MINISTÉRIO DA EDUCAÇÃO

SECRETARIA DE EDUCAÇÃO PROFISSIONAL E TECNOLÓGICA

INSTITUTO FEDERAL DE EDUCAÇÃO, CIÊNCIA E TECNOLOGIA CATARINENSE

***CAMPUS* AVANÇADO DE LUZERNA**

Engenharia de Controle e Automação

Informática para Automação

Aluno: _____

Prof: Ricardo Kerschbaumer

Luzerna, Fevereiro de 2012

INSTITUTO FEDERAL DE
EDUCAÇÃO, CIÊNCIA E TECNOLOGIA
CATARINENSE

Informática para Automação

Este material foi desenvolvido pelo professor Ricardo Kerschbaumer para ser utilizado na componente curricular de informática para automação do curso de engenharia de controle e Automação. O objetivo deste material é servir como material de apoio as aulas teóricas e práticas, bem como de material de estudo para que os alunos possam tirar suas dúvidas nos períodos extra classe.

Os assuntos são distribuídos em aulas, cada aula diz respeito a um determinado assunto e não necessariamente a um encontro.

No decorrer do material serão apresentados vários exercícios e vários exemplos. Os exemplos estarão dispostos em quadros com o fundo escurecido, como no exemplo a seguir.

```
#include<stdio.h>

int main()
{
printf(“Olá Mundo!\n”);
return(0);
}
```

Bons estudos.

Sumário

AULA 1 - NOÇÕES DE HARDWARE DE COMPUTADOR

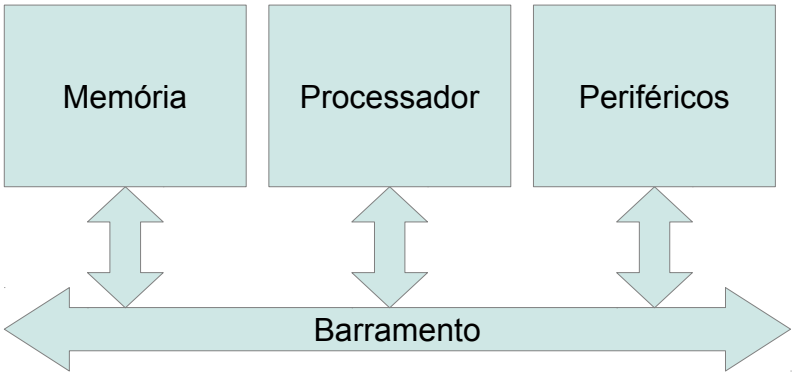
Noções básicas sobre sistemas de computação.

1.1 Objetivo:

O objetivo desta aula é mostrar aos alunos como funcionam internamente os computadores. O conhecimento do funcionamento dos computadores irá ajudar os alunos no desenvolvimento de programas, que é o objetico desta componente curricular.

1.2 Componentes básicos de um computador

Apesar da existência de uma grande diversidade de computadores em termos de arquiteturas, pode-se enumerar, num ponto de vista mais genérico os componentes básicos desta classe de equipamentos. A Figura a seguir apresenta o esquema de um computador, destacando os elementos que o compõem. Apesar da grande evolução ocorrida na área de informática desde o aparecimento dos primeiros computadores, o esquema apresentado na figura pode ser utilizado tanto para descrever um sistema computacional atual como os computadores da década de 40.



Elementos básicos de um computador

Os principais elementos do computador são:

- O processador (ou microprocessador) é responsável pelo tratamento das informações armazenadas em memória (programas em código de máquina e dos dados).
- A memória é responsável pela armazenagem dos programas e dos dados.
- Periféricos, que são os dispositivos responsáveis pelas entradas e saídas de dados do computador, ou seja, pelas interações entre o computador e o mundo externo. Exemplos de

periféricos são o monitor, teclados, mouses, impressoras, etc.

- Barramento, que liga todos estes componentes e é uma via de comunicação de alto desempenho por onde circulam os dados tratados pelo computador.

1.3 Processador

Um microprocessador, ou simplesmente processador, é um circuito integrado (ou chip), que é considerado o "cérebro" do computador (Figura a seguir). É ele que executa os programas, faz os cálculos e toma as decisões, de acordo com as instruções armazenadas na memória. Processador de computador

Os microprocessadores formam uma parte importantíssima do computador, chamada de UCP (Unidade Central de Processamento), ou em inglês, CPU (Central Processing Unit). Ligando-se um microprocessador a alguns chips de memória e alguns outros chips auxiliares, tornou-se possível construir um computador inteiro em uma única placa de circuito. Esta placa, como visto na figura a seguir, é comumente chamada de placa mãe.

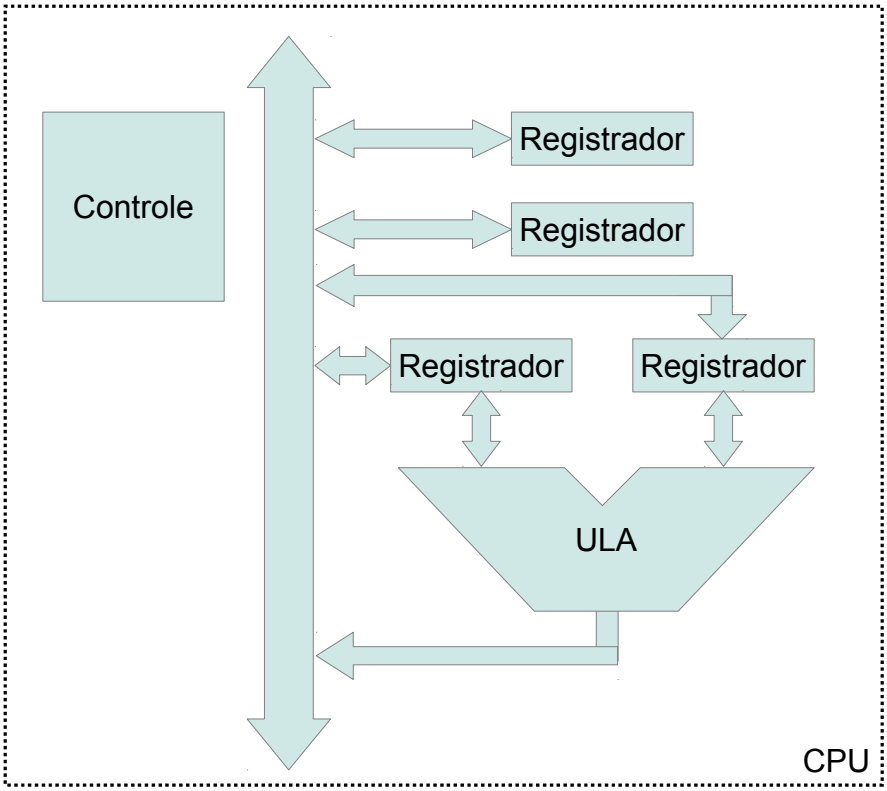
Não importa de que tipo de CPU estamos falando, seja um microprocessador, ou uma das várias placas que formam a CPU de um computador de grande porte, podemos dizer que a CPU realiza as seguintes tarefas:

a) são transferidos para a memória. Uma vez estando na memória, a CPU pode executar os programas e processar os dados.

b) Comanda todos os outros chips do computador.

Placa mãe de computador

A CPU é composta basicamente de três elementos: unidade de controle, unidade lógica e aritmética e registradores, conforme a figura a seguir.



As sessões que seguem apresentam cada um destes componentes.

1.3.1 Unidade Lógica e Aritmética

O primeiro componente essencial num computador (ou sistema computacional) é a Unidade Lógica e Aritmética (ALU), a qual, como o próprio nome indica, assume todas as tarefas relacionadas às operações lógicas (ou, e, negação, etc.) e aritméticas (adições, subtrações, etc...) a serem realizadas no contexto de uma tarefa.

Um parâmetro importante é o tamanho da palavra processada pela unidade lógica e aritmética. Como o sistema de numeração adotado nas arquiteturas de computadores é o binário, o tamanho de palavra é dado em números de bits. Quanto maior o tamanho da palavra manipulada pelo microprocessador, maior é o seu potencial de cálculo e maior a precisão das operações realizadas. Os computadores pessoais atualmente operam com palavras de 32 ou 64 bits.

A velocidade de cálculo obviamente é outro fator de peso para o desempenho do computador, uma vez que ela será determinante para o tempo de resposta de um sistema computacional com respeito à execução de uma dada aplicação. A velocidade de cálculo está diretamente relacionada com a frequência do relógio que pilota o circuito da CPU como um todo.

Outro parâmetro importante associado ao desempenho do computador é a quantidade de operações que ela suporta. Por exemplo, os primeiros processadores suportavam um conjunto relativamente modesto de operações lógicas e aritméticas. Em particular, no que diz respeito às operações aritméticas, os primeiros processadores suportavam apenas operações de adição e subtração, sendo que as demais operações tinham de ser implementadas através de sequências destas operações básicas. Os processadores suportando um conjunto mais complexo de instruções surgiu nos últimos anos, graças à adoção da tecnologia CISC (Complex Instruction Set Computer).

1.3.2 Unidade de Controle

A Unidade de Controle tem a maior importância na operação de um computador, uma vez que é esta unidade que assume toda a tarefa de controle das ações a serem realizadas pelo computador, comandando todos os demais componentes de sua arquitetura. É este elemento quem deve garantir a correta execução dos programas e a utilização dos dados corretos nas operações que as manipulam. É a unidade de controle que gerencia todos os eventos associados à operação do computador, particularmente as chamadas interrupções, tão utilizadas nos sistemas há muito tempo.

1.3.3 Registradores

A CPU contém internamente uma memória de alta velocidade que permite o armazenamento de valores intermediários ou informações de comando. Esta memória é composta de registradores

(ou registros), na qual cada registro tem uma função própria. Os registros, geralmente numerosos, são utilizados para assegurar o armazenamento temporário de informações importantes para o processamento de uma dada instrução. Conceitualmente, registro e memória são semelhantes: a localização, a capacidade de armazenamento e os tempos de acesso às informações que os diferenciam. Os registros se localizam no interior de um microprocessador, enquanto a memória é externa a este. Um registro memoriza um número limitado de bits, geralmente uma palavra de memória. Os registros mais importantes são:

- Contador de programa (PC - Program Counter), que aponta para a próxima instrução a executar.
- Registro de instrução (IR - Instruction Register) que armazena a instrução em execução.
- Outros registros que permitem o armazenamento de resultados intermediários.

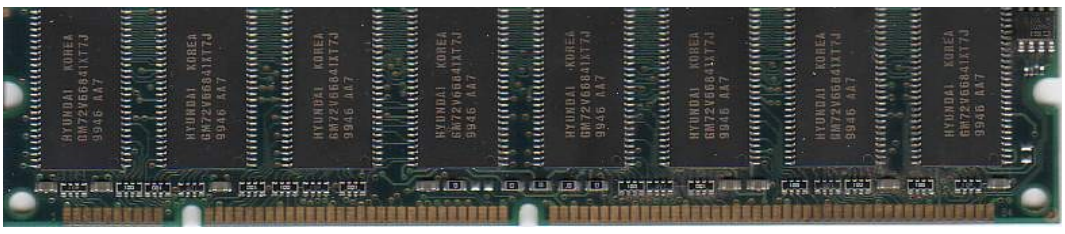
1.3.4 Clock

Clock é um circuito oscilador que tem a função de sincronizar e ditar a medida de velocidade de transferência de dados no computador, por exemplo, entre o processador e a memória principal. Esta frequência é medida em ciclos por segundo, ou Hertz. Existe a frequência própria do processador, comandando operações internas a ele, e a frequência do computador a ele associado, basicamente ciclos CPU - Memória principal.

1.4 Memória

Todo computador é dotado de uma quantidade de memória (que pode variar de máquina para máquina) a qual se constitui de um conjunto de circuitos capazes de armazenar os dados e os programas a serem executados pela máquina. Nós podemos identificar diferentes categorias de memória:

- A memória principal, ou memória de trabalho, onde normalmente devem estar armazenados os programas e dados a serem manipulados pelo processador, veja a figura a seguir.



Memória RAM

- A memória secundária que permitem armazenar uma maior quantidade de dados e instruções por um período de tempo mais longo; o disco rígido (figura a seguir) é o exemplo mais evidente de memória secundária de um computador, mas podem ser citados outros dispositivos como pendrives.

Disco rígido de computador

1.5 Dispositivos de entrada e saída

Todos os computadores necessitam interagir com o mundo exterior. A entrada e a saída de dados ou informações é critica para seu funcionamento. Existem muitos dispositivos que podem ser conectados aos computadores. Os mais comuns são teclado, mouse e monitor, porem existem outros não tão comuns, tais como equipamentos científicos, esquipamentos de aquisição e processamento de dados, máquinas industriais e assim por diante. Não é o objetivo desta componente curricular estudar a fundo os dispositivos de entrada e saída dos computadores.

1.6 conclusão

Esta aula apresentou de forma bastante superficial como é formado o núcleo de um computador. Porem os computadores são maquinas complexas, em constante atualização e envolvem muitos outros elementos de software e hardware para funcionarem. O conteúdo apresentado é suficiente para que os alunos desta componente curricular possam compreender o funcionamento do computador e possam desenvolver programas para o mesmo.

AULA 2 - LÓGICA DE PROGRAMAÇÃO

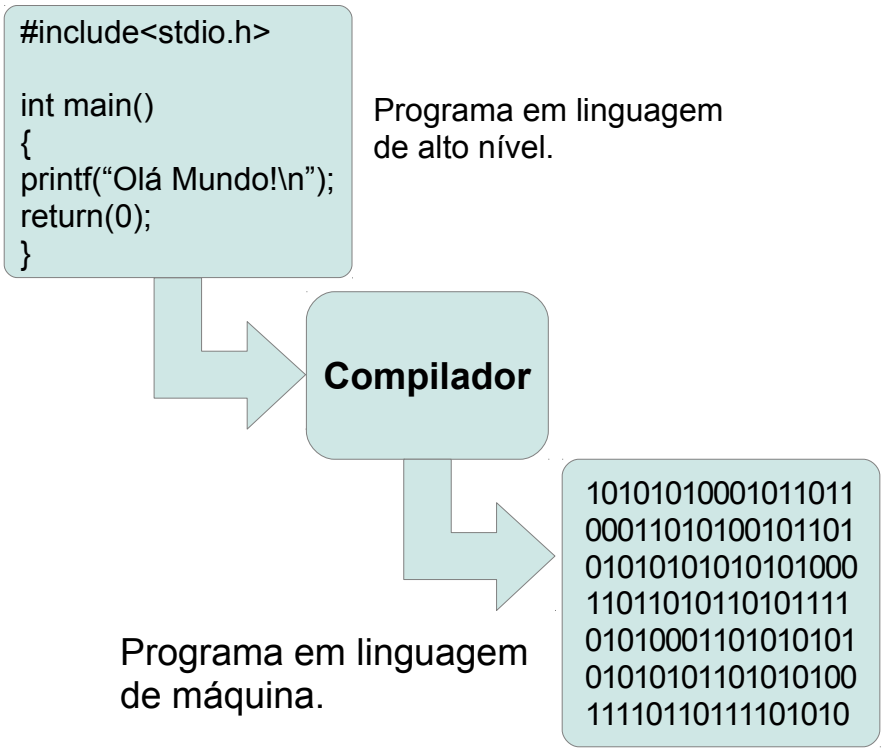
Noções sobre a lógica de programação, algoritmos e fluxogramas

2.1 Objetivo:

O objetivo desta aula é mostrar aos alunos como funciona um programa de computador e como é possível resolver problemas de várias áreas utilizando um computador. Inicialmente veremos o que é um programa de computador, na sequencia veremos como organizar as ideia de forma a resolver um determinado problema que são os chamados algoritmos. E finalmente abordaremos a representação gráfica de um algoritmo através de fluxogramas.

2.2 Programas de computador

Para desenvolver um programa de computador é necessário elaborar uma sequencia lógica de instruções de modo que estas instruções resolvam o problema em questão. O conjunto de instruções que podem ser utilizadas para resolver os problemas é definido pela linguagem de programação utilizada. Para facilitar a tarefa dos programadores as linguagens de programação atribuem nomes compreensíveis a suas instruções, e estas instruções são então convertidas para a linguagem do computador. A figura a seguir mostra este processo.



2.3 Lógica de programação

Lógica de programação é a técnica de encadear pensamentos para atingir determinado objetivo. Estes pensamentos, podem ser descritos como uma sequência de instruções, que devem ser seguidas para se cumprir uma determinada tarefa. Assim são executados passos lógicos até que se possa atingir um objetivo ou que se encontre a solução de um problema.

Nas linguagens de programação as instruções são um conjunto de regras ou normas definidas para a realização das tarefas. Na informática, porém, instrução é a informação que indica a um computador uma ação elementar a ser executada pelo processador.

As instruções por isoladamente não resolvem os problemas. É necessário ordenar as instruções de forma correta para que o programa funcione.

Como exemplo podemos imaginar a sequencia de atividades necessárias para que se construa uma casa. É claro que não podemos construir o telhado antes de construir as fundações. Na programação de computadores as coisas funcionam da mesma forma.

