

Arquivos e Strings

Alexsandro Santos Soares
`prof.asoares@gmail.com`

Universidade Federal de Uberlândia
Faculdade de Computação

Introdução

- Um programa é um processador de dados
 - Ele aceita dados de entrada, os processa e depois cria dados de saída.
- Os dados podem vir de diversas fontes e podem seguir para diversos destinos.
 - Por exemplo, os dados podem vir do teclado, de um arquivo no disco, de um sistema de aquecimento ou de ar condicionado ou de um canal de comunicação tal como uma rede ou Internet.
 - Os dados podem seguir para muitos destinos, tais como um monitor ou arquivo no disco ou para um canal de comunicação.
- Nesta aula nos concentraremos nos arquivos como fontes e/ou destinos de dados.

Arquivos

- Um **arquivo** é uma coleção externa de dados relacionados tratados como uma unidade.
- A função principal de um arquivo é manter um registro permanente de dados.
- Além disso, a coleção de dados é normalmente muito grande para residir inteiramente na memória principal.
 - Por isso, devemos ter à disposição meios de ler e/ou escrever parte dos dados enquanto o resto permanece em arquivos.
- Quando um programa lê um arquivo, os dados fluem do dispositivo externo para a memória principal e, quando escreve, os dados fluem no sentido inverso.

Arquivos

- À medida que o arquivo é lido, eventualmente chegaremos em um ponto os todos os dados foram lidos.
- Nesse ponto, o arquivo é dito ter chegado ao **fim do arquivo**.
- O fim de arquivo no dispositivo secundário é detectado automaticamente e comunicado ao programa.
- É tarefa do programador testar se o arquivo chegou ao fim.

Tipos de arquivos

- Existem dois tipos de arquivos que devemos conhecer:
 - ① Arquivos textos.
 - ② Arquivos binários.

Arquivo texto

- Arquivos textos são arquivos normais, geralmente associados à extensão `.txt`, que você cria usando um editor de textos como o Emacs, o Notepad, o Sublime, etc.
- Quando você abrir esse tipo de arquivo, conseguirá ver todo o conteúdo como texto e poderá usar um editor para alterar o conteúdo.
- Os arquivos de texto são fáceis de manter e de ler, mas a segurança é mínima e ocupam mais espaço para serem armazenados.

Arquivos binários

- Os arquivos binários em geral usam como extensão `.bin`.
- Ao invés de armazenar dados como texto puro, eles armazenam na forma binária, ou seja, usando a forma como são representados na memória.
- Eles podem guardar uma quantidade maior de dados e fornecem uma segurança maior que os arquivos textos, mas não são facilmente legíveis em um editor.

Operações sobre arquivos

Em C você pode realizar quatro operações principais sobre um arquivo, seja ele texto ou binário:

- ❶ Criar um novo arquivo.
- ❷ Abrir um arquivo existente.
- ❸ Fechar um arquivo.
- ❹ Ler ou escrever para um arquivo.

Trabalhando com arquivos

- Antes de trabalhar com arquivos, precisamos declarar um apontador para o tipo `FILE`.
- Esta declaração é necessária para a comunicação entre o arquivo e o programa:

```
FILE *fp; // fp significa file pointer (apontador para arquivo)
```

- Ler ou escrever para um arquivo em C envolve três passos:
 - ❶ Abrir o arquivo.
 - ❷ Fazer toda a leitura e/ou escrita.
 - ❸ Fechar o arquivo.
- Nos próximos slides veremos como realizar cada um desses passos.

Abrindo um arquivo – para criação e edição

- A abertura de um arquivo é feita com a função `fopen` da biblioteca `stdio.h`.
- A sintaxe para a abertura de um arquivo é

```
fp = fopen("caminho com nome do arquivo", "modo");
```

- Assuma que o arquivo de nome `novoarq.txt` não exista no diretório `/Documentos`. A função abaixo cria um novo arquivo com o nome `novoarq.txt` e o abre para escrita usando o modo `"w"`:

```
fp = fopen("~/Documentos/novoarq.txt", "w");
```

- O modo de escrita `w` permite que você crie um arquivo **texto** e o edite, sobrescrevendo o conteúdo do arquivo.
 - Se o arquivo existir, o conteúdo dele será apagado.
- Se desejar abrir um arquivo **binário** para escrita, o modo será `"wb"`.

