МГТУ им. Н.Э. Баумана

**Дисциплина Типы и Структуры данных.**

**Лабораторный практикум №5**

**по теме:** «Обработка очередей»

Работу выполнил:

студент группы ИУ7-35Б

Прянишников Александр

**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.

# Условие задачи. Вариант 3

# 

# Требования к задаче

**Входные данные**

Программа ничего не получает от пользователя на вход в режимах моделирования очереди. В ручном режиме работы с очередью единственные входные данные: элементы очереди

**Вывод данных**

К выходным данным относятся: сама очереди, адреса, занимаемые в стеке, адреса, освободившиеся после удаления, время исполнения программы (касается только вывода слов в обратном порядке), прогнозируемое время моделирования, реальное время моделирования очереди, средняя длина очереди, текущая длина очереди, среднее время ожидания, время простоя аппарата, количество обработанных заявок 1-го и 2-го типа, а также среднее время выполнения операций добавления и удаления элементов из массива.

Также в программу был добавлен интерфейс, упрощающий работу для пользователей: есть пояснительные надписи, перед вводом данных написано, как правильно вводить данные.

**Описание задачи, реализуемой программой**

Программа предоставляет пользователю интерфейс для работы с очередью. Она либо моделирует очередь на основе кольцевого массива/связного списка, либо работает с очередью на базе связного списка в ручном режиме.

**Способ обращения к программе.**

Программу можно запустить через терминал.

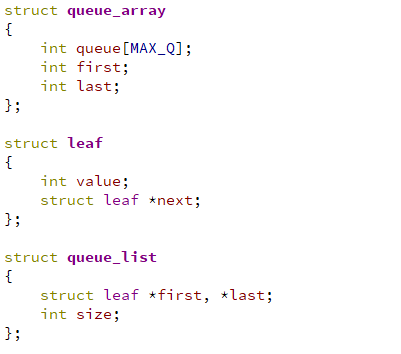
**Описание возможных аварийных ситуаций и ошибок пользователя**

* Выбор отсутствующего пункта меню (от 0 до 4)  
  *В этом случае программа сообщит пользователю о невозможности совершения операции и попросит ввести новое значение пункта меню.*
* Ввод букв при вводе числа  
  *В этом случае программа сообщит пользователю о невозможности совершения операции и попросит ввести новое значение пункта меню.*
* Пустой ввод  
  *В этом случае программа сообщит пользователю о невозможности совершения операции и попросит ввести новое значение пункта меню.*
* Слишком длинная введённая строка  
  *В этом случае программа сообщит пользователю о невозможности совершения операции и попросит ввести всё заново.*
* Добавление нового элемента в переполненную очередь.  
  *В этом случае программа сообщит пользователю о невозможности совершения операции в ручном режиме. В режиме моделирования ничего не произойдёт, заявка просто не будет рассматриваться.*
* Удаление элемента из пустой очереди  
  *В этом случае программа сообщит пользователю, что удаление невозможно. В режиме моделирования эта ситуация невозможна, так как перед вызовом проверяется, есть ли элементы в очереди.*

# Описание внутренних структур данных.

Для решения этой задачи я решил использовать две структуры: одна описывает стек через кольцевой массив, другая реализует стек через связный список.

Вот так описаны структуры:



# Описание алгоритма

Программа работает для двух разных структур по разным алгоритмам, выполняя одно и то же действие:

**1 – Моделирование очереди в виде массива**Программа запускает моделирование, используя для этого кольцевой массив. Моделирование происходит до тех пор, пока очередь не покинут 1000 заявок первого типа.

**2 – Моделирование очереди в виде связного списка**.  
Программа запускает моделирование, используя для этого кольцевой массив. Моделирование происходит до тех пор, пока очередь не покинут 1000 заявок первого типа.

**3 –**  **Сравнить время работы алгоритмов.**Программа создаёт две очереди: одна в виде массива, другая – в виде связного списка. Добавляя одинаковое число элементов, она смотрит, сколько времени в среднем тратится на операции добавления и удаления из массива, а затем выводит таблицу результатов.

**4 – Работа с очередью в ручном режиме**

Программа создаёт временную очередь на основе связного списка и позволяет пользователю работать с ней. Можно добавить новый элемент в очередь, вытащить элемент из очереди, а также вывести на экран текущее состояние очереди. При этом также показывается, какие адреса были освобождены и какие используются в очереди.

**0 – Выход.**

Теперь стоит поговорить про алгоритмы подсчёта времени. Программа генерирует два времени – время обработки ОА и время прихода следующей заявки в очередь. Затем подсчитывается минимальное из них, и программа переходит к ближайшему событию, добавляя при этом время в общий счётчик.

Моделирование происходит до тех пор, пока 1000 заявок 1-го типа не пройдут ОА. При этом в очереди постоянно крутится одна заявка 2-й очереди. Программа подсчитывает статистику в отдельных переменных, а после моделирования выводит всё на экран.

