

Aula Prática 8 - Roteiro

21/09/2021 - Roteiro referente à aula prática 8 - Algoritmos de Codificação Base16, Base32 e Base64.

Versão: 23/09/2021 (Correção do exemplo do item 4)

Versão: 01/10/2021 (Alteração no prazo de entrega)

Prazo: 02/10/2021 - 08:00

Valor: 10,0 - Peso: 3

- Leia este enunciado com **MUITA** atenção até o final antes de iniciar o trabalho.
- Os arquivos solicitados deverão estar disponíveis nos diretórios correspondentes (**Aulas-Práticas** e **RCS**) até o prazo estipulado acima. Cuidado com os nomes dos diretórios e dos arquivos. Deverão ser exatamente os definidos neste roteiro (maiúsculas, minúsculas, caracteres especiais e extensões, se existentes).
- **As tarefas deverão ser executadas na ordem solicitada neste roteiro.**
- Os arquivos de dependências deverão possibilitar que a compilação e a linkedição sejam executadas utilizando-se tanto o *gcc*, quanto o *clang*. A seleção da ferramenta utilizada deverá ser realizada no momento da execução do comando *make*. O *gcc* deverá ser considerado o valor padrão para a compilação e para a *linkedição*.

Para a definição da ferramenta desejada deverá ser definida uma macro (no *FreeBSD*) ou um argumento com o valor desejado (no *CentOS*). As duas macros deverão ser *GCC* e *CLANG* (definidas usando a opção *-D*). O argumento, identificado por *cc*, deverá ser igual a *GCC* ou *CLANG*.

- Independente da ferramenta utilizada para a compilação, o *flag* de compilação deverá ser definido no instante da execução do comando *make*. O valor padrão para este *flag* deverá ser *"-Wall -ansi"* (sem as aspas).

Durante a execução do comando *make* poderão ser definidos outros valores para este *flag* (mantendo a opção de exibir todas as mensagens de advertência) através de macros ou através de argumentos (de forma semelhante àquela utilizada para definir o compilador/linkeditor). No *FreeBSD* deverão ser definidas as macros *ANSI*, *C99* e *C11*, enquanto que no *CentOS* deverá ser definido o argumento *dialeto* com os valores *ANSI*, *C99* ou *C11*.

- Crie uma macro, *DIALETO*, contendo o dialeto a ser utilizado na compilação do código. Esta macro será inicialmente igual a *"ansi"* e poderá ser alterada para *"c99"* ou *"c11"* de acordo com o esquema definido acima.
- O *flag* de linkedição deverá ser igual a *"-Wall"* (sem as aspas).
- Seguem alguns exemplos:

make - compila/linkedita (tanto no *FreeBSD*, quanto no *CentOS*) com a ferramenta e dialeto padrões, ou seja, *gcc* e *ANSI* respectivamente.

make -DGCC - compila/linkedita usando o *gcc* e o dialeto *ANSI* (somente *FreeBSD*).

make -DCLANG - compila/linkedita usando o *clang* e o dialeto *ANSI* (somente *FreeBSD*).

make cc=GCC - compila/linkedita usando o *gcc* e o dialeto *ANSI* (somente *CentOS*).

make cc=CLANG - compila/linkedita usando o *clang* e o dialeto *ANSI* (somente *CentOS*).

make -DCLANG -DC11 - compila/linkedita usando o *clang* e o dialeto *C11* (somente *FreeBSD*).

make cc=CLANG dialeto=C99 - compila/linkedita usando o *clang* e o dialeto *C99* (somente *CentOS*).

- Inclua, no início de todos os arquivos solicitados, os seguintes comentários:

Universidade Federal do Rio de Janeiro
Escola Politecnica
Departamento de Eletronica e de Computacao

EEL270 - Computacao II - Turma 2021/1
Prof. Marcelo Luiz Drumond Lanza
Autor: <nome completo>
Descricao: <descrição sucinta dos objetivos do programa>

\$Author\$
\$Date\$
\$Log\$

- Inclua, no final de todos os arquivos solicitados, os seguintes comentários:

\$RCSfile\$

Referência: RFC 4648 (<https://datatracker.ietf.org/doc/html/rfc4648>)

1. Crie o arquivo "*aulao801.h*" contendo a definição dos tipos *byte* e *tipoErros*. Inclua neste arquivo o protótipo da função *CodificarBase16* conforme definido abaixo.

tipoErros

*CodificarBase16 (byte * /* (E) */, unsigned long long /* (E) */, char * /* (S) */);*

