**综合设计报告**

**课程名称：**  数据结构

**题 目：**  计算器

**指导教师：**  蔡英

**设计起始日期：** 2013.xx.xx-2013.xx.xx

**学 院：**  计算机学院

**系 别：** 计算机科学与技术系

**学生姓名：**  刘鸿喆

**班级/学号：**  计科1201/2012011107

**成 绩：**

**一、需求分析**

* 实验目的：掌握栈的运算及应用，了解对算法的健壮性要求。
* 实验内容：
  + 实现计算器类中lp、rp和operate函数；
  + 完善计算器类evaluate函数，增加对输入的合法性检查，包括滤掉所有非法输入及处理左右括号不配对的输入；
  + 编制应用程序测试这个计算器；

**二、概要设计**

**1、数据结构**

线性表：存储字符及优先级信息。

栈：存储计算式运算符和数字。

**2、使用算法**

利用运算符优先级表和栈决定计算顺序。

**三、详细设计**

**1、数据结构详细设计**

线性表：

优先级表：

int Calculator::lop[8];

int Calculator::rop[8;

字符：

SequenceList<char> Calculator::operatorList

SequenceList<char> Calculator::leftBracketList;

SequenceList<char> Calculator::rightBracketList;

SequenceList<char> Calculator::numberList;

SequenceList<char> Calculator::availableList;

栈：

Stack<char> oprt;

Stack<double> nmbr;

**2、算法**

通过优先级表：

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 符号 | + | - | \* | / | ^ | ( | ) | # |
| 左 | 3 | 3 | 5 | 5 | 7 | 1 | 8 | 0 |
| 右 | 2 | 2 | 4 | 4 | 6 | 8 | 1 | 0 |

决定运算顺序，栈空则计算完成。

**四、调试分析**

**1、调试过程中遇到的问题**

无。

**2、算法的时空分析**（n=算式长度）

时间：O(n)

空间：O(n)

**五、使用说明和测试结果**

**1、使用说明**

编译使用g++ -std=c++11 calculator.cpp -o calculator

运行，输入算式后回车，返回答案或错误。

**2、测试结果**

输入：

a1b-(2..86g.8+.2g5)\*g{[42/a(8)a]}a+..+y3.7-7

(56-23)/8-4

34+p(u89-12.3)k/3

67^3/70

89.5\*749+25)

(8\*(7-4)

65\*(72+98)(70-45)

6\*

)5+3(

()

输出

1-(2.868+.25)\*((42/(8)))+.+3.7-7=-18.6695

(56-23)/8-4=0.125

34+(89-12.3)/3=59.5667

67^3/70=4296.61

Syntax error: brackets did not match.

Syntax error: brackets did not match.

Syntax error: operators not found between brackets.

Syntax error: no more number after the last operator.

Syntax error: brackets did not match.

Syntax error: no more number after the last operator.

**六、心得体会**

封装好的数据结构类可以大大降低编程复杂度。

**七、附录**

**1、程序**

/\*\*

\* @file calculator.cpp

\* @brief A calculator in console.

\* @details Program for data structure experiment.

\* Tested under g++ (4.8.1) and clang (sorry for forgetting the version...).

\* Usage: g++ -std=c++11 calculator.cpp -o calculator.exe

\* \*-std=c++11 is an importent switch.

\* \*It will generate some warnings, please ignore it.

\* I haven't used any IDE, so there is no other files, please compile by yourself.

\* @author Darren Liu (MSR.B, msr-b)

\* @date 2013/10/10

\* @copyright Copyright (c) 2013 Darren Liu. All rights reserved.

\*/

/// @brief Debug switch, will enable file input

#define \_\_DEBUG\_\_

#include <iostream>

#ifdef \_\_DEBUG\_\_

#include <fstream>

#endif

#include "include/calculator.h"

/\*\*

\* @brief entry of the console application

\* @param argc number of console parameters

\* @param argv console parameter strings

\*/

int main(int argc, char const \*argv[]) {

char formula[1000]; // formula string

char \*correctFormula; // pointer for receiving correct formula

double result; // variable for receiving result

Calculator c; // calculator for calculating

#ifdef \_\_DEBUG\_\_ // debug switch, redirect input to file(calculater.in)

std::ifstream fin("calculator.in");

std::cin.rdbuf(fin.rdbuf());

#endif

while (std::cin.getline(formula, 999)) {

// std::cout << "formula: " << formula << std::endl; // print original formula

if (c.calculate(formula, correctFormula, &result)) {

// if calculate succeeded

std::cout << correctFormula << "=" << result << std::endl;

}

// an extra line for segmentation

}

return 0;

}

/\*\*

\* @file calculator.h

\* @brief A calculator class for calculating a formula.

