Technical University of Denmark

Written examination, December 09, 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 also upload your solutions on CodeJudge under `Exam December 2018` at `https://dtu.codejudge.net/02393-e18/exercises`. 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. Temperature Scales Converter (2.5 points)

Alice needs to implement a library to manipulate a data set of temperatures registered in Copenhagen. This is stored as an array where each entry takes a double value corresponding to a temperature in Celsius scale (e.g., water freezes at $0°C$, and boils at $100°C$ in standard pressure conditions).

Alice needs to perform some computations on such data. 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

```
double * createAndInitArray(unsigned int n, double value)
```

and correct it if necessary. The function should correctly create an array of doubles of length `n`, and return it. The function should allocate the required memory, and initialized each entry of the array to `value`.

(b) Implement function

```
double * duplicateArray(double * A, unsigned int n);
```

The function should create and return a copy of the array `A` of length `n`.

(c) Implement function

```
void deallocateArray(double * A);
```

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

Exercise follows in next page...

(d) Implement function

```
double * toFahrenheit(double * A, unsigned int n);
```

This function transforms the data set from the Celsius scale to the Fahrenheit scale. The function takes as input an array `A` of length `n`, and returns a new array of same length containing the measurements in `A` converted in Fahrenheit scale. The function should not modify `A`. Each value v_C in Celsius scale has to be transformed in the corresponding value v_F in Fahrenheit scale computed as follows:

$$v_F = v_C * 1.8 + 32$$

Notes:

- The function has to apply the conversion 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, the data set $\texttt{A} = [-10, 0, 5]$ in Celsius scale is converted in the data set `B` in Fahrenheit scale:

$$\texttt{B} = [-148, 32, 122]$$

because

$$\begin{aligned}(-10) * 1.8 + 32 &= -148\\ 0 * 1.8 + 32 &= 32\\ 5 * 1.8 + 32 &= 122\end{aligned}$$

Exercise follows in next page...

File ex01-main.cpp

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

using namespace std;

int main(void){

  // Array containing Celsius
  // temperatures initialized to 100
  unsigned int n = 4;
  double *A = createAndInitArray(n,100);

  // Setting some values in the array
  unsigned int m = 3;
  for(int i=0;i<n-1;i++){
    for(int j=0;j<m-1;j++){
      A[i]+=i*n+j*3;
    }
  }
  printArray(A,n,"main␣array");

  double * B = duplicateArray(A,n);
  printArray(B,n,"copy");

  //I change B, and I print B and A
  B[0] = B[0]+3;
  B[1] = B[0]+3;
  B[2] = B[1]+3;

  printArray(B,n,"modified␣copy");
  printArray(A,n,"main␣array");

  //I deallocate B
  deallocateArray(B);

  //I convert to Fahrenheit
  double * C = toFahrenheit(A,n);
  printArray(C,n,"Fahrenheit␣copy");
  printArray(A,n,"main␣array␣not␣modified");

  //I deallocate A and C
  deallocateArray(A);
  deallocateArray(C);

  cout << "Completed"<<endl;
  return 0;
}
```

File ex01-library.h

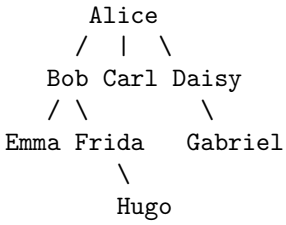
```
#ifndef EX01_LIBRARY_H_
#define EX01_LIBRARY_H_

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

double * createAndInitArray(unsigned int n, double value);
double * duplicateArray(double * A, unsigned int n);
void deallocateArray(double * A);
double * toFahrenheit(double * A, unsigned int n);
void printArray(double * A, unsigned int n, string descr);

