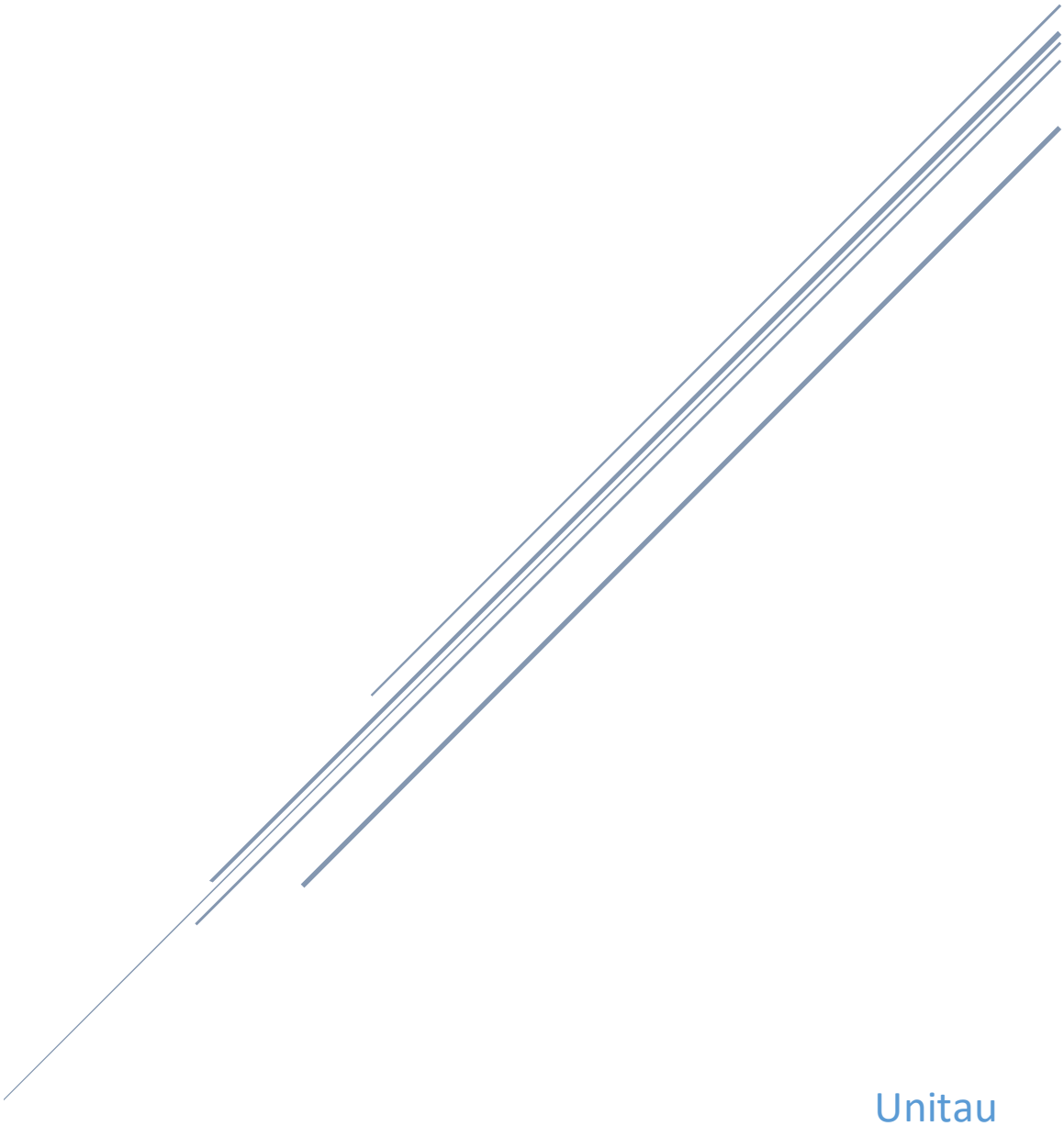
MATERIAL DE APOIO – MODULO 1

Seven Online Jugde



Unitau
Engenharia de Computação

Lista de figuras

Lista de tabelas

Sumário

Resumo

Este material visa auxiliar o aluno no aprendizado de programação assistido por computador. Neste modulo é apresentado ao estudante, algoritmos, tomadas de decisões através de fluxogramas, if/else, laços for e while e switch case.

1 Introdução

Nesta apostila, no capítulo 2, você terá acesso a história do computador para compreender o funcionamento do mesmo e sua origem.

No capítulo 3, será apresentado algoritmos, a fim de que o aluno aprenda sequencias lógicas, as quais são essenciais para o aprendizado de programação.

A partir do capítulo 4, é abordado o uso da linguagem de programação C, incluindo seus códigos, palavras-chaves, palavras reservadas pela linguagem, operadores aritméticos e lógicos, definições de variáveis e os seus respectivos tipos.

No capítulo 5 é mostrado a tomada de decisão em programação, mostrando a utilização do if/else (se/senão).

O capítulo 6 apresenta as estruturas de repetição que a linguagem utiliza, sendo elas os loops: for, while e do while.

Por fim, no capítulo 7 apresentamos o switch case, uma estrutura semelhante ao if/else que também é muito utilizada em linguagem C.

2 O Computador

O computador é uma máquina eletrônica que permite processar dados (Conceito .de, 2012). O termo provém do latim *computare* (“calcular”).

Os computadores podem ser mecânicos (computador analógico) ou eletrônicos (computadores digitais).

2.1 Histórico dos Computadores

2.1.1 Ábaco, a primeira calculadora da História

O ábaco é um instrumento mecânico usado para contar; não é uma calculadora no sentido da palavra que hoje usamos (Ryerson, 2004).

Seu primeiro registro é datado do ano de 5.500 a.C., pelos povos que constituíam a Mesopotâmia. Contudo, o ábaco também foi usado posteriormente por muitas outras culturas: Babilônia, Egito, Grécia, Roma, Índia, China, Japão, etc. (Tecmundo, 2009).

Figura 1 - Ábaco

O ábaco é utilizado para operações matemáticas de adição e subtração.

2.1.2 Régua de Cálculo

Durante vários séculos, o ábaco foi sendo desenvolvido e aperfeiçoado, sendo a principal ferramenta de cálculo por muito tempo. Entretanto, os principais intelectuais da época do renascimento necessitavam descobrir maneiras mais eficientes de efetuar cálculos. Logo, em 1638, depois de Cristo, um padre inglês William Oughtred, criou uma tabela para a realização de multiplicações muito grandes. A base de sua invenção foram as pesquisas sobre logaritmos, realizados pelo escocês John Napier.

Até este momento, a multiplicação de números muito grandes era algo muito trabalhoso e demorado de ser realizado (Kirst, 2011). Porém, Napier descobriu várias propriedades matemáticas e as deu o nome de logaritmos. Após disso, multiplicar valores se tornou uma tarefa mais simples.

O mecanismo de William era consistido de uma régua que já possuía uma quantidade de valores pré-calculados, organizados em forma que os resultados fossem acessados automaticamente. Uma espécie de ponteiro indicava o resultado do valor desejado.

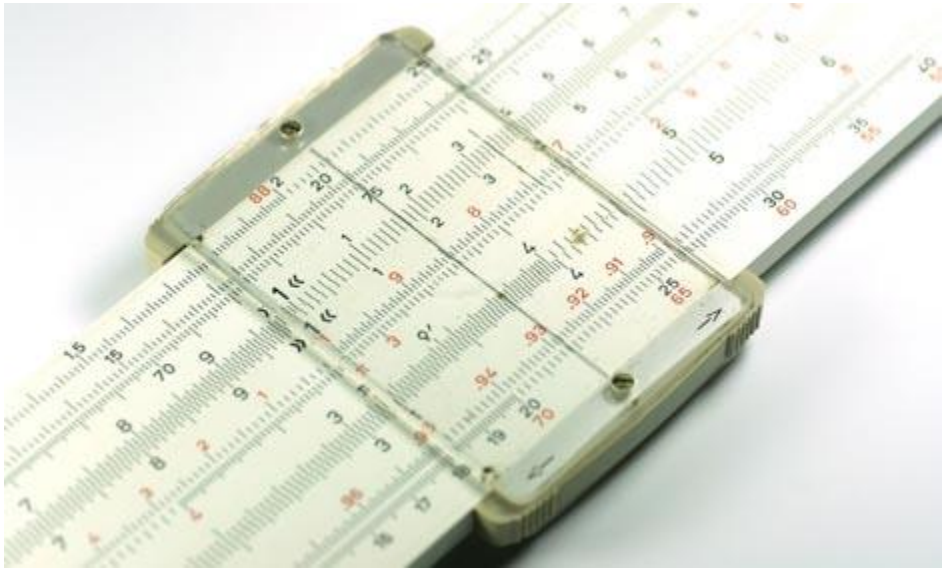


Figura 2 - Régua de Cálculo

2.1.3 Máquina de Pascal

Apesar da régua de cálculo de Oughtred ser útil, os valores presentes nela ainda eram pré-definidos, o que não funcionaria para calcular números que não estivessem presentes na tábua (Giovani, 2011). Pouco tempo depois, em 1642, o matemático francês Bleise Pascal desenvolveu o que pode ser chamado de primeira calculadora mecânica da história, a máquina de Pascal.