Abrindo um arquivo para leitura

- Assuma que um outro arquivo binário de nome `arqvelho.bin` já exista no mesmo diretório em que você executou o programa.
- Se desejar abrir esse arquivo para leitura em modo **binário** usamos

```
fp = fopen("arqvelho.bin", "rb");
```
- O modo de leitura permite apenas que o arquivo seja lido, mas não escrito.
 - Você não poderá escrever para um arquivo aberto desta forma.
- Se desejar abrir um arquivo **texto** para leitura, use o modo `"r"`.
- Nos dois próximos slides mostraremos todos os modos de abertura de arquivos em C.

Modos de abertura de arquivos

Modo	Significado	Inexistência do arquivo
r	Abrir para leitura.	se o arquivo não existir, fopen retorna NULL.
rb	Abrir para leitura em modo binário	se o arquivo não existir, fopen retorna NULL.
w	Abrir para escrita.	Se o arquivo existir, o conteúdo dele será sobrescrito. Se não existir, ele será criado.
wb	Abrir para escrita em modo binário.	Se o arquivo existir, o conteúdo dele será sobrescrito. Se não existir, ele será criado.
a	Abrir para concatenação, ou seja, os dados serão adicionados no final do arquivo.	Se o arquivo não existir, ele será criado.
ab	Abrir para concatenação em modo binário , ou seja, os dados serão adicionados no final do arquivo.	Se o arquivo não existir, ele será criado.

Modos de abertura de arquivos

Modo	Significado	Inexistência do arquivo
r+	Abrir tanto para leitura quanto para escrita.	se o arquivo não existir, fopen retorna NULL.
rb+	Abrir tanto para leitura quanto para escrita em modo binário	se o arquivo não existir, fopen retorna NULL.
w+	Abrir tanto para leitura quanto para escrita.	Se o arquivo existir, o conteúdo dele será sobrescrito. Se não existir, ele será criado.
wb+	Abrir tanto para leitura quanto para escrita em modo binário.	Se o arquivo existir, o conteúdo dele será sobrescrito. Se não existir, ele será criado.
a+	Abrir tanto para leitura quanto para concatenação	Se o arquivo não existir, ele será criado.
ab+	Abrir tanto para leitura quanto para concatenação em modo binário.	Se o arquivo não existir, ele será criado.

Fechando um arquivo

- O arquivo, seja ele texto ou binário, deve sempre ser fechado após seu uso.
- O fechamento é realizado usando a função `fclose` da biblioteca `stdio.h`:

```
fclose(fd);
```

- Lembre-se que `fd` é o apontador para o arquivo que deverá ser fechado.

Lendo e escrevendo para um arquivo texto

- Para ler ou escrever para um arquivo texto, usamos as funções `fprintf` e `fscanf`.
- Essas funções funcionam como `printf` ou `fscanf`, a diferença é que `fprintf` e `fscanf` esperam também como argumento um apontador para arquivo.

Exemplo 1 (Escrevendo para um arquivo texto)

O programa a seguir lê um número do usuário e o guarda no arquivo texto `programa.txt`.

```
9  #include <stdio.h>
10 #include <stdlib.h>
11
12 int main(void){
13     int num;
14     FILE *fp;
15
16     fp = fopen("programa.txt", "w"); // abre para escrita
17
18     if (fp == NULL){
19         fprintf(stderr, "Erro de escrita no arquivo programa.txt\n");
20         exit(1);
21     }
22
23     printf("Digite um número: ");
24     scanf("%d", &num);
25
26     fprintf(fp, "%d", num);
27     fclose(fp);
28
29     return 0;
30 } // main
```


Observações sobre o exemplo

Algumas coisas para se notar no código anterior:

- Na linha 19, ao imprimir a mensagem de erro usamos a função `fprintf` com primeiro argumento `stderr`. Essa é a saída de erro padrão, normalmente a tela do monitor do computador.
- Outras saídas ou entradas padrão são:
 - `stdin` entrada padrão, normalmente o teclado.
 - `stdout` saída padrão, normalmente a tela do monitor.
- Na linha 20, usamos a função `exit` para sair do programa devolvendo o código de erro 1. Essa função está definida na biblioteca `stdlib.h`.
 - Essa biblioteca foi incluída na linha 10.
- Sempre verifique a saída das funções `fopen`, `fprintf` ou `fscanf` como mostrado na linha 18.
- Quando o arquivo não for mais necessário, deve-se fechá-lo como feito na linha 27.

