Technical University of Denmark

Written examination, May 14, 2018

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 additionally upload your solutions on CodeJudge under `ReExam May 2018` at `https://dtu.codejudge.net/02393-e17/exercisegroup`. This is not an official submission, and will not be considered for evaluation. When you hand in a solution on CodeJudge, some tests will be run on it. Additional tests may be run on your solutions after the exam. You can upload to CodeJudge as many times as you like.

Exercise 1. Graphics Editor (2.5 points)

Alice needs to implement a library to manipulate black/white pictures, stored as matrices with one entry per pixel. Each entry takes integer values from 0 to 255:

- 0 corresponds to black,
- 255 corresponds to white
- intermediate values correspond to tonalities of grey.

Alice needs to perform some computations on such 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 (with additional comments) and in the next pages. Help Alice by solving the following tasks:

(a) Check the implementation of function

```
unsigned int ** createMatrix(int n, int m);
```

and correct it if necessary. The function should correctly create an `n` × `m` matrix of `unsigned int` elements, and return it. The function should only allocate the required memory. Initialization of the entries of the matrix is not required.

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

(b) Check the implementation of function

```
void deallocateMatrix(unsigned int ** A, int n);
```

This function is the dual of `createMatrix`. It should deallocate all memory allocated for the matrix `A`.

(c) Implement function

```
void fillWhite(unsigned int ** A, int n, int m);
```

The function should set to white (i.e., to value 255) all entries of the `n` × `m` matrix `A`.

Exercise follows in next page...

(d) Implement function

```
void lighten(unsigned int ** A, int n, int m);
```

This filter makes the picture brighter: each pixel's value gets incremented by 1 (preserving the range 0-255). This function should scan all elements of the matrix `A` given as parameter, and

- increase by one all values smaller than 255.
- leave unchanged all values equal to 255.

(e) Implement function

```
void darken(unsigned int ** A, int n, int m);
```

This filter makes the picture darker: each pixel's value gets decremented by 1 (preserving the range 0-255). This function should scan all elements of the matrix `A` given as parameter, and

- decrease by one all values greater than 0.
- leave unchanged all values equal to 0.

Exercise follows in next page...

File ex01-main.cpp

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

using namespace std;

int main(void){

  // Matrix representing a picture
  int n = 2;
  int m = 3;
  unsigned int **A = createMatrix(n,m);

  // Setting all values to 255 (white)
  fillWhite(A,n,m);
  // Setting some values in the matrix
  for(int i=0;i<n-1;i++){
    for(int j=0;j<m;j++){
      A[i][j]=i*n+j;
    }
  }

  // Printing the matrix
  printMatrix(A,n,m,"original␣matrix");
  cout << endl;

  //Making the figure brighter
  for(int i=1;i<3;i++){
    lighten(A,n,m);
    printMatrix(A,n,m,"brighter␣matrix␣("
      + to_string(i)+")");
    cout << endl;
  }

  //Making the figure darker
  for(int i=1;i<4;i++){
    darken(A,n,m);
    printMatrix(A,n,m,"darker␣matrix␣("
      + to_string(i)+")");
    cout << endl;
  }

  //Deallocating A
  deallocateMatrix(A,n);

  return 0;
}
```

File ex01-library.h

```
#ifndef EX01_LIBRARY_H_
#define EX01_LIBRARY_H_

#include <vector>
#include <string>
using namespace std;

unsigned int ** createMatrix(int n, int m);
void deallocateMatrix(unsigned int ** A, int n);
void fillWhite(unsigned int ** A, int n, int m);
void lighten(unsigned int ** A, int n, int m);
void darken(unsigned int ** A, int n, int m);
void printMatrix(unsigned int ** A, int n, int m,
                     string description);

#endif /* EX01_LIBRARY_H_ */
```

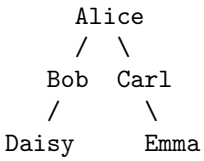
File ex01-library.cpp

```
#include <iostream>
#include <vector>
#include <iomanip>
#include "ex01-library.h"
using namespace std;