Figura 3 - Máquina de Pascal

A máquina contém como elemento essencial uma roda dentada construída com 10 “dentes”. Cada “dente” corresponde a um algarismo, de 0 a 9. A primeira roda da direita corresponde às unidades, a imediatamente à sua esquerda corresponde às dezenas, a seguinte às centenas e assim sucessivamente (Martins, 2011).

2.1.4 Tear Programável

Joseph Marie Jacquard, um mecânico francês, (1725-1834) conseguiu automatizar totalmente o tear mecânico controlado pela grande fita perfurada. O sistema permitia que os padrões dos tecidos fossem definidos pela maneira como os fios eram levantados ou abaixados. Pode ser considerada a primeira máquina mecânica programável da história, pois os cartões forneciam os comandos necessários para a tecelagem de padrões complicados em tecidos e o conjunto de cartões poderia ser trocado sem alterar a estrutura da máquina têxtil (Babbage, C., 2011).

Figura 4 - Tear programável

2.1.5 A máquina de diferenças e o Engenho Analítico

Em 1821, Charles Babbage, inventou a Máquina Diferencial para compilação de tabelas matemáticas (Paulo Heitlinger, 2012). Babbage afirmou que sua máquina era capaz de calcular funções de diversas naturezas (trigonometria, logaritmos) de forma muito simples.

Figura 5 - Máquina de diferenças

Foi com Charles, que o computador moderno começou a ganhar forma, através de seu trabalho no engenho analítico. O equipamento, apesar de nunca ter sido construído com sucesso, possuía todas as funcionalidades do computador moderno. Foi descrito originalmente em 1837, mais de um século antes que qualquer equipamento do gênero tivesse sido construído com sucesso. O grande diferencial do sistema de Babbage era o fato que seu dispositivo foi projetado para ser programável, item imprescindível para qualquer computador moderno.

Durante sua colaboração, a matemática Ada Lovalace publicou os primeiros programas de computador em uma série de notas para o engenho analítico. Por

isso, Lovelace é popularmente considerada como a primeira programadora (Wikidot, 2008).

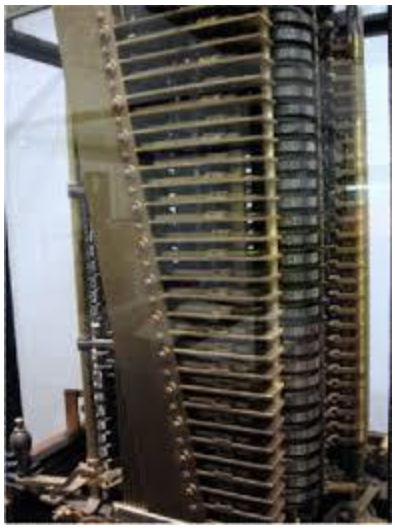


Figura 6 – Engenho Analítico

2.1.6 A Teoria de Boole

Se Babbage é o avô do computador do ponto de vista de arquitetura de hardware, o matemático George Boole pode ser considerado o pai da lógica moderna. Boole desenvolveu, em 1847, um sistema lógico que reduzia a representação de valores através de dois algarismos: 0 ou 1.

Em sua teoria, o número "1" tem significados como: ativo, ligado, existente, verdadeiro. Por outro lado, o "0" representa o inverso: não ativo, desligado, não existente, falso. Para representar valores intermediários, como "mais ou menos" ativo, é possível usar dois ou mais algarismos (bits) para a representação. Por exemplo:

- 00 – desligado
- 01 – carga baixa
- 10 – carga moderada
- 11 – carga alta

Todo o sistema lógico dos computadores atuais, inclusive o do qual você está usando, usa a teoria de Boole de forma prática.

2.1.7 Máquina de Hollerith

Herman Hollerith desenvolveu uma máquina que acelerava todo o processo de computação dos dados (tecmundo, 2009). Foi o responsável por uma grande mudança na maneira de se processar os dados dos censos da época através da

invenção de uma máquina capaz de processar dados baseada na separação de cartões perfurados (pelos seus furos).

A máquina de Hollerith foi utilizada para auxiliar no censo de 1890, reduzindo o tempo de processamento de dados. Ela foi a primeira a utilizar a eletricidade na separação, contagem e tabulação dos cartões.

Os dados do censo de 1880 eram processados manualmente e levaram sete anos e meio para serem compilados, com a ajuda de uma máquina de perfurar cartões e máquinas de tabular e ordenar, criadas por Hollerith os dados do censo de 1890 foram processados em dois anos e meio (Fonsecas e Menezes, 2014).

Figura 7 – Máquina de Hollerith

2.1.8 Computadores pré-modernos

Na primeira metade do século XX, vários computadores mecânicos foram desenvolvidos, sendo que, com o passar do tempo, componentes eletrônicos foram sendo adicionados aos projetos. Em 1931, Vannevar Bush implementou um computador com uma arquitetura binária propriamente dita, usando os bits 0 e 1. A base decimal exigia que a eletricidade assumisse 10 voltagens diferentes, o que era muito difícil de ser controlado. Por isso, Bush fez uso da lógica de Boole, onde somente dois níveis de voltagem já eram suficientes.

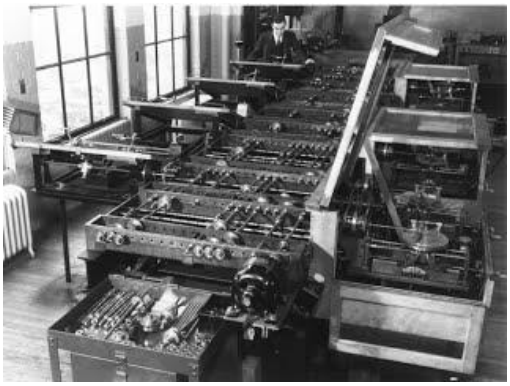


Figura 8 - Computador de Vannevar Bush

2.1.9 Computação moderna

A computação moderna pode ser definida pelo uso de computadores digitais, que não utilizam componentes analógicos com base de seu funcionamento. Ela pode ser dividida em várias gerações.

2.1.9.1 Primeira geração (1946 – 1959)

A primeira geração dos computadores é marcada pela utilização de válvulas. A válvula é tudo de vidro, similar a uma lâmpada fechada sem ar em seu interior, ou seja, um ambiente fechado a vácuo, e contendo eletrodos, cuja finalidade é controlar o fluxo de elétrons (Introdução a Computação, 2012). As válvulas aqueciam bastante e costumavam queimar com facilidade.

O armazenamento de dados era realizado em cartões perfurados, que depois passaram a ser feitos em fita magnética.

Figura 9 - Válvula

Um dos representantes desta geração é o ENIAC. Ele possuía 17.468 válvulas, pesava 30 toneladas, tinha 180m^2 de área construída, sua velocidade era da ordem de 100 kHz e possuía apenas 200 bits de memória RAM.

2.1.9.2 Segunda geração (1959 – 1964)

Na segunda geração, houve a substituição das válvulas eletrônicas por transístores, o que diminui em muito o tamanho do hardware. A tecnologia de circuitos impressos também foi criada, assim evitando que os fios e cabos elétricos ficassem espalhados por todo lugar (A história dos computadores, 2011).

Figura 10 - IBM 7030 (Modelo que marcou a segunda geração de computadores)

2.1.9.3 Terceira geração (1965 – 1970)

A terceira geração começa em 1965 com a substituição dos transistores pela tecnologia dos circuitos integrados (CI).

Os transistores e outros componentes eletrônicos são miniaturizados e montados em um único chip.

As principais características da terceira geração são as seguintes:

- Mais confiável em comparação as gerações anteriores.
- Tamanho menor.
- Gerava menos calor.
- Mais rápido que a segunda geração.
- Menor manutenção.
- Consumia menos energia elétrica.
- Maior capacidade de processamento.
- Início da utilização dos computadores pessoais.

Figura 11 - IBM-360 series

2.1.9.4 Quarta geração (1971 – 1980)

A quarta geração é marcada pelos primeiros microprocessadores, pelo surgimento de novas linguagens de programação e no modo de armazenamento através de disquetes (Silva, Rafael, 2014).

Figura 12 – Processador Intel 4004

2.1.9.5 Quinta geração (1980 – atualmente)

A quinta geração de computadores tem como característica a simplificação e miniaturização do computador, além de melhor desempenho e maior capacidade de armazenamento. O marco nesta evolução, para chegarmos aos computadores como conhecemos hoje, foi a invenção dos sistemas operacionais, dos quais o Windows é um exemplo. Estes sistemas permitem que vários programas estejam rodando ao mesmo tempo, conferindo grande flexibilidade ao uso do computador.

Por conta disso tudo, os computadores começaram a se tornar mais baratos, mais “amigáveis” e mais “úteis” às pessoas comuns. Por isso, sobretudo a partir da

década de 80, os computadores começaram a se popularizar, e hoje são realidade para milhões de pessoas no mundo inteiro.

2.2 Arquitetura de um Computador

Os circuitos de um computador que executam operações sobre dados, tais como adição e subtração, são isolados em uma região chamada Unidade Central de Processamento UCP (CPU – *Central Processing Unit*), ou simplesmente processador.

