Technical University of Denmark

Written examination, August 20, 2016

Course name: Programming in C++

Course number: 02393

Aids allowed: All aids allowed

Exam duration: 4 hours

Weighting: pass/fail

Exercises: 4 exercises of 2.5 points each for a total of 10 points.

Submission details:

1. You can hand-in your solutions manually (on paper). However, we strongly recommend you to submit them electronically.
2. For electronic submission, you **must** upload your solutions on CampusNet and you can do it only once: resubmission is not possible, so submit only when you have finished all exercises. Each exercise must be uploaded as one separate `.cpp` file, using the names specified in the exercises, namely `exZZ-library.cpp`, where `ZZ` ranges from `01` to `04`. The files must be handed in separately (not as a zip-file) and must have these exact filenames. Feel free to add comments to your code.
3. You *can also* upload your solutions individually on CodeJudge under `Re-Exam` at `https://dtu.codejudge.net/02393-f16/assignment`. When you hand in a solution on CodeJudge, the test example given in the assignment description will be run on your solution. Consider that additional tests may be run on your solutions after the exam. You can upload to CodeJudge as many times as you like during the exam.

Exercise 1. Reducing Matrices (2.5 points)

Alice needs to perform some computations on square matrices. She has already implemented part of the code but she is not sure about its correctness, and some parts are still missing. Her first test program is in file `ex01-main.cpp` and the (incomplete) code with some functions she needs is in files `ex01-library.h` and `ex01-library.cpp`. All files are available on CampusNet and in the next pages. Help Alice by solving the following tasks:

(a) Check the implementation of function `void display(double * * A, unsigned int n)` and correct it if necessary. The function should correctly display the `n` × `n` matrix `A`. For a `3`×`3` square matrix with all 0s the expected output is

```
0 0 0
0 0 0
0 0 0
```

Notice that Alice has decided to represent an `n`×`n` square matrix as an array of arrays (each of size n). Recall that in such a representation the element at row `i` and column `j` in a matrix `A` is accessed by `A[i][j]`.

(b) Implement function `reset(double * * A, unsigned int n, double x)`. The function should set all cells in the `n`×`n` square matrix `A` to value `x`.

(c) Implement function `vector<double> sumRows(double * * A, unsigned int n)`. This function should take as input an `n`×`n` square matrix `A` and should return a vector that contains the sums of values of each row in `A`. For the first matrix below in task (e), the vector would be $(1, 5, 4)$.

(d) Implement function `vector<double> sumCols(double * * A, unsigned int n)`. This function should take as input an `n`×`n` square matrix `A` and should return a vector that contains the sums of values of each column in `A`. For the first matrix below in task (e), the vector would be $(2, 5, 3)$.

(e) Implement the function `reduce(double * * A, unsigned int n)`. This function should take as input an `n`×`n` square matrix `A` and should update all elements of the matrix according to the following idea: each cell should take the sum of all *adjacent* cells. Adjacent cells are cells that are above, below, leftwards or rightwards. Let $a_{i,j}$ be the cell in row i and column j. For $a_{0,0}$ the adjacent cells are $a_{0,1}$ and $a_{1,0}$ only, since there is no cell leftwards or above $a_{0,0}$. As an example, reducing this matrix:

```
0 1 0                          3 0 4
2 0 3      Should update it to 0 10 0
0 4 0                          6 0 7
```

Exercise follows in next page...

File `ex01-main.cpp`

```
#include <iostream>
#include <string>
#include "ex01-library.h"

using namespace std;

int main(void){

    // I am building my initial matrix here
    unsigned int n = 3;
    double * * A = new double *[n];
    for(unsigned int i = 0; i<n; i++){
        A[i] = new double[n];
    }

    // Setting all values to 0
    reset(A,n,0);
    // Setting some values in the matrix
    A[0][1] = 1;
    A[1][0] = 2;
    A[1][2] = 3;
    A[2][1] = 4;
    display(A,n);
    cout << endl;

    // Summing up rows and values
    vector<double> v;
    v = sumRows(A,n);
    print(v);
    v = sumCols(A,n);
    print(v);
    cout << endl;

    // Reducing the matrix
    reduce(A,n);
    display(A,n);

    for(unsigned int i = 0; i<n; i++){
        delete [] A[i];
    }
    delete [] A;

    return 0;

}
```

File `ex01-library.h`

```
#ifndef __ex01_library__
#define __ex01_library__

#include <vector>

using namespace std;

void display(double * * A, unsigned int n);
void reset(double * * A, unsigned int n, double x);
void reduce(double * * A, unsigned int n);
vector<double> sumRows(double * * A, unsigned int n);
vector<double> sumCols(double * * A, unsigned int n);
void print(vector<double> & v);

#endif
```

