|  |  |  |
| --- | --- | --- |
|  | |  | | --- | | **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 ЛАБОРАТОРНОЙ РАБОТЕ 5

По дисциплине «Типы и структуре данных»

### Название Работа со очередью

### Студент Мансуров Владислав Михайлович

*фамилия, имя, отчество*

### Группа ИУ7-36Б

Тип лабораторной работы Учебная

### Название

предприятия

|  |  |  |
| --- | --- | --- |
| Студент |  | Мансуров В.М. |
| Преподаватель | *подпись, дата* | *фамилия, и.о.*  Никульшина Т.А. |
|  | *подпись, дата* | *фамилия, и.о.* |

*2021 г.*

***Содержание***

### Условие задачи 4

### Техническое задание… 4

### Описание алгоритма 5

### Анализ алгоритмов… 13

### Контрольные вопросы… 17

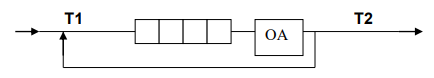
### Заключение… 19

***Цель работы*** *- 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.*

***Условия задачи***

***Вариант 7***

Система массового обслуживания состоит из обслуживающего аппарата (**ОА**) и очереди заявок.

******

*Заявки поступают в "хвост" очереди по случайному закону с интервалом времени* ***Т1****, равномерно распределенным* от 0 до 6 *единиц времени (е.в.). В* ***ОА*** *они поступают из "головы" очереди по одной и обслуживаются также равновероятно за время* ***Т2***от 0 до 1 *е.в., Каждая заявка после ОА вновь поступает в "хвост" очереди, совершая всего* 5 циклов *обслуживания, после чего покидает систему (все времена – вещественного типа). В начале процесса в системе заявок нет.*

***Техническое задание***

*Смоделировать процесс обслуживания до ухода из системы первых* ***1000******заявок****, выдавая после обслуживания каждых* ***100 заявок*** *информацию о текущей и средней длине очереди, а в конце процесса - общее время моделирования и количестве вошедших в систему и вышедших из нее заявок, количестве срабатываний* ***ОА****, время простоя аппарата. По требованию пользователя выдать на экран адресов элементов очереди при удалении и добавлении элементов. Проследить, возникает ли при этом фрагментация памяти.*

*Система массового обслуживания состоит из обслуживающих аппаратов (****ОА****) и очередей заявок двух типов, различающихся временем прихода и обработки. Заявки поступают в очереди по случайному закону с различными интервалами времени (в зависимости от варианта задания), равномерно распределенными от начального значения (иногда от нуля) до максимального количества единиц времени. В* ***ОА*** *заявки поступают из «головы» очереди по одной и обслуживаются за указанные в задании времена, распределенные равновероятно от минимального до максимального значений (все времена – вещественного типа).*

*Очередь необходимо представить в виде вектора и списка. Все операции должны быть оформлены подпрограммами****. Алгоритм для реализации задачи один, независимо от формы представления очереди****. Необходимо сравнить эффективность различного представления очереди по времени выполнения программы и по требуемой памяти. При реализации очереди списком нужно проследить, каким образом происходит выделение и освобождение участков памяти, для чего по запросу пользователя необходимо выдать на экран адреса памяти, содержащие элементы очереди при добавлении или удалении очередного элемента.*

***Входные данные***

В качестве указания пункта меню – целое число.

В подпунктах изменения времен обработки и прихода – по 2 целых числа

***Выходные данные***

При выводе каждых 100 вышедших заявок – результаты моделирования (см. условие задачи)

При выходе 1000 заявок – окончательные результаты моделирования.

При пункте меню «3» - наименование адресов, использованных при моделировании очереди-списка.

Данные об эффективности.

