Logging

```
/*
  * Nume fisier: log.h
  * Autor: Alexandru Mosoi (brtzsnr@gmail.com)
  * Ultima modificare: 07 Martie 2008
  *
  * Documentatie:
  *
  * Modulul pune la dispozitia studentilor cateva macrouri C menite
  * sa uniformizeze modul de tratare a erorilor si sa usureze depanarea
  * programelor. Mesajele vor fi scrise la stderr si vor contine, pe
  * langa mesajul utilizatorului, fisierul, functia si linia unde au
  * fost apelate macrourile.
  *
  * Acest modul este 'thread-safe'.
  *
  * LOG(nivel, format, parametri)
  *             - Afiseaza un mesaj de nivel 'nivel' daca 'nivel' este
  *             mai mic sau egal cu 'LOG_LEVEL'.
  *             - Exista 4 nivele implicite: FATAL, WARN, INFO, DEBUG.
  *             - 'LOG_LEVEL' este un macro. Daca nu este definit la
  *             includerea fisierului 'log.h' acesta ia valoarea implicita 'WARN'.
  *             - 'format' si 'parametri' au aceeasi semnificatie ca la functiile
  *             de tipul 'printf'. Intern, parametrii dupa 'nivel' sunt
  *             pasati functiei 'fprintf()'
  *             - Dupa afisarea mesajele de nivel 'FATAL' se incheie executia
  *             programului.
  *
  * LOG_ERROR(format, parametri)
  *             - analog cu 'LOG(FATAL, format, parametri)' numai ca afiseaza
  *             si ultima eroare.
  *
  * DLOG(nivel, format, parametri)
  *             - Afiseza un mesaj de depanare de nivel 'DEBUG + nivel'
  *
  * ASSERT(conditie)
  *             - Daca conditia nu e indeplinita, afiseaza un mesaj de eroare
  *             (continand conditia si ultima eroare de sistem), iar apoi
  *             intrerupe executa programului.
  *
  * CHECK_op(left, right)
  *             - verifica ca relatia binara 'op' e satisfacuta
  *      - 'op' poate fi:
  *          - EQ pentru "EQual'
  *             - NE pentru 'Not Equal'
  *
  * Pentru a modifica valoarea lui LOG_LEVEL se va compila cu optiunea:
  *             -D LOG_LEVEL=valoare (sub GCC/Linux)
  *      /D LOG_LEVEL=valoare (sub MSVS/Windows)
  *
  * Exemple:
  *
  * // LOG_LEVEL = FATAL
  * LOG(INFO, "resultat = %d", 3);    // nu afiseaza nimic
  * LOG(DEBUG, "resultat = %d", 3);   // nu afiseaza nimic
  * DLOG(666, "resultat = %d", 3);    // nu afiseaza nimic
  * ASSERT(3 == 4);                   // eroare, iese din program
  *
  * // LOG_LEVEL = DEBUG
  * LOG(INFO, "resultat = %d", 3);    // afiseaza 'resultat = 3'
```

```
 * LOG(DEBUG, "resultat = %d", 3);   // afiseaza 'resultat = 3'
 * DLOG(666, "resultat = %d", 3);    // nu afiseaza nimic
 * ASSERT(3 == 4);                   // eroare, iese din program
 *
 * // LOG_LEVEL = (DEBUG+1024)
 * LOG(INFO, "resultat = %d", 3);    // afiseaza 'resultat = 3'
 * LOG(DEBUG, "resultat = %d", 3);   // afiseaza 'resultat = 3'
 * DLOG(666, "resultat = %d", 3);    // afiseaza 'resultat = 3'
 * ASSERT(3 == 4);                   // eroare, iese din program
 *
 * CHECK_NE(fd = open("fisier", O_RDONLY), -1);
 * CHECK_EQ(function(), 0)
 *
 * // Urmatoarele linii intrerup executia programului
 * // daca 'f()' intoarece 0.
