



Disciplina: Fundamentos da Computação		Visto:
Professor: Abrantes Araújo Silva Filho		
Aluno:		
Turma:	Semestre:	Valor: —
Data:	Diário 3: C	

DIÁRIO DE APRENDIZAGEM:

- Este **Diário de Aprendizagem** é uma das atividades integrantes da disciplina de **Fundamentos da Computação** do curso de Ciência da Computação, Universidade Vila Velha (UVV).
- A confecção do diário de aprendizagem é atividade **obrigatória e altamente recomendada** por três motivos: a) você aprenderá muito mais a matéria se mantiver o diário; b) ao entregar todos os diários ao professor você está cumprindo parte das atividades avaliativas que contam pontos na disciplina (10% da nota); e c) as provas bimestrais discursivas seguirão o formato e conteúdo das perguntas do diário.
- Se você tiver dificuldade em responder alguma questão do diário, estude novamente a matéria. Se você realmente entendeu a matéria, não deveria ter muita dificuldade para responder o diário.
- Responda com caneta ou lápis escuro (2B, 4B, 6B).
- Verifique no calendário de sua turma a **data de entreg**. Após uma rápida avaliação e visto pelo professor ou pelos monitores, seu diário será devolvido.
- O diário não será corrigido pelo professor: cabe a você estudar e dar a resposta correta para todas as questões. Obviamente o professor está à disposição para esclarecimento de dúvidas, e os monitores podem auxiliar caso você tenha dificuldade.
- Manter o diário de aprendizagem atualizado pode ser a diferença entre você aprender a matéria e ser aprovado, ou não aprender a matéria e não ser aprovado.
- Bons estudos!

Programação em C e pensamento computacional

1. Explique o que é uma **linguagem de alto nível** e cite três exemplos desse tipo de linguagem.

2. Explique o que é uma **linguagem de baixo nível** e cite dois exemplos desse tipo de linguagem.

3. O que é a **linguagem de máquina** (também chamada de **código de máquina**)?

4. O que é o **código fonte**? É a mesma coisa que linguagem de alto nível?

5. O que devemos fazer para transformar nosso **código fonte** em **linguagem de máquina**?

6. Sobre os **compiladores**, responda:

(a) O que é um compilador?

(b) Qual entrada um compilador recebe?

(c) Qual saída um compilador produz?

(d) Cite dois compiladores para a linguagem C.

7. Podemos avaliar a **qualidade** do código de um programa através de alguns critérios como **corretude, design, estilo e eficiência**.

(a) O que é a corretude de um código?

(b) O que é o design de um código?

(c) O que é o estilo de um código?

(d) O que é a eficiência de um código?

8. Seu chefe pediu para que você fizesse um programa, em C, que solicita o nome de uma pessoa e imprime “Olá, pessoa!” (trocando a palavra pessoa pelo nome que que informado). Você pediu ajuda para seu colega, que fez o seguinte código:

```
1 #include <cs50.h>
2 #include <stdio.h>
3
4 int main(void)
5 {
6     string nome = get_string("Qual o seu nome: ");
7
8     printf("Olá, %s\n!", string);
9
10    return 0;
11 }
```

Analise o código que o seu colega fez em relação a três critérios: corretude, design e estilo.

9. Se você fosse classificar os critérios de corretude, design, estilo e eficiência em **ordem de importância**, em que ordem você os classificaria? Indique a ordem de importância dos critérios e explique porque você acredita que um critério é mais ou menos importante do que os outros.

10. Seu colega criou o seguinte código para imprimir um simples “Olá, mundo!” na tela do terminal:

```
1 #include <stdio.h>
2
3 int
4     main      (
5 void)        {
6     printf(   "%s\n",
7 "Olá, mundo!"
8 );
9     return 0; }
```

Analise o código que o seu colega fez em relação a três critérios: corretude, design e estilo.

11. Você está criando o código para um programa e, em um determinado momento, você acessa o **terminal de comandos** do Linux e executa os seguintes comandos:

```
1 $ make teste
2 $ ./teste
3 Olá, mundo!
4 $
```

Responda o seguinte:

(a) O que é o terminal de comandos do Linux? Para que serve?

(b) Qual é, provavelmente, o nome do arquivo de código fonte do programa?

