



Restart your subscription to keep accessing solutions. Your Chegg Study subscription will expire on September 30, 2019.

[CONTINUE MY SUBSCRIPTION](#)[home](#) / [study](#) / [engineering](#) / [computer science](#) / [computer science questions and answers](#) / the goal of this assignment is to ...

## Question: The goal of this assignment is to reinforce implementation of ...

(1 bookmark)

The goal of this assignment is to reinforce implementation of container class concepts in C++. Specifically, the assignment is to create a dynamic array implementation of a set. Add the efficiency of each function to the documentation in the header file Use test\_set.cpp as your test program.

**Set.h & Test\_Set.cpp is code that is already given & my code is the Set.cpp thats not working.**

**FILE: SET.H**

```
#ifndef _SET_H
#define _SET_H

#include <cstdlib>
#include <iostream>

class set
{
public:
    typedef int value_type;
    typedef std::size_t size_type;
    static const size_type INITIAL_CAPACITY = 30;

    set(size_type initial_capacity = INITIAL_CAPACITY);
    // postcondition: empty set has been created

    ~set();
    // postcondition: set has been deallocated

    set(const set& source);
    // postcondition: a copy of source has been created

    set& operator = (const set& source);
    // postcondition:

    void insert(const value_type& entry);
    // postcondition: entry is in the set

    void remove(const value_type& entry);
    // postcondition: entry is not in the set

    size_type size() const;
    // postcondition: number of elements in the set has been returned

    bool contains(const value_type& entry) const;
    // postcondition: whether entry is in the set has been returned

    friend set set_union(const set& s1, const set& s2);
    //postcondition: union of s1 & s2 has been returned

    friend set set_intersection(const set& s1, const set& s2);
    // postcondition: intersection of s1 & s2 has been returned

    friend set set_difference(const set& s1, const set& s2);
    // postcondition: difference of s1 - s2 has been returned

    friend bool is_subset(const set& s1, const set& s2);
    // postcondition: returned whether s1 is a subset of s2

    friend bool operator == (const set& s1, const set& s2);
    // postcondition: returned whether s1 & s2 are equal

    friend std::ostream& operator << (std::ostream& output, const set& s);
    // postcondition: s has been displayed on output
```

### Post a question

Answers from our experts for your tough homework questions

[Continue to post](#)

20 questions remaining

### My Textbook Solutions

Instant access to step-by-step solutions for your textbooks

[Add a textbook](#)
[Add a textbook](#)
[Add a textbook](#)

**Computer Science** Chegg tutors who can help right now



Madhur M.  
Maulana Azad N...

183



Scott M.  
University of Min...

479



Shawn F.  
University of Kan...

29

[Find me a tutor](#)

```
private:
    size_type find(const value_type& entry) const;
    // returned location of entry in the set if entry is in the set - used otherwise
    void resize(unsigned int new_size);
    value_type* data;
    size_type used;
    size_type capacity;
};
```

```
#endif
```

**File: Test\_Set.cpp**

```
#include "set.h"
#include <cassert>
#include <iostream>
```

```
int main ()
{
    set s;
    assert (!s.contains (7));
    s.insert (7);
    assert (s.contains (7));
    s.remove (7);
    assert (!s.contains (7));

    set s1;
    s1.insert (4);
    s1.insert (5);
    s1.insert (-24);
    s1.insert (89);
    s1.insert (34);
    s1.insert (11);
    s1.insert (0);
    s1.insert (3);
    s1.insert (14);
    s1.insert (28);
    std::cout << s1 << std::endl;

    set s2;
    s2.insert (6);
    s2.insert (-5);
    s2.insert (-24);
    s2.insert (-89);
    s2.insert (34);
    s2.insert (-11);
    s2.insert (0);
    s2.insert (3);
    std::cout << s2 << std::endl;

    set s3 = set_union (s1, s2);
    assert (s3.contains (4));
    assert (s3.contains (0));
    assert (s3.contains (-5));
    std::cout << s3 << std::endl;

    set s4 = set_intersection (s1, s2);
    assert (s4.contains (34));
    assert (!s4.contains (4));
    assert (!s4.contains (-5));
    std::cout << s4 << std::endl;

    set s5 = set_difference (s1, s2);
    assert (s5.contains (4));
    assert (!s5.contains (0));
    assert (!s5.contains (-5));
    std::cout << s5 << std::endl;

    assert (is_subset (s5, s1));

    set s6(s2);
    assert (s6 == s2);
    std::cout << "all tests passed" << std::endl;
    return 0;
}
```

