mapa do site acessibilidade contato

Portal do Departamento de Engenharia
Eletrônica e de Computação

buscar

página inicial | equipe | atividades | localização | contatos | área de membros | pastas das disciplinas | tutoriais | novidades | eventos

acessar

você está aqui: página inicial → pastas das disciplinas → eel270 - computação ii → turma 2019-2 → aulas práticas → roteiros → aula prática 11 - roteiro

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 11 - Roteiro

05/11/2019 - Roteiro referente à aula prática 11 - Etapa 1.1 do trabalho final.

Prazo: 12/11/2019 - 08:00

Observações:

- Nos itens a seguir, o uso da ferramenta de controle de versão (RCS) é obrigatório. Sempre que submeter um arquivo ao controle de versão (comando *ci*), recupere uma cópia de escrita do mesmo (comando *co -l*).
- Não se esqueça de incluir os respectivos comentários obrigatórios acima do protótipo e acima do cabeçalho de cada função conforme definido em Organização do Código Fonte.
- Nas funções faça sempre todas as validações de argumentos possíveis antes de utilizá-los.
- Cada argumento deverá ser tratado separadamente e em casos de erros deverão ser gerados códigos de erro diferentes para cada situação.
- Funções diferentes poderão retornar os mesmos erros em situações semelhantes, embora o ideal seja ter erros diferentes para cada função.
- Nos itens abaixo substitua as siglas mldl/Mldl/MLDL pelas siglas correspondentes ao seu nome.
- Para cada função auxiliar criada, crie um programa de testes com o mesmo nome da função auxiliar. Por exemplo, para a função *MldlUmlGetLanguageIndex* criada no arquivo *mldlUmlFunctions.c* crie o arquivo *mldlUmlTestGetLanguageIndex.c* contendo o programa de testes da função correspondente.
- Para cada programa solicitado e para cada programa de testes criado, inclua as declarações necessárias nos arquivos de dependências.

1. Leia com atenção as instruções sobre o trabalho final (Trabalho Final).

2. Crie os diretórios necessários para o desenvolvimento da biblioteca (etapa 1) - ver Estrutura de Diretórios.

3. Escreva, no diretório correspondente, ou seja, "*/users/username/private/EEL270/2019-1/MLDL_UML/Sources/HTML*" (ver Estrutura de Diretórios), o código dos arquivos "*index_en-us.html*" e "*index_pt-br.html*". Estes arquivos deverão conter o código HTML necessário para implementar o redirecionamento para a CGI (*Common Gateway Interface*) "*mldlUmlMain.cgi*" incluindo os argumentos "*mldlUmlConfigurationFileName*" - com o valor padrão "*mldl-uml.cfg*" - e "*mldlUmlLanguage*" com o valor correspondente (*mldlUmlEnglish* e *mldlUmlPortuguese* respectivamente).

   Se a(o) aluna(o) implementar a interface com o usuário (*CLI*, *Ncurses* e *Web*) em outro(s) idioma(s), deverá criar também o(s) arquivo(s) "index_*.html" correspondente(s). Neste caso solicite ao professor o(s) nome(s) do(s) arquivo(s) correspondente(s).

   Dicas:
   - Ver a meta informação *REFRESH* (HTML) - www.w3schools.com
   - Argumentos podem ser passados para uma CGI na própria URL (*Uniform Resource Locator*). Como exemplo, para uma CGI cujo nome é "fatorial.cgi", que tem um argumento cujo nome é "numero" recebendo o valor "10", a URL correspondente seria "fatorial.cgi?numero=10" (sem as aspas). No caso do trabalho, a URL deverá incluir também o diretório no qual a CGI está armazenda (ver Estrutura de Diretórios).
   - Para passar dois ou mais argumentos para a CGI o caractere & deverá ser utilizado entre o valor de um argumento e o nome do próximo argumento, como mostrado abaixo.

