|  |  |
| --- | --- |
| Gerb-BMSTU_01 | **Министерство науки и высшего образования 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)** |

**Типы и структуры данных.**

**Лабораторная работа №5.**

**«Обработка очередей»**

Студент **Леонов Владислав Вячеславович**

Группа **ИУ7-36Б**

Студент **\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_** Леонов В.В.

*подпись, дата фамилия, и.о.*

Преподаватель **\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_** Силантьева А.В.

*подпись, дата фамилия, и.о.*

*2020 г.*

Оглавление

[Условие и техническое задание 3](#_Toc58396317)

[Цель работы 3](#_Toc58396318)

[Задание 3](#_Toc58396319)

[Входные данные 4](#_Toc58396320)

[Выходные данные 4](#_Toc58396321)

[Обращение к программе 4](#_Toc58396322)

[Функции программы 4](#_Toc58396323)

[Аварийные ситуации 5](#_Toc58396324)

[Реализация 6](#_Toc58396325)

[Структуры данных 6](#_Toc58396326)

[Алгоритм 9](#_Toc58396327)

[Тестирование 10](#_Toc58396328)

[Аварийные ситуации 10](#_Toc58396329)

[Штатное поведение 11](#_Toc58396330)

[Оценка эффективности использования очереди на основе массива и списка 13](#_Toc58396331)

[Итоги моделирования (в мс) 13](#_Toc58396332)

[Объем занимаемой памяти (в байтах) 13](#_Toc58396333)

[Контрольные вопросы 14](#_Toc58396334)

[1.Что такое очередь? 14](#_Toc58396335)

[2. Каким образом, и какой объем памяти выделяется под хранение очереди при различной ее реализации? 14](#_Toc58396336)

[3. Каким образом освобождается память при удалении элемента из очереди при ее различной реализации? 14](#_Toc58396337)

[4. Что происходит с элементами очереди при ее просмотре? 14](#_Toc58396338)

[5. Каким образом эффективнее реализовывать очередь. От чего это зависит? 14](#_Toc58396339)

[6. Каковы достоинства и недостатки различных реализаций очереди в зависимости от выполняемых над ней операций? 15](#_Toc58396340)

[7. Что такое фрагментация памяти? 15](#_Toc58396341)

[8.На что необходимо обратить внимание при тестировании программы? 15](#_Toc58396342)

[9. Каким образом физически выделяется и освобождается память при динамических запросах? 15](#_Toc58396343)

[Вывод о проделанной работе 16](#_Toc58396344)

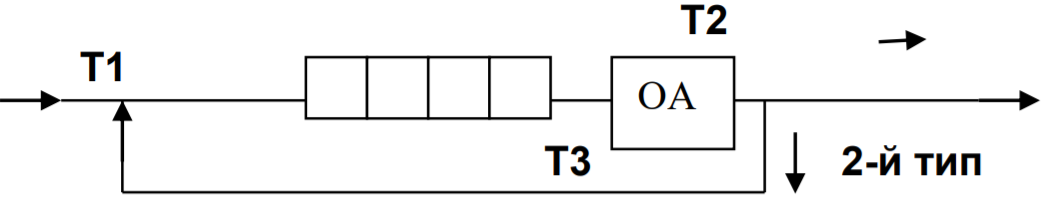
# Условие и техническое задание

## 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.

## Задание

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



Заявки 1-го типа поступают в "хвост" очереди по случайному закону с интервалом времени Т1, равномерно распределенным от 0 до 5 единиц времени (е.в.). В ОА они поступают из "головы" очереди по одной и обслуживаются также равновероятно за время Т2 от 0 до 4 е.в., после чего покидают систему.

Единственная заявка 2-го типа постоянно обращается в системе, обслуживаясь в ОА равновероятно за время Т3 от 0 до 4 е.в. и возвращаясь в очередь не далее 4-й позиции от "головы". В начале процесса заявка 2-го типа входит в ОА, оставляя пустую очередь. (Все времена – вещественного типа)

Смоделировать процесс обслуживания первых 1000 заявок 1-го типа. Выдавать после обслуживания каждых 100 заявок 1-го типа информацию о текущей и средней длине очереди, количестве вошедших и вышедших заявок и о среднем времени пребывания заявок в очереди. В конце процесса выдать общее время моделирования, время простоя аппарата, количество вошедших в систему и вышедших из нее заявок первого типа и количество обращений заявок второго типа. По требованию пользователя выдать на экран адреса элементов очереди при удалении и добавлении элементов. Проследить, возникает ли при этом фрагментация памяти.

## Входные данные

1. **Временные конфигурации** времени обработки и поступления заявок в очередь и обслуживающий аппарат.
2. **Опция меню** (целое число от 0 до 9).

## Выходные данные

1. **Таблица состояния очереди** для каждой 100 обработанных заявок 1 типа.
2. **Список адресов памяти элементов очереди**.
3. Результаты моделирования
   1. Ожидаемое время моделирования.
   2. Фактическое время моделирования.
   3. Время простоя обслуживающего аппарата.
   4. Количество вошедших заявок первого типа.
   5. Количество вышедших заявок первого типа.
   6. Количество обращений заявок второго типа.
   7. Время работы очереди на массиве в мс.
   8. Время работы очереди на списке в мс.
   9. Среднее время обработки заявки первого типа.
   10. Среднее время обработки заявки второго типа.