(c) Qual é o nome do arquivo executável do programa?

(d) O que é símbolo “\$” nas linhas 1, 2 e 4?

(e) O que é e para que serve o comando “**make**” que foi executado na linha 1?

(f) O que é o “./” que aparece na linha 2, imediatamente antes do nome do programa?

(g) O que esse programa faz?

12. Ao tentar compilar um programa, chamado de `teste.c`, você recebeu a seguinte mensagem de erro:

```
1 $ make teste.c
2 make: Nothing to be done for 'teste.c'.
3 $
```

O que você fez de errado nesse processo de compilação?

13. O programa abaixo é bem simples e apenas imprime um “Olá, mundo!” no terminal do computador. Essa frase impressa é o **retorno** da função `printf` ou é um **efeito colateral** dessa função? Qual a diferença entre o retorno e o efeito colateral de uma função?

```
1 #include <stdio.h>
2
3 int main(void)
4 {
5     printf("%s\n", "Olá, mundo!");
6     return 0;
7 }
```

14. Analise o seguinte código abaixo e responda às questões a seguir:

```
1 #include <stdio.h>
2
3 int soma(int a, int b);
4
5 int main(void)
6 {
7     int x = 3;
8     int y = 2;
9     int z = soma(2, 3);
10    printf("A soma de %i e %i é %i.\n", x, y, z);
11    return 0;
12 }
13
14 int soma(int a, int b)
15 {
16     return a + b;
17 }
```

- (a) Qual a função da linha 1?
-
-

(b) Por que a linha 3 é igual à linha 14? Por que é necessário que essa linha 3 exista?

(c) A função `soma` tem quantos parâmetros? Informe quais são os parâmetros e também o tipo de dados de cada parâmetro.

(d) Que tipo de dado a função `soma` retorna? E a função `main`?

(e) Em que linha do programa a função `soma` foi chamada?

(f) Ao ser chamada, a função `soma` recebeu quantos argumentos? Quais foram esses argumentos?

(g) Qual a diferença entre os parâmetros e os argumentos de uma função?

(h) Se as variáveis no programa têm o nome de “x” e “y”, porque a função `soma` faz a adição das variáveis “a” e “b”?

(i) Para que serve o comando “return” em uma função?

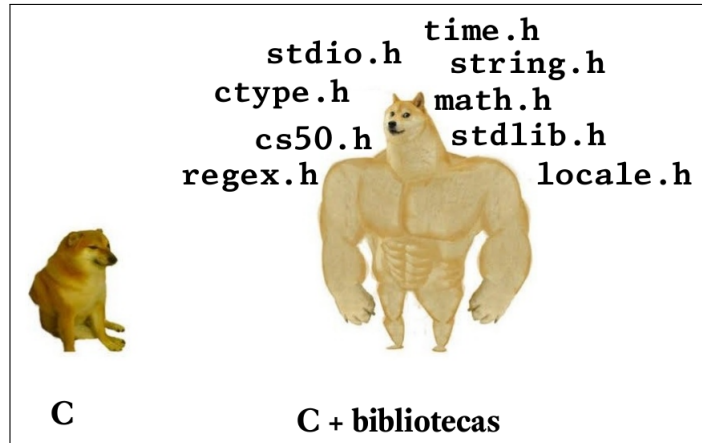
(j) A função `printf`, chamada na linha 10, recebeu quantos argumentos?

(k) O que é o “%i” que está na função `printf`? Para que ele serve? Como essa seqüência de código é oficialmente chamada?

(l) o que é o “\n” que está na função `printf`? Par que ele serve? Como essa seqüência de código é oficialmente chamada?

15. O que é, e para que serve, uma **biblioteca** de código? Como diferentes bibliotecas podem ser inseridas em seu programa?

16. Explique um importante conceito de programação C ilustrado no “meme” abaixo:



17. Qual a diferença entre as **libraries files**, terminadas em “.o” (por exemplo: `stdio.o`), das **header files** terminadas em “.h” (por exemplo: `stdio.h`)?