\* @author Darren Liu (MSR.B, msr-b)

\* @date 2013/10/12

\* @copyright Copyright (c) 2013 Darren Liu. All rights reserved.

\*/

#ifndef \_CALCULATOR\_H\_

#define \_CALCULATOR\_H\_

#include <iostream>

/// @brief i did not use <string>, because i want to use pointers more

#include <cstring>

/// @brief for using pow()

#include <cmath>

/// @brief my string category, includes strtrm() for trimming strings and strrpl(n)() for replacing chars (4 overloads)

#include "string-category.h"

/// @brief my range class for packaging few features

#include "range.h"

/// @brief my data structure class of sequence list, i cannot use stl in this class after all...

#include "sequence-list.h"

/// @brief my data structure class of stack, same as previous, it's also made by myself

#include "stack.h"

/// @brief a calculator class provide string interpretation

class Calculator {

public:

/\*\*

\* @brief constructor

\*/

Calculator() {}

/\*\*

\* @brief for calculate formula

\*

\* @param str (const char \*) formula string pointer

\* @param correctFormula (char\*&) correct formula pointer

\* @param result (double\*) if interpretation succeded, return result of the formula

\*

\* @return succeded or not

\* @retval true interpretation succeded

\* @retval false interpretation failed

\*/

bool calculate(const char \*str, char \*&correctFormula, double \*result) {

char \*formula = new char[strlen(str) + 1]; // create a duplicate for str

char \*error = new char[100];

strcpy(formula, str);

removeIllogicalChars(formula); // remove illogical chars, check syntax

if (!checkSyntax(formula, error)) {

std::cout << "Syntax error: " << error << std::endl;

return false; // if syntax error occured, return false

}

correctFormula = new char[strlen(formula) + 1];

strcpy(correctFormula, formula); // create a duplicate for formula

/\*\* start calculate \*/

{

int i, l;

int nint; // int part

double ndec; // dec part

double a; // number 1

double b; // number 2

int power; // dec power

char op1; // left oprt

char op2; // right oprt

char c; // current char

bool inInt; // indicator of number parts

strcat(formula, "#"); // add a '#' at the back of the formula

l = strlen(formula); // get the length of the formula

Stack<char> oprt; // operator stack

Stack<double> nmbr; // number stack

oprt.push('#'); // push front '#'

for (i = 0; i < l; i++) {

if (numberList.in(c = formula[i])) {

// if this char is digital

inInt = true; // start from int part

ndec = nint = power = 0; // init all to 0

for (i = i; numberList.in(c); c = formula[++i]) {

// get the number

if (c == '.') {

// when meet '.' start adding dec part

inInt = false;

continue;

}

if (inInt) {

// if in int part

nint = nint \* 10 + c - '0';

} else {

// or in dec part

power++;

ndec += (c - '0') / pow(10, power);

// i confess that it's not a good implementation

}

}

i--; // because of the loop before, we must go back to last char

nmbr.push(nint + ndec); // push our new number to the stack

} else {

// or not

op1 = oprt.top(); // get the left operator

op2 = c; // get the right operator

if (relation(op1, op2) < 0) {

// if the priority of left is lower than right

oprt.push(op2); // push new operator to the stack

} else {

while (relation(op1, op2) > 0) {

// else while left is higher than right

oprt.pop(); // pop left

b = nmbr.top();

nmbr.pop(); // pop number 2

a = nmbr.top();

nmbr.pop(); // pop number 1

if (op1 == '/' && b == 0) {

std::cout << "Can not divided by 0." << std::endl;

delete formula; // don't forget to release!

delete error;

return false;

}

nmbr.push(execute(a, op1, b)); // caculate 1 left 2

op1 = oprt.top(); // assign new top to left

}

if (relation(op1, op2) == 0) {

// if left equals to right

oprt.pop(); // pop left (and right)

if (oprt.empty()) {

// if empty, that means '#' meats '#', stop calculating

break;

}

} else {

// else if left is lower than right

oprt.push(op2); // push right in

}

}

}

}

\*result = nmbr.top(); // result is at the bottom of number stack

nmbr.pop();

}

delete formula; // don't forget to release!

delete error;

return true; // calculate succeeded.

