```cpp
/*************** PROGRAM IDENTIFICATION*****************************
//*                                                                  *
//* PROGRAM FILE NAME: Source.cpp Assisgnment #:6       Grade:_______*
//*                                                                  *
//* PROGRAM AUTHOR                        ____________________       *
//*                                          Rishika Swarnkar        *
//*                                                                  *
//*COURSE: CSC 36000 1                     DATE:April 22,2018        *
//*                                                                  *
/*******************************************************************/
//*************** PROGRAM DESCRIPTION ********************************
//*                                                                   *
//*  PROCESS:This program is desgined to read Information and make a  *
//* Binary Search tree baased on Id number of the inventory list.     *
//* It outputs various messsages based on the function performed on   *
//* the inventory.                                                     *
//*                                                                   *
//* USER DEFINED                                                      *
//* MODULES : newNode - Makes a new node                               *
//*         pageBreak -Insert Lines to Break the page                  *
//*         printInventory - Prints the Inventory                      *
//*         Header – Prints output Preamble                            *
//*         Footer – Prints output Footer                              *
//*         insert - Insert a node into the tree                       *
//*   updateSales  - Change on Hand Quantity                           *
//* updateStockOnHand - Change on Order Quantity                       *
//* updateRestock -Change OnOrder and In Hand Inventory                *
//* printNode  - Print the requested Node                              *
//*  minValueNode- Returns the min value in the tree                   *
//*********************************************************************/
#include<iostream>
#include<string>
#include<iomanip>
#include<fstream>
using namespace std;
int lineCount = 0;
struct Node
{
    //saves all the infromation about the items
    Node *left, *right, *parent;
    string id;
    string name;
    int quantityOnHand;
    int quantityOnOrder;
};
/***********************************/
/*******FUNCTION DECLARATIONS********/
Node *newNode(Node input);
Node *insert(Node *node, Node input, ofstream &fout);
```

```
50  void printInventory(Node *root, ofstream &fout);
51  void printInventory(Node *root, ofstream & fout);
52  Node * newNode(Node input);
53  void pageBreak(ofstream &fout, int & printedLine);
54  void updateSales(struct Node* root, string id, int sales, ofstream &fout);
55  void updateStockOnHand(struct Node* root, string id, int order, ofstream
      &fout);
56  void updateRestock(struct Node* root, string id, int orderQuan, ofstream
      &fout);
57  void printNode(struct Node* root, string id, ofstream &fout);
58  void Header(ofstream& fout);
59  void Footer(ofstream& fout);
60  void resetLinecounter(int lineCount);
61  struct Node * minValueNode(struct Node* nodePtr);
62  struct Node* deleteNode(struct Node* root, string id, ofstream &fout);
63  /*********************************/
64  Node *insert(Node *nodePtr, Node input, ofstream &fout)
65  {
66      //Receives - A Node pointer , A Node structure , the outfile file
67      //Task     - Inserts the node in The Binary Search Tree
68      //Returns  - A Node pointer
69      Node *leaf;
70      Node *currNode, *parNode;
71      bool found = false; // Check if the Node already exists
72      // Set the pointers to start at the root
73      currNode = nodePtr;
74      parNode = NULL;
75      while ((found == false) && (currNode != NULL))
76      {        // Set flag to true if we find the node
77          if (input.id == currNode->id)
78          {
79              found = true;
80              fout << "ERROR - Attempt to insert a duplicate item (<# "<<
                  input.id << "> ) into the database." << endl;
81              fout <<
                  "-------------------------------------------------------------" <<
                   endl;
82              lineCount += 2;
83          }
84          else
85          {
86              parNode = currNode;
87              // move down the appropriate branch of the tree
88              if (input.id < currNode->id)
89                  currNode = currNode->left;
90              else
91                  currNode = currNode->right;
92          }
93      }
```

```
94          /* If the tree is empty, return a new node */
95          if (nodePtr == NULL)
96          {
97              leaf = newNode(input);
98              fout << "Item ID Number < #" << leaf->id << "> successfully
                  entered into database." << endl;