A macro referente à combinação ifndef e define, como por exemplo _AULA08_, deverá ser definida como uma string valendo:

"@(#)aulao8.h \$Revision\$"

2. Crie o arquivo "*aulao801.c*" contendo o código fonte da função *CodificarBase16*. Esta função deverá receber um conjunto de bytes e o número de bytes recebidos. Além disso, deverá devolver a *string* correspondente (gerada utilizando-se a codificação em **Base16**). A função deverá retornar *ok* (ZERO) ou o código de erro correspondente. A definição do algoritmo **Base16** pode ser encontrada no **RFC4648**.
3. Inclua, nos arquivos de dependências, as macros *LIBBASEOBS* (correspondendo ao arquivo "*aulao801.o*") e *LIBBASE* (correspondendo ao arquivo "*libbase.a*"). O valor da macro *LIBS* deverá ser atualizado de forma que inclua o valor desta última macro. Inclua o objetivo correspondente, ou seja, *libbase.a*, com a(s) dependência(s) e comando(s) necessários para atingir este objetivo.
4. Crie o arquivo "*aulao802.c*" contendo o código fonte de um programa de testes para a função da *CodificarBase16*. Este programa deverá receber, através dos argumentos de linha de comando, o número de bytes a codificar, seguido pelos bytes em notação decimal (valores entre 0 e 255). O programa deverá exibir a *string* gerada pela função ou a mensagem de erro correspondente.

Exemplo:

./aulao802 10 32 171 224 80 24 41 113 120 255 0

5. Inclua, nos arquivos de dependências, as macros *AULA0802OBS* e *AULA08*. Altere o valor da macro *EXECS*, de forma que inclua o valor da macro *AULA08*. Inclua também os objetivos *aulao8* e *aulao802* com os comandos correspondentes (que deverão usar a biblioteca *libbase.a*).
6. Gere e teste as 16 versões do executável *aulao802*.
7. Submeta os arquivos "*aulao801.h*", "*aulao801.c*", "*aulao802.c*" e "**makefile*" ao sistema de controle de versão.
8. Recupere uma cópia de leitura do arquivo "*aulao802.c*" e uma cópia de escrita dos arquivos "*aulao801.h*", "*aulao801.c*" e "**makefile*".
9. Inclua, no arquivo "*aulao801.h*", a definição do protótipo da função *DecodificarBase16*.

tipoErros

*DecodificarBase16 (char * /* (E) */, byte * /* (S) */, unsigned long long * /* (S) */);*

10. Inclua, no arquivo "*aulao801.c*", o código fonte da função *DecodificarBase16*. Esta função deverá receber uma string (a princípio gerada utilizando-se o algoritmo Base16) e deverá devolver o conjunto de bytes original (resultante da decodificação Base16) e o número de bytes resultantes

desta decodificação. A função deverá retornar *ok* (ZERO) ou o código de erro correspondente.

11. Crie o arquivo "aulao803.c" contendo o código fonte de um programa de testes para a função da *DecodificarBase16*. Este programa deverá receber, através dos argumentos de linha de comando, a string a ser decodificada. O programa deverá exibir, em decimal sempre com três caracteres, o conjunto de bytes gerado pela função ou a mensagem de erro correspondente.

Exemplo:

```
./aulao803 666F6F626172FoFF
```

12. Inclua as declarações necessárias nos arquivos de dependências. Lembre-se que o executável *aulao803* deve ser gerado utilizando-se a biblioteca "libbase.a".
13. Gere e teste as doze versões do executável *aulao803*.
14. Submeta os arquivos "aulao801.h", "aulao801.c", "aulao803.c" e "makefile" ao sistema de controle de versão.
15. Recupere uma cópia de leitura do arquivo "aulao803.c" e uma cópia de escrita dos arquivos "aulao801.h", "aulao801.c" e "makefile".
16. Inclua, no arquivo "aulao801.h", a definição do tipo *tipoAlfabetoBase32* e do protótipo da função *CodificarBase32*. O tipo *tipoAlfabetoBase32* deverá ser um tipo enumerado contendo os valores *basico* e *estendido*.

tipoErros

```
CodificarBase32 (byte /* (E) */, unsigned long long /* (E) */, tipoAlfabetoBase32 /* (E) */,  
char /* (S) */);
```

17. Inclua, no arquivo "aulao801.c", o código fonte da função *CodificarBase32*. Esta função deverá receber um conjunto de bytes, o número de bytes recebidos e o alfabeto desejado. Além disso, deverá devolver a *string* correspondente (gerada utilizando-se a codificação em **Base32** e o alfabeto desejado). A função deverá retornar *ok* (ZERO) ou o código de erro correspondente.
18. Crie o arquivo "aulao804.c" contendo o código fonte de um programa de testes para a função da *CodificarBase32*. Este programa deverá receber, através dos argumentos de linha de comando, o alfabeto desejado para a codificação (0 - Básico e 1 - Estendido) e o número de bytes a codificar, seguido pelos bytes em notação hexadecimal. O programa deverá exibir a *string* gerada pela função ou a mensagem de erro correspondente.

