

1.Introdução a Linguagem C

1.1 Perspectiva

O mundo da programação permite desenvolver qualquer aplicação desejada, abrindo um leque de possibilidades e oportunidades de trabalho, porém introduzir o mundo computacional pode ser uma tarefa árdua e muitas vezes sem direcionamento. O seguinte material propõe introduzir a linguagem C, muito utilizada no ambiente educacional e de fácil acesso, com finalidade apresentar a lógica e sintaxe da linguagem para os interessados, para direcioná-los

A linguagem C foi desenvolvida como ferramenta em meados de 1970, no qual seus programas eram voltados a setores bancários, científicos e de engenharia. Atualmente, essa linguagem encontra-se na posição educacional por seu aspecto lógico, funcionando como uma ótima porta de entrada aos interessados em introduzir ao mundo das linguagens de computação.

1.2 Conceitos

Os fundamentos da linguagem C baseiam-se em saber comunicar com o computador de uma maneira uniforme e demonstrar claramente as instruções para o mesmo. Um programa tem como finalidade resolver um problema, seja ele fictício ou real. É necessário então estabelecer um sentido para o problema e definir as etapas para sua solução dentro de um algoritmo. Para inicializar um programa em C é preciso criar um código fonte que será escrito em uma interface de desenvolvimento. Após isso, é necessário compilar(traduzir a linguagem humana para a linguagem computacional binária)essa sequência de instruções escritas no código para assim gerar um arquivo executável que realiza as operações indicadas pelo desenvolvedor.

1.3 Algoritmos

A finalidade deste capítulo é introduzir o conceito e o formato de representação dos algoritmos. O significado de algoritmo para a informática é: “um conjunto de regras e procedimentos lógicos perfeitamente definidos que levam à solução de um problema em um número finito de etapas.”

Antes de resolver um problema por códigos, é importante descrevê-lo de forma clara, precisamos definir um passo a passo de como será feita a resolução do problema, então usamos os algoritmos.

Os algoritmos não são limitados apenas para a informática ou programação, por exemplo, uma planilha de treino para a academia é um algoritmo, nós seguimos uma sequência de passos para conseguir realizar um treino completo.

1.3.1 Construindo um algoritmo

O algoritmo a seguir é usado para fazer uma média das notas dos 4 bimestres de um aluno e exibir a mensagem “Aprovado” caso a média for maior ou igual à 60 e “Reprovado” caso contrário, nele, o usuário precisa informar quatro números de 0 a 100 como entrada das notas (n_1 , n_2 , n_3 , n_4), o programa então deve realizar a divisão dessas variáveis por 4 para obter o valor da média, essa divisão também precisa ser guardada em uma variável, nesse caso vamos chamá-la de media. Então, no final do programa, precisamos exibir para o usuário qual foi essa média

1.4 Interface de desenvolvimento

Para desenvolver um programa em C é necessário uma interface de desenvolvimento. O programa escolhido chama-se *CodeBlocks*, uma plataforma amplamente conhecida no mundo educacional principalmente por sua facilidade de acesso. Segue abaixo os links para download do *CodeBlocks* atualizado:

Windows:

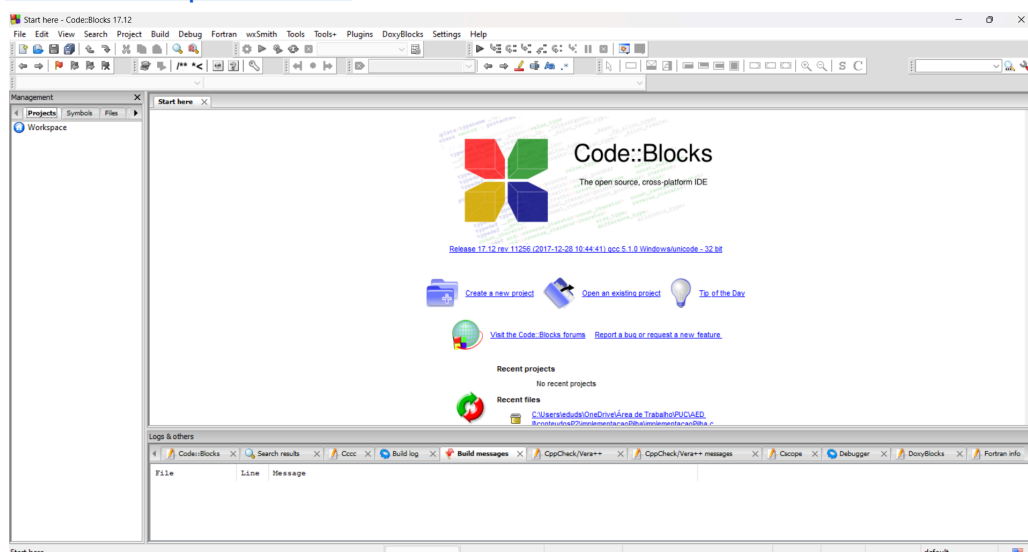
<https://sourceforge.net/projects/codeblocks/files/Binaries/20.03/Windows/codeblocks-20.03-setup.exe/download>

