Consulta simplificada para Prova 1 de *Programação de Computadores I*

Igor C. Guimarães [1]

[1]Instituto de Ciencias Exatas e Aplicadas – Universidade Federal de Ouro Preto (UFOP)

igor.cg@aluno.ufop.edu.br

Resumo. *Este guia de consulta da disciplina Programação de Computadores I serve como base para entender a correta utilização de funções e loops para a primeira prova e talvez para posteriores.Programação requer prática e persistência, nem sempre os programas terão o funcionamento esperado e teste de código é altamente recomendado.*

Sumário

1. Dicas úteis

Durante a prova, procure identificar quais serão as variáveis e se o problema irá envolver condicionais ou loops.
Um problema de loop sempre terá escrito *(até que o usuário **digite X palavra para sair**)* e um exercício que envolve condicional geralmente terá *(**caso X** > **imprima..**)*

Prova em sala **você não terá como testar**, logo sua única forma de verificação será o teste de mesa. No teste de mesa você deve utilizar as variáveis criadas durante o código e analisar a mudança das mesmas e como isso pode afetar o resto da execução do código.

Diagramas de Chapin e Fluxograma podem te ajudar na parte lógica do programa, se tiver tempo, faça-os.

Não se esqueça do ";"sempre após declaração de variáveis e em alguns outros momentos durante o código.

O while precisa de um contador para percorrer o loop, ele não se incrementa sozinho se comparado ao for.

As variáveis em C são tipadas, ou seja, requer que seja especificado o **tipo** da variável (Pag:5)

2. Nomenclatura de variáveis

Em C as variáveis requerem tipagem, ou seja, requer que se escreva 'int', 'float' ou 'char' para as mesmas funcionarem.
Exemplo de variáveis que devem funcionar:

- igorcg
- igor123
- AlUnO_11

Exemplo de variáveis que **não** devem funcionar:

- 123igor_cg (começar com números)
- 123-foo (uso de hífen)

Nomes de variáveis **proibidas**:

- auto
- double
- int
- struct
- break
- long
- else
- switch
- case
- return
- enum
- typedef

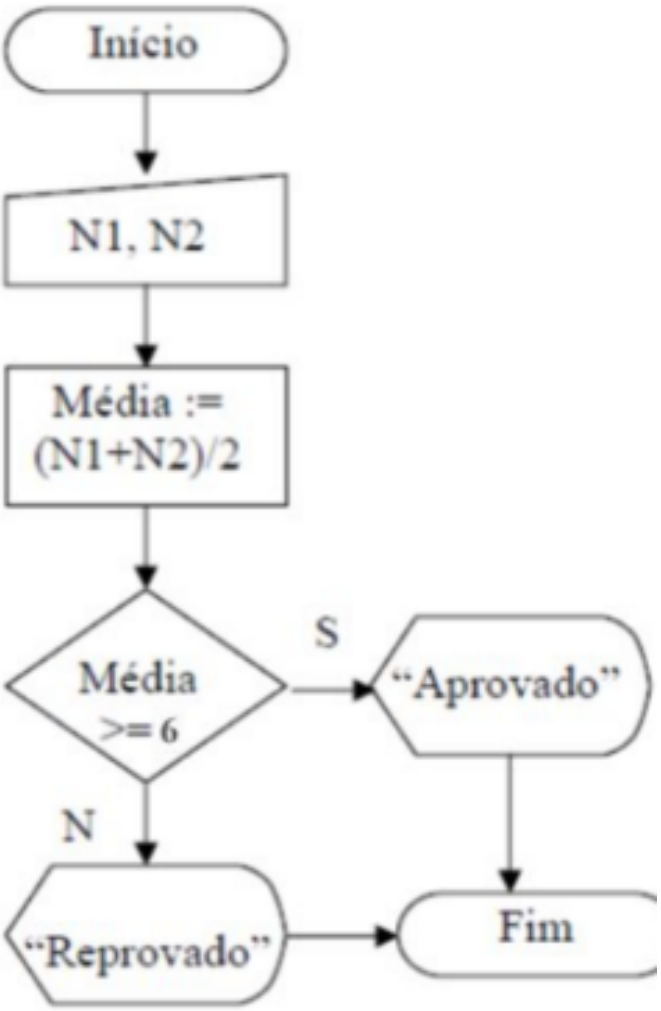

Figura 1. Fluxograma e sua representação

- char
- register
- extern
- union
- const
- short
- float
- unsigned
- continue
- signed
- for
- void
- default
- sizeof

