Aula Prática 8 – Roteiro

Objetivos:

- Algoritmos de Codificação Base16, Base32 e Base64.

Versão Inicial: 27/06/2023

Prazo: 11/07/2023 – 8:00

Observações:

- Leia este enunciado com **MUITA** atenção até o final antes de iniciar o trabalho.
- Os arquivos solicitados deverão estar disponíveis nos diretórios correspondentes (***Aulas-Praticas*** e ***RCS***) até o prazo estipulado acima. Cuidado com os nomes dos diretórios e dos arquivos. Deverão ser exatamente os definidos neste roteiro (incluindo 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 que 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 como o valor padrão para a ferramenta de compilação e de *linkedição*.

  Para a definição da ferramenta desejada, deverá ser utilizada uma macro no *FreeBSD* e um argumento com o valor desejado no *Linux*. As duas macros utilizadas deverão ser ***GCC*** e ***CLANG*** (definidas usando a opção de linha de comando **-D** do comando ***make***). O argumento, identificado por ***cc***, deverá ser igual a ***GCC*** ou a ***CLANG***.
- Independente da ferramenta utilizada para a compilação, as opções de compilação poderão ser redefinidas no instante da execução do comando ***make*** (mantendo-se a exibição de todas as mensagens de advertência, definida pelo valor ***-Wall***). O valor padrão para estas opções deverá ser ***-Wall -ansi***.

  Estas opções poderão ser redefinidas 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***, ***C89***, ***C90***, ***C99*** e ***C11***, enquanto que no *Linux* deverá ser definido o argumento ***dialeto*** com um dos seguintes valores ***ANSI***, ***C89***, ***C90***, ***C99*** ou ***C11***.
- Os arquivos de dependências deverão incluir a macro ***DIALECT*** contendo o dialeto a ser utilizado na compilação do código. Esta macro será inicialmente igual a ***ansi*** e poderá ser alterada para ***c89***, ***c90***, ***c99*** ou ***c11*** de acordo com o esquema definido acima.
- Os arquivos de dependências deverão incluir também a macro ***STANDARD*** contendo a opção de linha de comando correspondente ao dialeto selecionado. Se, por exemplo, o dialeto selecionado for o ***ANSI***, esta macro deverá ser igual a ***-ansi***. Por outro lado, se o dialeto for uma das outras quatro opções, esta macro deverá ser igual a ***-std=CXX***, onde ***XX*** deverá ser substituído pelo número correspondente (se o dialeto for igual a ***C89***, *XX* deverá ser igual a ***89***, se o dialeto for igual a ***C90***, ***XX*** deverá igual a ***90*** e assim por diante).
- A *linkedição* deverá utilizar a opção ***-Wall***.
- Cuidado com os nomes das macros e dos rótulos. Deverão ser exatamente os definidos neste roteiro (maiúsculas, minúsculas, caracteres especiais e extensões, se existentes).
- Todos os rótulos solicitados no roteiro são obrigatórios. Durante a correção, caso não seja possível alcançar os objetivos (binários e/ou bibliotecas e limpezas de código) solicitados, a nota correspondente ao item/aula em questão será igual a zero.

- Seguem alguns exemplos (todos devem funcionar):
  - ***make*** - compila/*linkedita* (tanto no *FreeBSD*, quanto no *Linux*) com a ferramenta e dialeto padrões, ou seja, ***gcc*** e **ANSI** respectivamente.
  - ***make clean-all all***
  - ***make clean-all aula01***
  - ***make clean aula0101***
  - ***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 *Linux*).
  - ***make cc=CLANG*** - compila/*linkedita* usando o ***clang*** e o dialeto **ANSI** (somente *Linux*).
  - ***make -DCLANG -DC89*** - compila/*linkedita* usando o ***clang*** e o dialeto **C89** (somente *FreeBSD*).
  - ***make -DCLANG -DC11*** - compila/*linkedita* usando o ***clang*** e o dialeto **C11** (somente *FreeBSD*).
  - ***make cc=CLANG dialero=C99*** - compila/*linkedita* usando o ***clang*** e o dialeto **C99** (somente *Linux*).
  - ***make cc=GCC dialeto=C90*** - compila/*linkedita* usando o ***gcc*** e o dialeto **ANSI** (somente *Linux*).

