Programkod

BST.cpp

```
#include "BST.h"
#include <cstring>
#include <iostream>
#include <fstream>


BST::BST()
{

  root = 0;
  size = 0;

}

BST::~BST()
{



}

bool BST::exists(int element)
{
  return existsPr(root, element);
}

bool BST::existsPr(TreeNode * n, int element)
{

  if(n != 0)
    {
      if(n->data == element)
        return true;
      else if(n->data > element)
        return existsPr(n->leftChild, element);
      else
        return existsPr(n->rightChild, element);

    }
  else
    {
      return false;
    }
}


void BST::insertPr(TreeNode * & n, int element)
{
  if(n==0)
    {
```

```
      TreeNode * newNode = new TreeNode;
      newNode -> data = element;
      newNode -> leftChild = 0;
      newNode -> rightChild = 0;
      n = newNode;
    }
  else if(element < n -> data)
    insertPr(n->leftChild, element);
  else
    insertPr(n->rightChild, element);
}

void BST::insert(int element)
{
  insertPr(root, element);
  size++;
}

void BST::printPreOrder()
{

  printPreOrderPr(root);

}

void BST::printPreOrderPr(TreeNode * n)
{
  using namespace std;

  if(n!=0)
    {
      cout << n->data << ",";
      printPreOrderPr(n->leftChild);
      printPreOrderPr(n->rightChild);

    }


}

void BST::printInOrderPr(TreeNode * n)
{

  if(n!=0)
    {

      printInOrderPr(n->leftChild);
      std::cout << n->data << ",";
      printInOrderPr(n->rightChild);


    }

}

void BST::printInOrder()
{
```

```
  printInOrderPr(root);

}

void BST::printPostOrderPr(TreeNode * n)
{

  if(n!=0)
    {

      printPostOrderPr(n->leftChild);
      printPostOrderPr(n->rightChild);
      std::cout << n->data << ",";

    }

}

void BST::printPostOrder()
{

  printPostOrderPr(root);
}

void BST::saveToFile(char filename[])
{

  int array[size];
  int talet = 0;

  printToFile(root, array, talet);

  ofstream skrivFil(filename, ios::out);
  if(!skrivFil)
    cerr << endl << "Det gick inte öppna filen för att spara data";
  else
    {

      for(int i = 0; i<size; i++)
        {
          skrivFil.write(reinterpret_cast<const char * > (&array[i]),
sizeof(talet));
        }

        skrivFil.close();
    }
}

void BST::printToFile(TreeNode * n, int arr[], int &i)
{

  using namespace std;


  if(n!=0)
    {
```

```
      arr[i] = n->data;
      i++;
      printToFile(n->leftChild, arr, i);
      printToFile(n->rightChild, arr, i);

    }

}

bool BST::openFromFile(char filename[])
{
  int talet = 0;
  int i =0;
  bool bra;

  ifstream lasFil(filename, ios::in);
  if(!lasFil)
    bra = false;
  else
    {
      while(lasFil.peek()!= EOF)
        {
          lasFil.read(reinterpret_cast<char * > (&talet), sizeof(talet));
          insert(talet);
          i++;
          lasFil.seekg((i)*sizeof(talet));

        }
      lasFil.close();
      bra = true;
    }

  return bra;

}

void BST::emptyTreePr(TreeNode *& n)
{

  if(n!=0)
    {

      emptyTreePr(n->leftChild);
      emptyTreePr(n->rightChild);
      delete n;
      n = 0;
      size--;

    }

}

void BST::emptyTree()
{

  emptyTreePr(root);
}
```

```
int BST::numEl()
{

  return size;

}

void BST::removeValue(int value)
{

  removeValuePr(root, value);

}


void BST::removeValuePr(TreeNode *& n, int value)
{

  if(n!=0)
    {
      if(value < n->data)
        removeValuePr(n->leftChild, value);
      else if(value > n->data)
        removeValuePr(n->rightChild, value);
      else if(value == n->data)
        {
          if((n->leftChild == 0) && (n->rightChild == 0))
            {
              delete n;
              size--;
              n = 0;
            }
          else if(n->rightChild == 0)
            {
              TreeNode * temp = n;
              n = n->leftChild;
              delete temp;
              temp = 0;
            }
          else if(n->leftChild == 0)
            {
              TreeNode * temp = n;
              n = n->rightChild;
              delete temp;
              temp = 0;
            }
          else
            {
              int temp = n->rightChild->data;
              findSmallPr(n->rightChild, temp);
              n->data = temp;
              removeValuePr(n->rightChild, temp);
            }
```

```
        }
    }

}

void BST::findSmallPr(TreeNode * n, int & aktuell)
{

  if(n != 0)
    {
      if(aktuell > n->data)
        aktuell = n->data;

       findSmallPr(n->leftChild, aktuell);
    }

}
```

