Technical University of Denmark

Written examination, December 10, 2017

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 on CodeJudge under `Exam December 2017` at `https://dtu.codejudge.net/02393-e17/exercisegroup`. This is not an official submission, and will no 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, usually stored as matrices with one entry per pixel. Each entry takes integer values from 0 to 255: 0 corresponds to black, 255 to white, and intermediate values 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

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

and correct it if necessary. The function should correctly create an `n` × `m` matrix, 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`). 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) Implement function

```
int ** duplicateMatrix(int ** A, unsigned int n, unsigned int m);
```

The function should create and return a copy of the `n` × `m` matrix `A`.

(c) Implement function

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

The function should set to 0 all entries of the `n` × `m` matrix `A`.

(d) Implement function

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

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

Exercise follows in next page...

(e) Implement function

```
int ** makeBitonal(int ** A, unsigned int n, unsigned int m, int threshold);
```

This function makes a picture *bi-tonal* according to a given *threshold*: each pixel gets either black (0) or white (255). The function takes as input an `n` × `m` matrix `A`, and a threshold. This function should scan all elements of `A`, and replace all values

- below `threshold` with value 0.
- equal to `threshold` with value 255.
- above `threshold` with value 255.

Hence, this filter sets to white all grey entries clearer than the given threshold, and to black all the ones darker than the given threshold.

Notes:

- The function has to apply the filter on a copy, let's say `B`, of `A`. Such modified copy `B` has to be returned.
- The function must not modify `A`.

For example, for threshold 100:

- all entries with value smaller than 100 get value 0.
- all entries with value equal to 100 get value 255.
- all entries with value greater than 100 get value 255.

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
  unsigned int n = 4;
  unsigned int m = 4;
  int **A = createMatrix(n,m);

  // Setting all values to 0
  initMatrix(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;
    }
  }
  A[1][0]=126;
  A[1][1]=127;
  A[1][2]=128;
  printMatrix(A,n,m,"main␣matrix");
  cout << endl;

  int **B = duplicateMatrix(A,n,m);
  printMatrix(B,n,m,"copy");

  //I change B, and I print B and A
  for(int j=0;j<m;j++){
    B[0][j]=B[0][j]+3;
  }

  printMatrix(B,n,m,"modified␣copy");
  printMatrix(A,n,m,"main␣matrix");

  //I deallocate B
  deallocateMatrix(B,n);

  int ** C = makeBitonal(A,n,m,127);
  printMatrix(C,n,m,"bi-tonal␣copy");

  deallocateMatrix(A,n);
  deallocateMatrix(C,n);
  return 0;
}
```

File `ex01-library.h`

```
#ifndef EX01_LIBRARY_H_
#define EX01_LIBRARY_H_

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

int ** createMatrix(unsigned int n, unsigned int m);
int ** duplicateMatrix(int ** A, unsigned int n, unsigned int m);
void initMatrix(int ** A, unsigned int n, unsigned int m);
void deallocateMatrix(int ** A, unsigned int n);
int ** makeBitonal(int ** A, unsigned int n, unsigned int m,
     int threshold);
void printMatrix(int ** A, unsigned int n, unsigned int m,
                     string description);

#endif /* EX01_LIBRARY_H_ */
```

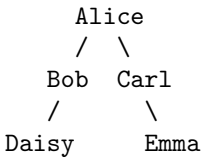
File `ex01-library.cpp`

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

//Exercise 1 (a) Check and correct if necessary
int ** createMatrix(unsigned int n, unsigned int m){
  int ** A = new int *[n];
  for(unsigned int i = 0; i<=n; i++){
    A[i] = new int[n];
  }
}
//Exercise 1 (b) Implement this function
int ** duplicateMatrix(int ** A, unsigned int n, unsigned int m){
  //Put your code here
}
//Exercise 1 (c) Implement this function
void initMatrix(int ** A, unsigned int n, unsigned int m){
  //Put your code here
}
//Exercise 1 (d) Implement this function
void deallocateMatrix(int ** A, unsigned int n){
  //Put your code here
}
//Exercise 1 (e) Implement this function
int ** makeBitonal(int ** A, unsigned int n, unsigned int m,
     int threshold){
  //Put your code here
}
//Do not modify
void printMatrix(int ** A, unsigned int n, unsigned int m,
                     string description){
  cout<< "Printing:␣" << description << endl;
  for(unsigned int i = 0; i < n; i++){
    for(unsigned 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 *descendants* of an individual, or the computation of all individuals that had children (i.e., that are *parents*). 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 computeParentNodes(Node *n, set<string> & parents)
```

and correct it if necessary. A *parent node* is a node with at least one child. This function should correctly compute all *parent nodes* met while navigating the tree starting from node `n` (including `n` itself, in case it has children). In particular, the function should add the name of the current node `n` to the set `parents` if `n` has at least one child, and then recursively call the function on the children of `n`. For instance, the parent nodes found starting from Alice's node are: Alice, Bob and Carl. Instead, the parent nodes met starting from Bob's node is just Bob itself.

