**

Колледж космического машиностроения и технологии

**КУРСОВОЙ ПРОЕКТ**

**По МДК.01.02 «Прикладное программирование»**

**Тема: «Разработка абстрактного типа данных «Множество»»**

Выполнил студент

Соколов Ярослав Алексеевич

Группа П2-17

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_(Подпись)

\_\_\_\_\_\_\_\_\_\_\_\_ (Дата сдачи работы)

Проверил

преподаватель

Гусятинер Леонид Борисович

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_(Подпись)

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_(Оценка)

Королев, 2020

**Оглавление:**

[**Введение** 2](#_gjdgxs)

[**Глава 1.**](#_30j0zll) **Теоретическая часть** 3

[**1.1.**](#_1fob9te) **Описание предметной области** 3

[**1.2.**](#_3znysh7) **Описание существующих разработок** 5

[**Глава 2.**](#_2et92p0) **Проектная часть** 6

[**2.1.**](#_tyjcwt) **Диаграмма прецедентов** 6

[**2.3.**](#_1t3h5sf) **Выбор инструментов** 7

[**2.5.**](#_4d34og8) **Проектирование сценария** 9

[**2.6.**](#_2s8eyo1) **Диаграмма классов** 10

[**2.7.**](#_17dp8vu) **Описание главного модуля** 11

[**2.8.**](#_3rdcrjn) **Описания спецификаций к модулям** 14

[**2.9.**](#_26in1rg) **Описание модулей:** 15

[**2.10.**](#_lnxbz9) **Описание текстовых наборов модулей** 27

[**2.11.**](#_35nkun2) **Описание применения средств отладки** 30

[**2.12.**](#_1ksv4uv) **Анализ оптимальности использования памяти и быстродействия** 30

[**Глава 3.**](#_44sinio) **Глава 3. Эксплуатационная часть** 31

[**3.1.**](#_2jxsxqh) **Руководство по установке.** 31

[**Заключение** 32](#_1y810tw)

[**Список литературы и интернет-источников** 33](#_4i7ojhp)

[**Приложения**](#_2xcytpi) 34

# **Введение**

Мир с каждым днём меняется, появляются новые технологии, но всё новое основывается на более ранних открытиях и изобретениях. Информационные технологии не являются исключением и одним из важных элементов являются множества.

Была поставлена цель реализовать операции над множествами и в будущем иметь возможность реализовать функционал операций над множествами в своем приложении.

Для реализации был использовал Dev-C++, который является свободной интегрированной средой разработки приложений для языков программирования “C/C++”

В первой части будут рассмотрены: предметная область, существующие решения.

Во второй части будут рассмотрены: диаграмма прецедентов, выбор инструментов, проектирования сценария, описание главного модуля, описание модулей, описание тестовых наборов модулей, описание применения средств отладки, анализ оптимальности использования памяти и быстродействия.

В третьей части будет освещено взаимодействие пользователя с программой, приведены изображения графических окон программы и сообщения оператору.

Для осуществления обозначенных целей служат следующие задачи: изучение теоретического материала и готовых решений, разработка программы и её функции, написание кода программы, руководство по установке.

# **Теоретическая часть**

## **Описание предметной области**

**Множество** является одним из ключевых понятий математики. Оно является математическим объектом, который является набором, совокупностью, собранием каких-либо объектов, которые называются **элементами** этого множества и обладают общим для всех их характеристическим свойством.

**Элемент множества** - Объект, из которых состоит множество.

**Пересечение множеств (**Рисунок 1**)** - это множество, которому принадлежат те и только те элементы, которые одновременно принадлежат всем данным множествам.

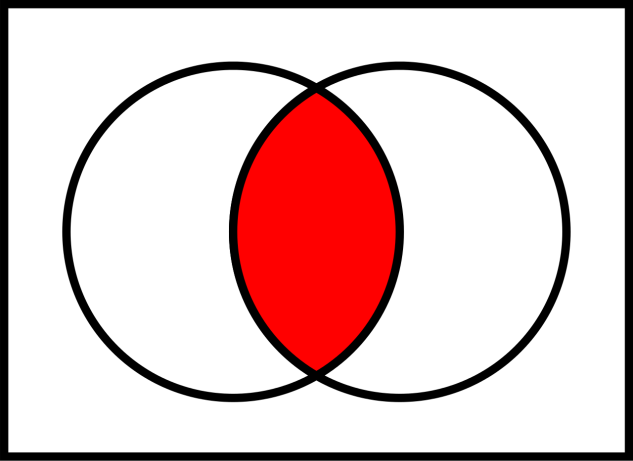


Рисунок 1 Пересечение множеств

**Объединение множеств (**Рисунок 2**)** - множество, содержащее в себе все элементы исходных множеств.

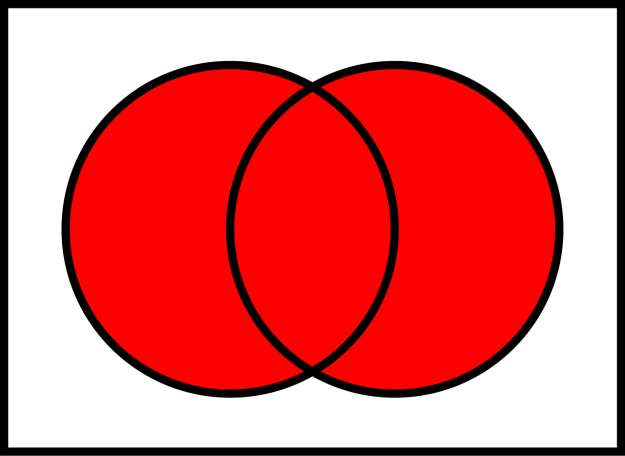


Рисунок 2 Объединение множеств

**Разность двух множеств (**Рисунок 3**)** - это операция, результатом которой является множество, в которое входят все элементы первого множества, не входящие во второе множество.

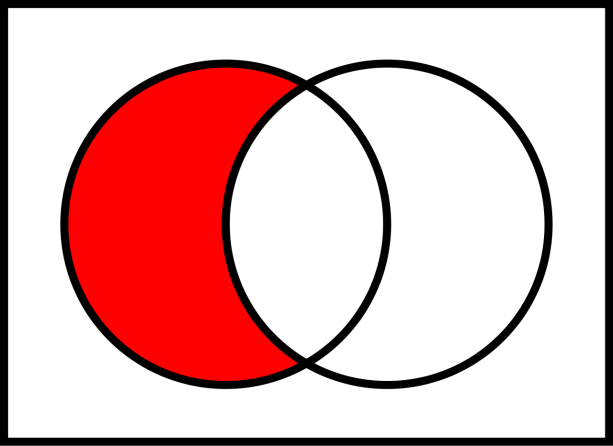


Рисунок 3 Разность двух множеств

**Симметрическая разность двух множеств (**Рисунок 4**)** - это операция, результатом которой является новое множество, включающее все элементы исходных множеств, не принадлежащие одновременно обоим исходным множествам.

