Advanced C++ Programming

Exercise

https://campus.cs.le.ac.uk/teaching/resources/CO7105/Code/TreeNode.zip

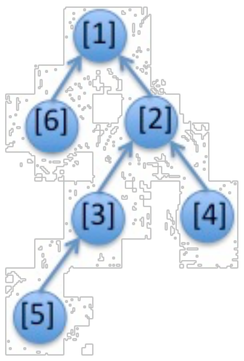
The following class (**`TreeNode`**) can be used to represent tree data structure in C++. Each tree node has an integer value (**`data`**) attached to it.

```
//TreeNode.h
class TreeNode{
public:
	TreeNode(TreeNode* parent, int data);    //Constructor
	~TreeNode();    //Destructor
	int getData() const;    //get data of the node
	int num_child() const;    //get the number of direct child
	void addChild(TreeNode* child);    //add new child node
	friend std::ostream& operator<<(std::ostream& os, const TreeNode& t); //overload <<
private:
	void clean_();
	void reallocate_();
	int numChild_;
	int data_;
	TreeNode** children_;
	TreeNode* parent_;
};
```

```
///main.cpp
//…
int main(){
	TreeNode *t1=new TreeNode(nullptr,1);
	TreeNode *t2=new TreeNode(t1,2);
	TreeNode *t3=new TreeNode(t2,3);
	TreeNode *t4=new TreeNode(t2,4);
	TreeNode *t5=new TreeNode(t3,5);
	TreeNode *t6=new TreeNode(t1,6);

	std::cout<<*t1;
	delete t1;

}
```



The main program on the left should create a tree like this.

Advanced C++ Programming

Tasks

(1) Complete the constructor.
(2) Implement all member functions.
(3) Implement the destructor (should also delete all its direct and indirect child nodes e.g. delete t1 should also delete t2, t3, t4, t5, t6)
(4) Complete the friend function so that std::cout<<*t1; will generate the output as shown below.

```
[1]
    [2]
         [3]
              [5]
         [4]
    [6]
```

[Hint: use recursion and and `setw(int)` to print different amounts of indentation for each level in the tree]

(5) Add the following member functions to the class `int maxDepth(…)`

Given a tree and a root node, find its maximum height, which is the number of nodes along the path from the root node to the deepest leaf node (For example, the max depth from node [1] is 4; the max depth from node [3] is 2.)

Solution

//TreeNode.cpp

```
TreeNode::TreeNode(TreeNode* parent, int data)
:numChild_{0},data_(data),children_(nullptr),parent_(parent){
    if(parent_!=nullptr){
        parent->addChild(this);
    }
}

TreeNode::~TreeNode(){
    std::cout<<"deallocated:"<<data_<<std::endl;
    clean_();
}


void TreeNode::addChild(TreeNode* child){
    reallocate_();
    children_[numChild_-1]=child;
}

void TreeNode::clean_(){
    for (unsigned int i=0 ; i<numChild_ ; ++i) {
        delete children_[i];
    }
    if (parent_) {
        parent_=nullptr;
    }
    if (children_) {
        delete [] children_;
        children_ = nullptr;
    }
}

void TreeNode::reallocate_(){
    numChild_++;
    TreeNode** temp = new TreeNode*[numChild_];
        for (unsigned int i=0 ; i<numChild_-1 ; ++i) {
            temp[i] = children_[i];
        }
        if (numChild_) {
            delete [] children_;
        }
        children_ = temp;
}

int TreeNode::num_child() const{
    return numChild_;
}

int TreeNode::getData() const{
    return data_;
}


int TreeNode::getMaxDepth(const TreeNode* node) const{
    if(node == nullptr){
      return 0;
    }
    int current_max=1;
    int depth=1;
    for(int i=0;i<node->numChild_;i++){
        depth=getMaxDepth(node->children_[i])+1;
        if(depth>=current_max){
            current_max=depth;
        }
    }
    return current_max;
}


void TreeNode::print(std::ostream& os, const TreeNode* node, int indent) const{
    os<<std::setw(indent)<<"["<<(node->getData())<<"]"<<std::endl;
    indent+=SPACE;
    for(int i=0;i<node->numChild_;i++){
        print(os,node->children_[i], indent);
    }
}

std::ostream& operator<<(std::ostream& os, const TreeNode& l){
    l.print(os,&l,0);
    return os;
}
```