Linux:

https://sourceforge.net/projects/codeblocks/files/Binaries/20.03/Linux/Debian%209/c%20odeblocks_20.03_amd64_oldstable.tar.xz/download

MacOS:

<https://sourceforge.net/projects/codeblocks/files/Binaries/13.12/MacOS/CodeBlocks-13.12-mac.zip/download>



Ao inicializar o programa, essa será a tela inicial. Para começar um código novo, basta clicar no botão *File*(encontrado no canto esquerdo superior)e depois em *New Empty File*. Agora a interface já está preparada para receber um código fonte.

1.5 Estrutura de um código em C

Um código em C precisa de alguns direcionamentos antes de começar a digitar. No topo do arquivo fonte é necessário incluir as bibliotecas que serão utilizadas no código. A mais popular e mais utilizada é a *stdio.h*(entrada e saída padrão). Dentro das bibliotecas localizam-se as expressões que serão utilizadas dentro do código. Para incluir basta digitar: *#include <stdio.h>*.

Em seguida é necessário instanciar a função *main*, que é a área principal de um código em C. Muitas funções(incluídas nas bibliotecas) serão executadas dentro do *main*, tais como *printf*(exibir um texto desejado na saída padrão) e *scanf*(receber uma entrada de um tipo específico do usuário). O *main* é inicializado e finalizado com chaves, no qual todo seu conteúdo ficará entre esses sinais. Após digitar, é necessário salvar o arquivo com extensão *.c* para que o computador reconheça que é de fato um arquivo de código fonte. Depois disso, basta clicar no botão *build and run* e será criado um arquivo executável do programa digitado.



botões *build*, *run* e *build and run*

É necessário lembrar que após digitar uma instrução, deve-se terminá-la com “;”(ponto e vírgula), tal como o exemplo abaixo:

```
1  #include <stdio.h>
2
3  main() {
4      printf("Hello World!");
5
6  }
```

A função *printf* exibe na tela de saída(chamada terminal) um texto desejado. Nota-se que após digitar o nome da função é necessário abrir parênteses e dentro dos parênteses digitar o texto desejado a ser exibido entre aspas duplas.

```
Hello world!  
Process returned 0 (0x0)   execution time : 0.038 s  
Press any key to continue.  
|
```

1.6 Variáveis

Para resolver um problema, é necessário atribuir alguns valores para chegar a uma solução. Assim, dentro do ambiente de C, chama-se de variável um elemento que em algum momento será atribuído um valor a ele. Existem alguns tipos padrões de variáveis, são eles:

- Inteiro(int): Valores numéricos sem parte fracionária
- Float(float): Valores numéricos com parte fracionária
- Caractere(char): Símbolos em geral captados pelo teclado

Para inicializar uma variável devemos colocar seu tipo e em seguida um nome que será usado para defini-la durante o código. Ao usar a variável do tipo *char* utiliza-se apenas um caractere captado pelo teclado. Para captar mais de um, deve-se iniciá-la com o número de elementos dentro de colchetes. Uma sequência de caracteres é chamada de *string*.

```
int idade;  
float salario;  
char nome[50];
```

Para demonstrar o valor de uma variável na saída deve-se utilizar sua assinatura, que é como a memória do computador a identifica na hora de processar os dados. São elas:

- int: %d
- float: %f
- char(apenas um): %c
- char(string): %s

A partir do que foi mostrado, podemos desenvolver um programa simples, tal como o de exibir o nome digitado pelo usuário como entrada. Para ler o nome é necessário usar a função *scanf* e dentro de parênteses especificarmos o tipo da variável e que ela será endereçada na memória como foi declarada anteriormente. Para endereçá-la, deve-se acrescentar um “&” antes de sua designação.

```
#include <stdio.h>

main() {
    int idade;
    float salario;
    char nome[50];

    printf("Qual o seu nome? \n");
    scanf("%s", &nome);
    printf("Ola, %s", nome);
}
```

Para exibir o nome digitado, usa-se novamente a função *printf* e dentro de parênteses escrever o texto desejado entre aspas duplas e utilizando a assinatura da variável na posição de exibição desejada e, após fechar as aspas, coloca-se uma vírgula e o nome da variável que foi captada pela entrada do usuário. Nota-se o uso de “\n” para pular uma linha para que o próximo texto seja exibido abaixo.

