Оглавление

Задание на курсовую работу

Написать программу, реализующую логическую структуру данных – дек множеств целого типа. Программа должна работать в диалоговом режиме. Каждая операция должна быть реализована в виде отдельной функции.

Дек и множество должны быть реализованы на базе структуры хранения двусвязный список.

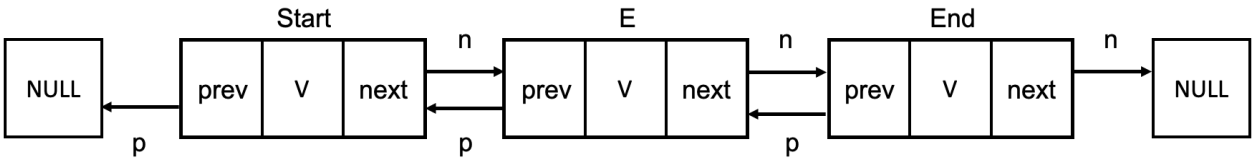
Написать отчёт по курсовой работе.

Описание структуры данных

Дек – это структура данных, позволяющая добавлять и удалять элементы как в начало, так и в конец. Реализуется на базе **двусвязного списка**.

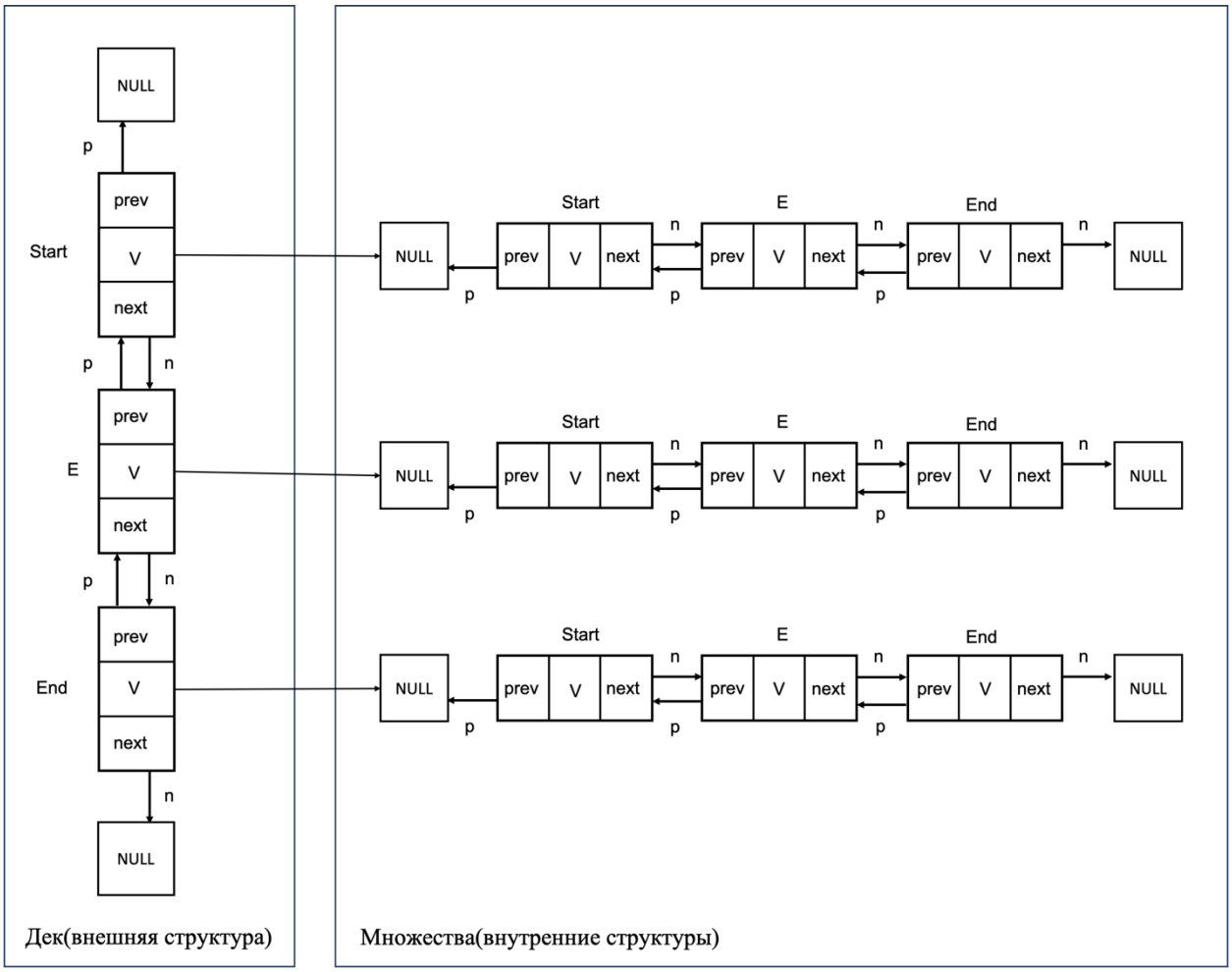
Множество – это структура данных, в которой элементы неупорядочены и нет повторяющихся значений элементов. Реализуется на базе **двусвязного списка**.

