**Московский государственный технический**

**университет им. Н.Э. Баумана**

Факультет «Информатика и системы управления»

Кафедра ИУ5 «Системы обработки информации и управления»

Курс «Программирование на основе классов и шаблонов»

Отчет по лабораторной работе №5

«Наследование. Множества»

|  |  |  |
| --- | --- | --- |
| Выполнил: |  | Проверил: |
| студент группы ИУ5-24б |  | преподаватель каф. ИУ5 |
| Байдаков В.М. |  |  |
| Подпись и дата: |  | Подпись и дата: |
|  |  |  |

Москва, 2024 г.

**Постановка задачи:**

Разработать шаблонный класс множество (MySet) на базе шаблонного класса вектор (MyVector) для выполнения операций над множествами (+, -, \*, +=, -=, \*=, ==) и функцию main() для его тестирования.

Класс вектор должен быть динамическим массивом, размер которого может автоматически изменяться (увеличиваться или уменьшаться) в процессе выполнения программы. Добавление элементов производится в конец вектора.

Для ускорения выполнения операций над множествами вектор, используемый классом множество, должен быть отсортирован (сортировку достаточно делать только при добавлении элемента в множество). Для поиска элементов множества следует использовать метод половинного деления.

Методы add\_element() и delete\_element() производного класса MySet переопределяют одноименные методы базового класса MyVector, а остальные элементы класса MyVector наследуются классом MySet.

**Разработка алгоритма:**

**Библиотека containers**

**Класс MyVector<T> -**

int max\_size - максимально возможное количество элементов в массиве

int size - текущее количество элементов в массиве

T\* pdata - указатель на начало массива данных

void resize - изменяет размер массива в зависимости от текущего количества элементов

MyVector - конструктор по умолчанию, инициализирует вектор с начальными значениями

MyVector(T el, int max\_size = MAX\_SIZE) - конструктор, инициализирует вектор одним элементом или пустым

MyVector(const MyVector &v) - конструктор копирования, копирует данные из другого вектора

~MyVector - деструктор, очищает выделенную память

void add\_element(T el) - добавляет элемент в конец вектора

bool delete\_element(int i) - удаляет элемент по заданному индексу

T& operator[](int i) const - предоставляет доступ к элементу по индексу

void sort - сортирует элементы вектора

int get\_size - возвращает текущее количество элементов

int get\_max\_size - возвращает максимально возможное количество элементов

int find(T el) const - поиск элемента, возвращает индекс или -1 если не найден

MyVector& operator=(const MyVector &v) - оператор присваивания, копирует элементы из другого вектора

std::ostream& operator<<(std::ostream &out, const MyVector &v) - оператор вывода вектора в поток

**Класс MySet<T> (наследуется от MyVector<T>) -**

int q\_find(const T &el) - выполняет бинарный поиск элемента, возвращает его индекс или -1

MySet(const T &el) - конструктор, инициализирует множество одним элементом

MySet() - конструктор по умолчанию

bool operator==(MySet &s) - сравнивает множества на равенство

MySet& operator+=(MySet &s) - добавляет элементы из другого множества

MySet& operator-=(MySet &s) - удаляет элементы, совпадающие с элементами другого множества

MySet& operator\*=(MySet &s) - пересекает множества

void add\_element(const T &el) - добавляет элемент, если он отсутствует

void delete\_element(const T &el) - удаляет элемент

bool is\_element(const T &el) - проверяет наличие элемента

std::ostream& operator<<(std::ostream &out, const MySet &s) - оператор вывода множества в поток

MySet operator+(const MySet &s1, const MySet &s2) - объединение множеств

MySet operator-(const MySet &s1, const MySet &s2) - разность множеств

MySet operator\*(const MySet &s1, const MySet &s2) - пересечение множеств

MySet& operator=(const MySet &el) - оператор присваивания, копирует множество

**Текст программы:**

**MyVector.h**

#pragma once

#include <iostream>

#include <cstring>

const int MAX\_SIZE = 4;

template <typename T>

class MyVector

{

protected:

    int max\_size;

    int size;

    T \*pdata;

    void resize()

    {

        if (size >= max\_size)

        {

            max\_size \*= 2;

        }

        else if (size < max\_size / 4 && max\_size / 2 >= 8)

        {

            max\_size /= 2;

        }

        else

        {

            return;

        }

        T \*newData = new T[max\_size];

        for (size\_t i = 0; i < size; i++)

        {

            newData[i] = pdata[i];

        }

        delete[] pdata;

        pdata = newData;

    }

public:

    MyVector()