   ```
   ...="./CGIs/mldlUmlMain.cgi?mldlUmlConfigurationFileName=mldl-uml.cfg&mldlUmlLanguage=mldlUmlEnglish".
   ```

4. Crie o esqueleto (comentários iniciais/finais e onde for apropriado as diretivas do pré-processador) dos arquivos definidos em Organização do Código Fonte (somente os relacionados com o desenvolvimento da biblioteca de gerenciamento de usuários). Nenhum destes arquivos deverá ser submetido à ferramenta de controle de versão neste instante.

5. Defina, no arquivo *"mldlUmlTypes.h"*, o tipo *mldlUmlLanguageType* (enumerado) incluindo os valores correspondentes aos idiomas usados na interface com o usuário. Deverão ser definidos no mínimo dois elementos: *mldlUmlEnglish* = 0 e *mldlUmlPortuguese* = 1. Não se esqueça de incluir um último elemento no tipo correspondendo à quantidade de idiomas (por exemplo, *mldlUmlLanguagesAmount*). Submeta o arquivo ao controle de versão.

6. Inclua, no arquivo "*mldlUmlFunctions.h*", o protótipo da função *MldlUmlGetLanguageIndex*. Está função deverá receber a *string* correspondente ao idioma desejado ("*mldlUmlEnglish*", "*mldlUmlPortuguese*", etc.) e deverá retornar o valor inteiro correspondente a este idioma (tipo *mldlUmlLanguageType*). Submeta o arquivo ao controle de versão.

   ```
   mldlUmlLanguageType
   MldlUmlGetLanguageIndex (char *);
   ```

7. Inclua, no arquivo "*mldlUmlFunctions.c*",a implementação da função *MldlUmlGetLanguageIndex*. Esta função deverá retornar *mldlUmlEnglish* em todos os casos em que não for possível identificar o idioma (ponteiro recebido é *NULL*, *string* recebida tem comprimento zero, etc.). Faça um programa para testar a função e após todos os testes necessários submeta o arquivo ao controle de versão.

8. Inclua, no arquivo "*mldlUmlErrors.h*", a definição do tipo enumerado *mldlUmlErrorType* e dos protótipos das funções de conversão de códigos de erro em mensagens de erro, ou seja, *MldlUmlGetCliErrorMessage*, *MldlUmlGetNcursesErrorMessage* e *MldlUmlGetWebErrorMessage* (ver Organização do Código). Por enquanto, o tipo enumerado deverá conter apenas o elemento correspondendo à execução da tarefa com sucesso (valor igual a 0) e o último elemento correspondendo à quantidade de códigos de erro existentes em um dado momento. Este tipo deverá ser alterado sempre que for necessário definir um novo código de erro.

   ```
   char *
   MldlUmlGetCliErrorMessage (mldlUmlErrorType, mldlUmlLanguageType);

   char *
   MldlUmlGetNcursesErrorMessage (mldlUmlErrorType, mldlUmlLanguageType);

   char *
   MldlUmlGetWebErrorMessage (mldlUmlErrorType, mldlUmlLanguageType);
   ```

9. Inclua, no arquivo "*mldlUmlErrors.c*", a definição das variáveis (globais) *mldlUmlCliErrorMessages*, *mldlUmlNcursesErrorMessages* e *mldlUmlWebErrorMessages*. Estas variáveis deverão ser matrizes bidemensionais de *strings* (ponteiro para *char*). A primeira dimensão da matriz deverá indicar o idioma (*mldlUmlEnglish* = 0, *mldlUmlPortuguese* = 1, ...), enquanto que a segunda dimensão deverá indicar o número do erro. Estas variáveis deverão ser preenchidas sempre que for necessário definir um novo código de erro. As dimensões, que no exemplo abaixo foram definidas através de macros, deverão ser definidas pelo último elemento dos tipos *mldlUmlLanguageType* e *mldlUmlErrorType*.

   Exemplo:

   ```
   char *mensagensErro [QUANTIDADE_IDIOMAS] [QUANTIDADE_CODIGOS_ERRO] =
   {
     {
       "Ok",
       "File error",
       "Memory alocation error",
       "Invalid password"
     },
     {
       "Sucesso",
       "Erro relacionado com arquivo",
       "Erro alocando memória",
       "Senha incorreta"
     }
   };
   ```

10. Inclua, no arquivo "*mldlUmlErrors.c*", a implementação das funções de conversão de códigos de erro em mensagens de erro, ou seja, *MldlUmlGetCliErrorMessage*, *MldlUmlGetNcursesErrorMessage* e *MldlUmlGetWebErrorMessage* (ver Organização do Código). Faça programas para testar as funções e após todos os testes necessários submeta os arquivos ao controle de versão.