Exercise follows in next page...

(b) Implement function

```
void computeMembersOfSubTree(Node * n, set<string> & members)
```

This function should compute the set of names of all members of the sub-tree starting from node n (including n itself). Such names have to be added to the set `members`. For instance, the names of the members of the sub-tree starting from Alice's node are: Alice, Bob, Carl, Daisy, and Emma. Instead, the sub-tree starting from Daisy's node contains only Daisy.

Exercise follows in next page...

File `ex02-main.cpp`

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

void experimentParentNodes(Node * n){
  set<string> parents;
  computeParentNodes(n,parents);
  cout<<"The␣nodes␣with␣children␣starting␣from␣"
      << n->name << "␣are:\n";
  printSet(parents);
}

void experimentSubtree(Node * n){
  set<string> members;
  computeMembersOfSubTree(n,members);
  cout<<"The␣sub-tree␣with␣root␣in␣"<< n->name
      << "␣contains:\n";
  printSet(members);
}

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␣parent␣nodes\n";
  experimentParentNodes(Alice);
  experimentParentNodes(Bob);
  experimentParentNodes(Carl);
  experimentParentNodes(Daisy);
  experimentParentNodes(Emma);
  cout << "Experiment␣about␣sub-tree\n";
  experimentSubtree(Alice);
  experimentSubtree(Bob);
  experimentSubtree(Carl);
  experimentSubtree(Daisy);
  experimentSubtree(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 computeParentNodes(Node* n, set<string> & parents);
void computeMembersOfSubTree(Node* n, set<string> & members);
#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 computeParentNodes(Node *n, set<string> & parents){
  if(n->left != nullptr || n->right != nullptr){
    parents.insert(n->name);
  }
  computeParentNodes(n->left,parents);
  computeParentNodes(n->right,parents);
}

//Exercise 2 (b) Implement this function
void computeMembersOfSubTree(Node * n, set<string> & members){
  //Put your code here
}
```

Exercise 3. Currency Converter (2.5 points)

Claire wants to implement a class `CurrencyConverter` to support some basic currency conversion functionalities, like converting from Danish krone (DKK) to EURO (EUR), and vice versa. 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 `CurrencyConverter` 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> currencies`: The set of currently supported currencies. We store the code (e.g., DKK for Danish krone, EUR for EURO, and USD for US dollars).
- `map<string,double> currencyToExchangeRate`: A mapping from strings (representing the code of a currency) into doubles (representing the exchange rate from the currency to DKK).

The idea is that this currency converter is based on DKK. Everytime we want to add a new currency, we have to provide its exchange rate in DKK. In other words:

- OTHER→DKK: by multiplying an amount in a given currency by its exchange rate we get the corresponding amount in DKK
  *(intuitively, DKK = exchangeRate*OTHER).*
- DKK→OTHER: by dividing an amount in DKK by the exchange rate of a given currency we get the corresponding amount in the given currency
  (intuitively, OTHER = DKK/exchangeRate).

Claire already implemented the default constructor of CurrencyConverter where the converter is initialized with one currency only, DKK, with exchange rate 1. Claire also provided a method

```
bool supportsCurrency(string currencyCode)
```

which returns true if the currency with code `currencyCode` is currently supported, 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 supported currencies. For each supported currency it should print a line (starting with an empty space) of this form:

Exercise follows in next page...

```
currency code has exchange rate exchangeRate
```

where `code` and `exchangeRate` are, respectively, the code and exchange rate of the considered currency.

(b) Check the implementation of method

```
bool addCurrency(string currencyCode,double exchangeRateToDKK)
```

and correct it if necessary. The method should add a new currency, together with its exchange rate to DKK. Special cases:

- If the currency is already supported, ignore the request, and return false.
- If the exchange rate is not positive, ignore the request, and return false.

If the request succeeds, the method should return true.

(c) Implement method

```
bool updateExchangeRate(string currencyCode, double newExchangeRate)
```

This method should update the exchange rate of an existing currency, and return true if none of the following special cases apply:

- If the currency is not supported, ignore the request, and return false.
- If the new exchange rate is not positive, ignore the request, and return false.
- If `currencyCode` is DKK, ignore the request and return false.

If the request succeeds, the method should return true.

(d) Implement method

