Министерство образования и науки Российской Федерации Федеральное государственное бюджетное образовательное учреждение высшего образования
«Московский государственный технический университет имени Н.Э. Баумана
(национальный исследовательский университет)» (МГТУ им. Н.Э. Баумана)

ФАКУЛЬТЕТ Информатика и системы управления
КАФЕДРА Программное обеспечение ЭВМ и информационные технологии

ОТЧЕТ ПО УЧЕБНОЙ ДИСЦИПЛИНЕ “ТИПЫ И СТРУКТУРЫ ДАННЫХ” ЛАБОРАТОРНАЯ РАБОТА №5

Студент Ляпина Наталья Викторовна
фамилия, имя, отчество

Группа ИУ7-32Б

Вариант 3

Студент _____ Ляпина Н.В.
подпись, дата *фамилия, и.о.*

Преподаватель _____ Силантьева А.В.
подпись, дата *фамилия, и.о.*

2021 г.

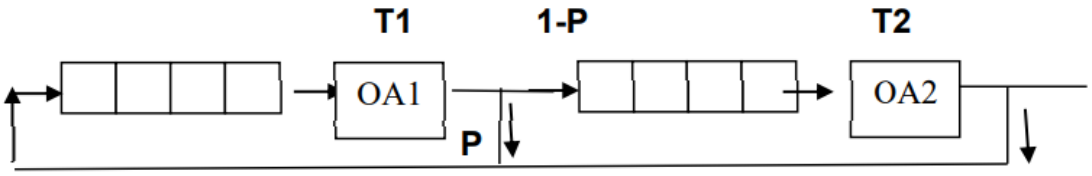
Оглавление

Задание

1. **Общее задание**
   отработка навыков работы с типом данных «очередь», представленным в виде одномерного массива и односвязного линейного списка. Сравнительный анализ реализации алгоритмов включения и исключения элементов из очереди при использовании двух указанных структур данных. Оценка эффективности программы (при различной реализации) по времени и по используемому объему памяти.

2. **Задание по варианту**

   Система массового обслуживания состоит из двух обслуживающих аппаратов (ОА1 и ОА2) и двух очередей заявок. Всего в системе обращается **100** заявок.



   Заявки поступают в "хвост" каждой очереди; в ОА они поступают из "головы" очереди по одной и обслуживаются по случайному закону за интервалы времени T1 и T2, равномерно распределенные **от 0 до 6** и **от 1 до 8** единиц времени соответственно (все времена – вещественного типа). Каждая заявка после ОА1 с вероятностью **P=0.7** вновь поступает в "хвост" первой очереди, совершая новый цикл обслуживания, а с вероятностью **1-P** входит во вторую очередь. В начале процесса все заявки находятся в первой очереди.

   Смоделировать процесс обслуживания до выхода из ОА2 первых **1000** заявок. Выдавать на экран после обслуживания в ОА2 каждых **100** заявок информацию о текущей и средней длине каждой очереди, а в конце процесса - общее время моделирования, время простоя ОА2, количество срабатываний ОА1, среднее времени пребывания заявок в очереди. Обеспечить по требованию пользователя выдачу на экран адресов элементов очереди при удалении и добавлении элементов. Проследить, возникает ли при этом фрагментация памяти.

3. **Входные данные**

- Команда для выбора действия (от 0 до 2)
  1. сравнение времени и памяти двух типов очередей
  2. моделирование процесса

- Количество добавляемых или удаляемых элементов очереди
- Непосредственно элементы очереди при добавлении

- **Требования к входным данным**

1) Нельзя добавлять больше элементов, чем допустимо его размером
2) Нельзя удалять элементов больше, чем есть в очереди
3) Количество добавляемых и удаляемых элементов должно быть натуральным числом
4) Очередь может принимать числа только типа int

4. Выходные данные
- Очередь, реализованная массивом
- Очередь, реализованная списком
- Массив освобожденных адресов списка

5. Действие программы

Программа выполняет добавление, удаление и обработку элементов очереди

5. Обращение к программе

Программа может быть вызвана через консоль компилятора с помощью команды “./app.exe”

6. Аварийные ситуации

В случае аварийной ситуации выводится сообщение об определенной ошибке.