Exemplos:

```
./aulao804 0 10 20 AB E0 50 18 29 71 78 FF 00
```

```
./aulao804 1 10 20 AB E0 50 18 29 71 78 FF 00
```

19. Inclua as declarações necessárias nos arquivos de dependências. Lembre-se que o executável *aulao804* deve ser gerado utilizando-se a biblioteca "libbase.a".
20. Gere e teste as 16 versões do executável *aulao804*.
21. Submeta os arquivos "aulao801.h", "aulao801.c", "aulao804.c" e "makefile" ao sistema de controle de versão.
22. Recupere uma cópia de leitura do arquivo "aulao804.c" e uma cópia de escrita dos arquivos "aulao801.h", "aulao801.c" e "makefile".
23. Inclua, no arquivo "aulao801.h", a definição do protótipo da função *DecodificarBase32*.

tipoErros

```
DecodificarBase32 (char /* (E) */, tipoAlfabetoBase32 /* (E) */, byte /* (S) */, unsigned  
long long /* (S) */);
```

24. Inclua, no arquivo "aulao801.c", o código fonte da função *DecodificarBase32*. Esta função deverá receber uma *string* (a princípio gerada utilizando-se o algoritmo **Base32**) e o alfabeto utilizado na codificação. Além disso, deverá devolver o conjunto de bytes original (resultante da decodificação **Base32** com o alfabeto correspondente) e o número de bytes resultantes desta decodificação. A função deverá retornar *ok* (ZERO) ou o código de erro correspondente.
25. Crie o arquivo "aulao805.c" contendo o código fonte de um programa de testes para a função da *DecodificarBase32*. Este programa deverá receber, através dos argumentos de linha de comando, o alfabeto a ser utilizado na decodificação (0 - Básico e 1 - Estendido) e a string a ser decodificada. O programa deverá exibir, em hexadecimal sempre com dois caracteres e utilizando letras maiúsculas, o conjunto de bytes gerado pela função ou a mensagem de erro

correspondente.

Exemplo:

./aulao805 0 MZXW6YTB

./aulao805 1 CPNMUOJ1

26. Inclua as declarações necessárias nos arquivos de dependências. Lembre-se que o executável *aulao805* deve ser gerado utilizando-se a biblioteca *"libbase.a"*.
27. Gere e teste as 16 versões do executável *aulao805*.
28. Submeta os arquivos *"aulao801.h"*, *"aulao801.c"*, *"aulao805.c"* e *"*makefile"* ao sistema de controle de versão.
29. Recupere uma cópia de leitura do arquivo *"aulao805.c"* e uma cópia de escrita dos arquivos *"aulao801.h"*, *"aulao801.c"* e *"*makefile"*.
30. Inclua, no arquivo *"aulao801.h"*, a definição do tipo *tipoFinalLinha* e do protótipo da função *CodificarBase64*. O tipo *tipoFinalLinha* deverá ser um tipo enumerado contendo os valores *desabilitado* e *habilitado*.

tipoErros

CodificarBase64 (byte * /* (E) */, unsigned long long /* (E) */, tipoFinalLinha /* (E) */, char * /* (S) */);

31. Inclua, no arquivo *"aulao801.c"*, o código fonte da função *CodificarBase64*. Esta função deverá receber um conjunto de bytes, o número de bytes recebidos e o indicador do uso ou não dos caracteres de final de linha na *string* gerada. Além disso, deverá devolver a *string* correspondente (gerada utilizando-se a codificação em **Base64** com ou sem caracteres de final de linha, conforme definido pelo terceiro argumento). A função deverá retornar *ok* (ZERO) ou o código de erro correspondente.
32. Crie o arquivo *"aulao806.c"* contendo o código fonte de um programa de testes para a função da *CodificarBase64*. Este programa deverá receber, através dos argumentos de linha de comando, o indicador de final de linha (0 - Desabilitado e 1 - Habilitado) e o número de bytes a codificar, seguido pelos bytes em notação hexadecimal. O programa deverá exibir a *string* gerada pela função ou a mensagem de erro correspondente.