    {

        max\_size = MAX\_SIZE;

        size = 0;

        pdata = new T[max\_size];

    }

    MyVector(T el, int max\_size = MAX\_SIZE)

    {

        this->max\_size = max\_size;

        size = 0;

        pdata = new T[max\_size];

        if (el)

        {

            add\_element(el);

        }

    }

    MyVector(const MyVector &v)

    {

        max\_size = v.max\_size;

        size = v.size;

        pdata = new T[max\_size];

        for (size\_t i = 0; i < size; i++)

        {

            pdata[i] = v.pdata[i];

        }

    }

    ~MyVector()

    {

        delete[] pdata;

    }

    void add\_element(T el)

    {

        resize();

        pdata[size++] = el;

    }

    bool delete\_element(int i)

    {

        if (i < 0 || i >= size)

        {

            return false;

        }

        delete pdata[i];

        for (int j = i + 1; j < size; j++)

        {

            pdata[j - 1] = pdata[j];

        }

        size--;

        resize();

        return true;

    }

    T &operator[](int i) const

    {

        if (i < 0 || i >= size)

        {

            std::cout << "\nwrong index, 0 index will be returned\n";

            return pdata[0];

        }

        return pdata[i];

    }

    void sort()

    {

        for (int i = 0; i < size; i++)

        {

            for (int j = i + 1; j < size; j++)

            {

                if (pdata[i] > pdata[j])

                {

                    std::swap(pdata[i], pdata[j]);

                }

            }

        }

    }

    int get\_size() const { return size; }

    int get\_max\_size() const { return max\_size; }

    int find(T el) const

    {

        for (size\_t i = 0; i < size; i++)

        {

            if (compare(el, pdata[i]) == 0)

            {

                return i;

            }

        }

        return -1;

    }

    MyVector &operator=(const MyVector &v)

    {

        if(this == &v){return \*this;}

        delete[] pdata;

        pdata = new T[v.max\_size];

        size = 0;

        max\_size = v.max\_size;

        for (int i = 0; i < v.size; i++)

        {

            add\_element(v.pdata[i]);

        }

        return \*this;

    }

    friend std::ostream &operator<<(std::ostream &out, const MyVector &v)

    {

        out << '{';

        for (size\_t i = 0; i < v.size; i++)

        {

            out << ' ' << v[i];

        }

        out << '}' << '\n';

        return out;

    }

};

//------------------------------------------------------------------------------

template <>

MyVector<char \*>::MyVector(const MyVector &v)

{

    max\_size = v.max\_size;

    size = v.size;

    pdata = new char \*[max\_size];

    for (size\_t i = 0; i < size; i++)

    {

        pdata[i] = new char[strlen(v.pdata[i]) + 1];

        strcpy(pdata[i], v.pdata[i]);

    }

}

template <>

MyVector<char \*>::~MyVector()

{

    for (size\_t i = 0; i < size; i++)

    {

        delete[] pdata[i];

    }

    delete[] pdata;

}

template <>

void MyVector<char \*>::add\_element(char \*el)

{

    resize();

    pdata[size] = new char[strlen(el) + 1];

    strcpy(pdata[size], el);

    size++;

}

template <>

MyVector<char \*>::MyVector(char \*el, int maxsize) : pdata(nullptr), size(0), max\_size(maxsize > MAX\_SIZE ? maxsize : MAX\_SIZE)

{

    pdata = new char \*[max\_size];

    add\_element(el);

}

template <>

bool MyVector<char \*>::delete\_element(int i)

{

    if (i < 0 || i >= size)

    {

        return false;

    }

    delete[] pdata[i]; //!!!!!!!тут исправление

    for (int j = i + 1; j < size; j++)

    {

        pdata[j - 1] = pdata[j];

    }

    size--;

    resize();

    return true;

}

template <>

MyVector<char \*> &MyVector<char \*>::operator=(const MyVector &v)

{

    if(this == &v){return \*this;}

    for (int i = 0; i < size; i++)

    {

        delete[] pdata[i];

    }

    delete[] pdata;

    pdata = new char \*[v.max\_size];

    size = 0;

    max\_size = v.max\_size;

    for (int i = 0; i < v.size; i++)

    {

        add\_element(v.pdata[i]);

    }

    return \*this;

}

template <>

void MyVector<char \*>::sort()

{

    for (int i = 0; i < size; i++)

    {

        for (int j = i + 1; j < size; j++)

        {

            if (strcmp(pdata[i],pdata[j])>0)

            {

                std::swap(pdata[i], pdata[j]);