18. O **padrão oficial da linguagem C**, atualmente, é o C17 (ISO/IEC 9899:2018). O que é e para que serve esse padrão?

19. Apesar do padrão oficial da linguagem C não estar disponível gratuitamente na Internet (ele pode ser adquirido no site da ISO), é possível encontrar extensa documentação da linguagem C (incluindo a documentação de suas bibliotecas e *header files*). Responda:

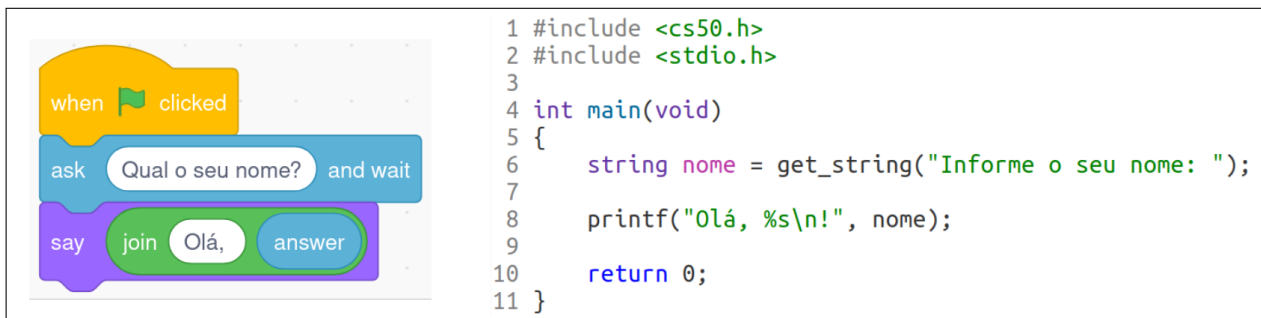
(a) Onde podemos encontrar, gratuitamente, os rascunhos dos padrões oficiais da linguagem C?

- (b) O *The Open Group* é um consórcio global de aproximadamente 900 empresas e organizações que trabalha na elaboração, divulgação e avaliação de diversos padrões tecnológicos (você pode visitar o site oficial aqui: www.opengroup.org). Um desses padrões tecnológicos é o padrão para o UNIX®, que pode ser visitado (ou baixado em HTML e/ou PDF) diretamente no endereço pubs.opengroup.org/onlinepubs/9699919799. De especial interesse nesse padrão é a documentação para diversas *header files* que podemos utilizar em nossos programas. Acesse esse site e cite cinco *header files* que estão documentadas lá.
-
-

- (c) Apesar da documentação das *header files* no site do *The Open Group* ser bem completa, ela é de difícil leitura para quem está começando. Para facilitar a documentação para os iniciantes, o pessoal da Harvard C50 preparou uma versão online simplificada para consulta. Em que endereço da Internet podemos acessar essa documentação? Qual a diferença entre a documentação no formato “*less comfortable*” e no formato “*more comfortable*”?
-
-

- (d) Ao tratar da documentação para diversas coisas, os desenvolvedores costumam escrever a sigla “**RTFM**” para programadores mais novos e, principalmente, programadores que não leram a documentação oficial. O que significa essa sigla?
-

20. A figura abaixo mostra um programa em **Scratch** e em **C**, que fazem exatamente a mesma coisa. Analise os programas e depois responda às questões que se seguem.



The image shows two equivalent programs. On the left is a Scratch script: a yellow 'when green flag clicked' block, followed by a blue 'ask' block with the text 'Qual o seu nome?' and 'and wait', and finally a purple 'say' block with 'Olá,' and 'answer'. On the right is the corresponding C code:

```

1 #include <cs50.h>
2 #include <stdio.h>
3
4 int main(void)
5 {
6     string nome = get_string("Informe o seu nome: ");
7
8     printf("Olá, %s\n!", nome);
9
10    return 0;
11 }

```

- (a) Qual o argumento da função `ask`?
-
- (b) Qual é o argumento da função `get_string`?
-
- (c) Qual é o efeito colateral da função `ask`?
-
- (d) Qual é o efeito colateral da função `get_string`?
-
- (e) Qual é o retorno da função `ask`?
-

(f) Qual é o retorno da função `get_string`?