Двусвязный список – это структура данных, состоящая из элементов одного типа, связанных между собой последовательно посредством указателей. Каждый элемент списка имеет указатель на следующий и на предыдущий элементы. Указатель на следующий элемент списка у последнего элемента и указатель на предыдущий элемент и первого элемента списка указывают на NULL. Также должны быть указатели на начало списка и на конец.



v	Переменная, хранящая значение элемента списка
prev	Переменная, хранящая указатель на предыдущий элемент списка
next	Переменная, хранящая указатель на следующий элемент списка
n	Указатель на следующий элемент списка
p	Указатель на предыдущий элемент списка
Start	Первый элемент списка
End	Последний списка
E	Очередной элемент списка

Конечная схема реализуемой структуры данных



Список реализуемых функций для внешней структуры

1. Начать работу с деком
2. Очистить дек
3. Проверить, пуст ли дек
4. Показать значение начала дека
5. Показать значение конца дека
6. Удалить начало дека
7. Удалить конец дека
8. Взять элемент из начала дека
9. Взять элемент из конца дека
10. Изменить значение начала дека
11. Изменить значение конца дека
12. Добавить в начало дека
13. Добавить в конец дека
14. Распечатать дек
15. Закончить работу с деком
16. Закончить работу программы

Список реализуемых функций для внутренней структуры

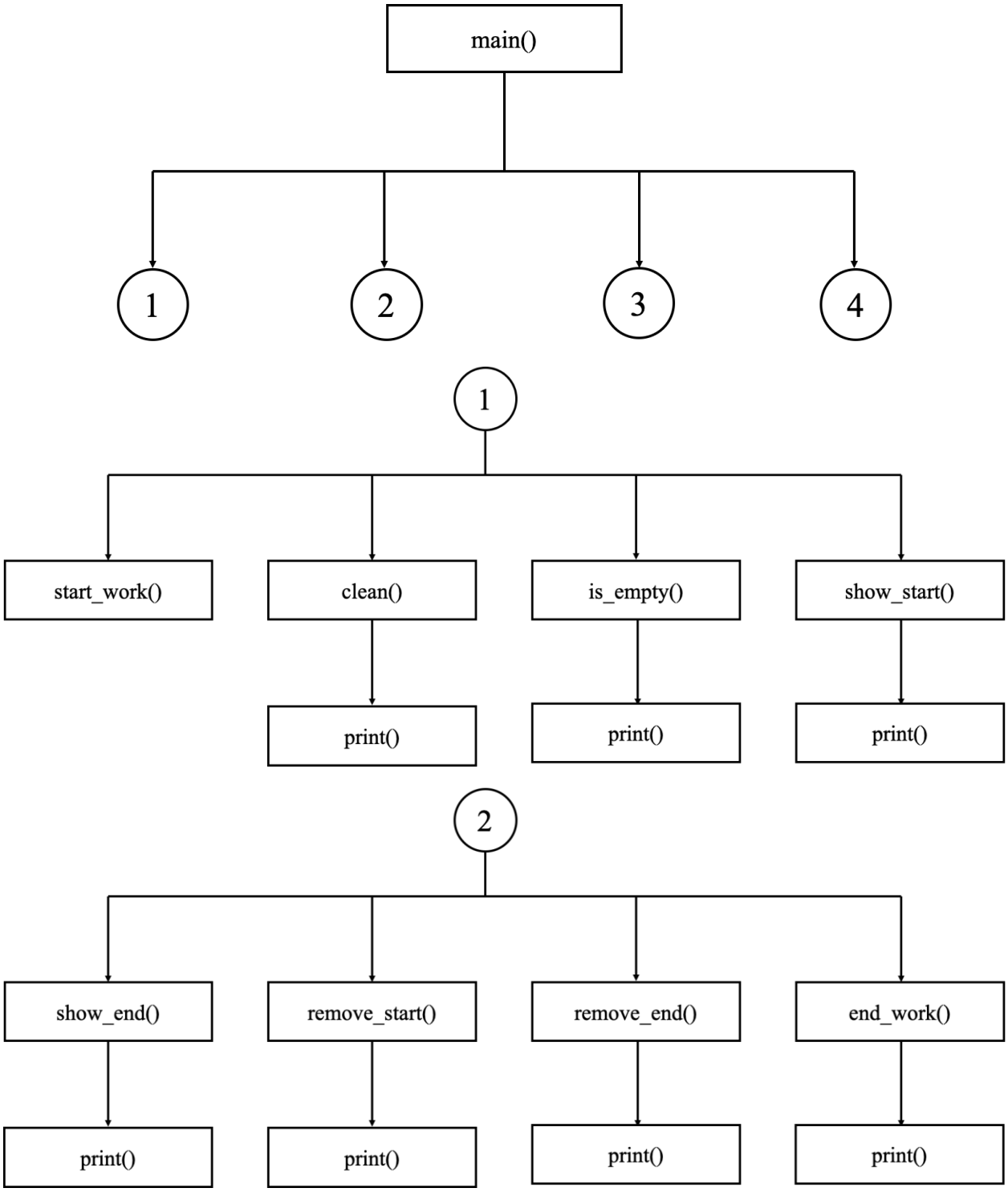
1. Начать работу с множеством
2. Очистить множество
3. Проверить, пусто ли множество
4. Удалить элемент из множества
5. Взять элемент из множества
6. Добавить элемент в множество
7. Проверить, принадлежит ли элемент множеству
8. Распечатать множество
9. Закончить работу с множеством