#endif /* EX01_LIBRARY_H_ */
```

File ex01-library.cpp

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

using namespace std;

//Exercise 1 (a) Check and correct if necessary
double * createAndInitArray(unsigned int n, double value){
  double * A = new double;
  for(unsigned int i = 0; i<=value+1; i++){
    A[i] = value;
  }
  return A;
}

//Exercise 1 (b) Implement this function
double * duplicateArray(double * A, unsigned int n){
  //put your code here
}

//Exercise 1 (c) Implement this function
void deallocateArray(double * A){
  //put your code here
}

//Exercise 1 (d) Implement this function
double * toFahrenheit(double * A, unsigned int n){
  //put your code here
}

//Do not modify
void printArray(double * A, unsigned int n, string descr){
  cout<< "Printing:␣" << descr << endl;
  for(unsigned int i = 0; i < n; i++){
    cout << A[i] << '␣';
  }
  cout << "\n\n";
}
```

Exercise 2. Genealogy (2.5 points)

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



Alice had three children: Bob, Carl, and Daisy. Bob had two children: Emma, and Frida, while Daisy had one, Gabriel, and Frida had one, Hugo. Finally, Carl, Emma, Gabriel, and Hugo had no children.

Bob would like to automate the computation of simple queries on family trees, such as the computation of all individuals that had children (i.e., that are *parents*), or the computation of the *descendants* of an individual. 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.

Bob has decided to represent family trees using tree data structures, where each node can have any number of children. Each node of the tree is stored in an object of class `Node` containing:

- `name`: the name of the corresponding individual;
- `children`: a vector of pointers to other `Node` objects, representing the children of the corresponding individual. The vector is empty if the individual had no children.

Help Bob by solving the following tasks:

(a) Implement the constructor

```
Node(string name)
```

which just sets the given parameter, `string name`, as the name of the node. Remember that you don't need to initialize vectors because they are initialized automatically. Implement the getter method

```
string getName()
```

which just returns the name of the node.

(b) Implement method

```
void addChild(Node * child)
```

which adds `child` to the vector `children` of the node.

Exercise follows in next page...

(c) Check the implementation of method

```
void printParentNodes()
```

and correct it if necessary. A *parent node* is a node with at least one child. This method should correctly print all parent nodes met while navigating the tree starting from node `n` on which the method has been invoked (including `n` itself, in case it has children). In particular, if the node has at least one child the method should:

- print the name of the current node, followed by a blank space (the char `'␣'`)
- after this, the method is recursively invoked on the children of the current node, following the order in which they appear in the vector `children`.

For instance, invoking `printParentNodes` on Alice's node we get:

```
Alice Bob Frida Daisy
```

Instead, invoking the method on Bob's node we get:

```
Bob Frida
```

(d) Implement method

```
void printMembersOfSubTree(int generation=0)
```

This method should print the names of all members of the sub-tree starting from node `n` on which the method has been invoked (including `n` itself). Also, for each considered node it should compute the *generation* of the node with repsect to the node `n` on which the method has been originally invoked. E.g., if we invoke the method on `Alice`, then she is in generation 0, while her sons Bob, Carl and Daisy are in generation 1, her grandsons are in generation 2, and so on.

In particular, for each considered node, the method should print the name of the node, followed by a blank space (i.e., the char `'␣'`), followed by the generation of the node with respect to the node `n` on which the method has been originally invoked, followed by another blank space. After this, the method should be recursively invoked on the children of the current node, following the order in which they appear in the vector `children`. Note that the parameter `generation` has default value `0`, meaning that invoking `printMembersOfSubTree()` is equivalent to `printMembersOfSubTree(0)`.

For instance, invoking `printMembersOfSubTree` on Alice's node we get:

```
Alice 0 Bob 1 Emma 2 Frida 2 Hugo 3 Carl 1 Daisy 1 Gabriel 2
```

Instead, invoking `printMembersOfSubTree` on Bob's node we get:

```
Bob 0 Emma 1 Frida 1 Hugo 2
```

Exercise follows in next page...