***Описание алгоритма:***

В реализации программы были созданы две структуры для хранения двух версий работы со стеком:

*Массив*

#define MAX\_QUEUE 1000

typedef struct st\_array\_queue  
{  
 int data[MAX\_QUEUE]; // массив очереди заявок  
 int size; // размер массива  
 int \*head; // указатель на начало очереди  
 int \*tail; // указатель на конец очереди  
} array\_queue\_t;

*Список*

typedef struct st\_node\_queue  
{  
 int data; // заявка из очереди  
 struct st\_node\_queue \*next; // указатель на следующую заявку   
} node\_queue\_t;  
  
typedef struct st\_list\_queue  
{  
 node\_queue\_t \*head; // указатель на начало очереди  
 int size; // размер очереди  
} list\_queue\_t;

Для моделирования были реализованы структуры:

*Структура данных о моделировании*

typedef struct st\_model  
{  
 int amount\_input; // количество вошедших заявок  
 int amount\_output; // количество вышедших заявок  
 int amount\_calls; // количество срабатываний аппарата  
  
 double model\_time; // общее время моделирования  
 double owntime; // время простоя АО  
 double consideration\_all; // общее время обработки заявок  
 double add\_all; // общее время добавления заявок  
  
 int current\_length\_queue; // текущая длина очереди   
 double average\_length\_queue; // средняя длина очереди   
 double average\_arrival\_time; // среднее время пребывания заявок  
} modeling\_t;

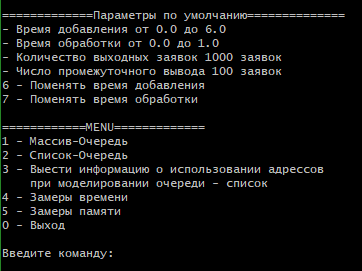
*Структура параметров моделирования*

typedef struct st\_params  
{  
 double add\_t1; // минимальное время добавления  
 double add\_t2; // максимальное время добавления  
 double consideration\_t1; // минимальное время обработки   
 double consideration\_t2; // максимальное время обработки  
  
 int amount\_iterations; // количество циклов проходов заявки   
 int amount\_application; // количество заявок для моделирования  
 int amount\_between\_print; // промежуточный вывод результатов  
} model\_params;

*Структура для хранения адресов*

typedef struct st\_address  
{  
 size\_t ptr; // значения адреса  
 \_Bool create; // значения для создания адреса  
 \_Bool free; // значения для освобождения адреса  
} address\_t;  
  
typedef struct st\_address\_library  
{  
 address\_t data[MAX\_QUEUE]; // массив указателей  
 int size; // размер массива  
} address\_library\_t;

Для взаимодействия с программой было создано консольное меню:



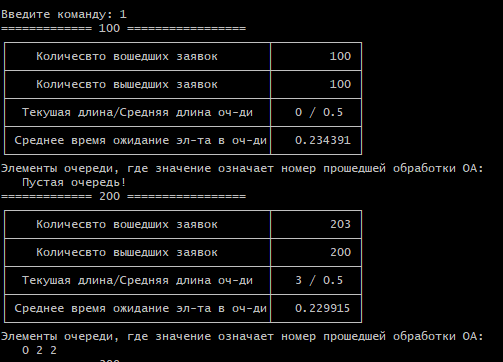
При вводе корректного значения выполняются определенные действия или операции, при некорректном выводится в консоль сообщение об ошибке и предоставляется ввести пункт еще раз. Так происходит пока не будет введен корректное значение меню.

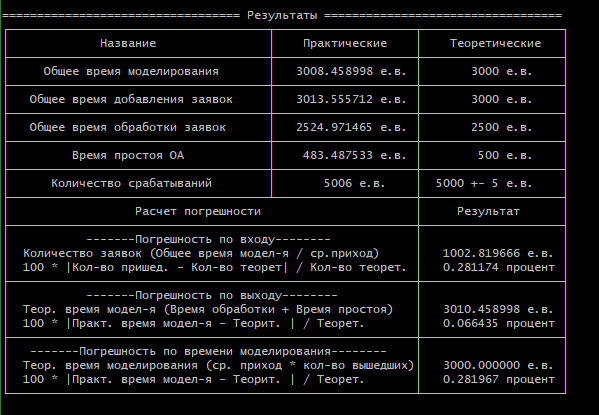
Для реализации программы, была использована статическая и динамическая типизация, поэтому были написаны функции выделения памяти и очищения памяти, чтобы при работе не возникали утечки памяти.