Um programa de computador é uma coleção de instruções que descrevem uma tarefa a ser realizada por um computador. O termo pode ser uma referência ao código fonte, escrito em alguma linguagem de programação, ou ao arquivo que contém a forma executável deste código fonte. São exemplos de programas de computador os editores de texto, jogos, navegadores de internet etc.

2.4 Linguagem de programação

Para construir os programas de computador são utilizadas linguagens de programação. Uma linguagem de programação é um método padronizado para expressar instruções para um computador. É um conjunto de regras sintáticas e semânticas usadas para definir um programa de computador. Uma linguagem permite que um programador especifique precisamente sobre quais dados um computador vai atuar, como estes dados serão armazenados ou transmitidos e quais ações devem ser tomadas sob várias circunstâncias.

O conjunto de palavras, compostos de acordo com essas regras, constituem o código fonte de um programa. Esse código fonte é depois traduzido para código de máquina, que é executado pelo processador. O responsável por traduzir o documento de texto que contem o código fonte em um programa é o compilador.

Uma das principais metas das linguagens de programação é permitir que programadores

tenham uma maior produtividade, permitindo expressar suas intenções mais facilmente do que quando comparado com a linguagem que um computador entende nativamente (código de máquina). Assim, linguagens de programação são projetadas para adotar uma sintaxe de nível mais alto, que pode ser mais facilmente entendida por programadores humanos. Linguagens de programação são ferramentas importantes para que programadores e engenheiros de software possam escrever programas mais organizados e com maior rapidez.

2.5 Algoritmos

Algoritmos são trechos de programas que tem a finalidade de resolver um problema ou executar uma função. O conceito de algoritmo é frequentemente ilustrado pelo exemplo de uma receita, embora muitos algoritmos sejam mais complexos. Eles podem repetir passos (fazer iterações) ou necessitar de decisões (tais como comparações ou lógica) até que a tarefa seja completada. Formalmente os algoritmos são uma sequência finita de passos que levam a execução de uma tarefa.

Qualquer tarefa, que pode ser descrita através de uma sequencia de passos, também pode ser representada através de um algoritmo. A seguir temos um exemplo de um algoritmo para fazer um bolo.

1. Escolher a receita do bolo.
2. Separar os ingredientes.
3. Misturar os engredientes conforme a receita.
4. Despejar a mistura em uma forma.
5. Ajustar a temperatura do forno.
6. Assar o bolo.
7. Retirar o bolo da forma.

Na programação de computadores é comum escrevermos algoritmos em pseudocódigo. Este nome é uma alusão à posterior implementação em uma linguagem de programação, ou seja, quando formos programar em uma linguagem, por exemplo a linguagem C, estaremos gerando código em C. Por isso os algoritmos são independentes das linguagens de programação.

Ao contrário de uma linguagem de programação, não existe um formalismo rígido de como

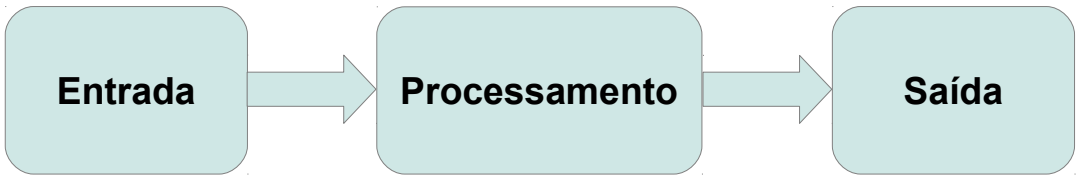
deve ser escrito o algoritmo. O algoritmo deve ser fácil de se interpretar e fácil de codificar. Ou seja, ele deve ser o intermediário entre a linguagem falada e a linguagem de programação.

2.5.1 Regras para construção do Algoritmo

Para escrever um algoritmo precisamos descrever a sequência de instruções, de maneira simples e objetiva. Para isso utilizaremos algumas técnicas:

- Usar somente um verbo por frase
- Imaginar que você está desenvolvendo um algoritmo para pessoas que não trabalham com informática
- Usar frases curtas e simples
- Ser objetivo
- Procurar usar palavras que não tenham sentido dúbio

Para montarmos um algoritmo, precisamos primeiro dividir o problema apresentado em três fases fundamentais, conforme figura a seguir.



Onde temos:

ENTRADA: São os dados de entrada do algoritmo

PROCESSAMENTO: São os procedimentos utilizados para chegar ao resultado final

SAÍDA: São os dados já processados

2.5.2 Exemplo de Algoritmo

Imagine o seguinte problema: Calcular a média final dos alunos. Os alunos realizarão quatro provas: P1, P2, P3 e P4.

Onde:

$$Média\ Final = \frac{P1 + P2 + P3 + P4}{4}$$

Para montar o algoritmo proposto, faremos três perguntas:

a) Quais são os dados de entrada? R:

Os dados de entrada são P1, P2, P3 e P4

b) Qual será o processamento a ser utilizado?

R: O procedimento será somar todos os dados de entrada e dividi-los por 4 (quatro)

$$\frac{P1 + P2 + P3 + P4}{4}$$

c) Quais serão os dados de saída?

R: O dado de saída será a média final

O algoritmo para realizar esta tarefa é apresentado a seguir.

```
Receba a nota da prova1
Receba a nota de prova2
Receba a nota de prova3
Receba a nota da prova4
Some todas as notas
Divida o resultado por 4
Mostre o resultado da divisão
```

Após elaborarmos um algoritmo é necessário testarmos seu funcionamento. Para isso podemos realizar simulações, utilizando amostras de dados e executando as tarefas manualmente.

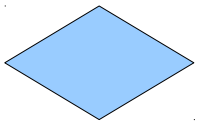
2.6 Fluxograma

Fluxograma é um tipo de diagrama, e pode ser entendido como uma representação esquemática de um processo, muitas vezes feito através de gráficos que ilustram de forma descomplicada a transição de informações entre os elementos que o compõem. Podemos entendê-lo, na prática, como a documentação dos passos necessários para a execução de um processo qualquer.

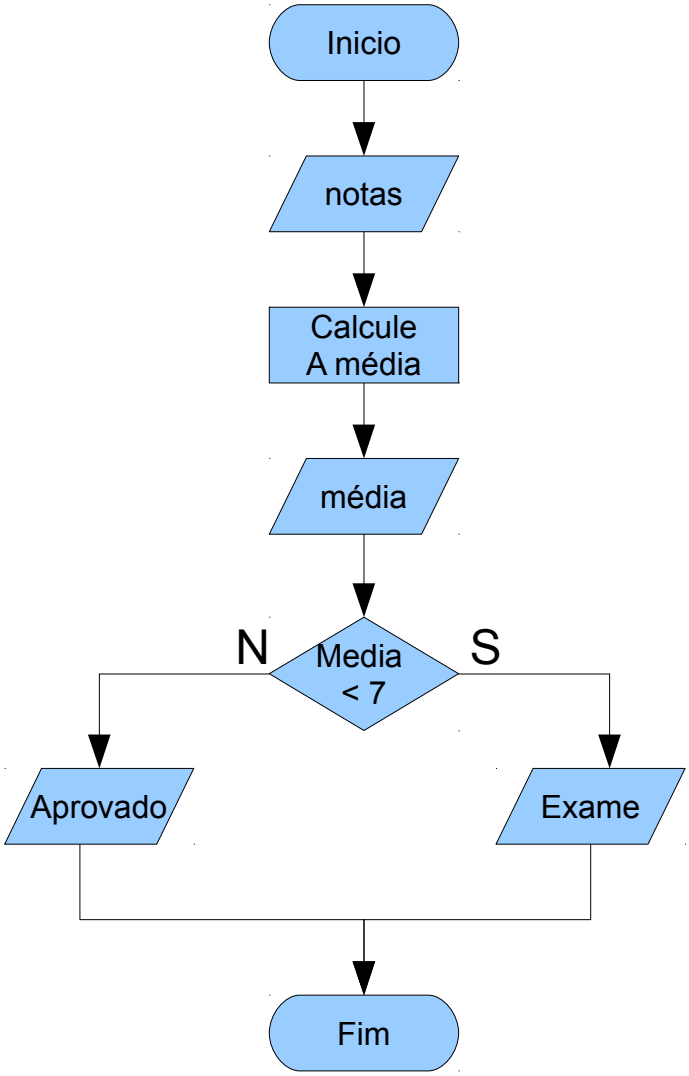
Na programação os fluxogramas são utilizados para representar graficamente o comportamento dos algoritmos, facilitando assim a compreensão da ordem de execução de cada um dos comandos.

O diagrama de blocos é uma forma padronizada e eficaz para representar os passos lógicos de um determinado processamento.

A tabela a seguir apresenta alguns dos principais símbolos usados nos fluxogramas

	Inicio ou fim
	Entrada ou saída
	Processamento
	Tomada de decisão

Para compreender melhor como funcionam os algoritmos observe o exemplo a seguir.



Neste exemplo temos um algoritmo para calcular a média e informar o operador se o aluno

esta ou não em exame.

Observando o algoritmo notamos que as setas indicam o sentido dos acontecimentos. Inicialmente é realizada a entrada das notas, na sequencia é calculada a média, após o cálculo da média, seu valor é apresentado ao operador, na sequencia é realizado um teste para verificar se a nota é menor que 7, se for é apresentada a mensagem “Exame”, se não é apresentada a mensagem “Aprovado”, e o fluxograma chega ao fim.

2.7 Conclusão

A elaboração de programas de computador é a arte de encontrar uma sequencia de comandos, capaz de executar a tarefa desejada. Esta sequencia de comandos pode ser representada através de algoritmos e fluxogramas, o que facilita seu desenvolvimento e sua compreensão.

As próximas aulas utilização os algoritmos e os fluxogramas para explicar os fundamentos da linguagem C.

2.8 Exercícios

1. Qual é a função do compilador?
2. Porque é necessário utilizarmos uma linguagem de programação? Não seria possível programar diretamente em linguagem de máquina?
3. Faça um algoritmo para calcular a nota necessária no exame, sendo que a entrada é a média.

   A formula para calcular a nota necessária no exame é: $Nota\ Exame = \frac{50 - (6 * Média)}{4}$
4. Faça um fluxograma para o algoritmo do exercício anterior.
5. Faça um algoritmo que recebe três números, determina qual é o maior dentre eles e apresenta este número.
6. Faça um fluxograma para o algoritmo do exercício anterior.

AULA 3 - INTRODUÇÃO A LINGUAGEM C

Fundamentação da linguagem C, sua estrutura e codificação

3.1 Objetivo:

O objetivo desta aula é mostrar aos alunos os fundamentos da linguagem C, suas origens e suas principais características. Para que se possa desenvolver programas em linguagem C é necessário conhecer um conjunto de regras padronizadas que definem a estrutura desta linguagem.

Os próximos parágrafos fazem uma introdução histórica e apresentam a estrutura da linguagem C.

3.2 A linguagem C

C é uma linguagem de programação compilada de propósito geral, de alto nível, padronizada pela ISO, criada em 1972, por Dennis Ritchie, no AT&T Bell Labs, para desenvolver o sistema operacional Unix (que foi originalmente escrito em Assembly). C é uma das linguagens de programação mais populares e existem poucas arquiteturas para as quais não existem compiladores para C. C tem influenciado muitas outras linguagens de programação, mais notavelmente C++, que originalmente começou como uma extensão para C.

A linguagem C pertence a uma família de linguagens cujas características são: portabilidade, modularidade, compilação separada, recursos de baixo nível, geração de código eficiente, confiabilidade, regularidade, simplicidade e facilidade de uso. A seguir temos um exemplo de programa escrito em linguagem C.

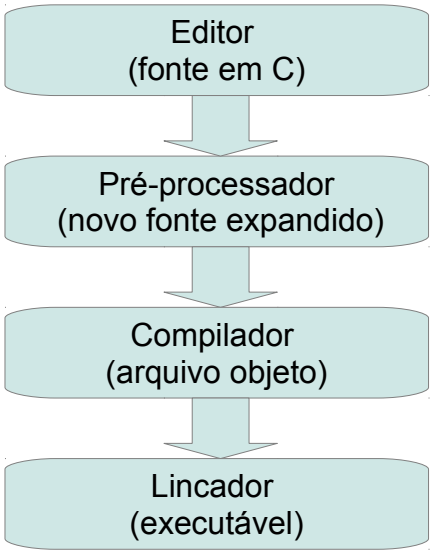
```
#include<stdio.h>

int main(void)
{
printf("Alo Mundo!\n");
return(0);
}
```

3.3 Visão geral de um programa C

A geração do programa executável a partir do programa fonte obedece a uma sequência de operações antes de tornar-se um executável. Depois de escrever o módulo fonte em um editor de

textos, o programador aciona o compilador. Essa ação desencadeia uma sequência de etapas, cada qual traduzindo a codificação do usuário para uma forma de linguagem de nível inferior, que termina com o executável criado pelo lincador. A figura a seguir esclarece este processo.



3.4 Sintaxe

A sintaxe são regras para a construção de um programa em linguagem C. Estas regras estão relacionadas com os **tipos**, as **declarações**, as **funções** e as **expressões**.

Os tipos definem as propriedades dos dados manipulados em um programa.

As declarações expressam as partes do programa, podendo dar significado a um identificador, alocar memória, definir conteúdo inicial, definir funções.

As funções especificam as ações que um programa executa quando roda.

A determinação e alteração de valores, e a chamada de funções de I/O são definidas nas expressões.

As funções são as entidades operacionais básicas dos programas em C, que por sua vez são a união de uma ou mais funções executando cada qual o seu trabalho. Há funções básicas que estão definidas nas bibliotecas do C. As funções printf() e scanf() por exemplo, permitem respectivamente escrever na tela e ler os dados a partir do teclado. O programador também pode definir novas funções em seus programas, como rotinas para cálculos, impressão, etc. Todo programa em C inicia sua execução chamando a função main(), sendo obrigatória a sua declaração no programa principal.

Comentários no programa são colocados entre /* e */ ou iniciados com //, não sendo considerados na compilação.

Cada instrução encerra com ; (ponto e vírgula) que faz parte do comando.

3.5 Identificadores

São nomes usados para se fazer referência a variáveis, funções, rótulos e vários outros objetos definidos pelo usuário. O primeiro carácter deve ser uma letra ou um sublinhado. Os 32 primeiros caracteres de um identificador são significativos. É “case sensitive”, ou seja, as letras maiúsculas diferem das minúsculas. E não se pode utilizar espaços nos nomes de identificadores. Também não devemos usar acentuação e caracteres especiais como o cedilha (ç).

3.6 Palavras reservadas

Na linguagem C são 32 palavras reservadas. Todas as palavras reservadas do C são minúsculas. Uma palavra reservada não pode ser usada para nenhum outro propósito em um programa. A tabela abaixo mostra as palavras reservadas conforme definido pelo padrão ANSI para a 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

3.7 Estrutura Básica de um Programa em C

A linguagem C é baseada no conceito de funções. Um programa C é um conjunto de funções. Uma biblioteca é um arquivo contendo as funções padrão que seu programa pode usar. Essas funções incluem todas as operações de entrada e saída (E/S) e também outras rotinas úteis para realizar as tarefas mais comumente necessárias.

Todos os programas escritos em linguagem C iniciam pela inclusão das bibliotecas de funções. A inclusão das bibliotecas é feita através da diretiva #include, seguida pelo nome da biblioteca entre sinais de maior e menor. Veja o exemplo a seguir.

```
#include<stdio.h>
```

A linguagem C possui um grande número de bibliotecas de funções, que podem variar de

um compilador para outro, porem as bibliotecas básicas são padronizadas. A tabela a seguir apresenta algumas das principais bibliotecas.

Biblioteca	Utilização
stdio.h	**Standard Input Output** (entradas e saídas padrão): esta biblioteca contém a definição da estrutura FILE, usada para todas as entradas (input) e saídas (output), além das definições de todas as funções que lidam com a abertura, fechamento, etc de arquivos. A famosa função printf também é definida aqui, juntamente com sprintf, fprintf e toda a família de funções relacionadas.
math.h	Funções Matemáticas: sin, cos, atan, log, exp, etc. Aqui encontramos trigonometria (sin, cos, tan, atan, etc), arredondamentos (ceil, floor), logaritmos (log, exp, log10, etc), raiz quadrada e cúbica (sqrt, cbrt) e constantes como pi.
stdlib.h	**Standard library functions** (funções da biblioteca padrão): Contém abort (término anormal do programa), exit (término normal), atoi, itoa (conversão de texto para inteiro), malloc, calloc, free (módulo de memória dinâmica), rand, srand (números randômicos), putenv, getenv (administração do ambiente), qsort (ordenação), strtod, strtol (conversão de string para double/long), sleep (suspender a execução por um certo período de tempo), etc.

É possível observar que as bibliotecas possuem a extensão .h, isso ocorre pois cada biblioteca é um arquivo armazenado junto ao compilador.

3.8 Funções

Tudo nos programas em C está associado a funções. Uma função é um trecho de programa independente que executa um conjunto de comandos para realizar determinada tarefa. A estrutura básica de uma função é a seguinte.

- As funções inicias com a identificação de seu tipo.
- Em seguida vem seu nome.
- Seguido de um conjunto de parênteses, que pode ou não conter parâmetros.
- Na sequência temos uma chave para iniciar o corpo da função, “{”.
- No corpo são introduzidos os comandos.
- Para encerrar temos outra chave fechando a função, “}”.