BST.h

```
#ifndef BST_CLASS_12344321
#define BST_CLASS_12344321

#include "TreeNode.h"
#include <cstring>
#include <fstream>
using namespace std;

class BST
{
 public:

  BST();
  ~BST();

  //Lägger till ett elemnt i trädet
  //Pre:  !exists(element)
  //Post: Elementet tillagt i trädet
  void insert(int element);

  //Tar bort ett värde ur trädet
  //Pre:  Värdet element finns i trädet
  //Post: Värdet/Elemetet har raderats från trädet
  void removeValue(int element);

  //Kontrollerar ifall ett element finns
  //Pre:  True
  //Post: Returnerat true ifall element finns annars elese
  bool exists(int element);

  //Skriver trädet till skrämen i PreOrder
  //Pre:  True
  //Post: Trädet har skrivits till skrämen
  void printPreOrder();

  //Skriver trädet till skrämen i PostOrder
```

```
  //Pre:  True
  //Post: Trädet har skrivits till skrämen
  void printPostOrder();

  //Skriver trädet till skrämen i InOrder
  //Pre:  True
  //Post: Trädet har skrivits till skrämen
  void printInOrder();

  //Sprarar trädet till fil
  //Pre:  strlen(filename[]) > 0
  //Post: Trädet har sparats till filen filename i aktuell path
  void saveToFile(char filename[]);

  //Öppnar ett träd från fil
  //Pre:  True
  //Post: Returnerat true om trädet lästs in, annars false
  bool openFromFile(char filename[]);

  //Tömmer trädet
  //Pre:  True
  //Post: Trädet har tömts
  void emptyTree();

  //Retunerar antalet element i trädet
  //Pre:  True
  //Post: Antalet element har returnerats
  int numEl();

 private:

  void insertPr(TreeNode * & n, int element);
  void removeValuePr(TreeNode *& n, int value);
  bool existsPr(TreeNode * n, int element);
  void printPreOrderPr(TreeNode * n);
  void printPostOrderPr(TreeNode * n);
  void printInOrderPr(TreeNode * n);
  void printToFile(TreeNode * n, int arr[], int & i);
  void emptyTreePr(TreeNode *& n);
  void findSmallPr(TreeNode * n, int & aktuell);

  TreeNode* root;
  int size;

};

#endif
```

