocesso (cursor volta para o início da linha) e em seguida foi impressa a seqüência "*111*". Observe que a seqüência "*111*" sobrescreveu a "*abcdefghijklm*".

LINGUAGEM C – UMA INTRODUÇÃO

AULA 5 – Expressões e Operadores

1 – Precedência dos operadores aritméticos

Na Aula 3 do nosso curso foram apresentados os Operadores Aritméticos (binários, unários, de incremento e decremento e operadores de atribuições). A primeira parte dessa nossa Aula 5 é apresentar maneiras de combinarmos esses operadores para formarmos expressões matemáticas ou expressões lógicas.

Apenas como revisão, segue uma lista com os operadores já estudados até esse ponto de nosso curso:

Operador binário	Descrição
=	Atribuição
+	Soma
-	Subtração
*	Multiplicação
/	Divisão
%	Resto da divisão

Operador unário	Descrição
-	Sinal negativo
+	Sinal positivo

Operador	Descrição
++	Incrementa 1
--	Decrementa 1

Instrução normal	Instrução reduzida
var = var + expr	var += expr
var = var - expr	var -= expr
var = var * expr	var *= expr
var = var / expr	var /= expr

Até este ponto, você deve ser capaz de entender expressões do tipo:

```
int nota1 = 7;
int nota2 = 8;
int soma = nota1 + nota2;
//O valor de soma é 15
```

```
float nota1 = 7.3;
float nota2 = 8.4;
float soma;
float media;
soma = nota1 + nota2;
media = soma/2;
```

```
//O valor de média é 7.85
```

```
int contador=1;
contador++;
//O valor de contador é 2
```

```
int contador=1;
contador += 5;
contador = -contador;
//O valor de contador é -6
```

Agora começaremos a analisar expressões um pouco mais complicadas, mais próximas de problemas reais implementados computacionalmente. Vejamos alguns exemplos:

```
float nota1 = 7.3;
float nota2 = 8.4;
float media;
media = nota1 + nota2 / 2; //Expressão 1
```

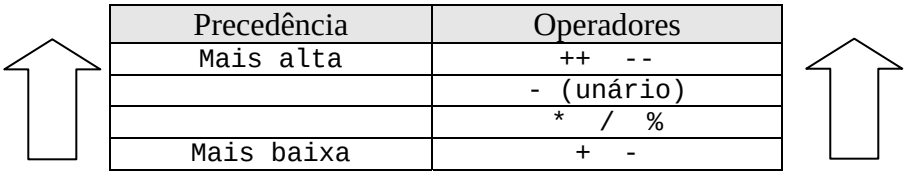
Pergunta 1: Qual o valor da variável *media*?

```
float nota1 = 7.3;
float nota2 = 8.4;
float media;
media = ++nota1 + nota2 / 2; //Expressão 2
```

Pergunta 2: Qual o valor da variável *media*?

1.2 – Tabela de precedência

A seguinte tabela resume a ordem de precedência dos operadores em Linguagem C (e em praticamente todas as linguagens de programação):



Precedência	Operadores
Mais alta	++ --
	- (unário)
	* / %
Mais baixa	+ -

Com base na tabela acima, vamos analisar as duas expressões propostas anteriormente:

Expressão 1: `media = nota1 + nota2 / 2;`

O operador divisão *(/)* possui uma precedência mais alta que o operador adição *(+)*, portanto a primeira operação que ocorre é a divisão de *nota2* por *2*. O resultado dessa divisão é somado ao valor de *nota1* e finalmente o valor dessa soma é atribuído à variável *media* através do operador de atribuição *(=)*.

Logo, a resposta da "Pergunta 1" é: `7.3 + 4.2 = 11.5`

Expressão 2: `media = ++nota1 + nota2 / 2;`

O operador de incremento *(++)* possui precedência sobre os operadores de divisão e soma, então a primeira operação realizada é o incremento da variável *nota1* (ela passa a valer 8.3) . Agora o operador divisão *(/)* possui uma precedência mais alta

que o operador adição *(+)*, portanto a próxima operação que ocorre é a divisão de *nota2* por *2*. O resultado dessa divisão é somado ao valor incrementado de *nota1* e finalmente o valor dessa soma é atribuído à variável *media* através do operador de atribuição (=).

Logo, a resposta da "Pergunta 4" é: `8.3 + 4.2 = 12.5`

1.3 – Parênteses

Os parênteses são utilizados para alterarmos a ordem de precedência dos operadores. Assim, o cálculo correto da média poderia ser feito da seguinte forma:

```
float nota1 = 7.3;
float nota2 = 8.4;
float media;
media = ( nota1 + nota2 ) / 2; //Expressão 3
```

Pergunta 3: Qual o valor da variável *media*?

Na expressão 3, primeiro será analisado o que está entre parênteses, ou seja, primeiro *nota1* será somado à *nota2*. Essa soma então será dividida por 2 e o resultado da divisão será atribuído à variável *média*.

Logo, a resposta da "Pergunta 3" é: `(15.7)/2 = 7.85`

Devemos lembrar que dentro dos parênteses, a tabela de precedência continua valendo, veja o exemplo abaixo:

```
float nota1 = 7.3;
float nota2 = 8.4;
float media;
media = ( ++nota1 + nota2 ) / 2; //Expressão 4
```

Pergunta 4: Qual o valor da variável *media*?