File ex02-main.cpp

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

int main() {
  /* Bob's family tree from text */
  Node *Alice = new Node("Alice");
  Node *Bob = new Node("Bob");
  Node *Carl = new Node("Carl");
  Node *Daisy = new Node("Daisy");
  Node *Emma = new Node("Emma");
  Node *Frida = new Node("Frida");
  Node *Gabriel = new Node("Gabriel");
  Node *Hugo = new Node("Hugo");

  cout << "Alice's␣name:␣" << Alice->getName();
  cout << endl << endl;

  Alice->addChild(Bob);
  Alice->addChild(Carl);
  Alice->addChild(Daisy);
  Bob->addChild(Emma);
  Bob->addChild(Frida);
  Daisy->addChild(Gabriel);
  Frida->addChild(Hugo);

  cout << "Experiments␣about␣parent␣nodes";
  cout << endl << "Alice:␣";
  Alice->printParentNodes();
  cout << endl << "Bob:␣";
  Bob->printParentNodes();
  cout << endl << "Carl:␣";
  Carl->printParentNodes();
  cout << endl << "Daisy:␣";
  Daisy->printParentNodes();
  cout << endl << "Emma:␣";
  Emma->printParentNodes();
  cout << endl << "Frida:␣";
  Frida->printParentNodes();
  cout << endl << endl;

  cout << "Experiment␣about␣sub-tree";
  cout << endl << "Alice:␣";
  Alice->printMembersOfSubTree();
  cout << endl << "Bob:␣";
  Bob->printMembersOfSubTree();
  cout << endl << "Carl:␣";
  Carl->printMembersOfSubTree();
  cout << endl << "Daisy:␣";
  Daisy->printMembersOfSubTree();
  cout << endl << "Emma:␣";
  Emma->printMembersOfSubTree();
  cout << endl << "Frida:␣";
  Frida->printMembersOfSubTree();
  cout << endl << "Gabriel:␣";
  Gabriel->printMembersOfSubTree();
  cout << endl << "Hugo:␣";
  Hugo->printMembersOfSubTree();
  cout << endl;

  return 0;
}
```

File ex02-library.h

```
#ifndef EX02_LIBRARY_H_
#define EX02_LIBRARY_H_

#include <vector>
#include <string>

using namespace std;

class Node{
  private:
    string name;
    vector<Node *> children;
  public:
    int countChildren();
    Node(string name);
    string getName();
    void addChild(Node * child);
    void printParentNodes();
    void printMembersOfSubTree(int generation=0);
};

#endif
```

File ex02-library.cpp

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

//Do not modify
int Node::countChildren(){
  return children.size();
}

//Exercise 2 (a) Implement the constructor and getName()
Node::Node(string name){
  //put your code here
}

string Node::getName(){
  //put your code here
}

//Exercise 2 (b) Implement this method
void Node::addChild(Node * child){
  //put your code here
}

//Exercise 2 (c) Check and correct if necessary
void Node::printParentNodes(){
  for(int i=0; i<children.size()-1;i++){
    children[i]->printParentNodes();
  }
  cout << this->name << '␣';
}

//Exercise 2 (d) Implement this method
void Node::printMembersOfSubTree(int generation){
  //put your code here
}
```

Exercise 3. Temperatures Scales Converter 2 (2.5 points)

Claire wants to implement a class `TemperatureScalesConverter` to support the storing of temperature measurements in different temperature scales: Celsius (C), and Fahrenheit (F). 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 and in the next pages. Help Claire by implementing the class `TemperatureScalesConverter` in file `ex03-library.cpp`.

Claire decided to use the `vector` containers of the standard library. So she has decided to use the following internal (`private`) representation for the library:

- `vector<double> CTemperatures`: A vector containing the temperature measurements in Celsius scale.
- `vector<double> FTemperatures`: A vector containing the temperature measurements in Fahrenheit scale.

The general idea is that the converter supports two scales: Celsius and Fahrenheit. Users can add new measurements in any of the two scales. Each measurement is converted to the remaining scale and added to both vectors. For example, if the user adds a Celsius measurement, then (i) it will be added to `CTemperatures`, (ii) it will be converted to Fahrenheit, and (iii) it will be added to `FTemperatures`.