Atenção: A linguagem C baseia-se no idioma inglês, que não possui acentos, sinais gráficos e “ç”, então, o compilador não entende esses caracteres nativamente. Para utilizar caracteres do idioma brasileiro é necessário a inclusão da biblioteca “locale.h” e dentro do main utilizar a função “setlocale(LC_ALL, “Portuguese”);”

```
Qual o seu nome?
Pedro
Ola, Pedro!

Process returned 0 (0x0)   execution time : 2.143 s
Press any key to continue.
```

No caso acima, “Pedro” foi a entrada do usuário que foi exibida posteriormente na tela. Nota-se que mesmo instanciando a variável com 50 caracteres, pode-se utilizar menos do que foi estabelecido, porém não mais.

```

#include <stdio.h>

main() {
    int idade;
    float salario;
    char nome[50];

    printf("Qual o seu nome?\n");
    scanf("%s", &nome);
    printf("Qual sua idade?\n");
    scanf("%d", &idade);
    printf("%s tem %d anos\n", nome, idade);
}

```

Nota-se neste exemplo o uso de uma variável inteira, cuja assinatura é %d.

```

Qual o seu nome?
Pedro
Qual sua idade?
21
Pedro tem 21 anos

Process returned 0 (0x0)   execution time : 5.296 s
Press any key to continue.
|

```

1.7 Operadores matemáticos

A partir do que foi demonstrado, podemos acrescentar operadores matemáticos para realizar alguns cálculos simples. São eles:

- Adição (+)
- Subtração (-)
- Multiplicação (*)
- Divisão (/)

Agora, podemos realizar o problema definido pelo algoritmo do cálculo de média de notas.

```

#include <stdio.h>

main() {
    float n1,n2,n3,n4,media;

    printf("Digite as notas: \n");
    scanf("%f",&n1);
    scanf("%f",&n2);
    scanf("%f",&n3);
    scanf("%f",&n4);

    media = (n1+n2+n3+n4)/4;

    printf("media: %f",media);
}

```

Nota-se que a variável `media` foi atribuída a um valor baseado em uma operação matemática. Para atribuir deve-se colocar um sinal de "=", tal como o exemplo acima. Nota-se também que dentro de uma operação matemática podem ter cálculos prioritários semelhantes aos de matemática. Utiliza-se parênteses para tal designação.

1.8 Operadores Condicionais

Dentro de um código, podemos ter condições para que algo aconteça ou não. Utilizando de exemplo o algoritmo de média, nota-se que exibe uma mensagem de "Aprovado" caso a média seja maior que 6 e caso não, seja exibido "Não aprovado". Para isso, utiliza-se operadores condicionais, conhecidos como if-else.

- if(se): caso sim
- else(senão): caso contrário

Ao utilizar if e else deve-se colocar a condição entre parênteses e o que ocorre na condição entre chaves. Condições estão relacionadas a operadores relacionais, que são:

- >(maior)
- <(menor)
- >=(maior ou igual)
- <=(menor ou igual)
- ==(igual)
- !=(diferente)

```

if (media > 6) {
    printf("Aprovado");
}
else {
    printf("Reprovado");
}

```

Pode-se ainda, colocar dentro de um if mais de uma condição, utilizando operadores lógicos, que são:

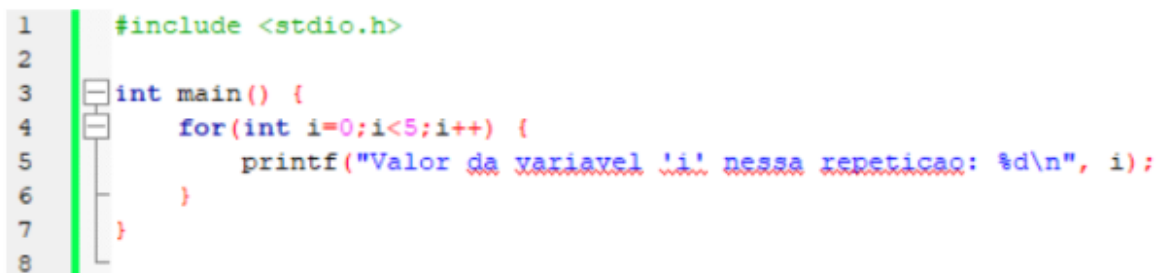
- && and(e)
- || or(ou)
- ! not(não)

Assim, a condicional será afetada diretamente pelos operadores, sendo que AND é necessário que todas as condições sejam verdadeiras, OR apenas uma e NOT apenas o contrário.

1.9 Estrutura de Repetição

Em C, também temos as estruturas de repetição, essas estruturas permitem que um trecho do código repita um determinado número de vezes, podendo ser fixo ou depender de condições que mudam durante a execução dessa repetição.