            }

        }

    }

}

**MySet.h**

#include <iostream>

#include "myvector.h"

template <typename T>

class MySet : public MyVector<T>

{

private:

  int q\_find(const T &el)

  {

    size\_t left = 0, right = this->size;

    while (left < right)

    {

      size\_t mid = left + (right - left) / 2;

      if (this->pdata[mid] < el)

      {

        left = mid + 1;

      }

      else if (this->pdata[mid] > el)

      {

        right = mid;

      }

      else

      {

        return mid;

      }

    }

    return -1;

  }

public:

  using MyVector<T>::max\_size;

  using MyVector<T>::size;

  using MyVector<T>::pdata;

  MySet(const T &el) : MyVector<T>(el) {}

  MySet() : MyVector<T>() {}

  bool operator==(MySet &s)

  {

    if (size != s.size)

    {

      return false;

    }

    for (size\_t i = 0; i < size; i++)

    {

      if (pdata[i] != s.pdata[i])

      {

        return false;

      }

    }

    return true;

  }

  MySet &operator+=(MySet &s)

  {

    \*this = \*this + s;

    return \*this;

  }

  MySet &operator-=(MySet &s)

  {

    \*this = \*this - s;

    return \*this;

  }

  MySet &operator\*=(MySet &s)

  {

    \*this = \*this \* s;

    return \*this;

  }

  void add\_element(const T &el)

  {

    if (is\_element(el))

    {

      return;

    }

    this->resize();

    size++;

    int ind = 0;

    while ((ind < size - 1) && (pdata[ind] < el))

    {

      ind++;

    }

    for (int i = size - 1; i > ind; i--)

    {

      pdata[i] = pdata[i - 1];

    }

    pdata[ind] = el;

  }

  void delete\_element(const T &el)

  {

    int index = q\_find(el);

    if (index == -1)

    {

      return;

    }

    delete pdata[index];

    for (size\_t i = index; i < this->size; i++)

    {

      pdata[i] = pdata[i + 1];

    }

    this->size--;

    pdata[size] = nullptr;

  }

  bool is\_element(const T &el)

  {

    return q\_find(el) != -1;

  }

  friend std::ostream &operator<<(std::ostream &out, const MySet &s)

  {

    out << '{';

    for (size\_t i = 0; i < s.get\_size(); i++)

    {

      out << ' ' << s[i];

    }

    out << '}' << std::endl;

    return out;

  }

  friend MySet operator+(const MySet &s1, const MySet &s2)

  {

    MySet s3(s1);

    for (size\_t i = 0; i < s2.get\_size(); i++)

    {

      if (!s3.is\_element(s2[i]))

      {

        s3.add\_element(s2[i]);

      }

    }

    return s3;

  }

  friend MySet operator-(const MySet &s1, const MySet &s2)

  {

    MySet s3(s1);

    for (size\_t i = 0; i < s2.get\_size(); i++)

    {

      if (s3.is\_element(s2[i]))

      {

        s3.delete\_element(s2[i]);

      }

    }

    return s3;

  }

  friend MySet operator\*(const MySet &s1, const MySet &s2)

  {

    MySet s3(s1);

    for (size\_t i = 0; i < s2.get\_size(); i++)

    {

      if (!s3.is\_element(s2[i]))

      {

        s3.delete\_element(s2[i]);

      }

    }

    return s3;

  }

  MySet &operator=(const MySet &el)

  {

    delete[] pdata;

    pdata = new T[el.max\_size];

    size = 0;

    max\_size = el.max\_size;

    for (int i = 0; i < el.size; i++)

    {

      add\_element(el.pdata[i]);

    }

    return \*this;

  }

};

//--------------------------------------------------------------------------

template <>

int MySet<char \*>::q\_find(char \*const &el)

{

  size\_t left = 0, right = this->size;

  while (left < right)

  {

    size\_t mid = left + (right - left) / 2;

    if (strcmp(this->pdata[mid], el) < 0)

    {

      left = mid + 1;

    }

    else if (strcmp(this->pdata[mid], el) > 0)

    {

      right = mid;

    }

    else

    {

      return mid;

    }

  }

  return -1;

}

template <>

void MySet<char \*>::add\_element(char \*const &el)

{

  if (is\_element(el))

  {

    return;

  }

  this->resize();

  size++;

  int ind = 0;

  while ((ind < size - 1) && (strcmp(pdata[ind], el) < 0))

  {

    ind++;

  }

  for (int i = size - 1; i > ind; i--)

  {

    pdata[i] = pdata[i - 1];

  }

  pdata[ind] = new char[strlen(el) + 1];

  strcpy(pdata[ind], el);

