Experimentos VPL AED-I

[EP] Fila dinâmica

Due date: Sunday, 19 March 2023, 3:59 PM
Requested files: fila.c (Download)
Type of work: Individual work
Grade settings: Maximum grade: 1 **Hidden**
Visible: No
Run: No. **Run script**: C. **Evaluate**: Yes
Automatic grade: Yes.

Implemente as funções de uma **Fila dinâmica** de números inteiros (implementada com lista ligada):

```
FilaDinamica *criar_fila();
int enfileirar(FilaDinamica *fila, int valor);
int desenfileirar(FilaDinamica *fila, int *valor);
void liberar_fila(FilaDinamica *fila);
```

O retorno das funções enfileirar e desenfileirar indica se operação foi executada com sucesso. Por exemplo, desenfileirar pode retornar 0 para fila vazia. Se a operação foi executada com sucesso, retorna 1.

Considere a seguinte estrutura para uma Fila:

```
typedef struct FilaDinamica FilaDinamica;
struct FilaDinamica {
   LinkedNode *inicio, *fim;
};
```

Considere a seguinte estrutura para um nó de lista ligada:

```
typedef struct LinkedNode LinkedNode;
struct LinkedNode {
   int data;
   LinkedNode *next;
};
```

Importante: submeta apenas as funções da fila. Não use variáveis globais. Não inclua o main. Não use printf/fprintf/puts/gets/fgets/scanf/fscanf e headers adicionais (por exemplo, stdio.h). Neste exercício, pode usar stdlib.h.

A definição da struct LinkedNode e da struct FilaDinamica já existe no sistema de correção automática (portanto, não inclua a definição dessas structs no código submetido). É necessário incluir a seguinte linha no início do código submetido:

#include "lista.h"
#include "fila.h"

Formato do caso de teste: esse é o formato dos casos de teste que aparecem ao avaliar a atividade; não inclua impressão de dados no código, essa impressão é feita automaticamente pelo sistema de correção de acordo com o retorno da função submetida.

Entrada:

- operações sobre a pilha:

```
E 10 (enfileirar o 10)
D (desenfileirar)
S (sair)
```

Saída:

- resultado das operações

Requested files

fila.c

```c
#include <stdlib.h>
#include "lista.h"
#include "fila.h"

FilaDinamica *criar_fila() {
    // Codigo da funcao aqui
}

int enfileirar(FilaDinamica *fila, int valor) {
    // Codigo da funcao aqui
}

int desenfileirar(FilaDinamica *fila, int *valor) {
    // Codigo da funcao aqui
}

void liberar_fila(FilaDinamica *fila) {
    // Codigo da funcao aqui
}
```

Execution files

vpl_run.sh

```
if egrep -zq 'printf|puts|fprintf|gets|fgets|scanf|fscanf|stdio' $VPL_SUBFILE0 ; then
    echo "Nao eh permitido utilizar printf/puts/fprintf/gets/fgets/scanf/fscanf e stdio.h." >> vpl_compilation_error.txt
    exit 0
fi

eval gcc -fno-diagnostics-color -o vpl_execution $2 vpl_test_main.c $VPL_SUBFILE0

chmod +x vpl_execution
```

vpl_evaluate.sh

```
# [14/10/2020 - modified by Paulo Pisani] - Added code to display compilation error in comments section - this code is based on branch 3.3.8
#                 of default_evaluate.sh at https://github.com/jcrodriguez-dis/moodle-mod_vpl/tree/v3.3.8/jail/default_scripts (download 13/10/2020)
#                 from Juan Carlos Rodríguez-del-Pino, 2014)
#                 License http://www.gnu.org/copyleft/gpl.html GNU GPL v3 or later

# Set to 1 to show compilation output in comments section
export VPLX_SHOW_COMPILATION_ERROR_OUTPUT=1


#!/bin/bash
# This file is part of VPL for Moodle
# Default evaluate script for VPL
# Copyright (C) 2014 onwards Juan Carlos Rodríguez-del-Pino
# License http://www.gnu.org/copyleft/gpl.html GNU GPL v3 or later
# Author Juan Carlos Rodríguez-del-Pino <jcrodriguez@dis.ulpgc.es>

#load VPL environment vars
. common_script.sh
if [ "$SECONDS" = "" ] ; then
    export SECONDS=20
fi
let VPL_MAXTIME=$SECONDS-5;
if [ "$VPL_GRADEMIN" = "" ] ; then
    export VPL_GRADEMIN=0
    export VPL_GRADEMAX=10
fi

#exist run script?
if [ ! -s vpl_run.sh ] ; then
    echo "I'm sorry, but I haven't a default action to evaluate the type of submitted files"
else
    #avoid conflict with C++ compilation
    mv vpl_evaluate.cpp vpl_evaluate.cpp.save
    #Prepare run
    ./vpl_run.sh &>>vpl_compilation_error.txt
    cat vpl_compilation_error.txt
    if [ -f vpl_execution ] ; then
        mv vpl_execution vpl_test
        if [ -f vpl_evaluate.cases ] ; then
            mv vpl_evaluate.cases evaluate.cases
        else
            echo "Error need file 'vpl_evaluate.cases' to make an evaluation"
            exit 1
        fi
        mv vpl_evaluate.cpp.save vpl_evaluate.cpp
        check_program g++
        g++ vpl_evaluate.cpp -g -lm -lutil -o .vpl_tester
        if [ ! -f .vpl_tester ] ; then
            echo "Error compiling evaluation program"
            exit 1
        else
            cat vpl_environment.sh >> vpl_execution
            echo "./.vpl_tester" >> vpl_execution
        fi
    else
        echo "#!/bin/bash" >> vpl_execution
        echo "echo" >> vpl_execution
        echo "echo '<|--'" >> vpl_execution
        echo "echo '-$VPL_COMPILATIONFAILED'" >> vpl_execution
        if [ -f vpl_wexecution ] ; then
            echo "echo '======================'" >> vpl_execution
            echo "echo 'It seems you are trying to test a program with a graphic user interface'" >> vpl_execution
        fi

        if [ $VPLX_SHOW_COMPILATION_ERROR_OUTPUT -eq 1 ] ; then
            echo "cat vpl_compilation_error.txt" >> vpl_execution
        fi

        echo "echo '--|>'" >> vpl_execution
        echo "echo" >> vpl_execution
        echo "echo 'Grade :=>>$VPL_GRADEMIN'" >> vpl_execution
    fi
    chmod +x vpl_execution
fi

```

vpl_evaluate.cases

```
case=Test 1
grade reduction=100%
input=E 10
E 20
D
E 30
D
D
D
E 40
E 50
E 60
D
S


output="E 10
E 20
D 10
E 30
D 20
D 30
Fila vazia
E 40
E 50
E 60
D 40
"

case=Test 2
grade reduction=100%
input=D
E 11
E 22
E 66
E 88
D
D
E 40
E 50
D
D
D
D
D
S


output="Fila vazia
E 11
E 22
E 66
E 88
D 11
D 22
E 40
E 50
D 66
D 88
D 40
D 50
Fila vazia
"

case=Test 3
grade reduction=100%
input=D
D
S

output="Fila vazia
Fila vazia
"

case=Test 4
grade reduction=100%
input=E 10
E 20
E 30
E 40
E 90
E 100
S

output="E 10
E 20
E 30
E 40
E 90
E 100
"

case=Test 5
grade reduction=100%
input=E 10
E 20
E 30
E 40
E 90
E 100
D
D
D
E 500
E 501
E 502
E 503
E 504
E 506
E 507
E 508
E 509
E 510
E 511
E 512
E 513
E 514
E 515
E 516
E 517
E 518
E 519
E 520
```

```
E 521
D
D
D
D
D
D
D
E 700
E 250
D
D
D
S

output="E 10
E 20
E 30
E 40
E 90
E 100
D 10
D 20
D 30
E 500
E 501
E 502
E 503
E 504
E 506
E 507
E 508
E 509
E 510
E 511
E 512
E 513
E 514
E 515
E 516
E 517
E 518
E 519
E 520
E 521
D 40
D 90
D 100
D 500
D 501
D 502
D 503
E 700
E 250
D 504
D 506
D 507
"
```

