Отчет по 5 лабораторной работе

По дисциплине «Типы и структуры данных»

Подготовил Жабин Дмитрий

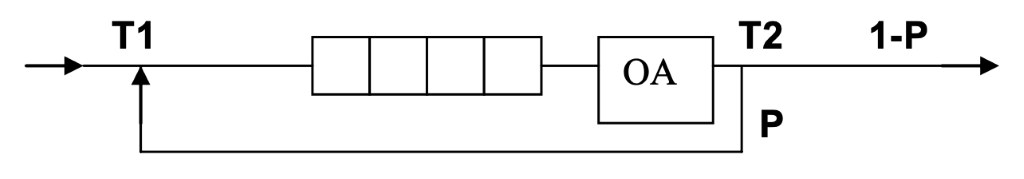
Группа ИУ7-34Б

Вариант 5

**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.



Заявки поступают в "хвост" очереди по случайному закону с интервалом времени Т1, равномерно распределенным от 0 до 6 единиц времени (е.в.). В OA они поступают из "головы" очереди по одной и обслуживаются также равновероятно за время Т2 от 0 до 1 е.в., Каждая заявка после ОА с вероятностью Р=0.8 вновь поступает в "хвост" очереди, совершая новый цикл обслуживания, а с вероятностью 1-Р покидает систему. (Все времена – вещественного типа). В начале процесса в системе заявок нет.

Смоделировать процесс обслуживания до ухода из системы первых 1000 заявок. Выдавать после обслуживания каждых 100 заявок информацию о текущей и средней длине очереди. В конце процесса выдать общее время моделирования и количество вошедших в систему и вышедших из нее заявок, среднее время пребывания заявки в очереди, время простоя аппарата, количество срабатываний ОА. Обеспечить по требованию пользователя выдачу на экран адресов элементов очереди при удалении и добавлении элементов. Проследить, возникает ли при этом фрагментация памяти.

**Внешняя спецификация программы.**

**Входные данные:** время поступления заявок в очередь, время обработки заявок – вещественного типа.

**Выходные данные:** после обслуживания каждых 100 заявок выводится информация о текущей и средней длине очереди, а в конце процесса – общее время моделирования, количество вошедших в систему и вышедших из нее заявок, время простоя аппарата, количество срабатываний аппарата, процент погрешности работы (по входу и выходу), при реализации списком – адреса добавленных и удаленных элементов очереди. Информация об эффективности двух реализаций очереди - массивом и списком.

**Задача программы:**

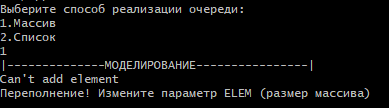
Произвести работу с очередью с помощью двух реализаций - списком и массивом. Сравнить затраты по времени и памяти в этих случаях.

**Способ обращения к программе:** Запуск программы в консоли (./app.exe).

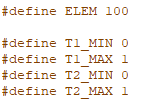
**Аварийные ситуации**

1) Переполнение при реализации массивом

Программа завершает работу с соответствующим сообщением.



Возможно при следующих параметрах:



**Структуры данных.**

1) При реализации списком:

typedef struct qnode //узел списка

{

struct qnode \*next; //указатель на следующий элемент списка

float time; //время обработки

} node\_t;

typedef struct queue\_list //очередь

{

struct qnode \*pin; //указатель на конец списка

struct qnode \*pout; //указатель на начало списка

int num\_of\_elems; //текущее количество элементов в очереди

int summ\_of\_elems; // общее количество элементов в очереди

} queue\_t\_list;

2) При реализации массивом:

typedef struct queue //очередь

{

float\* data\_store; //массив для хранения элементов очереди

int front; //индекс последнего элемента очереди

int num\_of\_elems; //количество элементов в очереди в данный момент

int summ\_of\_elems; //общее количество элементов в очереди

} queue\_t;