Se o final de linha estiver habilitado, os caracteres de final de linha deverão ser adicionados quando o número de caracteres atingir o comprimento máximo permitido para uma linha de um arquivo codificado em **Base64**.

Exemplos:

./aulao806 0 10 20 AB E0 50 18 29 71 78 FF 00

./aulao806 1 10 20 AB E0 50 18 29 71 78 FF 00

33. Inclua as declarações necessárias nos arquivos de dependências. Lembre-se que o executável *aulao806* deve ser gerado utilizando-se a biblioteca *"libbase.a"*.
34. Gere e teste as 16 versões do executável *aulao806*.
35. Submeta os arquivos *"aulao801.h"*, *"aulao801.c"*, *"aulao806.c"* e *"*makefile"* ao sistema de controle de versão.
36. Recupere uma cópia de leitura do arquivo *"aulao806.c"* e uma cópia de escrita dos arquivos *"aulao801.h"*, *"aulao801.c"* e *"*makefile"*.
37. Inclua, no arquivo *"aulao801.h"*, a definição do protótipo da função *DecodificarBase64*.

tipoErros

DecodificarBase64 (char * /* (E) */, tipoFinalLinha /* (E) */, byte * /* (S) */, unsigned long long /* (S) */);

38. Inclua, no arquivo *"aulao801.c"*, o código fonte da função *DecodificarBase64*. Esta função deverá receber uma *string* (a princípio gerada utilizando-se o algoritmo **Base64**) e o indicador de uso ou não de caracteres de final de linha na codificação. Além disso, deverá devolver o conjunto de bytes original (resultante da decodificação **Base64** correspondente) e o número de bytes resultantes desta decodificação. A função deverá retornar *ok* (ZERO) ou o código de erro correspondente.
39. Crie o arquivo *"aulao807.c"* contendo o código fonte de um programa de testes para a função da *DecodificarBase64*. Este programa deverá receber, através dos argumentos de linha de comando,

o indicador de uso ou não de caracteres de final de linha (0 - Desabilitado e 1 - Habilitado) e a *string* a ser decodificada. O programa deverá exibir, em hexadecimal sempre com dois caracteres e utilizando letras maiúsculas, o conjunto de bytes gerado pela função ou a mensagem de erro correspondente.

Exemplo:

```
./aulao807 0 Zm9vYmFy
```

```
./aulao807 1 Zm9vYmFy
```

40. Inclua as declarações necessárias nos arquivos de dependências. Lembre-se que o executável *aulao807* deve ser gerado utilizando-se a biblioteca *"libbase.a"*.
41. Gere e teste as 16 versões do executável *aulao807*.
42. Submeta os arquivos *"aulao801.h"*, *"aulao801.c"*, *"aulao807.c"* e *"*makefile"* ao sistema de controle de versão.
43. Recupere uma cópia de leitura dos arquivos *"aulao801.h"*, *"aulao801.c"* e *"aulao807.c"* e uma cópia de escrita dos arquivos *"*makefile"*.
44. Crie o arquivo *"aulao808.c"* contendo um programa de testes para a função *CodificarBase64*. Este programa deverá receber, via argumentos da CLI, o nome do arquivo a ser codificado e o nome do arquivo a ser gerado utilizando a função *CodificarBase64*.
45. Inclua as declarações necessárias nos arquivos de dependências. Lembre-se que o executável *aulao808* deve ser gerado utilizando-se a biblioteca *"libbase.a"*.
46. Gere e teste as doze versões do executável *aulao808*.
47. Submeta os arquivos *"aulao808.c"* e *"*makefile"* ao sistema de controle de versão.
48. Recupere uma cópia de leitura do arquivo *"aulao808.c"* e uma cópia de escrita dos arquivos *"*makefile"*.
49. Crie o arquivo *"aulao809.c"* contendo um programa de testes para a função *DecodificarBase64*. Este programa deverá receber, via argumentos da CLI, o nome do arquivo codificado e o nome do arquivo a ser gerado utilizando a função *DecodificarBase64*.
50. Inclua as declarações necessárias nos arquivos de dependências. Lembre-se que o executável *aulao809* deve ser gerado utilizando-se a biblioteca *"libbase.a"*.
51. Gere e teste as 16 versões do executável *aulao809*.
52. Submeta os arquivos *"aulao809.c"* e *"*makefile"* ao sistema de controle de versão.
53. Recupere uma cópia de leitura do arquivo *"aulao809.c"* e uma cópia de escrita dos arquivos *"*makefile"*.
54. Limpe o diretório (*make clean-all*).