O formato de uma função segue o seguinte modelo:

```
<tipo_da_funcao>  <NomeDaFuncao> ([Lista_de_Parametros])
{
// corpo da função
}
```

A seguir temos um exemplo de função.

```
void mensagem() //função para imprimir uma mensagem na tela do computador.
{
printf("Alo Alunos do IFC!");
}
```

O uso de funções permite reaproveitar trecho de código, com isso, minimizamos o tamanho final do programa pela redução do número de linhas e consequentemente o código-fonte fica melhor organizado.

Ainda, as funções podem se organizadas de tal forma que sejam fáceis de serem localizadas e alterada, permitindo manutenções e melhorias com maior agilidade.

3.8.1 Parâmetros

A fim de tornar mais amplo o uso de uma função, a linguagem C permite o uso de parâmetros. Este parâmetros possibilitam que se definida sobre quais dados a função deve operar.

Para definir os parâmetros de uma função o programador deve explicitá-los como se estive declarando uma variável, entre os parênteses do cabeçalho da função. Caso precise declarar mais de um parâmetro, basta separá-los por vírgulas. No exemplo a seguir temos a função SOMA que possui dois parâmetros, sendo o primeiro um **float** e o segundo um **int**.

```
#include <stdio.h>

void soma(float a, int b)
{
   float resultado;     // a declaração de variáveis é igual ao que
                        // se faz na função main
   resultado = a+b;
   printf("A soma de %.3f com %d é %.3f\n", a, b, resultado);
}

int main()
{
    soma(1.4, 1);
}
```

Note que a função **soma** está antes dela ser chamada, isso é necessário para que a função **main** conheça as funções que existem no projeto.

A princípio podemos tomar com regra a seguinte afirmativa toda função deve ser declarada antes de ser usada.

Alguns programadores preferem que o início do programa seja a primeira parte de seu programa. Para isto a linguagem C permite que se declare uma função, antes de defini-la. Esta declaração é feita através do protótipo da função. O protótipo da função nada mais é do que o trecho de código que especifica o nome e os parâmetros da função.

No exemplo a seguir a função SOMA é prototipada antes de ser usada e assim pôde ser chamada antes de ser definida.

```
#include <stdio.h>

void soma(float a, int b);  // Protótipo da função SOMA

int main()
{
   soma(1.4, 1);
}

void soma(float a, int b)
{
  float resultado;     // a declaração de variáveis é igual ao que
                       // se faz na função main
  resultado = a+b;
  printf("A soma de %.3f com %d é %.3f\n", a, b, resultado);
}
```

3.8.2 Retorno de uma função

A função pode também retornar um único valor para quem a chamou, o tipo deve ser respeitado para não ocorrer erros no retorno.

Veja o exemplo da função soma, agora quem irá imprimir o resultado é a função que chamou:

```
#include <stdio.h>

float soma(float a, int b);
```

```
int main()
{
    float retorno;
    retorno = soma(1.4, 1);
    printf("O resultado é %.3f\n", retorno);
   return 0;
}

float soma(float a, int b)
{
   float resultado;
   resultado = a+b;
   return resultado;

}
```

3.9 A função main()

Um programa em C consiste de uma ou várias funções. A função main() deve existir em algum lugar de seu programa e marca o início da execução do programa. Assim um programa típico em linguagem C tem a seguinte aparência.

```
#include<stdio.h>

int main(void)
{
printf("Alo Mundo!\n");
return(0);
}
```

No exemplo anterior a função principal (main), por onde o programa inicia, esta chamando duas outras funções. A primeira é a função “printf()”, que tem a finalidade de imprimir mensagens na tela do computador. A função “printf()” está definida na biblioteca stdio.h. Então, para que o seu programa possa utilizar a função “printf()” é necessário incluir a biblioteca stdio.h ao seu programa. A segunda função é a função “return()” que é responsável por informar o sistema operacional da situação de termino do programa. Estas funções serão estudadas mais profundamente no futuro, e foram introduzidas agora apenas como exemplo.

A função “printf()” recebeu como parâmetro o texto a ser apresentado na tela do computador. Este texto deve estar sempre entre aspas duplas (“”) . A instrução “\n” junto ao texto instrui o programa a iniciar uma nova linha em sua tela.

3.10 Exercícios

1. Qual é a função do compilador na criação de programas em C?
2. Quais dos seguintes nomes de identificadores são válidos e quais são inválidos? Justifique?
    - Variavel
    - nome 1
    - entrada2
    - saida_2
    - 1contador
    - _valor_TOTAL
3. Construa um programa que apresenta na tela do computador o seu nome.
4. Construa um programa que apresenta na tela do computador o seguinte texto, exatamente da forma que ele aparece.

```
Instituto Federal de Educação, Ciência e Tecnologia Catarinense
      Disciplina: Informática para Automação
      Nome: <seu nome>
      Data: DD/MM/AAAA

Exercício de programação em linguagem C
```

AULA 4 - VARIÁVEIS, TIPOS DE DADOS E OPERADORES

Criação de variáveis e manipulação de dados

4.1 Objetivo:

O objetivo desta aula é mostrar aos alunos como a linguagem C armazena os dados na memória. Este processo se da através das variáveis, cada uma com seu tipo de dado. A linguagem C apresenta uma grande variedade de tipos de dados que podem ser utilizados em nossos programas.

Também serão abordados os operadores que são responsáveis pela manipulação das informações contidas nas variáveis.

4.2 Variáveis na linguagem C

Todas as informações utilizadas em um programa estão armazenadas na memória. Cada tipo de informação tem um determinado tamanho na memória. Para que se possa manter um controle sobre os dados armazenados na memória são utilizadas as variáveis. As variáveis são responsáveis por dar um nome a um determinado local da memória, de forma que fique mais fácil manipular os dados ali armazenados. As variáveis também são responsáveis por informar ao compilador os tipos de dados armazenados naquele local.

4.2.1 Declaração de variáveis

Na linguagem C a declaração das variáveis é feita da seguinte forma.

tipo_da_variável nome_da_variável;

A seguir temos um exemplo de declaração de variável em C.

```
int x;
```

No exemplo “int” é o tipo da variável e “x” é o nome da variável. Como todos os comandos em C a declaração de uma variável termina com um ponto e vírgula. Na linguagem C é necessário declarar qualquer variável antes de utilizá-la. As variáveis devem ser declaradas no inicio programa, após a inclusão das bibliotecas, ou no início das funções.

4.3 Tipos de dados

A linguagem C tem um grupo grande de tipos de dados pré definidos, e também permite que

o programador crie seus próprios tipos de dados.

Neste momento estudaremos os dois principais tipos de dados. Os tipos inteiros e os tipos reais.

4.3.1 Tipos de dados inteiros

Os tipos de dados inteiros ocupam de 8 a 64 bits na memória, e podem conter valores numéricos e letras (caracteres).

Como existem várias arquiteturas, com processadores de barramentos com número de bits diferentes, também existem variações na definição dos tipos de dados na linguagem C. A tabela a seguir apresenta os tipos de dados adotados pelo compilador GCC. Alguns tipos podem conter o mesmo número de bits, isso é tolerado pela linguagem C.

Tipo	Bits	Range
char	8	-128 a 127 ou letras
unsigned char	8	0 a 255
short int	16	-32.768 a 32.767
unsigned short int	16	0 a 65.535
int	32	−2.147.483.648 a 2.147.483.647
unsigned int	32	0 a 4.294.967.295
long int	32	−2.147.483.648 a 2.147.483.647
long unsigned int	32	0 a 4.294.967.295
long long int	64	−9.223.372.036.854.775.808 a 9.223.372.036.854.775.807
unsigned long long int	64	0 to 18.446.744.073.709.551.615

A seguir temos alguns exemplos de declaração de variáveis do tipo inteiro.

```
int contador;
unsigned int y;
char temperatura;
char velocidade=0;
char letra='A';
long int Z=126543;
```

Observe que é possível atribuir um valor a variável utilizando o operador “=”.

O tipo de dados “char” também é utilizado para armazenar uma letra, assim para atribuir uma letra a uma variável a letra deve estar entre aspas simples.

4.3.2 Tipos de dados reais

Os tipos de dados inteiros podem armazenar apenas números inteiros, para armazenar números reais, ou seja, valores fracionários devem ser utilizados os tipos de dados reais. A tabela a seguir apresenta estes tipos.

Tipo	Bits	Range
float	32	3,4e-38 a 3.4e+38 com 7 dígitos de precisão.
double	64	1,7e-308 a 1,7e+308 com 15 dígitos de precisão.

A seguir temos alguns exemplos.

```
float x;
double y=12.124;
```

É importante salientar que o compilador utiliza a notação inglesa, assim o número fracionário utiliza o ponto (.) e não a virgula (,).

4.3.3 Definições

A linguagem C fornece uma ferramenta para definir constantes e definições. O comando utilizado é o “define”. Este comando da um “apelido” a uma expressão ou a um valor, e este apelido é usado no restante do programa. É importante salientar que a definição não pode ser alterada em tempo de definição. A seguir temos alguns exemplos de definições.

```
#define valor 123
#define pi 3.14
```

4.4 Operadores

A manipulação dos dados é realizada através dos operadores. Existem vários tipos de operadores, tais como, lógicos, aritméticos, de comparação etc.

4.4.1 Operador de atribuição

O operador de atribuição em C é o sinal de igual "=". O operador de atribuição é utilizado para atribuir um valor a uma variável. A seguir temos alguns exemplos.

```
X=12;
y=(X*5)-2;
```

4.4.2 Operadores aritméticos

Os operadores aritméticos tem a função de realizar as operações matemáticas básicas. A tabela a seguir apresenta estes operadores.

Operador	Exemplo	Descrição
++	a++	Incrementa o valor de a (a=a+1)
- -	a--	Decrementa o valor de a (a=a-1)
+	a + b	a mais b
-	a - b	a menos b
*	a * b	a vezes b
/	a / b	a dividido por b
%	a % b	Resto de a/b
=	a = b	a recebe o valor de b
+=	a += b	a recebe o valor de a + b
-=	a -= b	a recebe o valor de a - b
*=	a *= b	a recebe o valor de a * b
/=	a /= b	a recebe o valor de a / b
%=	a %= b	a recebe o resto de a / b

4.4.3 Operadores lógicos

Os operadores lógicos são aqueles que realizam operações sobre os bits das variáveis. A tabela a seguir apresenta estes operadores.

Operador	Exemplo	Descrição
>>	a >> b	A deslocado b bits para a direita
<<	a << b	A deslocado b bits para a esquerda
&	a & b	operação AND entre a e b
|	a | b	operação OR entre a e b
^	a ^ b	operação XOR entre a e b
!	!a	operação NOT em a

4.4.4 Outros operadores

A linguagem C fornece outros operadores, tais como, os operadores de comparação, etc. Estes operadores serão estudados em outras aulas. Os operadores podem ser mesclados para formar expressões complexas independente de seus tipos.

Outras funções matemáticas estão disponíveis através da biblioteca math.h, como por exemplo a função sqrt(x) que calcula a raiz quadrada de x e a função pow(x,y) que eleva o valor x na potência y.

4.5 Alguns exemplos

A seguir temos alguns exemplos de programas que utilizam variáveis e operadores.

```
#include<stdio.h>

int main(void)
{
int x;
int y=5;
x=y*2;
printf("x = %d\n",x);
return(0);
}
```

O programa irá apresentar na tela a mensagem “x = 10”.

```
#include<stdio.h>
#define pi 3.1415

int main(void)
{
float raio=5.0;
float per;
per=raio*pi;
printf("perimetro = %f\n",per);
return(0);
}
```

O programa irá apresentar na tela a mensagem “perimetro = 15.707500”.

```
#include<stdio.h>

int main(void)
{
int a,b,c;
a=10;
b=3;
c=a%b;
c+=10;
c--;
printf("Resultado %d\n",c);
return(0);
}
```

O programa irá apresentar na tela a mensagem “Resultado 10”.

4.6 Exercícios

1. Qual é saída do seguinte programa?

```
#include<stdio.h>

int main(void)
{
int x;
int y;
int z;
x=10;
y=-20;
z=x-y;
printf("Resultado 1 = %d\n",z);
z=y+x;
printf("Resultado 2 = %d\n",z);
z=z/(x/5);
printf("Resultado 3 = %d\n",z);
return(0);
}
```

2. Faça um programa para resolver a equação: $x = \frac{(32 - y) * 2}{a + b}$ Sendo a=10, b=2 e y =5;

AULA 5 - O AMBIENTE CODEBLOCKS

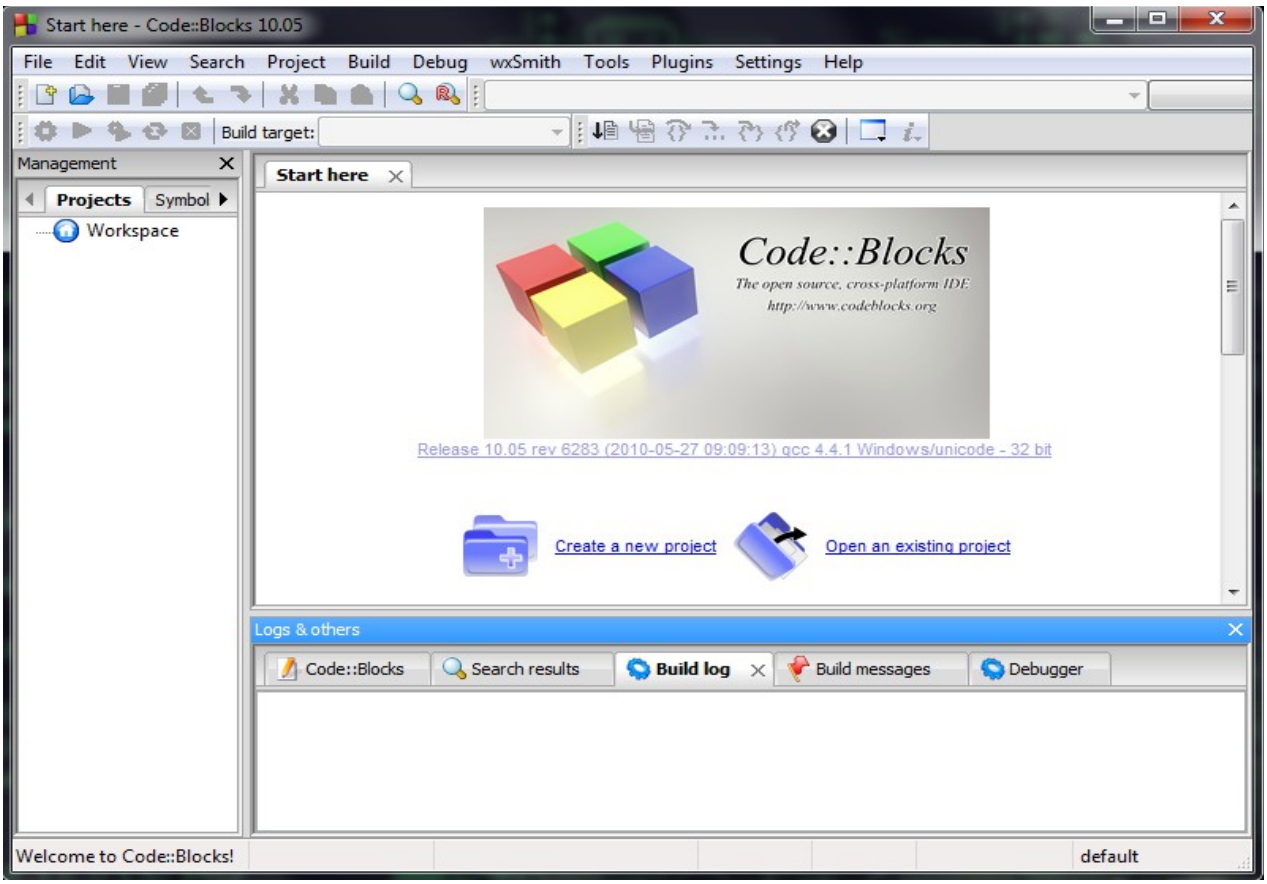
O Ambiente de Desenvolvimento CODEBLOCKS e o compilador GCC

5.1 Objetivo:

Esta aula tem o objetivo de descrever como obter, instalar e operar o software Codeblocks. Este software é um ambiente de desenvolvimento utilizado para a programação de computadores em linguagem C e C++. O Codeblocks é compatível com vários compiladores, nestas aulas usaremos o "GNU Compiler Collection" ou apenas GCC. Também serão estudadas funções de entrada e saída.