Неверный ввод:

- Ошибка при вводе команды
- Ошибка при вводе кол-ва элементов для добавления
- Ошибка при вводе кол-ва элементов для удаления
- Ошибка при вводе элемента очереди

Структура данных

<u>Элемент очереди</u>

```
typedef struct
{
        double time;
        int type;
```

```
        int value;
} elem_t;
```
time - время обработки элемента
type - тип очереди
value - значение элемента

<u>Очередь из массива</u>
```
typedef struct
{
        elem_t *arr;
        elem_t *head;
        elem_t *tail;
        size_t size;
        size_t max_size;
} array_turn_t;
```

head - голова очереди
tail - хвост очереди
size - размер

Элемент списка
```
typedef struct node_turn
{
        elem_t data;
        struct node_turn *next;
} node_turn_t;
```

data - значение элемента
next - указатель на следующий элемент

Очередь из списка
```
typedef struct
{
        node_turn_t *head;
        node_turn_t *tali;
        size_t size;
}
```
head - голова очереди

tail - хвост очереди
size - размер
Массив свободных адрессов
typedef struct el_stack_list
{
node_turn_t **arr;
size_t size;
} arr_free_areas_t;
arr - массив
size - размер

Интерфейс модулей

Для обработки типа array_turn_t используются функции:

int count_add_elems_array_turn(int *num, int size)
// Функция подсчета количества добавляемых элементов
// num - количество добавляемых элементов, size - текущий размер

int read_elem(int *el)
// Функция чтение элемента очереди с клавиатуры
// el - элемент

int push_turn_array(array_turn_t *turn, int el)
// Функция добавления одного элемента в очередь
// el - значение элемента, turn - очередь

int add_elems_array_turn(array_turn_t *turn, int n)
// Функция добавление некоторого количества элементов в очередь
// n - количество элементов, turn- очередь

int random_elems_array_turn (array_turn_t *turn, int n)
// Функция добавления рандомных элементов в очередь
// n - количество элементов, turn - очередь

int count_del_elems_array_turn(int *num, int size)
// Функция подсчета количества удаляемых элементов
// num - количество элементов, size - текущий размер очереди

int pop_array(array_turn_t *turn)
// Функция удаления одного элемента

// turn - очередь

int del_elems_array_turn(array_turn_t *turn, int n)
// Функция удаления некоторого количества элементов
// n - количество элементов, turn - очередь

int print_arr(array_turn_t *turn)
// Функция печати очереди
// turn - очередь

Для обработки типа list_turn_t используются функции:

void free_list(list_turn_t *turn)
// Функция освобождения списка
// turn - очередь

int read_elem_list(int *el)
// Функция чтение элемента очереди с клавиатуры
// el - элемент

int push_turn_list(list_turn_t *turn, elem_t el, size_t max_size)
// Функция добавления одного элемента в очередь
// turn - очередь , el - элемент, max_size - максимальный размер очереди

int add_elems_list_turn(list_turn_t *turn, int n)
// Функция добавление некоторого количества элементов в очередь
// n - количество элементов, turn- очередь

int random_elems_list_turn(list_turn_t *turn, int n)
// Функция добавления рандомных элементов в очередь
// n - количество элементов, turn - очередь

int get_array_of_freed_areas(arr_free_areas_t *arr, int n, list_turn_t *turn)
// Функция создания массива освобожденных адресов
// arr - массив, turn -очередь , n - размер массива

int pop_turn_list(list_turn_t *turn, elem_t *el)
// Функция удаления одного элемента
// el - элемент, turn - очередь

```
int del_elems_list_turn(list_turn_t *turn, int n)
// Функция удаления некоторого количества элементов
// n - количество элементов, turn - очередь

int print_list(list_turn_t *turn)
// Функция печати очереди
// turn - очередь

void print_f_arr(arr_free_aread_t *arr)
// Функция печати массива освобожденных адресов
// arr - массив
```

Описание алгоритма

1. Вводится команда для выполнения определенной функции программы.
2. Выполняется введенная функция
   - Добавить элементы в очередь
   - Добавить случайные элементы в очередь
   - Удалить элементы из очереди
   - Вывести текущее состояние очереди (массив)
   - Вывести текущее состояние очереди (список)
   - Вывести массив освобожденных адресов
   - Смоделировать обработку очереди
   - Выйти из программы.
3. При ошибке выполнения функции выводится сообщение об ошибке и программа завершается с ненулевым кодом возврата.