**Функции**

void refresh\_automat(queue\_t \*q1, automat \*a); - обновление текущего состояния ОА при работе с очередью на массиве

void refresh\_automat\_list(queue\_t\_list \*q1, automat\_list \*a); - обновление текущего состояния ОА при работе с очередью на списке

float get\_time(queue\_t \*q1, automat \*a); - итерация цикла работы ОА с очередью на массиве

float get\_time\_list(queue\_t\_list \*q1, automat\_list \*a); - итерация цикла работы ОА с очередью на списке

int add\_elem(queue\_t \*queue, float item); - добавить элемент в очередь на массиве

float delete\_elem(queue\_t \*queue); - удалить элемент из очереди на массиве

void queue\_free(queue\_t\* queue); - освободить память из-под очереди на массиве

node\_t \*get\_new\_node\_list(float time); - создать новый узел списка

int add\_elem\_list(queue\_t\_list \*queue, float time); - добавить элемент в очередь на списке

float delete\_elem\_list(queue\_t\_list \*queue); - удалить элемент из очереди на списке

void free\_queue\_list(queue\_t\_list \*queue); - освободить память из-под очереди на списке

**Алгоритм**

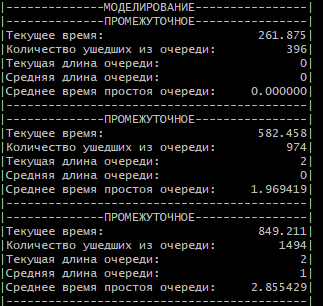
Очередь реализована на списке и сдвижном массиве.

На каждой итерации цикла в хвост очереди добавляются новые элементы (заявки) со значением времени поступления (генерируется случайное вещественное число в интервале от 0 до 6). При этом забираются элементы из головы очереди, для каждого генерируется время обработки (случайное вещественное число в интервале от 0 до 1). Параллельно ведутся два таймера, к которым прибавляются время поступления и обработки заявок соответственно, что позволяет следить за тем, свободен ли аппарат в данный момент (в этом случае из очереди берется следующая заявка на обработку, если же в данный момент очередь пуста - увеличивается таймер простоя аппарата). Для каждой обработанной заявки генерируется случайное число (вероятность), в зависимости от которого заявка покидает систему (увеличивается соответствующий счетчик) или же снова добавляется в очередь. Процесс продолжается, пока счетчик заявок, покинувших систему, не достигнет 1000.

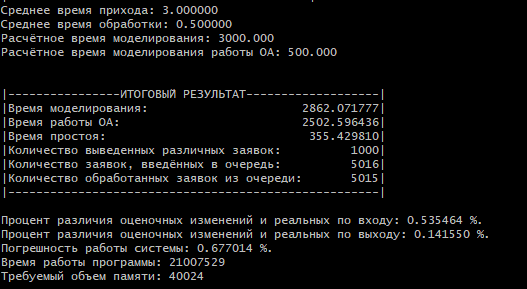
**Пример работы программы**

1) Реализация на массиве

Данные об очереди при обработке первых трех сотен заявок

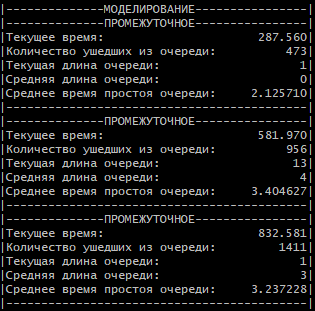


Итоговый результат

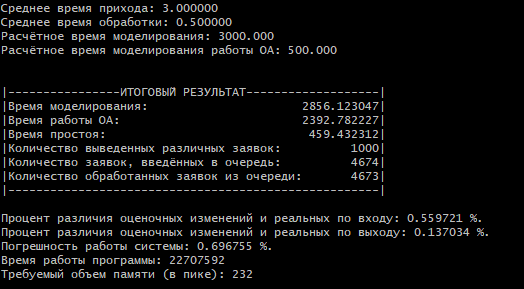


2) Реализация на списке

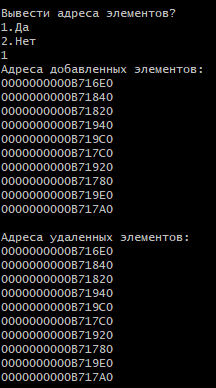
Данные об очереди при обработке первых трех сотен заявок



Итоговый результат



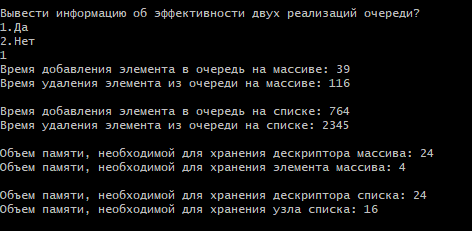
Адреса элементов



**Оценка эффективности**

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Размер очереди | Время, тактов процессора | | | | Память, байт | |
| Добавление | | Удаление | |
| Массив | Список | Массив | Список | Массив | Список |
| 10 | 34 | 546 | 108 | 2315 | 824 | 184 |
| 50 | 824 |
| 100 | 1624 |