## Обращение к программе

Обращение к программе **app.exe** осуществляется через запуск из терминала.

## Функции программы

1. Выполнить моделирование.
2. Изменить параметры моделирования.
3. Выполнить печать адресов памяти очереди.
4. Об авторе.
5. Завершить работу.

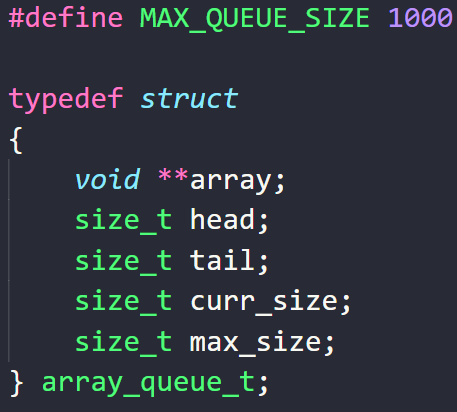
## Аварийные ситуации

1. Некорректная опция меню (буква или значение, несоответствующее ни одному пункту из меню.
2. Некорректный ввод параметров моделирования очереди (буква или отрицательное число).
3. Ошибка выделения памяти операционной системой.

# Реализация

## Структуры данных

Структура **array\_queue\_t**



Для реализации очереди на основе массива используется структура **array\_queue\_t**.

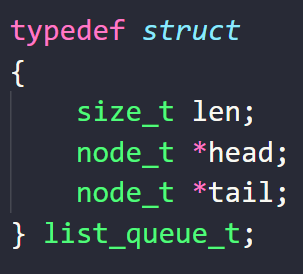
Константа **MAX\_QUEUE\_SIZE** определяет максимальный размер очереди на основе массива.

Поле **array** представляет собой массив без типовых указателей и служит для хранения данных очереди.

Поля **head** и **tail** представляют собой целые беззнаковые числа и служат для хранения индексов начала и конца элементов очереди.

Поля **curr\_size** и **max\_size** представляют собой целые беззнаковые числа и служат для хранения текущего и максимального размера очереди соответственно.

Структура **list\_queue\_t**

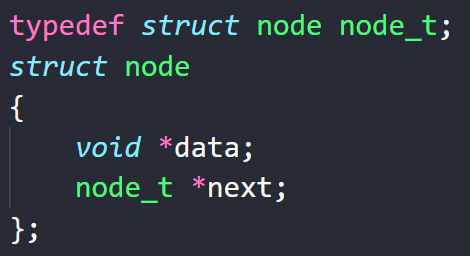


Для реализации очереди на основе списка используется структура **list\_queue\_t**.

Поле **len** представляет собой целое беззнаковое число и служит для хранения текущей длины очереди.

Поля **head** и **tail** представляют собой указатели узлы списка, являющимися «головой» и «хвостом» очереди соответственно.

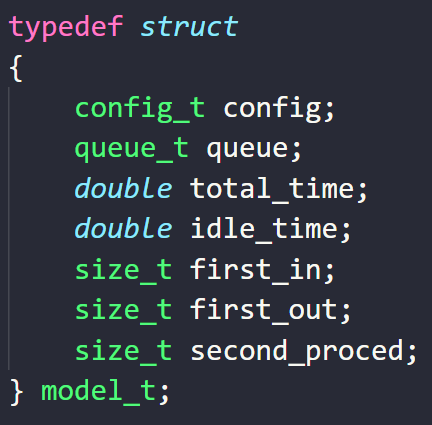
Структура **node\_t**



Для описания узла списка служит используется структура **node\_t.**

Поле **data** представляет собой указатель на данные, хранящиеся в узле списка, поле **next** представляет собой указатель на следующий узел списка.

Структура **model\_t**



Для описания модели очереди используется структура **model\_t**.

Поле **queue** служит для описания очереди и представляет собой объединение представлений очереди в виде массива и в виде списка.

Поле **total\_time** представляет собой вещественное число и служит для описания общего времени работы системы.

Поле **idle\_time** представляет собой вещественное число и служит для описания времени простоя обслуживающего аппарата системы.

Поля **first\_in** и **first\_out** представляют собой целые беззнаковые числа и служат для описания количества вошедших и вышедших из очереди заявок первого типа.

Поле **second\_proc** представляет собой целое беззнаковое число и служит для описания обращений заявки второго типа.

## Алгоритм

1. Пользователь вводит номер команды из меню.
2. Пока пользователь не введет 0 (выход из программы), ему будет предложено выполнять действия из меню программы.

Для реализации добавления в очередь на основе массива и списка заявок первого и второго типов используются функции **array\_queue\_f\_insert, array\_queue\_s\_insert** и **list\_queue\_f\_insert, list\_queue\_s\_insert** соответственно.