}

private:

/\*\*

\* @brief remove illogical charactors in the formula

\*

\* @param (char\*) formula

\*/

static void removeIllogicalChars(char \*formula) {

/\*\* remove unknown chars in formula \*/

{

strrpln(formula, availableList, ' ');

// replace all chars not in the available list

strtrm(formula);

// trim formula

}

/\*\* remove illogical dots \*/

{

int i, l;

bool dot;

l = strlen(formula); // get the length

for (i = 0; i < l; i++) {

dot = false; // init dot flag before find next number

for (; numberList.in(formula[i]) && i < l; i++) {

// if first char in number is found

if (formula[i] == '.') {

if (dot) {

// if there is already a dot

formula[i] = ' '; // replace it with ' '

continue; // goto next char

}

dot = true; // else if first find, turn flag to true

}

}

}

strtrm(formula); // trim formula

}

/\*\* replace brackets \*/

{

int i, l;

l = strlen(formula);

for (i = 0; i < l; i++) {

if (leftBracketList.in(formula[i])) {

// if a left bracket is found

formula[i] = '('; // replace it with '('

} else if (rightBracketList.in(formula[i])) {

// if a right bracket is found

formula[i] = ')'; // replace it with ')'

}

}

}

}

/\*\*

\* @brief check syntax of the formula

\*

\* @param formula (const char\*) the formula

\* @param error (char\*& error) error returning string

\*

\* @return bool

\* @retval true error not found

\* @retval false error found

\*/

static bool checkSyntax(const char \*formula, char \*&error) {

int i, l, layer;

bool num;

l = strlen(formula);

num = false;

for (i = 0, layer = 0; i < l; i++) {

if (formula[i] == '(') {

// when meets '('

layer++; // into a layer

if (i > 0 && (numberList.in(formula[i - 1]) || formula[i - 1] == ')')) {

// the char at the left side of '(' can not be digital or '.' or ')'

error = (char \*)("perators not found between brackets.");

return false;

} else if (i < l - 1 && operatorList.in(formula[i + 1])) {

// the char at the right side of '(' can not be operator

error = (char \*) "illogical operator found.";

return false;

}

} else if (formula[i] == ')') {

// when meets ')'

layer--; // escape a layer

if (layer < 0) {

// '(' must at the front of ')'

error = (char \*) "brackets did not match.";

return false;

} else if (i + 1 < l && (numberList.in(formula[i + 1]) || formula[i + 1] == '(')) {

// the char at the right side of ')' can not be digital or '.' or '('

error = (char \*) "operators not found between brackets.";

return false;

} else if (i > 0 && operatorList.in(formula[i - 1])) {

// the char at the left side of ')' can not be operator

error = (char \*) "illogical operator found.";

return false;

}

} else if (operatorList.in(formula[i])) {

// can not have two consecutive operator

// include \*((()))\*

if (!num) {

error = (char \*) "continuous operators found.";

return false;

}

num = false;

} else {

num = true;

}

}

if (!num) {

error = (char \*) "no more number after the last operator.";

return false;

}

if (layer) {

// if brackets are not matching

error = (char \*) "brackets did not match.";

return false;

}

return true;

}

/\*\*

\* @brief compare the priorities of two operators

\*

\* @param left (const char&) operator 1

\* @param right (const char&) operator 2

\*

\* @return the difference between priorities of two operators

\* @retval relation>0 priority of operator 1 is higher than operator 2

\* @retval relation=0 priority of operator 1 is equal to operator 2

\* @retval relation<0 priority of operator 1 is lower than to operator 2

\*/

static int relation(const char &left, const char &right) {

// return relation of operators (left - right)

return lop[oprtNum(left)] - rop[oprtNum(right)];

}

/\*\*

\* @brief get operator sn

\*

\* @param oprt (const char&) operator

\*

\* @return operator sn

\* @retval 0 +

\* @retval 1 -

\* @retval 2 \*

\* @retval 3 /

\* @retval 4 ^

\* @retval 5 (

\* @retval 6 )

\* @retval 7 #

\* @retval (exit) others

\*/

static int oprtNum(const char &oprt) {

// return sn of operator

switch (oprt) {

case '+':

return 0;

case '-':

return 1;

case '\*':

return 2;

case '/':

return 3;

case '^':

return 4;

case '(':

return 5;

case ')':

return 6;

case '#':

return 7;

default:

std::cerr << "Unknown operator" << std::endl;

exit(EXIT\_FAILURE);