 *
 * if (f() == 0) LOG(FATAL, "f() == 0");
 * CHECK_NE(f(), 0);
 * ASSERT(f() != 0);
 *
 *
 * Exemple de mesaje:
 *
 * LOG(WARN, "Mesaj de atentionare");
 * //  W main.c (main) 9: Mesaj de atentionare
 *
 * ASSERT(0 == 1);
 * //  F main.c (main) 11: ASSERT failed: 0 == 1 (last error = Success)
 *
 * CHECK_NE(open("nu_exista", O_RDONLY), -1);
 * //  F main.c (main) 15: CHECK failed: open("nu_exista", 00) != -1 (last error = No such file
 *
 * DLOG(99, "All work and no play makes %s a dull %s!", "Jack", "boy");
 * // 99 main.c (main) 15: All work and no play makes Jack a dull boy!
 *
 * open("nu_exista", O_RDONLY);
 * LOG_ERROR("%s", "Mesaj de eroare");
 * //    main.c (main) 18: Mesaj de eroare
 * //  F main.c (main) 18: Last error: No such file or directory.
 */

#ifndef LOG_H__
#define LOG_H__

#if !defined(__GNUC__) && !defined(_WIN32)
#error Unknown operating system. Please check this file.
#endif

#include <stdio.h>
#include <stdlib.h>

#ifndef LOG_LEVEL
#define LOG_LEVEL WARN + 1024
#endif

#ifdef __GNUC__  // Linux

#include <errno.h>
#include <string.h>

#define __FUNCTION__ __func__
#define LOG_GET_ERROR_MESSAGE \
```

```
char error_buffer[512]; \
        strerror_r(errno, error_buffer, sizeof(error_buffer));

 #else  // Windows

 #include <windows.h>

 #define LOG_GET_ERROR_MESSAGE \
char error_buffer[512]; \
        FormatMessage( \
                FORMAT_MESSAGE_FROM_SYSTEM | FORMAT_MESSAGE_MAX_WIDTH_MASK, \
NULL, \
            GetLastError(), \
0, \
                error_buffer, \
sizeof(error_buffer), \
NULL);

 #endif

 enum LogType {
        NONE, FATAL, WARN, INFO, DEBUG
 };

 #define LOG(type, ...) \
do { \
if (type <= LOG_LEVEL) { \
if (type <= DEBUG) \
       (stderr, " %c", " FWID"[type]); fprintf\
else \
       (stderr, "%02d", type - DEBUG); fprintf\
                        \
       (stderr, " %s (%s:%d:%s): ", fprintf, __FILE__, __FUNCTION__, __LINE__); \
       (stderr, __VA_ARGS__); fprintf \
       (stderr, "\n"); \fprintf
      (stderr); \       fflush
                        \
/* abort() is C89 compliant. */ \
if (type == FATAL) abort(); \
} \
} while (0)

 #define LOG_ERROR(...) \
do { \
                LOG_GET_ERROR_MESSAGE \
   (NONE, __VA_ARGS__) \
   (FATAL, "LastLOG Error: %s.", error_buffer); \
} while (0)

 #define DLOG(level, ...) LOG(DEBUG + (level), __VA_ARGS__)

 #define ASSERT(condition) \
do { \
if (!(condition)) { \
                        LOG_GET_ERROR_MESSAGE \
   (NONE, "ASSERT failed: %s.", #condition); \
   (FATAL, "Last error: %s", error_buffer); \
} \
} while (0)

 #define CHECK(left, operation, right) \
do { \
```

```
if (!((left) operation (right))) { \
                        LOG_GET_ERROR_MESSAGE \
   (NONE, "CHECK failed: %s %s %s.", #left, #operation, #right); \
   (FATAL, "Last error: %s", error_buffer); \
} \
} while (0)

 #define CHECK_EQ(left, right) CHECK(left, ==, right)
 #define CHECK_NE(left, right) CHECK(left, !=, right)

 #endif
```