File `ex01-library.cpp`

```
#include <iostream>
#include <vector>
#include "ex01-library.h"

using namespace std;

// Exercise 1 (a)
// Check and correct if necessary
void display(double * * A, unsigned int n){
    for(unsigned int i = 0; i <= n; i++){
        for(unsigned int j = 0; j <= n; j++){
            cout << A[i][j] << "␣";
        }
        cout << endl;
    }
}

// Exercise 1 (b)
// Implement this function
void reset(double * * A, unsigned int n, double x){
    // Put your code here
}

// Exercise 1 (c)
// Implement this function
vector<double> sumRows(double * * A, unsigned int n){
    // Put your code here
}

// Exercise 1 (d)
// Implement this function
vector<double> sumCols(double * * A, unsigned int n){
    // Put your code here
}

// Exercise 1 (e)
// Implement this function
void reduce(double * * A, unsigned int n){
    // Put your code here
}

// Do not modify
void print(vector<double> & v){
    for(unsigned int i=0; i<v.size(); i++){
        cout << v[i] << "␣";
    }
    cout << endl;
}
```

Exercise 2. Matching Sequences (2.5 points)

Bob works for a bioinformatics lab and needs to perform a complex operation to match sequences of elements related to DNA and other biological data. Given two sequences of elements $u = u_1, u_2, \ldots, u_k$ and $v = v_1, v_2, \ldots, v_l$ the *match* function returns a new sequence and is recursively defined as follows

$$\text{match}(u, v) = \begin{cases} \epsilon & \text{if } u = \epsilon \text{ or } v = \epsilon \\ \text{match}(p(u), p(v)), t(u) & \text{if } t(u) = t(v) \\ \text{match}(u, p(v)) & \text{if } |\,\text{match}(u, p(v))| \geq |\,\text{match}(p(u), v)| \\ \text{match}(p(u), v) & \text{otherwise} \end{cases}$$

where

- ϵ denotes the empty sequence;
- $|w|$ denotes the length of a sequence w;
- concatenation of sequences is denoted with a comma ",";
- $t(w)$ denotes the last element of a non-empty sequence, i.e. $t(w_1, w_2, w_3, \ldots, w_n) = w_n$;
- $p(w)$ denotes the sequence obtained by removing the last element of w, that is $p(w_1, w_2, w_3, \ldots, w_{n-1}, w_n) = w_2, w_3, \ldots, w_{n-1}$. If w is empty or has just one element then $p(w)$ is just ϵ;

As an example you can easily check that matching the sequences X, Y, Z and X, Z yields

	$match((X, Y, Z), (X, Z))$	
$=$	$match((X, Y), (X)), Z$	since $t(X, Y, Z) = t(X, Z) = Z$
$=$	$match((X), (X)), Z$	since $\lvert match((X), (X))\rvert \geq \lvert match((X, Y), (\epsilon))\rvert$
$=$	$(match((\epsilon), (\epsilon), X)), Z$	since $t(X) = t(X) = X$
$=$	X, Z	since $match(\epsilon, \epsilon) = \epsilon$ and $u, \epsilon = u$ for all sequences u

Bob has already written some code. His first test program is in file `ex02-main.cpp` and the (incomplete) code with some functions he needs is in files `ex02-library.h` and `ex02-library.cpp`. All files are available on CampusNet and in the next pages.

Exercise follows in next page...

As you can see Bob has decided to represent elements with strings and sequences of elements with vectors of strings. He has already implemented some functions to read the sequences from `stdin` and print a sequence in `stdout`. What he is missing is the implementation of function `match` in file `ex02-library.cpp`. Help Bob implementing such function.

File `ex02-main.cpp`

```
#include <iostream>
#include <vector>
#include <string>
#include "ex02-library.h"

int main(void){

    // Read two sequences of strings
    // end of sequence is denote by "STOP"
    vector<string> u = read_until("STOP");
    vector<string> v = read_until("STOP");

    // Match the sequences
    vector<string> w = match(u,v);

    // Display the result
    display(w);

    return 0;

}
```

File `ex02-library.h`

```
#ifndef __ex02_library__
#define __ex02_library__

#include <vector>
#include <string>

using namespace std;

vector<string> match(vector<string> & u,
                     vector<string> & v);

vector<string> read_until(string stop);

void display(vector<string> & u);

#endif
```

