Mini-Projeto de Algoritmos e Programação I

Marcelo Hashimoto

Última Atualização: 6 de abril de 2015

1 Introdução

Neste Mini-Projeto, cada grupo deve escrever um **codificador e decodificador hexadecimal**, baseado nos conceitos de **leitura e escrita de arquivos** e **conversão de base**. As primeiras seções do manual apresentam algumas orientações para implementação e as últimas seções apresentam a especificação propriamente dita.

2 Leitura Básica de Arquivos

Copie todos os arquivos anexados ao manual para a mesma pasta e abra o código-fonte `exemplo1.c`.

```
#include <stdio.h>

int main() {
  int total, n, i;
  FILE *f;

  f = fopen("exemplo1.txt", "r");

  if(f == NULL) {
    printf("Erro durante abertura do arquivo\n");
    return 0;
  }

  fscanf(f, "%d", &total);

  for(i = 0; i < total; i++) {
    fscanf(f, "%d", &n);

    printf("%d\n", n);
  }

  fclose(f);

  return 0;
}
```

Compile e execute esse código-fonte. Nas subseções a seguir, ele é explicado em detalhes.

2.1 Fluxos de arquivo

A variável `f`, um *apontador* para o tipo `FILE`, é o terceiro exemplo visto neste semestre do conceito de *fluxo de dados*. Os dois primeiros foram o `stdin` e o `stdout`, respectivamente a *entrada padrão* (tradicionalmente o teclado) e a *saída padrão* (tradicionalmente o terminal).

2.2 Abrindo um fluxo de arquivo para leitura

A variável `f` recebe um *endereço* da função `fopen`, que por sua vez recebe duas *strings*: a primeira é simplesmente o caminho onde o arquivo desejado se localiza e a segunda é o *modo de abertura*. Nesse exemplo em particular temos `"r"`, pois deseja-se abrir o arquivo para *leitura* (*read*).

Nem sempre essa abertura é bem-sucedida: o arquivo pode, por exemplo, não existir ou não permitir leitura. Nesse caso, a função devolve o endereço especial `NULL`. Comparando o valor de `f` com `NULL` depois da chamada de `fopen`, o programa decide se deve continuar ou terminar. No segundo caso, imprime uma mensagem de erro.

2.3 Lendo números de um arquivo

Sobre um arquivo aberto para leitura, você pode aplicar a função `fscanf`. Essa função é totalmente análoga à função `scanf`: a única diferença é ler de um fluxo qualquer em vez de necessariamente da entrada padrão. De fato, para simular `fscanf` considerando o fluxo de um arquivo, basta simular `scanf` considerando o conteúdo desse arquivo como a digitação do usuário.

Note que o laço funciona independentemente de quantos espaços, tabulações e quebras de linha existem entre os números. Também funcionaria se considerássemos valores e variáveis do tipo `float`, substituindo `%d` por `%f`.

2.4 Fechando um fluxo de arquivo

Se não há mais leituras ou escritas a serem feitas, um fluxo de arquivo deve ser fechado através da função `fclose`. Essa função recebe o endereço do fluxo e libera os recursos alocados durante sua abertura.

3 Leitura Avançada de Arquivos

Agora abra, compile e execute o código-fonte `exemplo2.c`.

```
#include <stdio.h>

#define MAX 25

int main() {
  char linha[MAX];
  FILE *f;

  f = fopen("exemplo2.txt", "r");

  if(f == NULL) {
    printf("Erro durante abertura do arquivo\n");
    return 0;
  }

  fgets(linha, MAX, f);
  printf("(%s)\n", linha);

  fgets(linha, MAX, f);
  printf("(%s)\n", linha);

  fgets(linha, MAX, f);
  printf("(%s)\n", linha);

  fgets(linha, MAX, f);
  printf("(%s)\n", linha);

  fclose(f);

  return 0;
}
```

Assim como `fscanf` é análoga a `scanf`, passar o endereço de um fluxo de arquivo para a função `fgets` é análogo a passar `stdin`: para simular o primeiro, basta simular o segundo considerando o conteúdo do arquivo como a digitação. Lembre que, ao combinar `fscanf` e `fgets`, atenção especial deve ser dada às quebras de linha.

4 Escrita de Arquivos

Por fim abra, compile e execute o código-fonte `exemplo3.c`.

```
#include <stdio.h>

int main() {
  FILE *f;

  f = fopen("exemplo3.txt", "w");

  if(f == NULL) {
    printf("Erro durante abertura do arquivo\n");
    return 0;
  }

  fprintf(f, "Hello World!\n");

  fprintf(f, "Hello World! Eis um inteiro: %d\n", 100);

  fprintf(f, "Hello World! Eis um real: %f\n", 0.5);

  fclose(f);

  return 0;
}
```

Nesse exemplo temos `"w"` como modo de abertura, pois deseja-se abrir o arquivo para *escrita* (*write*). Esse modo cria o arquivo quando ele não existe e apaga o conteúdo do arquivo quando ele existe, portanto tome cuidado para não passar para `fopen` o caminho de um arquivo existente cujo conteúdo é importante!

