BTree Implementation

Implement a BTree (**BT**). As you recall a BTree is a multi-way tree of degree ***m***. Where ***m*** defines the branching factor (or the maximum number of subtrees) for a node. For this assignment you can assume that the keys stored in the node are of **type integers** (your implementation must be generic, i.e. use templates). Your implementation you must provide the following methods:

- Insert key
- Find/Search for key
- Delete/Remove key
- Print tree (level order printing)

<u>Key Functionality</u>

- Command line arguments to load a driver file. No prompting or hard coding input data files. Your program will read commands from the driver file instead of from the user.
    - Commands in the driver file, one command per line
        - **L: loadfile**
          {open filename and insert keys into BTree}. Assume **<u>keys are integer values</u>** separated by a ***space. No assumption should be made about how many items are in the loadfile. Read until end of file.*** This will allow bulk-loading items in your BTree

        - **A: key**
          {insets the key to BTree }. If key is duplicate, print warning message, "**Warning, duplicate value, ignoring**"

        - **D: key**
          {delete **key** from BTree }. This will remove the specified value from BTree. Operation leaves your tree in a valid state, that is your tree satisfies conditions for BTree.

        - **S: key**
          {find/search for key in BTree}. Prints out each key examined on path to key. If key is not found, a not found message is printed out at the end.

        - **P:**
          {prints BTree in level order}.

        - **T:**

{Terminate program}
You can generate your own data, please test your implementation.

Documentation. Please write a concise paragraph explaining your design philosophy and implementation. Each method you write should specify pre/post condition and type of arguments, and if arguments are modified.

Public Interfaces that will be tested

Interface	**Comment**	**Tree Height(1)**	**Tree Height(2)**	**Tree Height(>=3)**
Insert(Type K)	inserts object Type into tree{Success, Duplicate}			
Print()	print tree using level order			
ShowFind(Type K)	Print the path of keys examined while searching for K			
Delete(Type K)	delete object with key **K** {SUCCESS, KEY_NOT_FOUND }			
reading from file	Pass command filename from command line.			
memory management	You must manage your memory allocation/deallocation. Failure to do so may result in a failing score			

```
// possible return status/values from methods
enum RETURN_FLAG {SUCCESS, PARTIAL_INSERT, DEL_UNDERFLOW, DUPLICATE, KEY_NOT_FOUND};

// Basic structure of a BTree node
template <typename T, int M=5>
struct BTreeNode {
        public:

        // number of keys stored in node
        int _keyCount;
        int _keyLink;
        // array to store keys in node
        T *keys;

        // array to store links/(pointer to substrees) in node
        T **links;

        // a node knows how to add a key to itself
        tuple<…> _addKey(const int kval, BTreeNode *ptr = nullptr);

        // a node knows how to remove a key from itself
        tuple<…> _removeKey(const int kval, BTreeNode *ptr = nullptr);

        // a node knows how split itself
        tuple<…> _split();

        // a node knows how merge with another node
        tuple<…> _merge(BTreeNode *ptr);


        // constructors and other member methods
        . . .
};

template <typename T, int SIZE=5>
class BTree {
        // root of BTree
        BTreeNode<T, SIZE> *root;

        // internal insert function
        tuple<…> _insert(BTreeNode<T, SIZE> *node, T &key, );

        // internal delete function
        tuple<…> _delete(BTreeNode<T, SIZE> *node, T &key);

        // Search keys in node. Returns the index where idx, where keyLookingFor >= keys[idx]
        int searchForKeyInNode(T &key, BTreeNode<T, SIZE>) ;
        . . .

public:
        // default constructor
        BTree() {
                …
        }
        // Public API to Tree
        RETURN_FLAG             Insert(T &key);
        RETURN_FLAG             Delete(T &key);

        // prints out the sequence of keys examined when searching for the supplied key
        void                    ShowFind(T &key);
        void                    Print();
        . . .
```

The template above is one possible means in starting your project. Good luck