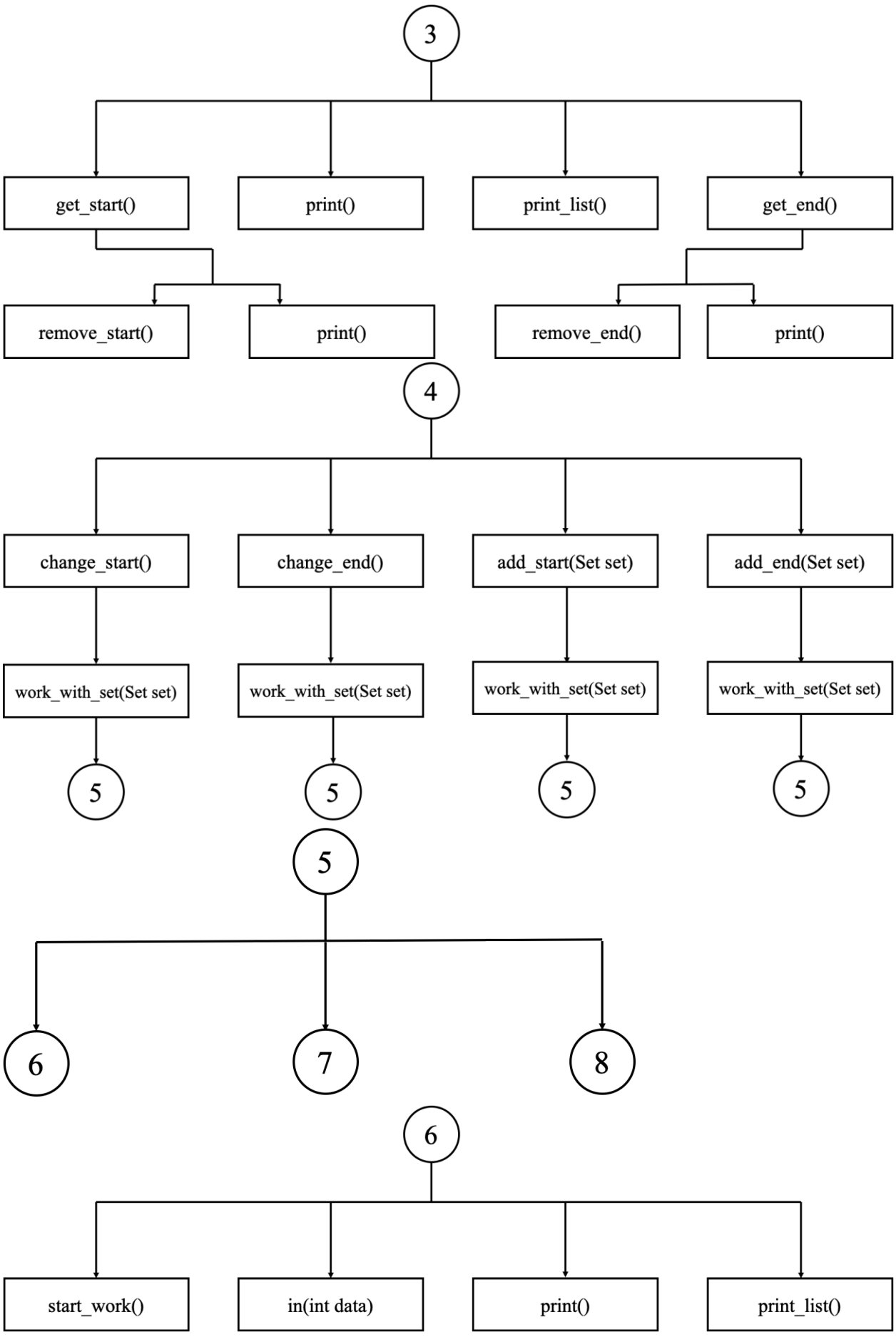
Описание структур на языке C++

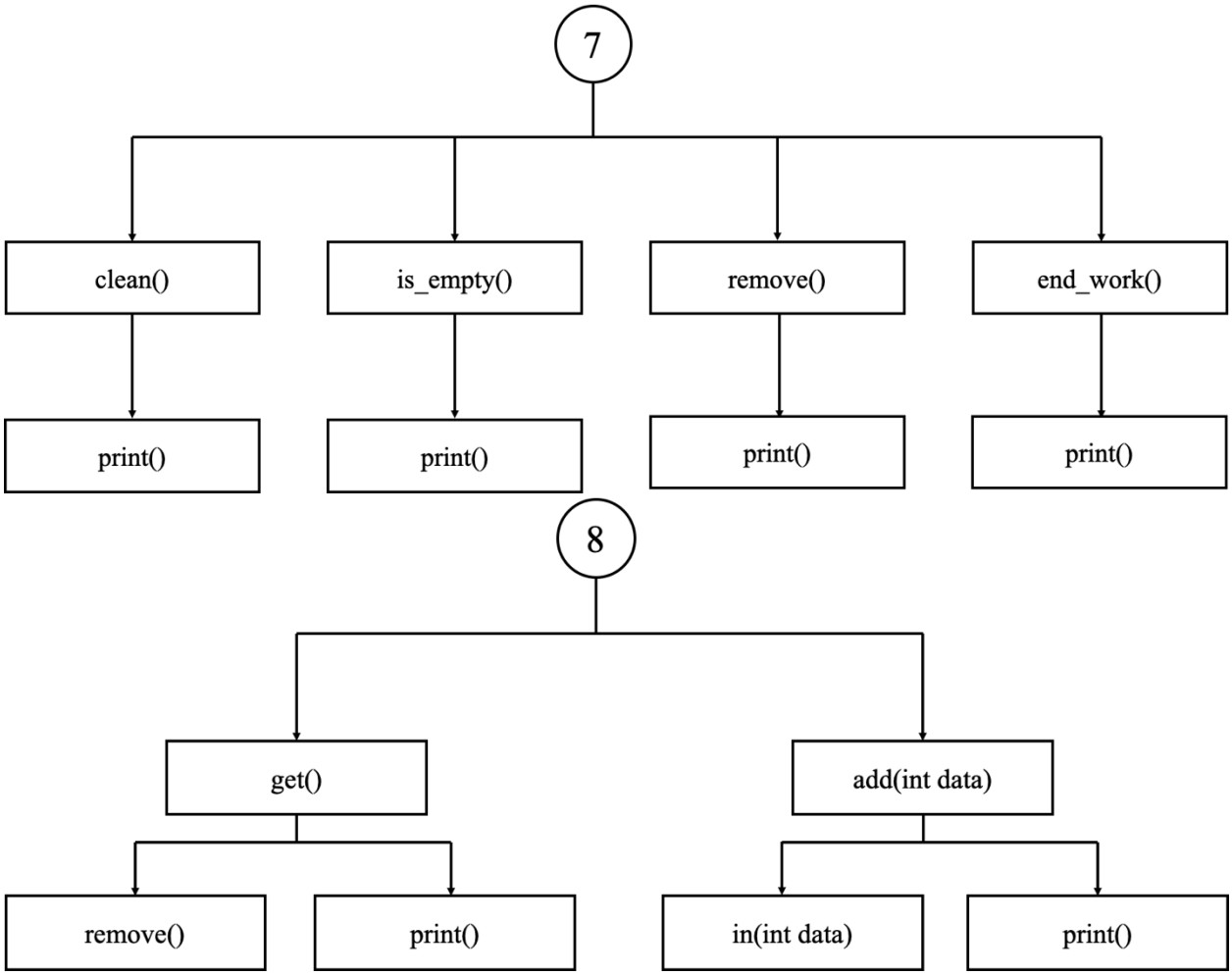
```
struct Node {        //Структура для хранения элемента множества
   int data;         //Значение элемента множества типа int
   Node *prev;       //Переменная для хранения указателя на след элемент
   Node *next;       //Переменная для хранения указателя на пред элемент
};

class Set {                 //Класс множества
   public:
      bool isStarted;       //Переменная для хранения состояния множества
      int size;             //Размер множества

   private:
      Node *start;          //Указатель на начало множества
      Node *end;            // Указатель на конец множества
};

struct DequeElement {    //Структура для хранения элемента дека
   Set data;             //Значение элемента дека - множество
   DequeElement *prev;   //Указатель на следующий элемент дека
   DequeElement *next;   //Указатель на предыдущий элемент дека
};

class Deque {
   public:
      bool isStarted;       //Переменная для хранения состояния множества
      int size;             //Размер множества
      int max_size;         //Глубина дека
```

```
    private:
        DequeElement *start;        //Указатель на начало дека
        DequeElement *end;          //Указатель на конец дека
};
```