vpl_evaluate.cpp

```
/**
 * [12/11/2020 - modified by Paulo Pisani] - Changed call to valgrind: uses /usr/bin/valgrind instead of /bin/bash + valgrind
 * [14/10/2020 - modified by Paulo Pisani] - Added code to check memory leak using valgrind - this code is based on branch 3.3.8 of vpl_evaluate.cpp
 *                  at https://github.com/jcrodriguez-dis/moodle-mod_vpl/tree/v3.3.8/jail/default_scripts (download 13/10/2020)
 *                  from Juan Carlos Rodriguez-del-Pino, 2019
 *                  License http://www.gnu.org/copyleft/gpl.html GNU GPL v3 or later
 */

const bool CHECK_LEAK_FLAG = true;
const float MEMORY_LEAK_REDUCTION = 0.2;

/**
 * VPL builtin program for submissions evaluation
 * @Copyright (C) 2019 Juan Carlos Rodriguez-del-Pino
 * @License http://www.gnu.org/copyleft/gpl.html GNU GPL v3 or later
 * @Author Juan Carlos Rodriguez-del-Pino <jcrodriguez@dis.ulpgc.es>
 */

#include <cstdlib>
#include <cstdio>
#include <climits>
#include <limits>
#include <errno.h>
#include <sys/types.h>
#include <sys/wait.h>
#include <poll.h>
#include <unistd.h>
#include <pty.h>
#include <fcntl.h>
#include <signal.h>
#include <cstring>
#include <string>
#include <iostream>
#include <sstream>
#include <vector>
#include <cmath>
#include <execinfo.h>
#include <regex.h>
#include <string>

using namespace std;

const int MAXCOMMENTS = 20;
const int MAXCOMMENTSLENGTH = 100*1024;
const int MAXCOMMENTSTITLELENGTH = 1024;
const int MAXOUTPUT = 256* 1024 ;//256Kb

/**
 * Class Tools Declaration
 */
class Tools {
public:
    static bool existFile(string name);
    static string readFile(string name);
    static vector<string> splitLines(const string &data);
    static int nextLine(const string &data);
    static string caseFormat(string text);
    static string toLower(const string &text);
    static string normalizeTag(const string &text);
    static bool parseLine(const string &text, string &name, string &data);
    static string trimRight(const string &text);
    static string trim(const string &text);
    static void fdblock(int fd, bool set);
    static bool convert2(const string& str, double &data);
    static bool convert2(const string& str, long int &data);
    static const char* getenv(const char* name, const char* defaultvalue);
    static double getenv(const char* name, double defaultvalue);
};

/**
 * Class Stop Declaration
 */
class Stop{
    static volatile bool TERMRequested;
public:
    static void setTERMRequested();
    static bool isTERMRequested();
};

/**
 * Class Timer Declaration
 */
class Timer{
    static time_t startTime;
public:
    static void start();
    static int elapsedTime();
};

/**
 * Class I18n Declaration
 */
class I18n{
public:
    void init();
    const char *get_string(const char *s);
};

/**
 * Interface OutputChecker
 */
class OutputChecker{
protected:
    string text;

public:
    OutputChecker(const string &t):text(t){}
    virtual ~OutputChecker(){};
    virtual string type(){return "";}
    virtual operator string (){return "";}
    virtual string outputExpected(){return text;}
    virtual string studentOutputExpected(){return text;}
    virtual bool match(const string&)=0;
    virtual OutputChecker* clone()=0;
};

/**
 * Class NumbersOutput Declaration
 */
class NumbersOutput:public OutputChecker{
    struct Number{
        bool isInteger;
        long int integer;
```

```
124         double cientific;
125
126         bool set(const string& str);
127         bool operator==(const Number &o)const;
128         bool operator!=(const Number &o)const;
129         operator string () const;
130     };
131
132     vector<Number> numbers;
133     bool startWithAsterisk;
134     string cleanText;
135
136     static bool isNum(char c);
137     static bool isNumStart(char c);
138     bool calcStartWithAsterisk();
139
140 public:
141     NumbersOutput(const string &text);//:OutputChecker(text);
142     string studentOutputExpected();
143     bool operator==(const NumbersOutput& o)const;
144     bool match(const string& output);
145     OutputChecker* clone();
146     static bool typeMatch(const string& text);
147     string type();
148     operator string () const;
149 };
150
151 /**
152  * Class TextOutput Declaration
153  */
154 class TextOutput:public OutputChecker{
155     vector<string> tokens;
156     bool isAlpha(char c);
157
158 public:
159     TextOutput(const string &text);//:OutputChecker(text);
160     bool operator==(const TextOutput& o);
161     bool match(const string& output);
162     OutputChecker* clone();
163     static bool typeMatch(const string& text);
164     string type();
165 };
166
167 /**
168  * Class ExactTextOutput Declaration
169  */
170 class ExactTextOutput:public OutputChecker{
171     string cleanText;
172     bool startWithAsterix;
173     bool isAlpha(char c);
174
175 public:
176     ExactTextOutput(const string &text);//:OutputChecker(text);
177     string studentOutputExpected();
178     bool operator==(const ExactTextOutput& o);
179     bool match(const string& output);
180     OutputChecker* clone();
181     static bool typeMatch(const string& text);
182     string type();
183 };
184
185 /**
186  * Class RegularExpressionOutput Declaration
187  * Regular Expressions implemented by:
188  * Daniel José Ojeda Loisel
189  * Juan David Vega Rodríguez
190  * Miguel Ángel Viera González
191  */
192 class RegularExpressionOutput:public OutputChecker {
193     string errorCase;
194     string cleanText;
195     regex_t expression;
196     bool flagI;
197     bool flagM;
198     int reti;
199
200 public:
201     RegularExpressionOutput (const string &text, const string &actualCaseDescription);
202
203     bool match (const string& output);
204         // Regular Expression compilation (with flags in mind) and comparison with the input and output evaluation
205
206     string studentOutputExpected();
207         // Returns the expression without flags nor '/'
208
209     OutputChecker* clone();
210
211     static bool typeMatch(const string& text);
212         // Tests if it's a regular expression. A regular expressions should be between /../
213
214     string type();
215 };
216 /**
217  * Class Case Declaration
218  * Case represents cases
219  */
220 class Case {
221     string input;
222     vector< string > output;
223     string caseDescription;
224     float gradeReduction;
225     string failMessage;
226     string programToRun;
227     string programArgs;
228     int expectedExitCode; // Default value std::numeric_limits<int>::min()
229     string variation;
230     bool checkLeak;
231 public:
232     Case();
233     void reset();
234     void addInput(string );
235     string getInput();
236     void addOutput(string );
237     const vector< string > & getOutput();
238     void setFailMessage(const string &);
239     string getFailMessage();
240     void setCaseDescription(const string &);
241     string getCaseDescription();
242     void setGradeReduction(float);
243     float getGradeReduction();
244     void setExpectedExitCode(int);
245     int getExpectedExitCode();
246     void setProgramToRun(const string &);
```

```
247       string getProgramToRun();
248       void setProgramArgs(const string &);
249       string getProgramArgs();
250       void setVariation(const string &);
251       string getVariation();
252       void setCheckLeak(bool cl);
253       bool getCheckLeak();
254   };
255
256   /**
257    * Class TestCase Declaration
258    * TestCase represents cases to tested
259    */
260   class TestCase {
261       const char *command;
262       const char **argv;
263       static const char **envv;
264       int id;
265       bool correctOutput;
266       bool checkLeak;
267       bool hasLeak;
268       bool outputTooLarge;
269       bool programTimeout;
270       bool executionError;
271       bool correctExitCode;
272       char executionErrorReason[1000];
273       int sizeReaded;
274       string input;
275       vector< OutputChecker* > output;
276       string caseDescription;
277       float gradeReduction;
278       float gradeReductionApplied;
279       string failMessage;
280       string programToRun;
281       string programArgs;
282       string variantion;
283       int expectedExitCode; // Default value std::numeric_limits<int>::min()
284       int exitCode; // Default value std::numeric_limits<int>::min()
285       string programOutputBefore, programOutputAfter, programInput;
286
287       void cutOutputTooLarge(string &output);
288       void readWrite(int fdread, int fdwrite);
289       void addOutput(const string &o, const string &actualCaseDescription);
290       void runTestExec(time_t timeout, bool check_leak_only);
291       void removeZerosFromVector(char buf[], int len);
292   public:
293       static void setEnvironment(const char **environment);
294       void setDefaultCommand();
295       TestCase(const TestCase &o);
296       TestCase& operator=(const TestCase &o);
297       ~TestCase();
298       TestCase(int id, const string &input, const vector<string> &output,
299               const string &caseDescription, const float gradeReduction,
300               string failMessage, string programToRun, string programArgs, int expectedExitCode, bool cl);
301       bool isCorrectResult();
302       bool hasMemoryLeak();
303       bool isExitCodeTested();
304       float getGradeReduction();
305       void setGradeReductionApplied(float r);
306       float getGradeReductionApplied();
307       string getCaseDescription();
308       string getCommentTitle(bool withGradeReduction/*=false*/); // Suui
309       string getComment();
310       void splitArgs(string programArgs, bool checkLeakOnly);
311       void runTest(time_t timeout);
312       bool match(string data);
313       bool matchMemoryLeak(string data);
314   };
315
316   /**
317    * Class Evaluation Declaration
318    */
319   class Evaluation {
320       int maxtime;
321       float grademin, grademax;