}

}

/\*\*

\* @brief calculate a simple formula

\*

\* @param a (const double&) number 1

\* @param oprt (const char&) operator charactor

\* @param b (const double&) number 2

\* @return result

\*/

static double execute(const double &a, const char &oprt, const double b) {

// calculate a simple formula

switch (oprt) {

case '+':

return a + b;

case '-':

return a - b;

case '\*':

return a \* b;

case '/':

return a / b;

case '^':

return pow(a, b);

default:

std::cerr << "Unknown operator" << std::endl;

exit(EXIT\_FAILURE);

}

}

/// @name priority\_variables

/// @{

/// @brief priority of left operator

static int lop[8];

/// @brief priority of right operator

static int rop[8];

/// @}

/// @name lists

/// @{

/// @brief operator list

static SequenceList<char> operatorList ;

/// @brief left bracket list

static SequenceList<char> leftBracketList ;

/// @brief right bracket list

static SequenceList<char> rightBracketList;

/// @brief number list

static SequenceList<char> numberList ;

/// @brief available list

static SequenceList<char> availableList ;

/// @}

};

/// @name inits

/// @{

int Calculator::lop[8] = {3, 3, 5, 5, 7, 1, 8, 0};

int Calculator::rop[8] = {2, 2, 4, 4, 6, 8, 1, 0};

// init chars list

SequenceList<char> Calculator::operatorList = {'+', '-', '\*', '/', '^',

'\0'

};

SequenceList<char> Calculator::leftBracketList = {'(', '[', '{', '<',

'\0'

};

SequenceList<char> Calculator::rightBracketList = {')', ']', '}', '>',

'\0'

};

// i have a smarter implementation of these two lists,

// but i can't place it on the outside of functions

// so i want to show it below

//

// SequenceList<char> Calculator::numberList = {'.', '\0'};

// Calculator::numberList.merge(Range<char>('0', '9'));

// SequenceList<char> Calculator::availableList = {'\0'};

// Calculator::numberList.merge(operatorList);

// Calculator::numberList.merge(leftBracketList);

// Calculator::numberList.merge(rightBracketList);

// Calculator::numberList.merge(numberList);

SequenceList<char> Calculator::numberList = {'.', '0', '1', '2',

'3', '4', '5', '6',

'7', '8', '9', '\0'

};

SequenceList<char> Calculator::availableList = {'+', '-', '\*', '/', '^',

'(', '[', '{', '<',

')', ']', '}', '>',

'.', '0', '1', '2',

'3', '4', '5', '6',

'7', '8', '9', '\0'

};

/// @}

#endif

/\*\*

\* @file range.h

\* @brief Packaging range into a class.

\* @author Darren Liu (MSR.B, msr-b)

\* @date 2013/10/12

\* @copyright Copyright (c) 2013 Darren Liu. All rights reserved.

\*/

#ifndef \_RANGE\_H\_

#define \_RANGE\_H\_

/// @brief A range.

template <typename \_Ty>

class Range {

public:

/// @brief constructor of range, don't care about the magnitude relation

Range(const \_Ty &a, const \_Ty &b) {

this -> lower = (a < b) ? a : b;

this -> upper = (a > b) ? a : b;

}

/\*\*

\* @brief find whether a element in this range

\* @param x element

\* @return in or not

\* @retval true in

\* @retval false not in

\*/

bool in(const \_Ty &x) const {

return lower <= x && x <= upper;

}

/\*\*

\* @brief get lower limit

\* @return lower limit

\*/

\_Ty getLower() const {

return lower;

}

/\*\*

\* @brief get upper limit

\* @return upper limit

\*/

\_Ty getUpper() const {

return upper;

}

private:

/// @name limit

/// @{

/// @brief lower limit

\_Ty lower;

/// @brief upper limit

\_Ty upper;

/// @}

};

#endif

/\*\*

\* @file sequence-list.h

\* @brief Data structure: sequence list

\* @brief \*Still some unsolved bugs exists.\*

\* @author Darren Liu (MSR.B, msr-b)

\* @date 2013/10/11

\* @copyright Copyright (c) 2013 Darren Liu. All rights reserved.