Na expressão 4, primeiro será analisado o que está entre parênteses, ou seja, `++nota1 + nota2`. Só que temos dois operadores dentro desses parênteses, então de acordo com a tabela de precedência, primeiro *nota1* será incrementada e depois esse resultado será somado à variável *nota2*. Essa soma então será dividida por 2 e o resultado da divisão será atribuído à variável média.

Logo, a resposta da "Pergunta 4" é: `(8.3 + 8.4)/2 = (16.7)/2 = 8.35`

Estamos prontos para resolver o seguinte problema: A média de um aluno é calculada da seguinte forma:

Existem duas provas, uma mensal e uma bimestral, cada uma valendo de 0 a 10.

Existe um trabalho valendo de 0 a 10.

80% da média correspondem à média de provas

20% da média correspondem à nota de trabalho

Problema: Implemente um programa em Linguagem C que calcule a média de um aluno de acordo com o critério de avaliação acima proposto.

Um algoritmo para resolver o problema proposto poderia ser:

- Receba a nota mensal
- Receba a nota bimestral

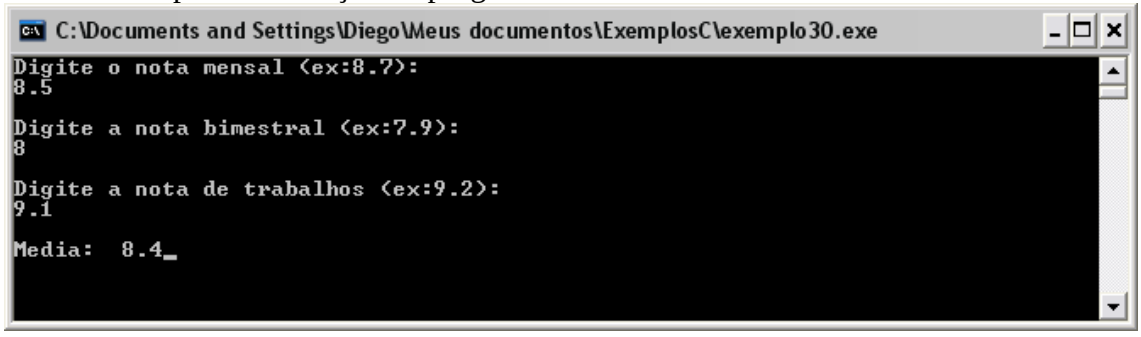
- Receba a nota de trabalhos
- Calcule a média de provas com a fórmula mediaProvas=(mensal+bimestral)/2
- Calcule a média do aluno com a fórmula media=0.8*mediaProvas+0.2*trabalhos
- Exiba a média do aluno a tela

Implementando em Linguagem C:

Exemplo 30

```
/*
 Exemplo de cálculo de média – Precedência de Operadores
 Linguagem C - Uma introdução
 --
 Diego M. Rodrigues
*/
#include <stdio.h>

int main() {
      //Declarando as variáveis
      float mensal=0, bimestral=0, trabalhos=0;
      float mediaProvas=0, media=0;

      //Recebendo a nota mensal
      printf("Digite o nota mensal (ex:8.7):\n");
      scanf("%f", &mensal);

      //Recebendo a nota bimestral
      printf("\nDigite a nota bimestral (ex:7.9):\n");
      scanf("%f", &bimestral);

      //Recebendo a nota bimestral
      printf("\nDigite a nota de trabalhos (ex:9.2):\n");
      scanf("%f", &trabalhos);

      //Calculando a média
      mediaProvas = (mensal + bimestral)/2;
      media=0.8*mediaProvas + 0.2*trabalhos;

      //Mostrando a média na tela
      printf("\nMedia: %4.1f", media);

      //Esperando o <ENTER> para sair com getch() ao invés de getchar()
      getch();

      return 0;
}
```

Exemplo de execução do programa anterior:



Poderíamos deixar o programa anterior um pouco mais elegante, calculando a média do aluno em uma única linha, sem usarmos a variável auxiliar mediaProvas. O novo algoritmo para resolver o problema proposto poderia ser:

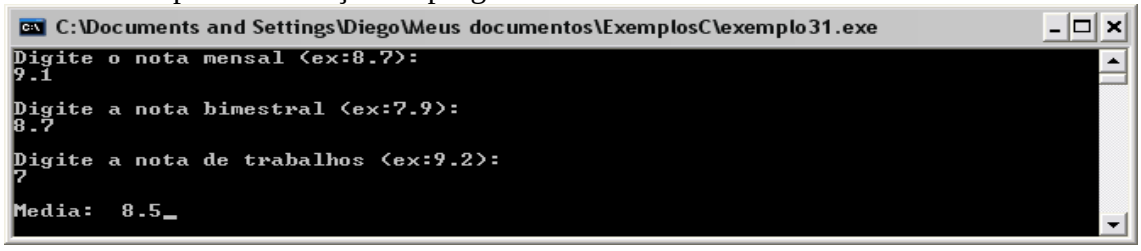
- Receba a nota mensal
- Receba a nota bimestral
- Receba a nota de trabalhos
- Calcule a média fórmula media=0.8*((mensal+bimestral)/2) + 0.2*trabalhos
- Exiba a média do aluno a tela

Implementando em Linguagem C:

Exemplo 31

```
/*
 Exemplo de cálculo de média – Precedência de Operadores
 Linguagem C - Uma introdução
 --
 Diego M. Rodrigues
*/
#include <stdio.h>

int main() {
      //Declarando as variáveis
      float mensal=0, bimestral=0, trabalhos=0;
      float media=0;

      //Recebendo a nota mensal
      printf("Digite o nota mensal (ex:8.7):\n");
      scanf("%f", &mensal);

      //Recebendo a nota bimestral
      printf("\nDigite a nota bimestral (ex:7.9):\n");
      scanf("%f", &bimestral);

      //Recebendo a nota bimestral
      printf("\nDigite a nota de trabalhos (ex:9.2):\n");
      scanf("%f", &trabalhos);

      //Calculando a média
      media=0.8*((mensal + bimestral)/2) + 0.2*trabalhos;

      //Mostrando a média na tela
      printf("\nMedia: %4.1f", media);

      //Esperando o <ENTER> para sair com getch() ao invés de getchar()
      getch();

      return 0;
}
```

Exemplo de execução do programa anterior:



Problema: Dado um número n maior do que 10, calcule o dobro do último algarismo desse número. Exemplos:

72 => 2*2 = 4
257 => 2*7 = 14
1095 => 2*5 = 10

Um algoritmo para resolver o problema proposto poderia ser:

- Receba o número n
- Calcule o resto da divisão de n por 10
- Multiplique o resto por 2
- Exiba o resultado na tela

Exemplo 32

```
/*
 Exemplo de Precedência de Operadores
 Linguagem C - Uma introdução
 --
 Diego M. Rodrigues
*/
#include <stdio.h>

int main() {
      //Declarando as variáveis
      int n=0;
      int resto=0;

      //Recebendo a nota mensal
      printf("Digite um número inteiro maior que 10:\n");
      scanf("%d", &n);

      //Calculando a média
      resto = n%10;
      resto = resto*2;

      //Mostrando a média na tela
      printf("\nResultado: %d", resto);

      //Esperando o <ENTER> para sair com getch() ao invés de getchar()
      getch();

      return 0;
}
```

Exemplo de execução do programa anterior:

Podemos agrupar as linhas...

```
resto = n%10;
resto = resto*2;
```

...de várias maneiras, por exemplo:

```
resto = (n%10)*2;
```

Na linha acima, como o operador resto *(%)* possui a mesma precedência do operador multiplicação *(*)*, é recomendado colocarmos os parênteses para garantir que o cálculo seja realmente feito na ordem que desejamos.

2. Operadores Relacionais

Os operadores relacionais são usados para determinar o relacionamento de uma quantidade com outra. Eles sempre retornam **0 (falso)** ou **1 (verdadeiro)**, dependendo do resultado do teste.

Alguns programadores preferem usar as palavras *"true"* que vem do inglês (verdadeiro) e *"false"* (do inglês falso) para representar os valores 0 ou 1 em resultados de testes lógicos.

Tabela de Operadores relacionais:

Operador relacional	Descrição
>	Maior que
>=	Maior ou igual que
<	Menor que
<=	Menor ou igual que
==	Igual
!=	Diferente (não igual)

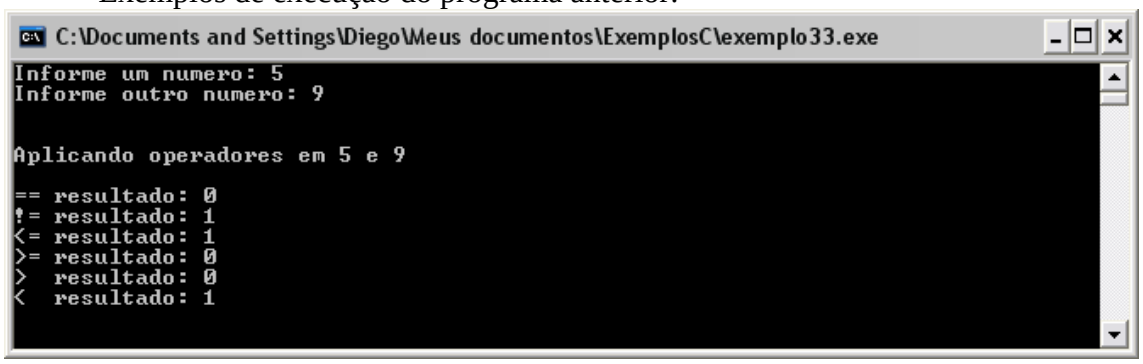
O programa seguinte ilustra o uso de cada operação e exibe o resultado de cada uma como 0 ou 1:

Exemplo 33

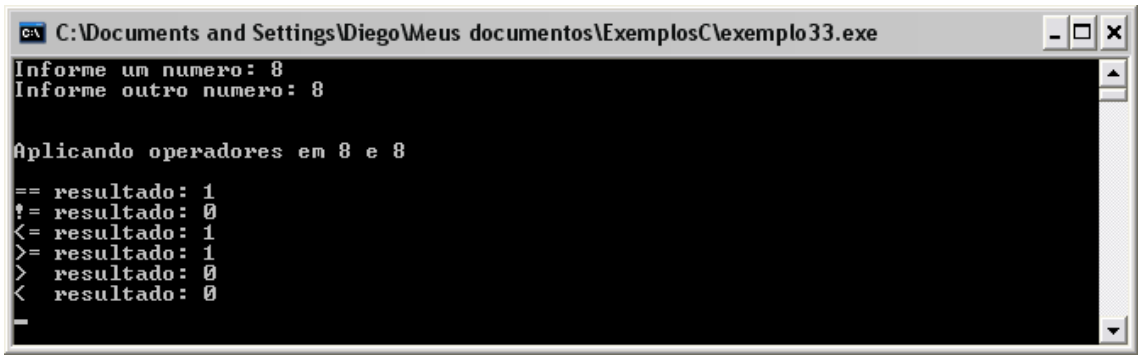
```
/*
 Exemplo de Operadores Relacionais
 Linguagem C - Uma introdução
 --
 Diego M. Rodrigues
*/
#include <stdio.h>

int main() {
    //Recebendo num1 e num2
    int num1=0, num2=0;
    printf("Informe um numero: ");
    scanf("%d", &num1);
    printf("Informe outro numero: ");
    scanf("%d", &num2);

    //Usando operadores e exibindo o resultado
    printf("\n\nAplicando operadores em %d e %d \n\n", num1, num2);
    printf("== resultado: %d \n", num1 == num2);
    printf("!= resultado: %d \n", num1 != num2);
    printf("<= resultado: %d \n", num1 <= num2);
```

```
    printf(">= resultado: %d \n", num1 >= num2);
    printf(">  resultado: %d \n", num1 > num2);
    printf("<  resultado: %d \n", num1 < num2);

    //Esperando o <ENTER> para sair com getch() ao invés de getchar()
    getch();
    return 0;
}
```

