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)

ФАКУЛЬТЕТ «ИНФОРМАТИКА И СИСТЕМЫ УПРАВЛЕНИЯ»

КАФЕДРА «ПРОГРАММНОЕ ОБЕСПЕЧЕНИЕ ЭВМ И ИНФОРМАЦИОННЫЕ ТЕХНОЛОГИИ»

ОТЧЕТ ПО ЛАБОРАТОРНОЙ РАБОТЕ №5 ПО ДИСЦИПЛИНЕ:
ТИПЫ И СТРУКТУРЫ ДАННЫХ

ОБРАБОТКА ОЧЕРЕДЕЙ

Студент **Попов Ю.А.**

Группа **ИУ7-32Б**

Вариант **7**

Название предприятия **НУК ИУ МГТУ им. Н. Э. Баумана**

Студент _____ **Попов Ю.А.**

Преподаватель _____ **Попов Ю.А.**

2024

Оглавление

Описание условия задачи

Отработка навыков работы с типом данных «очередь», представленным в виде одномерного статического массива (представление динамическим массивом можно добавить по желанию) 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.

Описание технического задания

Входные данные:

При работе с программой на экран выводится меню:

0 - Выход

1 - Тестирование очереди на статическом массиве

2 - Тестирование очереди на списке

3 - Симуляция работы очереди

Пользователь вводит целое число-команду от 0 до 3. Программа выполняет операции, в зависимости от выбранной опции. Если пользователь выбрал первую или вторую операцию, то пользователь попадает в подпрограмму для тестирования, где должен ввести номер операции (от 1 до 3) и элемент один символ для вставки в случае, если выбрана операция “добавления элемента в очередь”. Чтобы корректно завершить программу, необходимо воспользоваться соответствующей командой.

Выходные данные:

Если пользователь выбрал операцию “Симуляция”, то выводом будут - результаты симуляции, а именно состоянии очереди раз в 100 заявок,

информации о времени симуляции, реальное время работы операций над очередью.

При тестировании очереди, в зависимости от операции на экран выводится либо вся очередь, либо удаленный элемент.

Действие программы:

Программа работает до тех пор, пока пользователь не введет число 0. Программное обеспечение позволяет ввести команды от 0 до 4, в случае если команда выходит за этот диапазон, или не подходит по типу данных, выводится соответствующее сообщение, и программа завершает работу. Разработанная программа позволяет протестировать реализацию очереди на массиве и на связном списке, а также запустить симуляцию работы обслуживающего аппарата.

Пользователю доступны следующие операции при тестировании очереди:

1) Вывести на экран всю очередь;
2) добавить элемент в очередь;
3) удалить элемент из очереди;

Обращение к программе:

Программа запускается через терминал, в командной строке, с помощью команды ./app.exe. После запуска. пользователю будет доступно меню программы.

1) Выход
2) Тестирование очереди на массиве
3) Тестирование очереди на списке
4) Запуск симуляции
5) Изменить параметры симуляции

Аварийные ситуации:

1. ERR_OPERATION – Неверный выбор операции.
2. ERR_QUEUE_OVERFLOW – Переполнение очереди, происходит при попытке добавления элемента в “заполненную” очередь.
3. ERR_QUEUE_UNDERFLOW – Происходит при попытке удаления из пустой очереди.
4. ERR_MEMORY_ALLOCATION - Ошибка при выделении памяти
5. ERR_DATA – Ошибка данных при добавлении элемента в очередь.
6. ERR_SYMBOL_INPUT - Неверный символ при добавлении элемента в очередь.
7. ERR_SIMULATION_PARAMS - Неверные параметры симуляции

Во всех нештатных ситуациях, программа уведомляет о месте, где произошла ошибка.

Описание структуры данных

```
#define MAX_QUEUE_SIZE 1001
```

Максимальный размер очереди - 100000 элементов.