\*/

#ifndef \_SEQUENCE\_LIST\_H\_

#define \_SEQUENCE\_LIST\_H\_

/// @brief enabling va\_list

#include <cstdarg>

#include <cstdlib>

#include <cstring>

/// @brief test type, avoid type improvment error

#include <typeinfo>

#include "range.h"

/// @brief Data structure: sequence list

template <typename \_Ty>

class SequenceList {

public:

/// @brief default

SequenceList() : \_n(0) {

this -> \_list = (\_Ty \*) malloc(sizeof(\_Ty));

}

/\*\*

\* @brief sequence list can be initialized with any amount of values

\* @param arg1 (\_Ty) arg1

\* @param arg2 (\_Ty) arg2

\* @param arg3 (\_Ty) arg3

\* ...

\* @param argEnd (\_Ty) ended with logical null of your type

\*/

SequenceList(\_Ty arg, ...) : \_n(0) {

va\_list vl;

const char \*typeName = typeid(\_Ty).name();

this -> \_list = (\_Ty \*) malloc(sizeof(\_Ty));

va\_start(vl, arg);

while (1) {

/// @brief WARNING! this will cause error in many occasions

if (!arg) {

break;

}

this -> push(arg);

if (!strcmp(typeName, typeid(char).name())) {

arg = va\_arg(vl, int);

} else if (!strcmp(typeName, typeid(float).name())) {

arg = va\_arg(vl, double);

} else {

arg = va\_arg(vl, \_Ty);

}

}

va\_end(vl);

}

/\*\*

\* @brief copy constructor

\*

\* @param list (const SequenceList&) list you want to copy

\*/

SequenceList(const SequenceList &list) : \_n(list.\_n) {

\_Ty \*newList = (\_Ty \*) malloc((list.\_n + 1) \* sizeof(\_Ty));

for (int i = 0; i < \_n; i++) {

newList[i] = list[i];

}

\_list = newList;

}

/\*\*

\* @brief push element into the sequence list

\*

\* @param x (const \_Ty&) element pushed

\*

\* @return succeeded or not

\* @retval true push succeded

\* @retval false push failed

\*/

bool push(const \_Ty &x) {

\_Ty \*newList = (\_Ty \*) realloc(\_list, (\_n + 2) \* sizeof(\_Ty));

if (!newList) {

return false;

}

newList[\_n++] = x;

\_list = newList;

return true;

}

/\*\*

\* @brief merge a Range<\_Ty> object

\*

\* @param range (const \_Ty&) your range

\*

\* @return succeeded or not

\* @retval true merge succeded

\* @retval false merge failed

\* @see bool push(const \_Ty& x)

\*/

bool merge(const Range<\_Ty> &range) {

\_Ty lower = range.getLower();

\_Ty upper = range.getUpper();

for (\_Ty i = lower; i <= upper; i++) {

if (!push(i)) {

return false;

}

}

return true;

}

/\*\*

\* @brief merge a SequenceList<\_Ty> object

\*

\* @param list (const \_Ty&) your list

\*

\* @return succeeded or not

\* @retval true merge succeded

\* @retval false merge failed

\* @see bool push(const \_Ty& x)

\*/

bool merge(const SequenceList<\_Ty> &list) {

for (int i = 0; i < list.count(); i++) {

if (!push(list[i])) {

return false;

}

}

return true;

}

/\*\*

\* @brief return element at index, exit when violence access

\*

\* @param pos (const int&) index

\*

\* @return element at index

\*/

\_Ty &at(const int &pos) const {

if (pos < \_n) {

return (\_list) [pos];

}

exit(EXIT\_FAILURE);

}

/\*\*

\* @brief same as at()

\*

\* @param pos (const int&) index

\*

\* @return element at index

\* @see \_Ty& at(const int& pos) const

\*/

\_Ty &operator[](const int &pos) const {

return at(pos);

}

/\*\*

\* @brief find whether a element in list

\* @param x element

\* @return in or not

\* @retval true in

\* @retval false not in

\*/

bool in(const \_Ty &x) const {

for (int i = 0; i < \_n; i++) {

if (\_list[i] == x) {

return true;

}

}

return false;

}

/\*\*

\* @brief get amount of elements in list

\* @return element amount

\*/

int count() const {

return \_n;

}

/// @brief destructor of sequence list

~SequenceList() {

delete \_list;

}

private:

/// @brief list pointer

\_Ty \*\_list;

/// @brief element amount

int \_n;

};

#endif

/\*\*

\* @file stack.h

\* @brief Data structure: stack

\* @author Darren Liu (MSR.B, msr-b)

\* @date 2013/10/13

\* @copyright Copyright (c) 2013 Darren Liu. All rights reserved.