*Пункта 1 или 2 консольного меню:*

В этом пункте было реализовано моделирования очереди по заданию, представленную в виде массива или списка. Здесь моделируется 1000 вышедших заявок. Выводится информация каждые 100 вышедших заявок – о вошедших и вышедших заявок, текущей длине очереди, средней длине очереди и среднем времени прибывания заявки в очереди, а также саму очередь. После выхода 1000-й заявки выводится информация об общем времени моделирования, общее время добавление и обработки заявки, время простоя ОА(как практические данные, так и теоретические), и погрешности по входу, выходу и общему времени моделированию.

Все данные представлены в виде таблицы.

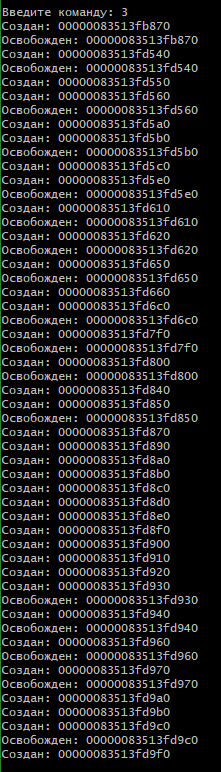




*Пункт 3 консольного меню*

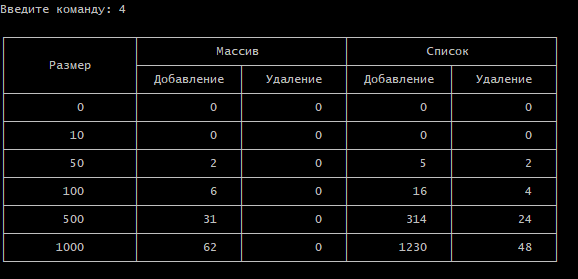
В этом пункте выводится использованные и освобожденные адреса заявок, при моделировании очереди, представленной в виде списка. Если моделирования не было – то выводится сообщение об том, что адреса не использованы.





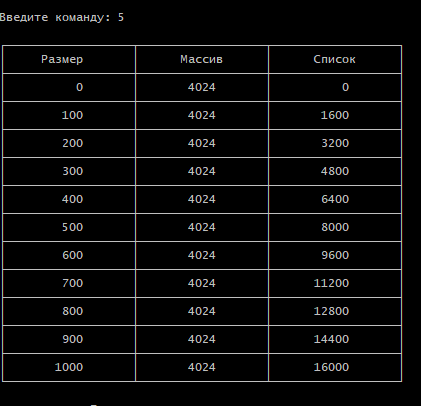
*Пункт 4 консольного меню*

В этом пункте реализована подпрограмма проверяющее время работы, добавления или удаления заявок в очередь или из очереди, представленную как в виде массива, так и в виде списка. Замеры проводятся от 0 – 1000 заявок – результат времени в микросекундах.



*Пункт 5 консольного меню*

В этом пункте реализована подпрограмма, которая считает размер использованные для хранения очереди в качестве массива или списка, результат выводится в байтах



*Пункт 5 и 6 консольного меню*

В этих пунктах реализованы подпрограммы, которые меняют, установленное по умолчанию в программе, время добавления и обработки заявок.

*Пункт 0 консольного меню* Выход из программы и очищение памяти матрицы и двух векторов-столбцов

***Алгоритм***

При моделирование очереди-массива был использована «кольцевая массив», то есть если указатель на начало очереди достиг конца массива, то при добавлении следующего элемента – он добавляется в начало массива, в случае когда нет переполнения массива.

При моделирование очереди-списка элемент добавляется в конец списка.

