計算機程式語言

作　　業　　七

姓名	潘廣霖
學號	B03203004
原始程式檔名	HW07_002.cpp (_ = [A-C]), where C is for the challenge problem

評　分　項　目

分數比重	項　　目	得　分
40%	程式是否能正確執行?	
40%	程式之使用者介面與輸出結果?	
	是否有繳交原始程式檔與執行檔?	
	程式中的註解是否恰當?	
	程式之結構與邏輯是否正確?	
	程式碼的格式是否合乎要求?	
20%	程式之綜合評分	
總　分		

評語:

A. (utilizes a hash structure)

```
//=================================================================
//  PROGRAMMER  : 潘廣霖
//  DATE        : 2015-12-17
//  FILENAME    : HW07A002.CPP
//  DESCRIPTION : Finding the longest string in a file, whose reversed one is also on a list.
//=================================================================

#include<iostream>
#include<algorithm>  // std::reverse
#include<cstdlib>
#include<fstream>

#define HASH_SIZE 100007

using namespace std;

string* strHash[HASH_SIZE];
int hashEntryUsed = 0;

char lower(char x) {
    if (x >= 'A' && x <= 'Z')
        return x - 'A' + 'a';
    return x;
}

bool hashEqual(string* s1, string* s2) {
    // ignore the last character
    // compare strings case-insensitively
    if (s1->length() != s2->length())
        return false;
    for (size_t i = 0, n = s1->length() - 1; i < n; i++)
        if (lower((*s1)[i]) != lower((*s2)[i]))
            return false;
    return true;
}

int main() {
    ifstream f("words.txt");
    if (!f) {
        cerr << "File `words.txt` does not exist! Exiting...";
        system("pause");
        return 1;
    }

    char buf[1024];
    string* longest = 0;
    string* longest_conj = 0;


    unsigned hashColl = 0;
    while (f >> buf) {
        int i = 0;
        size_t key = 0;

        for (; buf[i] != '\0'; i++)
            // calculate hash of the string
            key = (key + lower(buf[i]) * 3647) % HASH_SIZE;

        // i is now the length of the string
        if (lower(buf[0]) > lower(buf[i - 1])) {
            reverse(buf, buf + i);
            buf[i] = 2;
        } else buf[i] = 1;
        buf[i + 1] = '\0';


        string* s = new string(buf);

        // this should NOT happen
        if (hashEntryUsed == HASH_SIZE) {
            cerr << "Hash buckets full!" << endl;
            system("pause");
            return 1;
        }

        while (strHash[key] != 0) {
            // if the string is the same, we've found a candidate
            // otherwise use linear probing
            char c1 = (*strHash[key])[strHash[key]->length() - 1];
            char c2 = (*s)[s->length() - 1];

            if (c1 != c2 && hashEqual(strHash[key], s)) {
                if (!longest || longest->length() < s->length()) {
                    if (c1 == 1) {
                        longest = strHash[key];
                        longest_conj = s;
                    } else {
                        longest = s;
                        longest_conj = strHash[key];
                    }

                }
            }

            key = (key + 1) % HASH_SIZE;
            hashColl++;
        }
        strHash[key] = s;
        hashEntryUsed++;
    }

    cout << "collision count: " << hashColl << endl;

    // clear the flag of the answer
    longest->erase(longest->length() - 1);
    longest_conj->erase(longest_conj->length() - 1);
```

```
    // reversed flag is true, so do reverse it back
    reverse(longest_conj->begin(), longest_conj-
>end());

    // output the answer
    if (longest)
        cout << "The longest reversible string is \""
             << *longest << "\" and \""
             << *longest_conj << "\", with its length
"
             << longest->length() << "."
             << endl;
    else
        cout << "No reversible string found.";

    // for (size_t i = 0; i < HASH_SIZE; i++) {
    //     if (i % 10 != 0) continue;
    //     if (i && i % 1000 == 0) cout << "|" <<
endl;
    //     if (strHash[i]) {
    //         cout << "*";
    //     } else cout << " ";
    // }
    // cout << "#" << endl;

    // free the hash table
    for (size_t i = 0; i < HASH_SIZE; i++)
        delete strHash[i];

    // system("pause");
    return 0;
}
```

