APA254 - Data Structures - Assignment 3

Stanislas Lange - 9319520196

`Makefile`:

```
CC = g++
CFLAGS = -g -Wall
LFLAGS =
TARGET = binaryTreeTraversals

all: $(TARGET)

$(TARGET): $(TARGET).cpp
	$(CC) $(CFLAGS) -o $(TARGET) $(TARGET).cpp

clean:
	$(RM) $(TARGET)
```

`binaryTreeTraversals.cpp`:

```
#include <iostream>
#include "arrayQueue.h"
#include "binaryTreeNode.h"
#include "myExceptions.h"

using namespace std;

template <class T>
void visit(binaryTreeNode<T> *x)
{
    cout << x->element << ' ';
}

template <class T>
void preOrder(binaryTreeNode<T> *t)
{
    if (t != NULL) {
        visit(t);
        preOrder(t->leftChild);
        preOrder(t->rightChild);
    }
}

template <class T>
void inOrder(binaryTreeNode<T> *t)
{
    if (t != NULL) {
        inOrder(t->leftChild);
        visit(t);
        inOrder(t->rightChild);
    }
}

template <class T>
void postOrder(binaryTreeNode<T> *t)
{
    if (t != NULL) {
        postOrder(t->leftChild);
        postOrder(t->rightChild);
        visit(t);
    }
}

template <class T>
void levelOrder(binaryTreeNode<T> *t)
{
    if (t != NULL) {
        arrayQueue<binaryTreeNode<T>*> q;
        binaryTreeNode<int> *curr;

        // Enqueue root element and NULL node as delimiter for level
        q.push(t);
        q.push(NULL);
```

```
        // We visited all the elements when the queue contains
        // only the last NULL delimiter
        while (q.size() > 1)
        {
            // Front node of the queue becomes current node and is deleted from the queue
            curr = q.front();
            q.pop();

            if (curr == NULL) {
                // We're at NULL, which means we're starting to visit a new level
                // That means we just finished to enqueue this level
                // So we have to delimit its ending with a NULL
                q.push(NULL);
            } else {
                // We add the childs of the current to the next level in the queue
                if(curr->leftChild)
                    q.push(curr->leftChild);

                if(curr->rightChild)
                    q.push(curr->rightChild);

                // Then visit the node
                visit(curr);
            }
        }
    }

    int main(void)
    {
        binaryTreeNode<int> *root;
        root = new binaryTreeNode<int> (1);
        root->leftChild = new binaryTreeNode<int> (2);
        root->rightChild = new binaryTreeNode<int> (3);
        root->leftChild->leftChild = new binaryTreeNode<int> (4);
        root->leftChild->rightChild = new binaryTreeNode<int> (5);

        cout << "Inorder: ";
        inOrder(root);
        cout << "\nPreorder: ";
        preOrder(root);
        cout << "\nPostorder: ";
        postOrder(root);
        cout << "\nLevel order: ";
        levelOrder(root);

        return 0;
    }
```