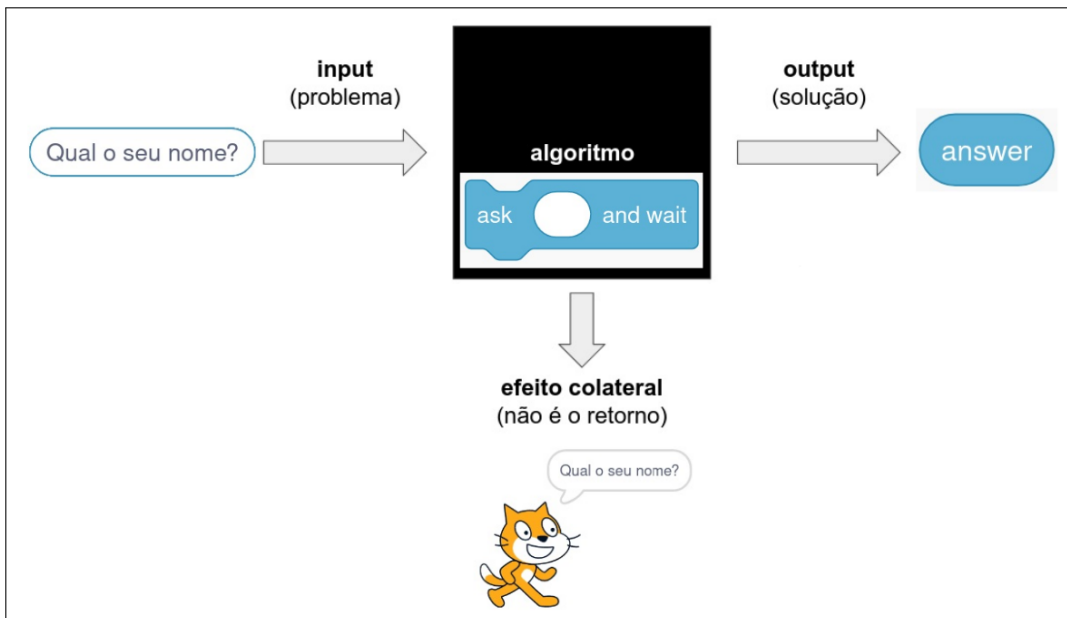
(g) Qual é o argumento da função `say`?

(h) Qual(is) o(s) argumento(s) da função `printf`?

(i) A função `say` está retornando alguma coisa? Se sim, o quê?

(j) A função `printf` está realizando algum efeito colateral? Se sim, qual?

21. Ainda em relação à diferença entre o **retorno** de uma função e algum **efeito colateral** que essa função cause, analise a figura abaixo e responda às questões a seguir.



(a) Qual foi o efeito colateral causado pela função `ask`?

(b) O que a função `ask` retornou?

(c) Onde o retorno da função `ask` foi armazenado?

(d) Por que é importante armazenar o retorno de uma função em algum lugar?

- (e) O retorno de uma função e a variável que armazena esse retorno são a mesma coisa? Sim? Não? Explique.

22. Analise a seguinte linha de código de um programa em C:

```
int idade = get_int("Informe sua idade (em anos completos): ");
```

- (a) Qual o **nome** e o **tipo de dado** da variável que foi criada?

- (b) O que a variável “idade” recebe?

- (c) Quantos argumentos a função `get_int` recebe? Quais são?

- (d) Qual o retorno da função `get_int`?

- (e) Qual o efeito colateral da função `get_int`?

- (f) A variável “idade” só poderá ser utilizada nessa linha do programa, ou poderá ser utilizada posteriormente? Explique.

23. O trecho de C, abaixo, ilustra os conceitos de **declaração**, **atribuição** e **inicialização** de variáveis:

```
string nome;  
nome = get_string("Qual o seu nome? ");  
int idade = get_int("Qual sua idade? ");
```

Explique a diferença entre esses três conceitos e indique qual linha representa cada um.