File `ex02-library.cpp`

```
#include <iostream>
#include <vector>
#include <string>
#include "ex02-library.h"

using namespace std;

// Exercise 2
vector<string> match(vector<string> & u,
                     vector<string> & v){

    // Put your code here

}

// Do not modify
vector<string> read_until(string stop){
    vector<string> u;
    string e;
    while(true){
        cin >> e;
        if(cin.fail() || e == stop) break;
        u.push_back(e);
    }
    return u;
}

// Do not modify
void display(vector<string> & u){
    for(unsigned int i=0; i<u.size(); i++)
        cout << u[i] << "␣" ;
    cout << endl;
}
```

Exercise 3. Lost in Translation (2.5 points)

Claire wants to implement a class `Dictionary` to support some basic translation functionalities, like translating a word between languages or obtaining a word's synonyms in a given language. Her first test program is in file `ex03-main.cpp` and the (incomplete) code with some functions he needs is in files `ex03-library.h` and `ex03-library.cpp`. All files are available on CampusNet and in the next pages. Help Claire by implementing the class `Dictionary` in file `ex03-library.cpp`.

Claire does not know how to implement the methods but she has been told that the `map` containers of the standard library already provide a lot of the functionalities she needs. So she has decided to use the following internal (`private`) representation for the library:

- `map<string,string> english2danish`: A mapping from strings (representing an english word) into strings (representing its direct translation into danish).
- `map<string,string> danish2english`: A mapping from strings (representing a danish word) into strings (representing its direct translation into english).
- `map<string,set<string> > english_synonyms`: A mapping from strings (representing an english word) into sets of strings (representing a set of synonyms).
- `map<string,set<string> > danish_synonyms`: A mapping from strings (representing a danish word) into sets of strings (representing a set of synonyms).

Help Claire by performing the following tasks:

(a) Check the implementation of method `void insert_words(string u, string v)` and correct it if necessary. The method should record that the english word `u` and the danish word `v` are the direct translations of each other, provided that neither `u` nor `v` have direct translations already.

(b) Check the implementation of method `string get_word(string lang, string u)` and correct it if necessary. The method should return the direct translation of the word `u` in language `lang` (either `"english"` or `"danish"`), or the string `"#unknown#"` if there is no direct translation.

(c) Implement method `void insert_synonym(string lang, string u, string v)`. This method inserts a synonym `v` and `u` as synonyms in language `lang`. If `lang` is neither `"english"` nor `"danish"` the method should do nothing.

(d) Implement method `set<string> get_synonyms(string lang, string u)`. Given a language `lang` and a word `u` the method should return the set of synonyms of `u`.

Exercise follows in next page...

(e) Implement method `set<string> translate(string lang, string u)`. Given a language `lang` and a word `u` the method should return the set of possible translations of `u`. These should include not only the direct translation of `u` but also the synonyms of the direct translation of `u`. For example, in the test in file `ex03-main.cpp`, Claire expects to get the output `car` (because it is the direct translation of "bil") and `auto` (because it is a synonym of "car").

Hints about using maps:

- A key `k` in a map `m` can be updated (mapped) to `v` with `m[k] = v`;
- The value mapped to a key `k` in a map `m` is obtained with `m[k]`;
- The above two methods create the entry for the key if it is not present in the map. To check if the key is present you can use the test `m.find(k) != m.end()`.

File ex03-main.cpp

```
#include <iostream>
#include <string>
#include <set>
#include "ex03-library.h"

using namespace std;

int main(void){
    Dictionary d;
    set<string> s;

    d.insert_words("car","bil");
    cout << d.get_word("english","car") << endl;
    cout << d.get_word("danish","bil") << endl ;

    d.insert_synonym("english","car","auto");

    s = d.get_synonyms("english","auto");
    for(set<string>::iterator it = s.begin();
        it != s.end(); it++)
        cout << *it << "␣";
    cout << endl;

    s = d.translate("danish","bil");
    for(set<string>::iterator it = s.begin();
        it != s.end(); it++)
        cout << *it << "␣";
    cout << endl;

    return 0;
}
```

File ex03-library.h

```
#ifndef __ex03_library__
#define __ex03_library__

#include <map>
#include <set>
#include <string>

using namespace std;

class Dictionary {

public:
    void insert_words(string u, string v);
    string get_word(string lang, string u);
    void insert_synonym(string lang, string u, string v);
    set<string> get_synonyms(string lang, string u);
    set<string> translate(string lang, string u);

private:
    map<string,string> english2danish;
    map<string,string> danish2english;
    map<string,set<string> > english_synonyms;
    map<string,set<string> > danish_synonyms;

};

#endif
```

