Assignment name: Polyset

Expected file: searchable_array_bag.cpp, searchable_array_bag.hpp, searchable_tree_bag.cpp, searchable_tree_bag.hpp, set.cpp, set.hpp

You will find in this directory some classes:
- bag: an abstract class representing a bag
- searchable_bag: an abstract class representing a bag with the ability to search in it.
- array_bag: an implementation of a bag with an array as underlying data structure.
- tree_bag: an implementation of a bag with a binary search tree as underlying data structure.

If you don't know what is a set or a bag (shame) you can read the attached file shame.txt.

First Part =>

Since a bag without a searching function isn't very useful, implement two classes searchable_array_bag and searchable_tree_bag that will inherit from array_bag and tree_bag and implement the searchable bag abstract class.

Second part =>

Implement the class set that will wrap a searchable_bag and turn it into a set.

You will find in this dir a main that must compile with your code. All classes should be under orthodox canonical form. Don't forget the const.

array_bag.hpp

```
class array_bag : virtual public bag {
 Protected:
    int *data;
    int size;
 Public:
   array_bag();
   array_bag(const array_bag &);
   array_bag operator=(const array_bag &other);
  ~ array_bag();
   void insert(int);
   void insert(int*, int);
   void print() const;
   void clear();
};
```

bag.hpp

```
class bag {
  Public:
    virtual void insert(int) =0;
    virtual void insert(int*, int) =0;
    virtual void print() const =0;
    virtual void clear() =0;
};
```

searchable_bag.hpp

```
class searchable_bag : virtual public bag {
 Public:
    virtual bool has(int) const =0;
};
```

tree_bag.hpp

```
class tree_bag : virtual public bag{
 Protected:
   struct node {
     node *l;
     node *r;
     int value;
    }
   node *tree;
 Public:
  tree_bag();
  tree_bag(const tree_bag &);
 ~tree_bag();
 node *extract_tree();
 void set_tree(node *);
 virtual void insert(int);
 virtual void insert(int *array, int size);
 Virtual void print() const;
 virtual void clear();
Private:
  static void destroy_tree(node *);
  static void print_node(node *);
  static node *copy_node(node *);
};
```

main.cpp

```
int main(int ac, char *av) {
  if (ac == 1)
    return 1;
  searchable_bag *t = new searchable_tree_bag;
  searchable_bag *a = new searchable_array_bag;
for (int i = 1; i < ac; i++) {
  t->insert(atoi(av[i]));
  a->insert(atoi(atoi(av[i])));
}
  t->print();
  a->print();
  for (int i = 1; i < ac; ++i)
    std::cout << t->has(atoi(av[i])) << std::endl;
  std::cout << a->has(atoi(av[i])) << std::endl;
  std::cout << t->has(atoi(av[i]) - 1) << std::endl;
  std::cout << a->has(atoi(av[i]) - 1) << std::endl;
}
t->clear();
a->clear();
const searchable_array_bag tmp(static_cast<
  searchable_array_bag &>(*a));
tmp.print();
tmp.has(1);
set sa(*a);
set st(*a);
for(int i = 1; i < ac; i++) {
  st.insert(atoi(av[i]));
  sa.insert(atoi(av[i]));
}
```

```
sa.has(atoi(av[i]));
sa.print();
sa.get_bag().print();
st.print();
sa.clear()
sa.insert((int[]){1,2,3,4,3,4});
```