Комментарии к написанию отчета по ТР (2 семестр):

Напоминаю требования к содержанию **Части 1:**

Задание на Часть 1. (Смоделировать… /Создать новый тип…)
Определение моделируемой структуры данных.
Способ 1

Описание структуры нового типа, схема, программный код.
Описание порядка выполнения всех операций и их программный код.

Способ 2

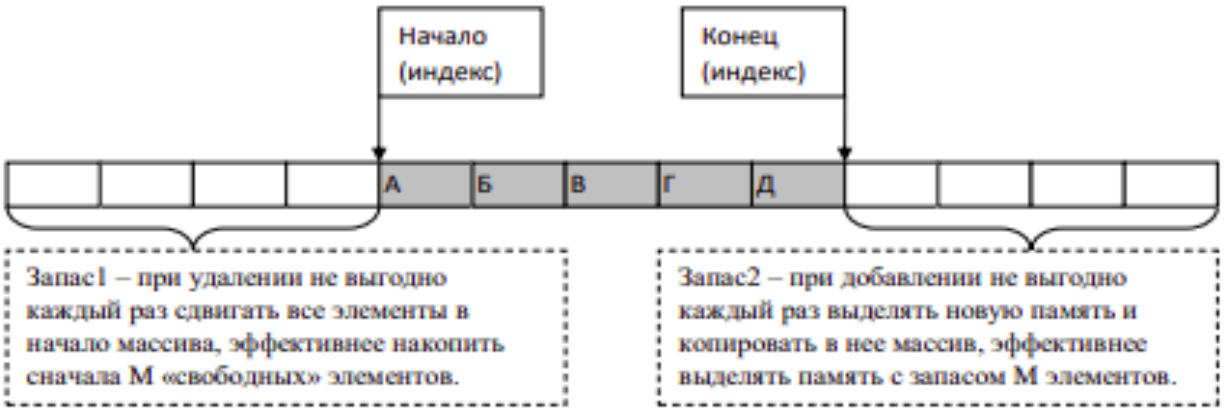
Описание структуры нового типа, схема, программный код.
Описание порядка выполнения всех операций и их программный код.

Задание на Часть 1 можно просто скопировать из файла с заданием.
Примеры определений и описаний остального:

Пример 1 (определение и начало описания типов для способа 1):

Список (англ. **list**) — **это** абстрактный тип данных, представляющий собой упорядоченный набор значений, в котором некоторое значение может встречаться более одного раза. Экземпляр списка является компьютерной реализацией математического понятия конечной последовательности.

Способ 1: на основе одномерного динамического массива;



Структура списка:

```
struct TList {
	TOrder* Mass;
	int begin, size, end;
};
```

Mass – массив, begin – кол-во дополнительных элементов в начале, size – размер самого списка, end – кол-во дополнительных элементов в конце.

Структура элемента:

```
struct TOrder {
	struct {
		int Day, Month, Year;
	} Data;
```

…

Пример 2 (Фрагмент более подробного описания типов)

1.2 Описание структуры «Матрица» на основе списка.

```
struct Node {
	int core;
	Node* right;
	Node* down;
};


typedef struct {

	int n;
	int m;
	Node* upperLeft;
	Node* current;
} Matrix;
```

Структура «Матрица» на основе списка реализована с помощью вспомогательного типа «Узел». Каждый элемент типа «Узел» содержит целое знаковое 32-битное число и два указателя. Один из них «right» - хранит адрес узла «справа» от текущего (или NULL). Другой, «down» - адрес узла «снизу» от текущего (или NULL).

Список – это множество объектов в оперативной памяти, которые хранят адреса друг друга. При этом они не обязаны лежать рядом. Пользователь списка обычно имеет доступ к первому узлу, и через него по цепочке ко всем другим.

Объект типа «Матрица» содержит 2 числа для хранения числа строк и столбцов. Также есть указатели на элемент типа «Узел» для хранения начала

…

Пример 3 (Описание базовых операций: код+абзац с пояснением)

Программный код модуля 1 (реализация на основе массива):
massive.cpp

```
#include "massive.h"

const TOrder Null_info = { 0, 0 , 0 , "-" , 0 , 0 };
const TList Null_List{ 0, 0 , 0, 0 };

bool Pusto(TList List) {
	if (List.size == 0)
		return true;
	else return false;
};
```

Проверка на пустоту списка: Если размер списка-массива равен 0, то возращается значение true(список пуст), иначе false(в списке есть элементы).

```
void List_of_one_Elem(TList& List, TOrder Order, int argc, char* argv[]) {
	List.begin = 5;
	List.size = 1;
	List.end = 5;
	List.Mass = new TOrder[List.begin + List.size + List.end];
	for (int i = 0; i < List.begin + List.size + List.end; i++)
		List.Mass[i] = Null_info;
	List.Mass[List.begin] = Order;
	return;
};
```