11. Inclua, no arquivo "*mldlUmlUserInterface.h*", a definição do tipo *mldlUmlUserInterfaceMessageNumberType* e dos protótipos das funções *MldlUmlGetCliUserInterfaceMessage*, *MldlUmlGetNcursesUserInterfaceMessage* e *MldlUmlGetWebUserInterfaceMessage*. (ver Organização do Código). Por enquanto, o tipo *mldlUmlUserInterfaceMessageNumberType* deverá conter apenas um elemento indicando a quantidade de *strings* utilizadas para a construção das interfaces com o usuário.

    ```
    char *
    MldlUmlGetCliUserInterfaceMessage (mldlUmlUserInterfaceMessageNumberType, mldlUmlLanguageType);

    char *
    MldlUmlGetNcursesUserInterfaceMessage (mldlUmlUserInterfaceMessageNumberType, mldlumlLanguageType);

    char *
    MldlUmlGetWebUserInterfaceMessage (mldlUmlUserInterfaceMessageNumberType, mldlUmlLanguageType);
    ```

12. Inclua, no arquivo "*mldlUmlUserInterface.c*", a definição das variáveis (globais) *mldlUmlCliUserInterfaceMessages*, *mldlUmlNcursesUserInterfaceMessages* e *mldlUmlWebUserInterfaceMessages*. Lembre-se que estas variáveis devem ser matrizes bidemensionais de *strings* (ponteiro para *char* semelhantes àquelas utilizadas para as mensagens de erro). Estas variáveis terão os seus conteúdos atualizados sempre que for necessário definir uma nova *string* para a composição das interfaces com o usuário.

13. Inclua, no arquivo "*mldlUmlUserInterface.c*", a implementação das funções *MldlUmlGetCliUserInterfaceMessage*, *MldlUmlGetNcursesUserInterfaceMessage* e *MldlUmlGetWebUserInterfaceMessage*. (ver Organização do Código). Faça programas para testar a função e após todos os testes necessários submeta os arquivos ao controle de versão.

14. Inclua, no arquivo "*mldlUmlConfig.h*", a definição da constante (MACRO) de configuração ***MLDL_UML_ADMINISTRATOR_USER_IDENTIFIER***. O valor desta macro deverá ser igual ao valor da "*user id*" da(o) aluna(o) na rede DEL. Para obter esse valor basta executar o comando "*id*" a partir do terminal no Linux/FreeBSD.

15. Inclua, no arquivo "*mldlUmlConfig.h*", as macros ***MLDL_UML_PRIVATE_ROOT_DIRECTORY***, ***MLDL_UML_DATA_DIRECTORY*** e ***MLDL_UML_COOKIES_DIRECTORY***, com valores iguais ao caminho absoluto dos diretórios correspondentes, incluindo a barra final (ver Estrutura de Diretórios).

16. Inclua, no arquivo "*mldlUmlConfig.h*", as macros ***MLDL_UML_USERS_DATA_FILENAME***, ***MLDL_UML_INVITED_USERS_DATA_FILENAME***, ***MLDL_UML_REQUESTING_USERS_DATA_FILENAME*** e ***MLDL_UML_LOCKED_USERS_DATA_FILENAME*** com valores iguais a "*users*", "*invited.users*", "*requesting.users*" e "*locked.users*" respectivamente. Submeta o arquivo ao controle de versão.

17. Inclua, no arquivo "*mldlUmlTypes.h*", a definição dos tipos *mldlUmlUserIdentifierType* e *mldlUmlUserProfileType*. Estes tipos deverão corresponder a "*unsigned long long*". Submeta o arquivo ao controle de versão.

