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?

Organização do Código Fonte

Descreve como deverá ser organizado o código fonte referente à biblioteca e como deverá ser organizado o código fonte referente ao sistema.

- **Biblioteca**

Todos os arquivos necessários para a implementação da biblioteca, incluindo os arquivos *.c e *.h, os arquivos *makefile*, os arquivos *.html estáticos - *index_en-us.html* e *index_pt-br.html* (arquivos correspondentes a outros idiomas, se implementados) - e os arquivos dinâmicos (gerados pelas CGIs), deverão incluir as seguintes informações em um comentário inicial (sem caracteres especiais) utilizando a sintaxe correspondente:

```
  Universidade Federal do Rio de Janeiro
  Escola Politecnica
  Departamento de Eletronica e de Computacao
  Prof. Marcelo Luiz Drumond Lanza
  EEL 270 - Computacao II - Turma 2019/2
  
  $Author$
  $Date$
  $Log$
```

Além desse comentário inicial, cada arquivo de cabeçalho deverá conter as diretivas de preprocessador (*ifndef*, *define* e *endif*):

```
#ifndef MACRO
#define MACRO "@(#)mldlUmlArquivo.h $Revision$"
/* codigo do arquivo */
#endif
```

Todos os arquivos deverão incluir um comentário com a seguinte macro RCS (no final do arquivo):

```
$RCSfile$
```

Para cada protótipo e para cada função, deverá ser incluído um comentário (imediatamente antes do protótipo e imediatamente antes do cabeçalho da função) no qual serão definidos os argumentos de entrada (indicado por um caractere I entre parênteses) e/ou de saída (indicado por um caractere O entre parênteses), os possíveis códigos de retorno e uma descrição resumida da função, como no exemplo a seguir (obviamente sem as reticências):

```
/*
 * MldlUmlErrorType
 * MldlUmlCreateRandomString (char *, unsigned, char *);
 *
 * Arguments:
 * char * - set of valid characters to create the string (I)
 * unsigned - string length (I)
 * char * - string (O)
 *
 * Returned code:
 * mldlUmlOk - Function has been executed successfully.
 * mldlUmlCreateRandomStringInvalidCharacterSet . . .
 . . .
 * Description:
 * This function creates a random . . .
 */
```

Todas as funções implementadas para a biblioteca deverão ser do tipo *mldlUmlErrorType* (definido no arquivo *mldlUmlErrors.h*), com exceção das funções para as quais for definido um outro tipo, como por exemplo das funções relacionadas com as mensagens de erro (do tipo char *).

As funções serão divididas em dois grupos: funções primárias e funções secundárias (ou auxiliares). Funções primárias corresponderão às opções do programa de linha de comando, às opções disponibilizadas através de menus na interface *Ncurses* e às opções disponibilizadas através de *links* na interface *Web*.

Para a etapa 1, os seguintes arquivos deverão ser criados no diretório "*.../Sources/C*" (trocando-se todas as ocorrências de mldl pela sigla correspondente à/ao autora/autor:

- *BSDmakefile* e *GNUmakefile*

  Estes arquivos deverá conter todas as definições necessárias para criar os programas de linha de comando e CGIs solicitadas neste roteiro. Deverão incluir também um rótulo para a limpeza do código objeto e dos executáveis (*clean*), além dos rótulos necessários para a instalação (*install*) e para a desinstalação (*deinstall*) do sistema de testes da biblioteca. O primeiro rótulo destes arquivos deverá ser *all* (compilação e linkedição de todo o código).

  Estes arquivos de dependências deverão incluir um rótulo correspondente à criação da biblioteca *UML*, ou seja, do arquivo "*libmldluml.a*".

  Além disso, estes arquivos deverão conter as definições de macros correspondendo ao compilador, ao *linkeditor*, aos *flags* de compilação, aos *flags* de *linkedição*, *às bibliotecas* e aos diretórios necessários para a instalação/desinstalação do sistema.

- *mldlUmlConfig.h*

  Este arquivo deverá conter a definição padrão de todas as constantes de configuração da biblioteca/do sistema, como por exemplo, *MLDL_UML_WEB_SERVER_URL*. Cada constante de configuração deverá ser definida através de uma macro, utilizando-se sempre o bloco de diretivas do preprocessador *ifndef*, *define* e *endif* (independente do bloco correspondente ao arquivo de cabeçalho). Como um exemplo, a constante *MLDL_UML_WEB_SERVER_URL* deverá ser definida através das seguintes linhas (representando o valor padrão para esta opção de configuração):

```
#ifndef MLDL_UML_CONFIG_H
#define MLDL_UML_CONFIG_H "@(#)mldlUmlConfig.h $Revision$"
. . .
#ifndef MLDL_UML_WEB_SERVER_URL
#define MLDL_UML_WEB_SERVER_URL
"http://www02.del.ufrj.br/~marcelo.lanza/MLDL_UML"
#endif /* #ifndef MLDL_UML_WEB_SERVER_URL */
. . .
#endif /* #ifndef MLDL_UML_CONFIG_H */
```

  Observações:

  Um valor padrão poderá ser substituído pelo valor desejado utilizando-se a opção -D do compilador na definição da macro CFLAGS no arquivo de dependências quando desejado.

  Como exemplo, -DXYZ_WEB_SERVER_URL="http://localhost/" substituiria o valor padrão pelo valor definido por esta opção.

  Alguns valores de configuração da biblioteca poderão ser redefinidos através do arquivo de configuração "mldl-uml.cfg" (ver Arquivo de Configuração).