Структура для хранения данных о запросе. Необходима при симуляции

processing_count - количество циклов, совершенных заявкой

arrival_time - время, когда заявка добавлена в очередь

```
typedef struct __request_t
{
    size_t processing_count;
    float arrival_time;
} request_t;
```

листинг 1. Структура request_t

Структура для хранения данных в очереди

element - элемент, для тестирования очереди

request_data - структурная переменная, необходимая для информации симуляции.

start - Указатель на начало очереди

end - Указатель на конец очереди

```
typedef struct __data_t
{
    char element;
    request_t request_data;
} data_t;
```

листинг 2. Структура data_t

Структура для очереди на основе массива

UP LIMIT - Максимальное количество элементов

count - Количество элементов

data - Массив элементов очереди

start - Указатель на начало очереди

end - Указатель на конец очереди

```
typedef struct __arr_queue_t
{
    size_t UP LIMIT; // Максимальное количество элементов
    size_t count; // Количество элементов
    data_t data[MAX_QUEUE_SIZE];
    data_t *start; // Указатель на начало
    data_t *end;   // Указатель на конец
} arr_queue_t;
```

листинг 3. Структура arr_queue_t

Структура для хранения информации об элементе списка.

next - указатель на следующий элемент списка

data - указатель на данные элемента

```
typedef struct __node_t
{
    struct __node_t *next;
    void *data;
```

```
} node_t;
```

листинг 2. Структура node_t

Структура для очереди на основе связного списка.

head - указатель на начало

end - указатель на конец

UP_LIMIT - максимальное количество элементов

count - текущее количество элементов

```
typedef struct __list_queue_t
{
    node_t *head; // Указатель на начало
    node_t *end;  // Указатель на конец

    size_t UP_LIMIT; // Максимальное количество элементов
    size_t count;    // Текущее количество элементов
} list_queue_t;
```

листинг 2. Структура list_stack_t

Описание алгоритма

1. После запуска программы выводит приглашение к вводу и меню.
2. Программа принимает номер операции, и запускает соответствующие действия в соответствии с номер команды
3. Добавление в очередь на основе списка происходит путем создания нового элемента и присваиванием указателя на него в поле next.
4. Для того чтобы удалить элемент очереди, нужно изменить указатель на начало очереди. и освободить элемент
5. Если пользователь вводит 0, то программа завершается.

Описание основных функций

```
void arr_queue_init(arr_queue_t *queue);
```

Инициализация статического очереди

Входные параметры: указатель на очередь

Выходные параметры: проинициализирована очередь

```
int arr_queue_push(arr_queue_t *queue, data_t *element);
```

функция добавляет элемент в очередь на массиве

Входные параметры: указатель на очередь, элемент для добавления

Выходные параметры: очередь, в которую был добавлен переданный элемент

Возвращаемый результат: код возврата.

```
int arr_queue_pop(arr_queue_t *queue, data_t *element);
```

Функция удаляет последний элемент из очереди

Входные параметры: указатель на очередь, указатель код возврата

Выходные параметры: удаленный элемент, код возврата

Возвращаемый результат: код возврата

```
void list_queue_init(list_queue_t *queue);
```

Инициализация очереди на списке

Входные параметры: указатель на очередь

Выходные параметры: проинициализирована очередь

```
int list_queue_push(list_queue_t *queue, const void *src_data, size_t
src_size);
```

функция добавляет элемент в очередь на списке

Входные параметры: указатель на очередь, элемент для добавления, размер памяти, занимаемой элементом

Выходные параметры: очередь, в которую был добавлен переданный элемент

Возвращаемый результат: код возврата.

```
int list_queue_pop(list_queue_t *queue, void *dst_data, size_t
src_size);
```

функция удаляет элемент из очереди на списке

Входные параметры: указатель на очередь, удаленный элемент, размер памяти, занимаемой элементом

Выходные параметры: очередь, из которой был удален элемент

Возвращаемый результат: код возврата.

```
void list_queue_free(list_queue_t *queue);
```

Функция освобождает память, выделенную под очередь

Входные параметры: указатель очередь

```
int input_menu_operation(operations_t *operation);
```

Функция обеспечивает получение операции от пользователя

Входные параметры: указатель операцию

Выходные параметры: операция, введенная пользователем

Возвращаемый результат: код возврата

Набор тестов

№	Описание теста	Ввод	Вывод
1	Некорректный ввод операции	abc	Ошибка при выборе операции.
2	Некорректный ввод операции. Выход за диапазон	10000	Ошибка при выборе операции.
3	Некорректный ввод операции. Отрицательное число	-100	Ошибка при выборе операции
6	Пустая строка		Ошибка при вводе строки
7	Удаление из пустой очереди		Ошибка, очередь пустая
8	Переполнение очереди	`#define MAX_QUEUE_SIZE 2` 1 2 3	Переполнение очереди
9	Вывод пустой очереди		Ошибка, очередь пустая
10	Неверное изменения параметров	-1 10	Ошибка, неверные