B.

```
//=====================================================================
//  PROGRAMMER  : 潘廣霖
//  DATE        : 2015-12-30
//  FILENAME    : HW07B002.CPP
//  DESCRIPTION : A naive, OOP fraction calculator
//=====================================================================

#include<stdafx.h>
#include<iostream>
#include<iomanip>
#include<sstream>
#include<cstdlib>
#include<cmath>

using namespace std;

// you can switch to long long here easily
typedef int INTEGER;

INTEGER gcd(INTEGER a, INTEGER b) {
    return a ? gcd(b % a, a) : abs(b);
}

class Fractions {
public:
    Fractions() { num = 0; denom = 1; }
    Fractions(INTEGER n, INTEGER d) {
        if (d == 0) throw false;
        num = n;
        denom = d;
        normalize();
    }
    friend ostream& operator << (ostream&, const
Fractions&);

    Fractions operator + (Fractions rhs) {
        INTEGER d = gcd(denom, rhs.denom);
        return Fractions(rhs.denom / d * num + denom
/ d * rhs.num, denom / d * rhs.denom);
    }
    Fractions operator - (Fractions rhs) {
        INTEGER d = gcd(denom, rhs.denom);
        return Fractions(rhs.denom / d * num - denom
/ d * rhs.num, denom / d * rhs.denom);
    }
    Fractions operator * (Fractions rhs) {
        // TODO: avoid overflow
        return Fractions(num * rhs.num, denom *
rhs.denom);
    }
    Fractions operator / (Fractions rhs) {
        // TODO: avoid overflow
        INTEGER n = denom * rhs.num;
        if (n == 0) throw false;
        return Fractions(num * rhs.denom, n);
    }
private:
    int num;
    int denom;

    void normalize() {
        int g = gcd(num, denom);
        num /= g;
        denom /= g;
        // take negative sign to the numerator
        if (denom < 0) {
            num *= -1;
            denom *= -1;
        }
    }
};

ostream& operator << (ostream& os, const Fractions&
f) {
    if (f.denom == 1) {
        // avoid fractions like (2/1), they are
integers
        os << f.num;
```

```
        return os;
    }
    os << "(" << f.num << "/" << f.denom << ")";
    return os;
}

inline bool isValidOp(char op) {
    return (op == '+' || op == '-' || op == '*' || op
== '/');
}

void printHeader() {
    cout << "========= FRACTION CALCULATOR ========="
<< endl;
    cout << "Syntax: <fraction> [<op> <fraction>]" <<
endl;
    cout << "Input '?' for more details." << endl;
    cout << "======================================="
<< endl;
    cout << endl;
}

void printUsage() {
    cout <<
"=============================================" <<
endl;
    cout << "Grammar:" << endl;
    cout << " <fraction>: <INT> ['/' <INT>]" << endl;
    cout << "       <op>: '+' | '-' | '*' | '/'" <<
endl;
    cout << "Examples:" << endl;
    cout << "  \"1/3+2/5\": (1/3) + (2/5) => (11/15)"
<< endl;
    cout << "  \"-3/4-4\": (-3/4) - (4/1) => (-19/4)"
<< endl;
    cout << "  \"6/2/2/1\": (6/2) / (2/1) => (3/2)"
<< endl;
    cout << "  \"7/0\": error, divided by zero!!" <<
endl;
    cout << "  \"7/2/0/1\": (7/2) / (0/1) => error!!"
<< endl;
    cout << "Notes:" << endl;
    cout << "  * Never divide with zero. Never." <<
endl;
    cout << "  * The input format is different from
common" << endl;
    cout << "    expressions in real life." << endl;
    cout <<
"=============================================" <<
endl;
    cout << endl;
}

bool parseError(stringstream& ss, string errMessage,
int offset = 0) {
    ss.clear();
    int colNo = (int) ss.tellg();
    if (colNo <= 0) colNo = 1;
    colNo += offset;
    cerr
    //   << ss.str() << endl
         << setw(colNo) << right << "^" << endl
         << "Parse error: col #" << colNo << ", " <<
errMessage << endl;
    return false;
}

bool inputOperator(stringstream& s, char& op) {
    s >> op;
    if (s.fail()) {
        op = '\0';
        return true;
    }
    if (!isValidOp(op))
        return parseError(s, "invalid operator!!");
    return true;
}

bool inputFraction(stringstream& s, INTEGER& x,
INTEGER& y) {
    s >> x;
    if (s.fail() && s.eof())
        return parseError(s, "numerator expected!!");
    if (s.fail())
        return parseError(s, "invalid numerator!!");

    char c;
    s >> c;
    if (c != '/') {
        y = 1;
        s.unget();
        return true;
        //return parseError(s, "\"/\" expected for a
fraction!!");
    }

    s >> y;
    if (s.fail() && s.eof())
        return parseError(s, "denominator
expected!!", 1);
    if (s.fail())
        return parseError(s, "invalid denominator!!",
1);

    if (y == 0)
        return parseError(s, "zero cannot be a
denominator!!");

    return true;
}

bool parseCommand(string l, Fractions*& fr1,
Fractions*& fr2, char& op) {
    INTEGER a, b;
    stringstream ss(l);

    if (!inputFraction(ss, a, b)) return false;
    fr1 = new Fractions(a, b);

    if (!inputOperator(ss, op)) return false;
    if (!op) return true;

    if (!inputFraction(ss, a, b)) return false;
    fr2 = new Fractions(a, b);

    // ensure no extra character
    char tmp;
    ss >> tmp;
```

```
    if (!ss.fail()) return parseError(ss, "extra
character!!");
    return true;
}

#ifdef _MSC_VER
int main(int argc, _TCHAR* argv[]) {
#else
int main() {
#endif // _MSC_VER
    char op;

    Fractions* fr1 = 0;
    Fractions* fr2 = 0;
    Fractions ans;

    printHeader();
    string l;
    while (getline(cin, l)) {

        if (l == "?") {
            printUsage();
            continue;
        }

        if (!parseCommand(l, fr1, fr2, op)) {
            cout << endl;
            continue;
        }

        if (op == '\0') {
            cout << "Input:  " << *fr1 << endl;
        } else {
            if (op == '+')
                ans = *fr1 + *fr2;
            else if (op == '-')
                ans = *fr1 - *fr2;
            else if (op == '*')
                ans = *fr1 * *fr2;
            else if (op == '/')
                try {
                    ans = *fr1 / *fr2;
                } catch (bool b) {
                    cout << "Exception: cannot
divided by zero!!" << endl;
                    cout << endl;
                    continue;
                }

            cout << "Input:  " << *fr1 << " " << op
<< " " << *fr2 << endl;
            cout << "Result: " << ans << endl;
        }

        // empty line for each line of input
        cout << endl;

        // recycle
        delete fr1; fr1 = 0;
        delete fr2; fr2 = 0;
    }
}
```