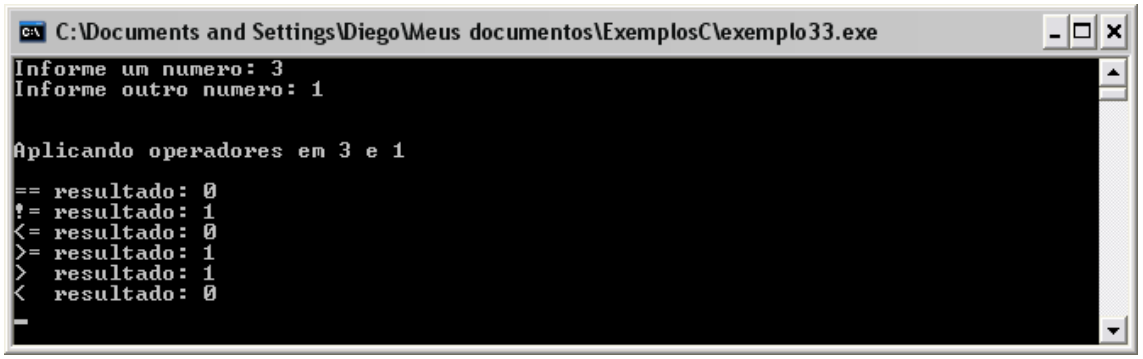
Exemplos de execução do programa anterior:



Tela A



Tela B



Tela C

Comentários sobre os resultados anteriores:

- Como já era esperado, todos os resultados das expressões com operadores relacionais foram 0 ou 1, que equivalem à falso ou verdadeiro (false ou true).

- Na **Tela A** entramos com os valores `5` e `9`, então:
  - No teste de igualdade com o operador == (`5==9`) o resultado foi `0` (`falso`).

- No teste de diferença com o operador `!=` (`5!=9`) o resultado foi `1` (`verdadeiro`).
- No teste "menor ou igual que" com o operador `<=` (`5<=9`) o resultado foi `1` (`verdadeiro, já que 5 é menor do que 9`).
- No teste "maior ou igual que" com o operador `>=` (`5>=9`) o resultado foi `0` (`falso, já que 5 não é 'maior ou igual que' 9`).
- No teste "maior que" com o operador `>` (`5>9`) o resultado foi `0` (`falso, já que 5 não é 'maior que' 9`).
- No teste "menor que" com o operador `<` (`5<9`) o resultado foi `1` (`verdadeiro, já que 5 é 'menor que' 9`).

- Na **Tela B** entramos com os valores `8` e `8`, então:
  - No teste de igualdade com o operador `==` (`8==8`) o resultado foi `1` (`verdadeiro`).
  - No teste de diferença com o operador `!=` (`8!=8`) o resultado foi `0` (`falso`).
  - No teste "menor ou igual que" com o operador `<=` (`8<=8`) o resultado foi `1` (`verdadeiro, já que 8 é igual que 8`).
  - No teste "maior ou igual que" com o operador `>=` (`8>=8`) o resultado foi `1` (`verdadeiro, já que 8 é igual que 8`).
  - No teste "maior que" com o operador `>` (`8>8`) o resultado foi `0` (`falso, já que 8 não é 'maior que' 8`).
  - No teste "menor que" com o operador `<` (`8<8`) o resultado foi `0` (`falso, já que 8 não é 'menor que' 8`).

- Na **Tela C** entramos com os valores `3` e `1`, então:
  - No teste de igualdade com o operador `==` (`3==1`) o resultado foi `0` (`falso`).
  - No teste de diferença com o operador `!=` (`3!=1`) o resultado foi `1` (`verdadeiro`).
  - No teste "menor ou igual que" com o operador `<=` (`3<=1`) o resultado foi `0` (`falso, já que 3 não é 'menor ou igual que' 8`).
  - No teste "maior ou igual que" com o operador `>=` (`3>=1`) o resultado foi `1` (`verdadeiro, já que 3 é maior que 1`).
  - No teste "maior que" com o operador `>` (`3>1`) o resultado foi `1` (`verdadeiro, já que 3 é maior que 8`).
  - No teste "menor que" com o operador `<` (`3<1`) o resultado foi `0` (`falso, já que 3 não é 'menor que' 1`).

ATENÇÃO
O operador = é usado para atribuição e o operador == é usado para comparação! Para testarmos se a variável *opcao* possui o valor *3*, devemos usar *opção==3* , nunca *opcao=3* !!