Uso

Execução do programa:

```
> ./exemplo1.exe
```

Digite um número: 123

Se olhar o diretório onde o programa foi executado notará a existência do arquivo `programa.txt`.

Para ver o conteúdo rapidamente use o comando `cat`:

```
> cat programa.txt  
123
```

Exemplo 2 (Lendo de um arquivo texto)

O programa a seguir lê um número inteiro presente no arquivo texto `programa.txt` e o imprime na tela.

```
9  #include <stdio.h>
10 #include <stdlib.h>
11
12 int main(void){
13     int num;
14     FILE *fp;
15
16     fp = fopen("programa.txt", "r"); // abre para leitura
17
18     if (fp == NULL){
19         fprintf(stderr, "Erro na abertura do arquivo programa.txt\n");
20         exit(1);
21     }
22
23     fscanf(fp, "%d", &num);
24
25     printf("Valor lido: %d\n", num);
26     fclose(fp);
27
28     return 0;
29 } // main
```

Uso

Execução do programa:

```
> ./exemplo2.exe
```

Valor lido: 123

Note que o programa deste exemplo pressupõe que o exemplo 1 foi executado corretamente, criando o arquivo `programa.txt`. Se ele não existir, uma mensagem de erro será escrita na tela.

Lendo e escrevendo para um arquivo binário

- Para ler ou escrever para um arquivo binário, usamos as funções `fread` e `fwrite`, respectivamente.
- Nos próximos slides veremos exemplos de uso delas.

Escrevendo para um arquivo binário

- Para escrever para um arquivo binário deve-se usar a função `fwrite`.
- Essa função recebe quatro argumentos:
 - O endereço dos dados a serem escritos para o arquivo.
 - O tamanho em bytes dos dados a serem escritos.
 - O número de valores a serem escritos, cada um com tamanho dado pelo argumento anterior.
 - O apontador para o arquivo onde os dados serão escritos.
- O formato de chamada é

```
fwrite(endereço_dos_dados, tamanho_unitário_dos_dados,  
       quantidade_de_dados, apontador_para_arquivo);
```

Exemplo 3 (Escrevendo para um arquivo binário)

O programa a seguir lê um número do teclado e o escreve para o arquivo **binário** `programa.bin`.

```
9  #include <stdio.h>
10 #include <stdlib.h>
11
12 int main(void){
13     int num;
14     FILE *fp;
15
16     fp = fopen("programa.bin", "wb"); // abre para escrita binária
17
18     if (fp == NULL){
19         fprintf(stderr, "Erro de escrita no arquivo programa.bin\n");
20         exit(1);
21     }
22
23     printf("Digite um número: ");
24     scanf("%d", &num);
25
26     fwrite(&num, sizeof(int), 1, fp);
27     fclose(fp);
28
29     return 0;
30 } // main
```

Uso

- Como estamos apenas escrevendo um único número, o terceiro parâmetro é 1.
- O último parâmetro `fp` aponta para o arquivo onde desejamos guardar os dados.

Execução do programa:

```
> ./exemplo3.exe
```

Digite um número: 123

Se olhar o diretório verá o arquivo `programa.bin`, mas, se tentar lê-lo, não verá muita coisa compreensível.

Exemplo 4 (Lendo de um arquivo binário)

O programa a seguir lê um número inteiro presente no arquivo binário `programa.bin` e o imprime na tela.

```
 9 #include <stdio.h>
10 #include <stdlib.h>
11
12 int main(void){
13     int num;
14     FILE *fp;
15
16     fp = fopen("programa.bin", "rb"); // abre para leitura binária
17
18     if (fp == NULL){
19         fprintf(stderr, "Erro na abertura do arquivo programa.bin\n");
20         exit(1);
21     }
22
23     if (fread(&num, sizeof(int), 1, fp) != 1){
24         fprintf(stderr, "Erro de leitura no arquivo programa.bin\n");
25         exit(1);
26     }
27
28     printf("Valor lido: %d\n", num);
29     fclose(fp);
30     return 0;
31 } // main
```

Observações sobre o exemplo

- Note que a função `fread` também recebe quatro argumentos de forma similar à função `fwrite`.