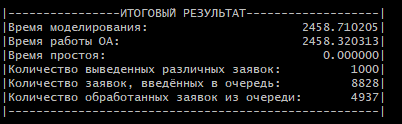
\*под массив резервируется память на 200 элементов



Объем памяти, необходимой для хранения дескриптора массива и списка совпадает и равен 24 байтам, для хранения элемента массива необходимо 4 байта, для узла списка - 16 байт, поэтому заполнение массива на 25% и больше обеспечивает выигрыш по памяти в сравнении с таким же количеством элементов на списке.

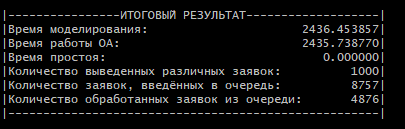
В данной задаче такое заполнение массива можно получить, задав интервал от 0 до 1 и для времени поступления заявок, и для времени их обработки. В этом случае по памяти будет выигрывать массив, а по времени - список.

1) Реализация на массиве





2) Реализация на списке





**Вывод**

Погрешность в целом примерно равна в обеих реализациях, колеблется от 0.2% до 1.5%. При работе программы возникает фрагментация памяти (выделяются непоследовательные адреса памяти).

Для реализации данной задачи в общем случае под массив выделяется память на 10000 элементов (40024 байта с учетом дескриптора), но при заданных значениях времен поступления и обработки заявок (время обработки значительно меньше) средняя длина очереди очень мала (в пике не превышает 20, что соответствует 344 байтам с учетом дескриптора), из-за чего список очень сильно выигрывает по памяти (объем памяти для его хранения определяется текущим количеством элементов). Также при малой длине очереди по времени выигрывает массив, так как при реализации очереди на списке необходимо освобождать и выделять память каждый раз, когда добавляется или удаляется элемент. Если изменить значения времен, то средняя длина очереди будет достигать больших значений и выигрывать по памяти будет массив. По времени же будет сильно выигрывать список за счет того, что при большой длине очереди много времени уходит на сдвиг элементов в массиве. Также в общем случае к недостаткам очереди-списка можно отнести возникновение фрагментации памяти.

**Контрольные вопросы**

**Что такое очередь?**

Очередь – это последовательный список переменной длины, включение в который идет с одной стороны (с хвоста), а исключение – с другой стороны (с головы).

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

Массив: выделяется последовательная область памяти константного размера. Выделение памяти происходит в начале работы программы. При необходимости память перевыделяется.

Список: память под элемент очереди выделяется непосредственно в процессе его добавления. Объем памяти, который занимает очередь, изменяется в процессе выполнения программы и напрямую зависит от количества элементов в очереди в каждый момент времени.

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

Массив: освобождение памяти происходит в конце работы программы (или при удалении очереди). При удалении элемента из очереди происходит только смещение указателя.

Список: при удалении элемента из очереди происходит освобождение памяти, которая была выделена под этот элемент.

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

При просмотре очереди хвостовой элемент из нее удаляется.

**Каким образом эффективнее реализовывать очередь. От чего это зависит?**

Если необходимо избежать фрагментации памяти, то лучше использовать очередь на массиве. Однако такой способ лучше использовать в том случае, если заранее известно количество элементов в очереди. Иначе лучше реализовывать список.

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

При реализации очереди массивом не возникает фрагментация памяти, однако может произойти переполнение очереди, а также затрачивается дополнительное время на сдвиг элементов (можно исключить сдвиг, используя кольцевой массив). При реализации очереди списком затрачивается большее количество времени при добавлении нового элемента, для хранения указателей требуется дополнительная память.

**Что такое фрагментация памяти?**

При последовательных запросах на выделение и освобождение памяти под элемент не всегда выделяется память, которая была только что освобождена.

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

При тестировании программы необходимо обратить внимание на переполнение очереди, фрагментацию памяти при реализации очереди списком.

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

Программа дает запрос ОС на выделение блока 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, однако указатель на этот блок может остаться в программе. Попытка считать данные из этого блока моет привести к непредвиденным последствиям, поскольку они могут быть изменены.