24. O código ilustrado abaixo tem um bug. Identifique que bug é esse e como ele deve ser consertado.

```
1 #include <cs50.h>
2 #include <stdio.h>
3
4 int main(void)
5 {
6     string nome = get_string("Informe o seu nome: ");
7     printf("%s\n!", "Olá, nome");
8     return 0;
9 }
```

25. Analise a seguinte linha de código em C:

```
printf("A divisão de %i por %i é igual a %.2f.\n", x, y, z);
```

Complete a frase a seguir com as palavras da seguinte lista (podem ser usadas mais de uma vez):

- *format specifier* (especificador de formato)
- *placeholder* (espaço reservado)
- *escape sequence* (seqüência de escape)
- `int`
- `float`
- formato de impressão
- `%i`
- `%.2f`
- primeiro
- segundo
- terceiro

“A função `printf` ilustrada acima está recebendo quatro argumentos. O primeiro argumento é o _____, que indica como a saída deverá ser impressa. Nesse argumento existem três _____ criados pelos _____: _____, _____ e _____. O valor da variável `x` será colocado no lugar do _____ especificador de formato, o valor da variável `y` será colocado no lugar do _____ especificador de formato, e o valor da variável `z` será colocado no lugar do _____ especificador de formato. Sabemos que as variáveis `x` e `y` são do tipo _____ (porque o especificador de formato é `%i`). Sabemos também que a variável `z` é do tipo _____ (porque o especificador de formato é `%.2f`). A seqüência “`\n`” é uma _____, que imprime um caractere especial (nesse caso está imprimindo uma nova linha no terminal).”

26. Em relação ao uso da função `printf`, qual a diferença entre os *format specifiers* (especificadores de formato) e as *escape sequences* (seqüências de escape)?

27. Ainda em relação à função `printf`, seu colega afirmou o seguinte: “Não é necessário ter um argumento para cada *format specifier*”. Você concorda ou discorda com seu colega? Justifique sua resposta.

28. O que é um **tipo de dado**?

29. Quais os tipos de dados mais comuns na linguagem C (considere também os tipos de dados fornecidos pela biblioteca `cs50.h`).

30. O que deve ser colocado dentro dos parênteses da estrutura de código abaixo? Explique como funciona essa estrutura.

```
if ( )  
{  
    // comandos  
}
```

31. Considere o seguinte trecho de código e responda às questões a seguir:

```
if (x < y)  
{  
    printf("%s\n", "laranja");  
}  
else  
{  
    printf("%s\n", "banana");  
}
```

(a) O uso do `else` é sempre obrigatório?

(b) Como esse trecho de código funciona?

(c) Em que situação o código não imprimiria nem “laranja” nem “banana”?

32. Considere o seguinte trecho de código e responda às questões a seguir:

```
if (x < y)
{
    printf("%s\n", "laranja");
}
else if (x > y)
{
    printf("%s\n", "banana");
}
```

(a) O uso do `else if` é sempre obrigatório?

(b) Como esse trecho de código funciona?

(c) Em que situação o código não imprimiria nem “laranja” nem “banana”?

33. Considere o seguinte trecho de código e responda às questões a seguir:

```
if (x < y)
{
    printf("%s\n", "laranja");
}
else if (x > y)
{
    printf("%s\n", "banana");
}
else if (x == y)
{
    printf("%s\n", "morango");
}
```

(a) É possível usarmos mais de um `else if`? Mesmo sem terminar com um `else final`?

(b) Como esse trecho de código funciona?

(c) Em que situação o código não imprimiria nem “laranja” nem “banana”? Será impresso alguma outra coisa?

(d) Há alguma situação em que esse trecho não imprime nenhuma das três frutas?

34. Em relação ao uso de estruturas condicionais em C, assinale verdadeiro (V) ou falso (F):

- (a) ___ O uso do `if` é sempre obrigatório.
- (b) ___ O `if` funciona através da avaliação de uma expressão booleana: se a expressão for falsa os comandos de dentro do bloco do `if` são executados, e se a expressão for verdadeira os comandos de dentro do bloco do `if` não são executados.
- (c) ___ É possível utilizar mais de um `if` ao mesmo tempo.
- (d) ___ É possível utilizar mais de um `else if` ao mesmo tempo, mas também é possível não utilizar nenhum.
- (e) ___ É obrigatório o uso do `else` sempre que usamos um `if`.
- (f) ___ É obrigatório o uso do `else` sempre que usamos um ou mais `else if`.
- (g) ___ Podemos usar o `else` entre um `if` e um `else if`.
- (h) ___ Podemos usar mais de um `else`.
- (i) ___ Não é obrigatório que o `else` fique no final.