При моделирование очередь в начале пуста. По данным времени прихода(образуется случайным образом) заявки в очередь, она добавляется и начинается расчет общего времени моделирования. Затем генерируется время обслуживается и из очереди удаляется заявку. При добавлении следующей заявки, снова генерируется время прихода, и рассчитывается разница между временем прихода и обслуживания. Если время добавления больше времени обслуживания, то к времени простоя прибавляется разница между временами, в противном случае рассчитывается сколько еще заявок может добавится, время которых генерируется случайным образом.

Были реализованы функции работы с массивом:

// функция устанавливает значения по умолчанию для структуры массива-очереди

void default\_array\_queue(array\_queue\_t \*queue);  
// добавления заявки в массив-очередь  
int array\_push\_application(array\_queue\_t \*queue, int element);

// удаления заявки из массива-очереди  
int array\_pop\_application(array\_queue\_t \*queue);

// вывод массива-очереди  
void print\_array\_application(const array\_queue\_t \*queue);

Были реализованы функции работы с списком:

// функция устанавливает значения по умолчанию для структуры списка-очереди

void default\_list\_queue(list\_queue\_t \*queue);  
// добавления заявки в список-очередь  
int list\_push\_application(list\_queue\_t \*queue, int element);

// удаления заявки из списка-очереди  
int list\_pop\_application(list\_queue\_t \*queue);

// вывод списка-очереди  
void print\_list\_application(list\_queue\_t queue);

// очищения памяти выделенной при добавлении заявки в список-очереди  
void free\_list\_application(list\_queue\_t \*queue);

Функции для моделирования очереди

// функция моделирования для массива-очереди

void array\_queue\_modeling(const model\_params \*params);

// функция моделирования для списка-очереди

void list\_queue\_modeling(const model\_params \*params, address\_library\_t \*lib\_address);

Функции вывода данных моделирования

void print\_model\_data\_between(const modeling\_t \*modeling, const model\_params \*params);  
void print\_model\_data(const modeling\_t \*modeling, const model\_params \*params);

Функции для установления значений по умолчанию

void default\_modeling\_t(modeling\_t \*modeling);

void default\_model\_params(model\_params \*params);

Функции для установления новых значений времени

void change\_add\_time(model\_params \*params);  
void change\_cons\_time(model\_params \*params);

Функции для реализации меню

// функция ввода пункта меню

void input\_step\_menu(int \*step, int min, int max);

// функция вывода информации меню  
void print\_menu();

Функции для замеров времени и памяти

// функция нахождения хранения памяти очереди

size\_t get\_size\_array();  
size\_t get\_size\_list(size\_t n);  
// функции инфомации замеров времени и памяти  
void print\_main\_table\_size();  
void print\_table\_size(size\_t count, size\_t array, size\_t list);  
void print\_main\_table\_time();  
void print\_table\_time(size\_t count, int64\_t time\_push\_array, int64\_t time\_pop\_array, int64\_t time\_push\_list, int64\_t time\_pop\_list);  
// зфункции замеров времени и памяти очереди  
void analys\_size\_queue();  
void analys\_time\_queue();

Функции работы с массивом памяти

void default\_address\_library\_t(address\_library\_t \*lib);  
  
void add\_address\_create(address\_library\_t \*lib\_address, size\_t address);  
void add\_address\_free(address\_library\_t \*lib\_address, size\_t address);  
  
void print\_library\_address(const address\_library\_t \*lib\_addresses);

***Аварийные выходы из программы***

Выходов аварийных нет, в том случае, если программа не будет принудительна закрыта.