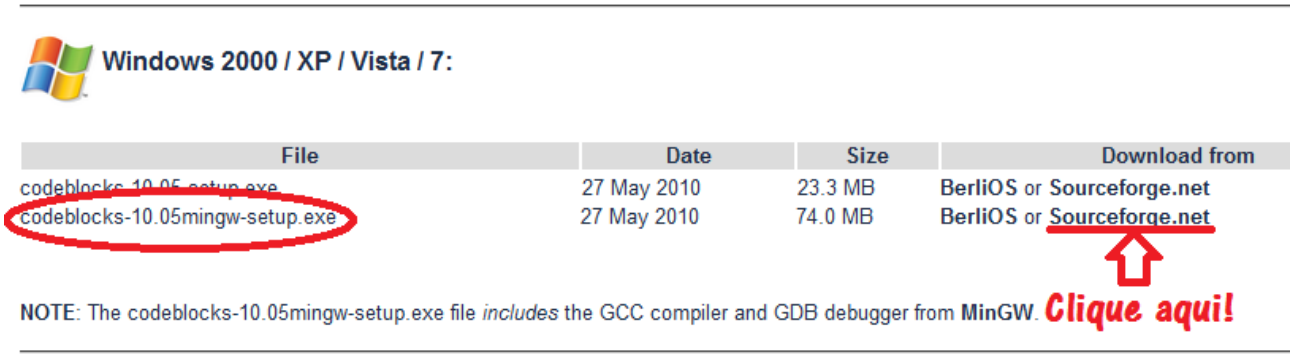
5.2 O ambiente de desenvolvimento Codeblocks

O Codeblocks é um software "open source", ou seja, ele é gratuito e de código aberto. Está disponível para várias plataformas e tem o objetivo de ser um ambiente integrado de desenvolvimento sem custos e de qualidade. Maiores informações sobre o Codeblocks podem ser obtidas em seu site, http://www.codeblocks.org/ . A figura a seguir apresenta a interface deste software.



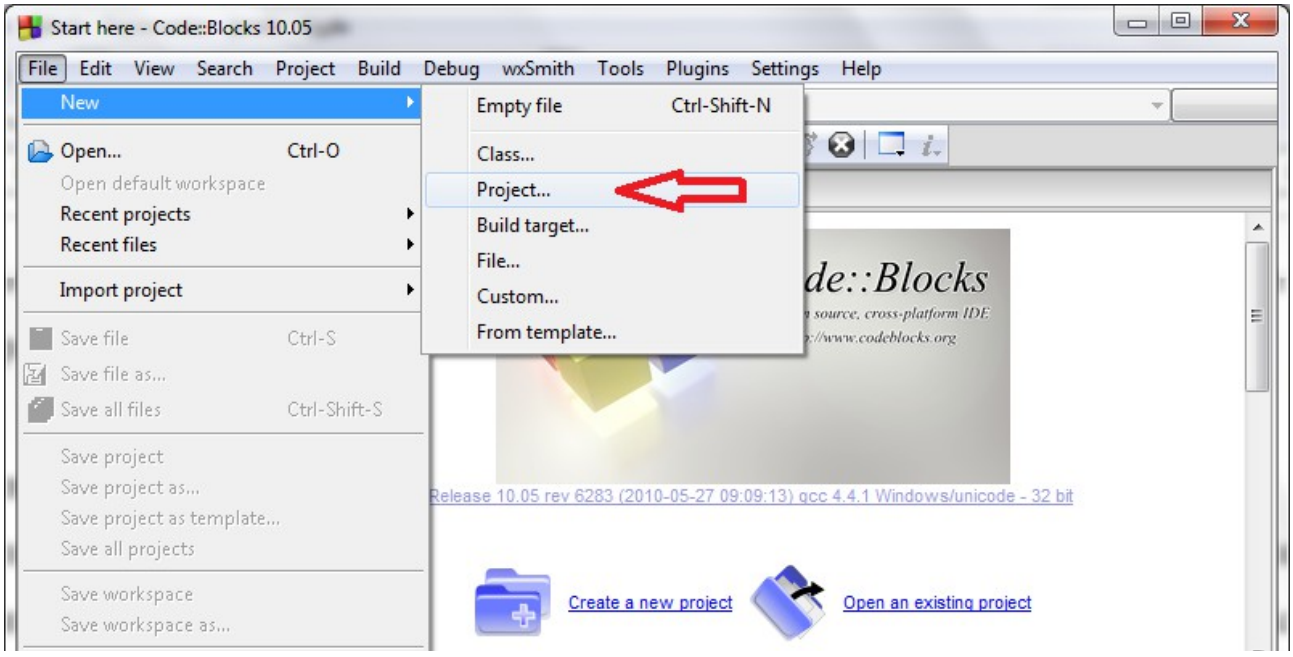
5.2.1 Obtenção do Codeblocks

A versão disponível no momento da elaboração deste material é a versão 10.05, e pode ser obtida gratuitamente da internet no site http://www.codeblocks.org/downloads/26 Existem várias opções de download, aqui estudaremos a versão para o sistema operacional windows. Para instalação no windows existem duas versões, uma com e outra sem o compilador. A opção correta de download é a versão completa. A figura a seguir mostra qual é a opção correta de download.



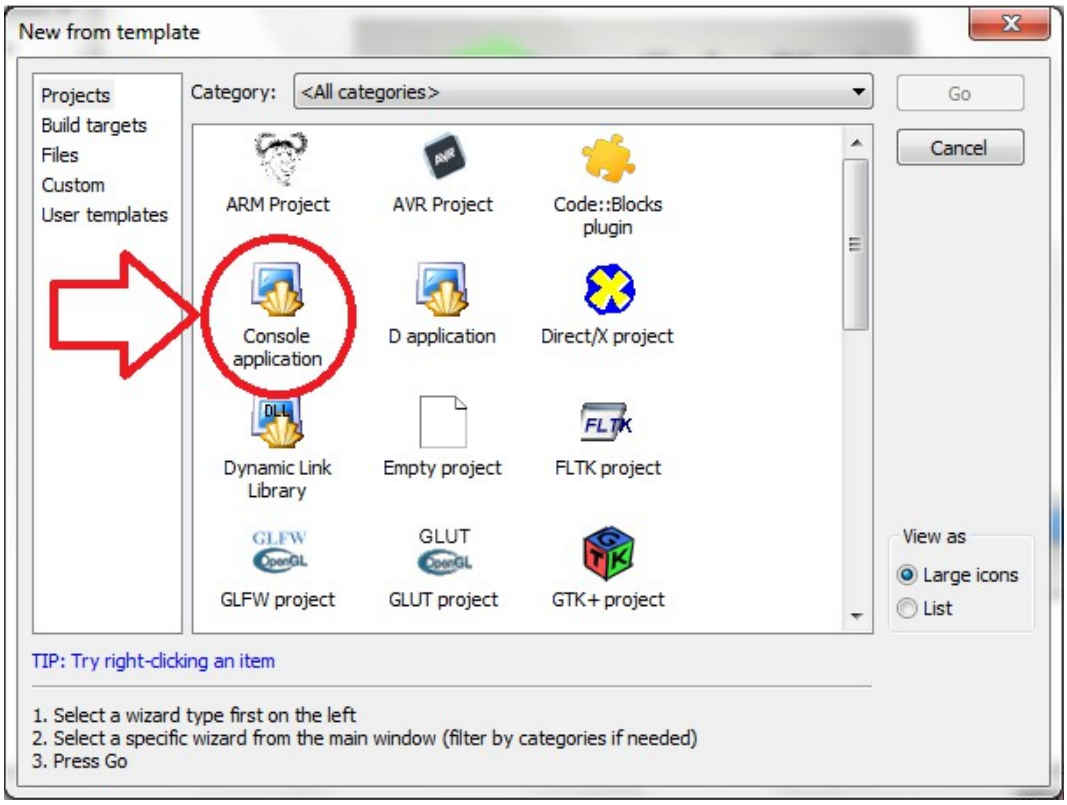
Apos o download o Codeblocks pode ser instalado normalmente como qualquer outro programa para windows.

5.2.2 A criação de um novo projeto no Codeblocks

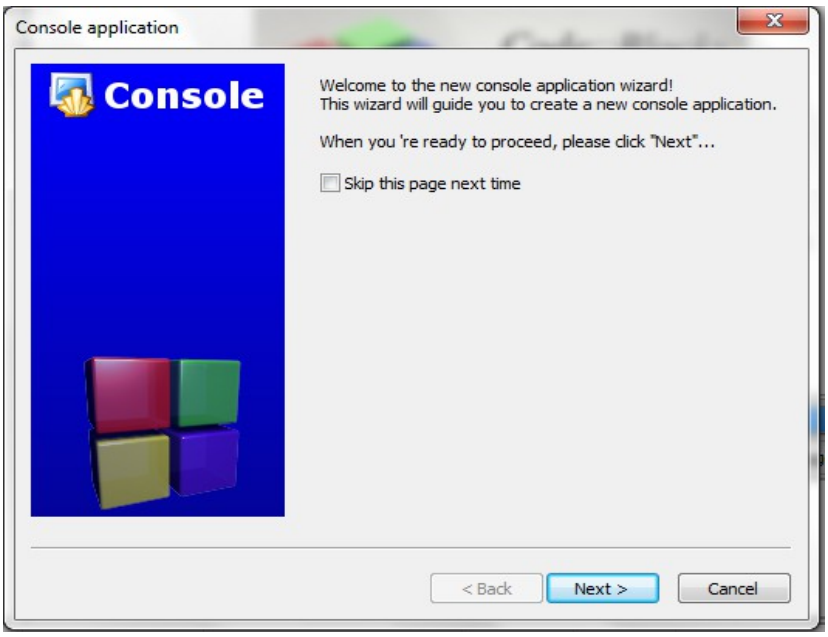


Para que se possa desenvolver um novo programa utilizando o Codeblocks é necessário criarmos um projeto. A figura a seguir mostra que para criarmos um novo projeto basta clicarmos em File > New > Project...

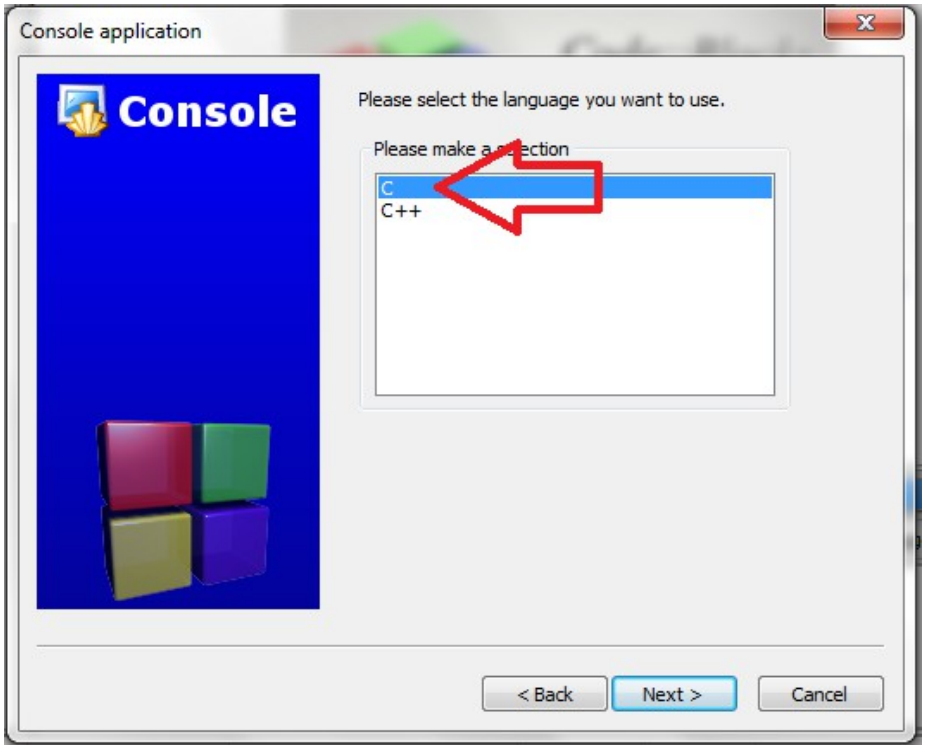
Apos iniciarmos um novo projeto é apresentada uma tela onde devemos escolher o tipo do projeto que queremos criar. O Codeblocks desenvolve uma infinidade de tipos de programas, porem para nossas aulas sempre usaremos a opção "**Console Application**". A figura a seguir apresenta esta tela.



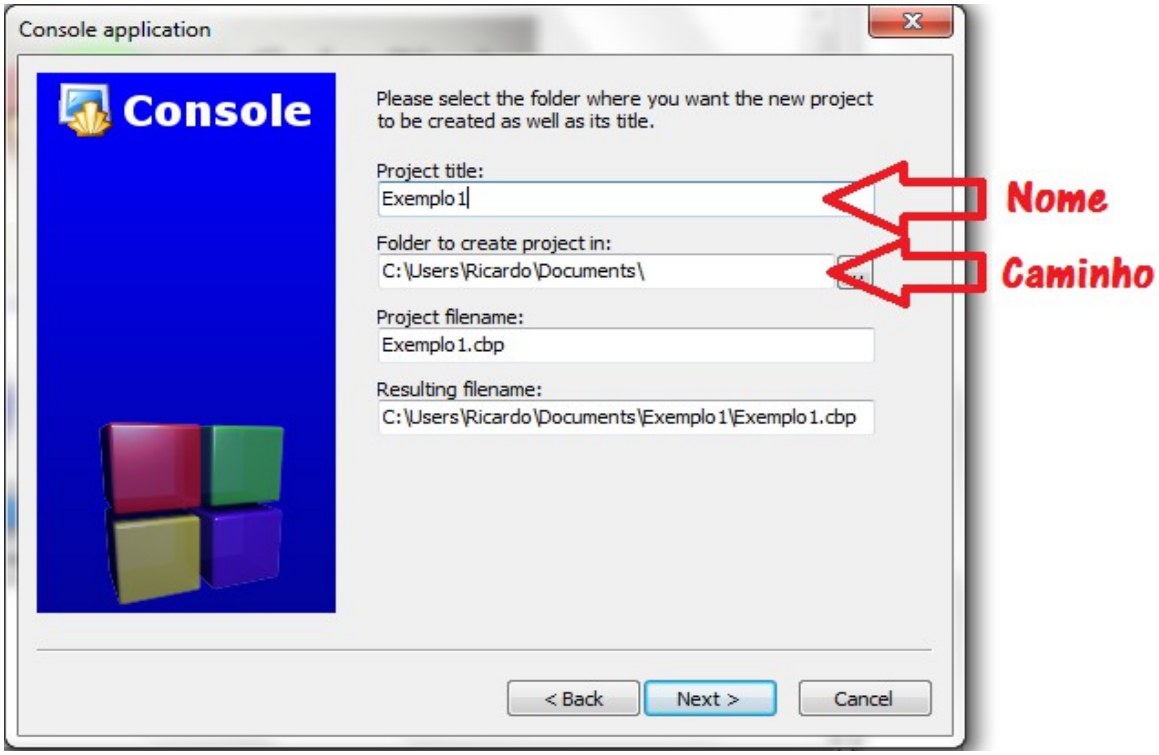
Apos selecionar a opção **Console Application**, é só clicar em **Go.** A seguir é apresentada a seguinte tela.



Basta clicar em **Next >** para prosseguir. A próxima tela pede se desejamos desenvolver um programa em linguagem C ou em linguagem C++.



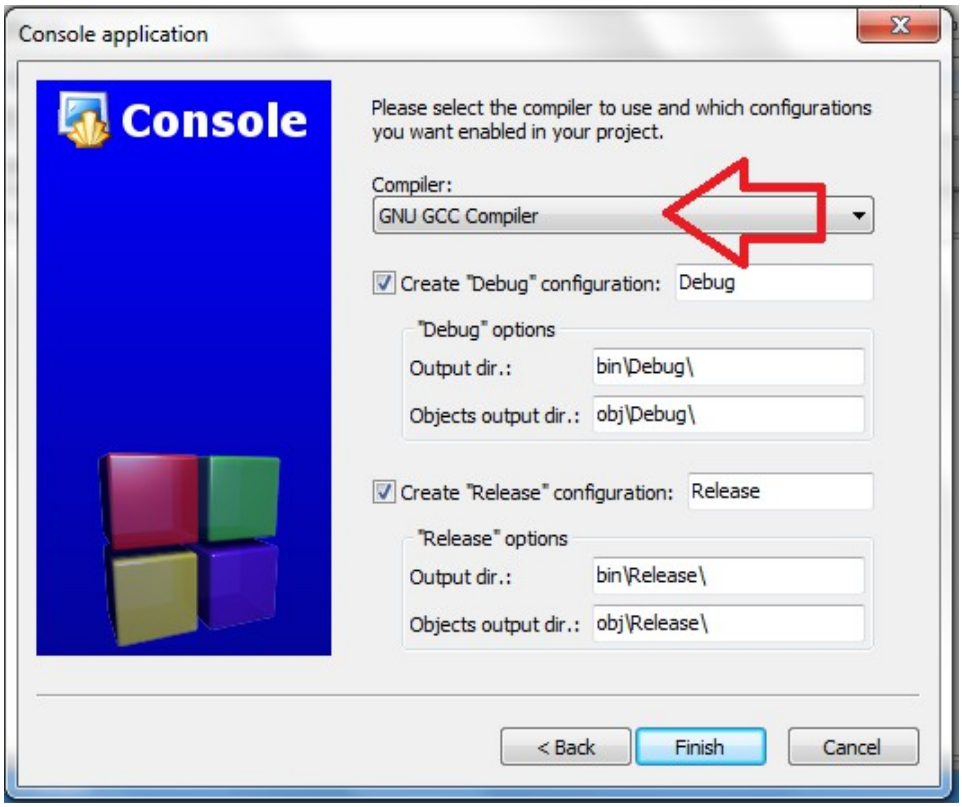
Devemos escolher a linguagem C e clicar em **Next >.** A seguir é apresentada a tela onde definimos o nome do projeto e o caminho onde o projeto deve ser salvo.



