navegação

- Página Inicial
- Equipe
- Atividades
- Localização
- Contatos
- Área de Membros
- Pastas das Disciplinas
  - EEL170 - Computação I
  - EEL270 - Computação II
  - EEL670 - Linguagens de Programação
  - EEL875 - Internet e Arquitetura TCP/IP
  - EEL878 - Redes de Computadores I
  - EEL879 - Redes de Computadores II
  - EEL480 - Sistemas Digitais
- Tutoriais
- Webmail
- Novidades
- Eventos

acessar

Nome do Usuário

Senha

acessar

Esqueceu sua senha?

Aula Prática 09 - Roteiro

15/10/2019 - Roteiro referente à aula prática 09 - Codificação e decodificação em Base64.

Versão: 15/10/2019

Prazo: 22/10/2019 - 6:00

Observações:

- Leia este enunciado com **MUITA** atenção até o final antes de iniciar o trabalho.
- Este roteiro está disponível no formato PDF. Para acessá-lo, clique aqui.
- 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 (maiúsculas, minúsculas, caracteres especiais e extensões, se existentes).
- As tarefas deverão ser executadas na ordem solicitada neste roteiro.
- A compilação e a *linkedição* deverão ser executadas utilizando-se tanto o *gcc*, quanto o *clang* . Em ambos os casos deverão ser utilizados os flags "*-Wall -std=c99*".
- Além disso, deverão ser executadas sem mensagens de advertência e sem mensagens de erro, tanto no *CentOS* 7.x, quanto no *FreeBSD* 11.x.
- No *CentOS* o comando *make* corresponde ao *GNU Make*, enquanto que no *FreeBSD* o comando é nativo. Estas duas variantes não são cem por cento compatíveis e por isso serão necessários dois arquivos de dependências, o *GNUmakefile* e o *BSDmakefile*. No *FreeBSD* o comando *gmake* poderia ser utilizado com o arquivo *GNUmakefile*, mas isto está fora do escopo desta aula.
- Inclua, sempre que necessário, o comando para criar uma cópia do binário com a identificação do sistema operacional e do *compilador/linkeditor* utilizados.
- Inclua, no início de todos os arquivos solicitados (*.c e *makefile), os seguintes comentários:

```
Universidade Federal do Rio de Janeiro
Escola Politecnica
Departamento de Eletronica e de Computacao
EEL270 - Computacao II - Turma 2019/2
Prof. Marcelo Luiz Drumond Lanza
Autor: <nome completo>
Descricao: <descrição sucinta dos objetivos do programa>

$Author$
$Date$
$Log$
```

A codificação/decodificação em Base64 é utilizada para armazenamento e transferência de dados. Esta codificação é definida no RFC (*Request For Comments*) número 4648 (https://www.ietf.org/rfc/rfc4648.txt) e pode ser utilizada por exemplo, para codificar o qualquer conteúdo de forma que o resultado possa ser anexado em uma mensagem eletrônica (*e-mail*).

No algoritmo o conteúdo a ser codificado deverá ser analisado em segmentos de 3 bytes (com exceção do último segmento que pode ter 1, 2 ou 3 bytes). Os 24 bits de cada segmento devem ser divididos em 4 grupos de 6 bits. Cada subgrupo de 6 bits dará origem aos 6 bits menos significativos de cada índice (byte) gerado. Os dois bits mais significativos de cada índice gerado serão sempre iguais a 0 (zero). Desta forma, para cada 3 bytes de entrada serão gerados 4 bytes. Estes quatro bytes deverão ser utilizados como índices para identificar os caracteres correspondentes na tabela Base64.

A tabela Base64 é composta pelos caracteres maiúsculos (A ... Z), seguidos pelos caracteres minúsculos (a ... z), pelos dígitos (0 ... 9), pelo sinal de soma (+) e pela barra (/), nesta ordem. Além disso, o caractere de igualdade (=) é utilizado para preenchimento (*padding*).

Quando o último segmento de dados contiver 2 bytes o último índice não poderá ser gerado. Além disso, os dois bits menos significativos do terceiro índice gerado serão iguais a zero.

Quando o último segmento de dados contiver apenas 1 byte os 2 últimos índices não poderão ser gerados. Além disso, os 4 bits mais significativos do segundo byte de entrada serão substituídos por 0 para a geração do segundo índice.

Sempre que um índice não puder ser gerado, o caractere de saída do algoritmo de codificação será o caractere de preenchimento, isto é, o sinal de igualdade.

Note que o resultado da codificação em Base64 é uma *string*. Para ser utilizada como anexo de um e-mail está *string* deverá ser dividida em linhas com comprimento máximo igual a 76 caracteres e terminadas "\r\n".