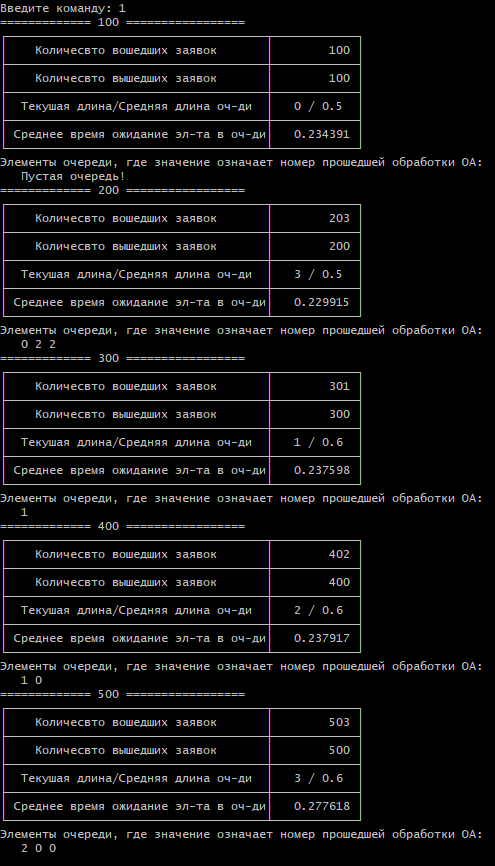
Выход осуществляется только пунктом меню «0»

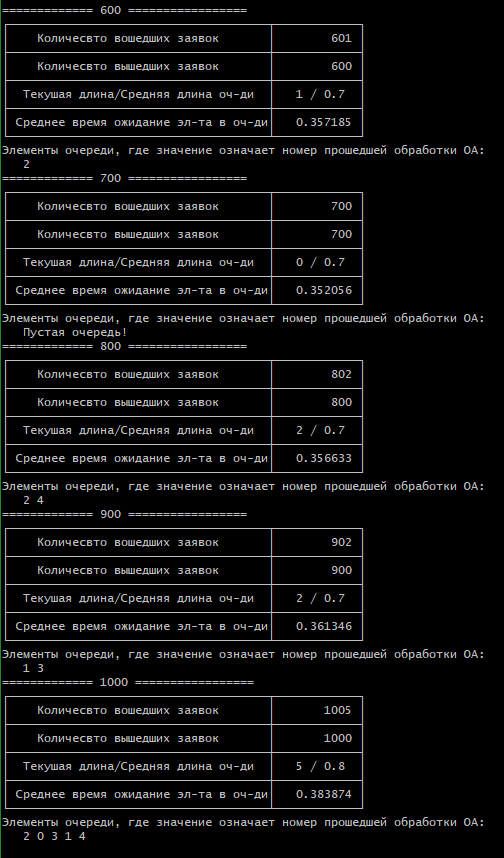
В ошибочных случаях выводится сообщение об ошибке и ввод повторяется, пока программа не получит корректные данные.

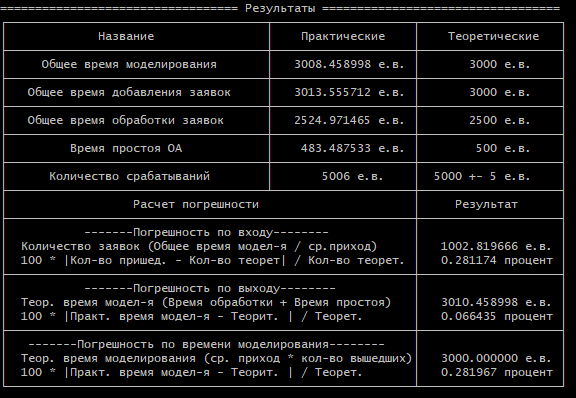
Если в результате операции – умножения или вводе матрицы и векторов, память не была выделена, то пользователь возвращается в главное меню и выводится сообщение об не выделении память в определенной операции.