O nome e o caminho do projeto são informações muito importantes, pois definem como e onde o programa gerado será salvo.

Em geral é interessante não utilizarmos espaços ou caracteres especiais nos nomes de pastas ou de projetos, pois em alguns casos o compilador tem problemas em compilar estes programas.

Apos definirmos o nome e o caminho do projeto basta clicar em **Next >**. A tela a seguir pede qual compilador desejamos usar.



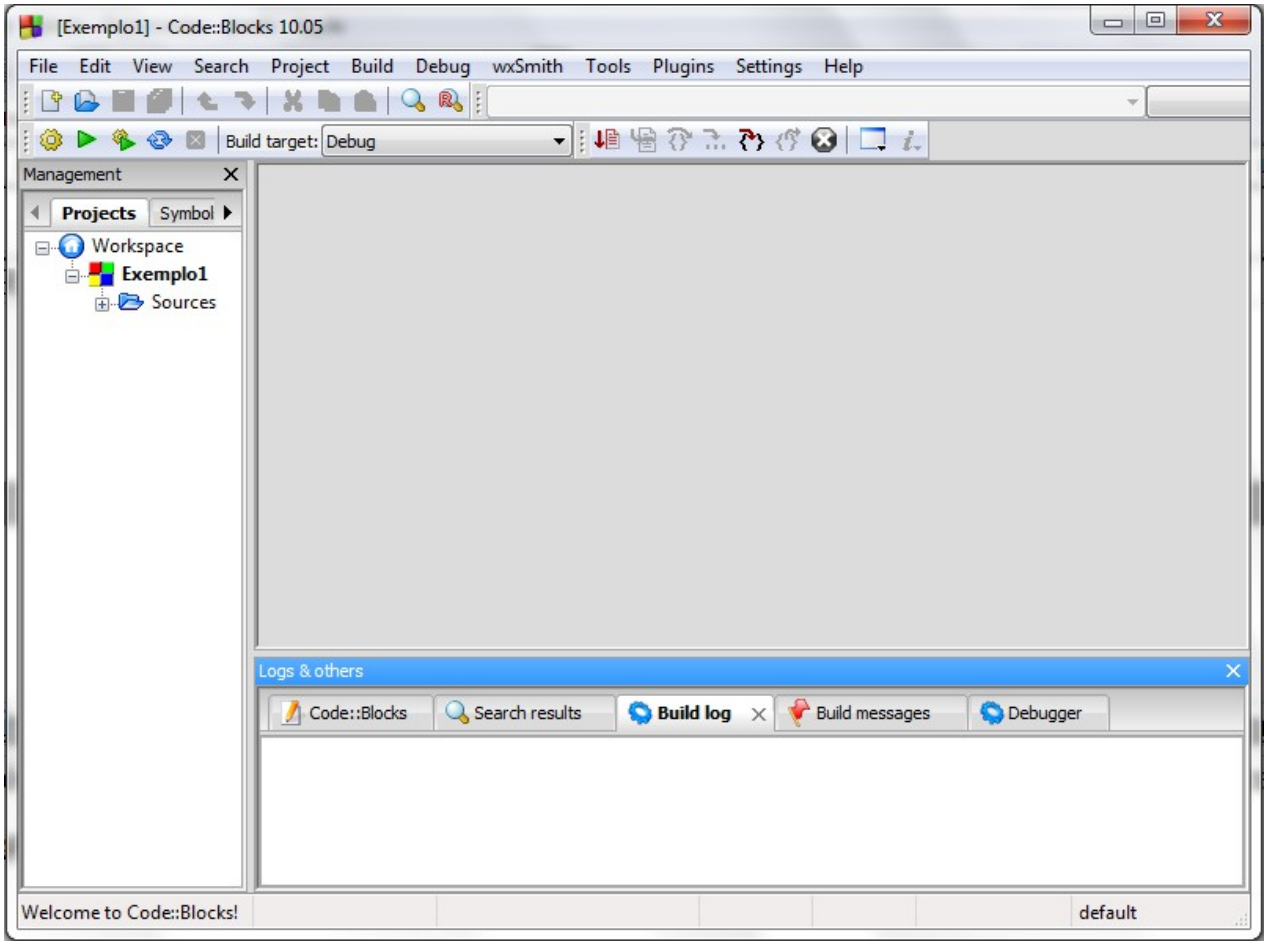
O compilador utilizado deve ser o **GNU GCC Compiller**. As opções nesta tela não dever ser alteradas, bastando clicar em **Finish**.

O ambiente de desenvolvimento Codeblocks é compatível com um grande número de compiladores, porem para que outros compiladores possam ser utilizados é necessário instalá-los no sistema separadamente e em seguida configurar o codeblocks.

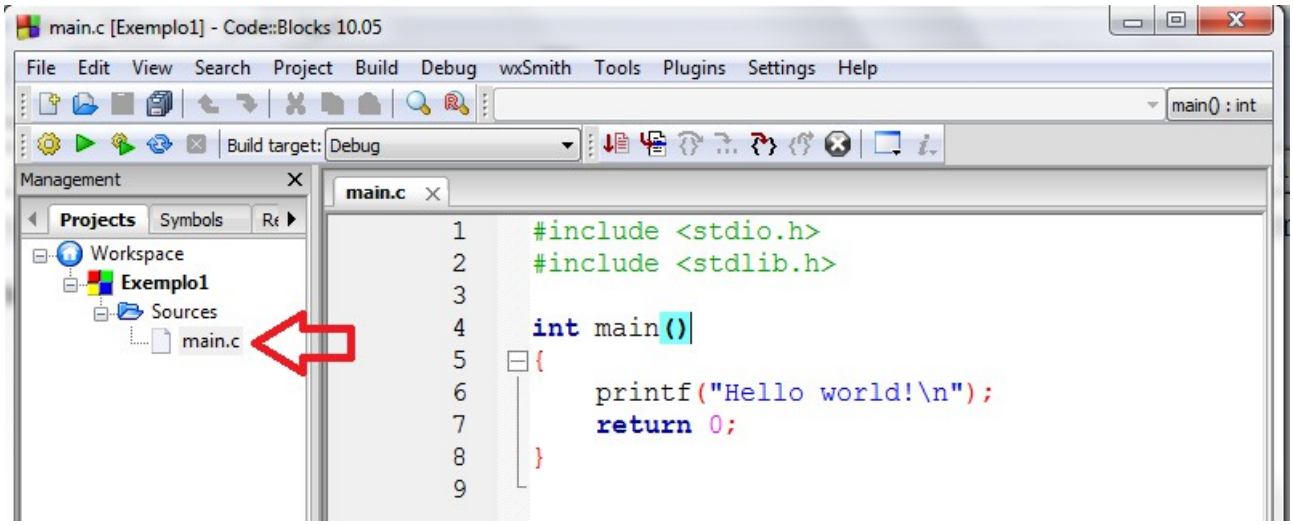
Em nossas aulas o compilador usado é o GNU GCC, assim se a versão instalada do codeblocks é a versão completa, o GNU GCC já é instalado automaticamente.

O codeblocks detecta automaticamente a presença do compilador GNU GCC, não sendo necessária nenhuma configuração adicional.

Após a criação de um novo projeto a tela apresentada esta em branco, como mostrado na figura a seguir.

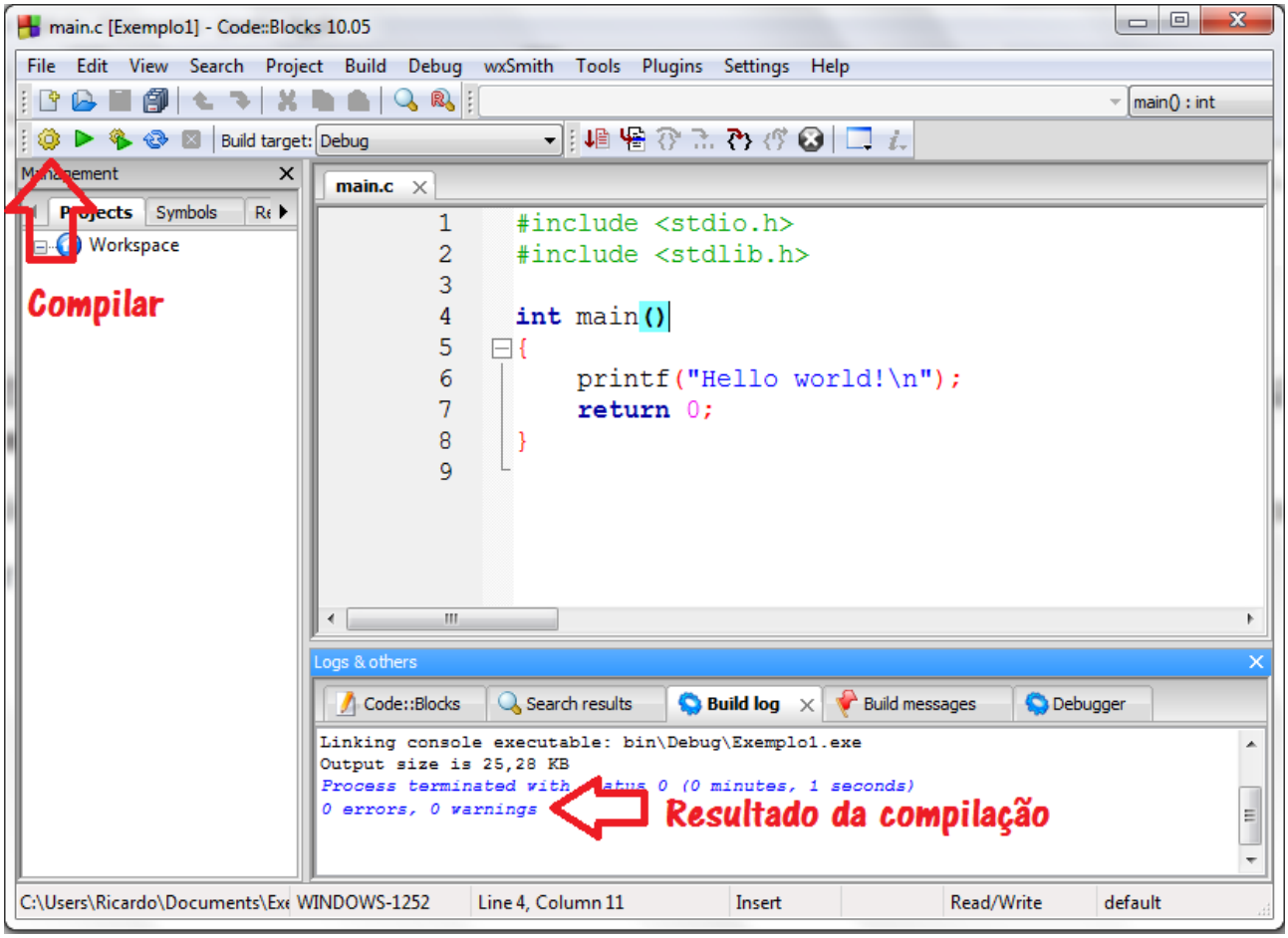


O Codeblocks já cria automaticamente um arquivo contendo a função **main()**, para abrirmos este arquivo basta localizá-lo na caixa a esquerda da tela e clicar duas vezes sobre ele com o mouse. A figura a seguir apresenta este processo.

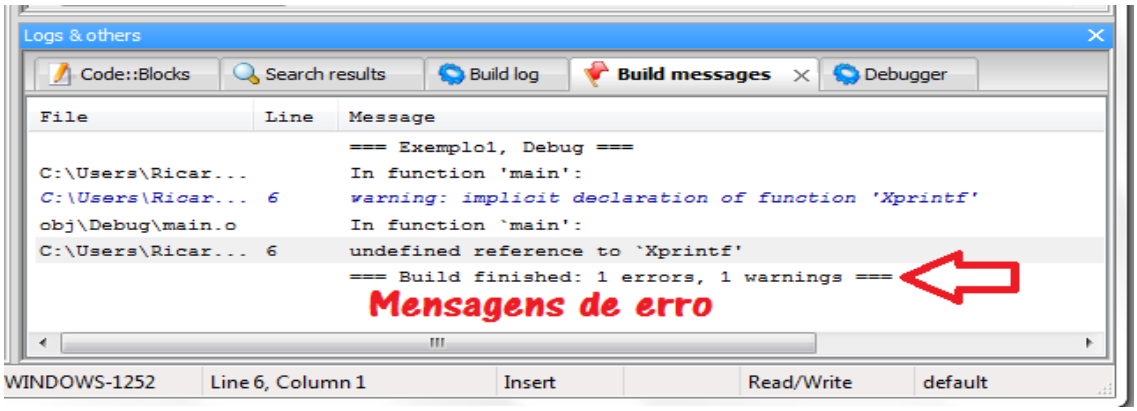


5.2.3 Compilando e rodando nossos programas

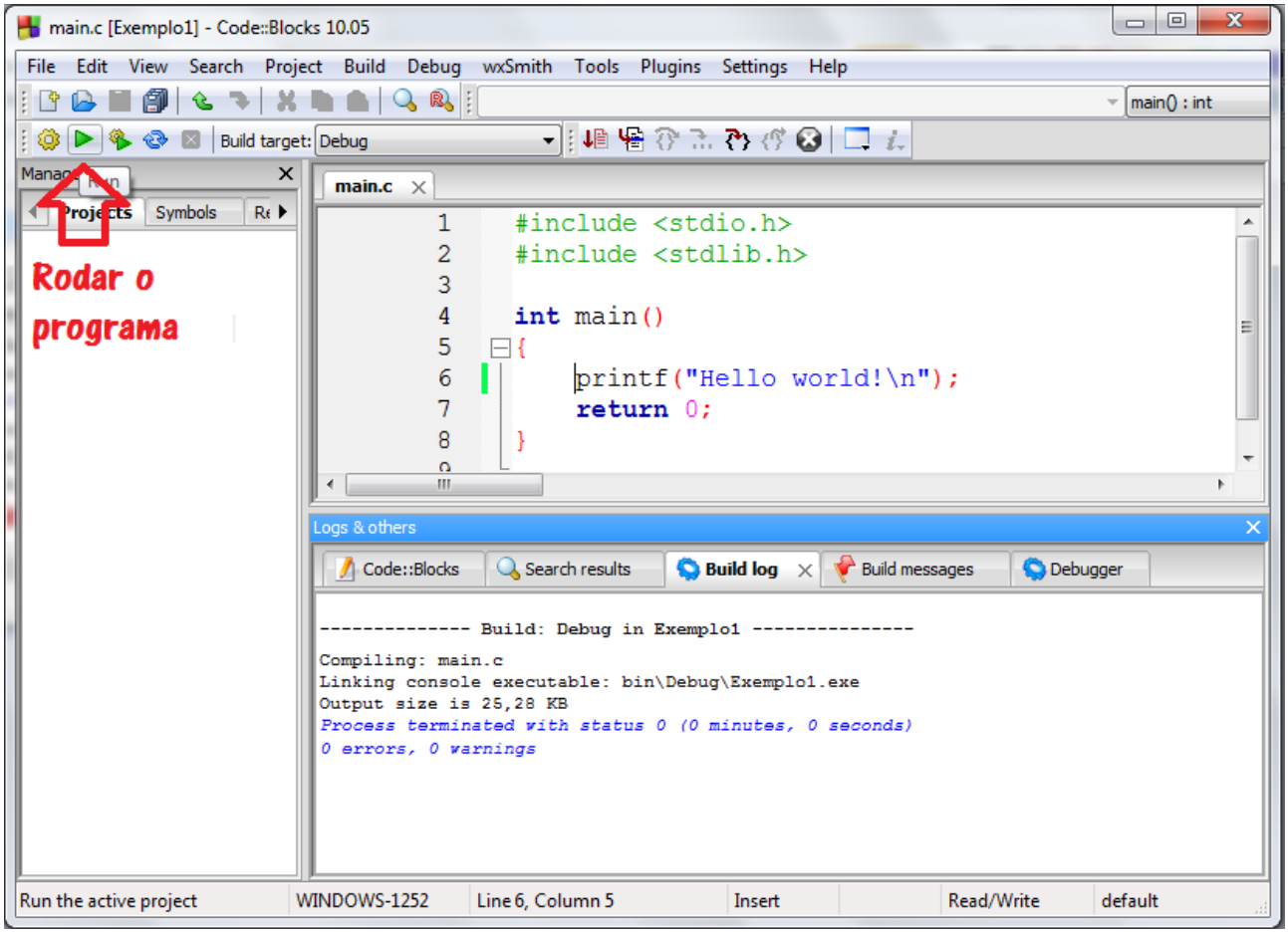
Após digitarmos nossos programas é necessário compilá-los. Para compilar um programa basta clicar no botão **Build** conforme a figura a seguir.



