```
//=================================================================
// Emma Steinman
// September 1, 2017
// set.h
// This is the header file for the set class. It declares methods
// to create and modify sets.
//=================================================================

#include <iostream>
#include <string>
#include <sstream>
using namespace std;

//=================================================================

#ifndef SET_H
#define SET_H

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


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

template <class Element>
class Set
{
public:
                                Set                     (void);
                //default constructor
                                Set                     (const Set<Element> & s);
//copy constructor
                                ~Set            (void);
        //destructor

        void            insert          (const Element & x);
        void            remove          (const Element & x);
        int                     cardinality     (void) const;
        bool            empty           (void) const;
        bool            contains        (const Element & x) const;

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

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

        string          toString        (void) const;

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

private:
        Node<Element> * head;
        int                             length;
        void            copy            (const Set<Element> & s);
        void            destroy         (void);
};


#include "set.cpp"
#endif
```

```
//==============================================================
// Emma Steinman
// September 1, 2017
// set.cpp
// This is the .cpp file for the Set class. It contains
// methods for creating and modifying sets.
//==============================================================

//==============================================================
//default constructor
//==============================================================
template <class Element>
                Set<Element>::Set               (void)
{
        head = NULL;
        length = 0;
}
//==============================================================
//copy constructor
//copies a set from an existing set
//==============================================================
template <class Element>
                Set<Element>::Set               (const Set<Element> & s)
{
        head = NULL;
        length = 0;
        copy(s);

}
//==============================================================
//destructor
//==============================================================
template <class Element>
                Set<Element>::~Set              (void)
{
        destroy();
}

//==============================================================
//toString
//inserts items in a set into a printable string
//==============================================================
template <class Element>
string  Set<Element>::toString  (void) const
{
        stringstream s;
        s << "{";
        Node<Element> * ptr = head;
        for (int i = 0; i < length; i++)
        {
                s << ptr->data;
                if (i+1 < length)
                        s << ", ";
                ptr = ptr->next;
        }
        s << "}";
        return s.str();
}

//==============================================================
//insert
//inserts an item into a set
//==============================================================
template <class Element>
void    Set<Element>::insert    (const Element & x)
{

        if (!contains(x))
        {
                Node<Element> * ptr = new Node<Element>(x);

                if (head == NULL)                                    //if set is empty
                        head = ptr;

                else
                {
                        Node<Element> * qtr = head;     //nonempty set
                        while (qtr->next!= NULL)
                                qtr = qtr->next;
                        qtr->next = ptr;
                }

                length += 1;                                         //increments length
        }

}

//==============================================================
//remove
//removes an item from a set
//==============================================================
template <class Element>
void    Set<Element>::remove    (const Element & x)
{
                if (!contains(x))                       //item already in set
                {
                        cout << "Error: item not in set." << endl;
                        return;
                }

                Node<Element> * ptr = head;
                Node<Element> * qtr = head;
                Node<Element> * rm;

                while (qtr->next->data != x)
                {
                        qtr = qtr->next;                //finds node before one to
                }                                               //be deleted

                ptr = qtr->next;
                rm = qtr->next;                         //node to be deleted
                ptr = ptr->next;                        //next node
                qtr->next = ptr;                        //skips node to be deleted
                delete rm;
                length -= 1;                            //decrements length

}

//==============================================================
//cardinality
//returns the number of items in a set
//==============================================================
template <class Element>
int             Set<Element>::cardinality (void) const
{
        return length;
}

//===============================================================
//empty