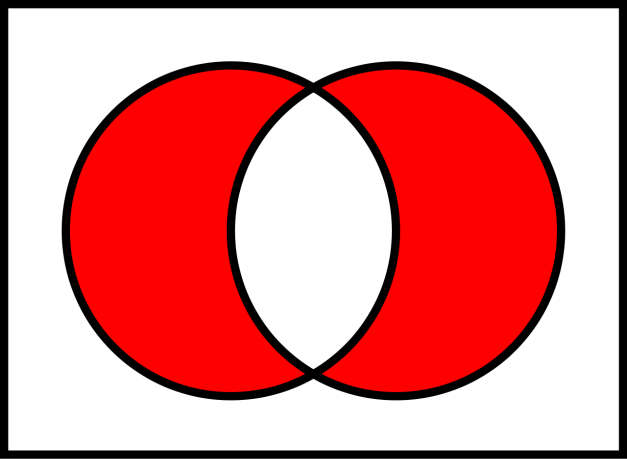


Рисунок 4 Симметрическая разность двух множеств

## **Описание существующих разработок**

Сравнение существующих реализаций представлено в таблице 1.

Вывод о существующих разработках: “Реализованный тип данных множество” упрощает и ускоряет работу с множествами, что делает его востребованным и множества активно используются в информационных технологиях. Например, для моделирования объектов реального мира и компактного отображения сложных логических взаимоотношений или для 2d и 3d графики.

Таблица 1

Сравнение реализаций множества.

|  |  |  |
| --- | --- | --- |
| Название языка программирования | Операции над множествами | Описание |
| Pascal | + | Объединение множеств |
| - | Пересечение множеств |
| \* | Разность множеств |
| >= | Проверка является множество “B” подмножеством “A |
| Python | Union | Объединение множеств |
| intersection | Пересечение множеств |
| Difference | Разность множеств |
| symmetric\_difference | Симметричная разность |
| >= | Проверка является множество “B” подмножеством “A” |

# **Проектная часть**

## **Диаграмма прецедентов**

Описание диаграммы прецедентов **(**Рисунок 5):

Разработчик выполняет поддержку программы посредством обновлений и сбором информации об ошибках в программе.

Системный администратор устанавливает обновления и выявляет ошибки программы.

Пользователь программы посредством вода множеств и выбора действия, которое выполнит программа, получает выходные данные.

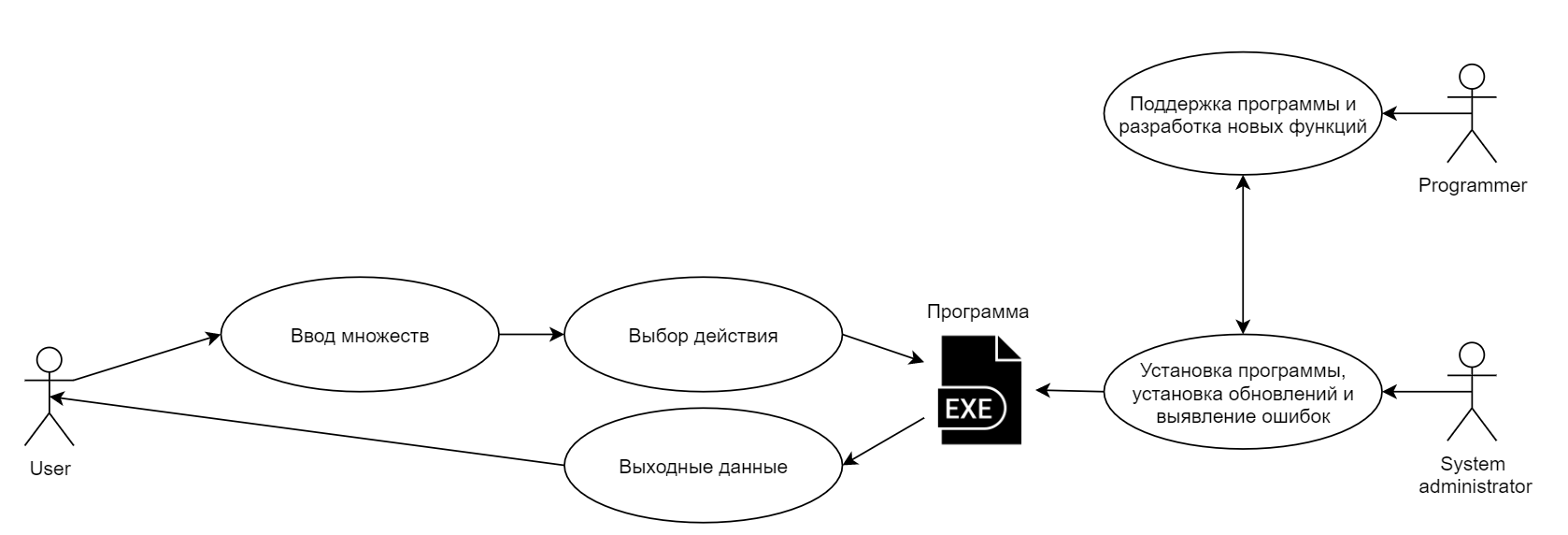


Рисунок 5 Диаграмма прецедентов

## 

## **Выбор инструментов**

Описание сред разработки:

**Eclipse[8]** - Свободная интегрированная среда разработки модульных кроссплатформенных приложений. Развивается и поддерживается Eclipse Foundation.

Распространяется с лицензией Eclipse Public License (EPL)

**NetBeans IDE[9]** — свободная интегрированная среда разработки приложений.

Проект NetBeans IDE поддерживается и спонсируется компанией Oracle, однако разработка NetBeans ведется независимым сообществом разработчиков-энтузиастов (NetBeans Community) и компанией NetBeans Org.

**Visual Studio Code[10]** - редактор исходного кода, разработанный Microsoft для Windows, Linux и macOS. Позиционируется как «лёгкий» редактор кода для кроссплатформенной разработки веб- и облачных приложений. Включает в себя отладчик, инструменты для работы с Git, подсветку синтаксиса, IntelliSense и средства для рефакторинга. Имеет широкие возможности для кастомизации: пользовательские темы, сочетания клавиш и файлы конфигурации. Распространяется бесплатно, разрабатывается как программное обеспечение с открытым исходным кодом, но готовые сборки распространяются под проприетарной лицензией.