Для удаления элемента из очереди на основе массива и списка используются функции **array\_queue\_del\_elem** и **list\_queue\_del\_elem.**

Для реализации создания и удаления очереди на основе массива и списка используются функции **array\_queue\_create, array\_queue\_del** и **list\_queue\_create, list\_queue\_del** соответственно.

При реализации моделирования выполняется последовательная генерация событий очереди с последующим выполнением ближайшего к данному моменту времени, после чего происходит смещение состояния по временной шкале (функция **model\_launch).**

# Тестирование

## Аварийные ситуации

|  |  |  |
| --- | --- | --- |
| № | Описание теста | Поведение программы |
| 1 | Некорректная опция из меню  (буква) |  |
| 2 | Некорректная опция из меню  (цифра) |  |
| 3 | Некорректное значение параметров моделирования |  |
| 4 | Некорректное значение параметров моделирования  (большая граница меньше меньшей) |  |

## Штатное поведение

|  |  |  |
| --- | --- | --- |
| № | Описание теста | Демонстрация работы программы |
| 1 | Выполнение моделирования |  |
| 2 | Изменение параметров моделирования |  |
| 3 | Печать адресов памяти |  |
| 4 | Печать информации об авторе |  |
| 5 | Завершение работы |  |

# Оценка эффективности использования очереди на основе массива и списка

## Итоги моделирования (в мс)

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Время прихода первой заявки | Время обслуж.  первой заявки | Время обслуж. второй заявки | Время списка | Время массива | Преимущество использования массива вместо списка (в %) |
| 1…5 | 1…2 | 1…3 | 0.617 | 0.150 | ~410% |
| 1…5 | 1…5 | 0.5...1.3 | 0.277 | 0.097 | ~307% |
| 1…5 | 10…15 | 3.15…5.25 | 1.576 | 0.489 | ~315% |

Исходя из приведенных выше данных можно сделать вывод, что использование массива для построения очереди более эффективно, получаем выигрыш по времени порядка 3-4 раз, что довольно существенно.

## Объем занимаемой памяти (в байтах)

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Время прихода первой заявки | Время обслуж.  первой заявки | Время обслуж. второй заявки | Объем памяти списка в пике | Объем памяти  массива | Преимущество использования массива вместо списка (в %) |
| 1…5 | 1…2 | 1…3 | 104 | 8000 | ~8000% |
| 1…5 | 1…5 | 0.5...1.3 | 1384 | 8000 | ~587% |
| 1…5 | 10…15 | 3.15…5.25 | 57336 | 8000 | ~ - 615% |

Исходя из приведенных выше данных можно сделать вывод, что использование списка для построения очереди более эффективно статического массива, если время обслуживания меньше, чем время прихода новый заявки (большинство времени массив почти пустой). В обратной ситуации, когда обслуживание происходит долго, получаем выигрыш по времени у массива, однако при данной реализации в случае полной заполненности происходит перезапись элементов очереди и некоторые заявки теряются.

# Контрольные вопросы

## 1.Что такое очередь?

Очередь - структура данных, для которой выполняется правило FIFO, то есть первым зашёл - первым вышел. Вход находится с одной стороны очереди, выход - с другой.

## 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 для каждого элемента отдельно.

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

При построении очереди на основе списка удаляются память под конкретный узел. В случае же использования массива смещается лишь указатель на конец очереди.

## 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 массив. В этом случае получаем выигрыш и по памяти, и по времени работы ввиду отсутствия необходимости хранения дополнительных указателей и обращения к памяти. 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 реализации очереди на основе списка нужно больше времени и возникает фрагментация в памяти компьютера. При реализации же на основе массива получаем выигрыш, как по времени, так и памяти (если известен размер), однако не решена проблема переполнения очереди.

## 7. Что такое фрагментация памяти?

- Неоднородное расположение участков памяти. (Адреса памяти не последовательные, в отличие от выделения памяти для массива, когда данные располагаются линейно друг за другом).

## 8.На что необходимо обратить внимание при тестировании программы?

При тестировании программы нужно обязательно обратить внимание на работу с ресурсами (отсутствие лишних действий, выполнение оптимизаций алгоритмов, выделение и освобождение динамической памяти). Следует также обеспечить удобный и понятный интерфейс, что позволит избежать дополнительных ошибок от пользователя.

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

При запросе памяти, ОС находит подходящий блок памяти и записывает его в «таблицу» занятой памяти. При освобождении, ОС удаляет этот блок памяти из «таблицы» занятой пользователями памяти.

# Вывод о проделанной работе

В результате лабораторной работы была изучена структура данных – очередь, методы ее реализации на основе известных ранее структур (список и массив), а также особенности обработки подобной структуры данных. Были закреплены навыки работы с динамической памятью.

Согласно полученным экспериментальным путем данных построение очереди на основе массива дает существенный выигрыш по времени обработки информации (порядка 3-4 раз) нежели чем на списке ввиду того, что происходит меньше обращений к памяти, операция которой является очень дорогостоящей по времени. Но в то же время присутствует проблема переполнения очереди при построении на основе массива, поэтому использование списка позволяет решить данную проблему.

Хочется отметить, что при выделении памяти под список данные не находятся последовательно в памяти (присутствует фрагментация).