- Inclua, no início de todos os arquivos solicitados (código-fonte e arquivos de dependências), os seguintes comentários (**sem caracteres especiais**):

```
Universidade Federal do Rio de Janeiro
Escola Politecnica
Departamento de Eletronica e de Computacao
EEL270 - Computacao II - Turma 2023/1
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, o seguinte comentário:

```
$RCSfile$
```

Referência:

RFC 4648:

https://datatracker.ietf.org/doc/html/rfc4648

PDF:

https://www.rfc-editor.org/rfc/pdfrfc/rfc4648.txt.pdf

1. Crie o arquivo ***aula0801.h*** contendo as definições 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***, ou seja, a macro **_AULA08_**, deverá ser definida como uma *string* valendo: "@(#)aula0801.h $Revision$".

2. Crie o arquivo ***aula0801.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 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 ***LIBBASEOBJS*** (correspondendo ao arquivo ***aula0801.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 ***aula0802.c*** contendo o código-fonte de um programa de testes para a função *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, no formato mostrado no exemplo abaixo, a *string* gerada pela função ou a mensagem de erro correspondente. Note que os 5 sinais de maior deverão estar em negrito (fundo branco, texto em preto) e que a *string* gerada deverá ser exibida na cor verde. Mensagens de erro deverão ser exibidas usando o vermelho.

   Exemplo:

```
./aula0802 10 32 171 224 80 24 41 113 120 255 0
>>>>> 20ABE05018297178FF00
```

5. Inclua, nos arquivos de dependências, as macros ***AULA0802OBJS*** e ***AULA08***. Altere o valor da macro ***EXECS***, de forma que inclua o valor da macro ***AULA08***. Inclua também os objetivos ***aula08*** e ***aula0802*** com os comandos correspondentes (que deverão usar a biblioteca ***libbase.a***).

6. Gere e teste as 20 versões do executável ***aula0802***.

7. Submeta os arquivos ***aula0801.h***, ***aula0801.c***, ***aula0802.c*** e arquivos de dependências ao sistema de controle de versão.

8. Recupere uma cópia de leitura do arquivo ***aula0802.c*** e uma cópia de escrita dos demais arquivos.

9. Inclua, no arquivo ***aula0801.h***, a definição do protótipo da função *DecodificarBase16*.

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

10. Inclua, no arquivo ***aula0801.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 em ***Base16***) e o número de bytes resultantes desta decodificação. A função deverá retornar ZERO ou o código de erro correspondente.

11. Crie o arquivo ***aula0803.c*** contendo o código-fonte de um programa de testes para a função *DecodificarBase16*. Este programa deverá receber, através dos argumentos de linha de comando, a *string* a ser decodificada. O programa deverá exibir o conjunto de bytes gerado pela função ou a mensagem de erro correspondente. Cada byte gerado deverá ser exibido em

notação decimal, usando sempre três dígitos e utilizando as cores mostradas no exemplo abaixo. Note que os 5 sinais de maior deverão estar em negrito (fundo branco, texto em preto) e que a saída gerada deverá ser exibida na cor verde. Mensagens de erro deverão ser exibidas usando o vermelho.

```
Exemplo:

./aula0803 666F6F626172F0FF
>>>>> 102 111 111 098 097 114 240 255
```

12. Inclua as declarações necessárias nos arquivos de dependências. Lembre-se que o executável ***aula0803*** deverá ser gerado utilizando-se a biblioteca ***libbase.a***.

13. Gere e teste as 20 versões do executável ***aula0803***.

14. Submeta os arquivos ***aula0801.h***, ***aula0801.c***, ***aula0803.c*** e arquivos de dependências ao sistema de controle de versão.

15. Recupere uma cópia de leitura do arquivo ***aula0803.c*** e uma cópia de escrita dos demais arquivos.

16. Inclua, no arquivo ***aula0801.h***, as definições 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) */,
                 unigned long long /* (E) */,
                 tipoAlfabetoBase32 /* (E) */,
                 char * /* (S) */);
```

17. Inclua, no arquivo ***aula0801.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 ZERO ou o código de erro correspondente.