Observação: Muitas pessoas chamam erroneamente o gabinete do computador de CPU. CPU é o processador, enquanto que o gabinete é o compartilhamento que contém a maioria dos componentes de um computador (normalmente, excluindo o monitor, teclado e mouse).

3 Algoritmos

Um algoritmo nada mais é do que uma receita que mostra passo a passo os procedimentos necessários para a resolução de uma tarefa. Ele não responde a pergunta "o que fazer?", mas sim "como fazer". Em termos mais técnicos, um algoritmo é uma sequência lógica, finita e definida de instruções que devem ser seguidas para resolver um problema ou executar uma tarefa (tecmundo, 2009).

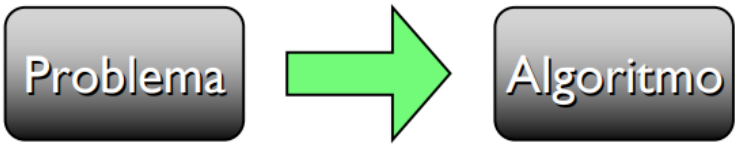


Figura 13 - Algoritmo

Algoritmos são muito utilizados na área de programação, descrevendo as etapas que precisam ser efetuadas para que um programa execute as tarefas que lhe são designadas. Existem diversas formas de escrever um algoritmo, podendo ser citadas o pseudocódigo (ou português estruturado), fluxogramas, diagrama de Chapin e descrição narrativa.

Os dois tipos mais comuns são o pseudocódigo que utiliza uma forma mais estruturada, assemelhando-se àquelas utilizadas pelas linguagens de programação e o fluxograma (assunto que iremos abordar no próximo capítulo) que emprega figuras geométricas para ilustrar os passos a serem seguidos.

Um algoritmo é algo que você já utilizava o tempo todo e nem percebia. Uma forma bem interessante de exercitar a lógica seria algum dia parar para pensar em sua rotina ao acordar, por exemplo. Você perceberia o quanto ela é mais complexa do que parece, se fosse para colocar no papel e como a execução disto é automática.

3.1 Exemplos

3.1.1 Troca de uma lâmpada

Inicio
Verifica se o interruptor está desligado;
Procura uma lâmpada nova;
Pega uma escada;
Leva a escada até o local;
Posiciona a escada;
Sobe os degraus;
Para na altura apropriada;
Retira a lâmpada queimada;
Coloca a lâmpada nova;

Desce da escada;
Aciona o interruptor;
Se a lâmpada não acender, então:
Retira a lâmpada queimada;
Coloca outra lâmpada nova
Senão
Tarefa terminada
Joga a lâmpada queimada no lixo;
Guarda a escada;
Fim

3.1.2 Receita de um bolo de chocolate

Inicio
Bater duas claras em neve;
Adicionar duas gemas;
Adicionar uma xícara de açúcar;
Adicionar duas colheres de margarina;
Adicionar uma xícara de farinha de trigo;
Adicionar uma colher de chá de fermento;
Adicionar uma xícara de chocolate em pó;
Levar a batedeira até formar uma massa homogênea;
Colocar numa forma e levar ao forno em fogo brando;
Fim

3.1.3 Trocar um pneu de um carro

Inicio
Afrouxar ligeiramente as porcas;
Suspender o carro com o macaco;
Retirar as porcas e o pneu;
Colocar o pneu reserva;
Apertar as porcas;
Abaixar o carro;
Dar o aperto final nas porcas
Fim

3.2 Pseudocódigo

Pseudocódigo é uma forma de representação de algoritmos, é praticamente um programa escrito em português que, depois, podemos passar para o computador (Embarcados, 2016). Para escrevermos códigos com pseudocódigo precisaremos conhecer alguns comandos básicos.

- **escreva (“ ”)** = comando usado para imprimir uma mensagem na tela;
- **leia ()** = comando usado para ler valores digitados no teclado;

- **<-** = comando de atribuição;
- **início** = palavra usada para iniciar o programa;
- **fim** ou **fimalgoritmo** = palavra usada para finalizar o algoritmo;
- **var** = palavra usada para declarar variáveis;
- **algoritmo** = palavra usada para indicar o início do programa.

3.2.1 Algoritmo para ler dois números

Construiremos um algoritmo que leia dois números. Em seguida, iremos calcular a soma desses números, armazenando o resultado em outra variável. Por fim, imprimiremos os dados iniciais e a soma.

```
1   algoritmo "Soma"
2   var
3   n1, n2, soma: decimal
4   inicio
5   escreva ("digite um número: ");
6   leia (n1);
7   escreva ("digite outro número: ");
8   leia (n2);
9   soma <- n1 + n2;
10  escreva ("Primeiro número: ", n1);
11  escreva ("Segundo número: ", n2);
12  escreva ("A soma é: ", soma);
13  fim
```

Figura 14 - Algoritmo para ler dois números

Vamos analisar o algoritmo desenvolvido para solucionar o problema de somar dois números.

Linha 1: Usamos a palavra-chave **algoritmo** para começar o nosso algoritmo. Observe que em seguida temos um nome entre as aspas duplas, então, sempre que for desenvolver um algoritmo você deve escrever na primeira linha **algoritmo "nome_do_algoritmo".**

Linha 2: Utilizamos a palavra **var** para indicar que naquele espaço devemos declarar as variáveis que usaremos em nosso programa.

Linha 3: aqui são declaradas as variáveis, com seus identificadores e tipos. Observe que primeiro vem o nome (identificador) da variável e após os dois pontos é que vem o tipo dela: n1 e n2 são os dois números, soma é a variável que armazenará o resultado da operação.

Linha 4: A palavra-chave **inicio** indica que a partir daquele ponto, tem-se o início do programa principal. Programa principal? Mas peraí, a gente já não está no programa? Pois é!!! Vamos esclarecer isso. O algoritmo inteiro, das linhas 01 à 13, é um programa, mas o programa é dividido em partes, como estamos percebendo aqui: nome do programa, declaração de variáveis, etc. Isso é norma para todas as linguagens de programação. O que ocorre é que, o que de fato o algoritmo vai fazer, vai no programa principal, ou seja, dentro de **inicio** e **fim.**

Linha 5: O comando **escreva** é usado para que seja impresso na tela do usuário um texto. Neste caso, está sendo impresso na tela um texto de solicitação. Solicitamos que o usuário digite um número.

Linha 6: O comando **leia** é empregado para que o valor digitado pelo usuário, no teclado, seja armazenado na variável correspondente. Portanto, será atribuído a variável n1 o valor que o usuário digitar.

Linha 7: Idem a linha 5.

Linha 8: Idem a linha 6.

Linha 9: Aqui é realizada uma operação matemática, mais especificamente aritmética. Observe que usamos o comando **<-** para que seja atribuído a variável soma o resultado da soma entre os valores armazenados em n1 e n2.

Linha 10: Novamente usamos o comando **escreva**, mas veja que desta vez há algo diferente em relação as linhas 5 e 7. Estamos imprimindo na tela, junto com o texto que queremos, os valores das variáveis. Isso é feito usando-se a virgula e, pondo após ela, a variável que queremos que seja impressa na tela.

Linha 11: Idem a 10.

Linha 12: Idem a 10.

Linha 13: Utilizamos a palavra-chave **fim** para finalizar o programa principal e terminar a execução do programa. Essa palavra-chave força a saída do programa, fechando a janela e liberando o espaço de memória que estavam reservados para as variáveis.

4 Linguagem de Programação

Uma linguagem de programação é um método padronizado para expressar instruções para um computador, ou seja, é um conjunto de regras sintáticas e semânticas usadas para definir um programa do computador (Tarcísio Ruas Ferraz, 2011). 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 tomadas sob várias circunstâncias (Fernando Pereira, 2009).

4.1 Linguagem C

A linguagem C foi criada inicialmente por Dennis M. Ritchie e Ken Thompson no laboratório Bell em 1972, baseada na linguagem B, de Thompson, evolução da antiga linguagem BCPL (Alfredo Boente).

Segundo Rodrigo Soster, 2012, C é uma linguagem vitoriosa como ferramenta na programação de qualquer tipo de sistema (sistemas operacionais, planilhas eletrônicas, processadores de textos, gerenciadores de bancos de dados, processadores gráficos, sistemas de transmissão de dados e telefonia, aparelhos de medicina, aparelhos de segurança para solução de problemas de engenharia ou física, etc.)

C foi desenhada para que o usuário possa planejar programas estruturados e modulares. O resultado é mais legibilidade e documentação. Os programas em C tendem a ser bastante compactos e de execução rápida (Victorine Viviane Mizrahi, 2008).

Um programa C consiste de funções sendo que, necessariamente, uma delas deve ser denominada *main* (Silvio do Lago Pereira, 1997).