Claire already implemented the default constructor of `TemperatureScalesConverter`, which adds a default measurement of 0 Celsius, equivalent to 32 Fahrenheit. 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 added measurements in this form:

```
n Celsius measurements: Cmeasurement1 Cmeasurement2 ... Cmeasurementn
n Fahrenheit measurements: Fmeasurement1 Fmeasurement2 ... Fmeasurementn
```

where `n` is the number of measurements added, `Cmeasurementi` is the i_th Celsius measurement, and `Fmeasurementi` is the i_th Fahrenheit measurement. Note that each measurement is followed by a blank space.

Exercise follows in next page...

(b) Implement method

```
double convertToF(double CTemperature)
```

which should convert the Celsius measurement `CTemperature` in Fahrenheit scale and return it.

A Celsius measurement v_C has to be converted to the corresponding value v_F in Fahrenheit scale as follows:

$$v_F = v_C * 1.8 + 32$$

(c) Implement method

```
double convertToC(double FTemperature)
```

which should convert the Fahrenheit measurement `FTemperature` in Celsius scale and return it.

A Fahrenheit measurement v_F has to be converted[1] to the corresponding value v_C in Celsius scale as follows:

$$v_C = (v_F - 32) * 0.56$$

(d) Implement method

```
bool addMeasurement(string scale, double temperature)
```

where `scale` denotes a temperature scale, i.e. `"C"` for Celsius and `"F"` for Fahrenheit, and `temperature` is a measurement in the scale denoted by the first parameter. This method should add `temperature` to the correct vector. Then, it should convert `temperature` in the other scale, and add it to the corresponding vector. Special cases:

- If `scale` is neither `"C"` nor `"F"`, the method should do nothing and return false. Otherwise the method should do what required, and return true.

Exercise follows in next page...

[1] The actual formula involves $0.5\overline{5}$, here rounded to 0.56

File `ex03-main.cpp`

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

int main() {
  TemperatureScalesConverter tsc;
  cout << "Default␣measurement␣of␣0C␣=␣32F␣:" << endl;
  tsc.print();
  cout << endl;

  double f = tsc.convertToF(20);
  cout << "20␣Celsius␣=␣" << f << "␣Fahrenheit" << endl;
  double c = tsc.convertToC(30);
  cout << "30␣Fahrenheit␣=␣" << c << "␣Celsius" << endl;

  bool res = tsc.addMeasurement("C", 20);
  if(!res){
    cout << "tsc.addMeasurement(\"C\",␣20);␣FAILED" << endl;
  }
  res = tsc.addMeasurement("F", 30);
  if(!res){
    cout << "tsc.addMeasurement(\"F\",␣30);␣FAILED" << endl;
  }
  tsc.print();
  cout << endl;

  res = tsc.addMeasurement("K", 50);
  if(!res){
    cout << "tsc.addMeasurement(\"K\",␣50);␣FAILED" << endl;
  }
  tsc.print();

  return 0;
}
```

File `ex03-library.h`

```
#ifndef EX03_LIBRARY_H_
#define EX03_LIBRARY_H_

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

class TemperatureScalesConverter {
private:
  vector<double> CTemperatures;
  vector<double> FTemperatures;
public:
  TemperatureScalesConverter();
  void print();
  double convertToF(double CTemperature);
  double convertToC(double FTemperature);
  bool addMeasurement(string scale, double temperature);
};