LINGUAGEM C – UMA INTRODUÇÃO

AULA 6 – Estruturas de decisão

As estruturas de decisão estão presentes em qualquer linguagem de programação. Com elas podemos realizar testes ao longo de nosso programa e decidir se o fluxo do mesmo deve ser desviado para outra direção ou não.

1 – O comando if()

If quer dizer "se" e é a estrutura de decisão mais usada em Linguagem C. Sua forma mais simples de uso é:

```
if ( "expressão de teste" ) {
      instruções;
      instruções;
}
```

Se a "expressão de teste" for verdadeira (algo diferente de 0), o comando *if* executa as instruções entre `{` e `}`. Caso contrário, o programa salta para a linha imediatamente após `}`. As "expressões de teste" normalmente são realizadas com operadores relacionais (`==`, `!=`, `>=`, `<=`, `>`, `<`).

Problema: Escreva um programa em Linguagem C que teste se um valor digitado pelo usuário é menor do que 10.

Um algoritmo para resolver o problema proposto poderia ser:

- Receba o número "n"
- Se n < 10
  - Escreva "Ok.. o número é menor do que 10"
- Se n >= 10
  - Escreva "O número não é menor do que 10"

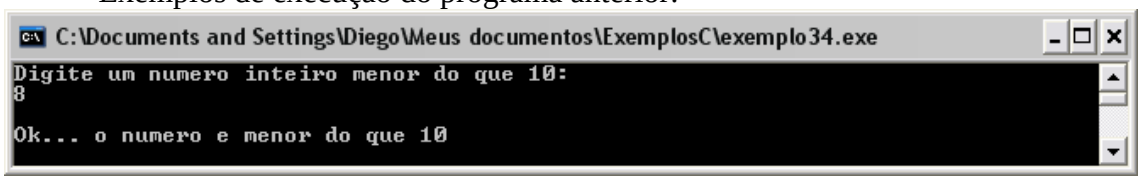
Implementando em Linguagem C:

Exemplo 34

```
/*
 Primeiro exemplo com IF
 Linguagem C - Uma introdução
 --
 Diego M. Rodrigues
*/
#include <stdio.h>

int main() {
      //Declarando as variáveis
      int n=0;

      //Recebendo a variável n
      printf("Digite um numero inteiro menor do que 10:\n");
```

```
      scanf("%d", &n);

      //Testando se n é menor do que 10
      if ( n < 10 ) {
            printf("\nOk... o numero e menor do que 10\n");
      }

      if ( n >= 10 ) {
            printf("\nO numero NAO e menor do que 10!!!\n");
      }

      //Esperando o <ENTER> para sair
      getch();

      return 0;
}
```

Exemplos de execução do programa anterior:



2. O comando else

Podemos pensar no comando *else* como um complemento (ou uma negação) do comando *if*. A forma um pouco mais completa da estrutura *if-else* tem a seguinte forma geral:

```
      if ( “expressão de teste” ) {
            instruções1;
            instruções2;
      } else {
            instruções3;
            instruções4;
      }
```

A “*expressão de teste*” será avaliada. Se ela for verdadeira (diferente de zero) o primeiro bloco de instruções será executado (instuções1; instruções2;). Caso a expressão de teste seja falsa (ou igual a zero) o segundo bloco de instruções será executado (instuções3; instruções4;). É importante ressaltar que quando usamos a estrutura *if-else*, estamos garantindo que um dos dois blocos de instruções será executado. Nunca serão executados os dois blocos ou nenhum deles.

Problema: Uma grande dificuldade em viagens internacionais é a conversão dos valores de temperatura de ºC para ºF. Faça um programa em Linguagem C que receba

uma temperatura em ºF e calcule seu equivalente em ºC. O programa deve exibir a mensagem “Levar roupas de frio.” quando a temperatura em ºC for menor ou igual à 20 e mostrar a mensagem “Não levar roupas de frio.” caso contrário.

Um algoritmo para resolver o problema proposto poderia ser:

- Receba a temperatura ºF em “f”
- Calcule a conversão para ºC com a fórmula c = ((f-32)*5)/9
- Se c <= 20
  - Escreva a temperatura em ºC na tela
  - Escreva “Levar roupas de frio.”

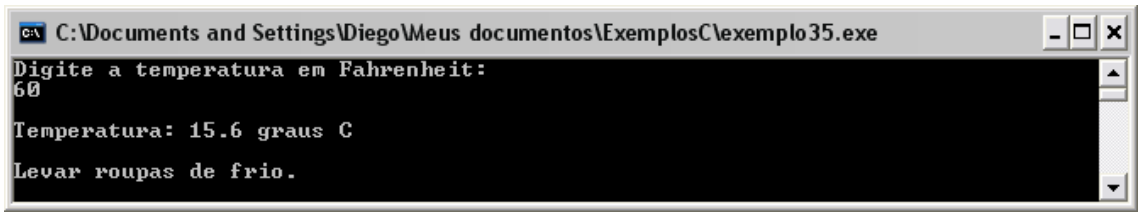
  Caso contrário
  - Escreva a temperatura em ºC na tela
  - Escreva “Não levar roupas de frio.”