**FILE: SET.CPP (NOT WORKING)**

```

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

using namespace std;

//default constructor
set::set()
{
    CAPACITY = 30;
    used = 0;
    data = new value_type[CAPACITY];
}

//method to insert the element in the set
void set::insert(const value_type& entry)
{
    if (!contains(entry))
    {
        if (size() == CAPACITY)
        {
            double_capacity();
        }
        data[used] = entry;
        used++;
    }
}

//method to resize the capacity of the set for union
void set::double_capacity()
{
    value_type *newData = new value_type[2 * CAPACITY];
    for (int i = 0; i < used; ++i)
    {
        newData[i] = data[i];
    }
    data = newData;
    CAPACITY *= 2;
}

//method to delete the element in the set
void set::deletion(const value_type& entry)
{
    size_type location = find(entry);
    if (location >= 0)
    {
        data[location] = data[used - 1];
        used--;
    }
}

//method to compute the size of the set
set::size_type set::size() const
{
    return used;
}

//destructor
set::~set()
{
    //delete the memory
    //delete data
}

//method to find the location of the element in the set
set::size_type set::find(const value_type& entry) const
{
    size_type location = 0;
    while (location < used && data[location] != entry)
        location++;
    return location;
}

//method to check the element in the set
bool set::contains(const value_type& entry) const
{
    return find(entry) < used;
}

//method to find union
set set::union(const set& s1, const set& s2)

```

```

set set_union(const set& s1, const set& s2)
{
    set result;
    for (set::size_type i = 0; i < s1.size(); i++)
        result.insert(s1.data[i]);
    for (set::size_type i = 0; i < s2.size(); i++)
        result.insert(s2.data[i]);
    return result;
}

//method to compute intersection
set set_intersection(const set& s1, const set& s2)
{
    set result;
    for (set::size_type i = 0; i < s1.size(); i++)
    {
        if (s2.contains(s1.data[i]))
            result.insert(s1.data[i]);
    }
    return result;
}

//relative_complement method
set relative_complement(const set& s1, const set& s2)
{
    set result;
    for (set::size_type i = 0; i < s2.size(); i++)
    {
        if (!s1.contains(s2.data[i]))
            result.insert(s2.data[i]);
    }
    return result;
}

//overloading assignment operator
std::ostream& operator<< (std::ostream& output, const set& s)
{
    for (set::size_type i = 0; i < s.size(); i++)
    {
        output << s.data[i] << " ";
    }
    return output;
}

```

## Expert Answer



Peter Adams answered this  
659 answers

Was this answer helpful?

0

0

Program code screen shot:

**//Set.h**

```

#pragma once

#ifndef _SET_H
#define _SET_H

#include <cstdlib>
#include <iostream>

class set
{
public:
    typedef int value_type;
    typedef std::size_t size_type;
    static const size_type INITIAL_CAPACITY = 30;

    set(size_type initial_capacity = INITIAL_CAPACITY);
    // postcondition: empty set has been created

    ~set();
    // postcondition: set has been deallocated

    set(const set& source);
    // postcondition: a copy of source has been created

```

```

    set& operator = (const set& source);
    // precondition:
    // postcondition:

    void insert(const value_type& entry);
    // precondition: entry is in the set
    // postcondition: entry is not in the set

    void remove(const value_type& entry);
    // precondition: entry is not in the set
    // postcondition: entry is in the set

    size_type size() const;
    // precondition: number of elements in the set has been returned
    // postcondition:

    bool contains(const value_type& entry) const;
    // precondition: whether entry is in the set has been returned
    // postcondition:

    friend set set_union(const set& s1, const set& s2);
    //postcondition: union of s1 & s2 has been returned

    friend set set_intersection(const set& s1, const set& s2);
    // precondition: intersection of s1 & s2 has been returned
    // postcondition:

    friend set set_difference(const set& s1, const set& s2);
    // precondition: difference of s1 - s2 has been returned
    // postcondition:

    friend bool is_subset(const set& s1, const set& s2);
    // precondition: returned whether s1 is a subset of s2
    // postcondition:

    friend bool operator == (const set& s1, const set& s2);
    // precondition: returned whether s1 & s2 are equal
    // postcondition:

    friend std::ostream& operator << (std::ostream& output, const set& s);
    // precondition: s has been displayed on output
    // postcondition:

private:
    size_type find(const value_type& entry) const;
    // returned location of entry in the set if entry is in the set - used
    otherwise
    void resize(unsigned int new_size);
    value_type* data;
    size_type used;
    size_type capacity;
};

#endif