99              fout <<
                  "------------------------------------------------------------" <<
                   endl;
100             lineCount += 2;
101             return leaf;
102         }
103         /* Otherwise, recur down the tree */
104         if (input.id < nodePtr->id)
105         {
106             nodePtr->left = insert(nodePtr->left, input, fout);
107             nodePtr->left->parent = nodePtr;
108         }
109         else if (input.id> nodePtr->id)
110         {
111             nodePtr->right = insert(nodePtr->right, input, fout);
112             nodePtr->right->parent = nodePtr;
113         }
114     return nodePtr;// root pointer
115 }
116 /**********************************/
117 void printInventory(Node *root, ofstream & fout)
118 {
119     //Receives - Node pointer and the outfile file
120     //Task     - Prints the Entire Inventory based on the key
121     //Returns  - Nothing
122     bool leftdone = false; // set flag for left print
123     // Start traversal from root
124     while (root)
125     {
126         if (!leftdone)
127         {
128             // If left child is not traversed, find the leftmost child
129             while (root->left)
130                 root = root->left;
131         }
132         // Print root's data
133         fout << setw(10) << root->id << setw(25) << root->name << setw(15) <<
              root->quantityOnHand;
134         fout << setw(10) << root->quantityOnOrder;
135         fout << endl;
136         lineCount++;
137         // Mark left as done
138         leftdone = true;
```

```
139         if (root->right)
140         {
141             leftdone = false;
142             root = root->right;
143         }
144         // If right child doesn't exist, move to parent
145         else if (root->parent)
146         {
147             // If this node is right child of its parent,
148             // visit parent's parent first
149             while (root->parent &&
150                 root == root->parent->right)
151                 root = root->parent;
152             if (!root->parent)
153                 break;
154             root = root->parent;
155         }
156         else break;
157     }
158 }
159 /*********************************/
160 Node * newNode(Node input)
161 {
162     //Receives - Node structure
163     //Task     - make a new node
164     //Returns  - Node pointer to itself
165     // A utility function to create a new BST node
166     Node *temp = new Node;
167     temp->id = input.id;
168     temp->name = input.name;
169     temp->quantityOnHand = input.quantityOnHand;
170     temp->quantityOnOrder = input.quantityOnOrder;
171     temp->parent = temp->left = temp->right = NULL;
172     return temp;
173 }
174 /***********************************/
175 void pageBreak(ofstream &fout, int & printedLine)
176 {
177     //Receives - Output file and number of lines already printed
178     //Task     - Insert Lines to Break the page
179     //Returns  - Nothing
180     for (int i = 0; i < 50 - printedLine; i++)
181     {
182         fout << endl;
183     }
184     return;
185 }
186 /************************************/
187 void updateSales(struct Node* root, string id, int sales, ofstream &fout)
```

```
188 {
189     //Receives - A Node pointer , id to be updated , quantity sold, the
          outfile file
190     //Task     - Update Quantity on hand
191     //Returns   - Nothing
192     //look if the node exist in the tree
193     Node *currNode, *parnode;
194    // Declare a flag to indicate  the node to be deleted  is found
195     bool found = false;
196     // Set the pointers to start at the root
197     currNode = root;
198     parnode = NULL;
199     // Search the tree until we find the node to be deleted or until there
200     //  are no more nodes to examine
201     while ((found == false) && (currNode != NULL))
202     {        // Set flag to true if we find the node
203         if (id == currNode->id)
204         {
205             found = true;
206             //update the Quantity on hand due to sales
207             currNode->quantityOnHand = currNode->quantityOnHand -  sales;
208             fout << "Quantity on Hand for item (<#" << id << "> ) successfully
                    updated." << endl;
209             fout <<
                   "------------------------------------------------------------" <<
                    endl;
210             lineCount += 2;
211             return;
212         }
213         else     // Otherwise keep track of the parent node and move down
214                  // the appropriate branch of the tree
215         {
216             parnode = currNode;
217             if (id < currNode->id)
218                 currNode = currNode->left;
219             else
220                 currNode = currNode->right;
221         }
222     }
223     if (found == false)
224     {
225         fout << "Item (<#" << id << ">) not in database.  Data not updated."