	симуляции (отрицательное время)		параметры симуляции
11	Неверное изменения параметров симуляции (символы)	0 a	Ошибка, неверные параметры симуляции
12	Неверное изменения параметров симуляции (верхняя граница интервала меньше нижней)	2 0	Ошибка, неверные параметры симуляции

Оценка эффективности

Объем занимаемой памяти (байты)

Объем, занимаемой памяти очередью в зависимости от их размера

Размер	**Очередь на статическом массиве (байт)**	**Очередь на связном списке (байт)**	**Отношение занимаемой памяти очереди на списке к очереди на массиве**
100	48032	1632	29.43
500	48032	8032	5.98
1000	48032	16032	2.99
1500	48032	24032	1.99
2000	48032	32032	1.49

Таблица 1. Оценка занимаемой памяти, относительно размера.

Временные замеры (нс)

Временные замеры осуществлялись с помощью библиотеки time.h. Замеры скорости выполнения проводились для симуляции, потому что в ней очередь используется активнее всего.

Результаты симуляции обслуживающего аппарата с очередью на списке:

№	Кол-во заявок 1-го типа	Время моделирования (условные е.в.)	Время моделирования (мкс)
1	1007	233.51	3035.27
2	1007	409.75	3036.24
3	1020	244.82	3046.82
4	1078	248.28	3154.53
5	1013	548.66	3026.74
6	1034	238.03	3088.42
7	1030	234.79	3044.69
8	1004	235.57	3038.76
9	1000	673.98	3063.01
10	1026	569.81	3065.4
Среднее	1021.9	363.72	3059.99

Таблица 2. Результаты симуляции обслуживающего аппарата с очередью на списке

Результаты симуляции обслуживающего аппарата с очередью на массиве:

№	Кол-во заявок 1-го типа	Время моделирования (условные е.в.)	Время моделирования (мкс)
1	1078	242.22	1244.69
2	1000	234.33	1265.4

3	1020	233.52	1246.82
4	1007	673.98	1185.27
5	1011	234.79	1226.74
6	1029	569.81	1248.42
7	1035	548.55	1305.33
8	1004	235.57	1238.76
9	1026	248.22	1190.01
10	1005	409.75	1262.1
Среднее	1021.53	363.13	1241.35

Таблица 2. Результаты симуляции обслуживающего аппарата с очередью на массиве

В результате единичных измерений времени выполнения функции добавление и удаление, было выяснено, что метод с использованием статического массива работает в 3 раза быстрее. Несмотря на то что обе операции имеют сложность O(N), операции для очереди на списке выполняются дольше из-за обращения к функции для работы с выделением и освобождением памяти. По таблице 2 видно, что использование очереди на массиве выгоднее. Однако для моего ограничения на количество элементов, очередь на основе списка эффективнее по объему занимаемой памяти.

Тестирование задания

Теоретический расчет времени

моделирования очереди = max(среднее время прихода заявки 1-го типа, среднее время обработки заявки 1-го типа) * количество.

В моем варианте время прихода больше времени обработки. Среднее время прихода новой заявки 1-го типа равно $(T1_{low} + T1_{up})/2 = \frac{0+6}{2} =$ 3 единиц условного времени. Так как мат ожидание времени прихода заявки 3 единицы, а время обработки [0, 1] е.в, то очередь будет накапливаться - стремиться к определенному числу элементов, в моем варианте это около 27 элементов.

Время моделирования будет равно:

1000 * (ср. Время прихода первой заявки) = 3000 е.в.

В среднем число вошедших заявок: 1020

Число вышедших заявок: 1000

В данном случае число срабатываний аппарата: N * k = 1000 * (5 + 1) = 6000

Так как все заявки должны пройти по 5 циклов обратки со временем [0, 1], а среднее время прихода 3 е.в, то к концу работы симуляции может появиться очередь. В моем случае она содержит около 40 элементов

```
Общее время моделирования: 3036.37
Время простоя ОА: 15.28
Кол-во вошедших заявок: 1024
Кол-во вышедших заявок: 1000
Кол-во срабатываний ОА: 6072

Вычисление погрешностей.
Вычисленное кол-во заявок: 1012.12, имеем: 1024, погрешность: 1.17%
Аппарат работал: 6072 е.в., простаивал: 15.28 е.в.
Значит время моделирования должно быть: 3000.00 е.в., имеем 3036.37
е.в., погрешность: 1.21%
```

Таким образом, можно увидеть, что результаты программы близки к вычисленным.