Алгоритм обработки очередей (задание по варианту)

```
пока из ОА2 не вышло 1000 заявок
    добавить текущую длину очереди 1 к средней длине очереди 1

    если нельзя удалить элемент из первой очереди
        удалить элемент из 2 очереди
        количество ОА2 вышедших элементов++
        добавить элемент в очередь в зависимости от вероятности
        переход на след итерацию
    иначе
        количество вышедших из ОА1 заявок ++
        общее время работы ОА1 += время обработки текущей заявки 1 очереди
```

если длина 2 очереди равна 0
время простоя ОА2 += время обработки текущей заявки 1 очереди
обработать элемент вышедший из ОА1 в зависимости от вероятности
переход на новую итерацию

время обработки элемента 2 очереди = время элемента из головы очереди 2

если время обр. элемента 1 очереди >= времени обр. элемента 2 очереди
пока время обработки 1 элемента больше 0
если время обр. элемента 1 очереди >= времени обр. эл. 2 очереди
время обр. эл. 1 очереди -= время обр. эл. 2 очереди
средняя длина 2 очереди += длина 2 очереди
время работы ОА2 += время обр. эл. 2 очереди

удаление элемента из 2 очереди

добавление удаленного элемента в конец 1 очереди
проверка не вышло ли из ОА2 100 заявок (печать результатов, если да)

если длина 2 очереди = 0
время простоя ОА2 += время обр. элемента 1
добавление элемента в зависимости от вероятности
выход из цикла
время обр. элемента 2 = время обр. элемента из головы очереди 2
иначе
общее время ОА2 += время обр. 1 элемента
время обр. элемента головы 2 очереди -= врем. обр. эл. 1 очере
добавление элемента в зависимости от вероятности
выход из цикла
все пока
если время обр. элемента 1 очереди < времени обр. элемента 2 очереди
общее время ОА2 += время обр. эл. 1 очереди
время обр. элемента головы 2 очереди -= врем. обр. эл. 1 очереди
добавление элемента в зависимости от вероятности
все пока

Тесты

Негативные тесты

№	Описание	Вводимая структура	Результат
1	Недопустимый символ команде меню	a2	Ошибка
2	Несуществующая команда меню	10	Ошибка
3	Недопустимое количество добавляемых элементов	-2	Ошибка
4	Недопустимый количество удаляемых элементов	-10	Ошибка
5	Недопустимое значение элемента очереди	1.3	Ошибка

Позитивные тесты

№	Описание	Входные данные	Результат
1	Правильный пункт меню	comand = 1	Выбор действия
2	Добавление элемента в очередь	size = 2 1 1	Успешное чтение очереди
3	Добавление рандомных элементов в очередь	size = 3 1 40 3	Успешное заполнение очереди
4	Удаление элемента из очереди	size = 2 40 3	Успешное удаление из очереди

Оценка эффективности

Сравнение эффективности

Добавление

Кол-во элементов	Массив	Список
10	3139	12370
100	32465	108296
250	98265	302378
500	162157	644409
1000	489117	1090400

Удаление

Кол-во элементов	Массив	Список
10	4874	1178
100	9739	47126
250	23491	124643
500	47083	323313
1000	99309	472548

Память

Кол-во элементов	Процент заполненности	Массив	Список
10	25%	192	64
10	50%	192	128
10	100%	192	256
100	25%	1632	816
100	50%	1632	1208
100	100%	1632	2416