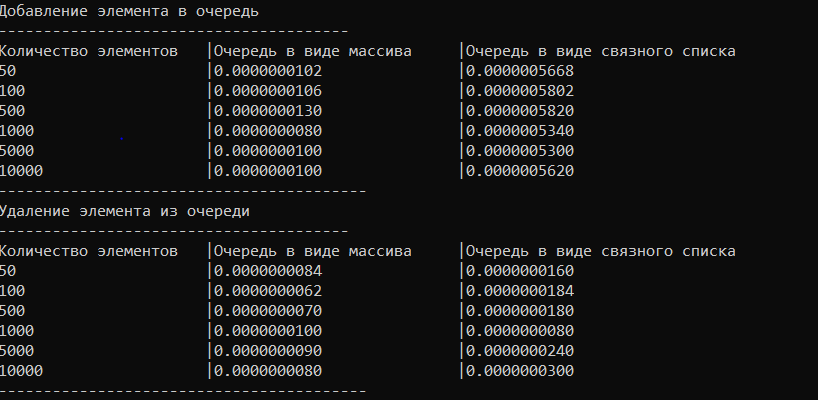
# Тесты

Тестирование происходило по такому принципу: каждый пункт тестировался отдельно, после чего контролировалась работа программы целиком. Поэтому здесь указаны только негативные тесты, которые в целом проверяли работоспособность модулей программы.

|  |  |
| --- | --- |
| **123345fd** | **Буквы при задании числа** |
|  | **Пустой ввод** |
| **–1** | **Выбор неправильного пункта меню** |
| **Добавить новый элемент нельзя!** | **Переполнение очереди** |
| **Взять элемент из очереди нельзя!** | **Удаление из пустой очереди** |

# Оценка эффективности

Сравнение эффективности по времени доступно в самой программе.



Как видно, массивная реализация намного эффективнее по времени, чем связный список. При этом от N работа функций практически не зависит.

Построим аналогичную таблицу для памяти, размер очереди – 10000 элементов.

|  |  |  |
| --- | --- | --- |
|  | Массив (Память) | Связный список (Память) |
| N = 50 | 40012 | 620 |
| N = 100 | 40012 | 1220 |
| N = 500 | 40012 | 6200 |
| N = 1000 | 40012 | 12200 |
| N = 5000 | 40012 | 62000 |
| N = 10000 | 40012 | 122000 |

Как видно, связный список по памяти при N до 2–3 тысяч, после чего массив становится эффективнее. Единственное преимущество связного списка – сверху его размер ограничен только размерами оперативной памяти.

Также построим таблицу для проверки того, что ожидаемое время работы совпадает с реальным

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| t\_OA\_0 | t\_OA\_1 | t\_1\_0 | t\_1\_1 | Expected | Real | Accuracy |
| 0 | 6 | 0 | 5 | 3632 | 3746 | 3.148% |
| 0 | 4 | 0 | 5 | 2500 | 2534 | 1.367% |
| 2 | 4 | 0 | 5 | 3630 | 3755 | 3.446% |
| 4 | 4 | 2 | 5 | 4882 | 5030 | 2.493% |
| 0 | 2 | 0 | 2 | 1250 | 1247 | 0.707% |
| 6 | 12 | 3 | 14 | 11140 | 11241 | 0.892% |

Как видно, ожидаемое время действительно совпадает.

Также стоит отметить, что в моей программе происходит фрагментация памяти: это можно отследить в ручном режиме: освободившиеся адреса не используются вновь.

# Выводы по проделанной работе

Сегодня я познакомился с двумя способами хранения очереди: в виде статического кольцевого массива, и в виде связного списка. Первый работает гораздо быстрее, при этом требует больше памяти, но при больших N массив нивелирует и это отставание. Связный список работает намного медленнее, но зато есть выигрыш по памяти для N при не слишком большом числе элементов, и количество возможных элементов зависит только от оперативной памяти.

# Ответы на контрольные вопросы

Что такое очередь?

Очередь – это последовательный список переменной длины, включение элементов в который идет с одной стороны (с «хвоста»), а исключение – с другой стороны (с «головы»). Принцип работы очереди: первым пришел – первым вышел. First In – First Out (FIFO).

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

При хранении статическим кольцевым массивом количество памяти = N \* размер одного элемента очереди. Память выделяется при компиляции.

При хранении списком: N \* (размер одного элемента очереди + указатель на следующий элемент). Память выделяется в куче для каждого элемента отдельно, в ходе исполнения программы.

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

При хранении кольцевым массивом память не освобождается, только указатель указывает на другой элемент. При хранении связным списком память освобождается сразу при удалении.

Что происходит с элементами очереди при его просмотре?

При просмотре очереди с элементами указатель сдвигается, и элемент уничтожается.

Каким образом эффективнее реализовывать очередь? От чего это зависит?

Очередь значительно эффективнее работает в массивном варианте, при этом его лучше использовать, если мы знаем, сколько точно элементов будет в стеке. Связный стек при небольших размерах будет эффективнее по памяти, и при этом если стек планируется слишком большой, то для статического массива может не хватить памяти, и тогда связный список – единственный способ реализации стека.

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

При реализации через массив программа работает значительно быстрее, но максимальное число элементов задаётся ещё при компиляции, во время исполнения программы изменить уже ничего нельзя.

Связный список гораздо медленнее, требует больше памяти при больших N, но при этом N ограничено только операционной системой.

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.

На что необходимо обратить внимание при тестировании программы?

На возможное переполнение очереди, на корректное освобождение памяти.

Каким образом физически выделяется и освобождается память при динамических запросах?

Операционная система ищет подходящий блок памяти и записывает его в «таблицу» занятой памяти. При освобождении система удаляет этот блок памяти из «таблицы» занятой пользователем памяти.