\*/

#ifndef \_STACK\_H\_

#define \_STACK\_H\_

#include <iostream>

#include <cstdlib>

/\*\*

\* @brief Data structure: stack

\*/

template <typename \_Ty>

class Stack {

public:

/\*\*

\* @brief constructor of stack

\*/

Stack() : \_top(NULL) {}

/\*\*

\* @brief push element into the stack

\*

\* @param data (const \_Ty&) the element

\*

\* @return succeeded or not

\* @retval true push succeded

\* @retval false push failed

\*/

bool push(const \_Ty &data) {

StackNode \*top = new StackNode(data, \_top);

if (!top) {

return false;

}

\_top = top;

return true;

}

/\*\*

\* @brief remove the top element

\*

\* @return succeed or not

\* @retval true pop succeded

\* @retval false pop failed

\* @see \_Ty top() const

\*/

bool pop() {

if (empty()) {

return false;

}

StackNode \*top = \_top;

\_top = \_top -> next;

delete top;

return true;

}

/\*\*

\* @brief test whether the stack is empty

\*

\* @return empty or not

\* @retval true empty

\* @retval false not empty

\*/

bool empty() {

return (\_top) ? false : true;

}

/\*\*

\* @brief return the top element

\*

\* @return top element

\*/

\_Ty top() const {

if (!\_top) {

std::cerr << "No elements in stack." << std::endl;

exit(EXIT\_FAILURE);

}

return \_top -> data;

}

/// @brief destructor of stack

~Stack() {

StackNode \*top = \_top;

for (; top; top = \_top) {

\_top = \_top -> next;

delete top;

}

}

private:

class StackNode {

public:

StackNode(const \_Ty &data, StackNode \*next = NULL) : data(data), next(next) {}

\_Ty data;

StackNode \*next;

};

/// @brief top pointer

StackNode \*\_top;

};

#endif

/\*\*

\* @file string-category.h

\* @brief My cstring category for c++.

\* @author Darren Liu (MSR.B, msr-b)

\* @date 2013/10/12

\* @copyright Copyright (c) 2013 Darren Liu. All rights reserved.

\*/

#ifndef \_STRING\_CATEGORY\_H\_

#define \_STRING\_CATEGORY\_H\_

#include <cstring>

#include "range.h"

#include "sequence-list.h"

/\*\*

\* @brief replace charactors in specific range

\*

\* @param str (char\*) string

\* @param range (Range<char>) charactor range

\* @param to (char) replace to

\*

\* @return amount of charactors replaced

\*/

int strrpl(char \*str, Range<char> range, char to) {

int i, n = 0, l = strlen(str);

for (i = 0; i < l; i++) {

if (range.in(str[i])) {

str[i] = to;

n++;

}

}

return n;

}

/\*\*

\* @brief replace charactors in specific list

\*

\* @param str (char\*) string

\* @param list (SequenceList<char>) charactor list

\* @param to (char) replace to

\*

\* @return amount of charactors replaced

\*/

int strrpl(char \*str, SequenceList<char> list, char to) {

int i, n = 0, l = strlen(str);

for (i = 0; i < l; i++) {

if (list.in(str[i])) {

str[i] = to;

}

}

return n;

}

/\*\*

\* @brief replace charactors \*not\* in specific range

\*

\* @param str (char\*) string

\* @param range (Range<char>) charactor range

\* @param to (char) replace to

\*

\* @return amount of charactors replaced

\*/

int strrpln(char \*str, Range<char> range, char to) {

int i, n = 0, l = strlen(str);

for (i = 0; i < l; i++) {

if (!range.in(str[i])) {

str[i] = to;

n++;

}

}

return n;

}

/\*\*

\* @brief replace charactors \*not\* in specific list

\*

\* @param str (char\*) string

\* @param list (SequenceList<char>) charactor list

\* @param to (char) replace to

\*

\* @return amount of charactors replaced

\*/

int strrpln(char \*str, SequenceList<char> list, char to) {

int i, n = 0, l = strlen(str);

for (i = 0; i < l; i++) {

if (!list.in(str[i])) {

str[i] = to;

}

}

return n;

}

/\*\*

\* @brief remove spaces and tabs in string

\*

\* @param str (char\*) string

\*

\* @return amount of charactors deleted

\*/

int strtrm(char \*str) {

char \*p;

int n = 0;

for (p = str; \*str != '\0'; str++) {

if (\*str != ' ' && \*str != '\t') {

\* (p++) = \*str;

} else {

n++;

}

}

\*p = '\0';

return n;

}

#endif