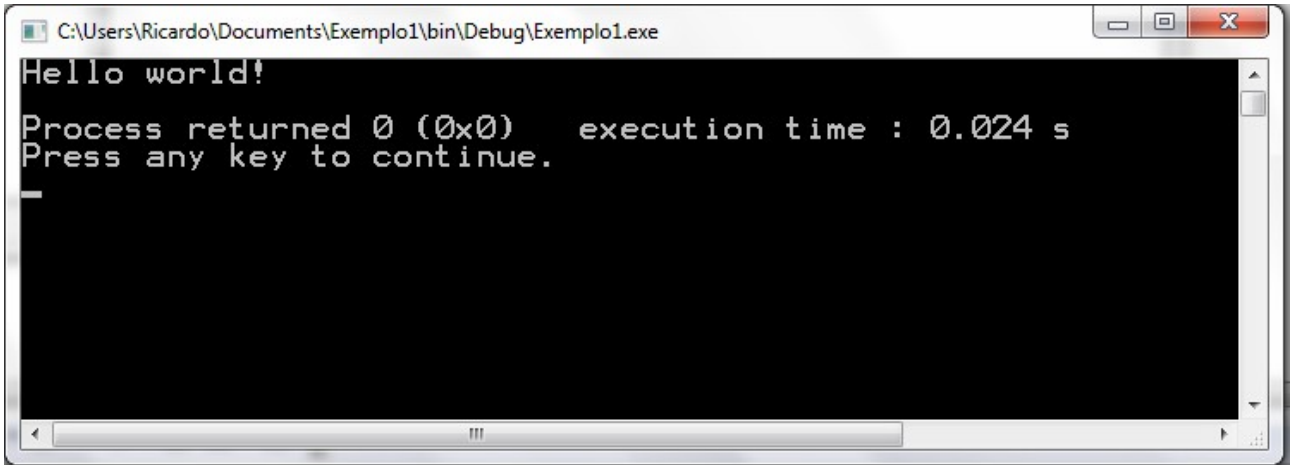
O resultado do processo de compilação é apresentado na parte inferior da tela, conforme figura acima. Nesta janela é possível verificar se o programa foi compilado com sucesso ou se houveram erros. A figura acima apresenta um resultado de compilação sem erros, já a figura a seguir apresenta um exemplo de resultado de compilação com erros.



Se o programa foi compilado com sucesso é possível testar o programa. Para testarmos o funcionamento de nosso programa basta clicar em **Run** como na figura a seguir.



Assim o programa é executado e o resultado é apresentado em uma nova janela. A figura a seguir mostra um exemplo de janela de saída.



Os programas desenvolvidos em nossas aulas são programas em modo texto.

5.3 O compilador GNU GCC

O GNU Compiler Collection (chamado usualmente por GCC) é um conjunto de compiladores de linguagens de programação produzido pelo projecto GNU. É software livre distribuído pela Free Software Foundation (FSF) sob os termos da GNU GPL, e é um componente-chave do conjunto de ferramentas GNU. É o compilador padrão para sistemas operacionais UNIX e Linux. Originalmente o GCC suportava somente a linguagem de programação C e era designado GNU C Compiler (compilador C GNU). Com o tempo ganhou suporte às linguagens C++, Fortran, Ada, Java e Objective-C, entre outras.

A figura a seguir apresenta o logotipo do GCC. (fonte: http://pt.wikipedia.org/wiki/GNU_Compiler_Collection)

Este compilador foi escolhido para ser utilizado em nossas aulas por ser gratuito, por ser muito popular, por ser compatível com vários sistemas operacionais e finalmente por seguir as normas que regulamentam a linguagem C.

5.4 Exercícios

1. Utilizando o Codeblocks crie um projeto de nome **exercicio_1**, salve-o na pasta documentos, digite o programa do exemplo dado na função scanf(), compile e teste o programa gerado. A seguir substitua todo o conteúdo do programa pelo exemplo dado na função prinf();

2. Crie um projeto no codeblocks e digite o seguinte programa. Apos digitá-lo, compile e verifique seu funcionamento.

```
#include<stdio.h>

int main(void)
{
int x;
int y=5;
x=y*2;
printf("x = %d\n",x);
return(0);
}
```

3. Crie um projeto no codeblocks e digite o seguinte programa. Apos digitá-lo, compile e verifique seu funcionamento.

```
#include<stdio.h>
#define pi 3.1415

int main(void)
{
float raio=5.0;
float per;
per=raio*pi;
printf("perimetro = %f\n",per);
return(0);
}
```

4. Crie um projeto no codeblocks e digite o seguinte programa. Apos digitá-lo, compile e verifique seu funcionamento.

```
#include<stdio.h>

int main(void)
{
int a,b,c;
a=10;
```

```
b=3;
c=a%b;
c+=10;
c--;
printf("Resultado %d\n",c);
return(0);
}
```

AULA 6 - FUNÇÕES DE ENTRADA E SAÍDA

Apresenta as funções que fazem a leitura e apresentação dos dados

6.1 Objetivo:

Esta aula tem o objetivo de apresentar aos alunos as principais funções de entrada e saída. As funções de entrada são responsáveis pela leitura de dados formatados do teclado, ou seja, as informações digitadas pelo usuário são convertidos nos tipos apropriados de dados e armazenados nas variáveis. Por sua vez, as funções de saída são responsáveis por apresentar as informações ao operador através da tela do computador. Estas funções transformam o conteúdo das variáveis em texto de forma que as mesmas sejam compreensíveis ao operador.

6.2 A função scanf()

A função scanf() é responsável pela leitura de dados da entrada padrão, no nosso caso, o teclado. Os dados são lidos e armazenados de acordo com os parâmetros informados. O programa a seguir apresenta um exemplo de utilização a função scanf().

```
#include<stdio.h>

int main(void)
{
float x;
printf("Digite um valor: ");
scanf("%f",&x);
printf("Valor = %f\n",x);
return(0);
}
```

A função scanf() tem o seguinte formato.

scanf (“formato”,&variável);

Onde o formato corresponde a:

Formato	Tipo da informação lida
%c	Um caractere
%d	Um número inteiro
%e, %E, %f, %g, %G	Um número real (com ponto decimal)
%o	Um número no formato octal

Formato	Tipo da informação lida
%s	Uma string de caracteres
%u	Um número inteiro sem sinal
%x, %X	Um número no formato hexadecimal

A variável é onde a função deve armazenar as informações lidas. É importante lembrar que a variável deve ser declarada antes de ser utilizada na função e o tipo de dado lido deve ser o mesmo da variável.

Também é importante enfatizar que na função scanf() o nome da variável vem precedido do caractere “&”.

6.3 A função printf()

A função printf() é responsável pela gravação de dados na saída padrão, no nosso caso, a tela do computador. Os dados são gravados de acordo com os parâmetros informados.

O trecho de programa a seguir apresenta um exemplo de utilização a função printf().

```
#include<stdio.h>

int main(void)
{
char c='A';
int cont=1234;
float x=43.234;
printf("A letra %c tem o valor %d \n",c,c);
printf("O numero inteiro  %d em hexadecimal fica %X\n",cont,cont);
printf("Numero real no formato 1 %.3f e no formato 2 %e\n",x,x);
return(0);
}
```

A função printf() tem o seguinte formato.

printf (“formato”,variável);

Onde o formato corresponde a:

Formato	Tipo da informação lida
%c	Um caractere
%d	Um número inteiro

Formato	Tipo da informação lida
%e, %E, %f, %g, %G	Um número real (com ponto decimal)
%o	Um número no formato octal
%s	Uma string de caracteres
%u	Um número inteiro sem sinal
%x, %X	Um número no formato hexadecimal

A variável é de onde a função lê a informação, esta variável deve receber um valor antes que possa ser utilizada nesta função.

6.4 Exercícios

1. Utilizando o codeblocks faça um programa que lê três números reais do teclado e apresenta na tela a média destes números.
2. Crie um novo projeto e faça um programa que pede para o operador digitar uma letra, um número inteiro e um número real. Em seguida o programa deve ler estes dados, armazená-los nos tipos de dados adequados e imprimi-los na tela.
3. Faça um programa que lê 5 números inteiros e apresenta seu equivalente em formato hexadecimal.

AULA 7 - O COMANDO DE CONTROLE IF

Estudo do comando de controle "if" e de seu complemento o "else"

7.1 Objetivo:

Esta aula tem o objetivo de ensinar aos alunos como funciona o comando de controle "if" e seu complemento o "else". Desta forma os alunos serão capazes de desenvolver programas que tomem decisões, tornando assim estes programas muito mais funcionáis.

7.2 O comando "if"

O comando "if" é utilizado quando queremos que um determinado trecho de código somente seja executado se uma determinada condição for verdadeira.

A sintaxe do comando "if" é a seguinte.

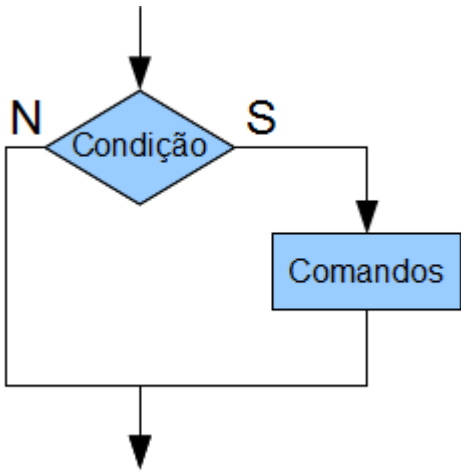
if(condição de teste) comando;

ou

if(condição de teste)

{
comando;
comando;
…
}

E seu fluxograma é:



Se utilizarmos apenas um comando para ser executado pelo "if", a utilização das chaves é desnecessária.

A seguir temos alguns exemplos de programas que utilizam o comando "if".

```
#include <stdio.h>
#include <stdlib.h>

int main()
{
int num;
```

```
printf("Digite um numero maior que 10: ");
scanf("%d",&num);
if(num>10)  printf("\nVoce digitou certo!");
printf("\nFinalizado!\n");
return 0;
}
```

Observe que a mensagem “Voce digitou certo!” só ira aparecer na tela se o número for maior que 10.

No próximo exemplo o comando “if” executa mais de um comando, e é necessário o uso de chaves.

```
#include <stdio.h>
#include <stdlib.h>

int main()
{
float num1, num2, res;
char op;
printf("\nDigite a expressao no formato: num op num: ");
scanf("%f %c %f", &num1, &op, &num2);
if( op == '+' )
    {
    res=num1+num2;
    printf(" = %.2f", res);
    }
if( op == '-' )
    {
    res=num1-num2;
    printf(" = %.2f", res);
    }
if( op == '*' )
    {
    res=num1*num2;
    printf(" = %.2f", res);
    }
if( op == '/' )
    {
    res=num1/num2;
    printf(" = %.2f", res);
    }
printf("\nFinalizado\n\n");
return 0;
}
```

7.3 O comando "if - else"

O comando "if - else" é utilizado quando queremos escolher entre dois determinados trechos de código, baseados no resultado de um teste.

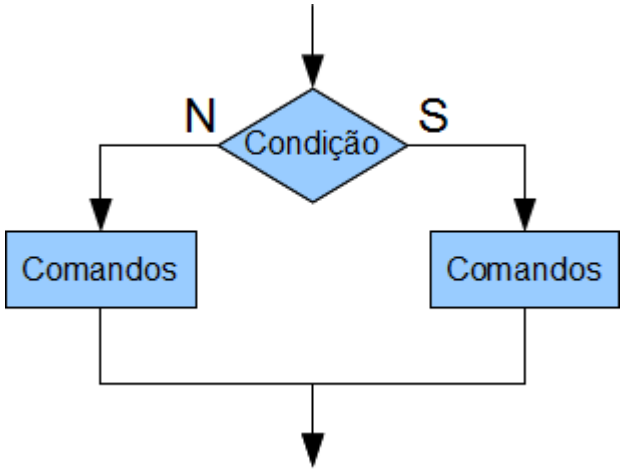
O formato do comando "if – else" é o seguinte:

if(condição de teste) comando;
else comando;

ou

```
if(condição de teste)
        {
        comando;
        comando;
        …
        }
else
        {
        comando;
        comando;
        …
        }
```

E seu fluxograma é:



Se utilizarmos apenas um comando para ser executado pelo "if" ou pelo "else", a utilização das chaves é desnecessária.

É importante salientar que o comando "else" é um complemento do comando "if", não podendo ser utilizado isoladamente.

A seguir temos um exemplo de programa utilizando o comando "if – else":

```
#include <stdio.h>
#include <stdlib.h>
#include <conio.h>

int main()
{
int magico, palpite;

srand((unsigned) time(NULL));
magico = rand()%10;
```

```
printf("\nAdivinhe o numero magico [0-9]: ");
scanf("%d", &palpite);
if( palpite == magico )
    {
    printf("Certo! ");
    printf("O numero magico eh: %d\n", magico);
    }
else
    {
    if( palpite > magico ) printf("Errado, muito alto\n");
    else printf("Errado, muito baixo\n");
    }
return 0;
}
```

No exemplo é possível observar a utilização de um comando “if” dentro de outro, quando isso ocorre costuma-se dizer que os comandos estão aninhados.

O exemplo utiliza duas novas funções, a função time(), que retorna um número inteiro contendo o número de segundos passados desde as zero horas de primeiro de janeiro de 1970 e a função srand(), que retorna um número pseudoaleatório.

7.4 Exercícios

1. Faça um programa que lê três números reis do teclado, calcula a media entre eles, imprime a média, testa se a média é menor que sete, se for imprime “Aluno em exame!” e se não for imprime “Aluno aprovado!”
2. Faça um programa que lê três números do teclado, descobre qual deles é o maior e imprime este número na tela.
3. Faça um programa que lê do teclado um número do tipo **unsigned char** (valores de 0 a 255) e apresenta na tela o valor dos oito bits que compões este número. Por exemplo, de a entrada for 12 a saída deve ser 00001100. (utilize o comando “if – else” para determinar o valor de cada bit)

AULA 8 - O COMANDO DE CONTROLE WHILE

Estudo do comando de controle "while" e suas aplicações

8.1 Objetivo:

Esta aula tem o objetivo de ensinar aos alunos como funciona o comando de controle "while". Desta forma os alunos serão capazes de desenvolver programas que executem laços de acordo com o resultado das operações de controle.

8.2 O comando "while"

O comando "while" é utilizado quando queremos repetir um comando ou um trecho de programa enquanto uma determinada condição for verdadeira.

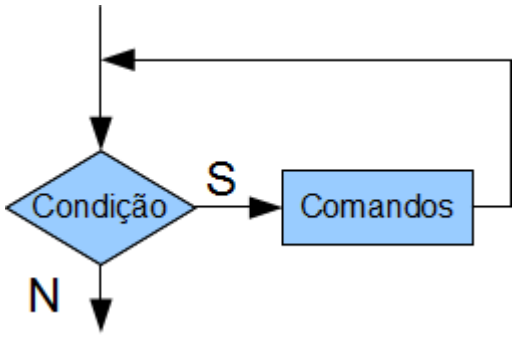
A sintaxe do comando "while" é a seguinte.

while(condição de teste) comando;

ou

while(condição de teste)
{
comando;
comando;
…
}

E seu fluxograma é:



Se utilizarmos apenas um comando para ser executado pelo "while", a utilização das chaves é desnecessária.

O funcionamento do comando "while" é bastante simples, o trecho de programa em seu interior é repetido enquanto a condição de teste for verdadeira.

A seguir temos alguns exemplos de programas que utilizam o comando "while".

```
#include <stdio.h>
#include <stdlib.h>

int main()
{
float num1, num2, res;
```

```
char op;
char controle=0;
while(controle!='s')
    {
    printf("\nDigite a expressao no formato: num op num: ");
    scanf("%f %c %f", &num1, &op, &num2);
    if( op == '+' )
        {
        res=num1+num2;
        printf(" = %.2f", res);
        }
    if( op == '-' )
        {
        res=num1-num2;
        printf(" = %.2f", res);
        }
    if( op == '*' )
        {
        res=num1*num2;
        printf(" = %.2f", res);
        }
    if( op == '/' )
        {
        res=num1/num2;
        printf(" = %.2f", res);
        }
    printf("\nDigite s para sair ou qualquer outra tecla para continuar\n");
    fflush(stdin); //limpa a memória de entrada
    scanf("%c",&controle);
    }
printf("\nFinalizado\n\n");
return 0;
}
```

O exemplo utiliza o comando “while” para repetir o programa enquanto o operador não escolher a opção sair.

Observe que o exemplo utiliza o comando fflush(stdin), para limpar a memória da entrada de dados. Esta operação é necessária pois as operações anteriores deixam um caractere na memória de entrada. A seguir temos outro exemplo da utilização do momando “while”.

```
#include <stdio.h>
#include <stdlib.h>
#include <conio.h>

int main()
{
```

```
int magico, palpite=0;

srand((unsigned) time(NULL));
magico = rand()%10;
while(palpite>=0&&palpite<=9)
   {
   printf("\nAdivinhe o numero magico [0-9] \ndigite qualquer outro numero para sair\n");
   scanf("%d", &palpite);
   if(palpite>=0&&palpite<=9)
      {
      if( palpite == magico )
         {
         printf("Certo! ");
         printf("O numero magico eh: %d\n", magico);
         break; //O comando break; finaliza o while imediatamente
         }
      else
         {
         if( palpite > magico ) printf("Errado, muito alto\n");
         else printf("Errado, muito baixo\n");
         }
      }
   }
return 0;
}
```

O exemplo utiliza o comando “while”, testando duas condições ao mesmo tempo, isto é feito utilizando a seguinte notação **while(condição1 && condição2).** O comando “&&” retorna uma condição verdadeira se as duas condições forem verdadeiras ao mesmo tempo. Também podemos utilizar o comando “ | | ”, se desejarmos que uma ou outra condição seja verdadeira, ou seja, se apenas uma das condições for verdadeira o resultado já é verdadeiro. Estes comandos podem ser utilizados com mais de duas condições e mesclados a vontade.