//Exercise 1 (a) Check and correct if necessary
unsigned int ** createMatrix(int n, int m){
  unsigned int ** A = new unsigned int *[n];
  for(int j = 1; j<n-1; j++){
    A[j] = new unsigned int[m];
  }
  return A;
}
void deallocateMatrix(unsigned int ** A, int n){
  delete[] A;
}
//Exercise 1 (c) Implement this function
void fillWhite(unsigned int ** A, int n, int m){
  //Put your code here
}
//Exercise 1 (d) Implement this function
void lighten(unsigned int ** A, int n, int m){
  //Put your code here
}
//Exercise 1 (e) Implement this function
void darken(unsigned int ** A, int n, int m){
  //Put your code here
}
//Do not modify
void printMatrix(unsigned int ** A, int n, int m,
                     string description){
  cout<< "Printing:␣" << description << endl;
  for(int i = 0; i < n; i++){
    for(int j = 0; j < m; j++){
      cout << setw(5) << A[i][j] << "␣";
    }
    cout << endl;
  }
}
```

Exercise 2. Genealogy (2.5 points)

Bob is interested in studying the geneaology of its family. So far he has built the following family tree:



Alice had two sons: Bob and Carl, which in turn had a daughter each, Daisy and Emma, respectively. Daisy and Emma currently have no descendants.

Bob would like to automate the computation of simple queries on family trees, like the computation of the number of *descendants* of an individual, or the computation of all individuals that had no children (i.e., that are *leaves* in the tree). 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 (with additional comments) and in the next pages.

As you can see Bob has decided to represent family trees using binary trees, where each node can have up to two children. Each node of the tree is stored in a struct `Node` containing:

- `name`: the name of the corresponding individual;
- `left` and `right`: pointers to other `Node` structures, representing the children of the corresponding individual. No particular order is assumed among the two pointers.

Note that such representation allows to represent families in which each individual had at most two children. In case the individual had no children, then both pointers have value `nullptr`. In case the individual had one children, then either of the two pointers will refer to the `Node` representing the child, while the other pointer will have value `nullptr`.

Help Bob by solving the following taks:

(a) Check the implementation of function

```
void computeLeaves(Node *n, set<string> & leaves);
```

and correct it if necessary. A *leaf node* is a node without children. This function should correctly compute all *leaf nodes* met while navigating the tree starting from node `n` (including `n` itself, in case it has no children). In particular, the function should add the name of the current node `n` to the set `leaves` if `n` has no children. Otherwise, the function should be recursively invoked for the children of `n`. For instance, the leaf nodes found starting from Alice's node are: Daisy and Emma. Instead, the only leaf node met starting from Bob's node is Daisy.

Exercise follows in next page...

(b) Implement function

```
int countDescendants(Node * n);
```

This function should count the number of descendants of node n. For instance, the number of descendants of Alice is 4 (Bob, Carl, Daisy and Emma). Instead, Bob has just one descendant (Daisy), while Daisy has no descendants.

Exercise follows in next page...

File ex02-main.cpp

```
#include <iostream>
#include "ex02-library.h"
using namespace std;

void experimentLeafNodes(Node * n){
  set<string> leaves;
  computeLeaves(n,leaves);
  cout<<"Leaf␣nodes␣starting␣from␣"
      << n->name << ":\n";
  printSet(leaves);
}

void experimentDescendants(Node * n){
  int descendants = countDescendants(n);
  cout<< n->name << "'s␣descendants:"
      << descendants << "\n";
}

int main() {
  /* Bob's family tree from text */
  Node *Alice = new Node;
  Alice->name="Alice";
  Node *Bob = new Node;
  Bob->name="Bob";
  Node *Carl = new Node;
  Carl->name="Carl";
  Node *Daisy = new Node;
  Daisy->name="Daisy";
  Node *Emma = new Node;
  Emma->name="Emma";
  Alice->left=Bob;
  Alice->right=Carl;
  Bob->left=Daisy;
  Bob->right=nullptr;
  Carl->right=Emma;
  Carl->left=nullptr;
  Daisy->left=nullptr;
  Daisy->right=nullptr;
  Emma->left=nullptr;
  Emma->right=nullptr;

  cout << "Experiments␣about␣leaf␣nodes\n";
  experimentLeafNodes(Alice);
  experimentLeafNodes(Bob);
  experimentLeafNodes(Carl);
  experimentLeafNodes(Daisy);
  experimentLeafNodes(Emma);

  cout << "Experiments␣about␣descendants\n";
  experimentDescendants(Alice);
  experimentDescendants(Bob);
  experimentDescendants(Carl);
  experimentDescendants(Daisy);
  experimentDescendants(Emma);
  return 0;
}
```

File ex02-library.h