              << endl;
226         fout << "------------------------------------------------------------"
              << endl;
227         lineCount += 2;
228         return;
229     }
230 }
```

```
231 /***********************************/
232 void updateStockOnHand(struct Node* root, string id, int order , ofstream
      &fout)
233 {
234     //Receives - A Node pointer , id to be updated, quantity ordered , the
        outfile file
235     //Task     - Update Quantity On order
236     //Returns   - Nothing
237     //look if the node exist in the tree
238     Node *currNode, *parnode;// , *node1, *node2, *node3;
239                              // Declare a flag to indicate the node to be
                        deleted  is found
240     bool found = false;
241     // Set the pointers to start at the root
242     currNode = root;
243     parnode = NULL;
244     // Search the tree until we find the node to be deleted or until there
245     //  are no more nodes to examine
246     while ((found == false) && (currNode != NULL))
247     {        // Set flag to true if we find the node
248         if (id == currNode->id)
249         {
250             found = true;
251             //update the Quantity on hand due to sales
252             currNode->quantityOnOrder= currNode->quantityOnOrder -  order;
253             currNode->quantityOnHand = currNode->quantityOnHand +  order;
254             fout << "Quantity on Hand and Quantity on Order for item (<#" <<
                id << "> ) successfully updated." << endl;
255             fout <<
                "------------------------------------------------------------" <<
                 endl;
256             lineCount += 2;
257             return ;
258         }
259         else     // Otherwise keep track of the parent node and move down
260                  // the appropriate branch of the tree
261         {
262             parnode = currNode;
263             if (id < currNode->id)
264                 currNode = currNode->left;
265             else
266                 currNode = currNode->right;
267         }
268     }
269     if (found == false)
270     {
271         fout << "Item (<#" << id << ">) not in database.  Data not updated."
              << endl;
272         fout << "------------------------------------------------------------"
```

```
              << endl;
273           lineCount += 2;
274           return ;
275       }
276   }
277   /*************************************************/
278   void updateRestock(struct Node* root, string id, int orderQuan, ofstream
        &fout)
279   {
280       ///Receives - A Node pointer , id to be updated, ordered quantity , the
            outfile file
281       //Task     - Update Quantity on hand
282       //Returns   - Nothing
283       //look if the node exist in the tree
284       Node *currNode, *parnode;// , *node1, *node2, *node3;
285                                // Declare a flag to indicate the node to be
                        deleted  is found
286       bool found = false;
287       // Set the pointers to start at the root
288       currNode = root;
289       parnode = NULL;
290       // Search the tree until we find the node to be deleted or until there
291       //  are no more nodes to examine
292       while ((found == false) && (currNode != NULL))
293       {        // Set flag to true if we find the node
294           if (id == currNode->id)
295           {
296               found = true;
297               //update the Quantity on hand due to sales
298               currNode->quantityOnHand = currNode->quantityOnHand + orderQuan;
299               fout << "Quantity on Order for item ( <#" << id << "> )
                    successfully updated." << endl;
300               fout <<
                    "------------------------------------------------------------" <<
                     endl;
301               lineCount += 2;
302               return ;
303           }
304           else     // Otherwise keep track of the parent node and move down
305                    // the appropriate branch of the tree
306           {
307               parnode = currNode;
308               if (id < currNode->id)
309                   currNode = currNode->left;
310               else
311                   currNode = currNode->right;
312           }
313       }
314       if (found == false)
```

```
315     {
316         fout << "Item (<#" << id << ">) not in database.  Data not updated."
              << endl;
317         fout << "-------------------------------------------------------- "
              << endl;
318         lineCount += 2;
319         return ;
320     }
321 }
322 /***********************************/
323 void printNode(struct Node* root, string id, ofstream &fout)
324 {
325     //Receives - A Node pointer , id to be updated , the outfile file
326     //Task     - Print the Node
327     //Returns  - Nothing
328     //look if the node exist in the tree
329     Node *currNode, *parnode;// , *node1, *node2, *node3;
330                              // Declare a flag to indicate the node to be
                      deleted  is found
331     bool found = false;
332     // Set the pointers to start at the root
333     currNode = root;
334     parnode = NULL;
335     // Search the tree until we find the node to be deleted or until there
336     //  are no more nodes to examine
337     while ((found == false) && (currNode != NULL))
