```
// set.h
// A Set ADT.
// This implementation uses a linked list.

#ifndef SET_H
#define SET_H

#include <iostream>
//#include "set_empty.cpp"
using namespace std;

template <class Element>
class Node
{
        public:
                Element value;
                Node<Element> *next;

                Node(Element item)
                {
                        value = item;
                        next = NULL;
                }
};

template <class Element>
class Set;

template <class Element>
ostream& operator<<(ostream& stream, const Set<Element>& s);

template <class Element>
class Set
{
        public:

                Set();                                  // default constructor
                Set(const Set<Element>& s);             // copy constructor
                ~Set();                                 // destructor

                void insert(const Element& x);          // add x to the set
                void remove(const Element& x);          // remove x from the set
                int cardinality() const;                // returns size of the set
                bool empty() const;                     // returns true if empty, false o/w
                bool contains(const Element& x) const;  // true if x is in set, false o/w

                bool operator==(const Set<Element>& s) const;           // equality operator
                bool operator<=(const Set<Element>& s) const;           // subset operator
                Set<Element> operator+(const Set<Element>& s) const;    // union operator
                Set<Element> operator&(const Set<Element>& s) const;    // intersection operat
or
                Set<Element> operator-(const Set<Element>& s) const;    // difference operator

                Set<Element>& operator=(const Set<Element>& s);         // assignment operator

                string toString() const;     // return a string representation of the set

                // stream insertion operator
                friend ostream& operator<< <Element>(ostream& stream, const Set<Element>& s);

        private:

                Node<Element> *head;
                int length;

                void copy(const Set<Element>& s);   // copy the set s to this set (common code
 called
                                                    //   by the copy constructor and the assig
nment operator)
                void destroy();                     // delete all elements in the set (common
code called
                                                    //   by the destructor and the assignment
operator)
};

#endif
```

```
// set.cpp

#include <sstream>
#include <iostream>
#include "set.h"

using namespace std;

template <class Element>
Set<Element>::Set()
{
   head = NULL;
   length = 0;
}

template <class Element>
void Set<Element>::copy(const Set<Element>& s)
{
  if(s.length == 0){
    head = NULL;
    length = 0;
  }else{
    Node<Element> *stemp = s.head;
    while(stemp != NULL){
      insert(stemp -> value);
      stemp = stemp -> next;
    }
  }
}

template <class Element>
void Set<Element>::destroy()
{
  while(head != NULL){
    Node<Element> *temp = head;
    head = head -> next;
    delete temp;
  }
  head = NULL;
  length = 0;
}

template <class Element>
Set<Element>::Set(const Set<Element>& s)
{
  copy(s);
}

template <class Element>
Set<Element>& Set<Element>::operator=(const Set<Element>& s)
{
  copy(s);
}

template <class Element>
Set<Element>::~Set()
{
  destroy();
}

template <class Element>
bool Set<Element>::empty() const
{
  if(length == 0){
    return true;
  }
  return false;
}

template <class Element>
int Set<Element>::cardinality() const
{
  return length;
}

template <class Element>
bool Set<Element>::contains(const Element& item) const
{
  Node<Element> *temp = head;
  for (int i = 0; i < length; i++){
    if (temp -> value == item){
      return true;
    }
    temp = temp -> next;
  }
  return false;
}

template <class Element>
void Set<Element>::insert(const Element& item)
{
  if (!contains(item)){
    Node<Element>* newNode = new Node<Element>(item);
    newNode->next = head;
    head = newNode;
    length++;
  }
}

template <class Element>
void Set<Element>::remove(const Element& item)
{
  if (contains(item)){
    Node<Element> *temp = head;
    Node<Element> *prev = head;
    while(temp != NULL){
      if(temp->value == item){
        if(head -> value == item){
          head = head -> next;
          delete temp;
          length--;
          return;
        }else{
          prev -> next = temp -> next;
          delete temp;
          length--;
          return;
        }
      }
      prev = temp;
      temp = temp -> next;
    }
  }
}

template <class Element>
```

```
bool Set<Element>::operator<=(const Set<Element>& s) const
{
  if(length >= s.cardinality()){
    Node<Element> *temp = s.head;
    while(temp != NULL){
      if(!contains(temp->value)){
        return false;
      }
      temp = temp -> next;
    }
  }else{
    return true;
  }
  return true;
}

template <class Element>
bool Set<Element>::operator==(const Set<Element>& s) const