Схема вызовов функций




3
get_start()
print()
print_list()
get_end()
remove_start()
print()
remove_end()
print()
4
change_start()
change_end()
add_start(Set set)
add_end(Set set)
work_with_set(Set set)
work_with_set(Set set)
work_with_set(Set set)
work_with_set(Set set)
5
5
5
5
5
6
7
8
6
start_work()
in(int data)
print()
print_list()


7
clean()
is_empty()
remove()
end_work()
print()
print()
print()
print()
8
get()
add(int data)
remove()
print()
in(int data)
print()

Список функций и их назначения

Для дека:

- start_work() – позволяет начать работу с деком
- clean() – очищает дек
- is_empty() – проверяет, пуст ли дек
- show_start() – показать значение начала дека
- show_end() – показать значение конца дека
- remove_start() – удаляет начало дека
- remove_end() – удаляет конец дека
- get_start() – получить начало дека
- get_end() – получить конец дека
- work_with_set(Set set) – создает множество или же использует существующее set для последующей работы с ним
- change_start() – изменяет значение начала
- change_end() – изменяет значение конца
- add_start(Set set) – добавляет множество set в начало дека
- add_end(Set set) – добавляет множество set в конец дека
- print() – вывести значение дека
- end_work() – закончить работу деком
- print_list() – вывести список функций для работы с деком