18. Crie o arquivo ***aula0804.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), o número de bytes a codificar, seguido pelos bytes em notação hexadecimal. O programa deverá exibir, no formato mostrado no exemplo abaixo, a *string* gerada pela função ou a mensagem de erro correspondente. Note que os 5 sinais de maior deverão estar em negrito (fundo branco, texto em preto) e que a *string* gerada deverá ser exibida na cor verde. Mensagens de erro deverão ser exibidas usando o vermelho.

```
Exemplos:

./aula0804 0 10 20 AB E0 50 18 29 71 78 FF 00
>>>>> ECV6AUAYFFYXR7YA
./aula0804 1 10 20 AB E0 50 18 29 71 78 FF 00
>>>>> 42LU0K0O55ONHVO0
```

19. Inclua as declarações necessárias nos arquivos de dependências. Lembre-se que o executável ***aula0804*** deverá ser gerado utilizando-se a biblioteca ***libbase.a***.

20. Gere e teste as 20 versões do executável **aula0804**.

21. Submeta os arquivos ***aula0801.h***, ***aula0801.c***, ***aula0804.c*** e arquivos de dependências ao sistema de controle de versão.

22. Recupere uma cópia de leitura do arquivo ***aula0804.c*** e uma cópia de escrita dos demais arquivos.

23. Inclua, no arquivo ***aula0801.h***, a definição do protótipo da função *DecodificarBase32*.

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

24. Inclua, no arquivo ***aula0801.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 em Base32 com o alfabeto correspondente) e o número de bytes resultantes desta decodificação. A função deverá retornar ZERO ou o código de erro correspondente.

25. Crie o arquivo ***aula0805.c*** contendo o código fonte de um programa de testes para a função *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 notação hexadecimal (sempre com dois caracteres e utilizando letras maiúsculas), o conjunto de bytes gerado pela função (utilizando as cores mostradas no exemplo abaixo) ou a mensagem de erro correspondente. Note que os 5 sinais de maior deverão estar em negrito (fundo branco, texto em preto) e que a saída gerada deverá ser exibida na cor verde. Mensagens de erro deverão ser exibidas usando o vermelho.

```
Exemplo:
./aula0805 0 MZXW6YTB
>>>>> 66 6F 6F 62 61
./aula0805 1 CPNMUOJ1
>>>>> 66 6F 6F 62 61
```

26. Inclua as declarações necessárias nos arquivos de dependências. Lembre-se que o executável ***aula0805*** deverá ser gerado utilizando-se a biblioteca ***libbase.a***.

27. Gere e teste as 20 versões do executável **aula0805**.

28. Submeta os arquivos ***aula0801.h***, ***aula0801.c***, ***aula0805.c*** e arquivos de dependências ao sistema de controle de versão.

29. Recupere uma cópia de leitura do arquivo ***aula0805.c*** e uma cópia de escrita dos demais arquivos.

30. Inclua, no arquivo ***aula0801.h***, as definições dos tipos *tipoFinalLinha* e *tipoAlfabeto64*. Inclua neste arquivo a definição do protótipo da função *CodificarBase64*. O tipo ***tipoFinalLinha*** deverá ser um tipo enumerado contendo os valores ***desabilitado*** e ***habilitado***. O tipo ***tipoAlfabetoBase64*** deverá ser um tipo enumerado contendo os valores ***padrao*** e ***seguro.***

```
tipoErros
CodificarBase64 (byte * /* (E) */,
                 unsigned long long /* (E) */,
                 tipoFinalLinha /* (E) */,
                 tipoAlfabeto64 /* (E) */,
                 char * /* (S) */);
```

31. Inclua, no arquivo ***aula0801.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 ZERO ou o código de erro correspondente.