//returns a boolean value indicating if the set is empty
//===============================================================
template <class Element>
bool    Set<Element>::empty             (void) const
{
        return (length == 0);
}

//===============================================================
//contains
//returns a boolean value indicating if a set contains an item
//===============================================================
template <class Element>
bool    Set<Element>::contains  (const Element & x) const
{
        Node<Element> * ptr = head;

        while (ptr != NULL)
        {
                if (ptr-> data == x)
                        return true;
                ptr = ptr->next;
        }

        return false;
}

//===============================================================
//operator ==
//returns a boolean value indicating if the two sets are equal
//===============================================================
template <class Element>
bool    Set<Element>::operator==        (const Set<Element> & s) const
{
        if (length != s.length)
                return false;

        else
        {
                Node<Element> * ptr = head;

                while (ptr != NULL)
                {
                        if (!s.contains(ptr->data))
                                return false;
                        ptr = ptr->next;
                }

                return true;
        }
}

//===============================================================
//operator <=
//returns a boolean value indicating if the set is a subset
//of another set
//===============================================================
template <class Element>
bool    Set<Element>::operator<=        (const Set<Element> & s) const
```

```
{
        if (s.length == 0)
                return true;                                    //empty set is always
                                                                        //a subset
        Node<Element> * ptr = head;

        while (ptr != NULL)
        {
                if (!s.contains(ptr->data))
                        return false;

                ptr = ptr->next;
        }

        return true;
}

//==============================================================
//operator +
//returns the union of two sets
//==============================================================
template <class Element>
Set<Element>&   Set<Element>::operator+ (const Set<Element> & s) const
{
        Set<Element> *s1 = new Set();
        Node<Element> * ptr = head;

        while (ptr != NULL)                             //inserts elements from
        {                                                       //first set
                s1->insert(ptr->data);
                ptr = ptr->next;
        }

        Node<Element> * qtr = s.head;
        while (qtr != NULL)                             //inserts elements from
        {                                                       //second set
                s1->insert(qtr->data);
                qtr=qtr->next;
        }

        return *s1;

}

//==============================================================
//operator &
//returns the intersection of two sets
//==============================================================
template <class Element>
Set<Element>&   Set<Element>::operator& (const Set<Element> & s) const
{
        Set<Element> *s1 = new Set();
        Node<Element> * ptr = head;

        while (ptr != NULL)
        {
                if (s.contains(ptr->data))              //inserts values in
                        s1->insert(ptr->data);          //both sets
                ptr = ptr->next;
        }

        return *s1;
}
```

```
//==============================================================
//operator -
//returns the difference of two sets
//==============================================================
template <class Element>
Set<Element>&   Set<Element>::operator- (const Set<Element> & s) const
{
        Set<Element> *s1 = new Set();
        Node<Element> * ptr = head;

        while (ptr != NULL)
        {
                if (!s.contains(ptr->data))             //inserts values not in
                        s1->insert(ptr->data);          //second set
                ptr = ptr->next;
        }

        return *s1;
}


//==============================================================
//operator =
//sets a set equal to an existing set
//==============================================================
template <class Element>
Set<Element>&   Set<Element>::operator= (const Set<Element> & s)
{
        if (length > 0)                                         //clears nonempty set
        {                                                               //before copyi
ng
                Node<Element> * ptr = head;
                Node<Element> * qtr = head;

                while (ptr != NULL)
                {
                        qtr = ptr->next;
                        delete ptr;
                        ptr = qtr;
                }

                head = NULL;
                length = 0;
        }

        this->copy(s);                                          //copies second set
        return *this;
}



//==============================================================
//copy
//copies the set s to this set
//==============================================================
template <class Element>
void                    Set<Element>::copy              (const Set<Element> &s)
{
        Node<Element> *ptr1;
        ptr1 = s.head;
        while (ptr1 != NULL)
        {
                insert(ptr1->data);
                ptr1 = ptr1->next;
        }
}



//==============================================================
//destroy
//deletes items from set and the memory
//==============================================================