Também é utilizado neste exemplo o comando “break”. O comando “break” serve para terminar a execução do laço “while” instantaneamente.

```
#include <stdio.h>
#include <stdlib.h>

int main()
{
int cont=0;
while(cont<10)
   {
   printf("\nValor de cont = %d",cont);
```

```
    cont++;
    }
printf("\n\nFinalizado\n\n");
return 0;
}
```

Este ultimo exemplo utiliza uma variável interna do programa para controlar a saída do laço “while”, sem a necessidade da intervenção do operador.

8.3 Exercícios

1. Faça um programa utilizando o comando “while”, que fica lendo números inteiros do teclado e só finaliza quando o número digitado for 10.
2. Faça um programa que lê dois números inteiros do teclado, testa se o primeiro é menor que o segundo, se for imprime todos os números do intervalo entre eles.
3. Faça um programa que lê um número real do teclado e utilizando o comando “while” calcula o fatorial deste número. (ex: 5! é igual a 5 * 4 * 3 * 2 * 1).
4. Faça um programa que lê um número do teclado, e utilizando dois comandos “while” apresenta na tela a seguinte matriz, onde n é o número lido e cada elemento da matriz é o resultado da operação indicada.

N+1 N+2 N+3 N+4
N+2 N+3 N+4 N+5
N+3 N+4 N+5 N+6
N+4 N+5 N+6 N+7

Exemplo: para n = 10 temos:
11 12 13 14
12 13 14 15
13 14 15 16
14 15 16 17

AULA 9 - LISTA DE EXERCÍCIOS

Lista de exercícios sobre os assuntos das ultimas aulas

9.1 Objetivo:

O objetivo desta aula é revisar o conteúdo das aulas anteriores de forma a integrar os assuntos abordados e sanar qualquer dúvida que ainda permaneça.

9.1.1 Exercícios

1. Descreva qual a função de cada um dos seguintes elementos em um computador.

    a) ULA (Unidade Lógica Aritmética)

    b) Registradores

    c) Memória

    d) Unidade de controle

2. Faça o fluxograma de um programa que fica lendo um número do teclado enquanto o número for diferente de 10.
3. Quais as regras básicas para a criação de nomes de identificadores na linguagem C?
4. Quais são as duas formas de fazer um comentário nos programas em C?
5. Qual o limite máximo e mínimo para cada um dos tipos de dados a seguir?

    a) unsigned char

    b) float

    c) unsigned int

    d) int

6. Escolha cinco operadores da linguagem C, descreva seu funcionamento e de um exemplo de cada um deles.
7. Crie uma pasta na área de trabalho chamada “exercicios”, em seguida crie um projeto chamado “exercicio7” e neste projeto faça um programa que imprime a mensagem “Alo Mundo”.

8. Faça um programa que pede ao operador para digitar a temperatura ambiente e utilizando comandos “if” ou “if-else”, faz o seguinte. Se a temperatura for maior ou igual a 40, imprime “Muito quente”, se a temperatura menor que 40 e maior ou igual a 30, imprime “Quente”, se a temperatura for menor que 30 e maior ou igual a 20, imprime “Agradável” e se a temperatura for menor que 20, imprime “Frio”.
9. Utilizando o comando “while” faça um programa que calcula o valor de x elevado a y, onde x é um número real lido do teclado e y é um número inteiro lido do teclado.
10. Faça um programa que apresenta na tela toda a tabela ASCII, ou seja, apresenta a tela todos os caracteres cujos valores estão entre 0 e 255.
11. Utilizando comandos “while”, escreva um programa que exibe as tabuadas de multiplicação dos números de 1 à 9 .
12. Escreva um programa que imprima todos os números pares entre 0 e 50 e em seguida imprima todos os números impares. Deixe um espaço entre os números.
13. Escreva um programa que lê 10 números reais. O programa deve imprimir a média, o maior e o menor dos números.
14. Crie um programa para verificar se um número dado é primo. Utilize apenas números inteiros.
15. Escreva um programa que lê um número inteiro do teclado e imprime todos os seus divisores,

AULA 10 - O COMANDO DE CONTROLE FOR

Estudo do comando de controle "for" e suas aplicações

10.1 Objetivo:

Esta aula tem o objetivo de ensinar aos alunos como funciona o comando de controle "for". Desta forma os alunos serão capazes de desenvolver programas que executem laços controlados por variáveis possibilitando assim uma infinidade de aplicações, como por exemplo operações com matrizes.

10.2 O comando "for"

O comando "for" é utilizado quando queremos repetir um trecho de código um determinado número de vezes.

A sintaxe do comando "for" é a seguinte.

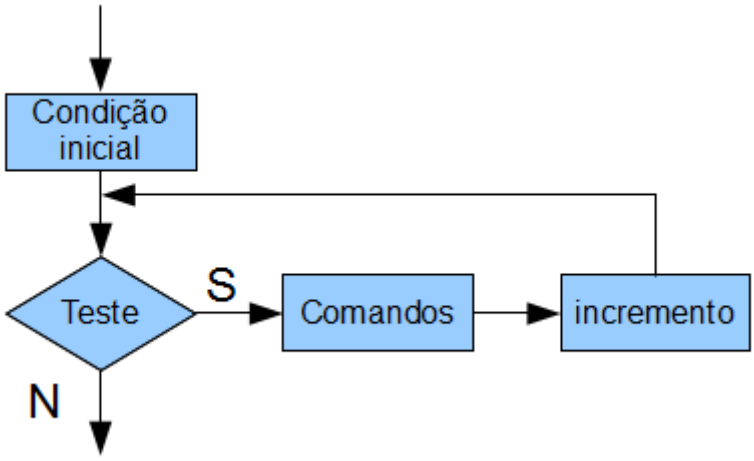
for(condição inicial; condição de teste; incremento) comando;

ou

for(condição inicial; condição de teste; incremento)

{
comando;
comando;
…
}

E seu fluxograma é:



Se utilizarmos apenas um comando no interior do "for", a utilização das chaves é desnecessária.

O funcionamento do comando "for" é o seguinte. O "for" necessita sempre de uma variável de controle, esta variável é atualizada a cada interação do laço e determina quando o laço deve

terminar. O “for” recebe três parâmetros, o primeiro é o valor inicial da variável de controle, o segundo é a condição que deve ser verdadeira para que o laço continue sendo executado, e o terceiro é o incremento da variável de controle.

A seguir temos alguns exemplos de programas que utilizam o comando “for”.

```
#include <stdio.h>
#include <stdlib.h>

int main()
{
int cont;

for(cont=0; cont<=10; cont++)
    {
    printf("%d\n",cont);
    }
return 0;
}
```

O exemplo utiliza o comando “for”, com uma variável de controle chamada “cont”. A variável “cont” é iniciada com o valor 0 e o laço fica se repetindo enquanto a variável for menor ou igual a 10. A cada interação do laço a variável é incrementada em 1.

O próximo exemplo utiliza o comando “for” para calcular o fatorial de um número.

```
#include<stdio.h>
#include<stdlib.h>

int main()
{
int numero,fat=1,i;
printf("\nEntre com um numero positivo: ");
scanf("%d",&numero);
for(i=numero; i>1; i--)
    {
    fat=fat*i;
    }
printf("O fatorial de %u vale %u.\n",numero,fat);
return 0;
}
```

O exemplo a seguir incrementa a variável de controle com um valor diferente de 1.

```
#include <stdio.h>

main()
{
```

```
int cont;
for(cont = 0; cont < 10; cont=cont+3)
    {
    printf("cont = %d\n", cont);
    }
return(0);
}
```

O comando “for” é extremamente flexível na linguagem C, o exemplo a seguir utiliza duas variáveis de controle ao mesmo tempo.

```
#include <stdio.h>

main()
{
int x, y;
for(x = 0, y = 0; x+y < 20; x++, y++)
    {
    printf("%d\n", x+y);
    }
return(0);
}
```

Qualquer tipo de operação lógica ou matemática pode ser utilizada no comando “for”, assim é possível criar laços cuja sequencia de interações é extremamente complexa, resolvendo inúmeros problemas computacionais.

10.3 Exercícios

1. Faça um programa utilizando o comando “for” que apresenta os números de 5 a 20.
2. Modifique o programa anterior para que a sequencia apresentada seja de 20 a 5.
3. Faça um programa utilizando o comando “for” que apresenta as tabuadas de 1 a 10.
4. Utilizando os comandos “for” e “if”, faça um programa que lê 5 valores do teclado e informa quantos deles são negativos e quantos são positivos.
5. Escrever um programa que lê um valor i inteiro e positivo e que calcula e escreve o valor da constante de Euler (e), com aproximação de i termos. A formula a seguir calcula o valor de e, e o resultado deve ser 2,718282. O programa deve ficar se repetindo enquanto o operador não escolher i =0;

$$e = \sum_{i=0}^{\infty} \frac{1}{i!}$$

6. Escrever um programa que calcula e escreve a soma dos números primos entre 92 e 1478.

AULA 11 - OUTROS COMANDOS DE CONTROLE

Estudo do comando “do – while” e “switch”

11.1 Objetivo:

A linguagem C possui um grande número de comandos de controle, as aulas anteriores estudaram alguns deles, porem ainda existem outros. Esta aula irá estudas mais dois comandos de controle, o comando “do – while” e o comando “switch”.

11.2 O comando “do – while”

O comando “do – while” é utilizado quando queremos repetir um trecho de código enquanto uma condição for verdadeira, porem o teste da condição só é realizado após a execução do trecho de código. Isso faz com que independente da condição ser falsa ou verdadeira o trecho de código é executado pelo menos uma vez. Se compararmos co comando “do – while” com o comando “while” veremos que a única diferença esta no momento em que o teste da condição é realizado, no “while” o teste é realizado antes de executar o trecho de código, e no “do – while” depois de executar o trecho de código.

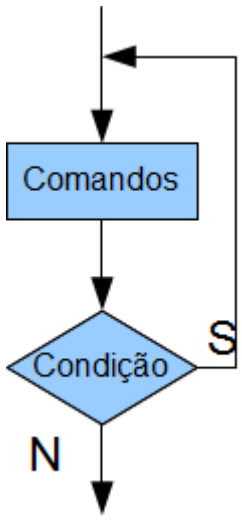
A sintaxe do comando “do - while” é a seguinte.

```
do
comandos;
while(condição de teste);
```

ou

```
do
        {
        comando;
        comando;
        …
        }
while(condição de teste);
```

E seu fluxograma é:



Se utilizarmos apenas um comando no interior do “do - while”, a utilização das chaves é desnecessária. A seguir temos um exemplo de programa que utiliza o comando “do – while”.

```
#include <stdio.h>
#include <stdlib.h>
int main()
{
int n;
printf("digite um numero: ");
scanf("%d",&n);
do
        {
        printf("->%d\n",n);
        n--;
        }
while(n>=10);
return 0;
}
```

Neste exemplo, se o operador digitar um número menor que 10, mesmo a condição sendo falsa os comandos no interior do "do – while" serão executados uma vez.

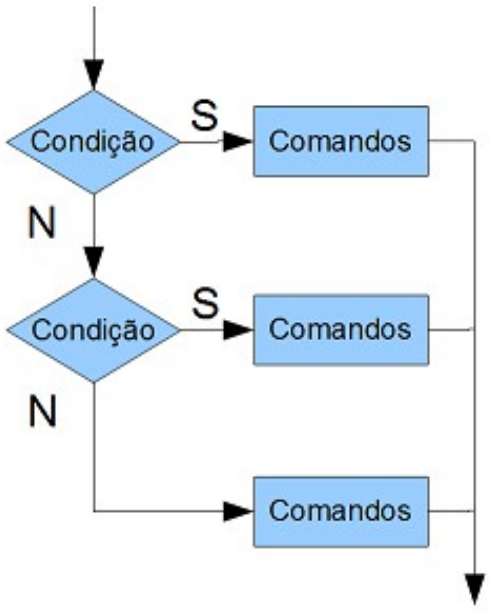
O comando de desvio "break" também pode ser utilizado com o comando "do – while".

11.3 O comando "switch"

O comando "switch" é utilizado quando queremos escolher uma entre várias opções. Este comando recebe como parâmetro uma expressão cujo resultado é um número inteiro ou um caractere, e este número é comparado com um conjunto de constantes. Se o valor for igual a constante os comandos que seguem aquela constante são executados até que se encontre o comando "break". Se nenhuma das opções é a correta a entrada "default" é executada. A sua sintaxe é a seguinte.

```
switch (expressão)
        {
        case constante1:
                comandos;
                break;
        case constante2:
                comandos;
                break;
        ...
        default:
                comandos;
        }
```

E seu fluxograma é:



A seguir temos um exemplo de programa que utiliza o comando “switch”.

```
#include <stdio.h>
#include <stdlib.h>

int main()
{
int n;
printf("digite um numero de 0 a 9: ");
scanf("%d",&n);
switch (n)
        {
        case 0:
        printf("zero\n");
        break;
        case 1:
        printf("um\n");
        break;
        case 2:
        printf("dois\n");
        break;
        case 3:
        printf("tres\n");
        break;
        case 4:
        printf("quatro\n");
        break;
        case 5:
        printf("cinco\n");
        break;
        case 6:
        printf("seis\n");
        break;
        case 7:
        printf("sete\n");
        break;
        case 8:
        printf("oito\n");
        break;
        case 9:
        printf("nove\n");
        break;
        default:
        printf("numero invalido!\n");
        }
return 0;
}
```

Podemos utilizar mais de um comando no interior do “switch” sem a utilização das chaves.

11.4 Exercícios

1. Faça um programa que lê um número inteiro de 1 a 7 e imprime o dia da semana, se o número estiver fora deste intervalo o programa deve imprimir uma mensagem de erro.
2. Faça um programa utilizando o comando “switch”. O programa deve se comportar como uma calculadora, onde o operador escolhe entre as operações soma (+), subtração (-), multiplicação (*), divisão (/), potência (^), e raiz quadrada (r). O programa deve apresentar todas as opções no inicio e o operador escolhe a operação desejada digitando o simbolo entre parenteses. O programa deve então ler os dois números, realizar a operação e reiniciar o processo até que o operador escolha uma operação inválida (qualquer outro símbolo ou letra).
3. Faça um programa utilizando o comando “do – while”, este programa deve ficar lendo uma letra por vez do teclado até que o operador digite a letra x. Após o operador digitar x o programa deve apresentar na tela o número de vezes que o operador digitou dois caracteres iguais em sequencia.
4. Faça um programa que apresente quatro opções: (a) consulta saldo, (b) saque, (c) depósito e (d) sair. O saldo deve iniciar em R$ 0,00. A cada saque ou depósito o valor do saldo deve ser atualizado e apresentado na tela.

AULA 12 - CARACTERES, VETORES E MATRIZES

Estudo das cadeias de caracteres, dos vetores e das matrizes

12.1 Objetivo:

Na programação, uma cadeia de caracteres, também chamada de string é uma sequência ordenada de símbolos. Cada símbolo armazenado na memória é representado por um valor numérico. Uma variável declarada com tipo de dado "char" é utilizada para armazenar as cadeias de caracteres. Da mesma forma um vetor é um conjunto ordenado de números, armazenados na memória do computados, estes números podem ser inteiros ou reais. Se tivermos um vetor com mais de uma dimensão estamos trabalhando com matrizes. A linguagem C fornece suporte aos vetores e as matrizes.

12.2 Cadeias de caracteres

É muito comum nos programas e computador que se trabalhe com palavras, frases e textos. A linguagem C fornece suporte as cadeias de caracteres através de vetores de letras, ou seja, em C uma palavra é um vetor que contém um grupo de letras. O tipo de dados utilizado para armazenar letras é o tipo "char". Para que a variável possa armazenar mais de uma letra devemos adicionar o número de letras desejado na declaração da variável. O número deve estar entre colchetes. A seguir temos alguns exemplos de variáveis que armazenam cadeias de caracteres.

```
char nome[30]; //cria a variável nome com capacidade de 30 letras.
char endereco[100]; //cria a variável endereco com capacidade de 100 letras.
```

As letras são armazenadas na memória, cada uma utilizando 1 byte. Cada letra é armazenada como um número inteiro, conforme a tabela **ascii**. A sequencia de caracteres é sempre terminada com um número zero para indicar ao programa o fim da sequencia.

A seguir é apresentado como a palavra teste é armazenada na memória em uma variável com capacidade de 10 letras.

t	e	s	t	e					
116	101	115	116	101	0				

É importante observar o valor zero finalizando a cadeia de caracteres.

A tabela a seguir apresenta os valores para todos os caracteres imprimíveis. Os caracteres de 0 a 31 são caracteres de controle e não são imprimíveis.

É importante salientar que se tentarmos escrever mais caracteres do que a capacidade da variável, o programa ira executar operações ilegais, podendo travar ou fechar.