{
  if (length != s.cardinality()){
    return false;
  }else{
    Node<Element> *temp = head;
    while(temp != NULL){
      if(!s.contains(temp -> value)){
        return false;
      }
      temp = temp -> next;
    }
  }
  return true;
}

template <class Element>
Set<Element> Set<Element>::operator+(const Set<Element>& s) const
{
  Set<Element> t;
  Node<Element> *temp = head;
  while(temp != NULL){
    t.insert(temp -> value);
    temp = temp -> next;
  }
  Node<Element> *stemp = s.head;
  while(stemp != NULL){
    t.insert(stemp -> value);
    stemp = stemp -> next;
  }
  return t;
}

template <class Element>
Set<Element> Set<Element>::operator&(const Set<Element>& s) const
{
  Set<Element> t;
  Node<Element> *temp = head;
  while(temp != NULL){
    if(s.contains(temp -> value)){
      t.insert(temp -> value);
    }
    temp = temp -> next;
  }
  return t;
}
```

```
template <class Element>
Set<Element> Set<Element>::operator-(const Set<Element>& s) const
{
  Set<Element> t;
  Node<Element> *temp = head;
  while(temp != NULL){
    t.insert(temp -> value);
    temp = temp -> next;
  }
  Node<Element> *stemp = s.head;
  while(stemp != NULL){
    t.remove(stemp -> value);
    stemp = stemp -> next;
  }
  return t;
}

template <class Element>
string Set<Element>::toString() const
{
  stringstream ss;
  if (length == 0){
    ss << "{}";
    return ss.str();
  }
  ss << "{";
  Node<Element> *temp = head;
  while(temp->next != NULL){
    ss << temp -> value << ",";
    temp = temp -> next;
  }
  ss << temp->value << "}";
  return ss.str();
}

template <class Element>
ostream& operator<<(ostream& stream, const Set<Element>& s)
{
   stream << s.toString();
   return stream;
}
```

```
#include <iostream>
#include <stdexcept>
#include <string>
#include "set_empty.cpp"
using namespace std;

int main(){
  Set<int> s;
  s.insert(3);
  s.insert(4);
  s.insert(4);
  cout << "s: " << s << endl;
  cout <<"s cardinality: " << s.cardinality() <<endl;
  if (s.contains(4)){
    cout << "s contains 4: " << endl;
  }
  s.remove(3);
  cout << "s after removing 3: " << s << endl;
  cout << s.contains(3) << endl;
  Set<int> t;
  t.insert(4);
  cout << "t: " << t << endl;
  if(s == t){
    cout << "s = t" << endl;
  }
  t.insert(9);
  cout << "t after inserting 9: " << t << endl;
  if(s <= t){
    cout << "s is in t" << endl;
  }
  s.insert(5);
  cout << "s after inserting 5: " << s << endl;
  Set<int> a = t + s;
  cout << "a is t + s: " << a << endl;
  cout << "now a - s: " << a - s << endl;
  Set<int> b = t & s;
  cout << "b is t & s: " << b << endl;
  Set<int> c;
  c = s;
  cout << "s is equal to c: " << c << endl;
  //cout << "c is a copy of s: " << c << endl;
  Set<int> d;
  if (d.empty()){
    cout << "d is empty: " << endl;
  }

  return 0;
}
```

```
#include <iostream>
#include <fstream>
#include <string>
#include <vector>
#include <stdexcept>
#include "set_empty.cpp"
using namespace std;

int main(){

  string name;
  string party;
  string state;
  string religion;
  string age;

  Set<string> VA;
  Set<string> NY;
  Set<string> MA;
  Set<string> OH;
  Set<string> OtherState;
  Set<string> Episcopalian;
  Set<string> Presbyterian;
  Set<string> Methodist;
  Set<string> OtherReligion;
  Set<string> forties;
  Set<string> fifties;
  Set<string> sixties;
  Set<string> Whig;
  Set<string> Democrat;
  Set<string> Republican;
  Set<string> DR;
  Set<string> OtherParty;

  string fileName = "pres.txt";
  ifstream input(fileName);
  while(!input.eof()){
    getline(input, name, '\t');
    getline(input, party, '\t');
    getline(input, state, '\t');
    getline(input, religion, '\t');
    getline(input, age, '\n');

    if(state == "VA"){
      VA.insert(name);
    }else if(state == "NY"){
      NY.insert(name);
    }else if(state == "MA"){
      MA.insert(name);
    }else if(state == "OH"){
      OH.insert(name);
    }else{
      OtherState.insert(name);
    }

    if(religion == "Episcopalian"){
      Episcopalian.insert(name);
    }else if(religion == "Presbyterian"){
      Presbyterian.insert(name);
    }else if(religion == "Methodist"){
      Methodist.insert(name);
    }else{
      OtherReligion.insert(name);
    }

    if(age.substr(0,1) == "4"){
      forties.insert(name);
    }else if(age.substr(0,1) == "5"){
      fifties.insert(name);
    }else if(age.substr(0,1) == "6"){
      sixties.insert(name);
    }

    if(party == "(W)"){
      Whig.insert(name);
    } else if(party == "(D)"){
      Democrat.insert(name);
    } else if(party == "(R)"){
      Republican.insert(name);
    } else if(party == "(DR)"){
      DR.insert(name);
    }else {
      OtherParty.insert(name);
    }
  }
  input.close();

  Set<string> OHMeth;
  OHMeth = OH & Methodist;
  cout << "Methodist Ohioians: " << OHMeth << endl;

  Set<string> VAEpisWhig;
  VAEpisWhig = VA & Episcopalian;
  VAEpisWhig = VAEpisWhig & Whig;
  cout << "VA, Episcopalian, and Whig: " << VAEpisWhig << endl;

  Set<string> WhigDR;
  WhigDR = Whig + DR;
  cout << "Whig or DR: " << WhigDR << endl;

  cout << "Presidents in 40s: " << forties << endl;
  return 0;
}
```