32. Crie o arquivo ***aula0806.c*** contendo o código-fonte de um programa de testes para a função ***CodificarBase64***. Este programa deverá receber, através dos argumentos de linha de comando, o indicador de final de linha (0 - Desabilitado e 1 – Habilitado), o indicador de alfabeto (0 – Basico e 1 - Seguro) e o número de bytes a codificar, seguido pelos bytes em notação hexadecimal. O programa deverá exibir, no formato mostrado no exemplo abaixo, a *string* gerada pela função ou a mensagem de erro correspondente. Note que os 5 sinais de maior deverão estar em negrito (fundo branco, texto em preto) e que a *string* gerada deverá ser exibida na cor verde. Mensagens de erro deverão ser exibidas usando o vermelho.

    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*** (ver ***RFC***).

```
Exemplos:
./aula0806 0 0 10 20 AB E0 50 18 29 71 78 FF 00
>>>>> IKvgUBgpcXj/AA==
./aula0806 1 0 10 20 AB E0 50 18 29 71 78 FF 00
>>>>> IKvgUBgpcXj/AA==
```

33. Inclua as declarações necessárias nos arquivos de dependências. Lembre-se que o executável ***aula0806*** deverá ser gerado utilizando-se a biblioteca ***libbase.a***.

34. Gere e teste as 20 versões do executável ***aula0806***.

35. Submeta os arquivos ***aula0801.h***, ***aula0801.c***, ***aula0806.c*** e arquivos de dependências ao sistema de controle de versão.

36. Recupere uma cópia de leitura do arquivo ***aula0806.c*** e uma cópia de escrita dos demais arquivos.

37. Inclua, no arquivo ***aula0801.h***, a definição do protótipo da função *DecodificarBase64*.

```
tipoErros
DecodificarBase64 (char * /* (E) */,
                   tipoFinalLinha /* (E) */,
                   tipoAlfabeto64 /* (E) */,
                   byte * /* (S) */,
                   unsigned long long * /* (S) */);
```

38. Inclua, no arquivo ***aula0801.c***, o código-fonte da função *DecodificarBase64*. Esta função deverá receber uma string (a princípio gerada utilizando-se o algoritmo Base64), o indicador de uso ou não de caracteres de final de linha na codificação e o alfabeto desejado. Além disso, deverá devolver o conjunto de bytes original (resultante da decodificação em Base64

correspondente) e o número de bytes resultantes desta decodificação. A função deverá retornar ZERO ou o código de erro correspondente.

39. Crie o arquivo ***aula0807.c*** contendo o código-fonte de um programa de testes para a função *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), o alfabeto utilizado (0 – Basico e 1 - Seguro) e a *string* a ser decodificada. O programa deverá exibir, em notação hexadecimal (sempre com dois caracteres e utilizando letras maiúsculas), o conjunto de bytes gerado pela função (utilizando as cores mostradas no exemplo abaixo) ou a mensagem de erro correspondente. Note que os 5 sinais de maior deverão estar em negrito (fundo branco, texto em preto) e que a saída gerada deverá ser exibida na cor verde. Mensagens de erro deverão ser exibidas usando o vermelho.

```
Exemplo:
./aula0807 0 0 Zm9vYmFy
>>>>> 66 6F 6F 62 61 72
./aula0807 1 1 Zm9vYmFy
>>>>> 66 6F 6F 62 61 72
```

40. Inclua as declarações necessárias nos arquivos de dependências. Lembre-se que o executável ***aula0807*** deverá ser gerado utilizando-se a biblioteca ***libbase.a***.

41. Gere e teste as 20 versões do executável ***aula0807***.

42. Submeta os arquivos ***aula0801.h***, ***aula0801.c***, ***aula0807.c*** e arquivos de dependências ao sistema de controle de versão.