- O formato de chamada é

```
fread(endereço_dos_dados, tamanho_unitário_dos_dados,  
      quantidade_de_dados, apontador_para_arquivo);
```

- A função `fread` retorna a quantidade de itens efetivamente escritos.
 - Em uma situação normal, essa quantidade deve ser sempre igual ao terceiro argumento.
 - Lembre-se sempre de verificar se isso de fato ocorre, como fizemos entre as linhas 23 a 26.

Uso

Execução do programa:

```
> ./exemplo4.exe
```

Valor lido: 123

Note que o programa deste exemplo pressupõe que o exemplo 3 foi executado corretamente, criando o arquivo binário `programa.bin`. Se ele não existir, uma mensagem de erro será escrita na tela.

Exemplos

- Veremos alguns exemplos de manipulação de arquivos.
- Antes porém vamos aprender um pouco sobre strings em C.

Strings

- C não possui o tipo string como um de seus tipos básicos.
- Em C uma string é um arranjo de caracteres terminado com o caractere nulo `'\0'`.
- Assim a string `"um bom dia"` será representado na memória como no arranjo a seguir:

```
char texto[11] = "um bom dia";
```

0	1	2	3	4	5	6	7	8	9	10
u	m		b	o	m		d	i	a	\0

texto

- No exemplo vemos uma forma de declarar uma string usando um arranjo.
- Uma outra forma é usar um apontador

```
char *p;
```

Inicialização de strings

- Em C, uma string pode ser inicializada de várias formas.
- Por conveniência e facilidade, tanto a inicialização quanto a declaração serão feitas no mesmo passo.

Inicialização usando arranjo

- Todos os exemplos abaixo produzem a mesma string.

```
char c[] = "abcd";  
char c[50] = "abcd";  
char c[] = {'a', 'b', 'c', 'd', '\0'};  
char c[5] = {'a', 'b', 'c', 'd', '\0'};
```

Inicialização usando apontador

- Uma string também pode ser inicializada usando um apontador:

```
char *c = "abcd";
```


Lendo strings do usuário

- Você pode usar `scanf` para ler uma string assim como para outros tipos de dados.
- Entretanto, `scanf` somente lê a primeira palavra. A função termina assim que encontrar um espaço branco (espaço, nova linha, tabulação).
- O formato será

```
char str[20];  
scanf("%s", str);
```

Exemplo 5 (Lendo uma string do teclado)

O programa a seguir lê uma string do terminal e a imprime na tela.

```
9 #include <stdio.h>
10
11 int main(void){
12     char nome[20];
13
14     printf("Digite o nome: ");
15     scanf("%s", nome);
16
17     printf("O nome digitado foi %s.\n", nome);
18
19     return 0;
20 } // main
```

Uso

Execução do programa:

```
> ./exemplo5.exe
```

```
Digite o nome: Maria Dolores
```

```
O nome digitado foi Maria.
```

Note que o programa deste exemplo somente leu a primeira palavra, pois esse é o comportamento de `scanf`.

Exemplo 6 (Lendo uma linha de texto)

O programa a seguir lê uma linha inteira de texto do terminal, caractere por caractere.

```
9 #include <stdio.h>
10
11 int main(void){
12     char nome[30] = {'\0'};
13     char c = '\0';
14     int i = 0;
15
16     printf("Digite o nome: ");
17     i = 0;
18     while (c != '\n'){ // termina se o usuário digitar enter
19         c = getchar();
20         nome[i] = c;
21         i++;
22     } // while
23
24     nome[i] = '\0';    // insira o caractere nulo no final
25
26     printf("O nome digitado foi %s.\n", nome);
27
28     return 0;
29 } // main
```

Uso

Execução do programa:

```
> ./exemplo6.exe
```

```
Digite o nome: Maria Dolores
```

```
O nome digitado foi Maria Dolores
```

```
.
```

A função `getchar` lê um único caracter por vez.

No próximo slide veremos uma forma mais cômoda de ler uma string.