```

---

```

//Test_set.cpp

```

```

// Declare the required header file
#include "stdafx.h"
#include "set.h"
#include <cassert>
#include <iostream>

// define the main function of the program
int main()
{
    set s;
    assert(!s.contains(7));
    s.insert(7);
    assert(s.contains(7));
    s.remove(7);
    assert(!s.contains(7));

    set s1;
    s1.insert(4);
    s1.insert(5);
    s1.insert(-24);
    s1.insert(89);
    s1.insert(34);
    s1.insert(11);
    s1.insert(0);
    s1.insert(3);
    s1.insert(14);
    s1.insert(28);
    std::cout << s1 << std::endl;
}

```

```

        set s1 = set_union(s1, s2);

        set s2;
        s2.insert(6);
        s2.insert(-5);
        s2.insert(-24);
        s2.insert(-89);
        s2.insert(34);
        s2.insert(-11);
        s2.insert(0);
        s2.insert(3);
        std::cout << s2 << std::endl;
        set s3 = set_union(s1, s2);
        assert(s3.contains(4));
        assert(s3.contains(0));
        assert(s3.contains(-5));
        std::cout << s3 << std::endl;

        set s4 = set_intersection(s1, s2);
        assert(s4.contains(34));
        assert(!s4.contains(4));
        assert(!s4.contains(-5));
        std::cout << s4 << std::endl;

        set s5 = set_difference(s1, s2);
        assert(s5.contains(4));
        assert(!s5.contains(0));
        assert(!s5.contains(-5));
        std::cout << s5 << std::endl;

        assert(is_subset(s5, s1));

        set s6(s2);
        assert(s6 == s2);
        // display all the test element
        std::cout << "all tests passed" << std::endl;
        system("pause");
        return 0;
}

```

## Set.cpp

```

// Declare the required header file
#include "stdafx.h"
#include "Set.h"
#include <cassert>
#include <iostream>

// define the name space of the program
using namespace std;

//default constructor for initial capacity
set::set(size_type initial_capacity)
{
    capacity = initial_capacity;
    used = 0;
    data = new value_type[capacity];
}

//default constructor for set the source
set::set(const set& source)
{
    capacity = source.capacity;
    used = source.used;
    data = new value_type[capacity];

    for (int i = 0; i < used; i++)
        data[i] = source.data[i];
}

// default destructor
set::~set()
{
    capacity = 0;
    used = 0;
    delete data;
}

```

```

//method to insert the element in the set
void set::insert(const value_type& entry)
{
    bool found = false;
    for (int i = 0; i < used; i++)
        if (data[i] == entry)
        {
            found = true;
            break;
        }
    if (!found && used < capacity)
    {
        data[used] = entry;
        used++;
    }
}

//method to delete the element in the set
void set::remove(const value_type& entry)
{
    int pos = -1;
    for (int i = 0; i < used; i++)
        if (data[i] == entry)
        {
            pos = i;
            break;
        }
    for (int i = pos; i < used - 1; i++)
        data[i] = data[i + 1];
    used--;
}

//method to find the size of the element in the set
set::size_type set::size() const
{
    return used;
}

//method to check the element in the set
bool set::contains(const value_type& entry) const
{
    for (int i = 0; i < used; i++)
        if (data[i] == entry)
            return true;
    return false;
}

//method to find union
set set_union(const set& s1, const set& s2)
{
    set temp;
    for (int i = 0; i < s1.size(); i++)
        temp.insert(s1.data[i]);
    for (int i = 0; i < s2.size(); i++)
        temp.insert(s2.data[i]);
    return temp;
}

//method to compute intersection
set set_intersection(const set& s1, const set& s2)
{
    set temp;
    for (int i = 0; i < s1.size(); i++)
        if (s2.contains(s1.data[i]))
            temp.insert(s1.data[i]);
    return temp;
}

// method to set the difference
set set_difference(const set& s1, const set& s2)
{
    set temp;
    for (int i = 0; i < s1.size(); i++)
        if (!s2.contains(s1.data[i]))
            temp.insert(s1.data[i]);
    return temp;
}

// method to compute subset
bool is_subset(const set& s1, const set& s2)
{
    for (int i = 0; i < s1.size(); i++)