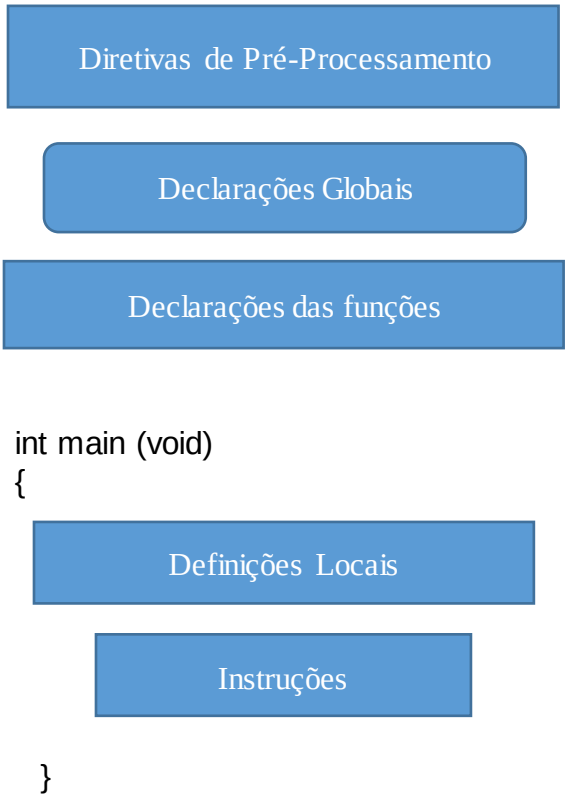
4.1.1 Compiladores

Um compilador é um programa de sistema que traduz um programa descrito em uma linguagem de alto nível para um programa equivalente em código de máquina para um processador (Ivan L. M. Ricarte, 2003).

Um compilador lê a primeira instrução do programa, faz uma consistência de sua sintaxe e, se não houver erro, converte-a para linguagem de máquina; segue para a próxima instrução, repetindo o processo até que a última instrução seja atingida ou a consistência aponte algum erro (José Rodrigues, 2013).

4.1.2 Estrutura básica de um programa C

4.1.2.1 Estrutura de um programa em linguagem C



4.1.2.2 Exemplo de um programa em linguagem C

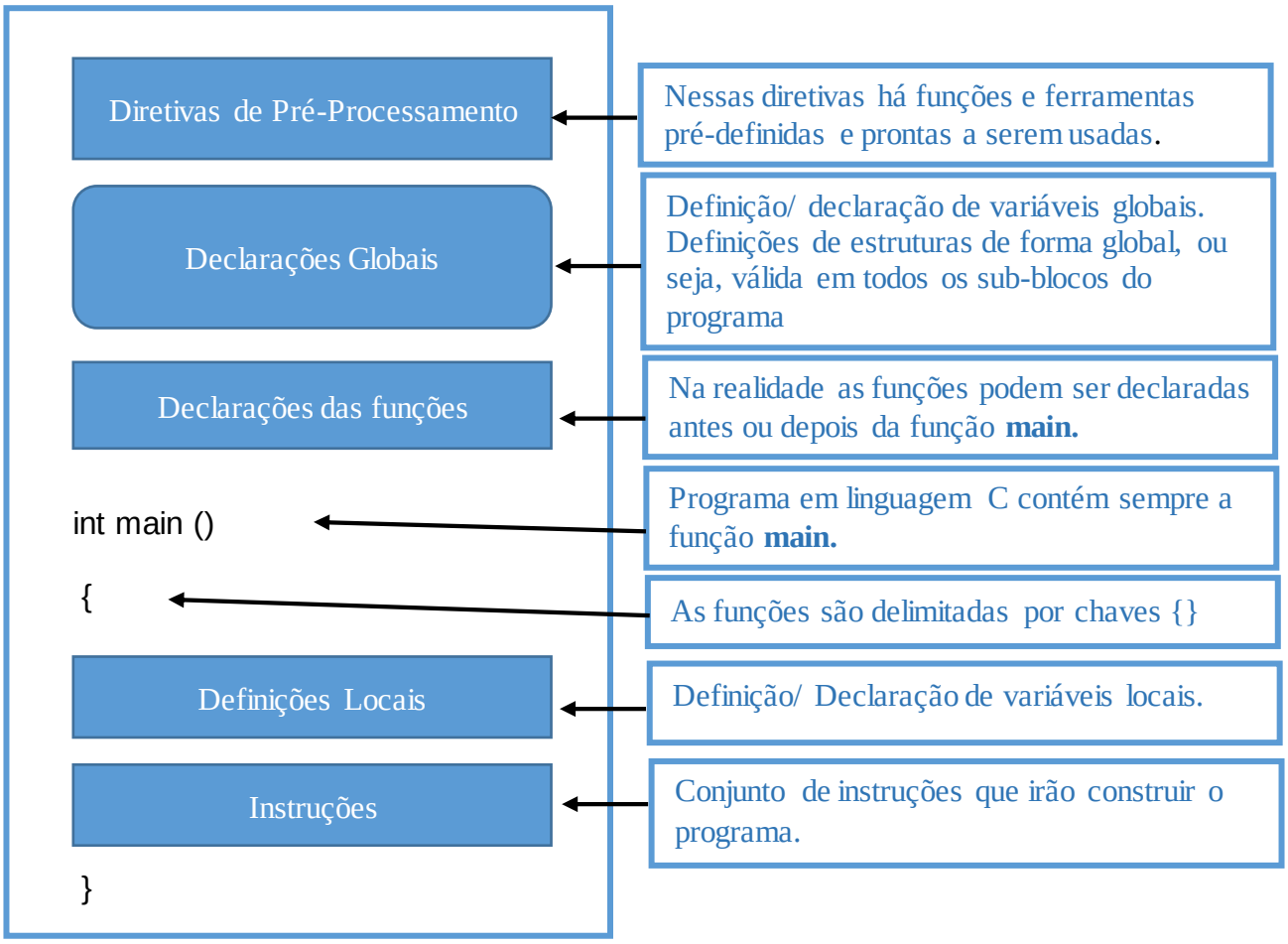
```
#include <stdio.h>
#include <stdilib.h>

int main ()
{

printf (“Hello World \n”);

return 0;

}
```



4.1.3 A diretiva #include

A diretiva **#include** provoca a inclusão de outro arquivo no programa fonte. Na verdade, o compilador substitui a linha contendo essa diretiva pelo conteúdo do arquivo indicado. Essa substituição é executada antes de o programa ser compilado. Assim, o efeito obtido é a apresentação de um texto, como se tivéssemos digitado todo o conteúdo do arquivo **stdio.h** e do arquivo **stdilib.h** na posição em que escrevemos as linhas.

4.1.3.1 Stdio.h e stdlib.h

O cabeçalho **stdio.h** define três tipos de variáveis, vários macros e várias funções para executar entrada e saída (Tutorials Point, 2017).

Essa biblioteca contém as funções **printf()** e **scanf().**

Já o cabeçalho **stdlib.h** funciona como um emulador do *prompt* do sistema operacional que está programando, caso seja Windows ele emula todos os comandos do Dos, se estiver programando no Linux, quem será emulado é o Terminal (Guilherme Carvalho, 2012).

4.1.4 Códigos especiais

Além do caractere [ENTER], vários outros caracteres não podem ser digitados diretamente do teclado para dentro do programa. Esses caracteres são codificados em C por meio da combinação do sinal \ (barra invertida) com outros caracteres.

A tabela seguinte mostra esses códigos.

Códigos especiais	Significado
\n	Nova linha
\t	Tabulação
\b	Retrocesso (usado p/ impressora)
\f	Salto de página de formulário
\a	Beep – Toque de autofalante
\r	CR – Retorno do cursor para o início da linha
****	\ - Barra invertida
\0	Zero
\'	Aspas simples (apóstrofo)
\"	Aspas dupla
\xdd	Representação hexadecimal
\ddd	Representação octal

Tabela 1 - Códigos especiais

Códigos de formatação para printf ()	Significado
%c	Caractere simples.
%d	Inteiro decimal com sinal.
%i	Inteiro decimal com sinal.
%e	Notação científica (e minúsculo).
%E	Notação científica (E maiúsculo).
%f	Ponto flutuante em decimal.
%g	Usa %e ou %f, o que for menor
%G	Usa %E ou %f, o que for menor.
%o	Inteiro decimal sem sinal.
%s	String de caracteres.
%u	Inteiro decimal sem sinal.
%x	Inteiro hexadecimal sem sinal (letras minúsculas)
%X	Inteiro hexadecimal sem sinal (letras maiúsculas)
%p	Ponteiro (endereço).
%n	Ponteiro inteiro.
%%	Imprime um caractere %.

Tabela 2 - Códigos de formatação para printf()

4.1.5 Comentários de programa

Os comentários de um programa devem ser colocados entre /* e */. O compilador ANSI C aceita os comentários entre /* e */. Quaisquer textos colocados entre estes dois símbolos serão ignorados pelo compilador (Wiki para programação em C).

Também há a possibilidade de se comentar utilizando //, para isso tem-se que colocar estes caracteres em cada linha antes de digitar o comentário.

Abaixo há 2 exemplos de comentários.

```
/* Este é meu primeiro programa feito em
linguagem c */
```

Figura 15 - Comentário utilizando /* e */

```
// Este é meu primeiro programa feito em
// linguagem c
```

Figura 16 - Comentário utilizando //

4.1.6 Variáveis

São espaços de memória reservados que guardam valores durante a execução de um programa (Leonardo Barreto Campos). Todas as variáveis em C devem ser declaradas, antes de serem usadas. Há as variáveis globais e as variáveis locais, vamos ser a seguir a diferença entre elas:

Variáveis globais:

- São aquelas declaradas fora de todos os blocos de funções;
- São acessíveis em qualquer parte do programa, ou seja, podem ser usadas e modificadas por todas as outras funções;
- Existem durante toda a execução do programa.