338     {        // Set flag to true if we find the node
339         if (id == currNode->id)
340         {
341             found = true;
342             fout << " JAKE'S HARDWARE INVENTORY REPORT " << endl;
343             fout << setw(10) << "Item" << setw(25) << "Item " << setw(10) <<
                "Quantity" << setw(10);
344             fout << "Quantity" << setw(10);
345             fout << endl;
346             fout << setw(10) << "ID Number" << setw(25) << "Description" <<
                setw(10) << "on hand" << setw(10);
347             fout << "on order" << endl;
348             fout <<
                "-------------------------------------------------------" <<
                 endl;
349             lineCount += 4;
350             fout << setw(10) << currNode->id << setw(25) << currNode->name <<
                setw(15) << currNode->quantityOnHand;
351             fout << setw(10) << currNode->quantityOnOrder;// r << setw(10) <<
                root-> << setw(10) << Inventory[i].sp;
352             fout << endl;
353             fout <<
                "-------------------------------------------------------" <<
```

```
                    endl;
354             lineCount += 2;
355             return;
356         }
357         else     // Otherwise keep track of the parent node and move down
358                  // the appropriate branch of the tree
359         {
360             parnode = currNode;
361             if (id < currNode->id)
362                 currNode = currNode->left;
363             else
364                 currNode = currNode->right;
365         }
366     }
367     if (found == false)
368     {
369         fout << "Item (<#" << id << ">) not in database to print." << endl;
370         fout << "------------------------------------------------------------" ↵
              << endl;
371         lineCount += 2;
372         return;
373     }
374 }
375 /***********************************/
376 void Header(ofstream& fout)
377 {
378     //Receives - the outfile file
379     //Task     - Prints the output preamble
380     //Returns  - Nothing
381     fout << setw(30) << "Rishika Swarnkar";
382     fout << setw(17) << "CSC 36000";
383     fout << setw(15) << "Section 11" << endl;
384     fout << setw(30) << "Spring 2018";
385     fout << setw(17) << "Assignment #5" << endl;
386     fout << setw(35) << "-------------------------------------";
387     fout << setw(35) << "-------------------------------------" << endl << ↵
          endl;
388     lineCount += 4;
389     return;
390 }
391 /************ END OF FUNCTION HEADER *****************/
392 //************************************************/
393 /*************  FUNCTION FOOTER **********************/
394 void Footer(ofstream& fout)
395 {
396     //Receives - the outfile file
397     //Task     - Prints the output preamble
398     //Returns  - Nothing
399     fout << endl;
```

```
400     fout << setw(35) << "------------------------------" << endl;
401     fout << setw(35) << "|     END OF PROGRAM OUTPUT    |" << endl;
402     fout << setw(35) << "------------------------------" << endl;
403     return;
404 }
405 /******* END OF FUNCTION FOOTER***********/
406 /***************************************/
407 struct Node * minValueNode(struct Node* nodePtr)
408 {
409     //Receives - Node Pointer
410     //Task     - Look for the Minimum Value in the tree
411     //Returns  - A node pointer
412     struct Node* current = nodePtr;
413     /* loop down to find the leftmost leaf */
414     while (current->left != NULL)
415         current = current->left;
416     return current;
417 }
418 /***********************************/
419 struct Node* deleteNode(struct Node* root,string id, ofstream &fout)// int
      key)
420 {
421     //Receives -  A Node pointer , id to be updated , the outfile file
422     //Task     - Checks if the requested id is there to delete,
423     //           deletes the item if it exist
424     //Returns  - New Node
425     //look if the node exist in the tree
426     Node *delnode, *parnode;//
427    // Declare a flag to indicate the node to be deleted  is found
428     bool found = false;
429     // Set the pointers to start at the root
430     delnode = root;
431     parnode = NULL;
432     while ((found == false) && (delnode != NULL))
433     {        // Set flag to true if we find the node
434         if (id == delnode->id)
435         {
436             found = true;
437         }
438         else     // Otherwise keep track of the parent node and move down
439                  // the appropriate branch of the tree
440         {
441             parnode = delnode;
442             if (id < delnode->id)
443                 delnode = delnode->left;
444             else
445                 delnode = delnode->right;
446         }
447     }
```

```
448     if (found == false)
449     {
450         cout << "Node is not in the tree !"<< endl;
451         fout << "ERROR --- Attempt to delete an item (<#"<<id<<"> ) not in the
              database list." << endl;
452         fout << "------------------------------------------------------------"
              << endl;
453         lineCount += 2;
454         return root;
455     }
456
457     // base case
458     if (root == NULL) return root;
459     // If the key to be deleted is smaller than the root's key,
460     // then it lies in left subtree
461     if (id < root->id)
462         root->left = deleteNode(root->left, id, fout);