template <class Element>
void                    Set<Element>::destroy   (void)
{
        Node<Element> * ptr, * qtr;
        ptr = head;
        qtr = head;

        while (ptr != NULL)
        {
                qtr = ptr -> next;
                delete ptr;
                ptr = qtr;

        }
        delete ptr;
        length = 0;
}
```

```
//==============================================================
//Emma Steinman
//September 2, 2017
//test_set.cpp
//This file contains non-terminal testing for the set class
//==============================================================

#include <iostream>
#include <sstream>
#include <string>
#include <assert.h>
#include "set.h"
using namespace std;

//==============================================================
// tests default constructor
//==============================================================

void test1 (void)
{
        Set<int> s1;
        string str = s1.toString();
        assert(str=="{}");

}

//==============================================================
// tests insert
//==============================================================

void test2 (void)
{
        Set<int> s1;
        s1.insert(1);
        s1.insert(4);
        s1.insert(89);
        s1.insert(3);
        string str = s1.toString();
        assert(str=="{1, 4, 89, 3}");
}

//==============================================================
// tests copy constructor
//==============================================================

void test3 (void)
{
        Set<int> s1;
        s1.insert(4);
        s1.insert(44);
        s1.insert(55);
        Set<int> s2(s1);
        string str = s2.toString();

        assert(str == "{4, 44, 55}");
}

//==============================================================
// tests remove
//==============================================================

void test4 (void)
{
        Set<char> s1;
        s1.insert('s');
        s1.insert('e');
        s1.insert('a');
        s1.insert('g');
        s1.insert('z');

        s1.remove('a');

        string str = s1.toString();
        assert(str == "{s, e, g, z}");
}

//==============================================================
// tests cardinality
//==============================================================

void test5 (void)
{
        Set<int> s1;
        s1.insert(4);
        s1.insert(8);
        s1.insert(12);
        s1.insert(16);
        int length = s1.cardinality();
        assert (length == 4);
}

//==============================================================
// tests ==
//==============================================================
void test6 (void)
{
        Set<char> s1;
        Set<char> s2;
        for (char letter = 'a';letter<='z';letter++)
        {
                s1.insert(letter);
                s2.insert(letter);
        }

        if (s1==s2)
                return;

        else
                cout << "Test 6 failed." << endl;
}
//==============================================================
// tests contains
//==============================================================

void test7 (void)
{
        Set<int> s1;
        for (int i = 0; i < 8; i++)
                s1.insert(i);
        if (s1.contains(3))
                return;
        else
                cout << "Test 7 failed" << endl;
}
```

```
//=============================================================
// tests union
//=============================================================

void test8 (void)
{
        Set<int> s1;
        for (int i = 0; i<6; i++)
                s1.insert(i);
        Set<int> s2;
        for (int i = 6; i<11; i++)
                s2.insert(i);
        Set<int> s3;
        s3.insert(5);
        s3.insert(3);
        s3.insert(6);
        s3 = s1+s2;
        string str = s3.toString();
        assert(str == "{0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10}");
}

//=============================================================
//tests copy to an empty list
//=============================================================

void test9 (void)
{
        Set<int> s1;
        for (int i = 0; i < 6; i++)
                s1.insert(i);
        Set<int> s2(s1);
        string str = s2.toString();
        assert(str == "{0, 1, 2, 3, 4, 5}");
}

//=============================================================
//tests operator =
//=============================================================

void test10     (void)
{
        Set<char> s1;
        for (char a = 'a'; a<='z'; a++)
                s1.insert(a);
        Set<char> s2;
        s2 = s1;
        string str = s2.toString();
        assert(str == "{a, b, c, d, e, f, g, h, i, j, k, l, m, n, o, p, q, r, s, t, u, v, w, x
, y, z}");
}


//=============================================================
//tests operator &
//=============================================================

void test11     (void)
{
        Set<char> s1;
        for (char a = 'a'; a<='e'; a++)
                s1.insert(a);
        Set<char> s2;
        for (char f = 'c'; f<= 'j'; f++)
```

```
                s2.insert(f);
        Set<char> s3;
        s3 = s1 & s2;
        string str = s3.toString();
        assert(str == "{c, d, e}");
}

//==============================================================
//tests remove with item not in list
// TERMINAL
//==============================================================

void test12     (void)
{
        Set<char> s1;
        for (char a = 'a'; a < 'r'; a++)
                s1.insert(a);
        s1.remove('z');


}


//==============================================================
//tests operator -
//==============================================================

void test13     (void)
{
        Set<int> s1;
        Set<int> s2;
        for (int i = 0; i <= 10; i++)
                s1.insert(i);