43. Recupere uma cópia de leitura dos arquivos ***aula0801.h***, ***aula0801.c*** e ***aula0807.c*** e uma cópia de escrita dos arquivos de dependências.

44. Crie o arquivo ***aula0808.c*** contendo um programa de testes para a função *CodificarBase64*. Este programa deverá receber, via argumentos da CLI, o alfabeto desejado (0 – Basico e 1 – Seguro), 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 ***aula0808*** deverá ser gerado utilizando-se a biblioteca **libbase.a**.

46. Gere e teste as 20 versões do executável ***aula0808***.

47. Submeta os arquivos ***aula0808.c*** e arquivos de dependências ao sistema de controle de versão.

48. Recupere uma cópia de leitura do arquivo ***aula0808.c*** e uma cópia de escrita dos arquivos de dependências.

49. Crie o arquivo ***aula0809.c*** contendo um programa de testes para a função *DecodificarBase64*. Este programa deverá receber, via argumentos da CLI, o alfabeto utilizado, 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 ***aula0809*** deverá ser gerado utilizando-se a biblioteca ***libbase.a***.

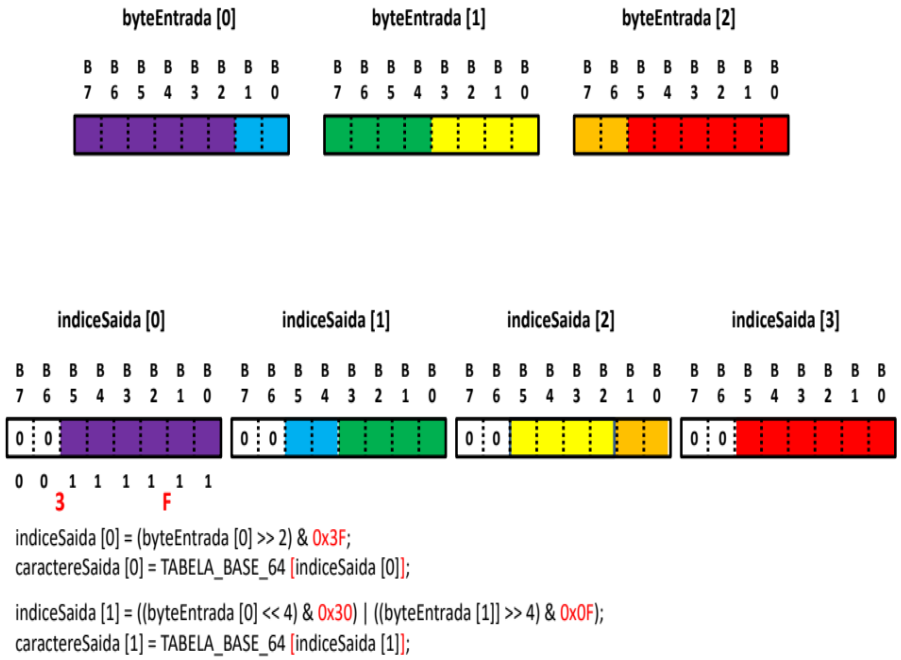
51. Gere e teste as 20 versões do executável ***aula0809***.

52. Submeta os arquivos ***aula0809.c*** e arquivos de dependências ao sistema de controle de versão.

53. Recupere uma cópia de leitura do arquivo ***aula0809.c*** e uma cópia de escrita dos arquivos de dependências.

54. Limpe o diretório (**make clean-all**)

Codificação/Decodificação Base64




```
indiceSaida [0] = (byteEntrada [0] >> 2) & 0x3F;
caractereSaida [0] = TABELA_BASE_64 [indiceSaida [0]];

indiceSaida [1] = ((byteEntrada [0] << 4) & 0x30) | ((byteEntrada [1]] >> 4) & 0x0F);
caractereSaida [1] = TABELA_BASE_64 [indiceSaida [1]];
```