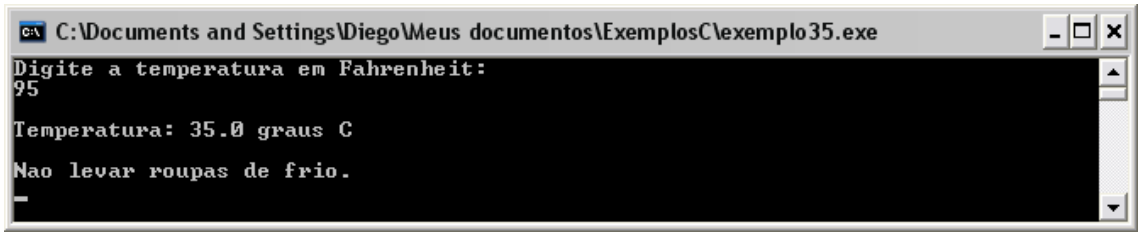
Implementando em Linguagem C:

Exemplo 35

```
/*
 Exemplo com IF-ELSE – Conversão Fahrenheit-Centígrados
 Linguagem C - Uma introdução
 --
 Diego M. Rodrigues
*/
#include <stdio.h>

int main() {
      //Declarando as variáveis
      float f=0, c=0;

      //Recebendo a temperatura f
      printf("Digite a temperatura em Fahrenheit:\n");
      scanf("%f", &f);

      //Calculando a conversão
      c=( (f-32)*5 )/9;

      //Testando se n é menor do que 10
      if ( c <= 20 ) {
            printf("\nTemperatura: %4.1f graus C\n", c);
            printf("\nLevar roupas de frio.\n");
      }
      else {
            printf("\nTemperatura: %4.1f graus C\n", c);
            printf("\nNão levar roupas de frio.\n");
      }

      //Esperando o <ENTER> para sair
      getch();

      return 0;
}
```

Exemplos de execução do programa anterior:





Problema: Escreva um programa em Linguagem C que calcule a média de um aluno, com base em duas provas (mensal e bimestral) e mostre a mensagem "Aluno Aprovado" se a média for maior ou igual do que 5 e "Aluno Reprovado" caso contrário.

Um algoritmo para resolver o problema proposto poderia ser:

- Receba a nota da prova mensal
- Receba a nota da prova bimestral
- Calcule a média com a fórmula media=(mensal+bimestral)/2
- Se media >= 5
  - Escreva "Aluno aprovado!"

  Caso contrário
  - Escreva "Aluno Reprovado!"
- Exiba a média na tela

O programa que realiza a tarefa acima poderia ser escrito em uma pseudo linguagem da seguinte forma:

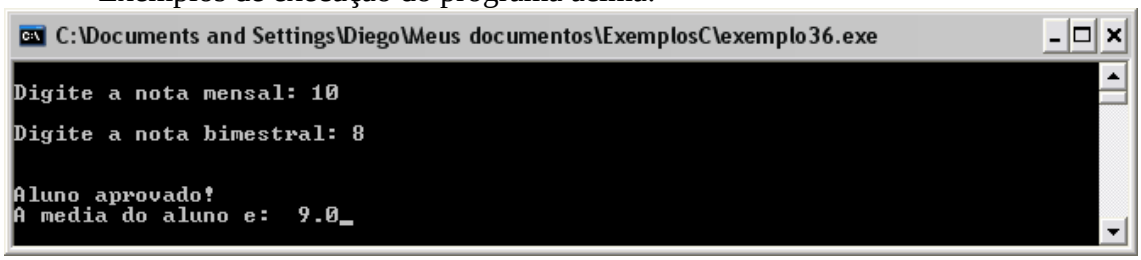
Implementando em Linguagem C:

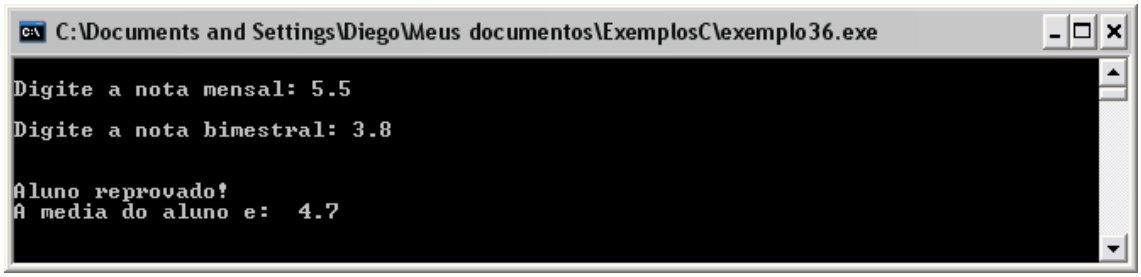
Exemplo 36

```
/*
 Exemplo com IF-ELSE – Cálculo de média
 Linguagem C - Uma introdução
 --
 Diego M. Rodrigues
*/
#include <stdio.h>

int main() {

    //Declarando as variáveis
    float mensal=0;
    float bimestral=0;
    float media=0;

    //Recebendo a nota mensal
    printf("\nDigite a nota mensal: ");
    scanf("%f",&mensal);
```

```
    //Recebendo a nota bimestral
    printf("\nDigite a nota bimestral: ");
    scanf("%f",&bimestral);

    //Calculando a média
    media = (mensal+bimestral)/2;

    //Mostando mensagens
    if ( media >= 5 ) {
        printf("\n\nAluno aprovado!");
    }
    else {
        printf("\n\nAluno reprovado!");
    }

    //Mostrando a média
    printf("\nA media do aluno e: %4.1f",media);

    //Esperando o <ENTER> para sair
    getch();
    return 0;
}
```

Exemplos de execução do programa acima:





3 – O else if

Em todos os testes realizados nos exemplos anteriores nossos programas escolhiam uma dentre duas opções (se... caso contrário...). Em muitos casos precisamos realizar testes mais elaborados, com mais de duas opções (se... se não, se... caso contrário...), para isso podemos usar o *else if*.