**Code::Blocks[11]** — свободная кроссплатформенная среда разработки. Code::Blocks написана на С++ и использует библиотеку wxWidgets. Имея открытую архитектуру, может масштабироваться за счёт подключаемых модулей. Поддерживается и разрабатывается The Code::Blocks team. Распространяется по лицензии GNU General Public License

**Atom[12]** — бесплатный текстовый редактор с открытым исходным кодом для Linux, macOS, Windows с поддержкой плагинов, написанных на Node.js, и встраиваемых под управлением Git. Большинство плагинов имеют статус свободного программного обеспечения, разрабатываются и поддерживаются сообществом. Поддерживается и разрабатывается GitHub Inc. Распространяется по лицензии MIT

**Dev-C++[13]** — свободная интегрированная среда разработки приложений для языков программирования C/C++. В дистрибутив входит компилятор MinGW. Сам Dev-C++ написан на Delphi. Разрабатывался Bloodshed Software / Orwell. Распространяется согласно GPL.

В результате сравнения была выбрана среда Dev C++, так как она наиболее простая в использовании (не имеет лишнего функционала) меньшую затратность по ресурсам системы и имеет хорошую скорость сборки проектов. В сумме это даёт очень хорошие условия для создания небольших проектов (Таблица 2).

Сравнение сред разработки

Таблица 2

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Название  среды разработки | Доступность | Функциональность | Документация | Общая оценка |
| Eclipse | 4 | 5 | 4 | 4,3 |
|
|
| NetBeans | 4 | 4 | 5 | 4,3 |
|
|
| Visual Studio Code | 3 | 5 | 4 | 4 |
|
|
| Code::Block | 4 | 4 | 4 | 4 |
|
|
| Atom | 4 | 4 | 4 | 4 |
|
|
| Dev C++ | 5 | 4 | 5 | 4,6 |
|
|

## **Проектирование сценария**

Сценарий использования (Рисунок 6):

* Пользователь запускает программу
* Вводит множества
* Выбирает действие:
  + Вызов процедуры Объединения множеств
  + Вызов процедуры Пересечение множеств
  + Вызов процедуры Проверка на равенство множеств
  + Вызов процедуры Разность множеств
  + Вызов процедуры Симметричная разность множеств
  + Вызов процедуры Проверка является множество 'B' подмножеством 'A'
  + Вызов справочной информации о действиях
  + Выход из программы
* Вывод данных, если не был выбрано действие, то выход из программы.
* Возврат к выбору действия, если не был выбрано действие, то выход из программы.

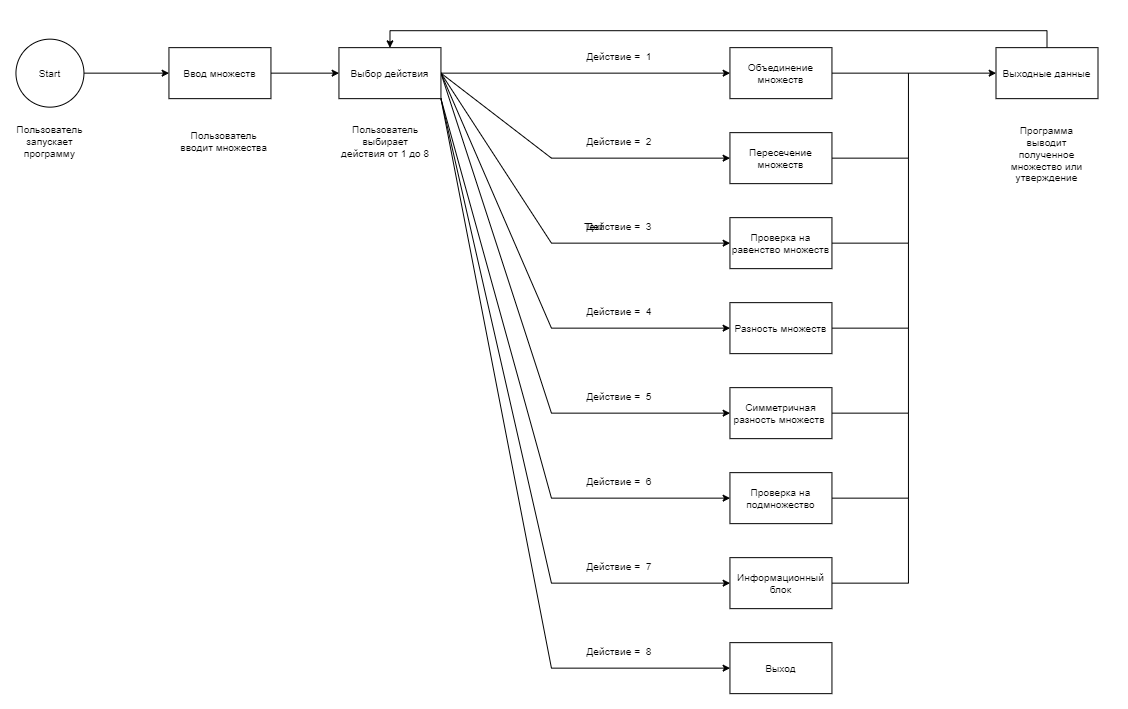


Рисунок 6 Сценарий использования

## **Диаграмма классов**

Описание диаграммы классов: Пользователь вводит данные, программа обращается к классу Set для того, чтобы вызвать процедуру для обработки действия.

Class Set содержит в себе процедуры:

* **Show** (Отображение результата)
* **get\_array** (Получение множеств от пользователя)
* **addition** (Объединение множеств)
* **disjunction** (Пересечение множеств)
* **equality** (Проверка на равенство множеств)
* **difference** (Разность множеств )
* **symmetrical\_difference** (Симметричная разность множеств)
* **subset** (Проверка является множество 'B' подмножеством 'A')

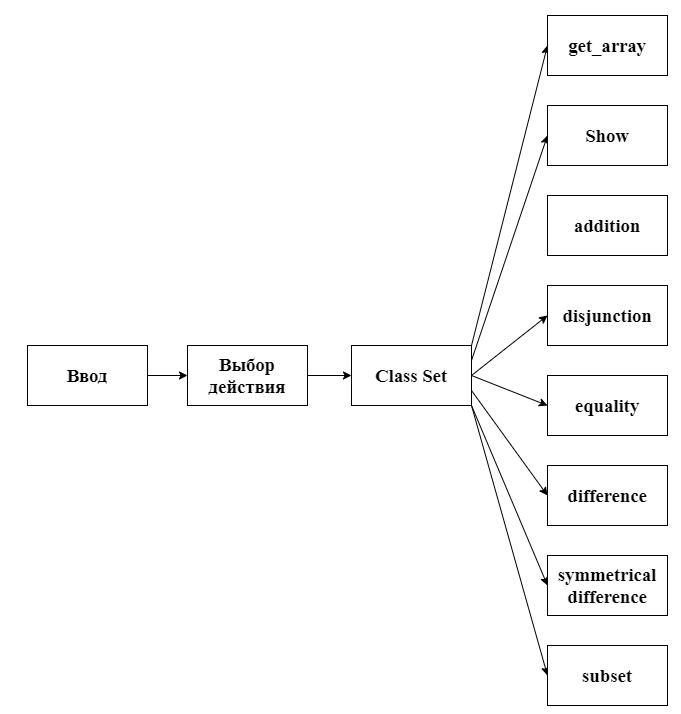


Рисунок 7 Диаграмма классов

## **Описание главного модуля**

Главный модуль состоит из 3 частей:

* Запрос множеств
* Циклическая часть
* Выбор действия

**Запрос множеств** представлен: Получением структуры из функционального модуля.

Листинг 1: **Запроса множеств:**

struct set Set\_A = get\_array();

//Получение множеств от пользователя

struct set Set\_B = get\_array();

//Получение множеств от пользователя

**Циклическая часть** представлена: циклом while который будет исполняться пока не изменится флаг.

**Выбор действий** представлен: Оператором switch, в который поступает выбранное действие.

Доступные действия:

* Вызов процедуры Объединения множеств
* Вызов процедуры Пересечение множеств
* Вызов процедуры Проверка на равенство множеств
* Вызов процедуры Разность множеств
* Вызов процедуры Симметричная разность множеств
* Вызов процедуры Проверка является множество 'B' подмножеством 'A'
* Вызов справочной информации о действиях
* Выход из программы

Листинг 2: **Выбор действия**:

switch(action){ // Выбор действия с множествами

case 1:

show(addition(Set\_A,Set\_B));// Объединение множеств

break;

case 2:

show(disjunction(Set\_A,Set\_B));//Пересечение множеств

break;

case 3:

printf("\n%d\n",equality(Set\_A,Set\_B));

//Проверка на равенство множеств

break;

case 4:

show(difference(Set\_A,Set\_B));//Разность множеств

break;

case 5:

show(symmetrical\_difference(Set\_A,Set\_B));

//Симметричная разность множеств

break;

case 6:

printf("\n%d\n",subset(Set\_A,Set\_B));

// Проверка является множество 'B' подмножеством 'A'

break;

case 7:

printf("\n1 - Unionem of sets; 2 - Intersectio of sets; 3 - Checking the equality of sets;");

printf("\n4 - Difference of sets; 5 - Symmetric difference of sets; 6 - Check if the set 'B' is a subset of 'A';\n8 - exit.\n");

break;

default : p = 1; return 0;

break;

}

## **Описания спецификаций к модулям**

Список модулей:

* **show**(Отображение результата)
* **get\_array**(Получение множеств от пользователя)
* **addition**(Объединение множеств)
* **disjunction**(Пересечение множеств)
* **equality**(Проверка на равенство множеств)
* **difference**(Разность множеств )
* **symmetrical\_difference**(Симметричная разность множеств)
* **subset**(Проверка является множество 'B' подмножеством 'A')

**show** (Отображение результата) описание: Функция получает множество представленное структурой. Если полученный массив из структуры пустой, то он выводит “Empty”

**get\_array** (Получение множеств от пользователя) описание: Получает из консоли строку. Возвращает множество.

**addition** (Объединение множеств) описание: Функция получает два множества представленные структурами. Возвращает множество.

**disjunction** (Пересечение множеств) описание: Функция получает два множества представленные структурами. Возвращает множество.

**equality** (Проверка на равенство множеств) описание: Функция получает два множества представленные структурами. Возвращает Bool значение.

**difference** (Разность множеств ) описание: Функция получает две структуры, которые состоят из массива и его размера.

**symmetrical\_difference** (Симметричная разность множеств) описание: Функция получает две структуры, которые состоят из массива и его размера. Возвращает множество.

**subset** (Проверка является множество 'B' подмножеством 'A') описание: Функция получает две структуры, которые состоят из массива и его размера. Возвращает Bool значение.

## **Описание модулей:**

show состоит из:

* Проверки на пустой массив
* Вывода полученного массива

Листинг 3: функции **show**:

void show(struct set data) //Отображение результата

{

printf("\n Set\_set =");

if (data.size > 0)

{

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

{

printf("%c ", data.set[i]);

}

printf("\n");

}

else

printf("Empty\n");

free(data.set);//Очищаем память

}

**get\_array** выполняет следующие действия:

* Выделяет подстроку массив;
* Заполняет её элементами множества.

**get\_array** состоит из:

* Считывания строк из консоли;
* Выделение памяти под массив;
* Разделения строки и записи её в массив.

Листинг 4: функции **get\_array:**

struct set get\_array(){//Получение множеств от пользователя

char str[in\_size];

char \*token, \*last;

struct set data;

data.size = 0;

printf("Enter set");

printf("\n Set =");

gets(str);//Получаем строку

data.set = (char\*)malloc(strlen(str) \* sizeof(int));//Выделяем место под моссив

token = strtok\_r(str, " ", &last);

while(token != NULL)//Записываем строку в массив

{

data.set[data.size] = \*token;

token = strtok\_r(NULL, " ", &last);

data.size++;

}

return data;

}

**addition** выполняет следующие действия:

* Вычисляет размер итогового массива;
* Выделяет место под массив; заполняет его элементами первого множества;
* Заполняет его элементами второго множества, которые подходят по условию.

**addition** состоит из:

* Вычисления размера итогового массива
* Выделения места под итоговый массив
* Заполнения массива элементами первого множества
* Заполнения массива элементами второго множества

Листинг 5: функции **addition:**

struct set addition(struct set Set\_A, struct set Set\_B){

// Объединение множеств

char \*work;

int k = 0;

struct set data;

data.size=Set\_A.size;

for (int i = 0; i < Set\_B.size; i++){//Вычисляем размер итогово моссива

for(int j = 0; j < Set\_A.size; j++){

if (Set\_A.set[j] != Set\_B.set[i]) k++;

}

if (k == Set\_A.size){

data.size++;

}

k=0;

}

data.set = (char\*)malloc(data.size sizeof(int));

//Выделяем место под массив

k = 0; data.size = Set\_A.size;

for (int i = 0; i < Set\_A.size; i++) data.set[i]=Set\_A.set[i];//Заполняем массив первым множеством

for (int i = 0; i < Set\_B.size; i++)

//Заполняем массив вторым множеством

{

for(int j = 0; j < Set\_A.size; j++)

{

if (Set\_A.set[j] != Set\_B.set[i]) k++;

}

if (k == Set\_A.size)

{

data.set[data.size]=Set\_B.set[i];

data.size++;

}

k=0;

}

return data; //Возвращаем результат

free(data.set);//Очищаем память

}

**disjunction** выполняет следующие действия:

* Вычисляет размер итогового массива, выделяет место под массив;
* Заполняет его элементами первого множества и заполняет его элементами второго множества, которые подходят по условию.