#endif /* EX03_LIBRARY_H_ */
```

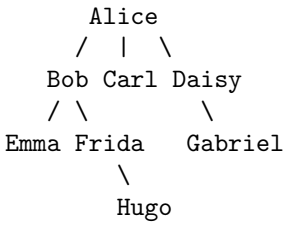
Exercise follows in next page...

File `ex03-library.cpp`

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

//Do not modify
TemperatureScalesConverter::TemperatureScalesConverter() {
  //By default we add a measurement of 0 C which corresponds to 32 F.
  CTemperatures.push_back(0);
  FTemperatures.push_back(32);
}

//Exercise 3 (a) Check and correct this method
void TemperatureScalesConverter::print(){
  cout << CTemperatures.size() <<"␣Celsius␣measurements:";
  for(int i = 0; i < CTemperatures.size() - 1; i++){
    cout<< '␣' << CTemperatures[i];
  }
  cout << endl;

  cout << FTemperatures.size() <<"␣Fahrenheit␣measurements:";
  for(int i = CTemperatures.size() - 1; i > 0; i--){
    cout<< '␣' << CTemperatures[i];
  }
  cout << endl;
}

//Exercise 3 (b) Implement this method
double TemperatureScalesConverter::convertToF(double CTemperature){
  //put your code here
}

//Exercise 3 (c) Implement this method
double TemperatureScalesConverter::convertToC(double FTemperature){
  //put your code here
}

//Exercise 3 (d) Implement this method
bool TemperatureScalesConverter::addMeasurement(string scale, double temperature){
  //put your code here
}
```

Exercise 4. Genealogy 2 (2.5 points)

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



Alice had three children: Bob, Carl, and Daisy. Bob had two children: Emma, and Frida, while Daisy had one, Gabriel, and Frida had one, Hugo. Finally, Carl, Emma, Gabriel, and Hugo had no children.

Bob has already written some code. His first test program is in file `ex04-main.cpp` and the (incomplete) code with some functions he needs is in files `ex04-library.h` and `ex04-library.cpp`. All files are available on CampusNet and in the next pages.

Bob would like to automate the computation of queries on family trees, like the computation of all individuals that had no children (i.e., *leaf* nodes), or counting of the number of *descendants* of an individual. For each individual, Bob wants to store his name. In order to make the code more general, Bob also wants to enrich each individual with a second information, of parametric type (omitted in the above diagram, but shown later in file `ex04-main.cpp`). This can be done using C++ generic programming, i.e. using templates.

Bob decided to represent family trees using trees, where each node can have any number of children. Each node is stored in an object of parametric class `Node<T>` containing:

- `string name`: the name of the corresponding individual;
- `T value`: a second information of parametric type `T` of the corresponding individual;
- `children`: a vector of pointers to `Node<T>` objects. These are the children of the corresponding individual. This vector is empty if the individual had no children.

Help Bob by solving the following tasks:

(a) Check and correct the implementation of method

```
Node(string name,T value)
```

which sets the `name` and `value` given as parameters as the name and value of the node, respectively. Remember that you don't need to initialize vectors because they are initialized automatically.

Exercise follows in next page...

(b) Implement the methods getName, getValue, and addChild. The method

```
string getName()
```

returns the name of the node. Instead,

```
T getValue()
```

returns the value of the node. While

```
void addChild(Node<T> * child)
```

adds `child` to the vector `children` of the node.

(c) Implement method

```
void printLeafNodes()
```

A *leaf node* is a node without children. This method should print all leaf nodes met while navigating the tree starting from node `n` on which the method has been invoked (including `n` itself, in case it has no children). When invoked on non-leaf nodes, the method should call recursively itself on the children of the current node, following the order in which they appear in the vector `children`. Instead, when invoked on leaf nodes, the method should print the `name` of the current node, followed by a blank space (i.e., the char '␣'), followed by `value`, followed by another blank space.

For instance, invoking `printParentNodes` on Alice's node we get:

```
Emma 01/01/1998 Hugo 01/01/2018 Carl 01/01/1978 Gabriel 01/02/1998
```

Instead, invoking the method on Carl's node we get:

```
Carl 01/01/1978
```

As you can see, in this example the `value` of each node is the birth date of the corresponding individual, as shown in `ex04-main.cpp` in the next page.

(d) Implement method

```
int countMembersOfSubTree()
```

This method should count the members of the sub-tree starting from node `n` on which the method has been invoked (including `n` itself). For instance, invoking `countMembersOfSubTree` on Alice's and Bob's node we get 8 and 4, respectively.