322       string variation;
323       bool noGrade;
324       float grade;
325       int nerrors, nruns;
326       vector<TestCase> testCases;
327       char comments[MAXCOMMENTS + 1][MAXCOMMENTSLENGTH + 1];
328       char titles[MAXCOMMENTS + 1][MAXCOMMENTSTITLELENGTH + 1];
329       char titlesGR[MAXCOMMENTS + 1][MAXCOMMENTSTITLELENGTH + 1];
330       volatile int ncomments;
331       volatile bool stopping;
332       static Evaluation *singlenton;
333       Evaluation();
334
335   public:
336       static Evaluation* getSinglenton();
337       static void deleteSinglenton();
338       void addTestCase(Case &);
339       void removeLastNL(string &s);
340       bool cutToEndTag(string &value, const string &endTag);
341       void loadTestCases(string fname);
342       bool loadParams();
343       void addFatalError(const char *m);
344       void runTests();
345       void outputEvaluation();
346   };
347
348   ////////////////////////////////////////////////////////////////////////////////////////////
349   ////////////////////////////////////////////////////////////////////////////////////////////
350   ////////////////////////////////////// END OF DECLARATIONS /////////////////////////////////
351   ////////////////////////////////////////////////////////////////////////////////////////////
352   ////////////////////////////////////////////////////////////////////////////////////////////
353
354   ////////////////////////////////////////////////////////////////////////////////////////////
355   ////////////////////////////////////////////////////////////////////////////////////////////
356   ////////////////////////////////////// BEGINNING OF DEFINITIONS ////////////////////////////
357   ////////////////////////////////////////////////////////////////////////////////////////////
358   ////////////////////////////////////////////////////////////////////////////////////////////
359
360   volatile bool Stop::TERMRequested = false;
361   time_t Timer::startTime;
362   const char **TestCase::envv=NULL;
363   Evaluation* Evaluation::singlenton = NULL;
364
365   /**
366    * Class Tools Definitions
367    */
368
369   bool Tools::existFile(string name) {
```

```
    FILE *f = fopen(name.c_str(), "r");
    if (f != NULL) {
        fclose(f);
        return true;
    }
    return false;
}

string Tools::readFile(string name) {
    char buf[1000];
    string res;
    FILE *f = fopen(name.c_str(), "r");
    if (f != NULL)
        while (fgets(buf, 1000, f) != NULL)
            res += buf;
    return res;
}

vector<string> Tools::splitLines(const string &data) {
    vector<string> lines;
    int len, l = data.size();
    int startLine = 0;
    char pc = 0, c;
    for (int i = 0; i < l; i++) {
        c = data[i];
        if (c == '\n') {
            len = i - startLine;
            if (pc == '\r')
                len--;
            lines.push_back(data.substr(startLine, len));
            startLine = i + 1;
        }
        pc = c;
    }
    if (startLine < l) {
        len = l - startLine;
        if (pc == '\r')
            len--;
        lines.push_back(data.substr(startLine, len));
    }
    return lines;
}

int Tools::nextLine(const string &data) {
    int l = data.size();
    for (int i = 0; i < l; i++) {
        if (data[i] == '\n')
            return i + 1;
    }
    return l;
}

string Tools::caseFormat(string text) {
    vector<string> lines = Tools::splitLines(text);
    string res;
    int nlines = lines.size();
    for (int i = 0; i < nlines; i++)
        res += ">" + lines[i] + '\n';
    return res;
}

bool Tools::parseLine(const string &text, string &name, string &data) {
    size_t poseq;
    if ((poseq = text.find('=')) != string::npos) {
        name = normalizeTag(text.substr(0, poseq + 1));
        data = text.substr(poseq + 1);
        return true;
    }
    name = "";
    data = text;
    return false;
}

string Tools::toLower(const string &text) {
    string res = text;
    int len = res.size();
    for (int i = 0; i < len; i++)
        res[i] = tolower(res[i]);
    return res;
}

string Tools::normalizeTag(const string &text) {
    string res;
    int len = text.size();
    for (int i = 0; i < len; i++) {
        char c = text[i];
        if (isalpha(c) || c == '=')
            res += tolower(c);
    }
    return res;
}

string Tools::trimRight(const string &text) {
    int len = text.size();
    int end = -1;
    for (int i = len - 1; i >= 0; i--) {
        if (!isspace(text[i])) {
            end = i;
            break;
        }
    }
    return text.substr(0, end + 1);
}

string Tools::trim(const string &text) {
    int len = text.size();
    int begin = len;
    int end = -1;
    for (int i = 0; i < len; i++) {
        char c = text[i];
        if (!isspace(c)) {
            begin = i;
            break;
        }
    }
    for (int i = len - 1; i >= 0; i--) {
        char c = text[i];
        if (!isspace(c)) {
            end = i;
            break;
        }
    }
    if (begin <= end)
```

```
493             return text.substr(begin, (end - begin) + 1);
494         return "";
495     }
496
497     void Tools::fdblock(int fd, bool set) {
498         int flags;
499         if ((flags = fcntl(fd, F_GETFL, 0)) < 0) {
500             return;
501         }
502         if (set && (flags | O_NONBLOCK) == flags)
503             flags ^= O_NONBLOCK;
504         else
505             flags |= O_NONBLOCK;
506         fcntl(fd, F_SETFL, flags);
507     }
508
509     bool Tools::convert2(const string& str, double &data){
510         if ( str == "." ){
511             return false;
512         }
513         stringstream conv(str);
514         conv >> data;
515         return conv.eof();
516     }
517
518     bool Tools::convert2(const string& str, long int &data){
519         stringstream conv(str);
520         conv >> data;
521         return conv.eof();
522     }
523     const char* Tools::getenv(const char* name, const char* defaultvalue) {
524         const char* value = ::getenv(name);
525         if ( value == NULL ) {
526             value = defaultvalue;
527             printf("Warning: using default value '%s' for '%s'\n", defaultvalue, name);
528         }
529         return value; // Fixes bug found by Peter Svec
530     }
531
532     double Tools::getenv(const char* name, double defaultvalue) {
533         const char* svalue = ::getenv(name);
534         double value = defaultvalue;
535         if ( svalue != NULL ) {
536             Tools::convert2(svalue, value);
537         } else {
538             printf("Warning: using default value '%lf' for '%s'\n", defaultvalue, name);
539         }
540         return value;
541     }
542
543
544     /**
545      * Class Stop Definitions
546      */
547
548     void Stop::setTERMRequested() {
549         TERMRequested = true;
550     }
551
552     bool Stop::isTERMRequested() {
553         return TERMRequested;
554     }
555
556     /**
557      * Class Timer Definitions
558      */
559
560     void Timer::start() {
561         startTime = time(NULL);
562     }
563
564     int Timer::elapsedTime() {
565         return time(NULL) - startTime;
566     }
567
568     /**
569      * Class Stop Definitions
570      */
571
572     void I18n::init(){
573
574     }
575
576     const char *I18n::get_string(const char *s){
577         return s;
578     }
579
580     /**
581      * Class NumbersOutput Definitions
582      */
583
584     // Struct Number
585     bool NumbersOutput::Number::set(const string& str){
586         isInteger=Tools::convert2(str, integer);
587         if(!isInteger){
588             return Tools::convert2(str, cientific);
589         }
590         return true;
591     }
592
593     bool NumbersOutput::Number::operator==(const Number &o)const{
594         if(isInteger)
595             return o.isInteger && integer == o.integer;
596         if(o.isInteger)
597             return cientific != 0?fabs((cientific - o.integer) / cientific) < 0.0001 : o.integer == 0;
598         else
599             return cientific != 0?fabs((cientific - o.cientific) / cientific) < 0.0001 : fabs(o.cientific) < 0.0001;
600     }
601
602     bool NumbersOutput::Number::operator!=(const Number &o)const{
603         return !((*this)==o);
604     }
605
606     NumbersOutput::Number::operator string() const{
607         char buf[100];
608         if(isInteger) {
609             sprintf(buf, "%ld", integer);
610         } else {
611             sprintf(buf, "%10.5lf", cientific);
612         }
613         return buf;
614     }
615
```

```
616
617 bool NumbersOutput::isNum(char c){
618     if(isdigit(c)) return true;
619     return c=='+' || c=='-' || c=='.' || c=='e' || c=='E';
620 }
621
622 bool NumbersOutput::isNumStart(char c){
623     if(isdigit(c)) return true;
624     return c=='+' || c=='-' || c=='.';
625 }
626
627 bool NumbersOutput::calcStartWithAsterisk(){
628     int l=text.size();
629     for(int i=0; i<l; i++){
630         char c=text[i];
631         if(isspace(c)) continue;
632         if(c=='*'){
633             cleanText = text.substr(i+1,text.size()-(i+1));
634             return true;
635         }else{
636             cleanText = text.substr(i,text.size()-i);
637             return false;
638         }
639     }
640     return false;
641 }
642
643 NumbersOutput::NumbersOutput(const string &text):OutputChecker(text){
644     int l=text.size();
645     string str;
646     Number number;
647     for(int i=0; i<l; i++){
648         char c=text[i];
649         if((isNum(c) && str.size()>0) || (isNumStart(c) && str.size()==0)){
650             str+=c;
651         }else if(str.size()>0){
652             if(isNumStart(str[0]) && number.set(str)) numbers.push_back(number);
653             str="";
654         }
655     }
656     if(str.size()>0){
657         if(isNumStart(str[0]) && number.set(str)) numbers.push_back(number);
658     }
659     startWithAsterisk=calcStartWithAsterisk();
660 }
661
662 string NumbersOutput::studentOutputExpected(){
