

Aula Prática 5 - Roteiro

27/04/2021 - Roteiro referente à aula prática 5 - CPF (geração e validação). Versões:

24/08/2021 - Versão inicial

Prazo: 28/08/2021 - 18:00

Valor: 10,0 - Peso: 2

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 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 itens marcados com (*) deverão ser entregues até às 19:00 do dia da aula em questão.

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 utilizada uma macro (no *FreeBSD*) ou um argumento com o valor desejado (no *CentOS*). As duas macros utilizadas 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*, *C89*, *C90*, *C99* e *C11*, enquanto que no *CentOS* deverá ser definido o argumento *dialeto* com os valores *ANSI*, *C89*, *C90*, *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 *"c89"*, *"c99"* ou *"c11"* de acordo com o esquema definido acima.

O *flag* de *linkedição* deverá ser igual a *"-Wall"* (sem as aspas).

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 questão será zero.

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 -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 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:

```
$RCSfile$
```

Um CPF tem a seguinte configuração: XXX.XXX.XXX/XX. Os dois últimos (após a barra) são os dígitos verificadores do CPF.

$$CPF = X_1 X_2 X_3 . X_4 X_5 X_6 . X_7 X_8 X_9 / X_{10} X_{11}$$

Os dígitos verificadores são gerados a partir dos dígitos anteriores. Para cada dígito verificador, deverá ser calculada a soma dos produtos dos dígitos anteriores pelos respectivos pesos. Este resultado (soma) deverá ser dividido por 11. Se o resto desta divisão for igual a 0 ou igual a 1 o dígito verificador em questão será igual a 0. Caso contrário, o dígito verificador será igual à diferença entre 11 e o resto da divisão anterior.

Os pesos dos dígitos para o cálculo do primeiro dígito verificador são (da esquerda para a direita) 10, 9, 8, 7, 6, 5, 4, 3 e 2.

Para o cálculo do segundo dígito verificador os pesos são 11, 10, 9, 8, 7, 6, 5, 4, 3 e 2. Note que o primeiro dígito verificador deverá ser utilizado para o cálculo do segundo dígito.

1. (*) Baseado nas definições acima, crie o arquivo "*aula0501.h*" contendo o protótipo da função *GerarDigitosVerificadoresCpf*. Este arquivo deverá conter também as macros e os tipos necessários para a implementação desta função. A macro referente à combinação *ifndef* e *define*, como por exemplo *_AULA0501_*, deverá ser definida como uma *string* valendo:

```
"@(#)aula0501.h $Revision$"
```

```
tipoErros
```

```
GerarDigitosVerificadoresCpf (byte [ ] /* entrada/saída */);
```

A função *GerarDigitosVerificadoresCpf* deverá receber, no primeiro argumento, os 9 primeiros

dígitos de um CPF e deverá devolver os dígitos verificadores correspondentes (nas décima e décima primeira posições do vetor recebido).

2. (*) Crie o arquivo "aula0501.c" contendo a implementação da função *GerarDigitosVerificadoresCpf*.
3. (*) Crie o arquivo "aula0502a.c" contendo a implementação de um programa de testes para a função *GerarDigitosVerificadoresCpf*. Este programa deverá receber os 10 primeiros dígitos do CPF desejado através de 10 argumentos da linha de comando (CLI). O programa deverá exibir o CPF completo no formato "XXX.XXX.XXX/XX" (sem as aspas). Todos os tratamentos de erro necessários e que não possam realizados na função *GerarDigitosVerificadoresCpf* deverão ser implementados neste programa.

Exemplo:

```
./aula0502a 1 1 1 1 1 1 1 1 1 1  
CPF: 111.111.111/11
```

4. (*) Inclua, nos arquivos de dependências, as macros *AULA0502AOBJS* e *AULA05*. Altere o valor da macro *EXECS*, de forma que inclua o valor da macro *AULA05*. Inclua também os objetivos *aula05* e *aula0502a* com os comandos correspondentes.
5. (*) Gere e teste as 16 versões do executável *aula0502a*.
6. (*) Submeta os arquivos "aula0501.h", "aula0501.c", "aula0502a.c" e "**makefile*" ao sistema de controle de versão.
7. (*) Recupere uma cópia de leitura do arquivo "aula0502a.c" e uma cópia de escrita dos arquivos "aula0501.h", "aula0501.c" e "**makefile*".
8. Crie o arquivo "aula0502b.c" contendo a implementação de um programa de testes para a função *GerarDigitosVerificadoresCpf*. Este programa deverá receber os 9 primeiros dígitos do CPF desejado através de um único argumento da linha de comando (CLI) no formato "XXX.XXX.XXX" (sem as aspas). O programa deverá exibir o CPF completo no formato "XXX.XXX.XXX/XX" (sem as aspas). Todos os tratamentos de erro necessários e que não possam realizados na função *GerarDigitosVerificadoresCpf* deverão ser implementados neste programa.