        for (int i = 0; i <=10; i+=2)
                s2.insert(i);
        Set<int> s3;
        s3 = s1-s2;
        string str = s3.toString();
        assert(str == "{1, 3, 5, 7, 9}");
}

//==============================================================
//tests operator = with items in set previously
//==============================================================

void test14     (void)
{
        Set<int> s1;
        for (int i = 0; i < 10; i++)
                s1.insert(i);
        Set<int> s2;
        for (int j = 0; j < 3; j++)
                s2.insert(j);
        s2 = s1;
        string str = s2.toString();
        assert(str=="{0, 1, 2, 3, 4, 5, 6, 7, 8, 9}");
}

//==============================================================
//tests operator<=
//==============================================================
void test15     (void)
{
        Set<int> s1;
        for (int i = 0; i < 10; i++)
                s1.insert(i);
        Set<int> s2;
        for (int j = 0; j < 3; j++)
                s2.insert(j);
        if (s2 <= s1)
                return;
        else
                cout << "test 15 failed" << endl;
}

//===============================================================
//tests insert with existing item
//TERMINAL
//===============================================================

void test16     (void)
{
        Set<int> s1;
        for (int i = 0; i < 10; i++)
                s1.insert(i);
        for (int j = 5; j < 15; j++)
                s1.insert(j);
        string str = s1.toString();
        assert(str=="{0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14}");
}

//===============================================================
//tests <= with empty set
//===============================================================

void test17     (void)
{
        Set<int> s1;
        for (int i = 0; i < 10; i++)
                s1.insert(i);
        Set<int> s2;
        if (s1 <= s2)
                return;
        else
                cout << "test 17 failed" << endl;
}

//===============================================================
//tests intersection between two different sets
//===============================================================

void test18()
{
        Set<int> s1;
        for (int i = 0; i<6; i++)
                s1.insert(i);
        Set<int> s2;
        for (int i = 6; i<11; i++)
                s2.insert(i);
        Set<int> s3;
        s3 = s1&s2;
        string str = s3.toString();
        assert(str == "{}");
}

//===============================================================
//tests difference between two different sets
//==============================================================

void test19()
{
        Set<int> s1;
        for (int i = 0; i<6; i++)
                s1.insert(i);
        Set<int> s2;
        for (int i = 6; i<11; i++)
                s2.insert(i);
        Set<int> s3;
        s3 = s1-s2;
        string str = s3.toString();
        assert(str == "{0, 1, 2, 3, 4, 5}");
}

//==============================================================
//tests cardinality of empty set
//==============================================================

void test20     (void)
{
        Set<char> s1;
        assert(s1.cardinality() == 0);
}

//==============================================================
//tests difference operator with a bigger second set
//==============================================================

void test21     (void)
{
        Set<int> s1;
        for (int i = 0; i<6; i++)
                s1.insert(i);
        Set<int> s2;
        for (int i = 0; i<11; i++)
                s2.insert(i);
        Set<int> s3 = s1 - s2;
        string str = s3.toString();
        assert(str == "{}");
}

//==============================================================
//tests == with non equal sets
//==============================================================

void test22     (void)
{
        Set<int> s1;
        for (int i = 0; i<6; i++)
                s1.insert(i);
        Set<int> s2;
        for (int i = 6; i<11; i++)
                s2.insert(i);
        if (s1 == s2)
                cout << "Test 22 failed" << endl;

}

//==============================================================
//tests string set
```

```
//==============================================================

void test23 (void)
{
        Set<string> s1;
        s1.insert("Emma");
        s1.insert("Eliza");
        s1.insert("Evelyn");
        assert(s1.toString()=="{Emma, Eliza, Evelyn}");
}

//==============================================================
// tests contains when item not in set
//==============================================================

void test24 (void)
{
        Set<int> s1;
        for (int i = 0; i<6; i++)
                s1.insert(i);
        bool cont = s1.contains(6);
        assert(cont==0);
}


int main (void)
{

        test1();
        test2();
        test3();
        test4();
        test5();
        test2();
        test6();
        test7();

        test8();
        test9();

        test10();
        test11();
        //test12();
        test13();
        test14();
        test15();
        test16();
        test17();
        test18();
        test19();
        test20();
        test21();
        test22();
        test23();
        test24();
        return 0;
}
```

```
//==============================================================
//Emma Steinman
//September 4, 2017
//presidents.cpp
//This file contains a program to
//==============================================================

#include <iostream>
#include <string>
#include <fstream>
#include "set.h"
using namespace std;
Set<string> Whig (void)
{
        Set<string> w;
        ifstream inFile;
        string line;
        inFile.open("pres.txt");
        if (!inFile)
        {