C. (challenge)

```
//================================================================
//  PROGRAMMER  : 潘廣霖
//  DATE        : 2015-12-25
//  FILENAME    : HW07C002.CPP
//  DESCRIPTION : A sudoku solver.
//================================================================

#include<iostream>
#include<algorithm>

using namespace std;

#define BLOCK_NUM(i,j) ((i) / 3 * 3 + (j) / 3)

int g[9][9];
unsigned ava[9][9];
unsigned cnt[9][9];

unsigned row[9];
unsigned col[9];
unsigned blk[9];

unsigned dfsList[81], dfsLen;

void setMask(int i, int j, unsigned b) {
    row[i] |= b;
    col[j] |= b;
    blk[BLOCK_NUM(i,j)] |= b;
}

void unsetMask(int i, int j, unsigned b) {
    row[i] ^= b;
    col[j] ^= b;
    blk[BLOCK_NUM(i,j)] ^= b;
}

inline bool isLegalMask(int i, int j, unsigned b) {
    return !(row[i]&b) && !(col[j]&b) && !
(blk[BLOCK_NUM(i,j)]&b);
}

bool backtrack(unsigned depth) {
    if (depth == dfsLen)
        return true;

    unsigned i = dfsList[depth] / 9;
    unsigned j = dfsList[depth] % 9;

    int k = 1;
```

```
    unsigned a = ava[i][j] >> 1;

    while (a > 0) {
        if (a & 1) {
            // fill in k
            unsigned b = 1 << k;
            if (isLegalMask(i, j, b)) {
                g[i][j] = k;
                setMask(i, j, b);
                if (backtrack(depth + 1)) return
true;
                // undo
                unsetMask(i, j, b);
            }
        }
        k++;
        a >>= 1;
    }
    return false;
}

bool inputEntry(int i, int j) {
    cin >> g[i][j];
    if (g[i][j] < 0 || g[i][j] > 9) return false;

    if (!g[i][j]) {
        return true;
    }

    unsigned bit = (1 << g[i][j]);
    if (!isLegalMask(i, j, bit)) return false;
    setMask(i, j, bit);
    return true;
}

int main() {
    // read in and validate a sudoku puzzle from
stdin
    for (int i = 0; i < 9; i++)
        for (int j = 0; j < 9; j++) {
            if (!inputEntry(i, j)) {
                cout << "Illegal entry (" << g[i][j]
<< ") for row #" << (i+1) << " col #" << (j+1) <<
"!!" << endl;
                return 1;
            }
        }

    cout << "Puzzle: " << endl;

    // generate the table of possible entries
    for (int i = 0; i < 9; i++) {
        for (int j = 0; j < 9; j++) {
            // print out original puzzle
            if (g[i][j])
                cout << g[i][j];
            else
                cout << "_";
            cout << " ";

            if (g[i][j]) continue;

            unsigned b = 2;
            for (int k = 1; k <= 9; k++) {
                if (isLegalMask(i, j, b)) {
                    // cout << k << " ";
                    ava[i][j] |= b;
                    cnt[i][j] += 1;
                }
                b <<= 1;
            }

            // categorize blanks by the number of
possible values
            // not knowing if vector is allowed...
pack them into an array
            dfsList[dfsLen++] = i * 9 + j;
        }
        cout << endl;
    }
    cout << endl;

    cout << "Solving..." << flush;

    bool result = backtrack(0);
    if (result) {
        cout << "\rResult: SOLVED" << endl;
        for (int i = 0; i < 9; i++) {
            for (int j = 0; j < 9; j++) {
                cout << g[i][j] << " ";
            }
            cout << endl;
        }
    } else {
        cout << "\rResult: NO SOLUTION" << endl;
        cout << "Sorry :(" << endl;
    }
    cin.get();

}
```