463     // If the key to be deleted is greater than the root's key,
464     // then it lies in right subtree
465     else if (id > root->id)//(key > root->key)
466         root->right = deleteNode(root->right, id, fout);// key);
467     // if key is same as root's key, then This is the node
468     // to be deleted
469     else if (id == root->id)
470     {
471         struct Node *temp;
472         // node with only one child or no child
473         if (root->left == NULL)
474         {
475             temp = root->right;
476             lineCount += 2;
477         }
478         else
479         {
480             temp = root->left;
481             lineCount += 2;
482         }
483         return temp;
484         fout << "Item ID Number (<#" << id << "> ) successfully deleted from
              database." << endl;
485         fout << "------------------------------------------------------------"
              << endl;
486         // node with two children: Get the inorder successor (smallest in the
              right subtree)
487         temp = minValueNode(root->right);
488         // Copy the inorder successor's content to this node
489         root->id = temp->id;
490         root->name = temp->name;
491         root->quantityOnHand = temp->quantityOnHand;
```

```
492         root->quantityOnOrder = temp->quantityOnOrder;
493
494         // Delete the inorder successor
495         root->right = deleteNode(root->right, temp->id, fout);
496
497     }
498     return root;
499 }
500 /*********************/
501 int main()
502 {
503     char key; // key to determine the operation to the inventory
504     Node * root = NULL;
505     ifstream fin;
506     ofstream fout;
507     fout.open("Inventory.txt");// open input file
508     string idprint;
509     fin.open("tree_in.txt");// open output file
510     fin >> key;
511     Header(fout); // display function Header
512     Node inputData;
513     do
514     {
515         switch (key)
516         {
517         case 'I':
518             // Reads the information from file to a Node
519             fin >> inputData.id;
520             fin >> ws;
521             getline(fin, inputData.name);
522             fin >> inputData.quantityOnHand;
523             fin >> inputData.quantityOnOrder;
524             // Insert the node in the tree
525             root = insert(root, inputData, fout);
526             break;
527         case 'P':
528             char c;
529             fin >> c;
530             // Print on a new page
531             pageBreak(fout,lineCount);
532             lineCount = 0;// reset lineCoounter
533             fout << left;
534             if (c == 'E')
535             {
536                 fout << " JAKE'S HARDWARE INVENTORY REPORT " << endl;
537                 fout << setw(10) << "Item" << setw(25) << "Item " << setw(10) ↵
                      << "Quantity" << setw(10);
538                 fout << "Quantity" << setw(10);
539                 fout << endl;
```

```
540                 fout << setw(10) << "ID Number" << setw(25) << "Description"
                      << setw(10) << "on hand" << setw(10);
541                 fout << "on order" << endl;
542                 fout <<
                      "------------------------------------------------------------
                      " << endl;
543                 lineCount +=3;
544                 printInventory(root, fout);
545                 fout <<
                      "------------------------------------------------------------
                      " << endl;
546                 lineCount += 1;
547             }
548             else if (c=='N')
549             {
550                 //Print just one node of the tree
551                 fin >> ws;
552                 getline(fin, inputData.id);
553                 idprint = inputData.id;// id to be printed
554                 printNode(root, idprint, fout);
555             }
556             break;
557         case 'D':
558             // Read the id to Delete
559             fin >> inputData.id;
560             fin >> ws;
561             getline(fin, inputData.name);
562             // Delete Node
563             deleteNode(root, inputData.id, fout);
564             break;
565         case 'S':
566                 int sales;
567                 fin >> inputData.id;
568                 fin >> sales;
569                 // Update the inventory when sale occurs
570                 updateSales(root,inputData.id,sales,fout);
571                 break;
572         case 'O':
573             int quantityOrdered;
574             fin >> inputData.id;
575             fin >> quantityOrdered;
576             // Update the inventory when quantity is ordered
577             updateRestock(root, inputData.id, quantityOrdered, fout);
578             break;
579         case 'R':
580             int order;
581             fin >> inputData.id;
582             fin >> order;
583             // Update stock when order is received
```

```
584             updateStockOnHand(root, inputData.id, order, fout);
585             break;
586         default:
587             cout << "Error Reading File";
588             break;
589         }
590         fin >> key;
591     }
592     while (key != 'Q');// Indicated end of input file
593     Footer(fout);// Ouput Footer
594     fin.close();// close input file
595     fout.close();// close output file
596     system("pause");
597     return 0;
598 }
```