                cout << "Unable to open file pres.txt" << endl;
                exit(1);
        }
        while (getline(inFile, line))
        {
                stringstream linestream(line);
                string name;
                string party;
                getline(linestream, name, '\t');
                linestream >> party;

                if (party == "(W)")
                        w.insert(name);
        }

        return w;
}

Set<string> Democrat (void)
{
        Set<string> d;
        ifstream inFile;
        string line;
        inFile.open("pres.txt");
        if (!inFile)
        {
                cout << "Unable to open file pres.txt" << endl;
                exit(1);
        }
        while (getline(inFile, line))
        {
                stringstream linestream(line);
                string name;
                string party;
                getline(linestream, name, '\t');
                linestream >> party;

                if (party == "(D)")
                        d.insert(name);
        }

        return d;
}
```

```
Set<string> Republican (void)
{
        Set<string> r;
        ifstream inFile;
        string line;
        inFile.open("pres.txt");
        if (!inFile)
        {
                cout << "Unable to open file pres.txt" << endl;
                exit(1);
        }
        while (getline(inFile, line))
        {
                stringstream linestream(line);
                string name;
                string party;
                getline(linestream, name, '\t');
                linestream >> party;

                if (party == "(R)")
                        r.insert(name);
        }

        return r;
}

Set<string> OtherParty (void)
{
        Set<string> op;
        ifstream inFile;
        string line;
        inFile.open("pres.txt");
        if (!inFile)
        {
                cout << "Unable to open file pres.txt" << endl;
                exit(1);
        }
        while (getline(inFile, line))
        {
                stringstream linestream(line);
                string name;
                string party;
                getline(linestream, name, '\t');
                linestream >> party;

                if (party != "(W)" && party != "(D)" && party != "(R)")
                        op.insert(name);
        }

        return op;
}

Set<string> VA (void)
{
        Set<string> VA;
        ifstream inFile;
        string line;
        inFile.open("pres.txt");
        if (!inFile)
        {
                cout << "Unable to open file pres.txt" << endl;
                exit(1);
        }
        while (getline(inFile, line))
        {
                stringstream linestream(line);
                string name;
                string party;
                string state;
                getline(linestream, name, '\t');
                linestream >> party >> state;

                if (state == "VA")
                        VA.insert(name);
        }

        return VA;
}

Set<string> NY (void)
{
        Set<string> NY;
        ifstream inFile;
        string line;
        inFile.open("pres.txt");
        if (!inFile)
        {
                cout << "Unable to open file pres.txt" << endl;
                exit(1);
        }
        while (getline(inFile, line))
        {
                stringstream linestream(line);
                string name;
                string party;
                string state;
                getline(linestream, name, '\t');
                linestream >> party >> state;
                if (state == "NY")
                        NY.insert(name);
        }

        return NY;
}

Set<string> MA (void)
{
        Set<string> MA;
        ifstream inFile;
        string line;
        inFile.open("pres.txt");
        if (!inFile)
        {
                cout << "Unable to open file pres.txt" << endl;
                exit(1);
        }
        while (getline(inFile, line))
        {
                stringstream linestream(line);
                string name;
                string party;
                string state;
                getline(linestream, name, '\t');
                linestream >> party >> state;
                if (state == "MA")
```

```
                        MA.insert(name);
        }

        return MA;
}

Set<string> OH (void)
{
        Set<string> OH;
        ifstream inFile;
        string line;
        inFile.open("pres.txt");
        if (!inFile)
        {
                cout << "Unable to open file pres.txt" << endl;
                exit(1);
        }
        while (getline(inFile, line))
        {
                stringstream linestream(line);
                string name;
                string party;
                string state;
                getline(linestream, name, '\t');
                linestream >> party >> state;
                if (state == "OH")
                        OH.insert(name);
        }

        return OH;
}

Set<string> OtherStates (void)
{
        Set<string> O;
        ifstream inFile;
        string line;
        inFile.open("pres.txt");
        if (!inFile)
        {
                cout << "Unable to open file pres.txt" << endl;
                exit(1);
        }
        while (getline(inFile, line))
        {
                stringstream linestream(line);
                string name;
                string party;
                string state;
                getline(linestream, name, '\t');
                linestream >> party >> state;

                if (state != "VA" && state != "NY" && state != "MA" && state != "OH")
                        O.insert(name);
        }

        return O;
}

Set<string> Episcopalian (void)
{
        Set<string> Ep;
        ifstream inFile;
        string line;
        inFile.open("pres.txt");
        if (!inFile)
        {