***Анализ результатов моделирования***

При моделировании массива-очереди были полученны следующие данные:



******

******

Как видно из данных были получены проценты погрешностей:

* *Погрешность по входу* была рассчитана следующим образом:

**|amount\_p – amount\_t| / amount\_t \* 100 %,** где

**amount\_p** – количество заявок, практический

**amount\_t** – количество заявок, теоретический

**amoun\_t = mode\_time / ((t2 – t1) / 2) =** 3008 / 3 = 1002,

где **model\_time** – общее время моделирования

**t1, t2** – время добавления очереди заявки

**|**1005-1002**| /** 1002 \* 100 = 0.281174%

* *Погрешность по выходу* была рассчитана следующим образом:

**|time\_p – time\_t| / time \_t \* 100 %,** где

**time\_p** – практический время моделирования

**time\_t** – теоретическое время моделирования (рассчитанных по времени моделированию программы)

**time\_t = time\_work + downtime =** 2524 + 483 = 3010

где **time\_work** – время обработки по моделированию программы

**downtime** – время простоя по моделированию программы

**|**3008 – 3010**| /** 3010 \* 100 = 0.066435%

* *Погрешность по времени моделированию* была рассчитана следующим образом:

**|mode\_p – mode\_t| / mode\_t \* 100 %,** где

**mode\_p** – практический время моделирования

**mode\_t** – теоретическое время моделирования

**mode\_t = ((t2 – t1) / 2) \* 1000 =** 3000

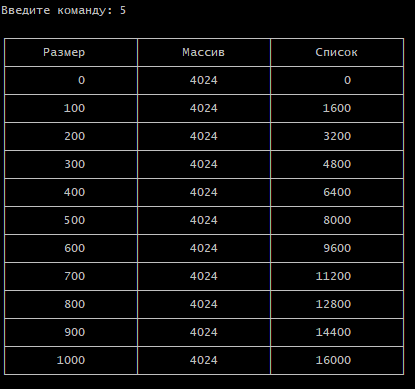
где **t1, t2** – время добавления очереди заявки

**|**3008 – 3000**| /** 3000 \* 100 = 0.281967 %

**Анализ эффективности методов**

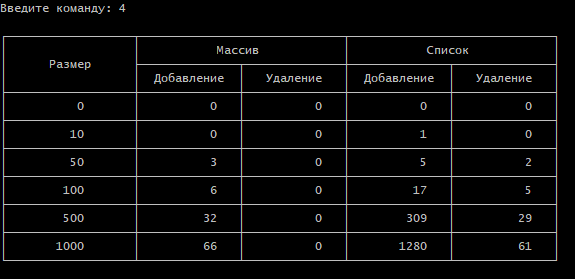
Для замеров эффективности методов представления очереди были взять размер от 0 – 1000 заявок:

* Замеры по памяти:



По результатам замеров можно сказать, что метод очереди в качестве списка, используется в том случае когда размер очереди не велик, достигаю примерно 250 элементов(~25%-30%) эффективнее использовать хранение в качестве массива.

* Замеры по времени:



По результатам замеров можно сказать, что время работы с массивом как при удалении, как при добавлении заявок быстрее, то есть эффективнее. Реализации алгоритма в массиве сложнее, чем в списке, так как в списке достаточно взять указатель на конец очереди и добавить заявку, а в массиве надо занять размер его, чтобы обратиться к концу очереди.

Эффективность по времени добавления в массив – (~60-80%), чем в списке, удаление эффективность показало, что по времени на (90-100%), так совершается лишь одна операции это уменьшение размера массив при использовании статическое выделение памяти под массив.

***Контрольные вопросы***

*1.Что такое FIFO и LIFO?*

Очередь - структура данных, для которой выполняется правило FIFO, то есть первым зашёл - первым вышел. Вход находится с одной стороны очереди, выход - с другой. На стек действует правило LIFO — последним пришел, первым вышел.

*2.Каким образом, и какой объем памяти выделяется под хранение очереди при различной ее реализации?*

При хранении 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(по мере необходимости) для каждого элемента отдельно.

*3.Каким образом освобождается память при удалении элемента из очереди при ее различной реализации?*

При хранении массивом память не освобождается, а просто меняется указатель на конец очереди и освобождается при завершении программы.

При хранении списком, при удалении каждый элемент очереди освобождается при каждом вызове процесса удаления.

*4.Что происходит с элементами очереди при ее просмотре?*

При классической реализации элементы очереди удаляется при ее просмотре.