```
#ifndef EX02_LIBRARY_H_
#define EX02_LIBRARY_H_

#include <set>
#include <string>
using namespace std;

struct Node{
  std::string name;
  Node * left;
  Node * right;
};

void printSet(set<string> s);
void computeLeaves(Node *n, set<string> & leaves);
int countDescendants(Node * n);
#endif
```

File ex02-library.cpp

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

//Do not modify
void printSet(set<string> s){
  if(s.size()==0){
    cout << "␣No␣nodes\n";
  }
  else{
    set<string>::iterator it;
    for (it=s.begin(); it!=s.end(); ++it){
      cout << '␣' << *it << "\n";
    }
  }
  cout << "\n";
}

//Exercise 2 (a) Check and correct if necessary
void computeLeaves(Node *n, set<string> & leaves){
  if(n->left == nullptr && n->right == nullptr){
    leaves.insert(n->name);
  }
  else{
    computeLeaves(n->left,leaves);
    computeLeaves(n->right,leaves);
  }
}

//Exercise 2 (b) Implement this function
int countDescendants(Node * n){
  //Put your code here
}
```

Exercise 3. Movie Evaluations (2.5 points)

Claire wants to implement a class `MovieEvaluations` to store her evaluations of movies. Her first test program is in file `ex03-main.cpp` and the (incomplete) code with some functions she needs is in files `ex03-library.h` and `ex03-library.cpp`. All files are available on CampusNet (with extra comments) and in the next pages. Help Claire by implementing the class `MovieEvaluations` 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:

- `set<string> movies`: The set of movies for which she currently has evaluations. We store the title of the movie.
- `map<string,double> movieToEvaluation`: A mapping from strings (the title of the movie) into doubles (the evaluation assigned to the movie). In particular, evaluations are doubles belonging to interval $[0, 10]$, that is:

  $$0 \leq \text{ evaluation } \leq 10$$

  Where 0 is the worst evaluation, and 10 is the best one.

Claire already implemented the default constructor of `MovieEvaluations`. Such constructor initializes the object with one movie, *The Godfather*, with evaluation 10. Claire believes that everyone watched such movie, and that everyone loves it. Claire also provided a method

```
bool hasEvaluation(string movie);
```

which returns true if the movie has an evaluation, and false otherwise. Help Claire by performing the following tasks:

(a) Check the implementation of method

```
void print();
```

and correct it if necessary. The method should correctly print information on the provided evaluations. For each evaluation, it should print a line (starting with an empty space) of this form:

```
 movie title has evaluation movieEval
```

where `title` and `movieEval` are, respectively, the title and the evaluation of the considered movie.

Exercise follows in next page...

(b) Implement method

```
bool addEvaluation(string movie,double evaluation);
```

The method should add an evaluation for a new movie, and return true. Special cases:

- If the movie has already an evaluation, ignore the request, and return false.
- If the evaluation is smaller than 0 or greater than 10, ignore the request, and return false.

(c) Implement method

```
bool updateEvaluation(string movie, double newEvaluation);
```

This method should update an existing evaluation of a movie, and return true if none of the following special cases apply:

- If the movie does not have evaluations yet, ignore the request, and return false.
- If the new evaluation is smaller than 0, ignore the request, and return false.
- If the new evaluation is greater than 10, ignore the request, and return false.

(d) Implement method

```
string computeVerboseEvaluation(string movie);
```

This method should return a sentence describing the evaluation assigned to the movie. In particular, it should return the following strings, depending on the evaluation:

"very bad"	if	$0.0 \leq$ evaluation ≤ 2.5
"bad"	if	$2.5 <$ evaluation ≤ 5.0
"good"	if	$5.0 <$ evaluation ≤ 7.5
"very good"	if	$7.5 <$ evaluation ≤ 10.0
"not evaluated"	if	the movie does not have an evaluation

Exercise follows in next page...

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 operator `m[k]` creates an entry for the key `k` if it is not present in the map `m`. To check if the key is present you can use the test `m.find(k) != m.end()`.

Exercise follows in next page...