35. Existem dois bugs no código abaixo. Identifique-os e diga como devem ser corrigidos.

```
#include <stdio.h>

int main(void)
{
    char c = "A";
    string s = "A vaca foi para o brejo.";
    return 0;
}
```

36. Em relação aos **operadores lógicos** que podem ser utilizados em expressões booleanas:

- (a) Para que serve o operador “&&”?

- (b) Para que serve o operador “||”?

- (c) Para que serve o operador “!”?

37. Cite 4 **operadores relacionais** que podemos utilizar em expressões booleanas.

38. Dentre os **aritméticos** que podemos usar em C, um dos mais interessantes é o operador “%”. Explique para que serve esse operador e dê um exemplo de como seria utilizado.

39. Na matemática a seguinte equação estaria **errada** pois x não é igual à soma de x mais 1.

$$x = x + 1$$

Mas, na programação, a seguinte linha de programação estaria correta:

```
x = x + 1;
```

Por que essa linha está “programaticamente” correta mesmo que esteja matematicamente errada?

40. Explique o seguinte trecho de código em C. Qual o resultado final e por que esse resultado ocorre?

```
(x == x + 1)
```

41. O que é notação *syntactic sugar*?

42. Escreva a(s) notação(ões) para as sentenças abaixo, em *syntactic sugar*.

(a) contador = contador * 2

(b) contador = contador / 3

(c) contador = contador % 9

(d) contador = contador + 1 (obs.: escreva 3 variações)

(e) contador = contador - 1 (obs.: escreva 3 variações)

43. Quais os três grandes tipos de **estruturas de repetição** (*loops*) na linguagem C?

44. O formato geral de um *loop while* é o seguinte:

```
while ( )  
{  
  
}
```

Explique como esse tipo de estrutura de repetição funciona. Dê um exemplo.

45. O formato geral de um *loop for* é o seguinte:

```
for ( ; ; )  
{  
  
}
```

Explique como esse tipo de estrutura de repetição funciona. Dê um exemplo.

46. O formato geral de um *loop do while* é o seguinte:

```
do  
{  
  
}  
while ( );
```

Explique como esse tipo de estrutura de repetição funciona. Dê um exemplo.

47. Qual a principal diferença entre um *loop while* e um *loop do while*?

48. Em relação aos *loops*, explique em que situação é mais indicado utilizar:

(a) *for*:

(b) *while*:

(c) *do while*:

49. Em relação aos *loops*, responda verdadeiro (V) ou falso (F):

- (a) ___ O *while* continua executando os comandos enquanto a expressão booleana que está sendo avaliada for falsa.
- (b) ___ O *while* é uma boa opção para quando sabemos exatamente a quantidade de vezes que a repetição será executada.
- (c) ___ O *while* pode não executar nenhuma vez.
- (d) ___ Se a expressão booleana que está sendo avaliada em um *while* nunca se torna falsa, então teremos uma repetição infinita (o que geralmente indica um erro no programa).
- (e) ___ O *for* executa os comandos um número pré-determinado de vezes;
- (f) ___ O *for* não é indicado para situações nas quais sabemos previamente quantas repetições realizaremos.
- (g) ___ O *for* pode não executar nenhuma vez.
- (h) ___ O *do while* avalia a expressão booleana antes da execução dos comandos.
- (i) ___ O *do while* pode não executar nenhuma vez.
- (j) ___ O *do while* é bom para solicitar dados do usuário e mostrar menus.
- (k) ___ Em geral os *loops* são intercambiáveis, ou seja, o que é possível fazer com um tipo é possível fazer com o outro (mas isso não quer dizer que não existam situações nas quais um seja mais indicado do que o outro).

50. O seguinte trecho de código imprime a palavra “banana” infinitas vezes:

```
#include <stdio.h>

int main(void)
{
    while ("laranja")
    {
        printf("%s\n", "banana");
    }
    return 0;
}
```

(a) Por que esse *loop* infinito ocorre?

(b) O que devemos fazer para interromper a repetição infinita?

51. O que é o **terminal** — *Command Line Interface (CLI)* — do Linux?

52. Explique as equações abaixo:

$$\text{CLI} = (\text{produtividade})^{\infty} \quad (1)$$

$$\text{Linux} + \text{CLI} = [(\text{produtividade})^{\infty}]^{(\text{produtividade})^{\infty}} \quad (2)$$