Dessa forma, chagamos à forma completa, *if - else if - else*:

```
      if ( “expressão de teste 1” ) {
            instruções1;
            instruções2;
      }
      else if ( “expressão de teste 2” ) {
            instruções3;
            instruções4;
      }
      else {
            Instruções5;
            Instruções6;
      }
```

A “*expressão de teste 1*” será a primeira avaliada. Se ela for verdadeira (diferente de zero) o primeiro bloco de instruções será executado (instuções1; instruções2;). Caso a “*expressão de teste 1*” seja falsa (ou igual a zero), a “*expressão de teste 2*” será avaliada. Caso ela seja verdadeira, o segundo bloco de instruções será executado (instuções3; instruções4;). Caso tanto a “*expressão de teste 1*” quanto a “*expressão de teste 2*” sejam falsas, o terceiro bloco de instruções será executado (instuções5; instruções6;).

Problema: Escreva um programa em Linguagem C que receba um número n e imprima na tela:

- “O número é positivo”, se o usuário digitar um número positivo
- “O número é negativo”, se o usuário digitar um número negativo
- “Você digitou zero”, se o usuário digitar o número zero

Um algoritmo para resolver o problema proposto poderia ser:

- Receba o número n
- Se n > 0
  - Escreva “Número positivo!”

  Se não, se n < 0
  - Escreva “Número negativo!”

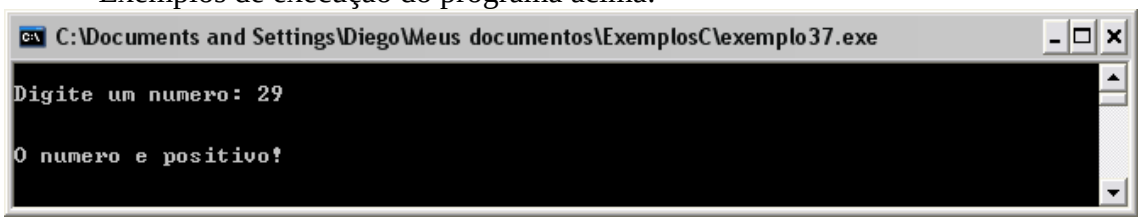
  Caso contrário
  - Escreva “Você digitou zero!”

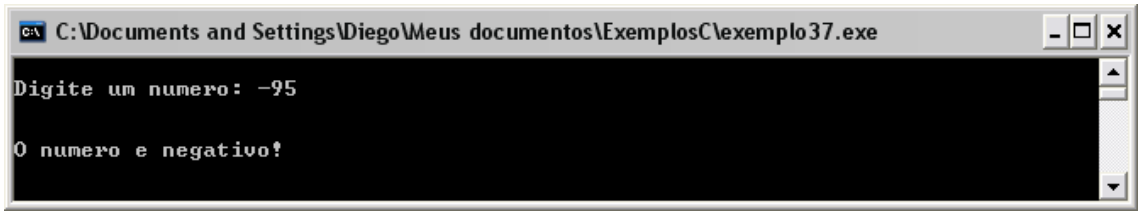
Exemplo 37

```
/*
 Exemplo com IF - ELSE IF - ELSE – Numeros positivos ou negativos
 Linguagem C - Uma introdução
 --
 Diego M. Rodrigues
*/
#include <stdio.h>

int main() {

    //Declarando as variáveis
    int n=0;

    //Recebendo o número
    printf("\nDigite um numero: ");
    scanf("%d",&n);


    //Mostando mensagens
```

```
    if ( n > 0 ) {
        printf("\n\nO numero e positivo!");
    }
    else if ( n < 0 ) {
        printf("\n\nO numero e negativo!");
    }
    else {
        printf("\n\nVoce digitou zero!");
    }

    //Esperando o <ENTER> para sair
    getch();
    return 0;
}
```

Exemplos de execução do programa acima:





Problema: Desenvolva uma calculadora simples em Liguagem C. Essa calculadora deve ser capaz de realizar as 4 operações básicas (`+,-,*,/`) sobre dois números quaisquer.

Observações:

- Para ler a operação que será realizada (`+,-,*,/`), use o *`%c`* dentro do *`scanf()`*.
- Como a operação será lida com *`%c`* (como caractere), você deve usar aspas simples na "*`expressão de teste`*", coisas do tipo: *`if ( operador == '+')`*
- Você pode ler mais de uma variável dentro do mesmo *`scanf()`*, desde que elas fiquem separadas por espaços. Ex:
  `scanf("%d %d", &num1, &num2)`

Um algoritmo para resolver o problema proposto poderia ser:

- Escreva uma mensagem de boas vindas
- Receba o operando 1
- Receba o operador

- Receba o operando 2
- Se `operador == ‘+’`
  - Calcular resultado = operando1 + operando2