Exercise follows in next page...

File `ex04-main.cpp`

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

using namespace std;

int main() {
  /* Bob's family tree from text */
  Node<string> *Alice = new Node<string>("Alice", "01/01/1957");
  Node<string> *Bob = new Node<string>("Bob", "01/01/1977");
  Node<string> *Carl = new Node<string>("Carl", "01/01/1978");
  Node<string> *Daisy = new Node<string>("Daisy", "01/12/1978");
  Node<string> *Emma = new Node<string>("Emma","01/01/1998");
  Node<string> *Frida = new Node<string>("Frida","01/12/1998");
  Node<string> *Gabriel = new Node<string>("Gabriel","01/02/1998");
  Node<string> *Hugo = new Node<string>("Hugo","01/01/2018");

  cout << "Alice's␣name␣is:␣" << Alice->getName() << endl;
  cout << "Alice's␣value␣is:␣" << Alice->getValue() << endl;

  Alice->addChild(Bob);
  Alice->addChild(Carl);
  Alice->addChild(Daisy);
  Bob->addChild(Emma);
  Bob->addChild(Frida);
  Daisy->addChild(Gabriel);
  Frida->addChild(Hugo);

  cout << endl;
  cout << "Experiments␣about␣leaf␣nodes";
  cout << endl << "Alice:␣";
  Alice->printLeafNodes();
  cout << endl << "Bob:␣";
  Bob->printLeafNodes();
  cout << endl << "Carl:␣";
  Carl->printLeafNodes();
  cout << endl << "Daisy:␣";
  Daisy->printLeafNodes();
  cout << endl << "Emma:␣";
  Emma->printLeafNodes();
  cout << endl << "Frida:␣";
  Frida->printLeafNodes();
  cout << endl;

  cout << endl;
  cout << "Experiment␣about␣sub-tree";
  cout << endl << "Alice:␣";
  cout << Alice->countMembersOfSubTree();
  cout << endl << "Bob:␣";
  cout << Bob->countMembersOfSubTree();
  cout << endl << "Carl:␣";
  cout << Carl->countMembersOfSubTree();
  cout << endl << "Daisy:␣";
  cout << Daisy->countMembersOfSubTree();
  cout << endl << "Emma:␣";
  cout << Emma->countMembersOfSubTree();
  cout << endl << "Frida:␣";
  cout << Frida->countMembersOfSubTree();
  cout << endl;

  return 0;
}
```

Exercise follows in next page...

File `ex04-library.h`

```
#ifndef EX04_LIBRARY_H_
#define EX04_LIBRARY_H_

#include <vector>
#include <string>

using namespace std;

template<class T>
class Node{
  private:
    string name;
    T value;
    vector<Node *> children;
  public:
    void printNode();
    Node(string name, T value);
    string getName();
    T getValue();
    void addChild(Node * child);
    void printLeafNodes();
    int countMembersOfSubTree();
};

#endif
```

File `ex04-library.cpp`

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

using namespace std;

//Do not modify
template<class T>
void Node<T>::printNode(){
  cout << "Name: " << name << ", value: " << value ;
  cout << ", children: " << children.size() << endl;
}

//Exercise 4 (a) Check and correct if necessary
template<class T>
Node<T>::Node(string name, T value){
}

//Exercise 4 (b) Implement getName, getValue, and addChild
template<class T>
string Node<T>::getName(){
  //put your code here
}

template<class T>
T Node<T>::getValue(){
  //put your code here
}

template<class T>
void Node<T>::addChild(Node<T> * child){
  //put your code here
}

//Exercise 4 (c) Implement this method
template<class T>
void Node<T>::printLeafNodes(){
  //put your code here
}

//Exercise 4 (d) Implement this method
template<class T>
int Node<T>::countMembersOfSubTree(){
  //put your code here
}

//Do not modify
template class Node<string>;
```