```
double convertToDKK(double amount,string currencyCodeOfSource)
```

This method should convert (and return) `amount` from `currencyCodeOfSource` to DKK. Special cases:

- If the source currency is not supported, return -1.
- If the amount is not positive, return -1.

(e) Implement method

```
double convertFromDKK(double amountDKK,string currencyCodeOfTarget);
```

This method should convert (and return) `amount` from DKK to `currencyCodeOfTarget`. Special cases:

Exercise follows in next page...

- If the target currency is not supported, return -1.
- If the amount is not positive, return -1.

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() {
  CurrencyConverter cc;
  cout << "\naddCurrency(\"EUR\",7.44416);\n";
  cc.addCurrency("EUR",7.44416);
  cc.print();

  cout << "\naddCurrency(\"EUR\",17.44416);\n";
  cc.addCurrency("EUR",17.44416);
  cc.print();

  cout << "\naddCurrency(\"USD\",16.31708);\n";
  cc.addCurrency("USD",16.31708);
  cc.print();

  cout << "\nupdateExchangeRate(\"USD\",6.31708);\n";
  cc.updateExchangeRate("USD",6.31708);
  cc.print();

  cout <<"\n\n";

  double amountDKK=100;
  double amountEUR=cc.convertFromDKK(amountDKK,"EUR");
  cout << amountDKK <<"␣DKK␣=␣" << amountEUR << "␣EUR\n";

  amountDKK = cc.convertToDKK(amountEUR,"EUR");
  cout << amountEUR <<"␣EUR␣=␣" << amountDKK << "␣DKK\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 CurrencyConverter {
private:
  set<string> currencies;
  map<string,double> currencyToExchangeRate;
public:
  CurrencyConverter();
  bool supportsCurrency(string currencyCode);
  void print();
  bool addCurrency(string currencyCode, double exchangeRateToDKK);
  bool updateExchangeRate(string currencyCode, double newExchangeRate);
  double convertToDKK(double amount,string currencyCodeOfSource);
  double convertFromDKK(double amountDKK,string currencyCodeOfTarget);
};

#endif /* EX03_LIBRARY_H_ */
```

Exercise follows in next page...

File `ex03-library.cpp`

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

//Do not modify
CurrencyConverter::CurrencyConverter() {
  currencies.insert("DKK");
  currencyToExchangeRate["DKK"]=1;
}

//Do not modify
bool CurrencyConverter::supportsCurrency(string currencyCode){
  if(currencies.find(currencyCode) != currencies.end()){
    //I have the currency.
    return true;
  }
  else{
    return false;
  }
}

//Exercise 3 (a) Check and correct if necessary
void CurrencyConverter::print(){
  cout << "The converter supports the following currencies:"<<endl;
  for (map<string,double>::iterator it=currencyToExchangeRate.begin(); it!=currencyToExchangeRate.end(); ++it){
    cout << ' '<< "currency " << it->second << " has exchange rate " << it->first << endl;
  }
}

//Exercise 3 (b) Check and correct if necessary
bool CurrencyConverter::addCurrency(string currencyCode,double exchangeRateToDKK) {
    if(supportsCurrency(currencyCode)){
    //I already have this element. Hence I return false
    return false;
  }
  else if(exchangeRateToDKK <= 0){
    //Exchange rates must be positive
    return false;
  }
  else{
    currencies.insert(currencyCode);
    currencyToExchangeRate[exchangeRateToDKK]=currencyCode;
    return true;
  }
}

//Exercise 3 (c) Implement this function
bool CurrencyConverter::updateExchangeRate(string currencyCode,double newExchangeRate) {
  //Put your code here
}

//Exercise 3 (d) Implement this function
double CurrencyConverter::convertToDKK(double amount, string currencyCodeOfSource) {
  //Put your code here
}