File ex03-library.cpp

```
#include <iostream>
#include <map>
#include <set>
#include <vector>
#include "ex03-library.h"

using namespace std;

// Exercise 3(b)
// Check and correct if necessary
void Dictionary::insert_words(string u, string v){
    english2danish[u] = v;
    danish2english[u] = v;
}

// Exercise 3(b)
// Check and correct if necessary
string Dictionary::get_word(string lang, string u){
    return english2danish[u];
}

// Exercise 3(c)
void Dictionary::insert_synonym(string lang, string u,
                                string v){
    // Put your code here
}

// Exercise 3(d)
set<string> Dictionary::get_synonyms(string lang, string u){
    // Put your code here
}

// Exercise 3(e)
set<string> Dictionary::translate(string lang, string u){
    // Put your code here
}
```

Exercise 4. Fun with Monoids (2.5 points)

A monoid expression is either a constant or the composition of two monoid expressions. Hugo is a fan of monoid expressions and their many applications and is implementing a C++ library for supporting them. He has prepared a test program in file `ex04-main.cpp`, the declaration of the class `Monoid` for monoid expressions in file `ex04-library.h` and a sketch of its implementation in file `ex04-library.cpp`. All files are available on CampusNet and in the next pages. Hugo has decided to use the following internal (`private`) representation for monoid expressions:

- `C constant`: A value of class `C` to store constant value expressions. If the expression is the binary composition of two expressions then the value of `constant` is irrelevant.
- `Monoid * m1`: A pointer to a monoid. If the monoid expression is a constant, `m1` is a `nullptr`, otherwise (the monoid expression is a binary composition) it points to the left operand.
- `Monoid * m2`: A pointer to a monoid. If the monoid expression is a constant, `m2` is a `nullptr`, otherwise it points to the right operand.

Hugo has already implemented several functions, including constructors, an operator `*` to compose monoids, and an assignment operator `=`. Help Hugo finish the implementation of class `Monoid` in file `ex04-library.cpp`. Your tasks are:

(a) Check the class destructor and correct it if necessary.

(b) Implement method `int constants(void)`. This method should return the number of constants in a monoid expression. In the example of the test program, the expected result is 3.

(c) Implement the method `void commute(void)`. If invoked for a monoid expression with two operands, it should reverse the order of the operands. For example, an expression $x * y$ should become $y * x$.

(d) Implement the method `void associate_left(void)`. If invoked for a monoid expression with two operands, such that the right operand has at least two operands, the expression should be re-arranged so that the operands are associated to the left. For example, an expression $x * (y * z)$ should become $(x * y) * z$.

(e) Implement the method `void associate_right(void)` (symmetric case for task (d)).

Exercise follows in next page...

File ex04-main.cpp

```
#include <iostream>
#include <string>
#include "ex04-library.h"

using namespace std;

int add(int u, int v){ return u + v; }
int substract(int u, int v){ return u - v; }

int main(void){

    Monoid<int> u(1);
    Monoid<int> v(2);
    Monoid<int> w(3);

    Monoid<int> x((u * v) * w);
    x.print(); cout << endl;
    cout << x.constants() << endl;

    x.commute();
    x.print(); cout << endl;

    x.associate_left();
    x.print(); cout << endl;

    x.associate_right();
    x.print(); cout << endl;


    return 0;
}
```

File ex04-library.h

```
#ifndef __ex04_library__
#define __ex04_library__

#include <string>
using namespace std;

template <class C>
class Monoid {
public:
    Monoid(C constant);
    Monoid(Monoid<C> & m1);
    Monoid(Monoid<C> & m1, Monoid<C> & m2);
    Monoid<C> & operator=(Monoid<C> & m);
    Monoid<C> & operator*(Monoid<C> & m);
    ~Monoid(void);
    C eval(C (*f)(C,C));
    int constants(void);
    void commute(void);
    void associate_left(void);
    void associate_right(void);
    void print(void);
private:
    C constant;
    Monoid<C> * m1;
    Monoid<C> * m2;
};

#endif
```

File ex04-library.cpp

```
#include <iostream>
#include "ex04-library.h"

using namespace std;

// Exercise 4(a)
template <class C>
Monoid<C>::~Monoid(void){
    if(m1 != nullptr) delete m1;
    if(m1 != nullptr) delete m2;
}

// Exercise 4(b)
template <class C>
int Monoid<C>::constants(void){
    // Put your code here
}

// Exercise 4(c)
template <class C>
void Monoid<C>::commute(void){
    // Put your code here
}

// Exercise 4(d)
template <class C>
void Monoid<C>::associate_left(void){
    // Put your code here
}

// Exercise 4(e)
template <class C>
void Monoid<C>::associate_right(void){
    // Put your code here
}

// Some code follows...
// You don't need to understand it
// but it could give you some hints
```