**disjunction** состоит из:

* Вычисления размера итогового массива
* Выделения места под итоговый массив
* Заполнения массива элементами первого множества
* Заполнения массива элементами второго множества

Листинг 6: функции **disjunction:**

struct set disjunction(struct set Set\_A, struct set Set\_B){//Пересечение множеств

char \*work;

struct set data;

data.size = 0;

for (int i = 0; i < Set\_B.size; i++){

//Вычисляем размер итогово моссива

for(int j = 0; j < Set\_A.size; j++){

if (Set\_A.set[j] == Set\_B.set[i]) data.size++;

}

}

data.set = (char\*)malloc(data.size \* sizeof(int));//Выделяем место под массив

data.size = 0;

for (int i = 0; i < Set\_A.size; i++) data.set[i]=Set\_A.set[i];//Заполняем массив первым множеством

for (int i = 0; i < Set\_B.size; i++)//Заполняем массив вторым множеством

{

for(int j = 0; j < Set\_A.size; j++)

{

if (Set\_A.set[j] == Set\_B.set[i])

{

data.set[data.size]=Set\_B.set[i];

data.size++;

}

}

}

return data; //Возвращаем результат

free(data.set);//Очищаем память

}

**equality** выполняет следующие действия:

* Поэлементно сравнивает множества;
* Возвращает результат посредством bool значения.

**equality** состоит из:

* Поэлементного сравнения
* Вывода результата

Листинг 7: функции **equality:**

bool equality(struct set Set\_A, struct set Set\_B){

//Проверка на равенство множеств

int k = 0;

for (int i = 0; i < Set\_B.size; i++){

//Проверяем на равенство по элементно

for(int j = 0; j < Set\_A.size; j++){

if (Set\_A.set[j] == Set\_B.set[i]) k++;

}

}

if (k = Set\_B.size && Set\_A.size == Set\_B.size) return true;

else return false;

}

**difference** выполняет следующие действия:

* Вычисляет размер итогового массива полученного в результате разности;
* Выделяет место под массив;
* Заполняет его элементами, которые не входят в первый массив.

**difference** состоит из:

* Вычисления размера итогового массива
* Выделения места под итоговый массив
* Заполнение массива

Листинг 8: функции **difference:**

struct set difference(struct set Set\_A, struct set Set\_B){//Разность множеств

char \*work;

int k = 0 ;

struct set data;

data.size = 0;

for (int i = 0; i < Set\_A.size; i++){

//Вычисляем размер итогово моссива

for(int j = 0; j < Set\_B.size; j++){

if (Set\_A.set[i] != Set\_B.set[j]) k++;

}

if (k == Set\_B.size) data.size++;

k=0;

}

data.set = (char\*)malloc(data.size \* sizeof(int));//Выделяем место под массив

data.size = 0; k = 0;

for (int i = 0; i < Set\_A.size; i++)//Заполняем массив

{

for(int j = 0; j < Set\_B.size; j++)

{

if (Set\_A.set[i] != Set\_B.set[j]) k++;

}

if(k == Set\_B.size){

data.set[data.size]=Set\_A.set[i];

data.size++;

}

k = 0;

}

return data; //Возвращаем результат

free(data.set);//Очищаем память

}

**symmetrical\_difference** выполняет следующие действия:

* Вычисляет размер итогового массива полученного в результате симметричной разности;
* Выделяет место под массив;
* Заполняет его элементами двух множеств кроме тех, которые присутствуют в обоих множествах.

**symmetrical\_difference** состоит из:

* Вычисления размера итогового массива ч.1
* Вычисления размера итогового массива ч.2
* Заполнения массива первым множеством
* Заполнения массива вторым множеством

Листинг 9: функции **symmetrical\_difference:**

struct set symmetrical\_difference(struct set Set\_A, struct set Set\_B){//Симметричная разность множеств

char \*work;

int k = 0 ;

struct set data;

data.size = 0;

for (int i = 0; i < Set\_A.size; i++){

//Вычисляем размер итогово моссива ч.1

for(int j = 0; j < Set\_B.size; j++){

if (Set\_A.set[i] != Set\_B.set[j]) k++;

}

if (k == Set\_B.size) data.size++;

k=0;

}

for (int i = 0; i < Set\_B.size; i++){

//Вычисляем размер итогово моссива ч.2

for(int j = 0; j < Set\_A.size; j++){

if (Set\_A.set[j] != Set\_B.set[i]) k++;

}

if (k == Set\_A.size) data.size++;

k=0;

}

data.set = (char\*)malloc(data.size \* sizeof(int));

data.size = 0; k = 0;

for (int i = 0; i < Set\_A.size; i++)

//Заполняем массив первым множеством

{

for(int j = 0; j < Set\_B.size; j++)

{

if (Set\_A.set[i] != Set\_B.set[j]) k++;

}

if(k == Set\_B.size){

data.set[data.size]=Set\_A.set[i];

data.size++;

}

k = 0;

}

for (int i = 0; i < Set\_B.size; i++)

//Заполняем массив вторым множеством

{

for(int j = 0; j < Set\_A.size; j++)

{

if (Set\_B.set[i] != Set\_A.set[j]) k++;

}

if(k == Set\_A.size){

data.set[data.size]=Set\_B.set[i];

data.size++;

}

k = 0;

}

return data; //Возвращаем результат

free(data.set);//Очищаем память

}

**subset** выполняет следующие действия:

* Поэлементно сравнивает множества;
* Затем выводит результат.

**subset** состоит из:

* Поэлементного сравнения
* Вывода результата

Листинг 10: функции **subset:**

bool subset(struct set Set\_A, struct set Set\_B){

// Проверка является множество 'B' подмножеством 'A'

int k = 0;

for (int i = 0; i < Set\_A.size ; i++)

//Поэлементно сравниваем множества

for(int j = 0; j < Set\_B.size ; j++){

if(Set\_A.set[i] == Set\_B.set[j]) k++;

}

if(k == Set\_B.size) return true;

else return false;

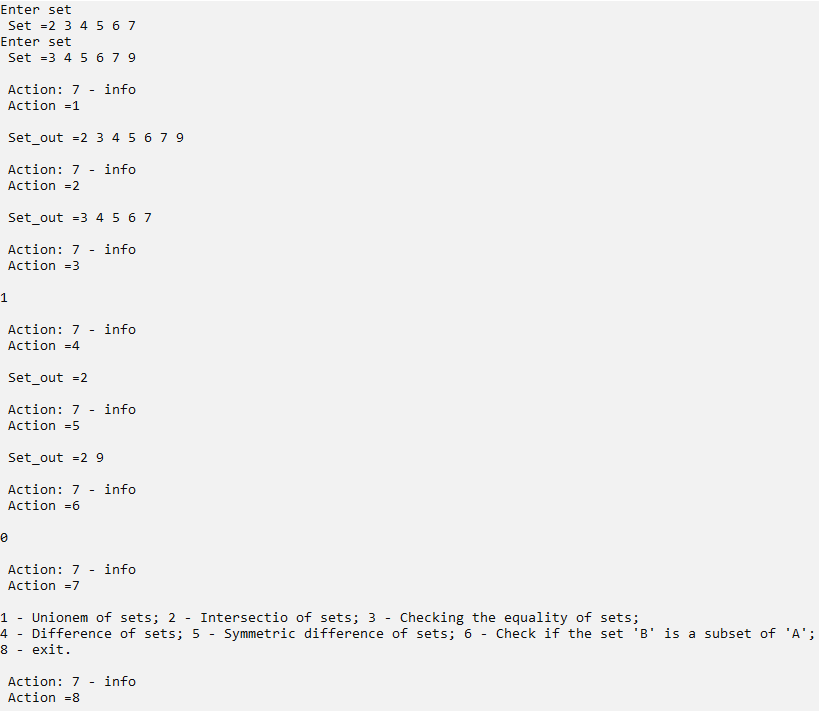
}

## **Описание текстовых наборов модулей**

Были разработаны 3 тестовых набора:

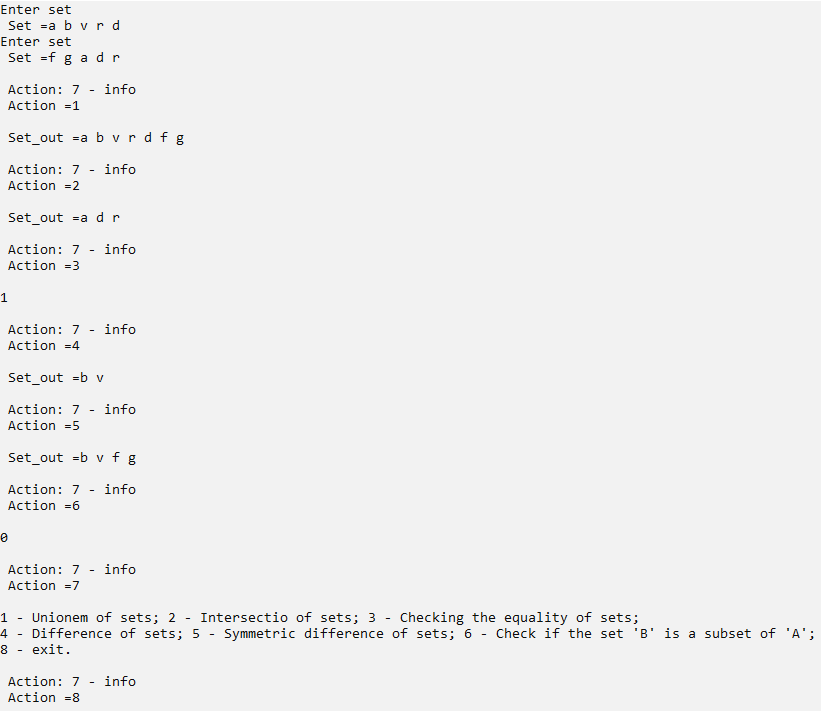
* Тестовый набор 1(Рисунок 8) содержащий одни цифры
* Тестовый набор 2(Рисунок 9) содержащий одни символы
* Тестовый набор 3(Рисунок 10) содержащий цифры и символы

Тестовый набор 1:



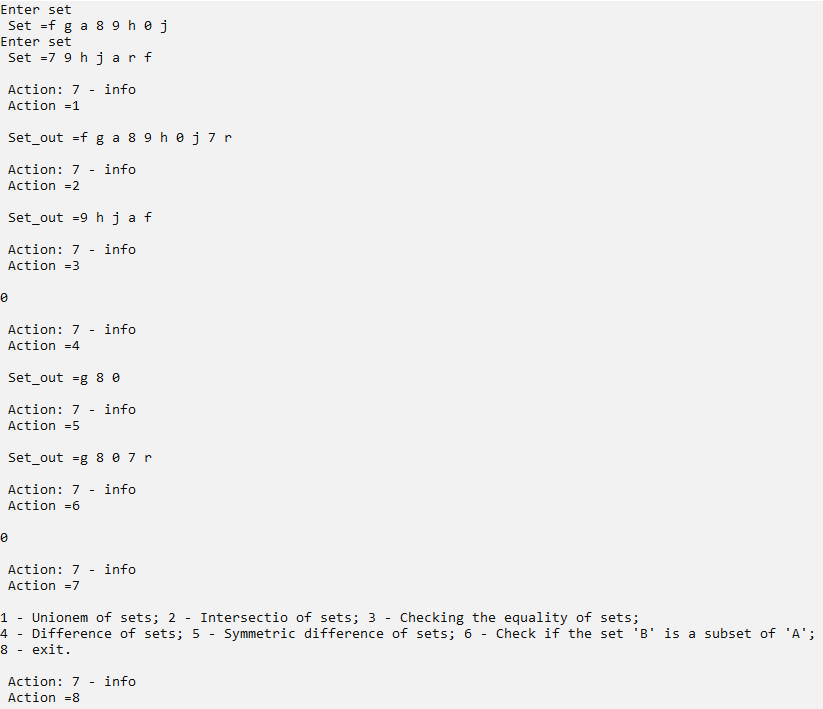
**Рисунок 8 Тестовый набор 1**

Тестовый набор 2:



**Рисунок 9 Тестовый набор 2**

Тестовый набор 3:



**Рисунок 10 Тестовый набор 3**

## **Описание применения средств отладки**

Для отладки использовался встроенный в среду разработки отладчик.

Отладка производилась посредством следующих действий:

* Отслеживания значений переменных;
* Проведение проверок процедур;
* Встраиванием отслеживающих флагов;
* Использование тестовых наборов.

## **Анализ оптимальности использования памяти и быстродействия**

Проект:

Информация от компилятора: Output Size: 135,3515625 KiB - Compilation Time: 0,19s

Максимальная нагрузка Загрузка ЦП 0,4%

Максимальная общая нагрузка на оперативную память 7,4 мб

Максимальная нагрузка раздельно на Рисунке 11.

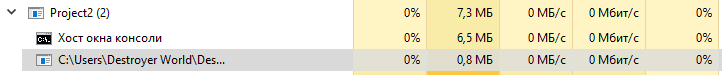


Рисунок 11 Максимальная нагрузка раздельно

Пустая программа:

Информация от компилятора: Output Size: 127,931640625 KiB

Compilation Time: 0,25s

Максимальная нагрузка Загрузка ЦП 0,1%

Максимальная общая нагрузка на оперативную память 6,5 мб

Максимальная нагрузка раздельно на Рисунке 12.

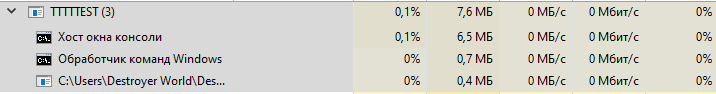


Рисунок 13 Максимальная нагрузка раздельно

Итог: Программа получилась не ресурсно-затратной, что делает её универсальной, так как она будет хорошо работать и на мощных системах, так и на слабых.