Exemplo 7 (Lendo uma linha de texto usando fgets)

O programa a seguir lê uma linha inteira de texto do terminal usando a função de biblioteca `fgets`.

```
10 #include <stdio.h>
11 #include <stdlib.h>
12
13 int main(void){
14     char nome[30] = {'\0'};
15
16     printf("Digite o nome: ");
17     if (fgets(nome, sizeof(nome), stdin) == NULL){
18         fprintf(stderr, "Erro ao ler a string\n");
19         exit(1);
20     }
21
22     printf("O nome digitado foi %s.\n", nome);
23
24     return 0;
25 } // main
```

Uso

Execução do programa:

```
> ./exemplo7.exe
```

```
Digite o nome: Maria Dolores
```

```
O nome digitado foi Maria Dolores
```

```
.
```

A função `fgets` recebe como argumentos:

- O arranjo onde a string será guardada.
- O número máximo de caracteres que cabem no arranjo anterior.
- O apontador para o arquivo de onde a string será lida.
 - Quando estamos lendo da entrada padrão, o teclado, usamos `stdin`.
- A função `fgets` devolve um apontador para o arranjo onde a string lida foi colocada ou `NULL` se algum problema aconteceu.
 - Lembre sempre de verificar a saída de `fgets`.

Passando strings para funções

- Strings são apenas arranjos.
- Tudo que aprendemos sobre arranjos também vale para strings, incluindo as formas de passá-las para funções

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

void exibeString(char str[]){
    printf("String impressa: %s\n", str);
    return;
}

int main(void){
    char str[50] = {'\0'};

    printf("Digite uma string: ");
    if (fgets(str, sizeof(str), stdin) == NULL){
        fprintf(stderr, "Erro ao ler a string\n");
        exit(1);
    }
    exibeString(str);
    return 0; } // main
```


Exemplos com arquivos

- Agora que vimos strings já temos o suficiente para entender os exemplos que seguirão.

Exemplo 8 (Lendo nome e notas e guardando em arquivo)

O programa a seguir lê do teclado n nomes e notas de estudantes e os guarda no arquivo texto `estudantes.txt`.

```
10 #include <stdio.h>
11 #include <stdlib.h>
12 #include <string.h>
13
14 int main(void){
15     char nome[50] = {'\0'};
16     int nota = 0;
17     int n = 0;
18
19     printf("Digite o número de estudantes: ");
20     scanf("%d", &n);
21     getchar(); // necessário para consumir o enter
22
23     FILE *fp;
24     fp = fopen("estudante.txt", "w");
25
26     if (fp == NULL){
27         fprintf(stderr, "Erro ao abrir o arquivo\n");
28         exit(1);
29     }
```

```
31  for(int i = 0; i < n; ++i){
32      printf("Para o estudante %d\n", i+1);
33      printf("Digite o nome: ");
34      if (fgets(nome, sizeof(nome), stdin) == NULL){
35          fprintf(stderr, "Erro ao ler o nome\n");
36          exit(1);
37      } else {
38          // retira o \n do final da string lida
39          nome[strcspn(nome, "\n")] = 0;
40      } // else
41
42      printf("Digite a nota: ");
43      scanf("%d", &nota);
44      getchar(); // necessário para consumir o enter
45
46      fprintf(fp, "\nNome: %s \nNota=%d \n", nome, nota);
47  } // for
48
49  fclose(fp);
50  return 0;
51 } // main
```

A função `strcspn`, da biblioteca `string.h`, conta o número de caracteres a partir do início da string até que ela encontre um `'\n'`.

Uso

Execução do programa:

```
> ./exemplo8.exe
```

Digite o número de estudantes: 3

Para o estudante 1

Digite o nome: Jackson do Pandeiro

Digite a nota: 98

Para o estudante 2

Digite o nome: Luiz Gonzaga

Digite a nota: 99

Para o estudante 3

Digite o nome: Genival Lacerda

Digite a nota: 96

Uso

Para ler o arquivo gerado:

```
> cat estudante.txt
```

Nome: Jackson do Pandeiro

Nota=98

Nome: Luiz Gonzaga

Nota=99

Nome: Genival Lacerda

Nota=96

Exemplo 9 (Lendo nomes e notas e concatenando ao arquivo)

O programa a seguir é uma modificação do exemplo anterior. Ele lê do teclado n nomes e notas de estudantes e os guarda no arquivo texto `estudantes.txt`. Entranto, se o arquivo já existir, o programa vai acrescentar a nova informação no final do arquivo.