Variáveis locais:

- São aquelas declaradas dentro do bloco de uma função;
- Não podem ser usadas ou modificadas por outras funções;
- Somente existem enquanto a função onde foi declarada estiver sendo executada.

```
1  #include <stdio.h>
2
3  int a = 1; // VARIÁVEL GLOBAL
4
5  void main()
6  {
7      int b = 2; // VARIÁVEL LOCAL
8
9      printf("Valor da variavel 'a': %d", a);
10     printf("\nValor da variavel 'b': %d", b);
11 }
```

Figura 17 - Exemplo Variável Local x Variável Global

Em nosso exemplo vemos que a variável **a** é uma variável global, uma vez que todo o programa pode usa-la, enquanto que a variável **b** é uma variável local, já que ela só pode ser enxergada pela função ***main*****,** se o programa tiver outra função, enxergará somente a variável **a.**

O nome de uma variável pode ser de uma letra até palavras com no máximo 32 caracteres.

OBS: O nome de uma variável não pode ser igual a uma palavra reservada, em outras palavras não pode ser igual a de uma função declarada pelo programador ou pelas bibliotecas de C.

auto	**double**	**int**	**struct**
break	**else**	**long**	**switch**
case	**ebun**	**register**	**typedef**
char	**extern**	**return**	**union**
const	**float**	**short**	**unsigned**
continue	**for**	**signed**	**void**
default	**goto**	**sizeof**	**volatile**
asm	**pascal**	**far**	**huge**
interrupt	**near**	**_cs**	**_ds**

Tabela 3 - Funções utilizadas pela linguagem C

4.1.6.1 Tipos de variáveis

Os dados podem assumir cinco tipos básicos em C que são:

- **char:** Caracter. O valor armazenado é um caracter. Caracteres geralmente são armazenados em códigos (usualmente o código ASCII).
- **int:** Número inteiro. É o tipo padrão e o tamanho do conjunto que pode ser representado, normalmente depende da máquina em que o programa está rodando.
- **float:** Número em ponto flutuante de precisão simples. São conhecidos normalmente como números reais.
- **double:** Número em ponto flutuante de precisão dupla.

Tipo	Valores Válidos
char	Letras e símbolos: ‘a’, ‘b’, ‘H’, ‘^’, ‘*’,’1’,’0’
int	De -32767 até 32767 (apenas números inteiros)
float	De -3.4 x 10^{38} até +3.4 x 10^{38} com até 6 dígitos de precisão
double	De -1,7 x 10^{308} até +1,7 x 10^{308} com até 10 dígitos de precisão

Tabela 4 - Tipos de Variáveis

Com exceção de **void** todos os tipos básicos de dados podem ser acompanhados por um modificado. Em C temos 3 modificadores (**short, long** e **unsigned)**.

Um modificador de tipo pode ser utilizado sem que seja especificado o tipo de variável. Quando isso é feito, o compilador assume, por padrão, que o tipo é **int.**

Tipo	Bits	Bytes	Escala
char	8	1	128 a 127
int	32	4	-2.147.483.648 a 2.147.383.647 (ambientes de 32 bits)
short	16	2	-32.765 a 32.767
long	32	4	-2.147.483.648 a 2.147.383.647
unsigned char	8	1	0 a 255
unsigned	32	4	0 a 4.294.967.295 (ambientes de 32 bits)
unsigned long	32	4	0 a 4.294.967.295
unsigned short	16	2	0 a 65.535
float	32	4	3.4×10^{-38} a 3.4×10^{38}
double	64	8	1.7×10^{-308} a 1.7×10^{308}
long double	80	10	3.4 a 10^{-4932} a 3.4 10^{4932}
void	0	0	Nenhum valor

Tabela 5 - Modificadores de tipo

4.1.7 Constantes

São usadas para armazenar valores que não podem ser modificados durante a execução de um programa (Eduardo Casavella, 2015).

Uma constante precisa ser declarada, e para tanto usamos a diretiva pré-processador **#define.**

A declaração da constante deve ser feita no início do programa.

Exemplos de declaração de constantes:

#define ICMS 0.18

#define MAX 100

#define ERRO “Erro!!!”

Quando o programa é compilado, o compilador substitui as ocorrências das constantes definidas pelo valor declarado.

As constantes têm duas vantagens principais:

- Facilitam a modificação do programa;
- Tornam o programa mais legível.

4.1.8 Operador de Atribuição

Os operadores indicam ao compilador a necessidade de se fazer manipulações matemáticas ou lógicas (Júlio Battisti).

4.1.8.1 Operadores aritméticos

Os operadores aritméticos são usados para calcular expressões matemáticas. . Veja a tabela na página a seguir.

Operador binário	Descrição
=	Atribuição
+	Soma
-	Subtração
/	Divisão
%	Modulo (obtém o resto da divisão)

Tabela 6 - Operadores binários

4.1.8.2 Operadores lógicos

Os operadores lógicos servem para interligar mais de uma operação relacional. E assim os relacionais retornam zero para falso e um para verdadeiro.

Operador	Descrição
&&	AND
||	OR
!	NOT (operador de negação)

Tabela 7 - Operadores lógicos

4.1.8.3 Operadores bit a bit

A linguagem C é considerada de baixo nível, pois permite a manipulação de bits. Isto é feito através dos operadores bit a bit listados a seguir.

Operador	Descrição
&	AND
|	OR
^	XOR (OR exclusivo)
<<	Deslocamento para esquerda
>>	Deslocamento para direita

Tabela 8 - Operadores bit a bit

Embora a descrição na tabela seja parecida com as dos operadores lógicos, eles não devem ser confundidos. Os operadores bit a bit são muito usados em rotinas de modens e de impressoras.

4.1.8.4 Operadores relacionais

Esses operadores são responsáveis pelas comparações de expressões nos programas. A lista completa se encontra na página a seguir:

Operador	Descrição
>	Maior
>=	Maior Igual
<	Menor
<=	Menor igual
==	Igualdade
!=	Diferente

Tabela 9 - Operadores relacionais

OBS: É importante distinguir (=) de (==). O primeiro atribui um valor e o segundo compara expressões.

4.1.9 A função printf() e scanf()

A saída de dados (impressão na tela) é realizada pela função printf(): Exemplo: printf("Digite o seu nome:"\n);

Printf() mandará para a tela o que houver entre o parênteses. Neste caso, a mensagem "Digite o seu nome". Printf() também pode imprimir valores de variáveis utilizando código especiais (BlogDeCodigo, 2011).
Exemplo: printf("Nome: %s", &nome);

A entrada de dados (recepção dos dados digitados) é responsabilidade da função scanf():
Exemplo: scanf("%s", &nome);

Em scanf(), o uso de especificadores de formato é obrigatório, pois ele define na memória se o espaço a ser reservado será para um número inteiro, real, um caractere, uma cadeia de caracteres (strings) ou uma notação científica. Após a vírgula, vem o endereço onde essa variável será alocada, obrigatoriamente, procedido do sinal **&**.

Importante

1. Nunca se esqueça do uso de ponto-e-vírgula para separar comandos;
2. Nunca utilize acentos nas mensagens exibidas por printf() e nas declarações de variáveis.

Um programa em linguagem C sempre é finalizado da seguinte maneira:

```
system("PAUSE");

return 0;
```

A função **system**() com o parâmetro **PAUSE** permite que o usuário veja os resultados antes que a janela seja fechada. Sem essa função, o programa seria executado e a janela seria automaticamente fechada.

O comando **return** indica o termino da função **main().** O retorno do valor 0 (positivo) indica que o programa foi executado normalmente, enquanto o retorno de um valor negativo indicaria erro.

4.1.10 Palavras-chave de C

Categoria	Palavras-Chave
Tipos de dados	char, int, float, double, void
Modificadores de tipo	long, short, signed, unsigned
Modificadores de tipo de acesso	const, volatile
Classes de armazenamento	auto, extern, static, register
Tipos definidos pelo usuário	struct, enum, union, typedef
Comandos condicionais	if, else, switchm casem default
Comandos de laço	while, for, do
Comandos de desvio	break, goto, return, continue
Operador	sizeof

Tabela 10 - Palavras-chave de C

4.1.11 End of line (Terminador de linha)

O comando end of line tem como objetivo permitir a inserção de novos comandos em outras linhas, além de facilitar a comparação da saída do código submetido com o a saída cadastrada no sistema.

4.1.12 Códigos existentes no sistema

4.1.12.1 Erro de sintaxe

Esses são erros de digitação ou outro erro no código que bloqueiam a execução de um script. Como por exemplo falta de um ponto-e-vírgula no final de cada instrução do programa.

4.1.12.2 Erro de resposta

Acontece quando a resposta enviada pelo usuário é diferente ao do que foi pedido no exercício, como por exemplo, pede-se para o usuário realizar uma soma entre dois números e o mesmo faz uma subtração. Neste caso, caracterize-se erro de resposta.