File `ex03-main.cpp`

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

int main() {
  MovieEvaluations me;

  cout << "\nExecuting␣addEvaluation(\"Alien\",-2.5);\n";
  me.addEvaluation("Alien",-2.5);
  me.print();

  cout << "\nExecuting␣addEvaluation(\"Alien\",12.5);\n";
  me.addEvaluation("Alien",12.5);
  me.print();

  cout << "\nExecuting␣addEvaluation(\"Alien\",7.6);\n";
  me.addEvaluation("Alien",7.6);
  me.print();

  cout << "\nExecuting␣addEvaluation(\"Alien\",6.6);\n";
  me.addEvaluation("Alien",6.6);
  me.print();

  cout << "\nExecuting␣updateEvaluation(\"Alien\",6.6);\n";
  me.updateEvaluation("Alien",6.6);
  me.print();

  cout << "\nExecuting␣updateEvaluation(\"The␣Wizard␣of␣Oz\",8);\n";
  me.addEvaluation("The␣Wizard␣of␣Oz",8);
  me.print();

  cout <<"\n\n";
  cout << "The␣Godfather␣is␣" << me.computeVerboseEvaluation("The␣Godfather") << "\n";
  cout << "Alien␣is␣" << me.computeVerboseEvaluation("Alien") << "\n";
  cout << "The␣Wizard␣of␣Oz␣is␣" << me.computeVerboseEvaluation("The␣Wizard␣of␣Oz") << "\n";
  cout << "Pulp␣Fiction␣is␣" << me.computeVerboseEvaluation("Pulp␣Fiction") << "\n";
  return 0;
}
```

File `ex03-library.h`

```
#ifndef EX03_LIBRARY_H_
#define EX03_LIBRARY_H_

#include <string>
#include <map>
#include <set>
using namespace std;

class MovieEvaluations {
private:
  set<string> movies;
  map<string,double> movieToEvaluation;
public:
  MovieEvaluations();
  bool hasEvaluation(string movie);
  void print();
  bool addEvaluation(string movie, double evaluation);
  bool updateEvaluation(string movie, double newEvaluation);
  string computeVerboseEvaluation(string movie);
};

#endif /* EX03_LIBRARY_H_ */
```

Exercise follows in next page...

File `ex03-library.cpp`

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

//Do not modify
MovieEvaluations::MovieEvaluations() {
  movies.insert("The␣Godfather");
  movieToEvaluation["The␣Godfather"]=10;
}

//Do not modify
bool MovieEvaluations::hasEvaluation(string movie){
  if(movies.find(movie) != movies.end()){
    //I have an evaluation for the movie
    return true;
  }
  else{
    return false;
  }
}

//Exercise 3 (a) Check and correct if necessary
void MovieEvaluations::print(){
  cout << "I␣have␣the␣following␣evaluations:"<<endl;
  for (map<string,double>::iterator it=movieToEvaluation.begin(); it!=movieToEvaluation.end(); ++it){
      cout << '␣'<< "movie␣" << it->second << "␣has␣evaluation␣" << it->first << endl;
  }
}

//Exercise 3 (b) Implement this function
bool MovieEvaluations::addEvaluation(string movie,double evaluation) {
  //Put your code here
}

//Exercise 3 (c) Implement this function
bool MovieEvaluations::updateEvaluation(string movie,double newEvaluation) {
  //Put your code here
}

//Exercise 3 (d) Implement this function
string MovieEvaluations::computeVerboseEvaluation(string movie) {
  //Put your code here
}
```

Exercise 4. Parametric Stack (2.5 points)

Daisy attended the course 02393 Programming in C++. Taking inspiration from some of the examples seen in class, she decided to implement a stack, that is a list where elements are retrieved in the reverse order with respect to how they have been inserted. This is obtained by inserting new elements in front of the stack, that is on its *top*. Note that this is the opposite of method push_back of vectors, where elements are inserted at the end.

Daisy is implementing a C++ library for manipulating parametric stacks. She has prepared a test program (`ex04-main.cpp`), the declaration of class `mystack` (`ex04-library.h`) and a sketch of its implementation (`ex04-library.cpp`). All files are available on CampusNet (with further comments) and in the next pages. Daisy has decided to represent an entry of the stack using the parametric struct `Node` containing two fields:

- `T content`: the element of (parametric) type `T` in that position of the stack.
- `Node<T> * next`: pointer to the `Node` storing the next element of the stack.

Daisy has decided to use the following internal[1] representation for stacks:

- `Node<T> * top`: pointer to the first element of the stack.
- `int size`: the current number of elements stored in the stack.