drivers.cpp

```
#include "drivers.h"

int mataInInt()
{

  char buffer[100];
```

```
  int varde;
  bool felKoll = false;
  do
    {
      if(felKoll == true)
        cout << "Felaktigt värde! Försök igen: ";
      cin.getline(buffer, 100);
      if(buffer[0]=='0')
        return 0;
          felKoll = true;
    }while(!(varde=atoi(buffer)));
  return varde;
}

void laggTill(BST &soktrad)
{

  using namespace std;

  int varde = 0;

  cout << endl << "Lägga till TAL i träd" << endl
       << "====================" << endl << endl;

  cout << "Ange tal att lägga till: " ;
  varde = mataInInt();

  while(soktrad.exists(varde))
    {

      cout << "Värdet fanns redan, dubletter ej tillåtna, ange nytt värde: ";
      varde = mataInInt();
    }

  soktrad.insert(varde);

  cout << "Värdet " << varde << " är adderat" << endl << endl << endl;

}

void soekVarde(BST &soktrad)
{
  using namespace std;

  int varde = 0;

  cout << endl << endl << "Sök efter ett värde i trädet" << endl
       << "============================" << endl << endl
       << "Ange ett värde att söka efter: ";

  varde = mataInInt();

  if(soktrad.exists(varde))
    cout << endl << "Värdet " << varde << " finns i trädet" << endl;
  else
    cout << endl << "Värdet " << varde << " finns INTE i trädet" << endl;
```

```
}

void toemTrad(BST &soktrad)
{

  using namespace std;
  char grejj;

  cout << endl << endl << "Töm träd" << endl
       << "========" << endl << endl;

    soktrad.emptyTree();

  cout << "Trädet har nu tömts!" << endl << endl;


}

void oppna(BST &soktrad)
{

  using namespace std;

  char filnamn[100];

  cout << endl << endl
       << "Öppna träd från fil" << endl
       << "===================" << endl << endl
       << "Ange filnamn för trädfil, (inkl. filextension): ";

  cin.getline(filnamn,99);

  soktrad.saveToFile("tempfile.tmp");

  soktrad.emptyTree();



      if(soktrad.openFromFile(filnamn))
        cout << endl << "Filen " << filnamn << " har lästs in!" << endl << endl;
      else
        {
          cout << endl << "Filen " << filnamn << " finns ej, eller är korrupt!" <<
endl << endl;
          soktrad.openFromFile("tempfile.tmp");
        }

      if( remove( "tempfile.tmp" ) == -1 )
        cout << "Det gick inte radera temporär datafil!" << endl << endl;



}

void spara(BST &soktrad)
```

```
{

  using namespace std;

  char filnamn[100];

  cout << endl << endl
       << "Spara träd till fil" << endl
       << "===================" << endl << endl;

  do
    {
      cout  << "Ange filnamn för trädfil, (inkl. filextension): ";
      cin.getline(filnamn, 99);

    }while(strlen(filnamn) < 1);

  soktrad.saveToFile(filnamn);

  cout << endl << "Filen " << filnamn << " med aktuellt träd har sparats" << endl
<< endl;

}

void skrivPre(BST &soktrad)
{

  using namespace std;

  cout << endl << endl
       << "Skriv träd i PreOrder" << endl
       << "=====================" << endl << endl;

    soktrad.printPreOrder();

  cout << endl << endl;

}

void skrivIn(BST &soktrad)
{

  using namespace std;

  cout << endl << endl
       << "Skriv träd i InOrder" << endl
       << "====================" << endl << endl;

    soktrad.printInOrder();

  cout << endl << endl;

}

void skrivPost(BST &soktrad)
{
```

```
  using namespace std;

  cout << endl << endl
       << "Skriv träd i PostOrder" << endl
       << "======================" << endl << endl;

    soktrad.printPostOrder();

  cout << endl << endl;

}


void check(BST &soktrad)
{

  using namespace std;

  cout << endl << endl
       << "Trädstatus" << endl
       << "==========" << endl << endl;

  cout << "Trädet innehåller " << soktrad.numEl() << " element";

  cout << endl << endl;

}

void taBoert(BST & soktrad)
{

 using namespace std;

 int tabort;

 cout << endl << endl
      << "Ta bort värde från träd" << endl
      << "======================" << endl << endl;

 if(!(soktrad.numEl() == 0))
   {

     do
       {
         cout << "Ange vilket värde du vill ta bort:";
         tabort = mataInInt();
       }while(!(soktrad.exists(tabort)));

     soktrad.removeValue(tabort);

     cout << endl << endl << tabort << " har raderats från trädet!";

     cout << endl << endl;
   }
 else
   cout << "Inga värden!" << endl << endl;
```

```
}
```