Для множества:

- start_work() – позволяет начать работу с множеством
- clean() – очищает множество
- is_empty() – проверяет, пусто ли множество
- get() – получить значение множества
- remove() – удалить значение из множества
- in(int data) – проверить, содержится ли элемент data в множестве
- add(int data) – добавить элемент в множество со значением data
- print() – вывести значение множества
- print_list() – вывести список функций для работы с множеством

Исходный код программы с комментариями

main.cpp

```
#include "deque.hpp"


int main() {
    int size = 5;
    Deque deque = Deque(size);      //Создаём дек
    DequeElement *element;      //Здесь будем хранить элемент дека при извлечечнии
    int command;
    while(true) {
        deque.print_list();         //Выводим список функций для дека
        std::cin >> command;        //Вводим номер функции
        switch(command) {
            case 1:
                deque.start_work();
                break;
            case 2:
                deque.clean();
                break;
            case 3:
                deque.is_empty();
                break;
            case 4:
                deque.show_start();
                break;
            case 5:
```

```
            deque.show_end();
            break;
        case 6:
            deque.remove_start();
            break;
        case 7:
            deque.remove_end();
            break;
        case 8:
            if(deque.isStarted) {              //Элемент можно получить только если мы начали работу
                element = deque.get_start();
                if(element != NULL)            //Если элемент существует, то выводим его значение
                    element->data.print();
            }
            else
                std::cout << "Вы не начали работу" << std::endl;
            break;
        case 9:
            if(deque.isStarted) {
                element = deque.get_end();
                if(element != NULL)
                    element->data.print();
            }
            else
                std::cout << "Вы не начали работу" << std::endl;
            break;
        case 10:
```

```
            deque.change_start();
            break;
        case 11:
            deque.change_end();
            break;
        case 12:
            deque.add_start();
            break;
        case 13:
            deque.add_end();
            break;
        case 14:
            deque.print();
            break;
        case 15:
            deque.end_work();
            break;
        case 16:
            if(deque.isStarted) {   //Закончить работу программы можно только если мы закончили работу с деком
                std::cout << "Вы не закончили работу с деком" << std::endl;
                break;
            }
            return 0;
      }
   }
   return 0;
}
```

deque.hpp

```
#include "set.hpp"

struct DequeElement {
   Set data;
   DequeElement *prev;
   DequeElement *next;
};

class Deque {
   public:
      Deque(int size) {
         start = end = NULL;      //При инициализации дека начало и конец равны NULL
         isStarted = false;
         this->size = 0;      //Текущий размер дека
         max_size = size;  //Передаем глубину дека
      }

      void start_work();
      void clean();
      void is_empty();
      void show_start();
      void show_end();
      void remove_start();
      void remove_end();
      DequeElement *get_start();
      DequeElement *get_end();
```

```
        Set work_with_set(Set);
        void change_start();
        void change_end();
        void add_start();
        void add_end();
        void print();
        void print_list();
        void end_work();
        bool isStarted;
        int size;
        int max_size;

    private:
        DequeElement *start;
        DequeElement *end;
};
```