4.1.12.3 Overflow (tempo excedido)

Overflow pode ser traduzido como “estouro”. O significado em geral é a condição que ocorre quando os dados resultantes da entrada ou do processamento exigem mais bits do que os que foram fornecidos no hardware ou no software para armazenar os dados (Portal Terra, 2014). Outra definição é parte de um item de dados que não pode ser armazenada porque os dados excedem a capacidade da estrutura disponível.

4.1.12.4 Presention error (erro de apresentação)

Caracteriza-se um erro de apresentação quando o usuário coloca um espaço a mais ou não coloca espaço, como no exemplo a seguir:

Resposta esperada: X = 10

Resposta submetida: X = 10

No exemplo acima, vimos que na resposta submetida há um espaço mais do que deveria ter, o usuário acertou a resposta, porém errou na apresentação do código, logo ele receberá uma mensagem de erro em que é mostrada o erro de apresentação.

4.1.12.5 Código aceito

Ao receber este código o usuário fica ciente de que solucionou o problema de acordo com a questão proposta.

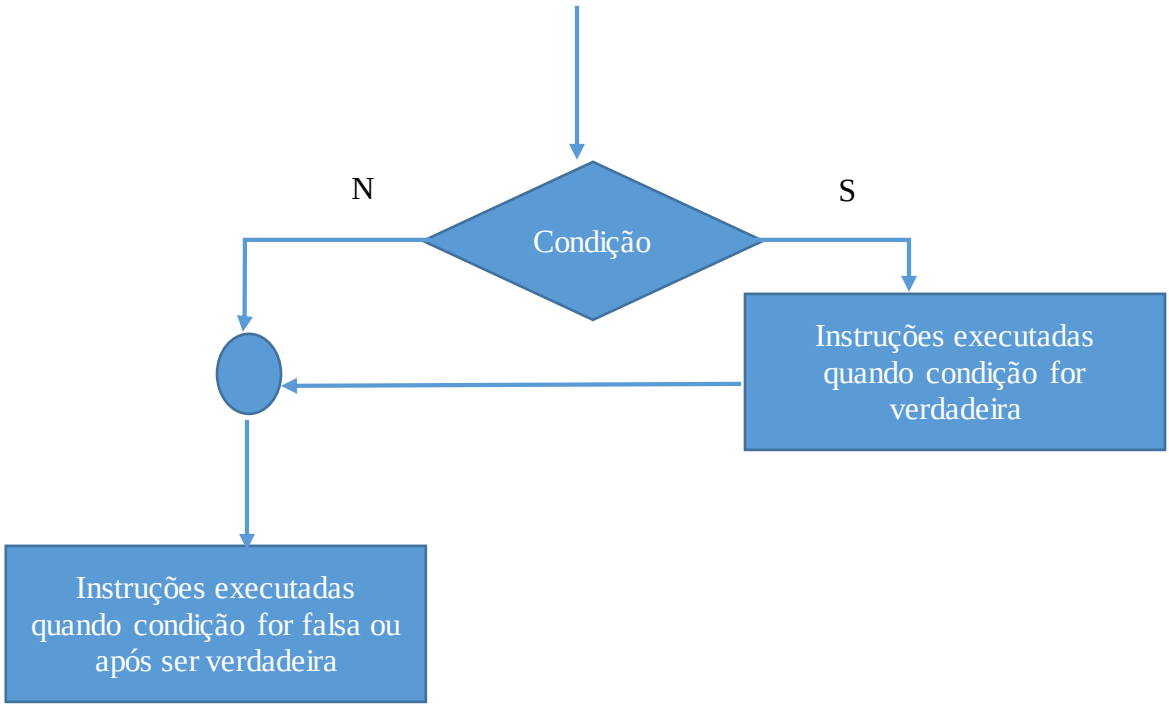
5 Tomada de decisão

Tomadas de decisão são ferramentas importantes em programação. Decidir qual ação tomar, quando e o que fazer são peças chave em qualquer linguagem de programação (Viva o Linux, 2008).

5.1 Diagrama de blocos ou fluxograma

Segundo Juliano Schimiguel, 2013 Diagrama de blocos ou fluxograma é uma forma padronizada eficaz para representar os passos lógicos de um determinado processamento (algoritmos).

Através do fluxograma iremos apresentar um exemplo de tomada de decisão.



No exemplo acima vimos a forma como a tomada de decisão funciona, você tem uma condição e verifica se ela é verdadeira ou falsa, de acordo com a resposta o programa executa a instrução programada.

A seguir, veremos outro exemplo do uso de tomada de decisão:

```
} if (op == '/'){
    divisao = x / y;
    printf ("O resultado da divisao e: %.2f", divisao);
}
```

Figura 18 - Tomada de decisão IF

No exemplo da página anterior temos uma tomada de decisão onde:

- Na primeira linha é feita a decisão, em que a variável é feita uma comparação, para verificar se a variável **op** é igual a **/.**
- Caso seja igual ele entra dentro do **if** e realiza a operação x / y, a qual é armazenada dentro da variável **divisao**
- Na terceira linha é impresso o resultado da divisão

Importante: Essa operação somente será feita se a atender a condição imposta pelo if. A seguir veremos um caso de if/ else (se/ senão).

A estrutura if/ else é utilizada para casos em que se pretende verificar mais de um caso, como, por exemplo, quando é desejado mostrar a situação do aluno mediante a sua média, no exemplo abaixo, é mostrado se o aluno tiver uma nota menor do que 4, é impresso "Reprovado", **senão se** a média do aluno for maior ou igual a 4 **e** menor do que 6, é imprimido "Em recuperação", **senão**, caso nenhum dos testes acima forem verdadeiros é mostrado na tela que o aluno foi aprovado.

```
if(media < 4) {
    printf("Reprovado\n");
}
else if(media >=4 && media <6){
    printf("Em recuperacao\n");

} else{
    printf("Aprovado\n");
      }
```

Figura 19 - Exemplo de if else

Nota-se que na figura 19 é evidenciado como é a estrutura de um if/ else. Na primeira linha compara-se a média do aluno, que, caso for menor do que 4 entrará dentro deste if. Se este caso for falso, vai-se para o próximo caso, a estrutura senão se, na qual, é feita novamente um novo teste, em que se a média for menor do que 4 e (&&) menor do que 6, mostrará na tela que o aluno está em recuperação.

Por último caso nenhuma das duas condições acima forem verdadeiras, será imprimido que o aluno foi aprovado.

Nota: Diferença entre **else if** e **else.** No caso de **else if**, há um teste dentro dele enquanto que na estrutura **else** não é feita nenhuma comparação. Fique atento quando for usar **else if** e **else.**

6 Estruturas de repetição

Estruturas de repetição nada mais é do que laços de repetições. Laços de repetições permitem que um conjunto de instruções seja repetido até que se faça a condição desejada (Alisson Santos, 2014).

As estruturas de repetição, executam a repetição de um conjunto de instruções enquanto uma determinada condição seja verdadeira (Eduardo Casavella, 2013).

6.1 Laço for

Laço utilizado geralmente quando existe um término definido para esse laço implementado no início.

Em pseudocódigo o laço for da linguagem C é equivalente ao comando **Para**.

Vejamos seu funcionamento abaixo:

Para (valor_inicial até condição_final passo n) faça
Inicio
  Instruções;
Fim

Onde:

- valor_inicial é uma instrução de atribuição do valor inicial do laço para a variável de controle.
- condição_final é uma condição que controla o laço.
- passo é o incremento do laço.

Vamos a um exemplo de código em C.

```
for(i = a; i > 0; i--)
{
    if(a % i == 0)
    {
        b++;
    }
}
```

Figura 20 - Laço for

Na figura 20, vemos que a estrutura **for** é dividida em 3 partes, separadas pelo ponto e vírgula, na primeira temos o valor_inicial em que **i** recebe o valor de **a**, a condição_final é que **i** seja maior do que 0, ou seja, neste exemplo o loop será realizado enquanto i for maior do que 0. E por fim, o passo é decrementar **i**, isto é, a cada iteração do laço, o valor de **i** diminui.

Teste lógico, para i igual a 10, testa-se se i é maior do que 0, 10 é maior do que 0, portanto ele entra no loop, feita a primeira iteração, decrementa-se 1 de **i**, portanto i passa a valer 9, e então, novamente faz-se o teste se i é maior do que 0, 9 é maior do que 0, então novamente ele realiza o que estiver dentro do loop.

O loop será feito até **i** ser igual a 0, no momento do teste quando **i** for igual 0, é feita a comparação se 0 é maior do que 0, como 0 não é maior do que 0, não se executa o loop e sai da estrutura e continua a programa, caso o mesmo tenha mais instruções depois.

6.1.1 For dentro de for

Também é possível utilizar um for dentro de um outro for, isso é muito utilizado para se percorrer matrizes. Vamos a um exemplo:

```
//para encontrar o valor alvo
int count = 0;
for(i=0;i<1800;i++) //loop para percorrer a linha da matriz até 1800
{
 for(j=0;j<920;j++) //loop para percorrer a coluna da matriz até 920
  {
   if(alvo == matriz[i][j]) //verifica se o alvo existe na matriz
        count++; //incrementa count

  }
}
```

Figura 21 - For dentro de um for

Na figura 21, temos um exemplo do uso do laço for dentro de um outro laço for, neste caso, deseja-se encontrar um valor alvo e contar quantas vezes o mesmo aparece.