18. Inclua o protótipo e a implementação da função *MldlUmlGetAbsoluteFileName* nos arquivos "*mldlUmlFunctions.h*" e "*mldlUmlFunctions.c*" respectivamente. Esta função deverá receber duas *strings* representando um diretório (caminho absoluto podendo conter ou não uma barra final) e um arquivo. Esta função deverá devolver, no terceiro argumento, a *string* resultante da concatenação dos dois primeiros argumentos. A função deverá retornar *mldlUmlOk* (0) ou o código de erro correspondente. Adicione os códigos de erro necessários ao tipo *mldlUmlErrorType* (arquivo "*mldlUmlErrors.h"*) e as *strings* de erro correspondentes às variáveis *mldlUml[Cli|Ncurses|Web]ErrorMessages* (arquivo "*mldlUmlErrors.c*").

    Considere que o comprimento máximo de um caminho absoluto é igual 4096 (incluindo uma barra final), enquanto que o comprimento máximo de um nome de arquivo é igual a 255 (incluindo uma possível extensão). Estes dois comprimentos deverão ser declarados como macros no arquivo "*mldlUmlConst.h*".

    ```
    mldlUmlErrorType
    MldlUmlGetAbsoluteFileName (char *, char *, char *)
    ```

    Observação: Utilizar a função *snprintf*.

19. Inclua, nos arquivos "*mldlUmlFunctions.h*" e "*mldlUmlFunctions.c*", o protótipo e a implementação da função *MldlUmlCheckStringField.* Esta função deverá receber uma *string*, o conjunto de caracteres válidos para a composição desta *string* e os comprimentos mínimo e máximo permitidos para esta *string*. Se a *string* avaliada só contiver caracteres válidos e se o seu comprimento estiver no intervado permitido, esta função deverá retornar *mldlUmlOk*. Caso contrário deverá retornar o código de erro correspondente.

    ```
    mldlUmlErrorType
    MldlUmlCheckStringField (char *, char *,  size_t, size_t);
    ```

20. Inclua, nos arquivos "*mldlUmlFunctions.h*" e "*mldlUmlFunctions.c*", o protótipo e a implementação da função *MldlUmlCheckNickname.* Esta função deverá receber uma *string* correspondendo a um apelido, o conjunto de caracteres válidos para a composição deste apelido e os comprimentos mínimo e máximo permitidos para este campo. Se a *string* avaliada só contiver caracteres válidos, se o seu comprimento estiver no intervado permitido e se ela possuir apenas uma ocorrência do caractere ponto, esta função deverá retornar *mldlUmlOk*. Caso contrário deverá retornar o código de erro correspondente. Um nickname válido é composto por um nome, um ponto e um sobrenome e deverá ter no mínimo 3 e no máximo 65 caracteres. Inclua as macros necessárias no arquivo de constantes.

    ```
    mldlUmlErrorType
    MldlUmlCheckNickname (char *, char *, size_t, size_t);
    ```

21. Inclua, nos arquivos "*mldlUmlFunctions.h*" e "*mldlUmlFunctions.c*", o protótipo e a implementação da função *MldlUmlCheckEmail.* Esta função deverá receber uma *string* correspondendo a um endereço eletrônico, o conjunto de caracteres válidos para a composição deste endereço e os comprimentos mínimo e máximo permitidos para este campo. Se a *string* avaliada só contiver caracteres válidos, se o seu comprimento estiver no intervado permitido e se ela possuir apenas uma ocorrência do caractere arroba, esta função deverá retornar *mldlUmlOk*. Caso contrário deverá retornar o código de erro correspondente.

    ```
    mldlUmlErrorType
    MldlUmlCheckEmail (char *, char *, size_t, size_t);
    ```

22. Inclua, nos arquivos "*mldlUmlFunctions.h*" e "*mldlUmlFunctions.c*", o protótipo e a implementação da função *MldlUmlCreateRandomString.* Esta função deverá receber o conjunto de caracteres válidos (primeiro argumento) e o comprimento desejado para esta *string*. Se nenhum erro foi detectado, a função deverá retornar *mldlUmlOk* e deverá devolver a *string* criada (terceiro argumento). Caso contrário deverá retornar o código de erro correspondente.