  Se não, se `operador == ‘-’`
  - Calcular resultado = operando1 - operando2

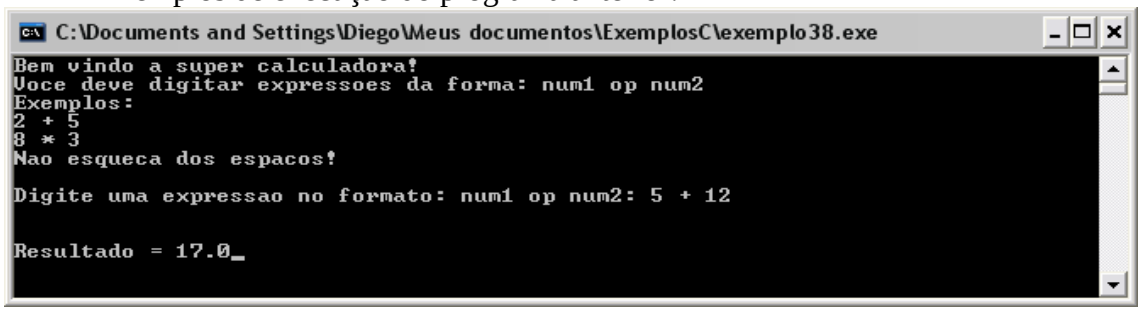
  Se não, se `operador == ‘*’`
  - Calcular resultado = operando1 * operando2

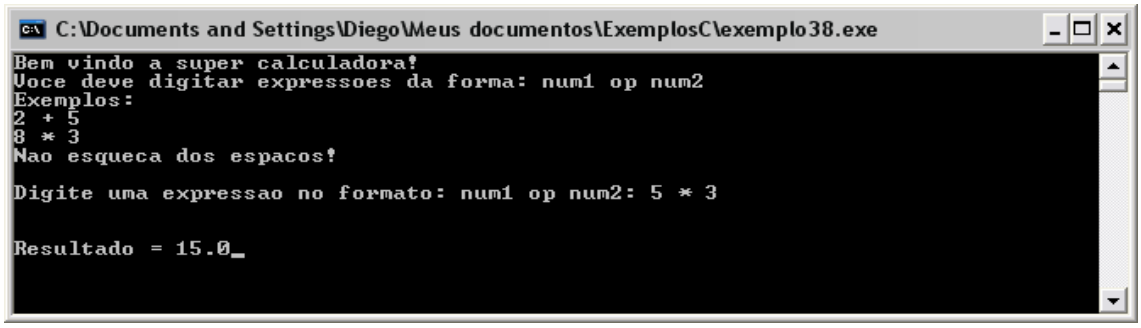
  Se não, se `operador == ‘/’`
  - Calcular resultado = operando1 / operando2
- Escreva o resultado na tela

Exemplo 38

```
/*
 Exemplo com IF - ELSE IF - ELSE – Numeros positivos ou negativos
 Linguagem C - Uma introdução
 --
 Diego M. Rodrigues
*/
#include <stdio.h>

int main() {
    //Declarando as variáveis
    float num1=0, num2=0, resultado=0;
    char operador;

    //Mensagem de boas vindas
    printf(“Bem vindo a super calculadora!”);
    printf(“\nVoce deve digitar expressoes da forma: num1 op num2”);
    printf(“\nExemplos:”);
    printf(“\n2 + 5”);
    printf(“\n8 * 3”);
    printf(“\nNao esqueca dos espacos!”);

    //Recebendo o número
    printf(“\n\nDigite uma expressao no formato: num1 op num2: ”);
    scanf(“%f %c %f”, &num1, &operador, &num2);

    //Mostando mensagens
    if ( operador == ‘+’ ) {
        resultado = num1 + num2;
    }
    else if ( operador == ‘-’ ) {
        resultado = num1 - num2;
    }
    else if ( operador == ‘*’ ) {
        resultado = num1 * num2;
    }
    else if ( operador == ‘/’ ) {
        resultado = num1 / num2;
    }

    //Mostrando o resultado
    printf(“\n\nO resultado = %4.1f”,resultado);

    //Esperando o <ENTER> para sair
    getch();
    return 0;
}
```

Exemplos de execução do programa anterior:



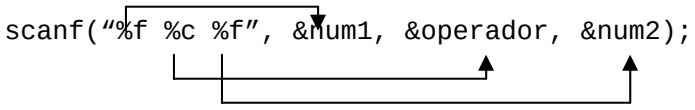


<u>Observações</u> finais sobre esse exemplo:

A expressão matemática foi lida na linha:

```
scanf("%f %c %f", &num1, &operador, &num2);
```

Como já foi dito, podemos ler mais de uma variável do teclado dentro de um mesmo *scanf()*, desde que tudo fique separado por espaços. No *scanf()* acima o primeiro *%f* indica que a primeira variável (*num1*) irá receber um valor do tipo *float*. O *%c* indica que a segunda variável (*operador*) irá receber um *char* e finalmente o último *%f* indica que a última variável (*num3*) irá receber um valor *float*.



Vamos dar outro exemplo de *scanf()* composto, mas dessa vez com quatro variáveis: idade(*int*), peso(*float*), sexo(*char*) e saldo(*double*):

```
scanf("%d %f %c %g", &idade, &peso, &sexo, &saldo);
```

Outra coisa que deve ser ressaltada é que sempre que usamos **caracteres isolados no programa, estes devem estar em ASPAS SIMPLES**. Observe como exemplo o pequeno trecho de código abaixo que recebe um *char* dentro de uma variável *sexo* para testar se o valor digitado foi *H* ou *M*.

```
char sexo;
printf("\nQual o sexo (H/M)? ");
scanf("%c", &sexo);
if ( sexo == 'H') {
      printf("Homem!");
} else if ( sexo == 'M' ) {
      printf("Mulher!");
} else {
      printf("Opção inválida!");
}
```