12.3 Leitura e impressão de cadeias de caracteres

A leitura de cadeias e caracteres é realizada através do comando “scanf”, com o uso do operador “%s”. Veja o exemplo a seguir.

```
scanf("%s",&nome);
```

O comando “scanf” encera a leitura ao encontrar um espaço ou um retorno de carro, isso é inconveniente, pois se tentarmos ler do teclado uma frase contendo espaços ela ficará incompleta. Para resolver este problema temos o comando “gets”, cuja função é unicamente ler uma cadeia de caracteres do teclado. Sua utilização é muito simples, basta fornecer o nome da variável onde as letras devem ser armazenadas. Veja o exemplo a seguir.

```
gets(nome);
```

Da mesma forma, a impressão de cadeias de caracteres é feita através do comando “printf”, com o uso do operador “%s”. Veja o exemplo a seguir.

```
printf("Senhor %s\n",nome);
```

A seguir temos um exemplo exemplo de programa que lê uma frase do teclado e imprime a frase na tela.

```
#include <stdio.h>
#include <stdlib.h>

int main()
{
char nome[50];
gets(nome);
printf("%s\n",nome);
return 0;
}
```

Também é possível ler ou escrever uma letra de cada vez com os comandos “getchar” e “putchar” veja o exemplo a seguir.

```
#include <stdio.h>
#include <stdlib.h>

int main()
{
char L;
L=getchar(); //le uma letra e armazena na variavel L
putchar(L); //imprime a letra armazenada em L
return 0;
}
```

12.4 Manipulação de cadeias de caracteres

Existem alguns comandos úteis para manipular cadeias de caracteres. O primeiro é o “strcpy”, este comando copia uma cadeia de caracteres em uma variável. A seguir temos um exemplo.

```
char nome[50];
char txt[50];
strcpy(nome,"teste de texto"); // armazena a cadeia de caracteres na variável nome
strcpy(txt,nome); // armazena o conteúdo da variável nome na variável txt
```

Outra função importante na manipulação de arquivos é a função “strcmp”, que é usada para comparar duas cadeias de caracteres. A função “strcmp” recebe duas sequencias de caracteres e retorna 0 se as duas forem iguais. Se as duas cadeias de caracteres forem diferentes a função retorna um valor diferente de 0.

A seguir temos um exemplo de programa que utiliza o comando “strcmp”.

```
#include <stdio.h>
#include <stdlib.h>

int main()
{
char txt[50];
printf("digite uma palavra; ");
gets(txt);
if(strcmp(txt,"palavra")==0)
    {
    printf("\nAcertou\n");
    }
else
    {
    printf("\nErrou\n");
    }
return 0;
}
```

Se a palavra digitada for “palavra” o programa imprime “Acertou” senão o programa imprime “Errou”.

Também é possível acessar uma única letra dentro de uma cadeia de caracteres, basta colocar número da posição da letra dentro da sequencia de caracteres, conforme o exemplo a seguir.

```
#include <stdio.h>
#include <stdlib.h>

int main()
{
char txt[50];
char letra;
printf("digite uma palavra; ");
gets(txt);
letra=txt[3]; // letra recebe a quarta letra da variável txt
printf("\n%c",letra);
return 0;
}
```

É importante salientar que a contagem das letras inicia de zero.

12.5 Vetores

Os programas de computador tem uma grande facilidade de trabalhar com números. A linguagem C tem a capacidade de trabalhar com vetores. Assim como acontece com os caracteres, para declarar uma variável como vetor devemos colocar entre colchetes o número de elementos do

vetor, conforme os exemplos a seguir.

```
int x[100]; // cria um vetor com 100 números inteiros
float vet[10]; // cria um vetor com 10 números reais
```

Para operar os valores no interior de um vetor, basta referenciar o elemento do vetor através do número entre colchetes. Veja um exemplo.

```
#include <stdio.h>
#include <stdlib.h>

int main()
{
int vet[10];
vet[0]=123;
printf("Digite um numero: ");
scanf("%d",&vet[1]);
printf("item 0 = %d\n",vet[0]);
printf("item 1 = %d\n",vet[1]);
return 0;
}
```

12.6 Matrizes

Vetores são casos particulares de matrizes, onde temos apenas uma dimensão. Em C podemos criar matrizes com qualquer número de dimensões, para facilitar nosso estudo, utilizaremos apenas matrizes com duas dimensões, porem é muito simples extrapolar este conhecimento para matrizes com mais dimensões.

Para declararmos uma variável como uma matriz basta apenas adicionar dois pares de colchetes com as dimensões da matriz após a declaração da variável. Veja um exemplo.

```
#include <stdio.h>
#include <stdlib.h>

int main()
{
int mat[3][3]; // declara uma matriz de 3 x 3 números inteiros
int cont_x, cont_y;

printf("Digite os 9 elementos da matriz: ");
scanf("%d",&mat[0][0]);
scanf("%d",&mat[0][1]);
scanf("%d",&mat[0][2]);
scanf("%d",&mat[1][0]);
scanf("%d",&mat[1][1]);
```

```
scanf("%d",&mat[1][2]);
scanf("%d",&mat[2][0]);
scanf("%d",&mat[2][1]);
scanf("%d",&mat[2][2]);

printf("\nA matriz ficou:\n\n");

for(cont_x=0; cont_x<=2; cont_x++)
    {
    for(cont_y=0; cont_y<=2; cont_y++)
        {
        printf("%d ",mat[cont_x][cont_y]);
        }
    printf("\n");
    }
return 0;
}
```

O programa do exemplo acessa cada elemento da matriz através das variáveis cont_x e cont_y.

12.7 Exercícios

1. Faça um programa que lê três palavras do teclado e imprime as três palavras na ordem inversa.
2. Faça um programa que lê duas palavras do teclado e diz se elas são iguais ou diferentes. O programa deve dizer ainda se alguma das palavras digitadas é igual a “papagaio”.
3. Faça um programa que cria um vetor com 5 elementos inteiros, lê 5 números do teclado, armazena os números no vetor e imprime o vetor na ordem inversa.
4. Faça um programa que lê uma matriz de 3 x 3 elementos usando um comando for, multiplica cada elemento por 5 e imprime o resultado.
5. Faça um programa que lê um vetor de 3 elementos e uma matriz de 3 x 3 elementos. Em seguida o programa deve fazer a multiplicação do vetor pelas colunas da matriz.

AULA 13 - SALVANDO E LENDO DADOS EM ARQUIVOS

Estudo das funções de leitura e escrita de arquivos no disco

13.1 Objetivo:

Há muitas situações nas quais precisamos ler ou gravar dados em arquivos a partir de nossos programas C, pois qualquer informação que não é armazenada em arquivos é perdida quando o programa é finalizado. A linguagem C fornece um conjunto de ferramentas para executar esta tarefa, a seguir veremos algumas delas.

13.2 Abertura de Arquivos

Para que possamos ter acesso a um arquivo no computador é necessário inicialmente abrir ou criar este arquivo. O primeiro passo para esta tarefa é aprender a usar a função fopen(). Esta função é a responsável por conectar um ponteiro de arquivos ao arquivo que queremos manipular.

A sintaxe da função “fopen” é a seguinte.

FILE *ponteiro_arquivo=fopen(const char *nome, const char *modo);

O parâmetro nome é uma cadeia de caracteres que indica o nome do arquivo a ser aberto ou criado. O parâmetro modo indica se o arquivo será aberto para leitura, escrita ou ambos, veja na tabela a seguir os modos possíveis.

Modo	Significado
"r"	Abre um arquivo texto para leitura. O arquivo deve existir antes de ser aberto.
"w"	Abrir um arquivo texto para gravação. Se o arquivo não existir, ele será criado. Se já existir, o conteúdo anterior será destruído.
"a"	Abrir um arquivo texto para gravação. Os dados serão adicionados no fim do arquivo ("append"), se ele já existir, ou um novo arquivo será criado, no caso de arquivo não existente anteriormente.
"rb"	Abre um arquivo binário para leitura. Igual ao modo "r" anterior, só que o arquivo é binário.
"wb"	Cria um arquivo binário para escrita, como no modo "w" anterior, só que o arquivo é binário.
"ab"	Acrescenta dados binários no fim do arquivo, como no modo "a" anterior, só que o arquivo é binário.
"r+"	Abre um arquivo texto para leitura e gravação. O arquivo deve existir e pode ser modificado.
"w+"	Cria um arquivo texto para leitura e gravação. Se o arquivo existir, o conteúdo anterior será destruído. Se não existir, será criado.
"a+"	Abre um arquivo texto para gravação e leitura. Os dados serão adicionados no fim do

	arquivo se ele já existir, ou um novo arquivo será criado, no caso de arquivo não existente anteriormente.
"r+b"	Abre um arquivo binário para leitura e escrita. O mesmo que "r+" acima, só que o arquivo é binário.
"w+b"	Cria um arquivo binário para leitura e escrita. O mesmo que "w+" acima, só que o arquivo é binário.
"a+b"	Acrescenta dados ou cria uma arquivo binário para leitura e escrita. O mesmo que "a+" acima, só que o arquivo é binário

A seguir temos um exemplo onde é criado um arquivo chamado “teste.txt” e em seu interior é gravada a frase “escrevendo em um arquivo”.

```
#include <stdio.h>
#include <stdlib.h>

int main()
{
FILE *arquivo = fopen("teste.txt", "w"); // cria ou abre o arquivo

if(arquivo == NULL) // testa se o arquivo foi aberto com sucesso
    {
    printf("\n\nImpossivel abrir o arquivo!\n\n");
    return 0;
    }
fprintf(arquivo,"escrevendo em um arquivo\n");
fclose(arquivo);
printf("Concluido!\n\n");
return 0;
}
```

Este exemplo testa se a a variável arquivo é igual a “NULL”, se for significa que ouve algum erro em abrir o arquivo e o programa é finalizado apresentando uma mensagem de erro.

Todos os arquivos que são abertos devem ser fechados utilizando o comando “fclose” e passando como parâmetro o nome da variável que contem o ponteiro do arquivo. A seguir veremos como gravar e como ler informações de um arquivo.

13.3 Gravando Informações em um Arquivo

Existem várias formas de gravar informações em um arquivo, a mais fácil delas é utilizar o comando “fprintf”, que é muito similar ao comado “printf”, com a única diferença de receber a variável que contem o ponteiro do arquivo como parâmetro. Veja o exemplo anterior e o exemplo a seguir.

```
#include <stdio.h>
#include <stdlib.h>

int main()
{
int cont;
FILE *arq = fopen("C:\\Users\\Ricardo\\Desktop\\teste.txt", "w"); // cria ou abre o arquivo

if(arq == NULL) // testa se o arquivo foi aberto com sucesso
    {
    printf("\n\nImpossivel abrir o arquivo!\n\n");
    return 0;
    }
cont=0;
fprintf(arq,"Agora cont vale %d\n",cont);
cont++;
fprintf(arq,"Agora cont vale %d\n",cont);
cont++;
fprintf(arq,"Agora cont vale %d\n",cont);
cont++;
fprintf(arq,"Agora cont vale %d\n",cont);
fclose(arq);
printf("Concluido!\n\n");
return 0;
}
```

É Importante observar que as barras que separam os nomes de arquivos e pastas devem ser duplos na linguagem C.

13.4 Lendo Informações de um Arquivo

Assim como existem comando para gravar nos arquivos existem também comandos para ler os arquivos. O principal deles é o comando “fscanf”. Este comando é muito similar ao comado “scanf”, com a única diferença de receber a variável que contem o ponteiro do arquivo como parâmetro. Veja o exemplo a seguir.

```
#include <stdio.h>
#include <stdlib.h>

int main()
{
int x;
FILE *arq = fopen("C:\\Users\\Ricardo\\Desktop\\numero.txt", "r");

if(arq == NULL) // testa se o arquivo foi aberto com sucesso
    {
```

```
   printf("\n\nImpossivel abrir o arquivo!\n\n");
   return 0;
   }
fscanf(arq,"%d",&x);
fclose(arq);
printf("Valor lido = %d\n\n",x);
return 0;
}
```

É importante observar que o comando “fscanf” lê apenas uma palavra de cada vez, se desejarmos ler uma frase completa devemos usar o comando “fgets”. Este comando recebe três parâmetros, o primeiro é a variável onde a frase deve ser gravada, o segundo é o número máxixo de letras que podem ser lidas e o terceiro é a variável que contem o ponteiro do arquivo. Veja o exemplo a seguir.

```
#include <stdio.h>
#include <stdlib.h>

int main()
{
char x[100];

FILE *arq = fopen("C:\\Users\\Ricardo\\Desktop\\vet.txt", "r");

if(arq == NULL) // testa se o arquivo foi aberto com sucesso
   {
   printf("\n\nImpossivel abrir o arquivo!\n\n");
   return 0;
   }

fgets(x,100,arq);
fclose(arq);
printf("\nfrase lida = %s\n\n",x);
return 0;
}
```

Existem ainda muitos outros comandos para manipular arquivos, porem estes que foram vistos são suficientes para nossos estudos. Contudo se for necessário construir programas que trabalhem com arquivos mais complexos é necessário utilizar outros comandos mais poderosos. Uma pesquisa na internet fornece mais informações nestes casos.

A seguir temos mais dois exemplos, onde um vetor de 5 números reais é gravado em um arquivo e em seguida lido novamente.

O primeiro grava os valores do vetor no arquivo.

```
#include <stdio.h>
#include <stdlib.h>

int main()
{
float vet[5];
int cont;
FILE *arq = fopen("C:\\Users\\Ricardo\\Desktop\\vet.txt", "w+");

vet[0]=1000.0;
vet[1]=2000.0;
vet[2]=3000.0;
vet[3]=4000.0;
vet[4]=5000.0;

if(arq == NULL) // testa se o arquivo foi aberto com sucesso
    {
    printf("\n\nImpossivel abrir o arquivo!\n\n");
    return 0;
    }

for(cont=0; cont<5; cont++)
    {
    fprintf(arq,"%f\n",vet[cont]);
    }
fclose(arq);
printf("concluido\n\n");
return 0;
}
```

E o segundo exemplo lê os valores do arquivo e os apresenta na tela.

```
#include <stdio.h>
#include <stdlib.h>

int main()
{
float vet[5];
int cont;
FILE *arq = fopen("C:\\Users\\Ricardo\\Desktop\\vet.txt", "r");

if(arq == NULL) // testa se o arquivo foi aberto com sucesso
    {
    printf("\n\nImpossivel abrir o arquivo!\n\n");
    return 0;
    }

for(cont=0; cont<5; cont++)
    {
    fscanf(arq,"%f",&vet[cont]);
```

```
    printf("%f\n",vet[cont]);
    }
fclose(arq);
printf("\nconcluido\n\n");
return 0;
}
```

É importante salientar que para escrever nos arquivos usamos na função “fopen” o parametro “w+”, e para ler usamos o parametro “r”.

13.5 exercícios

1. Faça um programa que grava uma palavra em um arquivo chamado “palavra.txt”, a palavra deve ser digitada pelo operador.
2. Faça um programa que lê a palavra do arquivo do exercício anterior e a apresenta na tela.
3. Faça um programa que grava uma matriz de 3 x 3 números inteiros em um arquivo chamado “mat.txt”, os números devem ser digitados pelo operador.
4. Faça um programa que lê a matriz do arquivo do exercício anterior e a apresenta na tela.
5.

AULA 14 - LISTA DE EXERCÍCIOS

Lista de exercícios sobre os assuntos das ultimas aulas

14.1 Objetivo:

O objetivo desta aula é revisar o conteúdo das aulas anteriores de forma a integrar os assuntos abordados e sanar qualquer dúvida que ainda permaneça.

14.2 Exercícios

1. Faça um programa utilizando o comando "for" que lê 10 números do teclado e armazena estes 10 números em um arquivo chamado "num.txt" na área de trabalho do computados.
2. Faça um programa que lê uma frase do teclado e indica na tela quantas vezes a letra "s" aparece na frase.
3. Faça um programa que lê 20 palavras de um arquivo e verifica se a palavra "casa" esta entre elas.
4. Faça um programa que lê 5 notas de um arquivo chamado "notas.txt", calcula a média destas notas, imprime a média destas notas e salva a média no final deste mesmo arquivo.
5. Faça um programa utilizando o comando "switch-case" que pede para o operado escolher entre três opções: (S) para saír, (L) para ler a matriz e (G) para gravar a matriz. Se o operador escolher a opção S o programa é finalizado. Se o operador escolher a opção L o programa deve pedir ao operador qual o nome do arquivo e em seguida ler a matriz deste arquivo e apresentar a matriz na tela. E se o operador escolher escolher a opção G o programa deve pedir para o operador digitar o nome do arquivo, o número de linhas e o número de colunas da matriz, e em seguida deve pedir para o operador digitar os números que compõem a matriz. A matriz deve ser salva no arquivo especificado. Não esqueça de incluir no arquivo o número de linhas e colunas da matriz para que a leitura dos dados seja possível.

REFERENCIAS BIBLIOGRÁFICAS

- SCHILDT, H. **C Completo e Total**, Makron Books, 3ª Ed, 1997
- Paulo Sérgio de Moraes, **Curso Básico de Lógica de Programação,** Unicamp - Centro de Computação - DSC 2000
- MONTEIRO, M. A. **Introdução à Organização de Computadores.** Rio de Janeiro: Editora LTC, 2007. 5ª Edição.