If the stack is empty, then `top` has value `nullptr`. If the stack has size one, then `top` points to the only `Node` in the stack, whose `next` pointer has value `nullptr`. If the stack has size greater than one, then `top` points to the `Node` representing the last element added to the stack, while its `next` pointer points to the `Node` representing second-last element added to the stack, and so on.

Daisy has already implemented several functions, including:

- a constructor and a destructor.
- a public method to print information on the list (`print()`).

Help Daisy implementing class `mystack` in file `ex04-library.cpp`. Your tasks are:

(a) Check the class destructor and correct it if necessary. It should deallocate all memory used by each of its `Node` structs. The method terminates printing using the command:

```
cout << "Destructor completed\n";
```

Exercise follows in next page...

[1] These two fields should be private. However, in order to obtain better tests, these fields have been made public.

(b) Implement the method `print_top(T v)`. This method should print the top element of the stack, if any, without modifying the stack, The method should print just the content of the `Node` pointed by `top`, without any additional text. Special cases:

- If the method is invoked on an empty stack, then it should print using command:

  ```
  cout << "The stack is empty.";
  ```

(c) Implement the method `push(T v)`. This method should add an element `v` at the top of the stack. Hence `v` becomes the top element of the stack, while the old top element, if present, should be pointed by the `next` pointer of the newly added element.

(d) Implement method `pop()`. This method should remove the top element in the stack, effectively reducing the stack size by one. The memory allocated to store the removed element should be deallocated. The method should return true if the stack contained at least an element, meaning that the pop succeeded. Instead, it should return false it the stack was empty, and hence the pop failed. Special cases:

- If the method is invoked on an empty stack, nothing is done, and the value false is returned.
- If the method is invoked on a stack with one element, the stack becomes empty, and the value true is returned.
- If the method is invoked on a stack with more than one element, the `Node` pointed by the pointer `next` of `top` becomes the new `top`.

Exercise follows in next page...

File `ex04-main.cpp`

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

using namespace std;

int main() {
  mystack<int> stack;

  stack.push(1);
  stack.push(2);
  stack.push(3);

  stack.print();

  cout <<"\n";
  cout << "The␣element␣on␣the␣top␣is:␣";
  stack.print_top();
  cout <<"\n";

  do{
    cout <<"\n";
    cout << "pop():␣";
    if(stack.pop()){
      cout <<"succeeded\n";
    }
    else{
      cout <<"failed\n";
    }

    stack.print();
  }while(stack.getSize()>0);

  cout <<"\n";
  cout << "pop():␣";
  if(stack.pop()){
    cout <<"succeeded\n";
  }
  else{
    cout <<"failed\n";
  }

  return 0;
}
```

Exercise follows in next page...

File `ex04-library.h`

```
#ifndef EX04_LIBRARY_H_
#define EX04_LIBRARY_H_

template<class T>
struct Node{
  T content;
  Node<T> * next;
};

template <class T>
class mystack {
public:
  Node<T> * top;
  int size;
  mystack();
  ~mystack();
  int getSize();
  void print_top();
  void print();
  void push(T v);
  bool pop();
};
#endif
```

File `ex04-library.cpp`

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

using namespace std;

//Do not modify
template<class T>
mystack<T>::mystack() {
  size=0;
  top=nullptr;
}

//Do not modify
template<class T>
int mystack<T>::getSize() {
  return size;
}

//Do not modify
template<class T>
void mystack<T>::print() {
  if(size==0){
    cout << "The␣stack␣is␣empty.\n";
  }
  else{
    cout << "The␣stack␣has␣size␣" << size << ":\n";
    Node<T> * current = top;
    while(current!=nullptr){
      cout << "␣␣" <<current->content << "\n";
      current = current->next;
    }
  }
}

//Exercise 4 (a) Check and correct if necessary
template<class T>
mystack<T>::~mystack() {
  Node<T> * current = top;
  while(current!=nullptr){
    delete current;
    Node<T> * next = current->next;
    current = next;
  }
  cout << "Destructor␣completed\n";
}

//Exercise 4 (b) Implement this function
template<class T>
void mystack<T>::print_top() {
  //Put your code here
}

//Exercise 4 (b) Implement this function
template<class T>
void mystack<T>::push(T v) {
  //Put your code here
}

//Exercise 4 (d) Implement this function
template<class T>
bool mystack<T>::pop() {
  //Put your code here
}

//Do not modify
template class mystack<int>;
```