- *mldlUmlErrors.h*

  Este arquivo deverá conter a definição do tipo enumerado *mldlUmlErrorType* (códigos de erro) e dos protótipos das funções *MldlUmlGetCliErrorMessage*, *MldlUmlGetNcursesErrorMessage* e *MldlUmlGetWebErrorMessage*.

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

char *
MldlUmlGetNcursesErrorMessage (mldlUmlErrorType, mldlUmlLanguageType);

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

- *mldlUmlErrors.c*

  Este arquivo deverá conter a definição das variáveis globais *mldlUmlCliErrorMessages*, *mldlUmlNcursesErrorMessages* e *mldlUmlWebErrorMessages* (matrizes bidimensionais de *strings* que contêm as mensagens, nos idiomas implementados, correspondentes aos códigos de erro definidos no arquivo *xyzErrors.h*), além das implementações das funções *MldlUmlGetCliErrorMessage*, *MldlUmlGetNcursesErrorMessage* e *MldlUmlGetWebErrorMessage*. As variáveis e as funções deverão ser definidas de forma que a escolha do idioma possa ser feita em tempo de execução. Estas três funções deverão ter um parâmetro do tipo *mldlUmlErrorType* e um parâmetro identificando o idioma (do tipo *mldlUmlLanguageType* - definido no arquivo *mldlUmlTypes.h*). Além disso, deverão retornar a *string* correspondente ao erro no idioma desejado.

- *mldlUmlUserInterface.h*

  Este arquivo deverá conter a definição do tipo *mldlUmlUserInterfaceMessageNumberType* e dos protótipos das funções *MldlUmlGetCliUserInterfaceMessage*, *MldlUmlGetNcursesUserInterfaceMessage* e *MldlUmlGetWebUserInterfaceMessage*.

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

char *
MldlUmlGetNcursesUserInterfaceMessage (mldlUmlUserInterfaceMessageNumberType,
mldlUmlLanguageType);

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

- *mldlUmlUserInterface.c*

  Este arquivo deverá conter a definição das variáveis globais *mldlUmlCliUserInterfaceMessages*, *mldlUmlNcursesUserInterfaceMessages* e *mldlUmlWebUserInterfaceMessages* (matrizes bidimensionais de *strings* que contêm as mensagens, nos idiomas implementados, necessárias para a construção das interfaces com o usuário - correspondendo aos números definidos no tipo *mldlUmlUserInterfaceMessageNumberType*), além da implementação das funções *MldlUmlGetCliUserInterfaceMessage*, *MldlUmlGetNcursesUserInterfaceMessage* e *MldlUmlGetWebUserInterfaceMessage*. As variáveis e as funções deverão ser definidas de forma que a escolha do idioma possa ser feita em tempo de execução. Estas três funções deverão ter um parâmetro do tipo *mldlUmlUserInterfaceMessageNumberType* e um parâmetro identificando o idioma (do tipo *mldlUmlLanguageType*). Além disso, deverão retornar a *string* correspondente à mensagem (para a construção da interface) no idioma desejado e de acordo com a interface em questão.

- *mldlUmlConst.h*

  Este arquivo deverá conter todas as constantes definindo tamanhos e conjuntos de caracteres válidos, ou seja, todas as constantes que não se enquadrem nas categorias "constantes de configuração" ou "constantes de erro".

- *mldlUmlTypes.h*

  Este arquivo deverá conter todos os tipos globais, necessários para a implementação do sistema. Este arquivo não deverá conter a definição dos tipos relacionados com os códigos de erro e com as mensagens necessárias para a construção das interfaces com o usuário.

- *mldlUmlFunctions.h*

  Este arquivo deverá conter os protótipos de todas as funções auxiliares (secundárias).

  A(O) aluna(o) poderá definir quantas funções auxiliares julgar necessário, desde que obedecendo às regras de nomenclatura já definidas neste roteiro.

- *mldlUmlFunctions.c*

  Este arquivo deverá conter a implementação de todas as funções auxiliares.

- *mldlUml.c*

  Este arquivo deverá conter o código do programa usando a biblioteca de opções de linha comando em formato longo (funções *getopt_long* e *getsubopt*) e a biblioteca *Ncurses*.

- *mldlUmlCleanAbeyances.c*

  Este programa deverá apagar os dados obsoletos que forem encontrados no diretório de dados e no diretório de *cookies* (ver Arquivos de Dados).

Para a implementação de cada função primária deverão ser criados dois arquivos utilizando o seguinte padrão:"*mldlUmlNomedaFuncaoPrimaria.h*" e "*mldlUmlNomedaFuncaoPrimaria.c*". O primeiro deverá conter apenas o protótipo da função correspondente, enquanto que o segundo deverá conter a implementação desta função.

Para a implementação de cada CGI deverá ser criado um arquivo utilizando o seguinte padrão: "*mldlUmlCgi**NomedaCGI**.c*". Como exemplo, o código fonte da CGI *mldlUmlLogin.cgi* deverá ser armazenado no arquivo "*mldlUmlCgiLogin.c*".

As CGIs que geram o código HTML referente aos formulários deverão ser criadas em arquivos utilizando o seguinte padrão: "*mldlUmlCgiShow**NomedaFuncaoPrimaria**Form.c*".

Como exemplo, o código fonte da CGI que gera o formulário com os campos necessários para trocar a senha de um usuário deverá ser armazenado no arquivo "*mldlUmlCgiShowChangePasswordForm.c*".

- **Sistema**

A organização do código fonte referente ao desenvolvimento do sistema é semelhante à organização do código fonte referente ao desenvolvimento da biblioteca UML, trocando-se a sigla UML pela sigla correspondente ao nome do sistema, conforme definido pela(o) aluna(o) - ver Nome do Sistema.

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...