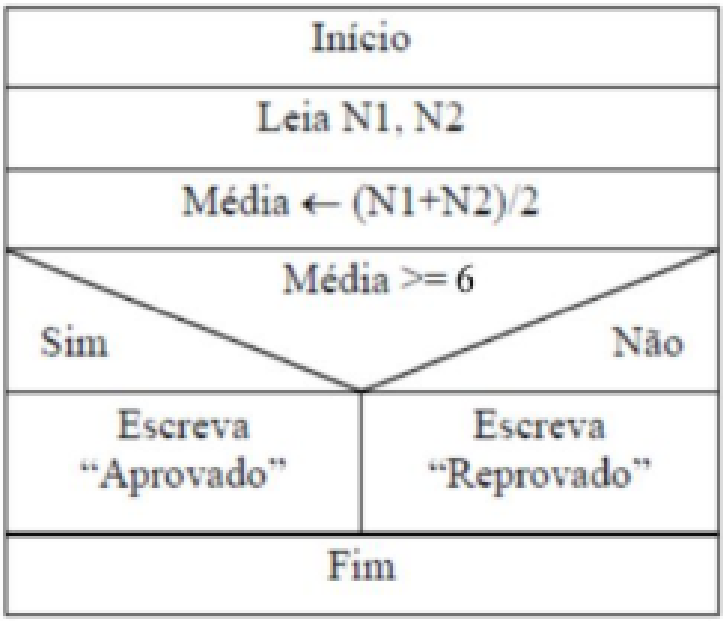

Figura 2. Diagrama de Chapin

- goto
- volatile
- do
- static
- if
- while

Tipos de variáveis

- **int**: Inteiro (positivo ou negativo).
- **float**: Número com partes fracionadas.
- **double**: Número com partes fracionadas de maior precisão.
- **char**: Apenas um caractere.

Para números com partes fracionadas, deve-se utilizar "."(Vírgula não terá o funcionamento esperado)
O tipo booleano (verdadeiro ou falso) é representado por inteiro sendo:

- 0 : Falso
- 1 ou valor diferente de 0: Verdadeiro

Exemplo de uso das variáveis

```
int iNumero = 9090;
char cLetra = 'c';
float fGrande = 9090.1234;
```

De forma geral, variáveis sempre devem começar com letra e evitando o uso de hífen em caso de múltiplas palavras.

3. Operadores

Algumas das operações possíveis a serem realizadas com as variáveis são:

- +, adição
- -, subtração
- *, multiplicação
- /, divisão
- %, resto da divisão

Alguns atalhos para operações:

```
int iNumero += 10;
```

é equivalente a

```
int iNumero = iNumero +10;
```

O mesmo vale para as outras operações.

3.1. Incremento e Decremento

Geralmente em uso de loops são utilizados incrementos ou decrementos para alcançar o final das iterações desejado. Ex

- iNumero++ (Número sera adicionado o valor 1 após a sua leitura).
- ++iNumero (Número sera adicionado o valor 1 antes de sua leitura).
- iNumero−− (Número sera subtraído o valor 1 após a sua leitura).
- −−iNumero (Número sera subtraído o valor 1 antes de sua leitura).

4. Interação com Usuário

Na linguagem C, existem algumas funções para interagir com usuário e cada uma delas possui um foco específico.

Printf

- Imprimir mensagens para o usuário
- Exibir valores de variáveis
- Exibir ao usuário que o programa está respondendo
- Pode conter filtros para impressão de dados.
  Exemplo: Imprimir apenas duas casas decimais de um número float:

```
float fNumero = 99.9999;
printf("%.2f",fNumero); //99.99
```

[1]

Scanf

- Solicita valores de variáveis

[1] Utilizar printf e scanf de forma inteligente, evita possíveis erros e até mesmo validações desnecessárias.