Sobre um arquivo aberto para escrita, você pode aplicar a função `fprintf`. Essa função é totalmente análoga à função `printf`: a única diferença é escrever em um fluxo qualquer em vez de necessariamente na saída padrão.

5 Tabela ASCII

Conceitualmente, variáveis do tipo `char` armazenam caracteres como `'a'`, `'b'`, `'c'`, etc. Tecnicamente, no entanto, o que essas variáveis realmente armazenam são *códigos numéricos* que representam caracteres. Você pode facilmente descobrir qual código representa um caractere se atribuir o valor do respectivo `char` a um `int`.

```
char c;
int i;

c = 'a';
i = c;

printf("%d\n", i);
```

O programa acima imprime o código `97`. Substituindo `'a'` por `'b'`, imprime `98`. Os códigos utilizados na linguagem `C` para caracteres do alfabeto latino podem ser vistos na tabela abaixo, parte do sistema *ASCII*.

A	65	N	78	a	97	n	110
B	66	O	79	b	98	o	111
C	67	P	80	c	99	p	112
D	68	Q	81	d	100	q	113
E	69	R	82	e	101	r	114
F	70	S	83	f	102	s	115
G	71	T	84	g	103	t	116
H	72	U	85	h	104	u	117
I	73	V	86	i	105	v	118
J	74	W	87	j	106	w	119
K	75	X	88	k	107	x	120
L	76	Y	89	l	108	y	121
M	77	Z	90	m	109	z	122

6 Base Hexadecimal

Podemos observar que todos os códigos da tabela acima são inteiros entre 0 e 255. Podemos observar também que, para qualquer inteiro n nesse intervalo, existem dois inteiros x e y, ambos entre 0 e 15, tais que

$$n = 16x + y.$$

Descobrir esses dois inteiros é simples: ao dividir n por 16, obtemos x como quociente e y como resto. Considere agora a seguinte representação dos inteiros entre 0 e 15 como caracteres:

0	representado por	`'0'`;	8	representado por	`'8'`;
1	representado por	`'1'`;	9	representado por	`'9'`;
2	representado por	`'2'`;	10	representado por	`'A'`;
3	representado por	`'3'`;	11	representado por	`'B'`;
4	representado por	`'4'`;	12	representado por	`'C'`;
5	representado por	`'5'`;	13	representado por	`'D'`;
6	representado por	`'6'`;	14	representado por	`'E'`;
7	representado por	`'7'`;	15	representado por	`'F'`.

Podemos concluir que, sob o esquema acima, qualquer inteiro entre 0 e 255 pode ser representado por dois caracteres dentro do conjunto $\mathcal{H} = \{$`'0'`, `'1'`, `'2'`, `'3'`, `'4'`, `'5'`, `'6'`, `'7'`, `'8'`, `'9'`, `'A'`, `'B'`, `'C'`, `'D'`, `'E'`, `'F'`$\}$.

7 Codificação

A partir das duas seções anteriores, considere a seguinte sequência de transformações:

caractere latino $\rightarrow$ inteiro entre 0 e 255 $\rightarrow$ dois inteiros entre 0 e 15 $\rightarrow$ dois caracteres em $\mathcal{H}$.

Alguns exemplos dessa sequência são dados a seguir:

`'A'` $\rightarrow$ 65 $\rightarrow$ 4 e 1 $\rightarrow$ `'4'` e `'1'`
`'R'` $\rightarrow$ 82 $\rightarrow$ 5 e 2 $\rightarrow$ `'5'` e `'2'`
`'i'` $\rightarrow$ 105 $\rightarrow$ 6 e 9 $\rightarrow$ `'6'` e `'9'`
`'z'` $\rightarrow$ 122 $\rightarrow$ 7 e 10 $\rightarrow$ `'7'` e `'A'`

8 Decodificação

Analogamente, considere a sequência inversa de transformações:

dois caracteres em $\mathcal{H}$ $\rightarrow$ dois inteiros entre 0 e 15 $\rightarrow$ inteiro entre 0 e 255 $\rightarrow$ caractere latino.