    ```
    mldlUmlErrorType
    MldlUmlCreateRandomString (char *, size_t, char *);
    ```

23. Inclua, nos arquivos "*mldlUmlFunctions.h*" e "*mldlUmlFunctions.c*", o protótipo e a implementação da função *MldlUmlCreateNickname*. Esta função deverá receber um nome completo (primeiro argumento) e deverá devolver duas possibilidades para o apelido. A primeira (segundo argumento) deverá composta pelo primeiro nome, seguido por ponto e pelo último sobrenome. A segunda (terceiro argumento) deverá ser composta pelo primeiro nome, seguido por ponto e pelo penúltimo sobrenome. Caso o nome completo só possua duas palavras, a segunda possibilidade deverá ser uma *string* vazia. Esta função deverá retornar *mldlUmlOk* ou o código de erro correspondente. O apelido criado deverá ter no máximo 65 caracteres (incluindo o ponto).

    ```
    mldlUmlErrorType
    MldlUmlCreateNickname (char *, char *, char *);
    ```

24. Inclua, no arquivo "*mldlUmlTypes.h*", a definição do tipo *mldlUmlCryptAlgorithms*. Este tipo deverá ser um enumerado contendo os valores correspondentes aos principais algoritmos de codificação utilizados pela função *crypt*, ou seja *mldlUmlDes*, *mldlUmlMd5*, *mldlUmlSha256* e *mldlUmlSha512*.

25. Inclua, nos arquivos "*mdlUmlFunctions.h*" e "*mldlUmlFunctions.c*", o protótipo e a implementação da função *MldlUmlGetCryptAlgorithm.* Esta função deverá receber uma senha codificada e deverá devolver o algoritmo utilizado para na sua codificação. A função deverá retornar *mldlUmlOk* ou o código de erro correspondente.

    ```
    mldlUmlErrorType
    MldlUmlGetCryptAlgorithm (char *, mldlUmlCryptAlgorithms *);
    ```

26. Inclua, nos arquivos "*mldlUmlFunctions.h*" e "*mldlUmlFunctions.c*", o protótipo e a implementação da função *MldlUmlEncodePasswordWithSpecificAlgorithm.* Esta função deverá receber uma senha plana (primeiro argumento) e o identificador de um algoritmo de codificação (segundo argumento) e deverá devolver a senha codificada (no terceiro argumento) utilizando este algoritmo. A função deverá retornar *mldlumlOk* ou o código de erro correspondente. A parte aleatória dos *salts* deverá ter o comprimento máximo permitido para o algoritmo em questão e deverá ser criada usando a função *MldlUmlCreateRandomString.*

    ```
    mldlUmlErrorType
    MldlUmlEncodePasswordWithSpecificAlgorithm (char *, mldlUmlCryptAlgorithms, char *);
    ```

27. Inclua, nos arquivos "*mldlUmlFunctions.h*" e "*mldlUmlFunctions.c*", o protótipo e a implementação da função *MldlUmlEncodePasswordWithSpecificSalt.* Esta função deverá receber uma senha plana (primeiro argumento) e um *salt* completo (segundo argumento), ou seja, incluindo o identificador de algoritmo e a parte aleatória do *salt*. A função deverá devolver a senha codificada (terceiro argumento) utilizando este *salt*. A função deverá retornar *mldlUmlOk* ou o código de erro correspondente.

    ```
    mldlUmlErrorType
    MldlUmlEncodePasswordWithSpecificSalt (char *, char *, char *);
    ```

28. Inclua, nos arquivos "*mldlUmlFunctions.h*" e "*mldlUmlFunctions.c*", o protótipo e a implementação da função *MldlUmlCheckPassword.* Esta função deverá receber uma senha plana e uma senha codificada e deverá retornar *mldlUmlOk* se a senha plana corresponder à senha codificada. Caso contrário, a função deverá o código de erro correspondente.

    ```
    mldlUmlErrorType
    MldlUmlCheckPassword (char *, char *);
    ```

Novembro 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

notícias

Horário 2017/2
31/07/2017

Estágio Supervisionado - Importante
05/03/2012

Mais notícias...