663     return cleanText;
664 }
665
666 bool NumbersOutput::operator==(const NumbersOutput& o)const{
667     size_t l=numbers.size();
668     if( o.numbers.size() < l ) return false;
669     int offset = 0;
670     if(startWithAsterisk)
671         offset = o.numbers.size()-l;
672     for(size_t i = 0; i < l; i++)
673         if(numbers[i] != o.numbers[offset+i])
674             return false;
675     return true;
676 }
677
678 bool NumbersOutput::match(const string& output){
679     NumbersOutput temp(output);
680     return operator==(temp);
681 }
682
683 OutputChecker* NumbersOutput::clone(){
684     return new NumbersOutput(outputExpected());
685 }
686
687 bool NumbersOutput::typeMatch(const string& text){
688     int l=text.size();
689     string str;
690     Number number;
691     for(int i=0; i<l; i++){
692         char c=text[i];
693         // Skip spaces/CR/LF... and *
694         if(!isspace(c) && c!='*') {
695             str += c;
696         }else if(str.size()>0) {
697             if (!isNumStart(str[0])||
698                 !number.set(str)) return false;
699             str="";
700         }
701     }
702     if(str.size()>0){
703         if(!isNumStart(str[0])||!number.set(str)) return false;
704     }
705     return true;
706 }
707
708 string NumbersOutput::type(){
709     return "numbers";
710 }
711
712 NumbersOutput::operator string () const{
713     string ret="[";
714     int l=numbers.size();
715     for(int i=0; i<l; i++){
716         ret += i > 0 ? ", " : "";
717         ret += numbers[i];
718     }
719     ret += "]";
720     return ret;
721 }
722
723 /**
724  * Class TextOutput Definitions
725  */
726
727 bool TextOutput::isAlpha(char c){
728     if ( isalnum(c) ) return true;
729     return c < 0;
730 }
731
732 TextOutput::TextOutput(const string &text):OutputChecker(text){
733     size_t l = text.size();
734     string token;
735     for(size_t i = 0; i < l; i++){
736         char c = text[i];
737         if( isAlpha(c) ){
738             token += c;
```

```
        }else if(token.size() > 0){
            tokens.push_back(Tools::toLower(token));
            token="";
        }
    }
    if(token.size()>0){
        tokens.push_back(Tools::toLower(token));
    }
}

bool TextOutput::operator==(const TextOutput& o) {
    size_t l = tokens.size();
    if (o.tokens.size() < l) return false;
    int offset = o.tokens.size() - l;
    for (size_t i = 0; i < l; i++)
        if (tokens[i] != o.tokens[ offset + i ])
            return false;
    return true;
}

bool TextOutput::match(const string& output) {
    TextOutput temp(output);
    return operator== (temp);
}

OutputChecker* TextOutput::clone() {
    return new TextOutput(outputExpected());
}

bool TextOutput::typeMatch(const string& text) {
    return true;
}

string TextOutput::type(){
    return "text";
}

/**
 * Class ExactTextOutput Definitions
 */

bool ExactTextOutput::isAlpha(char c){
    if(isalnum(c)) return true;
    return c < 0;
}

ExactTextOutput::ExactTextOutput(const string &text):OutputChecker(text){
    string clean = Tools::trim(text);
    if(clean.size() > 2 && clean[0] == '*') {
        startWithAsterix = true;
        cleanText = clean.substr(2, clean.size() - 3);
    }else{
        startWithAsterix =false;
        cleanText=clean.substr(1,clean.size()-2);
    }
}

string ExactTextOutput::studentOutputExpected(){
    return cleanText;
}

bool ExactTextOutput::operator==(const ExactTextOutput& o){
    return match(o.text);
}

bool ExactTextOutput::match(const string& output){
    if(cleanText.size()==0 && output.size()==0) return true;
    string clean;
    // Clean output if text last char is alpha
    if(cleanText.size()>0 && isAlpha(cleanText[cleanText.size()-1])){
        clean=Tools::trimRight(output);
    }else{
        clean=output;
    }
    if(startWithAsterix){
        size_t start=clean.size()-cleanText.size();
        return cleanText.size()<=clean.size() &&
                cleanText == clean.substr(start,cleanText.size());
    }
    else
        return cleanText==clean;
}

OutputChecker* ExactTextOutput::clone(){
    return new ExactTextOutput(outputExpected());
}

bool ExactTextOutput::typeMatch(const string& text){
    string clean=Tools::trim(text);
    return (clean.size()>1 && clean[0]=='"' && clean[clean.size()-1]=='"')
            ||(clean.size()>3 && clean[0]=='*' && clean[1]=='"' && clean[clean.size()-1]=='"');
}

string ExactTextOutput::type(){
    return "exact text";
}

/**
 * Class RegularExpressionOutput Definitions
 */

RegularExpressionOutput::RegularExpressionOutput(const string &text, const string &actualCaseDescription):OutputChecker(text) {

    errorCase = actualCaseDescription;
    size_t pos = 1;
    flagI = false;
    flagM = false;
    string clean = Tools::trim(text);

    while (clean[pos] != '/' && pos < clean.size()) {
        pos++;
    }
    cleanText = clean.substr(1,pos-1);
    if (pos + 1 != clean.size()) {
        pos = pos + 1;

        // Flag search
        while (pos < clean.size()) {

            switch (clean[pos]) {
                case 'i':
                    flagI=true;
                    break;
```

```
                case 'm':
                    flagM=true;
                    break;
                default:
                    Evaluation* p_ErrorTest = Evaluation::getSinglenton();
                    char wrongFlag = clean[pos];
                    string flagCatch;
                    stringstream ss;
                    ss << wrongFlag;
                    ss >> flagCatch;
                    string errorType = string("Flag Error in case ")+ string(errorCase)+ string (", found a ") + string(flagCatch) + string (" used as a
                    const char* flagError = errorType.c_str();
                    p_ErrorTest->addFatalError(flagError);
                    p_ErrorTest->outputEvaluation();
                    abort();
            }
            pos++;
        }
    }
}

// Regular Expression compilation (with flags in mind) and comparison with the input and output evaluation
bool RegularExpressionOutput::match (const string& output) {

    reti=-1;
    const char * in = cleanText.c_str();
    // Use POSIX-C regrex.h
    // Flag compilation
    if (flagI || flagM) {
        if (flagM && flagI) {
            reti = regcomp(&expression, in, REG_EXTENDED | REG_NEWLINE | REG_ICASE);
        } else if (flagM) {
            reti = regcomp(&expression, in, REG_EXTENDED | REG_NEWLINE);
        } else {
            reti = regcomp(&expression, in, REG_EXTENDED | REG_ICASE);
        }

    // No flag compilation
    } else {
        reti = regcomp(&expression, in, REG_EXTENDED);
    }

    if (reti == 0) { // Compilation was successful

        const char * out = output.c_str();
        reti = regexec(&expression, out, 0, NULL, 0);

        if (reti == 0) { // Match
            return true;
        } else if (reti == REG_NOMATCH){ // No match
            return false;

        } else { // Memory Error
            Evaluation* p_ErrorTest = Evaluation::getSinglenton();
            string errorType = string("Out of memory error, during matching case ") + string(errorCase);
            const char* flagError = errorType.c_str();
            p_ErrorTest->addFatalError(flagError);
            p_ErrorTest->outputEvaluation();
            abort();
        }

    } else { // Compilation error
        size_t length = regerror(reti, &expression, NULL, 0);
        char* bff = new char[length + 1];
        (void) regerror(reti, &expression, bff, length);
        Evaluation* p_ErrorTest = Evaluation::getSinglenton();
        string errorType = string("Regular Expression compilation error")+string (" in case: ")+ string(errorCase) +string (".\n")+ string(bff);
        const char* flagError = errorType.c_str();
        p_ErrorTest->addFatalError(flagError);
        p_ErrorTest->outputEvaluation();
        abort();
        return false;
    }
}

// Returns the expression without flags nor '/'
string RegularExpressionOutput::studentOutputExpected() {return cleanText;}

OutputChecker* RegularExpressionOutput::clone() {
    return new RegularExpressionOutput(outputExpected(), errorCase);
}

// Tests if it's a regular expression. A regular expressions should be between /../
bool RegularExpressionOutput::typeMatch(const string& text) {
    string clean=Tools::trim(text);
    if (clean.size() > 2 && clean[0] == '/') {
        for (size_t i = 1; i < clean.size(); i++) {
            if (clean[i] == '/') {
                return true;
            }
        }
    }
    return false;
}

string RegularExpressionOutput::type() {
    return "regular expression";
}
/**
 * Class Case Definitions
 * Case represents cases
 */
Case::Case() {
    reset();
}

void Case::reset() {
    input = "";
    output.clear();
    caseDescription = "";
    gradeReduction = std::numeric_limits<float>::min();
    failMessage = "";
    programToRun = "";
    programArgs = "";
    variation = "";
    expectedExitCode = std::numeric_limits<int>::min();
}

void Case::addInput(string s) {
    input += s;
}

string Case::getInput() {
```

```
    return input;
}

void Case::addOutput(string o) {
    output.push_back(o);
}

const vector< string > & Case::getOutput() {
    return output;
}

void Case::setFailMessage(const string &s) {
    failMessage = s;
}

string Case::getFailMessage() {
    return failMessage;
}
void Case::setCaseDescription(const string &s) {
    caseDescription = s;
}

string Case::getCaseDescription() {
    return caseDescription;
}
void Case::setGradeReduction(float g) {
    gradeReduction = g;
}

float Case::getGradeReduction() {
    return gradeReduction;
}

void Case::setExpectedExitCode(int e) {
    expectedExitCode = e;
}

int Case::getExpectedExitCode() {
    return expectedExitCode;
}
void Case::setProgramToRun(const string &s) {
    programToRun = s;
}

string Case::getProgramToRun() {
    return programToRun;
}

void Case::setProgramArgs(const string &s) {