deque.cpp

```
#include "deque.hpp"

void Deque::start_work() {
    isStarted = true;
    std::cout << "Вы начали работу с деком" << std::endl;
}

void Deque::clean() {
    if(isStarted == false) {
        std::cout << "Вы не начали работу" << std::endl;
```

```
        return;
    }

    if(start == NULL) {             //Если дек пуст, то очищать не надо
        std::cout << "Дек пуст" << std::endl;
        return;
    }

    DequeElement *tmp = start;
    DequeElement *next;
    while(tmp != NULL) {            //Проходим по всему деку и удаляем элементы
        next = tmp->next;
        free(tmp);
        tmp = next;
    }
    start = end = NULL;             //Начало и конец теперь NULL
    print();
    size = 0;
}

void Deque::is_empty() {
    if(isStarted == false) {
        std::cout << "Вы не начали работу" << std::endl;
        return;
    }

    if(start == NULL) {
        std::cout << "Дек пуст" << std::endl;
    }
```

```
    else {
        std::cout << "Дек непуст" << std::endl;
    }
    print();
}

void Deque::show_start() {
    if(isStarted == false) {
        std::cout << "Вы не начали работу" << std::endl;
        return;
    }

    if(start == NULL) {              //Если начало == NULL, то значит дек пуст
        std::cout << "Дек пуст" << std::endl;
        return;
    }

    start->data.print();       //Вызываем функцию вывода для множества
}

void Deque::show_end() {
    if(isStarted == false) {
        std::cout << "Вы не начали работу" << std::endl;
        return;
    }

    if(start == NULL) {
        std::cout << "Дек пуст" << std::endl;
        return;
```

```
    }

    end->data.print();              //Вызываем функцию вывода для множества
}

void Deque::remove_start() {
    if(isStarted == false) {
        std::cout << "Вы не начали работу" << std::endl;
        return;
    }

    if(start == NULL) {             //Если дек пуст, то удалять нечего
        std::cout << "Дек пуст" << std::endl;
        return;
    }

    DequeElement *to_del = start;
    if(start == end) {              //Если в деке только 1 элемент, то вызываем
функцию очистки
        clean();
    }
    else {
        start->next->prev = NULL;       //Теперь указатель на прошлый элемент у
нового начала == NULL
        start = start->next;            //Сдвигаем начало на следующий элемент
        free(to_del);                   //Удаляем старое начало
        size--;
    }
    print();
```

```
}

void Deque::remove_end() {
    if(isStarted == false) {
        std::cout << "Вы не начали работу" << std::endl;
        return;
    }

    if(start == NULL) {
        std::cout << "Дек пуст" << std::endl;
        return;
    }

    DequeElement *to_del = end;
    if(start == end) {
        clean();
        size = 0;
    }
    else {
        end->prev->next = NULL;                 //Теперь у нового конца указатель
на следующий элемент == NULL
        end = end->prev;            //Сдвигаем конец
        free(to_del);               //Удаляем старый конец
        size--;
    }
    print();
}

DequeElement* Deque::get_start() {
```

```
    if(isStarted == false) {
        std::cout << "Вы не начали работу" << std::endl;
        return NULL;
    }

    if(start == NULL) {                    //Если дек пуст, то и возвращать нечего
        std::cout << "Дек пуст" << std::endl;
        return NULL;
    }

    DequeElement *start_to_return = start;    //Сохраняем начало
    remove_start();                        //Удаляем начало
    start_to_return->data.print();         //Выводим значение начала
    print();                               //Вывод всего дека
    size--;
    return start_to_return;                //Возвращаем начало в соответствующую
переменную
}

DequeElement* Deque::get_end() {
    if(isStarted == false) {
        std::cout << "Вы не начали работу" << std::endl;
        return NULL;
    }

    if(start == NULL) {
        std::cout << "Дек пуст" << std::endl;
        return NULL;
    }
```

```
    DequeElement *end_to_return = end;              //Сохраняем конец
    remove_end();                                   //Удаляем конец
    end_to_return->data.print();                    //Выводим значение конца
    print();                                        //Выводим весь дек
    size--;
    return end_to_return;
}

Set Deque::work_with_set(Set _set = Set()) {        //Вызываем данную функцию, если нужно добавить новый элемент и или изменить старый
    Set set = _set;

    Node *element;
    int command, data;

    while(true) {
        set.print_list();       //Выводим список функций для множества
        std::cin >> command;    //Вводим номер функции
        switch(command) {
            case 1:
                set.start_work();
                break;
            case 2:
                set.clean();
                break;
            case 3:
                set.is_empty();
                break;
```

```
            case 4:
                set.remove();
                break;
            case 5:
                if(set.isStarted) {
                    element = set.get();
                    if(element != NULL) {   //Только когда извлечённый элемент не NULL можно вывести его значение
                        std::cout << "set data: ";
                        std::cout << element->data << std::endl;
                    }
                }
                else
                    std::cout << "Вы не начали работу" << std::endl;
                break;
            case 6:
                if(set.isStarted) {
                    std::cout << "Введите значение: ";
                    std::cin >> data;
                    set.add(data);
                }
                else
                    std::cout << "Вы не начали работу" << std::endl;
                break;
            case 7:
                if(set.isStarted) {
                    std::cout << "Введите значение: ";
                    std::cin >> data;
                    if(set.in(data))
```

```
                std::cout << "Да" << std::endl;
            else
                std::cout << "Нет" << std::endl;
            set.print();
        }
        else
            std::cout << "Вы не начали работу" << std::endl;
        break;
    case 8:
        if(set.isStarted) {
            set.print();
        }
        else
            std::cout << "Вы не начали работу" << std::endl;
        break;
    case 9:
        return set;
    }
  }
}

void Deque::change_start() {
  if(isStarted == false) {
    std::cout << "Вы не начали работу" << std::endl;
    return;
  }

  if(start == NULL) {        //Если дек пуст, то изменять нечего
    std::cout << "Дек пуст" << std::endl;
```

```
        return;
    }

    start->data.isStarted = false;
    Set set = work_with_set(start->data);   //Получаем множество из начала дека и обрабатываем его
    if(set.size == 0) {                     //Если мы его очистили, то его нужно удалить из дека
        remove_start();
        return;
    }
    start->data = set;                      //Кладем изменённое множество
}

void Deque::change_end() {
    if(isStarted == false) {
        std::cout << "Вы не начали работу" << std::endl;
        return;
    }

    if(start == NULL) {                 //Если дек пуст, то изменять нечего
        std::cout << "Дек пуст" << std::endl;
        return;
    }

    end->data.isStarted = false;
    Set set = work_with_set(end->data);     //Получаем множество из конца дека и обрабатываем его
    if(set.size == 0) {                     //Если в процессе обработки мы его очистили, то нужно удалить его из конца
```

```
        remove_end();
        return;
    }
    end->data = set;                    //Кладем изменённое множество
}

void Deque::add_start() {
    if(isStarted == false) {
        std::cout << "Вы не начали работу" << std::endl;
        return;
    }

    if(size == max_size) {              //Если дек полон, значит нельзя добавить
новый элемент
        std::cout << "В деке максимальное кол-во элементов" << std::endl;
        return;
    }

    Set set = work_with_set();          //Создаем новое множество и
обрабатываем его
    if(set.size == 0)                   //Если оно пустое, то не надо его добавлять
        return;
    DequeElement *new_start = (DequeElement *)malloc(sizeof(DequeElement));
//Создаем новый элемент
    new_start->data = set;

    if(start == NULL) {     //Если дек пуст
        new_start->next = NULL;
        start = end = new_start;    //То начало == концу
        print();
```

```
        size++;
        return;
    }

    new_start->next = start;
    if(start == end) {      //Если в деке только 1 элемент, то надо разъединить начало и конец
        start->next = end;  //У начала указатель на следующий теперь конец
        start = new_start;  //Обновляем начало
        end->prev = start;  //Указатель на предыдущий у конца теперь начало
        end->next = NULL;   //Указатель на следующий у конца теперь NULL
        print();
        size++;
        return;
    }

    start->prev = new_start;    //Указатель на предыдущий у старого начала теперь указывает на новое
    start = new_start;          //Обновляем начало
    print();
    size++;
}

void Deque::add_end() {
    if(isStarted == false) {
        std::cout << "Вы не начали работу" << std::endl;
        return;
    }
```

```
    if(size == max_size) {      //Если дек полон, то нельзя добавить новый элемент
        std::cout << "В деке максимальное кол-во элементов" << std::endl;
        return;
    }

    Set set = work_with_set();   //Создаём новое множество
    if(set.size == 0)            //Если оно пустое, то не добавляем его
        return;
    DequeElement *new_end = (DequeElement *)malloc(sizeof(DequeElement)); //Создаем новый элемент
    new_end->data = set;    //Кладем в значение наше новое множество

    if(start == NULL) {     //Если дек пуст
        start = end = new_end;  //То начало == концу
        print();
        size++;
        return;
    }

    new_end->prev = end;    //У нового конца указатель на предыдущий есть старый конец
    if(start == end) {      //Если в деке только 1 элемент
        end->prev = start;  //Указатель на предыдущий у старого конца есть начало
        end = new_end;      //Обновляем конец
        start->next = end;  //У начала указатель на следующий теперь конец
        start->prev = NULL; //Указатель на предыдущий теперь NULL
        end->next = NULL;   //Указатель на следующий у конца теперь NULL
        print();
```

```
        size++;
        return;
    }

    end->next = new_end;    //Теперь у старого конца есть указатель на следующий - это новый конец
    end = new_end;          //Обновляем конец
    print();
    size++;
}

void Deque::print() {
    if(isStarted == false) {
        std::cout << "Вы не начали работу" << std::endl;
        return;
    }

    DequeElement *current = start;
    while(current != NULL) {    //Сначала делаем прямой выводд
        current->data.print();  //Вызываем функцию вывода для множества
        current = current->next;
    }
    std::cout << std::endl;
    current = end;
    while(current != NULL) {    //Теперь обртаный вывод
        current->data.print();
        current = current->prev;
    }
    std::cout << std::endl;
```

```
}

void Deque::print_list() {
    std::cout << std::endl;
    std::cout << "1. Начать работу" << std::endl;
    std::cout << "2. Очистить" << std::endl;
    std::cout << "3. Проверить, пуст ли дек" << std::endl;
    std::cout << "4. Показать значение начала" << std::endl;
    std::cout << "5. Показать значение конца" << std::endl;
    std::cout << "6. Удалить начало" << std::endl;
    std::cout << "7. Удалить конец" << std::endl;
    std::cout << "8. Взять элемент из начала" << std::endl;
    std::cout << "9. Взять из элемент из конца" << std::endl;
    std::cout << "10. Изменить значение начала" << std::endl;
    std::cout << "11. Изменить значение конца" << std::endl;
    std::cout << "12. Добавить в начало" << std::endl;
    std::cout << "13. Добавить в конец" << std::endl;
    std::cout << "14. Распечатать" << std::endl;
    std::cout << "15. Закончить работу со структурой" << std::endl;
    std::cout << "16. Закончить работу программы" << std::endl;
    std::cout << "Введите номер команды: ";
}

void Deque::end_work() {
    clean();        //Очищаем дек, когда завершаем работу
    isStarted = false;
    std::cout << "Вы закончили работу с деком" << std::endl;
}
```

