Technical University of Denmark

Written examination, May 17, 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. Each assignment 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 the `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 further tests may be run on your solutions, also 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 she needs 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 `n`×`n` square matrices with single index arrays of size $n \times n$ (as we did in several examples during the course).

(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 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 can be found above, below, leftwards or rightwards. Let $a_{i,j}$ denote 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
```

(d) 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 last matrix above, the vector would be $(7, 10, 13)$.

(e) 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 last matrix above, the vector would be $(9, 10, 11)$.

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*n];

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

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

    // Finally, I am summing up rows and values
    vector<double> v;
    v = sumRows(A,n);
    print(v);
    v = sumCols(A,n);
    print(v);

    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 = 1; i <= n; i++){
        for(unsigned int j = 1; j <= n; j++){
            cout << A[i*n+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
void reduce(double * A, unsigned int n){

    // Put your code here

}

// Exercise 1 (d)
// Implement this function
vector<double> sumRows(double * A, unsigned int n){

    // Put your code here

}

// Exercise 1 (e)
// Implement this function
vector<double> sumCols(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

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

where

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

As an example you can easily check that matching the sequences A, B, C and A, C yields

$$\begin{array}{lll} & match((A, B, C), (A, C)) & \\ = & A, match((B, C), (C)) & \text{since } h(A, B, C) = h(A, C) = A \\ = & A, match((C), (C)) & \text{since } |match((C), (C))| \geq |match((B, C), (\epsilon))| \\ = & A, C, match((\epsilon), (\epsilon)) & \text{since } h(C) = h(C) = C \\ = & A, C & \text{since } match(\epsilon, \epsilon) = \epsilon \text{ and } u, \epsilon = u \text{ for all sequences } u \end{array}$$

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:

- `vector<map<string,string> > words`: A vector of mappings from strings (representing a word) into strings (representing its translation). The idea is that the vector will have size 2: one for each possible direction of the translation. The first language of the dictionary will be denoted and indexed with `0` and the second one by `1`. So the idea is that if Claire introduces the pair of English/Danish words ("car","bil") these will be stored as follows: `words[0]["car"]` will be mapped to `"bil"` and `words[1]["bil"]` will be mapped to `"car"`.

- `vector<map<string,set<string> > > synonyms`: A vector of mappings from strings (representing a word) into sets of strings (representing a set of synonyms). Again, the idea is that the vector will have size 2: one for each possible language. For example, if Claire introduces the synonym "auto" for "car" then `synonyms[0]["car"]` will be a set containing `"auto"` (and possibly other synonyms introduced before).

Help Claire by performing the following tasks:

(a) Implement method `Dictionary(void)`. This is the constructor of the dictionary.

(b) Check the implementation of methods `void insert_words(string u, string v)` and `string get_word(int lang, string u)` and correct them if necessary. The first method inserts a word `u` from the first language and its translation `v` in the second language in the dictionary. The second method should return the direct translation of the `lang` word `u`, or the empty string if there is no direct translation. Hint: see how Claire uses these two functions in her test program.

(c) Implement method `void insert_synonym(int lang, string u, string v)`. This method inserts a synonym `v` for word `u` of language `lang`.

(d) Implement method `set<string> get_synonyms(int 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(int 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 pair of words introduced with `insert_words` but also those words obtained through synonyms, in particular the method should return the translation of `u` and of the synonyms of `u`, and their respective synonyms. For example, in the test in file `ex03-main.cpp`, Claire expects to get the output `car` (because the direct translation of "bil" is in the dictionary), `auto` (because it is a synonym of "car") , `wagon` (because it is the direct translation of "vogn", which is a synonym of "bil") and `van` (because it is a synonym of "wagon").

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;

    d.insert_words("car","bil");
    cout << d.get_word(0,"car") << endl;
    cout << d.get_word(1,"bil") << endl ;

    d.insert_words("wagon","vogn");
    d.insert_synonym(0,"car","auto");
    d.insert_synonym(1,"bil","vogn");
    d.insert_synonym(0,"wagon","van");

    set<string> s = d.translate(1,"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 <vector>
#include <string>

using namespace std;

class Dictionary {

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

private:
    vector<map<string,string> > words;
    vector<map<string,set<string> > > synonyms;

};

#endif
```

File `ex03-library.cpp`

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

using namespace std;