1. Baseado nas definições acima, crie o arquivo "*aula0901.h*" contendo a definição da macro CONJUNTO_BASE_64, do tipo *byte*, do tipo *tipoErros* e dos protótipos das funções *CodificarBase64* e *DecodificarBase64*.

   A função *CodificarBase64* deverá receber um vetor de bytes (definido como ponteiro para bytes) e o número de bytes deste vetor. O resultado da codificação utilizando o algoritmo Base64 deverá ser devolvido como uma *string* válida - linhas com comprimento máximo igual a 76 caracteres e terminadas com a sequência "\r\n". A string deverá ser terminada com um caractere *End of String (EOS)* que é igual a '\0'.

   A função *DecodificarBase64* deverá receber uma *string* contendo o resultado correspondente à codificação utilizando o algoritmo Base64. Esta função deverá devolver o conjunto de bytes original e o comprimento (em bytes) deste conjunto.

   As funções acima deverão retornar zero ou o código de erro correspondente.

   O valor da macro correspondente ao arquivo aula0901.h deverá ser igual a:

   "@(#)aula0901.h $Revision$"

   Protótipos:

   tipoErros
   *CodificarBase64 (byte *entrada, unsigned numeroBytes, char *saida);*

   tipoErros
   *DecodificarBase64 (char *entrada, byte *saida, unsigned *numeroBytes);*

2. Crie o arquivo "*aula0901.c*" contendo a implementação da função *CodificarBase64*.

3. Crie o arquivo "*aula0902.c*" contendo a implementação de um programa de testes para a função *CodificarBase64*. Este programa deverá receber, através dos argumentos de linha de comando (CLI), o número de bytes a codificar, seguido pelo valor de cada byte em hexadeciimal (um valor por argumento). A função deverá exibir o ´codigo Base64 gerado ou a mensagem de erro apropriada. Considere que o usuário poderá entrar com no máximo 1024 bytes.

   ./aula0902 5 B0 00 5F 44 32

4. Inclua, nos arquivos de dependências, a macro AULA0902OBJS - correspondendo aos arquivos necessários para criar o executável a partir dos arquivos "*aula0901.c*" e "*aula0902.c*". Além disso, defina a macro AULA09, equivalendo ao executável *"aula0902*", e altere a macro EXECS, de forma que o valor da mesma inclua os executáveis criados na aula 09. Os arquivos de dependências deverão incluir ainda os objetivos *aula09* (todos os executáveis da aula 09) e *aula0902* (executável criado a partir dos arquivos definidos pela macro AULA0902OBJS) com os comandos correspondentes.

5. Crie e teste as quatro versões do executável *aula0902*.

   Submeta os arquivos *aula0901.h*, *aula0901.c*, *aula0902.c* e **makefile* ao sistema de controle de versão.

6. Recupere uma cópia de leitura do arquivo *aula0902.c* e uma cópia de escrita dos arquivos *aula0901.h*, *aula0901.c* e **makefile*.

7. Inclua, no arquivo "*aula0901.c*", a implementação da função *DecodificarBase64*.

8. Crie o arquivo "*aula0903.c*" contendo a implementação de um programa de testes para a função *DecodificarBase64*. Este programa deverá receber, através dos argumentos de linha de comando (CLI), uma string gerada pela função anterior. O programa deverá exibir o código original em hexadecimal sempre com 2 dígitos e usando, quando necesário, letras maiúsculas. Entre 2 bytes exiba um caractere de espaço.
9. Inclua, nos arquivos de dependências, a macro AULA0903OBJS - correspondendo aos arquivos necessários para criar o executável a partir dos arquivos "*aula0901.c*" e "*aula0903.c*". Além disso, altere a macro AULA09, de forma que o valor da mesma inclua o executável "*aula0903*". O arquivo de dependências deverá incluir ainda o objetivo *aula1003* (executável criado a partir dos arquivos definidos pela macro AULA1003OBJS) com os comandos correspondentes.

10. Crie e teste as quatro versões do executável *aula0903*.

11. Submeta os arquivos *aula0901.h*, *aula0901.c*, *aula0903.c* e **makefile* ao sistema de controle de versão.

12. Recupere uma cópia de leitura dos arquivos *aula0901.h*, *aula0901.c* e *aula0903.c* e uma cópia de escrita dos arquivos **makefile*.

Outubro 2019

Do	Se	Te	Qu	Qu	Se	Sa
		1	2	3	4	5
6	7	8	9	10	11	12
13	14	15	16	17	18	19
20	21	22	23	24	25	26
27	28	29	30	31		

notícias

Horário 2017/2
31/07/2017

Estágio Supervisionado - Importante
05/03/2012

Mais notícias...