set.hpp

```
#include <stdio.h>
#include <iostream>

struct Node {
   int data;
   Node *prev;
   Node *next;
};

class Set {
   public:
      Set() {
         start = end = NULL; //Начало и конец == NULL
         isStarted = false;
         size = 0;   //Множество пусто -> размер == 0
      }

      void start_work();
      void clean();
      void is_empty();
      void remove();
      Node *get();
      void add(int);
      bool in(int);
      void print();
      void print_list();
      void end_work();
```

```
        bool isStarted;
        int size;

    private:
        Node *start;
        Node *end;
};
```

set.cpp

```
#include "set.hpp"

void Set::start_work() {
    isStarted = true; //Состояние работы - начали работу
    std::cout << "Вы начали работу с множеством" << std::endl;
}

void Set::clean() {
    if(isStarted == false) {
        std::cout << "Вы не начали работу" << std::endl;
        return;
    }

    if(start == NULL) { //Если множество пусто, то очищать ничего не надо
        std::cout << "Множество пусто" << std::endl;
        return;
    }

    Node *tmp = start;
    Node *next;
```

```
    while(tmp != NULL) { //Проходим по всему множеству и удаляем каждый элемент
        next = tmp->next;
        free(tmp);
        tmp = next;
    }
    start = end = NULL;
    print();
    size = 0;   //После очистки текущий размер дека равен 0
}

void Set::is_empty() {
    if(isStarted == false) {
        std::cout << "Вы не начали работу" << std::endl;
        return;
    }

    if(start == NULL) {
        std::cout << "Множество пусто" << std::endl;
    }
    else {
        std::cout << "Множество непусто" << std::endl;
    }
    print();
}

void Set::remove() { //Всегда удаляю первый элемент множества
    if(isStarted == false) {
        std::cout << "Вы не начали работу" << std::endl;
```

```
        return;
    }

    if(start == NULL) { //Если множество пусто, то удалять ничего не надо
        std::cout << "Множество пусто" << std::endl;
        return;
    }

    Node *to_del = start;
    if(start == end) { //Если один элемент в множестве, то вызываю функцию очистки
        clean();
    }
    else {
        start->next->prev = NULL;
        start = start->next;
        free(to_del);
        size--;
    }
    print();
}

Node* Set::get() {          //Всегда извлекаю первый элемент
    if(isStarted == false) {
        std::cout << "Вы не начали работу" << std::endl;
        return NULL;
    }

    if(start == NULL) {
```

```
        std::cout << "Множество пусто" << std::endl;
        return NULL;
    }

    Node *to_return = start;        //Сохраняем первый элемент
    remove();                       //Удаляем его
    std::cout << "Значение элемента: " << to_return->data << std::endl;
    print();     //Выводим всё множество
    size--;
    return to_return;//Возвращаю значение в соответствующую переменную
}

void Set::add(int data) {  //Всегда добавляю в начало множества
    if(isStarted == false) {
        std::cout << "Вы не начали работу" << std::endl;
        return;
    }

    if(in(data)) {      //Если элемент уже есть в множестве, то добавлять его не нужно
        std::cout << "Данный элемент уже содержится в множестве" << std::endl;
        return;
    }

    Node *new_node = (Node *)malloc(sizeof(Node));     //Создаем новый элемент
    new_node->data = data;                             //Кладем в него значение
    new_node->prev = NULL;                             //У первого элемента указатель на предыдущий всегда NULL
```

```
    if(start == NULL) {             //Если множество пусто то
        new_node->next = NULL;
        start = end = new_node;    //Начало == концу
        print();          //Выводим множество
        size++;                   //Увеличиваем текущий размер
        return;
    }

    new_node->next = start;
    if(start == end) {                  //Если в множестве всего 1 элемент, то надо разъединить начало и конец
        start->next = end;    //У начала указатель на следующий элемент теперь конец
        start = new_node;    //Начало теперь новый элемент
        end->prev = start;    //У конца указатель на предыдущий теперь начало
        end->next = NULL;//И указатель на следующий NULL
        print();
        size++;
        return;
    }

    start->prev = new_node;       //У старого начала теперь указатель на предыдущий есть новое начало
    start = new_node;                 //Меняем начало
    size++;
    print();
}

bool Set::in(int data) {
    if(isStarted == false) {
```

```
        std::cout << "Вы не начали работу" << std::endl;
        exit(1);
    }

    Node *current = start;
    while(current != NULL) { //Проходим по всему множеству и ищем совпадение с нашим значением
        if(current->data == data) {
            return true;
        }
        current = current->next;
    }
    return false;
}

void Set::print() {
    Node *current = start;
    while(current != NULL) {     //Сначала делаем прямой вывод
        std::cout << current->data << " ";
        current = current->next;
    }
    std::cout << std::endl;
    current = end;
    while(current != NULL) {     //Потом обратный
        std::cout << current->data << " ";
        current = current->prev;
    }
    std::cout << std::endl;
}
```

```
void Set::print_list() {
    std::cout << std::endl;
    std::cout << "1. Начать работу" << std::endl;
    std::cout << "2. Очистить множество" << std::endl;
    std::cout << "3. Проверить, пусто ли множество" << std::endl;
    std::cout << "4. Удалить элемент из множества" << std::endl;
    std::cout << "5. Взять элемент из множества" << std::endl;
    std::cout << "6. Добавить элемент в множество" << std::endl;
    std::cout << "7. Проверить принадлежит ли элемент множеству" <<
std::endl;
    std::cout << "8. Распечатать множество" << std::endl;
    std::cout << "9. Закончить работу" << std::endl;
    std::cout << "Введите команду: ";
}

void Set::end_work() {
    clean();    //Когда завершаем работу, надо очистить дек
    isStarted = false;
}
```