250	25%	4032	1504
250	50%	4032	3008
250	100%	4032	6016
500	25%	8032	3004
500	50%	8032	6008
500	100%	8032	12016
1000	25%	16032	6004
1000	50%	16032	12008
1000	100%	16032	24016

Результаты моделирования

```
Рабочее время ОА1 : 10087.645802 (ожидаемое: 10000.000000, погрешность: 0.876458%)
Рабочее время ОА2 : 4613.360393 (ожидаемое: 4500.000000, погрешность: 2.519120%)
Количество срабатываний ОА1 : 3338 (ожидаемое: 3333)
Время простоя ОА2 : 5474.285409
Теоретическое время простоя: 5500.000000
```

Вывод

Если очередь реализована циклическим массивом, то добавление в неё новых элементов будет происходить быстрее (в 4 раза для 10 элементов и в 2 раза для 1000 элементов), чем в очередь, реализованную списком.

Удаление элементов из массива происходит быстрее (в 4 раза для 100 элементов и в 4 раз для 1000 элементов), чем из списка.

Список также проигрывает по памяти циклическому массиву в 1,5 - 2 раза, при заполненности массива в 100%. При заполненности массива в 66% и меньше список выигрывает по памяти.

В ходе тестирования программы фрагментация данных очереди не выявлена.

Ответы на контрольные вопросы

1. Что такое FIFO и LIFO?

FIFO - первым пришел - первым ушел.

LIFO - последним пришел – первым ушел,

2. Каким образом, и какой объем памяти выделяется под хранение очереди при различной ее реализации?
При хранении кольцевым массивом: кол-во элементов * размер одного элемента очереди. Память выделяется на стеке при компиляции, если массив статический. Либо память выделяется в куче, если массив динамический. При хранении списком: кол-во элементов * (размер одного элемента очереди + указатель на следующий элемент). Память выделяется в куче для каждого элемента отдельно.

3. Каким образом освобождается память при удалении элемента из очереди при ее различной реализации?
При хранении кольцевым массивом память не освобождается, а просто меняется указатель на конец очереди. При хранении списком, память под удаляемый кусок освобождается.

4. Что происходит с элементами очереди при ее просмотре?
Эти элементы удаляются из очереди.

5. Каким образом эффективнее реализовывать очередь. От чего это зависит? Зная максимальный размер очереди, лучше всего использовать кольцевой статический массив. Не зная максимальный размер, стоит использовать связанный список, так как такую очередь можно будет переполнить только если закончится оперативная память.

6. Каковы достоинства и недостатки различных реализаций очереди в зависимости от выполняемых над ней операций?
При использовании кольцевого массива тратится больше времени на обработку операций с очередь, а так же может возникнуть фрагментация памяти. При реализации статическим кольцевым массивом, очередь всегда ограничена по размеру.

7.Что такое фрагментация памяти?
Фрагментация памяти — разбиение памяти на куски, которые лежат не рядом друг с другом. Можно сказать, что это чередование свободных и занятых кусков памяти.

8. Для чего нужен алгоритм “близнецов”?
Алгоритм близнецов снижает фрагментацию памяти и ускоряет поиск блоков

9. Какие дисциплины выделения памяти вы знаете?
1) Самый подходящий
2) Первый подходящий

По дисциплине "самый подходящий" выделяется тот свободный участок, размер которого равен запрошенному или превышает его на минимальную величину.
По дисциплине "первый подходящий" выделяется первый же найденный свободный участок, размер которого не меньше запрошенного (эффективнее).

10. На что необходимо обратить внимание при тестировании программы?

Корректное освобождение памяти. Проверка переполнения или проверка удаления из пустой очереди.

11. Каким образом физически выделяется и освобождается память при динамических запросах?

Программа запрашивает блок памяти необходимого размера. ОС находит подходящий блок, записывает его адрес и размер в таблицу адресов, а затем возвращает данный адрес в программу. При запросе на освобождение указанного блока программы, ОС убирает его из таблицы адресов, адрес считается освобожденным. При попытке доступа к освобожденной памяти могут возникнуть ошибки.