    programArgs = s;
}

string Case::getProgramArgs() {
    return programArgs;
}

void Case::setVariation(const string &s) {
    variation = Tools::toLower(Tools::trim(s));
}

string Case::getVariation() {
    return variation;
}

void Case::setCheckLeak(bool cl) {
    this->checkLeak = cl;
}

bool Case::getCheckLeak() {
    return checkLeak;
}

/**
 * Class TestCase Definitions
 * TestCase represents cases of test
 */

void TestCase::cutOutputTooLarge(string &output) {
    if (output.size() > MAXOUTPUT) {
        outputTooLarge = true;
        output.erase(0, output.size() - MAXOUTPUT);
    }
}

void TestCase::removeZerosFromVector(char buf[], int len) {
    for (int i = 0; i < len-1; i++)
        if (buf[i] == 0)
            buf[i] = 32; // Troca por espacos
}

void TestCase::readWrite(int fdread, int fdwrite) {
    const int MAX = 1024* 10 ;
    // Buffer size to read
    const int POLLREAD = POLLIN | POLLPRI;
    // Poll to read from program
    struct pollfd devices[2];
    devices[0].fd = fdread;
    devices[1].fd = fdwrite;
    char buf[MAX];
    devices[0].events = POLLREAD;
    devices[1].events = POLLOUT;
    int res = poll(devices, programInput.size()>0?2:1, 0);
    if (res == -1) // Error
        return;
    if (res == 0) // Nothing to do
        return;
    if (devices[0].revents & POLLREAD) { // Read program output
        int readed = read(fdread, buf, MAX);
        if (readed > 0) {
            sizeReaded += readed;
            if (programInput.size() > 1) {
                programOutputBefore += string(buf, readed);
                cutOutputTooLarge(programOutputBefore);
            } else {
                programOutputAfter += string(buf, readed);
                cutOutputTooLarge(programOutputAfter);
            }
        }
    }
    if (programInput.size() > 0 && devices[1].revents & POLLOUT) { // Write to program
        int written = write(fdwrite, programInput.c_str(), Tools::nextLine(
                programInput));
        if (written > 0) {
```

```
            programInput.erase(0, written);
        }
        if(programInput.size()==0){
            close(fdwrite);
        }
    }
}

void TestCase::addOutput(const string &o, const string &actualCaseDescription){
// actualCaseDescripction, used to get current test name for Output recognition
    if(ExactTextOutput::typeMatch(o))
        this->output.push_back(new ExactTextOutput(o));
    else if (RegularExpressionOutput::typeMatch(o))
        this->output.push_back(new RegularExpressionOutput(o, actualCaseDescription));
    else if(NumbersOutput::typeMatch(o))
        this->output.push_back(new NumbersOutput(o));
    else
        this->output.push_back(new TextOutput(o));
}

void TestCase::setEnvironment(const char **environment) {
    envv = environment;
}

void TestCase::setDefaultCommand() {
    command = "./vpl_test";
    argv = new const char*[2];
    argv[0] = command;
    argv[1] = NULL;
}

TestCase::TestCase(const TestCase &o) {
    id=o.id;
    correctOutput=o.correctOutput;
    hasLeak=o.hasLeak;
    checkLeak=o.checkLeak;
    correctExitCode = o.correctExitCode;
    outputTooLarge=o.outputTooLarge;
    programTimeout=o.programTimeout;
    executionError=o.executionError;
    strcpy(executionErrorReason,o.executionErrorReason);
    sizeReaded=o.sizeReaded;
    input=o.input;
    caseDescription=o.caseDescription;
    gradeReduction=o.gradeReduction;
    expectedExitCode = o.expectedExitCode;
    exitCode = o.exitCode;
    failMessage=o.failMessage;
    programToRun=o.programToRun;
    programArgs=o.programArgs;
    gradeReductionApplied=o.gradeReductionApplied;
    programOutputBefore=o.programOutputBefore;
    programOutputAfter=o.programOutputAfter;
    programInput=o.programInput;
    for(size_t i = 0; i < o.output.size(); i++){
        output.push_back(o.output[i]->clone());
    }
    setDefaultCommand();
}

TestCase& TestCase::operator=(const TestCase &o) {
    id=o.id;
    correctOutput=o.correctOutput;
    hasLeak=o.hasLeak;
    checkLeak=o.checkLeak;
    correctExitCode = o.correctExitCode;
    outputTooLarge=o.outputTooLarge;
    programTimeout=o.programTimeout;
    executionError=o.executionError;
    strcpy(executionErrorReason,o.executionErrorReason);
    sizeReaded=o.sizeReaded;
    input=o.input;
    caseDescription=o.caseDescription;
    gradeReduction=o.gradeReduction;
    failMessage=o.failMessage;
    programToRun=o.programToRun;
    programArgs=o.programArgs;
    expectedExitCode = o.expectedExitCode;
    exitCode = o.exitCode;
    gradeReductionApplied=o.gradeReductionApplied;
    programOutputBefore=o.programOutputBefore;
    programOutputAfter=o.programOutputAfter;
    programInput=o.programInput;
    for(size_t i=0; i<output.size(); i++)
        delete output[i];
    output.clear();
    for(size_t i=0; i<o.output.size(); i++){
        output.push_back(o.output[i]->clone());
    }
    return *this;
}

TestCase::~TestCase() {
    for(size_t i = 0; i < output.size(); i++)
        delete output[i];
}

TestCase::TestCase(int id, const string &input, const vector<string> &output,
        const string &caseDescription, const float gradeReduction,
        string failMessage, string programToRun, string programArgs, int expectedExitCode, bool cl) {
    this->id = id;
    this->input = input;
    for(size_t i = 0; i < output.size(); i++){
        addOutput(output[i], caseDescription);
    }
    this->caseDescription = caseDescription;
    this->gradeReduction = gradeReduction;
    this->expectedExitCode = expectedExitCode;
    this->programToRun = programToRun;
    this->programArgs = programArgs;
    this->failMessage = failMessage;
    this->checkLeak = cl;
    exitCode = std::numeric_limits<int>::min();
    outputTooLarge = false;
    programTimeout = false;
    executionError = false;
    correctOutput = false;
    hasLeak = false;
    correctExitCode = false;
    sizeReaded = 0;
    gradeReductionApplied =0;
    strcpy(executionErrorReason, "");
    setDefaultCommand();
```

```
}

bool TestCase::isCorrectResult() {
    bool correct = correctOutput &&
                   ! programTimeout &&
                   ! outputTooLarge &&
                   ! executionError;
    return correct || (isExitCodeTested() && correctExitCode);
}

bool TestCase::hasMemoryLeak() {
    return (checkLeak && hasLeak);
}

bool TestCase::isExitCodeTested() {
    return expectedExitCode != std::numeric_limits<int>::min();
}

float TestCase::getGradeReduction() {
    return gradeReduction;
}

void TestCase::setGradeReductionApplied(float r) {
    gradeReductionApplied=r;
}

float TestCase::getGradeReductionApplied() {
    return gradeReductionApplied;
}

string TestCase::getCaseDescription(){
    return caseDescription;
}

string TestCase::getCommentTitle(bool withGradeReduction=false) {
    char buf[100];
    string ret;
    sprintf(buf, "Test %d", id);
    ret = buf;
    if (caseDescription.size() > 0) {
        ret += ": " + caseDescription;
    }
    if(withGradeReduction && getGradeReductionApplied()>0){
        sprintf(buf," (%.3f)", -getGradeReductionApplied());
        ret += buf;
    }
    ret += '\n';
    return ret;
}

string TestCase::getComment() {
    string ret;
    if (checkLeak && hasLeak) {
        ret += "[Memory leak check failed]\n";
    }
    if (isCorrectResult() && !(checkLeak && hasLeak)) {
        return ret;
    }
    char buf[100];
    if(output.size()==0){
        ret += "Configuration error in the test case: the output is not defined";
    }
    if (programTimeout) {
        ret += "Program timeout\n";
    }
    if (outputTooLarge) {
        sprintf(buf, "Program output too large (%dKb)\n", sizeReaded / 1024);
        ret += buf;
    }
    if (executionError) {
        ret += executionErrorReason + string("\n");
    }
    if (isExitCodeTested() && ! correctExitCode) {
        char buf[250];
        sprintf(buf, "Incorrect exit code. Expected %d, found %d\n", expectedExitCode, exitCode);
        ret += buf;
    }
    if (!correctOutput || (checkLeak && hasLeak)) {
        if (failMessage.size()) {
            ret += correctOutput ? "[Correct program output]\n" : "[Incorrect program output]\n";
            ret += failMessage + "\n";
        } else {
            ret += correctOutput ? "[Correct program output]\n" : "[Incorrect program output]\n";
            ret += " --- Input ---\n";
            ret += Tools::caseFormat(input);
            ret += "\n --- Program output ---\n";
            ret += Tools::caseFormat(programOutputBefore + programOutputAfter);
            if(output.size()>0){
                ret += "\n --- Expected output ("+output[0]->type()+")---\n";
                ret += Tools::caseFormat(output[0]->studentOutputExpected());
            }
        }
    }
    return ret;
}

void TestCase::splitArgs(string programArgs, bool checkLeakOnly) {
    const char* original_command = (checkLeakOnly ? argv[1] : command);
    int offsetParams = (checkLeakOnly ? 1 : 0);

    int l = programArgs.size();
    int nargs = 1 + offsetParams;
    char *buf = new char[programArgs.size() + 1];
    strcpy(buf, programArgs.c_str());

    delete argv;
    argv = (const char **) new char*[programArgs.size() + 1 + offsetParams];

    if (checkLeakOnly) {
        argv[0] = "/usr/bin/valgrind";
        argv[1] = original_command;
    } else {
        argv[0] = original_command;
    }
    bool inArg = false;
    char separator = ' ';
    for(int i=0; i < l; i++) { // TODO improve
        if ( ! inArg ) {
            if ( buf[i] == ' ' ) {
                buf[i] = '\0';
                continue;
            } else if ( buf[i] == '\'' ) {
                argv[nargs++] = buf + i + 1;
```

```
                separator = '\'';
            } else if ( buf[i] == '"' ) {
                argv[nargs++] = buf + i + 1;
                separator = '"';
            } else if ( buf[i] != '\0') {
                argv[nargs++] = buf + i;
                separator = ' ';
            }