Ответы на контрольные вопросы

1. Что такое FIFO и LIFO?

LIFO (Last In First Out) “первый пришел, последний ушел” - лежит в основе стека, можно привести аналогию со стопкой бумаги.
FILO (First In First Out) “первый пришел, последний ушел” - лежит в основе очереди.

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

Размер данных: 24 байта

При реализации очереди статическим массивом выделяется 48 + (N * 24) где N - это максимальное количество элементов. При реализации очереди листом один элемент очереди занимает 16 байт, а структура под очередь занимает 80 байт. В итоге очередь занимает 80 + (n) * 16 байт, где n - количество элементов. Память выделяется под каждый элемент.

Примечание: В очереди на основе массива хранится указатель на данные, а не сами данные, поэтому она занимает объем меньше.

3. Каким образом освобождается память при удалении элемента очереди при различной реализации стека?

При реализации стека на основе статического массива, работа с памятью возлагается на систему. При реализации на основе списка сначала освобождается память, выделенная под данные, а потом освобождается элемент списка.

```
node_t *next = queue→head;
free(next→data);
```

```
free(next);
```

4. Что происходит с элементами очереди при ее просмотре?

Для просмотра очереди, в функцию передается ее копия. Далее цикл проходимся по копии списка, которая в конечном итоге будет “испорчена”. Исходный стек останется нетронутым.

5. От чего зависит эффективность физической реализации очереди?

Выяснилось, что эффективнее по времени реализовывать очередь массивом. А по памяти выгоднее использовать связный список, но это зависит от самой структуры элемента в случае памяти.

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

В случае массива недостатком является фиксированный размер очереди.

В случае односвязного списка недостатками являются затраты по скорости и фрагментация.

7. Что такое фрагментация памяти?

Фрагментация — возникновение участков памяти, которые не могут быть использованы. Фрагментация может быть внутренней — при выделении памяти блоками остается не задействованная часть, может быть внешней — свободный блок, слишком малый для удовлетворения запроса

8. Для чего нужен алгоритм «близнецов»?

Идея этого алгоритма состоит в том, что организуются списки свободных блоков отдельно для каждого размера 2k
, 0 <= k <= m. Вся область памяти кучи состоит из 2m элементов, которые, можно считать, имеют адреса с 0
по 2m – 1. Первоначально свободным является весь блок из 2m элементов. Далее, когда требуется блок из 2k
элементов, а свободных блоков такого размера нет, расщепляется на две равные части блок большего размера; в результате появится блок размера 2k
(т.е. все блоки имеют длину, кратную 2). Когда
один блок расщепляется на два (каждый из которых равен половине первоначального), эти два блока называются близнецами. Позднее, когда оба близнеца освобождаются, они опять объединяются в один блок.

9. Какие дисциплины выделения памяти вы знаете?

Две основные дисциплины сводятся к принципам "самый подходящий" и "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.

10. На что необходимо обратить внимание при тестировании программы?

На соответствие ожидаемого и фактического результата, на правильность работы с памятью

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

В оперативную память поступает запрос, содержащий необходимый размер выделяемой памяти. Выше нижней границы свободной кучи осуществляется поиск блока памяти подходящего размера. В случае если такой найден, в вызываемую функцию возвращается указатель на эту область и внутри кучи она помечается как занятая. Если же найдена область, большая необходимого размера, то блок делится на две части, указатель на одну возвращается в вызываемую функцию и помечается как занятый, указатель на другую остается в списке свободных областей. В случае если области памяти необходимого размера не было найдено, в функцию возвращается NULL. При освобождении памяти происходит обратный процесс. Указатель на освобождаемую область поступает в оперативную память, если это возможно объединяется с соседними свободными блоками, и помечается свободными.

ВЫВОД

Очередь на массиве работает быстрее, и имеет недостаток в виде фиксированного размера, занимаемой памяти. Если сделать очередь на динамическом массиве, то этот недостаток можно нивелировать.