# **Эксплуатационная часть**

## **Руководство по установке.**

* Заходим на <https://github.com/DestroyerWorld/Coursework>
* Скачиваем Project out.zip(Рисунок 14)

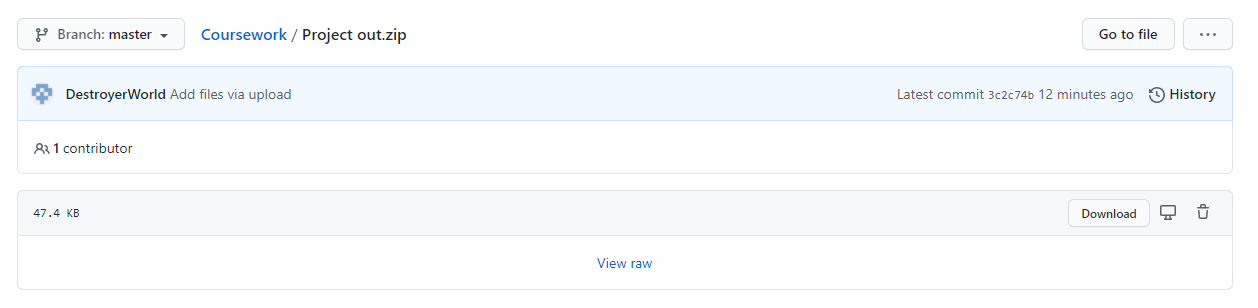


Рисунок 14 скачивание файла

* Распаковываем папку
* Запускаем start.exe (Рисунок 15)

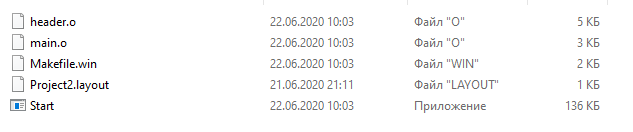


Рисунок 15 запуск программы

# 

# **Заключение**

В данном курсовом проекте были реализованы функции:

* **show**(Отображение результата)
* **get\_array**(Получение множеств от пользователя)
* **addition**(Объединение множеств)
* **disjunction**(Пересечение множеств)
* **equality**(Проверка на равенство множеств)
* **difference**(Разность множеств )
* **symmetrical\_difference**(Симметричная разность множеств)
* **subset**(Проверка является множество 'B' подмножеством 'A')

В результате проделанной работы был получен опыт по созданию разработке абстрактного типа данных, также были закреплены знания, что такое множество, элемент множества, пересечение множеств, проверка на равенство множеств, проверка на равенство множеств, разность двух множеств, симметричная разность множеств.

Так же планируется продолжать работу над данным проектом с целью создания кроссплатформенного приложения для работы с множествами.

# **Список литературы и интернет-источников**

**Литература:**

1. Виленкин Н. Я. Рассказы о множествах. 3-е издание, 2005 год.

**Интернет-источники:**

1. [https://ru.wikipedia.org/wiki/Множество](https://ru.wikipedia.org/wiki/%D0%9C%D0%BD%D0%BE%D0%B6%D0%B5%D1%81%D1%82%D0%B2%D0%BE)
2. [https://ru.wikipedia.org/wiki/Пересечение\_множеств](https://ru.wikipedia.org/wiki/%D0%9F%D0%B5%D1%80%D0%B5%D1%81%D0%B5%D1%87%D0%B5%D0%BD%D0%B8%D0%B5_%D0%BC%D0%BD%D0%BE%D0%B6%D0%B5%D1%81%D1%82%D0%B2)
3. [https://ru.wikipedia.org/wiki/Объединение\_множеств](https://ru.wikipedia.org/wiki/%D0%9E%D0%B1%D1%8A%D0%B5%D0%B4%D0%B8%D0%BD%D0%B5%D0%BD%D0%B8%D0%B5_%D0%BC%D0%BD%D0%BE%D0%B6%D0%B5%D1%81%D1%82%D0%B2)
4. [https://ru.wikipedia.org/wiki/Разность\_множеств](https://ru.wikipedia.org/wiki/%D0%A0%D0%B0%D0%B7%D0%BD%D0%BE%D1%81%D1%82%D1%8C_%D0%BC%D0%BD%D0%BE%D0%B6%D0%B5%D1%81%D1%82%D0%B2)
5. [https://ru.wikipedia.org/wiki/Симметрическая\_разность](https://ru.wikipedia.org/wiki/%D0%A1%D0%B8%D0%BC%D0%BC%D0%B5%D1%82%D1%80%D0%B8%D1%87%D0%B5%D1%81%D0%BA%D0%B0%D1%8F_%D1%80%D0%B0%D0%B7%D0%BD%D0%BE%D1%81%D1%82%D1%8C)
6. <https://pas1.ru/set0>
7. <https://python-scripts.com/sets#set-union>
8. <https://www.eclipse.org>
9. <https://netbeans.org>
10. <https://code.visualstudio.com>
11. <http://www.codeblocks.org>
12. <http://orwelldevcpp.blogspot.com>

# **Приложения**

Приложение 1: Main.cpp

/\*

Входные данные строка с однасимвальными членами множества без дубликатов через пробел

\*/

#include <stdlib.h>

#include <stdio.h>

#include "header.h"

#include<windows.h>

int main(int argc, char\*\* argv) {

int action = 0, p = 0;

system("color F0");

struct set Set\_A = get\_array();//Получение множеств от пользователя

struct set Set\_B = get\_array();//Получение множеств от пользователя

while (p != 1){

printf("\n Action: 7 - info");

printf("\n Action =");

scanf("%d",&action);

switch(action){ // Выбор действия с множествами

case 1:

show(addition(Set\_A,Set\_B));// Объединение множеств

break;

case 2:

show(disjunction(Set\_A,Set\_B));//Пересечение множеств

break;

case 3:

printf("\n%d\n",equality(Set\_A,Set\_B));//Проверка на равенство множеств

break;

case 4:

show(difference(Set\_A,Set\_B));//Разность множеств

break;

case 5:

show(symmetrical\_difference(Set\_A,Set\_B));//Симетричная разность множеств

break;

case 6:

printf("\n%d\n",subset(Set\_A,Set\_B));// Проверка является множество 'B' подмножеством 'A'

break;

case 7:

printf("\n1 - Unionem of sets; 2 - Intersectio of sets; 3 - Checking the equality of sets;");

printf("\n4 - Difference of sets; 5 - Symmetric difference of sets; 6 - Check if the set 'B' is a subset of 'A';\n8 - exit.\n");

break;

default : p = 1; return 0;

break;

}

}

system("pause");

return 0;

}

Приложение 2: header.h

#include <stdlib.h>

#include <stdio.h>

struct set{

char \*set;//Множество

int size;//Размер множества

};

struct set get\_array();//Получение множеств от пользвателя

void show(struct set data);//Отображение результата

struct set addition(struct set Set\_A, struct set Set\_B);//Объединение множеств

struct set disjunction(struct set Set\_A, struct set Set\_B);//Пересечение множеств

bool equality(struct set Set\_A, struct set Set\_B);//Проверка на равенство множеств

struct set difference(struct set Set\_A, struct set Set\_B);//Разность множеств

struct set symmetrical\_difference(struct set Set\_A, struct set Set\_B);//Симметричная разность множеств

bool subset(struct set Set\_A, struct set Set\_B);// Проверка является множество 'B' подмножеством 'A'