            inArg = true;
        } else {
            if ( buf[i] == separator  ) {
                buf[i] = '\0';
                separator = ' ';
                inArg = false;
            }
        }
    }
    argv[nargs] = NULL;
}


void TestCase::runTest(time_t timeout) {
    if (checkLeak) runTestExec(timeout, true);
    runTestExec(timeout, false);
}

void TestCase::runTestExec(time_t timeout, bool checkLeakOnly) {//timeout in seconds
    time_t start = time(NULL);
    int pp1[2]; // Send data
    int pp2[2]; // Receive data
    if (pipe(pp1) == -1 || pipe(pp2) == -1) {
        executionError = true;
        sprintf(executionErrorReason, "Internal error: pipe error (%s)",
                strerror(errno));
        return;
    }
    command = "./vpl_test";
    if ( programToRun > "" && programToRun.size() < 512) {
        command = programToRun.c_str();
    }
    if ( ! Tools::existFile(command) ){
        executionError = true;
        sprintf(executionErrorReason, "Execution file not found '%s'", command);
        return;
    }

    // Updates command and argv if checkLeakOnly
    if (checkLeakOnly) {
        delete argv;
        argv = new const char*[3];
        argv[0] = "/usr/bin/valgrind";
        argv[1] = command; // First get the original command
        argv[2] = NULL;
        command = "/usr/bin/valgrind"; // Update command
    } else {
        delete argv;
        argv = new const char*[2];
        argv[0] = command;
        argv[1] = NULL;
    }

    pid_t pid;
    if ( programArgs.size() > 0) {
        splitArgs(programArgs, checkLeakOnly);
    }

    if ((pid = fork()) == 0) {
        // Execute
        close(pp1[1]);
        dup2(pp1[0], STDIN_FILENO);
        close(pp2[0]);
        dup2(pp2[1], STDOUT_FILENO);
        dup2(STDOUT_FILENO, STDERR_FILENO);
        setpgrp();
        execve(command, (char * const *) argv, (char * const *) envv);
        perror("Internal error, execve fails");
        abort(); //end of child
    }
    if (pid == -1) {
        executionError = true;
        sprintf(executionErrorReason, "Internal error: fork error (%s)",
                strerror(errno));
        return;
    }
    close(pp1[0]);
    close(pp2[1]);
    int fdwrite = pp1[1];
    int fdread = pp2[0];
    Tools::fdblock(fdwrite, false);
    Tools::fdblock(fdread, false);
    programInput = input;
    if(programInput.size()==0){ // No input
        close(fdwrite);
    }
    programOutputBefore = "";
    programOutputAfter = "";
    pid_t pidr;
    int status;
    exitCode = std::numeric_limits<int>::min();
    while ((pidr = waitpid(pid, &status, WNOHANG | WUNTRACED)) == 0) {
        readWrite(fdread, fdwrite);
        usleep(5000);
        // TERMSIG or timeout or program output too large?
        if (Stop::isTERMRequested() || (time(NULL) - start) >= timeout
                || outputTooLarge) {
            if ((time(NULL) - start) >= timeout) {
                programTimeout = true;
            }
            kill(pid, SIGTERM); // Send SIGTERM normal termination
            int otherstatus;
            usleep(5000);
            if (waitpid(pid, &otherstatus, WNOHANG | WUNTRACED) == pid) {
                break;
            }
            if (kill(pid, SIGQUIT) == 0) { // Kill
                break;
            }
        }
    }
    if (pidr == pid) {
        if (WIFSIGNALED(status)) {
            int signal = WTERMSIG(status);
            executionError = true;
```

```
1477                sprintf(executionErrorReason,
1478                        "Program terminated due to \"%s\" (%d)\n", strsignal(
1479                                signal), signal);
1480            }
1481            if (WIFEXITED(status)) {
1482                exitCode = WEXITSTATUS(status);
1483            } else {
1484                executionError = true;
1485                strcpy(executionErrorReason,
1486                        "Program terminated but unknown reason.");
1487            }
1488        } else if (pidr != 0) {
1489            executionError = true;
1490            strcpy(executionErrorReason, "waitpid error");
1491        }
1492        readWrite(fdread, fdwrite);
1493
1494        if (checkLeakOnly)
1495            hasLeak = !matchMemoryLeak(programOutputAfter);
1496        else {
1497            correctExitCode = isExitCodeTested() && expectedExitCode == exitCode;
1498            correctOutput = match(programOutputAfter)
1499                        || match(programOutputBefore + programOutputAfter);
1500        }
1501    }
1502
1503    bool TestCase::matchMemoryLeak(string data) {
1504        if (!checkLeak) return true;
1505
1506        const char *out = data.c_str();
1507
1508        regex_t regexp_str;
1509        regcomp(&regexp_str, "All heap blocks were freed -- no leaks are possible", REG_EXTENDED|REG_NOSUB);
1510
1511        int retRegex = regexec(&regexp_str, out, 0, NULL, 0);
1512        return (retRegex == 0); // 0 = Match
1513    }
1514
1515    bool TestCase::match(string data) {
1516        for (size_t i = 0; i < output.size(); i++)
1517            if (output[i]->match(data))
1518                return true;
1519        return false;
1520    }
1521
1522    /**
1523     * Class Evaluation Definitions
1524     */
1525
1526    Evaluation::Evaluation() {
1527        grade = 0;
1528        ncomments = 0;
1529        nerrors = 0;
1530        nruns = 0;
1531        noGrade = true;
1532    }
1533
1534    Evaluation* Evaluation::getSinglenton() {
1535        if (singlenton == NULL) {
1536            singlenton = new Evaluation();
1537        }
1538        return singlenton; // Fixes by Jan Derriks
1539    }
1540
1541    void Evaluation::deleteSinglenton(){
1542        if (singlenton != NULL) {
1543            delete singlenton;
1544            singlenton = NULL;
1545        }
1546    }
1547
1548    void Evaluation::addTestCase(Case &caso) {
1549        if ( caso.getVariation().size() && caso.getVariation() != variation ) {
1550            return;
1551        }
1552        testCases.push_back(TestCase(testCases.size() + 1, caso.getInput(), caso.getOutput(),
1553                caso.getCaseDescription(), caso.getGradeReduction(), caso.getFailMessage(),
1554                caso.getProgramToRun(), caso.getProgramArgs(), caso.getExpectedExitCode(), caso.getCheckLeak() ) );
1555    }
1556
1557    void Evaluation::removeLastNL(string &s) {
1558        if (s.size() > 0 && s[s.size() - 1] == '\n') {
1559            s.resize(s.size() - 1);
1560        }
1561    }
1562
1563    bool Evaluation::cutToEndTag(string &value, const string &endTag) {
1564        size_t pos;
1565        if (endTag.size() && (pos = value.find(endTag)) != string::npos) {
1566            value.resize(pos);
1567            return true;
1568        }
1569        return false;
1570    }
1571
1572    void Evaluation::loadTestCases(string fname) {
1573        if(!Tools::existFile(fname)) return;
1574        const char *CASE_TAG = "case=";
1575        const char *INPUT_TAG = "input=";
1576        const char *INPUT_END_TAG = "inputend=";
1577        const char *OUTPUT_TAG = "output=";
1578        const char *OUTPUT_END_TAG = "outputend=";
1579        const char *GRADEREDUCTION_TAG = "gradereduction=";
1580        const char *FAILMESSAGE_TAG = "failmessage=";
1581        const char *PROGRAMTORUN_TAG = "programtorun=";
1582        const char *PROGRAMARGS_TAG = "programarguments=";
1583        const char *EXPECTEDEXITCODE_TAG = "expectedexitcode=";
1584        const char *VARIATION_TAG = "variation=";
1585        const char *HIDE_INPUT_OUTPUT_TAG = "hideinputoutput=";
1586
1587        enum {
1588            regular, ininput, inoutput
1589        } state;
1590        bool inCase = false;
1591        vector<string> lines = Tools::splitLines(Tools::readFile(fname));
1592        remove(fname.c_str());
1593        string inputEnd = "";
1594        string outputEnd = "";
1595        Case caso;
1596
1597        caso.setCheckLeak(CHECK_LEAK_FLAG);
1598
1599        string output = "";
```

```
    string tag, value;
    /* must be changed from String
     * to pair type (regexp o no) and string. */
    state = regular;
    int nlines = lines.size();
    for (int i = 0; i < nlines; i++) {
        string &line = lines[i];
        Tools::parseLine(line, tag, value);
        if (state == ininput) {
            if (inputEnd.size()) { // Check for end of input.
                size_t pos = line.find(inputEnd);
                if (pos == string::npos) {
                    caso.addInput(line + "\n");
                } else {
                    cutToEndTag(line, inputEnd);
                    caso.addInput(line);
                    state = regular;
                    continue; // Next line.
                }
            } else if (tag.size() && (tag == OUTPUT_TAG || tag
                    == GRADEREDUCTION_TAG || tag == CASE_TAG)) {// New valid tag.
                state = regular;
                // Go on to process the current tag.
            } else {
                caso.addInput(line + "\n");
                continue; // Next line.
            }
        } else if (state == inoutput) {
            if (outputEnd.size()) { // Check for end of output.
                size_t pos = line.find(outputEnd);
                if (pos == string::npos) {
                    output += line + "\n";
                } else {
                    cutToEndTag(line, outputEnd);
                    output += line;
                    caso.addOutput(output);
                    output = "";
                    state = regular;
                    continue; // Next line.
                }
            } else if (tag.size() && (tag == INPUT_TAG || tag == OUTPUT_TAG
                    || tag == GRADEREDUCTION_TAG || tag == CASE_TAG)) {// New valid tag.
                removeLastNL(output);
                caso.addOutput(output);
                output = "";
                state = regular;
            } else {
                output += line + "\n";
