Homework Statement:

In this homework you are expected to implement a database that stores student information and grades in an array. The access to the database is managed using a Balanced-tree (B-tree) of order 3 data structure.

You are given the following class declarations:

```
class Student
{public:
int studentID;
bool record_valid;
int index;
Student (int ID=0, bool valid=false, int i=-1);//constructor
};

class StudentRecord
{public:
bool valid;
int studentID;
char studentName [100];
int studentGrades[3];/*array element i stores the grade of
course id i (course ids are selected from 0 to 2).*/
StudentRecord (bool v=false, int ID=0, char* name="", int
grade1=0, int grade2=0, int grade3=0);//constructor};
```

The database *`Student_Record_Database`* is implemented as an array of 100 `StudentRecord` objects. An empty location in the array is indicated by `valid=false`.

A new `StudentRecord` is stored in the database in the first empty location in the array. At the same time a corresponding `Student` object is created and inserted in a B-tree of order 3 . The key for search in *`Student_Info_Tree`* is the `studentID` data member of the `Student` object. Deleting data in *`Student_Info_Tree`* is by setting `record_valid =false`. Efficient search for the first empty location in the *`Student_Record_Database`* array is out of the scope of this homework.

Searching for a record in the database:
`int searchStudent (int ID, BtreeNode<Student>* Tree, bool& valid)` function implements the searching for a record in the database.

When `searchStudent (StudentID, Student_Info_Tree, v)` is called, `studentID` is searched for in *`Student_Info_Tree`* in the minimum possible time.

If a `Student` object is found with a matching `studentID, searchStudent` returns `index` value of the object and sets `v=record_valid` of the object. Note that `v=false` indicates a previously deleted record with `studentID` and the record is not currently stored in *`Student_Record_Database.`*
If no `Student` object is found with a matching `studentID, searchStudent` returns `-1` and sets `v=false.`

Accessing a record in the database:
`void PrintStudent (int ID, StudentRecord* Database, BtreeNode<Student>* Tree)` function implements accessing a record in the database in minimum possible time.

When `PrintStudent (studentID,` *`Student_Record_Database`*`,` *`Student_Info_Tree`*`)` is called:
If the student does not exist in the database some error message is printed on the screen.
Otherwise `studentName` and `studentGrades` are printed on the screen.

`UpdateStudentGrades (int ID, int * grades, StudentRecord* Database, BtreeNode<Student>* Tree)` function implements updating the grades of the student record.

When `UpdateStudentGrades (studentID, newgrades,` *`Student_Record_Database`*`,` *`Student_Info_Tree`*`)` is called:
If the student does not exist in the database some error message is printed on the screen.
Otherwise the `studentGrades` of the student is updated in *`Student_Record_Database`*.

Inserting a record in the database:
`InsertStudent (int ID, char* Name, int* grades, StudentRecord* Database, BtreeNode<Student>* Tree)` function implements inserting a record in the database.
When `InsertStudent (studentID, studentName, studentGrades,` *`Student_Record_Database`*`,` *`Student_Info_Tree`*`)` is called:

If the record already exists in the database, the function returns.

If the record does not exist in the database, find the first empty location in *Student_Record_Database*. Assume that the location you find in *Student_Record_Database* is at index `i`. If there is no empty location you will produce an error message.

You create an object of `StudentRecord` class and write it in location `i` in *Student_Record_Database.*

If the record for this student is inserted for the first time in the database then you create a `Student` object with `index =i,` make the neceesary additional updates in the `Student` object and insert it in *Student_Info_Tree.* If necessary create a new tree node by dynamic memory allocation. If there is previously deleted record in the *Student_Info_Tree* then you make the necessary updates in the record without reinserting it.

Deleting a record in the database:

`DeleteStudent (int ID, StudentRecord* Database, BtreeNode<Student>* Tree)` function implements deleting a record in the database.
When `DeleteStudent (studentID, studentName, studentGrades,` *Student_Record_Database*, *Student_Info_Tree*`)` is called:

If the record does not exist in the database, the function returns.
If the record exists, make the necessary changes in *Student_Record_Database*, *Student_Info_Tree*

Listing the records in the database:
`void List (BtreeNode<Student>* Tree)` function implements the listing of the `studentID's` of all stored student records in the database in ascending order.
When `List (`*Student_Info_Tree*`)` is called, the `studentIDs` of all stored records in *Student_Record_Database* are printed on the screen in ascending order.

`void PrintTree (BtreeNode<Student>* Tree)` function visits all nodes in breadthfirst order.
When `PrintTree (`*Student_Info_Tree*`)` is called, `studentID, record_valid` and if `record valid==true, index` is printed on the screen.

Part 1: Implement the BtreeNode Class defined as follows:
```
template <class T>
class BTreeNode
{
 public:
 BTreeNode<T> * Children[3];
          T data[2];
```

```
      // constructor initializes all children pointers to
null, inserts the items in the data field of the BTreeNode in
the correct order
      BTreeNode (T* items, BTreeNode<T> ** C);
   };
```

Implement the constructors for `Student` and `StudentRecord`.

Part2:

Implement the global functions

- `int searchStudent (int ID, BtreeNode<Student>* Tree, bool& valid)`
- `void PrintStudent (int ID, StudentRecord* Database, BtreeNode<Student>* Tree)`
- `UpdateStudentGrades (int ID, int * grades, StudentRecord* Database, BtreeNode<Student>* Tree)`
- `InsertStudent (int ID, char* Name, int* grades, StudentRecord* Database, BtreeNode<Student>* Tree)`
- `DeleteStudent (int ID, StudentRecord* Database, BtreeNode<Student>* Tree)`
- `void List (BtreeNode<Student>* Tree)`
- `void PrintTree (BtreeNode<Student>* Tree)`