Приложение 3: header.cpp

#include <stdio.h>

#include <malloc.h>

#include <stdlib.h>

#include <string.h>

#include "header.h"

#define in\_size 256

struct set get\_array(){//Получение множеств от пользвателя

char str[in\_size];

char \*token, \*last;

struct set data;

data.size = 0;

printf("Enter set");

printf("\n Set =");

gets(str);//Получаем строку

data.set = (char\*)malloc(strlen(str) \* sizeof(int));//Выделяем место под моссив

token = strtok\_r(str, " ", &last);

while(token != NULL)//Записываем строку в массив

{

data.set[data.size] = \*token;

token = strtok\_r(NULL, " ", &last);

data.size++;

}

return data;

}

struct set addition(struct set Set\_A, struct set Set\_B){// Объединение множеств

char \*work;

int k = 0;

struct set data;

data.size=Set\_A.size;

for (int i = 0; i < Set\_B.size; i++){//Вычисляем размер итогово моссива

for(int j = 0; j < Set\_A.size; j++){

if (Set\_A.set[j] != Set\_B.set[i]) k++;

}

if (k == Set\_A.size){

data.size++;

}

k=0;

}

data.set = (char\*)malloc(data.size \* sizeof(int));//Выделяем место под массив

k = 0; data.size = Set\_A.size;

for (int i = 0; i < Set\_A.size; i++) data.set[i]=Set\_A.set[i];//Заполняем массив первым множеством

for (int i = 0; i < Set\_B.size; i++)//Заполняем массив вторым множеством

{

for(int j = 0; j < Set\_A.size; j++)

{

if (Set\_A.set[j] != Set\_B.set[i]) k++;

}

if (k == Set\_A.size)

{

data.set[data.size]=Set\_B.set[i];

data.size++;

}

k=0;

}

return data; //Возвращаем результат

free(data.set);//Очищаем память

}

struct set disjunction(struct set Set\_A, struct set Set\_B){//Пересечение множеств

char \*work;

struct set data;

data.size = 0;

for (int i = 0; i < Set\_B.size; i++){//Вычисляем размер итогово моссива

for(int j = 0; j < Set\_A.size; j++){

if (Set\_A.set[j] == Set\_B.set[i]) data.size++;

}

}

data.set = (char\*)malloc(data.size \* sizeof(int));//Выделяем место под моссив

data.size = 0;

for (int i = 0; i < Set\_A.size; i++) data.set[i]=Set\_A.set[i];//Заполняем моссив первым множеством

for (int i = 0; i < Set\_B.size; i++)//Заполняем моссив вторым множеством

{

for(int j = 0; j < Set\_A.size; j++)

{

if (Set\_A.set[j] == Set\_B.set[i])

{

data.set[data.size]=Set\_B.set[i];

data.size++;

}

}

}

return data; //Возвращаем результат

free(data.set);//Очищаем память

}

bool equality(struct set Set\_A, struct set Set\_B){//Проверка на равенство множеств

int k = 0;

for (int i = 0; i < Set\_B.size; i++){//Проверям на равенство по элементно

for(int j = 0; j < Set\_A.size; j++){

if (Set\_A.set[j] == Set\_B.set[i]) k++;

}

}

if (k = Set\_B.size && Set\_A.size == Set\_B.size) return true;

else return false;

}

struct set difference(struct set Set\_A, struct set Set\_B){//Разность множеств

char \*work;

int k = 0 ;

struct set data;

data.size = 0;

for (int i = 0; i < Set\_A.size; i++){//Вычисляем размер итогово моссива

for(int j = 0; j < Set\_B.size; j++){

if (Set\_A.set[i] != Set\_B.set[j]) k++;

}

if (k == Set\_B.size) data.size++;

k=0;

}

data.set = (char\*)malloc(data.size \* sizeof(int));//Выделяем место под моссив

data.size = 0; k = 0;

for (int i = 0; i < Set\_A.size; i++)//Заполняем моссив

{

for(int j = 0; j < Set\_B.size; j++)

{

if (Set\_A.set[i] != Set\_B.set[j]) k++;

}

if(k == Set\_B.size){

data.set[data.size]=Set\_A.set[i];

data.size++;

}

k = 0;

}

return data; //Возвращаем результат

free(data.set);//Очищаем память

}

struct set symmetrical\_difference(struct set Set\_A, struct set Set\_B){//Симетричная разность множеств

char \*work;

int k = 0 ;

struct set data;

data.size = 0;

for (int i = 0; i < Set\_A.size; i++){//Вычисляем размер итогово моссива ч.1

for(int j = 0; j < Set\_B.size; j++){

if (Set\_A.set[i] != Set\_B.set[j]) k++;

}

if (k == Set\_B.size) data.size++;

k=0;

}

for (int i = 0; i < Set\_B.size; i++){//Вычисляем размер итогово моссива ч.2

for(int j = 0; j < Set\_A.size; j++){

if (Set\_A.set[j] != Set\_B.set[i]) k++;

}

if (k == Set\_A.size) data.size++;

k=0;

}

data.set = (char\*)malloc(data.size \* sizeof(int));

data.size = 0; k = 0;

for (int i = 0; i < Set\_A.size; i++)//Заполняем моссив первым множеством

{

for(int j = 0; j < Set\_B.size; j++)

{

if (Set\_A.set[i] != Set\_B.set[j]) k++;

}

if(k == Set\_B.size){

data.set[data.size]=Set\_A.set[i];

data.size++;

}

k = 0;

}

for (int i = 0; i < Set\_B.size; i++)//Заполняем моссив вторым множеством

{

for(int j = 0; j < Set\_A.size; j++)

{

if (Set\_B.set[i] != Set\_A.set[j]) k++;

}

if(k == Set\_A.size){

data.set[data.size]=Set\_B.set[i];

data.size++;

}

k = 0;

}

return data; //Возвращаем результат

free(data.set);//Очищаем память

}

bool subset(struct set Set\_A, struct set Set\_B){// Проверка является множество 'B' подмножеством 'A'

int k = 0;

for (int i = 0; i < Set\_A.size ; i++)//Поэлементно сравниваем множества

for(int j = 0; j < Set\_B.size ; j++){

if(Set\_A.set[i] == Set\_B.set[j]) k++;

}

if(k == Set\_B.size) return true;

else return false;

}

void show(struct set data){//Отображение результата

printf("\n Set\_out =");

if (data.size > 0){

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

printf("%c ", data.set[i]);

}

printf("\n");

}

else printf("Empty\n");

free(data.set);//Очищаем память

}