Создание списка из 1 элемента: кол-во дополнительных элементов в начале и в конце равно 5, размер становится 1. Создаем массив и заполняем его нулевой информацией. Первому элементу после дополнительной памяти в начале присваиваем значение считанной записи.

```
void AddFirst(TOrder Order, TList& List) {
```

Пример 4 (Описание базовых операций: прототип (код в приложении) +абзац с пояснением)

Создает пустую матрицу (выделяет память). Получает число строк и столбцов. В цикле создаётся n*m узлов типа Node. Каждый узел заполняется нулём. После этого двумерный список готов к заполнению полезными данными

```
Matrix* createEmptyMatrixNM(int N, int M)
```

Печать матрицы в файл или в консоль. Ф-ия получает указатель на объект матрица и если этот указатель не нулевой, начинает обход узлов. Главный цикл обходит по строкам, а внутренний по столбцам.

```
void printMatrix(FILE* f, Matrix* p)
```

Пример 5 (Описание базовых операций: код+абзац с пояснением справа)

- добавление элемента в список (в начало/в конец/после текущего);

```
void add_first(t_library* st, t_info& info)
{
	check_min_max(st);
	const auto tmp = new t_info[st->max_count];
	tmp[st->start_pos] = info;
	for (auto i = st->start_pos; i < st->count; i++)
	{
		tmp[i + 1] = st->info[i];
	}
	delete[] st->info;
	st->info = tmp;
	st->count++;
}

void add_last(t_library* st, t_info& info)
{
	check_min_max(st);
	st->info[st->count] = info;
	st->count++;
}

void add_after_pos(t_library* st, t_info& info)
{
	check_min_max(st);
	const auto tmp = new t_info[st->max_count];
	for (auto i = st->start_pos; i <= st->pos; i++)
	{
		tmp[i] = st->info[i];
```

/* Создание временного массива, первым элементом которого является новый элемент, а все последующие копируются из начального массива, освобожается памать, занимаемая начальным массивом и ему присваивается адрес временного. */

/* Провека размера буффера в конце массива, добавление нового элеента после посленего, увеличение количества на единицу. */

/* Копирование всех элементов до текущего во временный массив, добавление нового элемента во временный массив и копирование

Пример 6 (Описание базовых операций: код с предварительным пояснением параметром и выполняемой функции и с комментариями вместо абзаца пояснения — это хуже, так как текст несвязный получается, это просто набор фраз, а не основная идея этой функции!):

Созданы функции для реализации проекта

1.Проверка на пустоту дека. Параметры- дек, который передается по константой ссылке.

```
bool IsEmpty(const TDeq& book) {//дек пуст?
  if (book.End <= book.Start)//индекс старт стал больше конца
    return true; //значит пуст
  else if (book.End < 0)// вышел за пределы массива
    return true; //значит пуст
  else//иначе
    return false; //не пуст
}
```

2. Создание дека из 1 элемента. Параметры- дек, который передается по константой ссылке и флаг, который определяет направление(true, если слева направо, false-если справа налево)

```
void CreateDeq(TDeq& book, bool route) { //создание дека из одного элемента
  TBook* new_field;

  new_field = new TBook[1]; //выделяем память для нового элемента
  book.Start = 0; //индекс головы равен нулю
  book.End = 0; //индекс хвоста равен нулю
  book.field = new_field; //передаём информацию в поле
  book.Max = 1; //размер равен единице
  book.route = route; //направление задаём в параметрах
}
```

Пример 7 (Описание базовых операций сразу двумя способами, но **не хватает абзацев с пояснениями** к каждой из этих четырех функций, но среди случайной выборки просмотренных старых отчетов не нашла подходящего с пояснениями)

Проверка очереди на пустоту

Линейный односвязный список

```
bool Empty(Queue Q)
{
	if (Q.first == Q.last) return true;
	else return false;
}
```

Двоичный файла

```
bool Empty(Queue Q) {
	if (Q.k == 0) return true;
	else return false;
}
```

Добавление в конец очереди

Линейный односвязный список

```
void PushQueue(Queue* Q,Trab R)
{
	Q->last->next = new Node;
	Q->last->rab = R;
	Q->last = Q->last->next;
	Q->last->next = NULL;
	Q->k++;
}
```

Двоичный файла

```
void PushQueue(Queue* Q, Trab R) {
	FILE* f = fopen(Q->fileB, "ab");
	fwrite(&R, sizeof(R), 1, f);
	Q->k++;
	fclose(f);
}
```