}

template <>

void MySet<char \*>::delete\_element(char \*const &el)

{

  int index = q\_find(el);

  if (index == -1)

  {

    return;

  }

  delete[] pdata[index];

  for (size\_t i = index; i < this->size; i++)

  {

    pdata[i] = pdata[i + 1];

  }

  this->size--;

  pdata[size] = nullptr;

}

template <>

bool MySet<char \*>::operator==(MySet &s)

{

  if (this->size != s.size)

  {

    return false;

  }

  for (size\_t i = 0; i < this->size; i++)

  {

    if (strcmp(this->pdata[i], s.pdata[i]) != 0)

    {

      return false;

    }

  }

  return true;

}

В срр файлах просто заинклюжены заголовочные файлы

**Main.cpp**

#include <iostream>

#include "containers/myvector.h"

#include "containers/myset.h"

int main()

{

    system("chcp 65001");

    MyVector<char \*> v((char\*)"Hello!");

    v.add\_element("Привет!");

    v.add\_element("Привет!");

    v.add\_element("Привет!");

    v.add\_element("Привет!");

    v.add\_element("Привет!");

    std::cout << "Вектор v: " << v << std::endl;

    v.add\_element("Привет!");

    v.add\_element("Привет!");

    v.add\_element("Привет!");

    std::cout << "Вектор v: " << v << std::endl;

    MyVector<char \*> v1 = v;

    std::cout << "Вектор v1: " << v1 << std::endl;

    for (int i = 0; i < v1.get\_size()-1; i++)

        v1.delete\_element(0);

    std::cout << "Вектор v1: " << v1 << std::endl;

    MySet<char \*> s("Yes"), s1, s2;

    s.add\_element("Привет!");

    s.add\_element("No");

    char \*str = "Hello!";

    s.add\_element(str);

    std::cout << "Множество s: " << s << std::endl;

    s1.add\_element("Cat");

    s1.add\_element("No");

    s1.add\_element("Привет!");

    std::cout << "Множество s1: " << s1 << std::endl;

    s2 = s1 - s;

    std::cout << "Множество s2=s1-s: " << s2 << std::endl;

    std::cout << "Множество s1: " << s1 << std::endl;

    std::cout << "Множество s: " << s << std::endl;

    s2 = s - s1;

    std::cout << "Множество s2=s-s1: " << s2 << std::endl;

    std::cout << "Множество s1: " << s1 << std::endl;

    std::cout << "Множество s: " << s << std::endl;

    s2 = s1 + s;

    std::cout << "Множество s2=s1+s: " << s2 << std::endl;

    std::cout << "Множество s1: " << s1 << std::endl;

    std::cout << "Множество s: " << s << std::endl;

    s2 = s1 \* s;

    std::cout << "Множество s2=s1\*s: " << s2 << std::endl;

    std::cout << "Множество s1: " << s1 << std::endl;

    std::cout << "Множество s: " << s << std::endl;

    MySet<char \*> s3 = s2;

    std::cout << "Множество s3=s2: " << s3 << std::endl;

    if (s3 == s2)

        std::cout << "Множество s3=s2\n";

    else

        std::cout << "Множество s3!=s2\n";

    if (s3 == s1)

        std::cout << "Множество s3=s1\n";

    else

        std::cout << "Множество s3!=s1\n";

    if (s1 == s3)

        std::cout << "Множество s1=s3\n";

    else

        std::cout << "Множество s1!=s3\n";

    MySet<int> s4;

    s4.add\_element(5);

    s4.add\_element(2);

    std::cout<<s4;

    return 0;

}

**CMakeLists.txt**

cmake\_minimum\_required(VERSION 3.23)

set(project "lab5")

project(${project})

set(CMAKE\_CXX\_STANDARD 17)

set(${project}\_SOURCES

        main.cpp)

add\_subdirectory(containers)

set(${project}\_SOURCE\_LIST

        ${${project}\_SOURCES})

add\_executable(${project}

        ${${project}\_SOURCE\_LIST})

target\_link\_libraries(${project} containers)

cmake\_minimum\_required(VERSION 3.23)

set(project "containers")

project(${project})

set(CMAKE\_CXX\_STANDARD 17)

set(${project}\_SOURCES

        myset.cpp myvector.cpp)

set(${project}\_HEADERS

        myset.h myvector.h)

set(${project}\_SOURCE\_LIST

        ${${project}\_SOURCES}

        ${${project}\_HEADERS})

add\_library(${project}

        STATIC

        ${${project}\_SOURCE\_LIST})

**Анализ результатов**