// Exercise 3(a)
Dictionary::Dictionary(void){
    // Put your code here
}

// Exercise 3(b)
// Check and correct if necessary
void Dictionary::insert_words(string u, string v){
    words[0][u] = v;
}

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

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

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

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

Exercise 4. Fun with Monoids (2.5 points)

A monoid is an algebraic structure with a single associative binary operation and an identity element. Typical examples are sequences with concatenation as binary operation and the empty string as identity element, or natural numbers with addition as binary operation and 0 as identity element. Hugo is a fan of monoids and their many applications and wants to implement a C++ library for supporting monoid expressions. A monoid expression can be either a constant or the composition of two monoid expressions. Hugo 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 is unsure about the implementation but he 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.

Help Hugo implementing the class `Monoid` in file `ex04-library.cpp`. Your tasks are:

(a) Implement the constructor `Monoid(C constant)`. This is a constructor of constant monoid expressions. The constructed monoid expression should contain the value `constant`. See for example how Hugo builds the monoid expression for constant `"Hello"` with `Monoid<string> a("Hello")`.

(b) Implement the constructor `Monoid(Monoid<C> & m1)`. This is a *copy* constructor. It builds a monoid expression which is a new copy of `m1`.

(c) Implement the constructor `Monoid(Monoid<C> & m1, Monoid<C> & m2)`. This constructor takes two monoid expressions as arguments and builds a new expression (representing a binary composition) with new copies of `m1` and `m2` as left and right arguments, respectively. See for example how Hugo builds the monoid expression that results from combining monoid expressions `a` and `b` with `Monoid<string> d(a,b)`.

(d) Implement method `Monoid<C> operator*(Monoid<C> & m)`. This is a binary operation. It takes two monoid expressions (the current object and `m`) and returns a new monoid expression with a new copy of the the current object as left operand and a new copy of `m` as right operand. Note the similarity with the binary constructor. Indeed, `Monoid<string> d(a,b)` and `a * b` build identical monoid expressions.

Exercise follows in next page...

(e) Implement method `C eval(C (*f)(C,C))`. This method takes as parameter a function `f` (that returns a value of type `C` when applied to two arguments of type `C`), and should return the result of evaluating the monoid expression using `f` as interpretation of the monoid operator. That is, if the monoid expression is a constant it should just return the value of the constant, otherwise it should return the result of applying `f` to the evaluation of the two operands (`m1`, `m2`) of the monoid expression.

For example, in Hugo's test program, he expects to get the following output (as the result of invoking `eval`)

```
Hello world!
Hello world!
6
```

Note: we have not seen how to pass functions as parameters in the course, but all you need here is to use `f` within the method just as it would be a normal function (i.e. one not passed as argument), e.g. `f(x,y)` applies `f` with arguments `x` and `y`.

Exercise follows in next page...

File ex04-main.cpp

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

using namespace std;

string concat(string u, string v){
    return u + v;
}

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

int main(void){
    Monoid<string> a("Hello");
    Monoid<string> b("␣");
    Monoid<string> c("world!");

    Monoid<string> d(a,b);
    Monoid<string> e(d,c);

    cout << e.eval(concat) << endl;
    cout << (a * b * c).eval(concat) << endl;

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

    cout << (u * v * w).eval(add) << 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<C> operator*(Monoid<C> & m);
    Monoid(Monoid<C> & m1, Monoid<C> & m2);
    C eval(C (*f)(C,C));
    void print(void);

private:
    Monoid(Monoid<C> & m1);
    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(C constant){
    // Put your code here
}

// Exercise 4 (b)
template <class C>
Monoid<C>::Monoid(Monoid<C> & m1, Monoid<C> & m2){
    // Put your code here
}

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

// Exercise 4 (d)
template <class C>
Monoid<C> Monoid<C>::operator*(Monoid<C> & m){
    // Put your code here
}

// Exercise 4 (e)
template <class C>
C Monoid<C>::eval(C (*f)(C,C)){
    // Put your code here
}

// Do not modify
template <class C>
void Monoid<C>::print(void){
    if (m1 == nullptr) cout << constant ;
    else {
        m1->print();
        cout << "␣*␣" ;
        m2->print();
    }
}

// Do not modify
template class Monoid<string>;
template class Monoid<unsigned int>;
```