1.9.1 Comando FOR

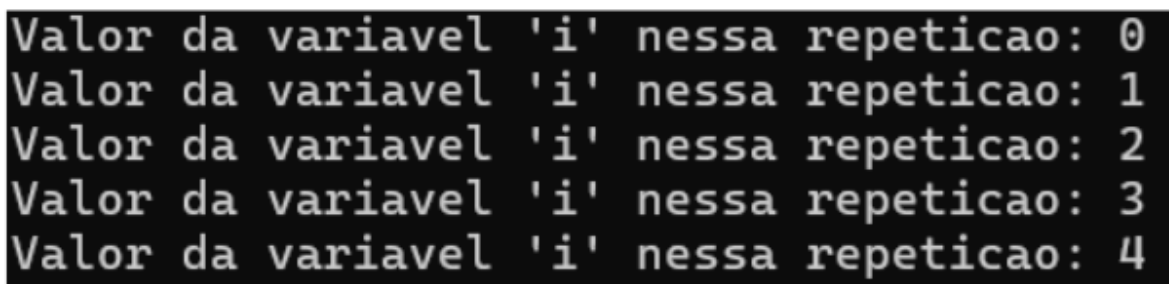


```

1  #include <stdio.h>
2
3  int main() {
4      for(int i=0; i<5; i++) {
5          printf("Valor da variavel 'i' nessa repeticao: %d\n", i);
6      }
7  }
8

```

Imagem 2.1.1 - Representação de uma estrutura for em código.



```

Valor da variavel 'i' nessa repeticao: 0
Valor da variavel 'i' nessa repeticao: 1
Valor da variavel 'i' nessa repeticao: 2
Valor da variavel 'i' nessa repeticao: 3
Valor da variavel 'i' nessa repeticao: 4

```

Imagem 2.1.2 - Saída da imagem 2.1.1.

Na imagem 2.1 está um exemplo simples de código de uma estrutura for, onde podemos ver que, mesmo a printf sendo escrita apenas uma vez em código, aparece 5 vezes na saída, como mostrado na imagem 2.2.

A linha 4 do código é como o comando for é chamado: primeiro escolhemos uma variável que será um contador para decidir quantas vezes aquele trecho de código dentro do for será executado, no exemplo acima escolhemos a variável `i` do tipo inteiro (perceba que essa variável for declarada dentro do for, ou seja, ela só poderá ser utilizada ali dentro do for, não existindo mais fora dele. Fazemos isso quando queremos usar uma variável exclusivamente para aquela ocasião), depois de declarar a variável como contador, escolhemos o valor que ela vai começar, definimos como 0 e essa foi nossa primeira expressão.

As expressões no for são separadas por “;”. A segunda expressão é o critério de parada da repetição, no caso escolhemos como sendo menor que 5, ou seja, quando `i` estiver valendo 5 ou um valor maior, a repetição será encerrada, continuando o código.

A terceira e última expressão é necessária para somar o contador e evitar que a repetição continue infinitamente, então depois que o trecho do for chegar no fim, a variável `i` vai somar mais 1, podemos fazer ao contrário e subtrair 1 a cada repetição, utilizando `i--`, ou então somar/subtrair mais valores por vez, usando `i+=2` ou `i-=2`. As chaves depois do parênteses são para informar quais linhas fazem parte da repetição, ela fecha na linha 6.

1.9.2 Comando WHILE

```
1  #include <stdio.h>
2
3  int main() {
4      int i;
5
6      i = 1;
7
8      while (i <= 10) {
9          printf ("Numero %d\n", i);
10         i++;
11     }
12
13     return 1;
14 }
```

Imagem 2.2.1 - Exemplo de código utilizando *while*.

```
Numero 1
Numero 2
Numero 3
Numero 4
Numero 5
Numero 6
Numero 7
Numero 8
Numero 9
Numero 10

Process returned 1 (0x1)   execution time : 0.036 s
Press any key to continue.
```

Imagem 2.2.2 - Saída da imagem 2.2.1.

Na imagem 2.2.1 vemos como a estrutura while é inserida em código, sendo:

```
while(expressão){  
    comandos }
```

Primeiro, a expressão é avaliada, se for verdadeira, então o bloco de comando é executado, caso contrário a execução do código é terminada. Após a execução do código, volta para o primeiro passo.

1.9.3 Comando DO WHILE

Muito semelhante ao while, a estrutura do while é a seguinte:

```
do {  
    comandos  
} while(expressão)
```

A diferença entre while e do while é que o segundo é executado pelo menos uma vez, ele executa o comando e somente depois avalia se a expressão é verdadeira, e se for volta para o primeiro passo, caso contrário ele interrompe o do while e continua o código.

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

Imagem 2.3.1 - Exemplo de código utilizando do while.