drivers.h

```
#include "BST.h"
#include <iostream>
#include <cstring>

//Returnerar inmatat heltal
//Pre:  True
//Post: Returnerat inmatat haltal
int mataInInt();

//Lägger till heltal i träd
//Pre:  True
//Post: Ett heltal har lagts till
void laggTill(BST &soktrad);

//Söker ifall ett visst värde finns i trädet
//Pre:  True
//Post: Talat om, om ett tal finns i trädet
void soekVarde(BST &soktrad);

//Tömmer trädet
//Pre:  True
//Post: Trädet har tömts
void toemTrad(BST &soktrad);

//Öppnar en fil och läser in dess innehåll till trädet
//Pre:  True
//Post: Trädet är fyllt med värden från filen
void oppna(BST &soktrad);

//Sparar trädet till fil
//Pre:  True
//Post: Trädet har sparats till fil
void spara(BST &soktrad);

//Skriver ut trädet till skrämen i PreOrder
//Pre:  True
//Post: Trädet utskrivet på skrämen
void skrivPre(BST &soktrad);

//Skriver ut trädet till skrämen i InOrder
//Pre:  True
//Post: Trädet utskrivet på skrämen
void skrivIn(BST &soktrad);

//Skriver ut trädet till skrämen i PostOrder
//Pre:  True
//Post: Trädet utskrivet på skrämen
void skrivPost(BST &soktrad);

//Skriver ut antalet element i trädet
//Pre:  True
//Post: Antalet element utskrivet på skrämen
```

```
void check(BST &soktrad);

//Tar bort ett värde från trädet
//Pre:  True
//Post: Värdet har tagits bort från trädet om trädet inte är tomt.
void taBoert(BST &soktrad);
```

main.cpp

```
#include <iostream>
#include "BST.h"
#include "drivers.h"


int meny()
{

  using namespace std;

  cout << "  Meny" << endl
       << "========" << endl << endl
       << "1. Lägg till heltal i träd" << endl
       << "2. Ta bort ett heltal från träd" << endl
       << "3. Sök efter ett heltal i träd"<< endl
       << "4. Töm träd" << endl
       << "5. Kontrollera trädets status" << endl
       << "   Skriv ut" << endl
       << "    6. PreOrder" << endl
       << "    7. InOrder" << endl
       << "    8. PostOrder" << endl
       << "9. Spara träd till fil" << endl
       << "10.Öppna träd från fil" << endl
       << "0. Avsluta" << endl << endl
       << "Ange ditt val: ";

  return mataInInt();
}

int main()
{

  using namespace std;

  cout << "Det söta, lilla trädprogrammet" << endl << endl
       << "      Skrivet av: Mathias Andersson" << endl
       << "                  Henrik Bäck" << endl
       << endl << endl;


  BST trollgurka;
  bool bort = false;

  do
    {
      switch(meny())
        {
```

```
        case 1: laggTill(trollgurka); break;
        case 2: taBoert(trollgurka);break;
        case 3: soekVarde(trollgurka); break;
        case 4: toemTrad(trollgurka); break;
        case 5: check(trollgurka); break;
        case 6: skrivPre(trollgurka); break;
        case 7: skrivIn(trollgurka); break;
        case 8: skrivPost(trollgurka); break;
        case 9: spara(trollgurka); break;
        case 10: oppna(trollgurka); break;
        case 0: bort = true; break;
        default: cout << "Felaktigt val" << endl;
        }
    }while(!bort);


}
```

TreeNode.h

```
/*
  class TreeNode represents a Node in a binary search tree.
  Its sole purpose is to serve as a tree node för the class BST.
*/

#if !defined(TREENODE_H___INCLUDED_)
#define TREENODE_H___INCLUDED_

class TreeNode
{
    friend class BST;

    private:
            int data;
            TreeNode* leftChild;
            TreeNode* rightChild;
};

#endif // !defined(TREENODE_H___INCLUDED_)
```

MakeFile

```
#MakeFile för binärt sökträd
affar: main.o drivers.o BST.o
       g++ -o traed main.o drivers.o BST.o

main.o: main.cpp
       g++ -c main.cpp

drivers.o: drivers.cpp drivers.h
       g++ -c drivers.cpp

Person.o: BST.cpp BST.h
       g++ -c Person.cpp
```