```

```

        if (!s2.contains(s1.data[i]))
            return false;
        return true;
    }

// method to boolean operator
bool operator == (const set& s1, const set& s2)
{
    if (is_subset(s1, s2) && is_subset(s2, s1))
        return true;
    return false;
}

//overloading assinment operator
std::ostream& operator << (std::ostream& output, const set& s)
{
    output << "{ ";
    for (int i = 0; i < s.size() - 1; i++)
        output << s.data[i] << ", ";
    output << s.data[s.size() - 1] << " }";
    return output;
}

```

Sample output:

```

{ 4, 5, -24, 89, 34, 11, 0, 3, 14, 28 }
{ 6, -5, -24, -89, 34, -11, 0, 3 }
{ 4, 5, -24, 89, 34, 11, 0, 3, 14, 28, 6, -5, -89, -11 }
{ -24, 34, 0, 3 }
{ 4, 5, 89, 11, 14, 28 }
all tests passed
Press any key to continue . . .

```

Program code to copy:

```

//Set.h

#pragma once

#ifndef _SET_H
#define _SET_H

#include <cstdlib>
#include <iostream>

class set
{
public:
    typedef int value_type;

    typedef std::size_t size_type;

    static const size_type INITIAL_CAPACITY = 30;
    set(size_type initial_capacity = INITIAL_CAPACITY);

    // postcondition: empty set has been created
    ~set();

    // postcondition: set has been deallocated
    set(const set& source);

    // postcondition: a copy of source has been created
    set& operator = (const set& source);

    // postcondition:
    void insert(const value_type& entry);

```



```

// postcondition: entry is in the set
void remove(const value_type& entry);

// postcondition: entry is not in the set

size_type size() const;

// postcondition: number of elements in the set has been returned

bool contains(const value_type& entry) const;

// postcondition: whether entry is in the set has been returned

friend set set_union(const set& s1, const set& s2);

//postcondition: union of s1 & s2 has been returned

friend set set_intersection(const set& s1, const set& s2);

// postcondition: intersection of s1 & s2 has been returned

friend set set_difference(const set& s1, const set& s2);

// postcondition: difference of s1 - s2 has been returned

friend bool is_subset(const set& s1, const set& s2);

// postcondition: returned whether s1 is a subset of s2

friend bool operator == (const set& s1, const set& s2);

// postcondition: returned whether s1 & s2 are equal

friend std::ostream& operator << (std::ostream& output, const set& s);

// postcondition: s has been displayed on output

private:

    size_type find(const value_type& entry) const;

    // returned location of entry in the set if entry is in the set - used otherwise

    void resize(unsigned int new_size);

    value_type* data;

    size_type used;

    size_type capacity;

};

#endif

//Test_set.cpp

// Declare the required header file

#include "stdafx.h"

#include "set.h"

#include <cassert>

#include <iostream>

// define the main function of the program

int main()

{

    set s;

    assert(!s.contains(7));

    s.insert(7);

    assert(s.contains(7));

    s.remove(7);

    assert(!s.contains(7));

```

```
set s1;

s1.insert(4);
s1.insert(5);
s1.insert(-24);
s1.insert(89);
s1.insert(34);
s1.insert(11);
s1.insert(0);
s1.insert(3);
s1.insert(14);
s1.insert(28);
std::cout << s1 << std::endl;

set s2;

s2.insert(6);
s2.insert(-5);
s2.insert(-24);
s2.insert(-89);
s2.insert(34);
s2.insert(-11);
s2.insert(0);
s2.insert(3);

std::cout << s2 << std::endl;

set s3 = set_union(s1, s2);
assert(s3.contains(4));
assert(s3.contains(0));
assert(s3.contains(-5));

std::cout << s3 << std::endl;

set s4 = set_intersection(s1, s2);
assert(s4.contains(34));
assert(!s4.contains(4));
assert(!s4.contains(-5));

std::cout << s4 << std::endl;

set s5 = set_difference(s1, s2);
assert(s5.contains(4));
assert(!s5.contains(0));
assert(!s5.contains(-5));

std::cout << s5 << std::endl;

assert(is_subset(s5, s1));

set s6(s2);
assert(s6 == s2);

// display all the test element

std::cout << "all tests passed" << std::endl;

system("pause");
```

```
        return 0;
    }

Set.cpp

// Declare the required header file
#include "stdafx.h"
#include "Set.h"
#include <cassert>
#include<iostream>

// define the name space of the program
using namespace std;

//default constructor for initial capacity
set::set(size_type initial_capacity)
{
    capacity = initial_capacity;
    used = 0;
    data = new value_type[capacity];
}

//default constructor for set the source
set::set(const set& source)
{
    capacity = source.capacity;
    used = source.used;
    data = new value_type[capacity];
    for (int i = 0; i < used; i++)
        data[i] = source.data[i];
}

// default destructor
set::~set()
{
    capacity = 0;
    used = 0;
    delete data;
}

//method to insert the element in the set
void set::insert(const value_type& entry)
{
    bool found = false;
    for (int i = 0; i < used; i++)
        if (data[i] == entry)
        {
            found = true;
            break;
        }
}
```

```

    ,
    if (!found && used < capacity)
    {
        data[used] = entry;
        used++;
    }
}

//method to delete the element in the set
void set::remove(const value_type& entry)
{
    int pos = -1;
    for (int i = 0; i < used; i++)
        if (data[i] == entry)
        {
            pos = i;
            break;
        }
    for (int i = pos; i < used - 1; i++)
        data[i] = data[i + 1];
    used--;
}

//method to find the size of the element in the set
set::size_type set::size() const
{
    return used;
}

//method to check the element in the set
bool set::contains(const value_type& entry) const
{
    for (int i = 0; i < used; i++)
        if (data[i] == entry)
            return true;
    return false;
}

//method to find union
set set_union(const set& s1, const set& s2)
{
    set temp;
    for (int i = 0; i < s1.size(); i++)
        temp.insert(s1.data[i]);
    for (int i = 0; i < s2.size(); i++)
        temp.insert(s2.data[i]);
    return temp;
}