```
11 #include <stdio.h>
12 #include <stdlib.h>
13 #include <string.h>
14
15 int main(void){
16     char nome[50] = {'\0'};
17     int nota = 0;
18     int n = 0;
19
20     printf("Digite o número de estudantes: ");
21     scanf("%d", &n);
22     getchar(); // necessário para consumir o enter
23
24     FILE *fp;
25     fp = fopen("estudante.txt", "a"); // note o modo de abertura 'a'
26
27     if (fp == NULL){
28         fprintf(stderr, "Erro ao abrir o arquivo\n");
29         exit(1);
30     }
```

```
32  for(int i = 0; i < n; ++i){
33      printf("Para o estudante %d\n", i+1);
34      printf("Digite o nome: ");
35      if (fgets(nome, sizeof(nome), stdin) == NULL){
36          fprintf(stderr, "Erro ao ler o nome\n");
37          exit(1);
38      } else {
39          // retira o \n do final da string lida
40          nome[strcspn(nome, "\n")] = 0;
41      } // else
42
43      printf("Digite a nota: ");
44      scanf("%d", &nota);
45      getchar(); // necessário para consumir o enter
46
47      fprintf(fp, "\nNome: %s \nNota=%d \n", nome, nota);
48  } // for
49
50  fclose(fp);
51  return 0;
52 } // main
```

Uso

Execução do programa:

```
> ./exemplo9.exe
```

```
Digite o número de estudantes: 2
```

Para o estudante 1

```
Digite o nome: Dominginhos
```

```
Digite a nota: 97
```

Para o estudante 2

```
Digite o nome: Elba Ramalho
```

```
Digite a nota: 98
```


Verificando o arquivo

Para ler o arquivo gerado:

```
> cat estudante.txt
```

Nome: Jackson do Pandeiro

Nota=98

Nome: Luiz Gonzaga

Nota=99

Nome: Genival Lacerda

Nota=96

Nome: Dominginhos

Nota=97

Nome: Elba Ramalho

Nota=98

Exemplo 10 (Escrevendo um arranjo para um arquivo e lendo de volta)

Neste exemplo o programa escreverá para um arquivo todos os elementos de arranjo de inteiros. Depois, ele lerá o arranjo do mesmo arquivo e o imprimirá na tela.

```
10 #include <stdio.h>
11 #include <stdlib.h>
12
13 #define TAM_MAX 6
14
15 int main(void){
16     int vet[TAM_MAX] = {0}; // arranjo original
17     int vet2[TAM_MAX] = {0}; // conterá a cópia de vet lida do arquivo
18     int i = 0;
19     FILE *fp;
20
21     fp = fopen("arranjo.dat", "wb"); // escrita em modo binário
22
23     if (fp == NULL){
24         fprintf(stderr, "Erro ao abrir o arquivo\n");
25         exit(1);
26     }
```

```
28 // Lendo o arranjo do teclado
29 printf("\nDigite %d inteiros:\n", TAM_MAX);
30 for(i = 0; i < TAM_MAX; ++i){
31     scanf("%d", &vet[i]);
32 } // for
33
34 // Escrevendo o arranjo todo para o arquivo
35 fwrite(vet, sizeof(vet), 1, fp);
36 fclose(fp); // fechando o arquivo
37
38 fp = fopen("arranjo.dat", "rb"); // leitura em modo binário
39
40 // Lendo o arranjo todo do arquivo
41 fread(vet2, sizeof(vet2), 1, fp);
42 fclose(fp);
43
44 printf("\nArranjo lido:\n");
45 for(i = 0; i < TAM_MAX; ++i)
46     printf("%d: %d\n", i, vet2[i]);
47
48 return 0;
49 } // main
```

Uso

Execução do programa:

```
> ./exemplo10.exe
```

Digite 6 inteiros:

```
7 8 9 10 11 12
```

Arranjo lido:

```
0: 7
```

```
1: 8
```

```
2: 9
```

```
3: 10
```

```
4: 11
```

```
5: 12
```

Para saber mais

- Forouzan, B. A and Gilbert, R. F. *Computer Science: a structured programming approach using C*. 3rd edition. Cengage Learning, 2007.

Fontes

- Forouzan, B. A and Gilbert, R. F. *Computer Science: a structured programming approach using C*. 3rd edition. Cengage Learning, 2007.
- *C Programming Files I/O*. Disponível em <https://www.programiz.com/c-programming/c-file-input-output>
- *C Programming File examples*. Disponível em <https://www.programiz.com/c-programming/c-file-examples>.