- %d para tipo inteiro
- %f para tipo float
- %c para tipo caractere
- %s para vetor de caractere (string e frases)

- Pode conter filtros para coleta de um número especifico de dados. Exemplo: Pegar apenas 2 dígitos de um número inteiro:

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

5. Funções e retornos

```
int calculaRaizInteira (int numero){
        return sqrt(numero);
}
//MAIN
int main(){
        int n = calculaRaizInteira(9);
}
```

No exemplo acima, o tipo da variável da chamada da função (no caso a main) deve possuir o mesmo tipo de retorno da função calculaRaizInteira (também int) para evitar conflitos de números imprecisos.

6. Operadores lógicos

Uma forma de compreensão mais fácil é por um artifício denominado Tabela Verdade.

A	B	A || B (A E B)
0	0	0
0	1	1
1	0	1
1	1	1

A	B	A & B (A E B)
0	0	0
0	1	0
1	0	0
1	1	1

A	!A (negação de A)
0	1
1	0

Resumo dos operadores lógicos
A comparação deve ser feita entre duas variáveis (& ou ||)

- (!) Inverte o valor do operador booleano(apenas pra 0 ou 1 como resposta). Se for 0, tornará 1. Se for 1, tornará 0.
- (&) Requer que **AMBOS** os valores comparados sejam verdadeiros, dessa forma a resposta será verdadeira.
- (||) Requer que **PELO MENOS** um dos valores comparados sejam verdadeiros, para que a resposta seja verdadeira.

```
int A = 1;
int inverso = !A; //Inverso tera o valor 0.

int A2 = 1, B2 = 0;
int AeB = A2 && B2; //AeB tera o valor 0.

int A3 = 1, B3 = 1;
int AouB = A3 || B3; //AouB tera o valor 1
```

7. Operadores relacionais

Geralmente utilizados como comparação de duas variáveis e retornando verdadeiro ou falso conforme o resultado da comparação.

- A <B (menor que)
- A <= (menor ou igual)
- A > B (maior que)
- A >= B (maior ou igual a)
- A == B (igual a)
- A $\neq$ B (diferente de)

Exemplo:

```
int A = 1, B = 2;
if(A <= B) //Retorna verdadeiro
  printf("A menor que B");
```

8. Condicionais

Na linguagem C, utilizamos o *if* e o *switch* para realizar operações que dependem de uma certa condição para serem iniciadas.

8.1. IF e Else

8.1.1. IF

```
int A = 1, B = 2;
if(A <= B) //Retorna verdadeiro
 printf("A menor que B");
```

O else if é semelhante ao if normal, porém só será executado se o *if* anterior não for verdadeiro.

8.1.2. Else

```
int A = 1, B = 2;
if(A > B) //Retorna verdadeiro
        printf("A maior que B");
else if(A == B)
        printf("A igual a B");
else
        printf("A menor que B");
```

8.2. Switch

```
int A = 1, B = 2;
switch(A) //Variavel a ser usada de comparacao
case (A>B)
        printf("A maior que B");
        break;
case (A == B)
        printf("A igual a B");
        break;
default: //Caso nenhuma das condicoes se encaixe
        printf("A menor que B");
```

9. Repetição

9.1. While

Executa um loop enquanto a condição de parada não for alcançada.Por padrão um contador é utilizado como condição para que as iterações chegue ao fim ou até mesmo um valor digitado pelo usuário para que se saia da região do loop. Ex: contador++

```
while ( condicao )
{
        Comandos a serem executados;
        IncrementaCondicao para alcancar a parada
}
```

9.2. Do While

Semelhante ao While, porém garante que será executado ao menos uma vez pela parte do código **do**.

```
do{
        Comandos a serem executados;
        IncrementaCondicao para alcancar a parada
}
while ( condicao );
```

9.3. For

Semelhante ao 9.1 porém com o diferencial que pode ser utilizado a inicialização, condição de parada e incremento em apenas um comando.

```
int i; //Necessario declaracao antes do for
for(i = 0; i¡10; i++){
        printf("%d ",i); //Ira imprimir de 0 a 9
}

for(inicio; parada; incre ou decre){

}
```

Referências