*5.От чего зависит эффективность физической реализации очереди?*

Практически всегда эффективнее использовать очередь на основе массива, особенно статического. Он выигрывает по всем параметрам с большим отрывом (в 2-3 раза по памяти и в 7-12 раз по времени в случае статической реализации). Это происходит за счет того, что для статического массива память выделяется сразу же, а для списка на куче.

Однако память для хранения очереди на основе массива ограничена размером памяти, а память очереди на основе списка ограничена лишь размером доступной оперативной памяти, т.к. память выделяется в куче, поэтому, когда размер очереди достигает больших размеров, необходимо использовать очередь на основе списка, поэтому если размер очереди предполагается большим, нужно использовать список.

Массив занимает всю память может оказаться менее эффективнее, когда заполнен меньше 25%.

*6.Каковы достоинства и недостатки различных реализаций очереди в зависимости от выполняемых над ней операций?*

При кольцевой структуре сложнее реализовать алгоритмы включения и исключения элементов, однако статической памятью легче управлять, чем динамической.

Достоинство массива – скорость.

Достоинство списка – неограниченность по памяти.

Недостатки массива – ограниченность по память, фиксированная память.

Недостатки списка – возможность фрагментации, скорость выполнения операций.

*7.Что такое фрагментация памяти?*

Фрагментация памяти - явление, при котором информация записывается в использованных и освобожденных блоках памяти, таким образом она находится в разных местах доступной памяти. В результате появляются большое количество свободных участков памяти небольшого размера, расположенных разрозненно.

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

При запуске программы операционная система загружает программу в некоторую часть этой памяти. Память, используемая программой, разделена на несколько частей, каждая из которых выполняет определенную задачу. При запросе памяти операционная система, если может, выделяет кусок памяти и “отдает” ее для использования программой. Освобождение памяти - задача вызывающей стороны. Когда память освобождается, операционная система снова может распределять ее.

*9.Для чего нужен алгоритм «близнецов».*

Метод близнецов (*или buddy system*) является схемой выделения памяти, сочетающей в себе возможность слияния буферов и распределитель по степени числа 2. В основе метода лежит создание буферов малого размера путем деления пополам больших буферов и слияния смежных буферов по мере возможности. При разделении буфера на два каждая половина называется близнецом (buddy) второй.

Использование алгоритма «близнецов» для ускорение работы программы, то есть увеличение эффективности, при этом он уменьшает вероятность фрагментации памяти и улучшает поиск блок памяти для выделения.

*10.Какие дисциплины выделения памяти вы знаете?*

* Первый подходящий

Выбирается первая найденная подходящая память для выделения под данный элемент. При этом является более эффективным по поиску выделение памяти.

* Самый подходящий

Выбирается из всей возможной памяти самая «лучшая» подходящая память для выделения, но при этом вероятность нахождения такой памяти мала, поэтому бывают случай когда выделяется память, так что образуются пустые места в памяти, то есть выделение памяти, больше чем ожидалось.

*11.На что необходимо обратить внимание при тестировании программы?*

При тестировании программы необходимо обратить внимание на корректное выделение и освобождение памяти менеджером памяти. Также необходимо проверить правильность реализации операций для кольцевой очереди.

***Вывод***

В результате выполнения данной лабораторной работы, было использованы два алгоритма хранения и операции работы с очередью, при этом использование двух способа работы с очередью:

* Очередь, как массив показал большую эффективность по времени работы различных операции, то есть такая очередь использовать лучше когда нам необходим, чтобы программа работала более быстрее.
* Очередь, как список показал эффективность только в том случае, если количество элементов не больше 250-300 элементов, в таком случае список хранить лучше, то есть такая очередь использовать лучше, тогда, когда необходима эффективность по памяти хранения, но при этом время программы будет работать менее эффективнее, чем очередь-массив, но доступ к элементам будет достаточно просто получить, так список представлена в виде линейности и ограничивается памятью оперативной памятью оборудования.