//Exercise 3 (e) Implement this function
double CurrencyConverter::convertFromDKK(double amountDKK,string currencyCodeOfTarget) {
  //Put your code here
}
```

Exercise 4. Parametric Double-ended Queue (2.5 points)

Daisy attended the course 02393 Programming in C++ at DTU. Taking inspiration from some of the examples seen in class, she decided to implement a queue (i.e. a list) in which it is possible to add elements both at the end (push_back, like for vectors), and at the beginning (push_front). Data structures with these functionalities are known as double-ended queues, or **deque**s.

Daisy is implementing a C++ library for manipulating parametric deques. She has prepared a test program (`ex04-main.cpp`), the declaration of class `mydeque` (`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 deque using the parametric struct `Node` containing two fields:

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

Daisy has decided to use the following internal (`private`) representation for deques:

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

If the deque is empty, then both `first` and `last` have value `nullptr`. If the deque has size one, then `first` and `last` point to the same `Node`.

Daisy has already implemented several functions, including:

- a constructor and a destructor.
- a method `push_when_empty(T v)` to insert elements in an empty deque. [1]
- public methods to print the value stored in the first (`print_front()`) and last position of the deque (`print_back()`), and to print information on the list (`print()`).

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

(a) Check the class destructor and correct it if necessary. This method should deallocate all memory used by each of its `Node` structs.

(b) Implement the method `push_back(T v)`. This method should add an element `v` at the end of the deque. Hence `v` becomes the last element of the deque. Special cases:

- If the method is invoked on a empty deque, then this method just invokes the method `push_when_empty(T v)`

Exercise follows in next page...

[1] This method should be private, but we made it public to ease the development of tests.

(c) Implement the method `push_front(T v)`. This method should add an element `v` at the beginning of the deque. Hence `v` becomes the first element of the deque. Special cases:

- If the method is invoked on a empty deque, then this method just invokes the method `push_when_empty(T v)`

(d) Check method `print_back()`, and correct it if necessary. The method should print the last element of the deque (if any). Similarly to method `print`, if the method is invoked on an empty deque, then it should print on screen using command:

```
cout << "The deque is empty.\n";
```

(e) Complete the implementation of method `pop_front()`. This method should remove the first element in the deque, effectively reducing the deque size by one. The memory allocated to store the element in the deque should be deallocated. The method should return true if the deque contained at least an element, meaning that the pop succeeded. Instead, it should return false it the deque was empty, and hence the pop failed. Special cases:

- If the method is invoked on an empty deque, nothing is done, and the value false is returned.
- If the method is invoked on a deque with one element, the deque becomes empty, and the value true is returned.

Daisy already wrote the code necessary for the special cases. Help Daisy by providing the code for the general case of deques with 2 or more elements.

Exercise follows in next page...

File `ex04-main.cpp`

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

using namespace std;

int main() {
  mydeque<int> deque;

  deque.push_front(2);
  deque.push_front(1);
  deque.push_back(3);

  deque.print();

  cout <<"\n";
  cout << "The␣first␣element␣is:␣";
  deque.print_front();
  cout << "The␣last␣element␣is:␣";
  deque.print_back();

  do{
    cout <<"\n";
    cout << "pop_front():␣";
    cout <<deque.pop_front() <<"\n";
    deque.print();
  }while(deque.getSize()>0);

  cout <<"\n";
  cout << "pop_front():␣";
  cout << deque.pop_front() <<"\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 mydeque {
private:
  Node<T> * first;
  Node<T> * last;
  int size;
public:
  mydeque();
  void push_when_empty(T v);
  ~mydeque();
  int getSize();
  void print_front();
  void print_back();
  void print();
  void push_back(T v);
  void push_front(T v);
  bool pop_front();
};
#endif
```

File `ex04-library.cpp`

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

using namespace std;

//Do not modify
template<class T>
mydeque<T>::mydeque() {
  size=0;
  first=nullptr;
  last=nullptr;
}

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

//Do not modify
template<class T>
void mydeque<T>::print_front() {
  if(size==0){
    cout << "The deque is empty.\n";
  }
  else{
    cout << first->content <<"\n";
  }
}

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

//Do not modify
template<class T>
void mydeque<T>::push_when_empty(T v) {
  Node<T> * node = new Node<T>;
  node->content=v;
  node->next=nullptr;
  first=node;
  last=node;
  size=1;
}

//Exercise 4 (a) Check and correct if necessary
template<class T>
mydeque<T>::~mydeque() {
  Node<T> * current = first;
  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 mydeque<T>::push_back(T v) {
  //Put your code here
}

//Exercise 4 (c) Implement this function
template<class T>
void mydeque<T>::push_front(T v) {
  //Put your code here
}

//Exercise 4 (d) Check and correct if necessary
template<class T>
void mydeque<T>::print_back() {
  if(size!=0){
    cout << "The deque is empty.\n";
  }
  else{
    cout << first->content <<"\n";
  }
}

//Exercise 4 (e) Complete body of last else
template<class T>
bool mydeque<T>::pop_front() {
  if(size==0){
    //Cannot pop from an empty deque. I return false;
    return false;
  }
  else if(size==1){
    //Since size is 1, the deque becomes empty
    delete first;
    first=nullptr;
    last=nullptr;
    size=0;
    return true;
  }
  else{
    //The deque has at least 2 elements.
    //Put your code here
  }
}

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