                continue; // Next line.
            }
        }
        if (state == regular && tag.size()) {
            if (tag == INPUT_TAG) {
                inCase = true;
                if (cutToEndTag(value, inputEnd)) {
                    caso.addInput(value);
                } else {
                    state = ininput;
                    caso.addInput(value + '\n');
                }
            } else if (tag == OUTPUT_TAG) {
                inCase = true;
                if (cutToEndTag(value, outputEnd))
                    caso.addOutput(value);
                else {
                    state = inoutput;
                    output = value + '\n';
                }
            } else if (tag == GRADEREDUCTION_TAG) {
                inCase = true;
                value = Tools::trim(value);
                // A percent value?
                if( value.size() > 1 && value[ value.size() - 1 ] == '%' ){
                    float percent = atof(value.c_str());
                    caso.setGradeReduction((grademax-grademin)*percent/100);
                }else{
                    caso.setGradeReduction( atof(value.c_str()) );
                }
            } else if (tag == EXPECTEDEXITCODE_TAG) {
                caso.setExpectedExitCode( atoi(value.c_str()) );
            } else if (tag == PROGRAMTORUN_TAG) {
                caso.setProgramToRun(Tools::trim(value));
            } else if (tag == PROGRAMARGS_TAG) {
                caso.setProgramArgs(Tools::trim(value));
            } else if (tag == FAILMESSAGE_TAG) {
                caso.setFailMessage(Tools::trim(value));
            } else if (tag == VARIATION_TAG) {
                caso.setVariation(value);
            } else if (tag == INPUT_END_TAG) {
                inputEnd = Tools::trim(value);
            } else if (tag == OUTPUT_END_TAG) {
                outputEnd = Tools::trim(value);
            } else if (tag == HIDE_INPUT_OUTPUT_TAG) {
                string hide_input_output_str = Tools::trim(value);
                bool hideInputOutput = (hide_input_output_str.compare("True") == 0);
                if (hideInputOutput && !caso.getFailMessage().size()) caso.setFailMessage( caso.getCaseDescription() );
            } else if (tag == CASE_TAG) {
                if (inCase) {
                    addTestCase(caso);
                    caso.reset();
                }
                inCase = true;
                caso.setCaseDescription( Tools::trim(value) );
            } else {
                if ( line.size() > 0 ) {
                    char buf[250];
                    sprintf(buf,"Syntax error: unexpected line %d", i+1);
                    addFatalError(buf);
                }
            }
        }
    }
    // TODO review
    if (state == inoutput) {
        removeLastNL(output);
        caso.addOutput(output);
    }
    if (inCase) { // Last case => save current.
        addTestCase(caso);
    }
}

bool Evaluation::loadParams() {
```

```
1723        grademin= Tools::getenv("VPL_GRADEMIN", 0.0);
1724        grademax = Tools::getenv("VPL_GRADEMAX", 10);
1725        maxtime = (int) Tools::getenv("VPL_MAXTIME", 20);
1726        variation = Tools::toLower(Tools::trim(Tools::getenv("VPL_VARIATION","")));
1727        noGrade = grademin >= grademax;
1728        return true;
1729    }
1730
1731    void Evaluation::addFatalError(const char *m) {
1732        float reduction = grademax - grademin;
1733        if (ncomments >= MAXCOMMENTS)
1734            ncomments = MAXCOMMENTS - 1;
1735
1736        snprintf(titles[ncomments], MAXCOMMENTSTITLELENGTH, "%s", m);
1737        snprintf(titlesGR[ncomments], MAXCOMMENTSTITLELENGTH, "%s (%.2f)", m, reduction);
1738        strcpy(comments[ncomments], "");
1739        ncomments ++;
1740        grade = grademin;
1741    }
1742
1743    void Evaluation::runTests() {
1744        if (testCases.size() == 0) {
1745            return;
1746        }
1747        if (maxtime < 0) {
1748            addFatalError("Global timeout");
1749            return;
1750        }
1751        nerrors = 0;
1752        nruns = 0;
1753        bool hasMemoryLeakFlag = false;
1754        grade = grademax;
1755        float defaultGradeReduction = (grademax - grademin) / testCases.size();
1756        int timeout = maxtime / testCases.size();
1757        for (size_t i = 0; i < testCases.size(); i++) {
1758            printf("Testing %lu/%lu : %s\n", (unsigned long) i+1, (unsigned long)testCases.size(), testCases[i].getCaseDescription().c_str());
1759            if (timeout <= 1 || Timer::elapsedTime() >= maxtime) {
1760                grade = grademin;
1761                addFatalError("Global timeout");
1762                return;
1763            }
1764            if (maxtime - Timer::elapsedTime() < timeout) { // Try to run last case
1765                timeout = maxtime - Timer::elapsedTime();
1766            }
1767            testCases[i].runTest(timeout);
1768            nruns++;
1769            if (!testCases[i].isCorrectResult() || testCases[i].hasMemoryLeak()) {
1770                if (Stop::isTERMRequested())
1771                    break;
1772                if (!testCases[i].isCorrectResult()) {
1773                    float gr = testCases[i].getGradeReduction();
1774                    if (gr == std::numeric_limits<float>::min())
1775                        testCases[i].setGradeReductionApplied(defaultGradeReduction);
1776                    else
1777                        testCases[i].setGradeReductionApplied(gr);
1778                    grade -= testCases[i].getGradeReductionApplied();
1779                    if (grade < grademin) {
1780                        grade = grademin;
1781                    }
1782                }
1783                if (testCases[i].hasMemoryLeak())
1784                    hasMemoryLeakFlag = true;
1785                nerrors++;
1786                if(ncomments<MAXCOMMENTS){
1787                    strncpy(titles[ncomments], testCases[i].getCommentTitle().c_str(),
1788                            MAXCOMMENTSTITLELENGTH);
1789                    strncpy(titlesGR[ncomments], testCases[i].getCommentTitle(true).c_str(),
1790                            MAXCOMMENTSTITLELENGTH);
1791                    strncpy(comments[ncomments], testCases[i].getComment().c_str(),
1792                            MAXCOMMENTSLENGTH);
1793                    ncomments++;
1794                }
1795            }
1796        }
1797
1798        if (hasMemoryLeakFlag) {
1799            grade -= MEMORY_LEAK_REDUCTION;
1800            printf("\n<|--\n");
1801            printf("-Memory leak: -%.2f\n", MEMORY_LEAK_REDUCTION);
1802            printf("--|>\n");
1803        }
1804        if (grade < grademin)
1805            grade = grademin;
1806    }
1807
1808    void Evaluation::outputEvaluation() {
1809        const char* stest[] = {" test", "tests"};
1810        if (testCases.size() > 0) {
1811            if (ncomments > 1) {
1812                printf("\n<|--\n");
1813                printf("-Failed tests\n");
1814                for (int i = 0; i < ncomments; i++) {
1815                    printf("%s", titlesGR[i]);
1816                }
1817                printf("--|>\n");
1818            }
1819            if ( ncomments > 0 ) {
1820                printf("\n<|--\n");
1821                for (int i = 0; i < ncomments; i++) {
1822                    printf("-%s", titlesGR[i]);
1823                    printf("%s\n", comments[i]);
1824                }
1825                printf("--|>\n");
1826            }
1827            int passed = nruns - nerrors;
1828            if ( nruns > 0 ) {
1829                printf("\n<|--\n");
1830                printf("-Summary of tests\n");
1831                printf(">+------------------------------+\n");
1832                printf(">| %2d %s run/%2d %s passed |\n",
1833                        nruns, nruns==1?stest[0]:stest[1],
1834                        passed, passed==1?stest[0]:stest[1]); // Taken from Dominique Thiebaut
1835                printf(">+------------------------------+\n");
1836                printf("\n--|>\n");
1837            }
1838            if ( ! noGrade ) {
1839                char buf[100];
1840                sprintf(buf, "%5.2f", grade);
1841                int len = strlen(buf);
1842                if (len > 3 && strcmp(buf + (len - 3), ".00") == 0)
1843                    buf[len - 3] = 0;
1844                printf("\nGrade :=>>%s\n", buf);
1845            }
```

```
    } else {
        printf("<|--\n");
        printf("-No test case found\n");
        printf("--|>\n");
    }
    fflush(stdout);
}

void nullSignalCatcher(int n) {
    //printf("Signal %d\n",n);
}

void signalCatcher(int n) {
    //printf("Signal %d\n",n);
    if (Stop::isTERMRequested()) {
        Evaluation* obj = Evaluation::getSinglenton();
        obj->outputEvaluation();
        abort();
    }
    Evaluation *obj = Evaluation::getSinglenton();
    if (n == SIGTERM) {
        obj->addFatalError("Global test timeout (TERM signal received)");
    } else {
        obj->addFatalError("Internal test error");
        obj->outputEvaluation();
        Stop::setTERMRequested();
        abort();
    }
    alarm(1);
}

void setSignalsCatcher() {
    // Removes as many signal controllers as possible
    for(int i=0;i<31; i++)
        signal(i, nullSignalCatcher);
    signal(SIGINT, signalCatcher);
    signal(SIGQUIT, signalCatcher);
    signal(SIGILL, signalCatcher);
    signal(SIGTRAP, signalCatcher);
    signal(SIGFPE, signalCatcher);
    signal(SIGSEGV, signalCatcher);
    signal(SIGALRM, signalCatcher);
    signal(SIGTERM, signalCatcher);
}

int main(int argc, char *argv[], const char **env) {
    Timer::start();
    TestCase::setEnvironment(env);
    setSignalsCatcher();
    Evaluation* obj = Evaluation::getSinglenton();
    obj->loadParams();
    obj->loadTestCases("evaluate.cases");
    obj->runTests();
    obj->outputEvaluation();
    return EXIT_SUCCESS;
}
```