                cout << "Unable to open file pres.txt" << endl;
                exit(1);
        }
        while (getline(inFile, line))
        {
                stringstream linestream(line);
                string name;
                string party;
                string state;
                string religion;
                getline(linestream, name, '\t');
                linestream >> party >> state >> religion;

                if (religion == "Episcopalian")
                        Ep.insert(name);
        }

        return Ep;
}

Set<string> Presbyterian (void)
{
        Set<string> P;
        ifstream inFile;
        string line;
        inFile.open("pres.txt");
        if (!inFile)
        {
                cout << "Unable to open file pres.txt" << endl;
                exit(1);
        }
        while (getline(inFile, line))
        {
                stringstream linestream(line);
                string name;
                string party;
                string state;
                string religion;
                getline(linestream, name, '\t');
                linestream >> party >> state >> religion;

                if (religion == "Presbyterian")
                        P.insert(name);
        }

        return P;
}

Set<string> Methodist (void)
{
        Set<string> M;
        ifstream inFile;
        string line;
        inFile.open("pres.txt");
        if (!inFile)
        {
                cout << "Unable to open file pres.txt" << endl;
                exit(1);
        }
        while (getline(inFile, line))
        {
                stringstream linestream(line);
                string name;
                string party;
                string state;
                string religion;
                getline(linestream, name, '\t');
                linestream >> party >> state >> religion;

                if (religion == "Methodist")
                        M.insert(name);
        }

        return M;
}

Set<string> OtherReligion (void)
{
        Set<string> OR;
        ifstream inFile;
        string line;
        inFile.open("pres.txt");
        if (!inFile)
        {
                cout << "Unable to open file pres.txt" << endl;
                exit(1);
        }
        while (getline(inFile, line))
        {
                stringstream linestream(line);
                string name;
                string party;
                string state;
                string religion;
                getline(linestream, name, '\t');
                linestream >> party >> state >> religion;

                if (religion != "Episcopalian" && religion != "Presbyterian" && religion != "M
ethodist")
                        OR.insert(name);
        }

        return OR;
}

Set<string> Forties(void)
{
        Set<string> A40;
        ifstream inFile;
        string line;
        inFile.open("pres.txt");
        if (!inFile)
        {
                cout << "Unable to open file pres.txt" << endl;
                exit(1);
        }
        while (getline(inFile, line))
        {
                stringstream linestream(line);
                string name;
                string party;
                string state;
                string religion;
                int age;
                getline(linestream, name, '\t');
                linestream >> party >> state >> religion >> age;

                if (age >= 40 && age <=49)
                        A40.insert(name);
        }

        return A40;
}

Set<string> Fifties(void)
{
        Set<string> A50;
        ifstream inFile;
        string line;
        inFile.open("pres.txt");
        if (!inFile)
        {
                cout << "Unable to open file pres.txt" << endl;
                exit(1);
        }
        while (getline(inFile, line))
        {
                stringstream linestream(line);
                string name;
                string party;
                string state;
                string religion;
                int age;
                getline(linestream, name, '\t');
                linestream >> party >> state >> religion >> age;

                if (age >= 50 && age <=59)
                        A50.insert(name);
        }

        return A50;
}

Set<string> Sixties(void)
{
        Set<string> A60;
        ifstream inFile;
        string line;
        inFile.open("pres.txt");
        if (!inFile)
        {
                cout << "Unable to open file pres.txt" << endl;
                exit(1);
        }
        while (getline(inFile, line))
        {
                stringstream linestream(line);
                string name;
                string party;
                string state;
                string religion;
                int age;
                getline(linestream, name, '\t');
                linestream >> party >> state >> religion >> age;
                if (age >= 60 && age <=69)
                        A60.insert(name);
        }

        return A60;
}

int main (void)
{
        Set<string> va = VA();
        Set<string> ny = NY();
        Set<string> ma = MA();
        Set<string> oh = OH();
        Set<string> otherstate = OtherStates();
        Set<string> episcopalian = Episcopalian();
        Set<string> presbyterian = Presbyterian();
        Set<string> methodist = Methodist();
        Set<string> otherreligion = OtherReligion();
        Set<string> forties = Forties();
        Set<string> fifties = Fifties();
        Set<string> sixties = Sixties();
        Set<string> whig = Whig();
        Set<string> democrat = Democrat();
        Set<string> republican = Republican();
        Set<string> otherparty = OtherParty();


        cout << (episcopalian & va & whig) << endl;
        cout << (methodist & oh) << endl;
        cout << (whig + democrat) << endl;
        cout << forties << endl;
        return 0;
}
```