Neste exemplo temos uma matriz de 1800 x 920. Sendo assim é necessário que se percorra a matriz inteira e para fazermos isso, temos que percorrê-la linha a linha e coluna a coluna.

Vamos ao teste:

A variável **i** começa em 0 e fará o loop enquanto **i** for menor do que 1800, sempre incrementando 1 a **i.**

- para i igual a 0, então entra na próximo loop, o qual começa com **j** igual a 0, enquanto **j** for menor que 920, ele percorrerá a coluna da matriz, incrementando 1 a **j** em cada iteração.
- Neste caso para o primeiro caso **i** = 0, **j** = 0, então realiza-se o loop de dentro.
- Na segunda iteração **i** = 0, **j** = 1.
- Note que o segundo loop será feito até que **j** chegue a 920, somente após isso que se sai deste loop.
- Então **i** passa a ser igual a 1. Novamente entra-se no loop de baixo.
- Teremos **i** = 1, **j** =0, faça-se o loop de **j**.
- Depois **i** = 1, **j** = 1, e assim por diante até que j chegue a 920.
- Este loop só terminará quando **i** chegar a 1800.
- Neste caso, garantimos que a matriz inteira será percorrida.

6.2 While

Laço que pode ter uma condição de término definida já no início ou não. Necessariamente ele testa a condição e se caso for verdadeira executa o bloco abaixo, caso seja faço falso ele vai para a próxima instrução fora do laço.

Sintaxe:
Iniciar a variável de controle
Enquanto (condição) faça
Início
    Instruções;
    Atualizar a variável de controle;
Fim

Lembrando que chamamos de variável de controle a variável testada na condição.

Para que seja possível fazer o teste, a variável de controle deve ter sido inicializada previamente.

Observe que o teste da condição ocorre no início do laço, enquanto a condição permanecer verdadeira, são executadas as instruções.

Quando a condição se tornar falsa, o processamento será desviado para fora do laço.

```
int contador = 1; //declarando e inicializando a variável de controle

while (contador <= 10) // Testando a condição
{
      printf("%d ", contador); //Executando um comando dentro do laço

      contador++; //atualizando a variável de controle
}
```

Figura 22 - Exemplo laço while

Na figura 22, temos uma variável do tipo int, a qual começa em 1. No loop, enquanto o contador for menor ou igual a 10, ele realizará as instruções da estrutura de repetição, que neste caso imprime o contador, em seguida incrementa o contador.

Note que o diferente da estrutura de repetição for, o incremento é feito dentro das instruções do while, se o incremento não for feito, o programa cairá no que chamamos de loop infinito, que é quando a instrução é feita infinitamente e consequentemente isso acarretará em estouro de tempo.

6.3 Do while

Laço semelhante ao while, porém ele primeiro executa um bloco e testa a condição, caso seja falsa vai para a próxima instrução.

Essa estrutura de repetição, garante que o bloco de instruções seja executado no mínimo uma vez, já que a condição que controla o laço é testada apenas no final do comando.

A diferença entre o comando **while** e **do while** é justamente o local onde a condição que controla o laço é testada.

Utilizaremos o mesmo exemplo do **while** para mostrarmos a diferença entre os dois.

```
int contador = 1;

do
{
    printf("%d\n", contador);
    contador++;
} while(contador <= 10);
```

Figura 23 - Laço do while

Diferente do **while** o **do while** executa as instruções primeiro e somente depois verifica a condição de repetição.

O laço **while** é comumente mais utilizado que o laço **do while**, porém ambos são extremamente úteis.

7 Switch case

O comando **switch** permite selecionar uma entre várias alternativas. Ele consiste da palavra-chave **switch** seguida do nome de uma variável ou de um valor numérico constante entre parênteses.

A expressão entre parênteses após a palavra-chave **switch** determina para qual caso será desviado o controle do programa.

O corpo de cada caso é composto por qualquer número de instruções. Geralmente, a última instrução é **break**, o que causa a saída imediata de todo o corpo do **switch.**

Na falta do comando **break**, todas as instruções, a partir do caso escolhido até o término do comando, serão executadas, mesmo sendo pertencentes aos casos seguintes.

O comando **break** tem somente dois usos em C: em laços ou no comando **swtich.**

Sintaxe:
Escolha (variável)
Caso a:
Soma = x + y
Pare
Caso b:
Subtração = x – y
Pare

Acima, vemos a sintaxe do **switch** onde:

- Escolha: no pseudocódigo significa **switch** em português
- Caso: em C é escrito **case**, caso o usuário escolha a, faz a soma entre dois números (x e y), se o usuário escolher b, é feita uma subtração de x e y.
- Pare: em C é comando **break**, significa que feita o caso acima, o programa imediatamente sairá do **switch case.**

```
{ char op;
  float a,b,c;
  printf("Escolha o operador:\n");
  scanf("%c",&op);
  printf("entre com o valor de a:\n");
  scanf("%f",&a);
  printf("entre com o valor de b:\n");
  scanf("%f",&b);

  switch (op)
  {
        case '*': c = a * b;
        printf("%.2f\n", c);
        break;
        case '/': c = a /b;
        printf("%.2f\n", c);
        break;
```

Figura 24 - Switch case exemplo

Na página anterior há um exemplo de switch case, onde o usuário escolhe o operador que deve utilizar, em seguida coloca um valor para **a** e para **b.**

De acordo com o operador informado pelo usuário, o programa executará o **case** que se encontra dentro do **switch.**

Portanto, se o usuário escolher ***** o programa executará a linha c = a * b, depois imprimirá o valor de c com precisão de duas casas decimais e em seguida sairá do comando **switch,** pois na linha de baixo há o comando **break.**

Referências Bibliográficas

[1] Tecmundo – Ana Paula Pereira. **O que é algoritmo?, 2009.** Disponível em: <https://www.tecmundo.com.br/programacao/2082-o-que-e-algoritmo-.htm>. Acesso em: 06. Jul 2017

[2] Pontifícia Universidade Católica. **Algoritmos e Pseudocódigo.** Disponível em: <http://www.inf.puc-rio.br/~inf1005/material/slides/backup/2010_2/tema02_algoritmos.pdf>. Acesso em: 06. Jul 2017

[3] Embarcados – Elaine Cecília Gatto. **Pseudocódigo.** Disponível em: <https://www.embarcados.com.br/pseudocodigo/>. Acesso em: 17. Jul 2017

[4] Fabricio Ferrari, Cristian Cechinel. **Introdução a algoritmos e programação.** Disponível em: <http://www.ferrari.pro.br/home/documents/FFerrari-CCechinel-Introducao-a-algoritmos.pdf>. Acesso em: 18. Jul 2017

[5] Universidade Federal do Rio Grande do Norte. **Algoritmo e Lógica de Programação.** Disponível em: <http://www.dca.ufrn.br/~affonso/DCA800/pdf/algoritmos_parte1.pdf>. Acesso em: 18. Jul 2017

[6] Conceito .de. **Conceito de computador, 2012.** Disponível em: <http://conceito.de/computador>. Acesso em: 18. Jul 2017

[7] Produção virtual. **Arquitetura de um Computador.** Disponível em: <http://producao.virtual.ufpb.br/books/camyle/introducao-a-computacao-livro/livro/livro.chunked/ch04s01.html>. Acesso em: 18. Jul 2017

[8] Tecmundo – Gabriel Gugik. **A história dos computadores e da computação, 2009.** Disponível em: <https://www.tecmundo.com.br/tecnologia-da-informacao/1697-a-historia-dos-computadores-e-da-computacao.htm>. Acesso em: 01. Ago 2017

[9] Ryerson. **Ábaco: Breve introdução, 2004.** Disponível em: <http://www.ee.ryerson.ca/~elf/abacus/portugues/intro.html>. Acesso em: 01. Ago 2017

[10] Hestoria do pc. **Régua de Cáclulo, 2011.** Disponível em: <https://hestoriadopc.wordpress.com/2011/06/24/regua-de-calculo/>. Acesso em: 01. Ago 2017

[11] Hestoria do pc. **Máquina de Pascal, 2011.** Disponível em: <https://hestoriadopc.wordpress.com/2011/06/27/maquina-de-pascal/>. Acesso em: 01. Ago 2017

[12] Ufpa. **História da informática e da internet.** Disponível em: <http://www.ufpa.br/dicas/net1/int-h180.htm>. Acesso em: 01. Ago 2017

[13] Tipografos. **Charles Babbage, 2012.** Disponível em: <http://www.tipografos.net/internet/babbage.html>. Acesso em: 01. Ago 2017

[14] A computação. **Engenho Analitico, 2008.** Disponível em: <http://a-computcao.wikidot.com/engenho-analitico>. Acesso em: 01. Ago 2017

[15] De boa na rede. **Máquinas de Tabulação, 2014.** Disponível em: <http://dboanarede.blogspot.com/2014/08/maquinas-de-tabulacao.html>. Acesso em: 01. Ago 2017

[16] Produção virtual. **As gerações dos computadores.** Disponível em: <http://producao.virtual.ufpb.br/books/camyle/introducao-a-computacao-livro/livro/livro.chunked/ch01s02.html>. Acesso em: 01. Ago 2017