vpl_test_main.c

```c
#include <stdio.h>
#include <stdlib.h>
#include "fila.h"

int main() {
    int ret, valor;
    char operacao;

    FilaDinamica *fila = criar_fila();

    scanf(" %c", &operacao);
    while (operacao != 'S') {
        if (operacao == 'E') {
            scanf("%d", &valor);
            ret = enfileirar(fila, valor);
            if (ret)
                printf("E %d\n", valor);
            else
                printf("Fila cheia\n");
        }
        if (operacao == 'D') {
            ret = desenfileirar(fila, &valor);
            if (ret)
                printf("D %d\n", valor);
            else
                printf("Fila vazia\n");
        }
        scanf(" %c", &operacao);
    }

    liberar_fila(fila);

    return 0;
}
```

fila.h

```c
#include "lista.h"

#ifndef FILA_DINAMICA
#define FILA_DINAMICA

typedef struct FilaDinamica FilaDinamica;
struct FilaDinamica {
   LinkedNode *inicio, *fim;
};

FilaDinamica *criar_fila();
int enfileirar(FilaDinamica *fila, int valor);
int desenfileirar(FilaDinamica *fila, int *valor);
void liberar_fila(FilaDinamica *fila);

#endif
```

lista.h

```
#ifndef LISTA_LIGADA
#define LISTA_LIGADA

typedef struct LinkedNode LinkedNode;
struct LinkedNode {
    int data;
    LinkedNode *next;
};

#endif
```

VPL

◄ [EP] Pilha dinâmica (hidden)

Jump to...

[EP] Duplicar números pares na lista (hidden) ►