```

```

,

//method to compute intersection
set set_intersection(const set& s1, const set& s2)
{
    set temp;

    for (int i = 0; i < s1.size(); i++)
        if (s2.contains(s1.data[i]))
            temp.insert(s1.data[i]);

    return temp;
}

// method to set the difference
set set_difference(const set& s1, const set& s2)
{
    set temp;

    for (int i = 0; i < s1.size(); i++)
        if (!s2.contains(s1.data[i]))
            temp.insert(s1.data[i]);

    return temp;
}

// method to compute subset
bool is_subset(const set& s1, const set& s2)
{
    for (int i = 0; i < s1.size(); i++)
        if (!s2.contains(s1.data[i]))
            return false;

    return true;
}

// method to boolean operator
bool operator == (const set& s1, const set& s2)
{
    if (is_subset(s1, s2) && is_subset(s2, s1))
        return true;

    return false;
}

//overloading assinment operator
std::ostream& operator << (std::ostream& output, const set& s)
{
    output << "{ ";

    for (int i = 0; i < s.size() - 1; i++)
        output << s.data[i] << ", ";

    output << s.data[s.size() - 1] << " }";

    return output;
}

```

Comment >

Questions viewed by other students

Q: The goal of this assignment is to reinforce implementation of dynamic arrays in C++. Specifically, the assignment is to implement a set using a dynamic array. You need to implement the following set operations union intersection relative complement insertion - if the element is already in the set, then nothing happens deletion - if the element is not in the set, then nothing happens...

A: [See answer](#)

Q: The goal of this assignment is to reinforce implementation of container class concepts in C++. Specifically, the assignment is to create a static implementation of a set. Set Header File: #ifndef \_SET\_H #define \_SET\_H #include #include class set { public: typedef int value\_type; typedef std::size\_t size\_type; static const size\_type CAPACITY = 30; set...

A: [See answer](#)

Show more v

ABOUT CHEGG

Become a Tutor  
Chegg For Good  
College Marketing  
Corporate Development  
Investor Relations  
Jobs  
Join Our Affiliate Program  
Media Center  
Site Map

LEGAL & POLICIES

Advertising Choices  
Cookie Notice  
General Policies  
Intellectual Property Rights  
International Privacy Policy  
Terms of Use  
Chegg Tutors Terms of Service  
US Privacy Policy  
Your CA Privacy Rights  
Honor Code

CHEGG PRODUCTS AND SERVICES

Cheap Textbooks  
Chegg Coupon  
Chegg Play  
Chegg Study Help  
College Textbooks  
eTextbooks  
Chegg Math Solver  
Mobile Apps

Online Tutoring  
Sell Textbooks  
Solutions Manual  
Study 101  
Test Prep  
Textbook Rental  
Used Textbooks  
Digital Access Codes

CHEGG NETWORK

Easybib  
Internships.com  
Studyblue

CUSTOMER SERVICE

Customer Service  
Give Us Feedback  
Help with Chegg Tutors  
Help with eTextbooks  
Help to use EasyBib Plus  
Manage Chegg Study Subscription  
Return Your Books  
Textbook Return Policy



© 2003-2019 Chegg Inc. All rights reserved.