[17] Hestoria do pc. **A História dos Computadores, 2011.** Disponível em: <https://hestoriadopc.wordpress.com/2011/07/04/computacao-moderna-segunda-geracao-1959-1964/>. Acesso em: 01. Ago 2017
[18] Computer story. **Circuitos integrados.** Disponível em: <http://computerstory.weebly.com/3ordf-geraccedilatildeo---circuitos-integrados.html>. Acesso em: 01. Ago 2017
[19] Tutorials point. **Computador – Terceira geração, 2016.** Disponível em: <https://www.tutorialspoint.com/pg/computer_fundamentals/computer_third_generation.htm>. Acesso em: 01. Ago 2017
[20] Fundação bradesco. **Microinformática.** Disponível em: <http://www.fundacaobradesco.org.br/vv-apostilas/mic_pag3.htm>. Acesso em: 01. Ago 2017
[21] Evolução da informática. **4ª Geração: Microprocessador (1971-1980).** Disponível em: <http://aevolucaoinformatica.blogspot.com/p/4-geracao.html>. Acesso em: 01. Ago 2017
[22] Introdução a informática. **5ª Geração.** Disponível em: <http://introducao-a-informatica.webnode.com/a5%C2%AA-gera%C3%A7%C3%A3o/>. Acesso em: 01. Ago 2017
[23] PUCRS. **Introdução a Linguagem C.** Disponível em: <http://www.inf.pucrs.br/~pinho/LaproI/IntroC/IntroC.htm>. Acesso em: 15. Ago 2017
[24] Silvio do Lago Pereira. **Linguagem C.** Disponível em: <https://www.ime.usp.br/~slago/slago-C.pdf>. Acesso em: 15. Ago 2017
[25] PUCRS. **Histórico da Linguagem C.** Disponível em: <http://www.inf.pucrs.br/~pinho/LaproI/Historico/Historico.htm>. Acesso em: 15. Ago 2017
[26] Aprendendo a Programar em Linguagem C do básico ao avançado – Alfredo Boente. **Editora Brasport, 2003.**
[27] Universidade Tecnológica Federal do Paraná – Rodrigo André Soster. **Redução de oscilações de frequência com o uso de carga "dump" em micro centrais elétricas isoladas, 2012.** Disponível em: <http://repositorio.roca.utfpr.edu.br/jspui/bitstream/1/4091/1/PB_COELT_2012_1_04.pdf>. Acesso em: 16. Ago 2017
[28] Unicamp. **Compiladores, 2003.** Disponível em: <http://www.dca.fee.unicamp.br/cursos/EA876/apostila/HTML/node37.html>. Acesso em: 15. Ago 2017
[29] Academia – José Rodrigues. **Introdução a Programação, 2013.** Disponível em: <http://www.academia.edu/31094557/Introdu%C3%A7%C3%A3o_%C3%A0_Programa%C3%A7%C3%A3o_IP>. Acesso em: 16. Ago 2017
[30] Universidade Metodista de Angola – Gustavo Sebastião. **Estrutura de um programa em linguagem C.** Disponível em: <http://web.ist.utl.pt/ist153068/ficheiros/teoricas/Programacao_I_Cap_2_Estrutura_basica_de_um_programa_em_linguagem_C.pdf>. Acesso em: 16. Ago 2017
[31] Standardlibs. **Bibliotecas entandardizadas.** Disponível em: <http://www.di.ubi.pt/~operativos/praticos/pdf/6-standardlibs.pdf>. Acesso em: 16. Ago 2017

[32] Univasf – Críston. **Linguagem C: diretivas, compilação separada.** Disponível em: <http://www.univasf.edu.br/~criston.souza/algoritmos/arquivos/aula16.pdf>. Acesso em: 16. Ago 2017
[33] Tutorials point. **C Library - <stdio.h>.** Disponível em: <https://www.tutorialspoint.com/c_standard_library/stdio_h.htm>. Acesso em: 16. Ago 2017
[34] Programando – Guilherme Carvalho. **Para que servem as bibliotecas <stdio.h> e <stdlib.h>, 2012.** Disponível em: <http://programando-ads.blogspot.com/2012/02/para-que-servem-as-bibliotecas-e.html>. Acesso em: 16. Ago 2017
[35] Wiki para programação em C. **Comentários, 2009.** Disponível em: <http://www.br-c.org/doku.php?id=comentarios>. Acesso em: 17. Ago 2017
[36] Univasf – Leonardo Barreto Campos. **Linguagem C: Variáveis e Operadores.** Disponível em: <http://www.univasf.edu.br/~leonardo.campos/Arquivos/Disciplinas/Alg_Prog_I_2007_1/Aula_02.pdf>. Acesso em: 30. Ago 2017
[37] Linguagem C – Eduardo Casavella. **Funções e escopo de variáveis.** Disponível em: <http://linguagemc.com.br/funcoes-e-escopo-de-variaveis/>. Acesso em: 23. Set 2017
[38] Master da web – Lucas Viana. **Variáveis globais e locais – Linguagem C/C++, 2011.** Disponível em: <http://blog.masterdaweb.com/programacao-1/linguagem-c/variaveis-globais-e-locais-linguagem-cc/>. Acesso em 23. Ago 2017
[39] Equipe NCE – Adriano Joaquim de Oliveira Cruz. **Tipos de dados, constantes e variáveis, 1997.** Disponível em: <http://equipe.nce.ufrj.br/adriano/c/apostila/tipos.htm>. Acesso em: 23. Ago 2017
[40] PUCRS – Márcio Sarroglia Pinho. **Introdução a Linguagem C.** Disponível em: <http://www.inf.pucrs.br/~pinho/LaproI/IntroC/IntroC.htm>. Acesso em: 27. Ago 2017
[41] Linguagem C – Eduardo Casavella. **Constantes em C usando #define.** Disponível em: <http://linguagemc.com.br/constantes-em-c-usando-define/>. Acesso em: 27. Ago 2017
[42] Julio Battisti. **Linguagem C – Operadores.** Disponível em: <https://juliobattisti.com.br/tutoriais/katiaduarte/cbasico002.asp>. Acesso em: 27. Ago 2017
[43] Wikipedia. **Sieve of Eratosthenes, 2017.** Disponível em: <https://en.wikipedia.org/wiki/Eratosthenes>. Acesso em: 28. Ago 2017
[44] Marathoncode. **Crivo de Erastóstenes, 2014.** Disponível em: <http://marathoncode.blogspot.com.br/2012/03/numeros-primos-iii.html>. Acesso em 01. Set 2017
[45] Wikipedia. **Crivo de Erastóstenes, 2017.** Disponível em: <https://pt.wikipedia.org/wiki/Crivo_de_Erat%C3%B3stenes>. Acesso em: 01. Set 2017
[46] BlogDeCodigo – Edson Pessoa. **Linguagem C – Bibliotecas, Funções main, printf, scanf e mais – Drops I, 2011.** Disponível em: <https://blogdecodigo.wordpress.com/2011/07/12/linguagem-c-bibliotecas-funcoes-main-printf-scanf-e-mais-drops-i/>. Acesso em: 04. Set 2017
[47] Php Brasil – Adhemar Zerlotini Neto. **Os 4 tipos de erros em programação, 2003.** Disponível em: <http://www.phpbrasil.com/artigo/yAVSjjsFq2vR/os-4-tipos-de-erros-em-programacao>. Acesso em: 10. Set 2017
[48] Terra. **Overflow, 2014.** Disponível em: <http://tecnologia.terra.com.br/noticias/0,,OI500595-EI15607,00-Overflow.html>. Acesso em 10. Set 2017

[49] Viva o Linux – Cicero Julião da Silva Junior. **Tomada de decisão, 2008.** Disponível em: <https://www.vivaolinux.com.br/artigo/Programacao-Tomada-de-decisao>. Acesso em: 14. Set 2017
[50] DevMedia – Juliano Schimiguel. **Fluxogramas, diagrama de blocos e de Chapin no desenvolvimento de algoritmos, 2013.** Disponível em: <http://www.devmedia.com.br/fluxogramas-diagrama-de-blocos-e-de-chapin-no-desenvolvimento-de-algoritmos/28550>. Acesso em: 23. Set 2017
[51] Slideshare – Aislan Rafael. **Algoritmo e Programação, 2007.** Disponível em: <https://pt.slideshare.net/aislan/aula-04-logica-de-programacao>. Acesso em: 23. Set 2017
[52] DevMedia – Alisson Santos. **Estrutura de Repetição: C++.** Disponível em: <http://www.devmedia.com.br/estrutura-de-repeticao-c/24121>. Acesso em: 23. Set 2017
[53] Linguagem C – Eduardo Casavella. **O comando for em C.** Disponível em: <http://linguagemc.com.br/a-estrutura-de-repeticao-for-em-c/>. Acesso em: 23. Set 2017
[54] Linguagem C – Eduardo Casavella. **O comando while em C.** Disponível em: <http://linguagemc.com.br/o-comando-while-em-c/>. Acesso em: 23. Set 2017
[55] Linguagem C – Eduardo Casavella. **Comando do while em C.** Disponível em: <http://linguagemc.com.br/comando-do-while/>. Acesso em: 23. Set 2017
[56] Treinamento em Linguagem C 2ª Edição – Victorine Viviane Mizrahi. São Paulo. Editora Pearson. 2008.