Exemplos correspondentes aos anteriores são dados a seguir:

`'4'` e `'1'` $\rightarrow$ 4 e 1 $\rightarrow$ 65 $\rightarrow$ `'A'`
`'5'` e `'2'` $\rightarrow$ 5 e 2 $\rightarrow$ 82 $\rightarrow$ `'R'`
`'6'` e `'9'` $\rightarrow$ 6 e 9 $\rightarrow$ 105 $\rightarrow$ `'i'`
`'7'` e `'A'` $\rightarrow$ 7 e 10 $\rightarrow$ 122 $\rightarrow$ `'z'`

9 Especificação Geral

O programa entregue deve ser capaz de **codificar linhas de texto**, como descrito na Seção 7, e **decodificar linhas de texto**, como descrito na Seção 8. Além disso, essas linhas de texto devem ser **lidas de arquivos** e **escritas em arquivos**. Por fim, essas codificações e decodificações devem ser **repetidas quantas vezes o usuário desejar**. Espera-se que uma execução do programa siga especificamente a sequência de passos abaixo:

1. pede para o usuário digitar `1` para codificação ou `2` para decodificação;
2. pede para o usuário digitar um caminho para um arquivo de entrada;
3. pede para o usuário digitar um caminho para um arquivo de saída;
4. lê do arquivo de entrada e escreve no arquivo de saída, de acordo com a operação selecionada;
5. pede para o usuário digitar `s` ou `n`. No primeiro caso, repete os passos. No segundo, termina o programa.

O exemplo abaixo ilustra o comportamento esperado. Os trechos em azul representam digitação do usuário.

```
Selecione a operacao:
1 - Codificar
2 - Decodificar
0
Selecione a operacao:
1 - Codificar
2 - Decodificar
4
Selecione a operacao:
1 - Codificar
2 - Decodificar
1

Digite o caminho do arquivo de entrada: caminhoinvalido.txt
Digite o caminho do arquivo de entrada: entrada.txt

Digite o caminho do arquivo de saida: caminhoinvalido.txt
Digite o caminho do arquivo de saida: saida.txt

OPERACAO EFETUADA COM SUCESSO

Deseja efetuar outra operacao? (s/n): a
Deseja efetuar outra operacao? (s/n): s

Selecione a operacao:
1 - Codificar
2 - Decodificar
```

Não é necessário utilizar exatamente os mesmos textos, porém as restrições abaixo devem ser respeitadas.

- No Passo 1, enquanto o usuário selecionar uma opção que não seja `1` ou `2`, o pedido deve ser repetido.
- Nos Passos 2 e 3, enquanto o usuário selecionar um caminho inválido, o pedido deve ser repetido.
- No Passo 5, enquanto o usuário selecionar uma resposta que não seja `s` ou `n`, o pedido deve ser repetido.

Por fim, os arquivos de entrada e saída devem conter exatamente:

1. um inteiro positivo n;
2. uma quebra de linha, ou seja, o caractere `'\n'`;
3. exatamente n linhas de texto.

No caso da codificação, espera-se que cada linha de texto da entrada contenha no máximo 25 caracteres *sem contar a quebra de linha* e que cada um deles seja um dos 52 caracteres latinos da Seção 5 ou um dos seis abaixo.

	32	,	44
!	33	.	46
"	34	?	63

No caso da decodificação, espera-se que cada linha de texto da entrada contenha no máximo 50 caracteres *sem contar a quebra de linha* e que cada um deles pertença ao conjunto $\mathcal{H}$ da Seção 6.

O programa deve garantir que o arquivo de saída satisfaz todas essas restrições, mas você pode supor que o arquivo de entrada as satisfaz. Não é necessário verificá-las.

10 Exemplos

Para o seguinte exemplo de arquivo decodificado

```
5
Hello World!
mlkjihgfedcba
nopqrstuvwxyz !"
MLKJIHGFEDCBA
NOPQRSTUVWXYZ,.?
```

temos o seguinte exemplo de arquivo codificado

```
5
48656C6C6F20576F726C6421
6D6C6B6A696867666564636261
6E6F707172737475767778797A202122
4D4C4B4A494847464544434241
4E4F505152535455565758595A2C2E3F
```

Espera-se que a codificação transforme o primeiro no segundo e a decodificação transforme o segundo no primeiro.

11 Regras

- O programa deve ser feito individualmente ou em grupo de 2 a 4 alunos.
- Utilize apenas a linguagem de programação C.
- Utilize apenas os recursos da linguagem que foram ensinados em aula e neste manual.
- Legibilidade será considerada na correção. Capriche na indentação e na organização.
- A primeira versão deve ser entregue pelo Blackboard até as **23:50** do dia **15 de maio**.
- A nota dessa primeira versão será devolvida no dia **22 de maio**.
- Caso o aluno queira melhorar a nota, uma segunda versão deve ser entregue pelo Blackboard até as **23:50** do dia **5 de junho**.
- Entregue apenas um arquivo de código-fonte, como anexo e com extensão `.c`.
- Constatação de plágio resultará em nota zero para todos os envolvidos, independentemente de quem foi ativo e quem foi passivo.
- O enunciado foi projetado para um prazo menor que o estabelecido acima. Você pode entregar em cima da hora, mas **problemas técnicos do Blackboard não serão aceitos como justificativa para atrasos.**