Exemplo:

```
./aula0502b 222.222.222  
CPF: 222.222.222/22
```

9. Inclua, nos arquivos de dependências, a macro *AULA0502BOBJS* e o objetivo *aula0502b* com os comandos correspondentes. Altere o valor da macro *AULA05*, incluindo o binário correspondente.
10. Gere e teste as 16 versões do executável *aula0502b*.
11. Submeta os arquivos "aula0502b.c" e "**makefile*" ao sistema de controle de versão. 12. Recupere uma cópia de leitura do arquivo "aula0502b.c" e uma cópia de escrita dos arquivos "**makefile*".
13. Inclua, no arquivo "aula0501.h", o protótipo da função *ValidarCpf* e a definição dos tipos necessários.

tipoErros

ValidarCpf (byte [] /* entrada */);

A função *ValidarCpf* deverá receber os 11 dígitos de um CPF (como um vetor de bytes) e deverá retornar *ok* se os dígitos verificadores forem válidos ou o código de erro correspondente. Esta função deverá utilizar a função *GerarDigitosVerificadoresCpf* na sua implementação.

14. Inclua, no arquivo "aula0501.c", a implementação da função *ValidarCpf*. 15. Crie o arquivo "aula0503a.c" contendo a implementação de um programa de testes para a função *ValidarCpf*. Este programa deverá receber os 11 dígitos do CPF desejado através de 11 argumentos da linha de

comando (CLI). Deverá exibir o CPF em questão no formato "XXX.XXX.XXX/XX" (sem aspas), indicando se o mesmo é válido ou inválido. Todos os tratamentos de erro necessários e que não possam realizados na função *ValidarCpf* deverão ser implementados neste programa.

Exemplos:

```
./aula0503a 0 0 0 0 0 0 0 0 0 0
```

CPF: 000.000.000/00 - valido.

```
./aula0503a 0 0 0 0 0 0 0 0 0 1
```

CPF: 000.000.000/01 - invalido.

16. Inclua, nos arquivos de dependências, a macro *AULA0503AOBJS* e o objetivo *aula0503a* com os comandos correspondentes. Altere o valor da macro *AULA05*, incluindo o binário correspondente.
17. Gere e teste as 16 versões do executável *aula0503a*.
18. Submeta os arquivos "*aula0501.h*", "*aula0501.c*", "*aula0503a.c*" e "**makefile*" ao sistema de controle de versão.
19. Recupere uma cópia de leitura do arquivo "*aula0503a.c*" e uma cópia de escrita dos arquivos "*aula0501.h*", "*aula0501.c*" e "**makefile*".
20. Crie o arquivo "*aula0503b.c*" contendo a implementação de um programa de testes para a função *ValidarCpf*. Este programa deverá receber os 11 dígitos do CPF desejado através de um único argumento da linha de comando (CLI) no formato "XXX.XXX.XXX/XX" (sem aspas). Deverá exibir o CPF em questão no formato "XXX.XXX.XXX/XX" (sem aspas), indicando se o mesmo é válido ou inválido. Todos os tratamentos de erro necessários e que não possam realizados na função *ValidarCpf* deverão ser implementados neste programa. Esta função deverá utilizar a função *GerarDigitosVerificadoresCpf* na sua implementação.

Exemplos:

```
./aula0503a 000.000.000/00
```

CPF: 000.000.000/00 - valido.

```
./aula0503a 000.000.000/01
```

CPF: 000.000.000/01 - invalido.

21. Inclua, nos arquivos de dependências, a macro *AULA0503BOBJS* e o objetivo *aula0503b* com os comandos correspondentes. Altere o valor da macro *AULA05*, incluindo o binário correspondente.
22. Gere e teste as 16 versões do executável *aula0503b*.
23. Submeta os arquivos "*aula0503b.c*" e "**makefile*" ao sistema de controle de versão.
24. Recupere uma cópia de leitura do arquivo "*aula0503b.c*" e uma cópia de escrita dos arquivos "**makefile*".
25. Repita todos os itens anteriores trocando o vetor de bytes por uma *string* de acordo com os protótipos abaixo:

tipoErros

```
GerarDigitosVerificadoresCpf(char /* entrada */, char /* saida - DV1 */, char /* saida - DV2 */);
```

Esta função deverá receber, através do primeiro argumento, uma *string* contendo os 9 primeiros dígitos de um possível CPF (incluindo pontos e barra) e deverá devolver, através dos 2 últimos argumentos, os dígitos verificadores correspondentes.

tipoErros

```
ValidarCpf(char *);
```

A função *ValidarCpf* deverá receber uma *string* no formato "XXX.XXX.XXX/XX" (sem aspas) e deverá retornar se esta *string* contém um CPF válido ou o código de erro correspondente.

Não se esqueça de renumerar corretamente os arquivos, ou seja:

"aula0501.h" será renomeado para "aula0504.h".

"aula0501.c" será renomeado para "aula0504.c".

"aula0502a.c" será renomeado para "aula0505a.c".

"aula0502b.c" será renomeado para "aula0505b.c".

"aula0503a.c" será renomeado para "aula0506a.c".

"aula0503b.c" será renomeado para "aula0506b.c".

Não se esqueça de incluir as macros e rótulos necessários nos arquivos de dependências e de submeter os arquivos ao sistema RCS.

26. Limpe o diretório (make clean-all).