Лабораторная работа №4 по дисциплине «Типы и структуры данных»

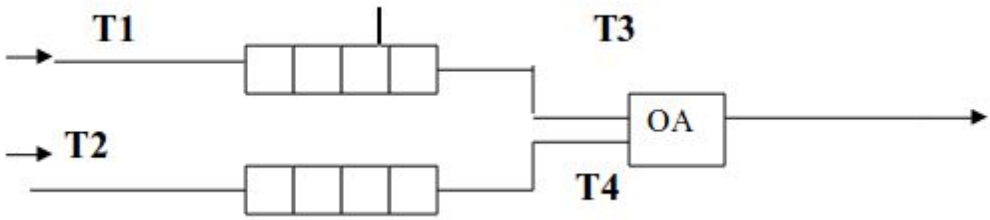
Обработка очередей

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 (Вариант 6):

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



Заявки 1-го и 2-го типов поступают в "хвосты" своих очередей по случайному закону с интервалами времени **T1 и T2**, равномерно распределенными от **1 до 5** и от **0 до 3** единиц времени (е.в.) соответственно. В ОА они поступают из "головы" очереди по одной и обслуживаются также равновероятно за времена **T3 и T4**, распределенные от **0 до 4** е.в. и от **0 до 1** е.в. соответственно, после чего покидают систему. (Все времена – **вещественного** типа) В начале процесса в системе заявок нет.

Заявка 2-го типа может войти в ОА, если в системе нет заявок 1-го типа. Если в момент обслуживания заявки 2-го типа в пустую очередь входит заявка 1-го типа, то она немедленно поступает на обслуживание; обработка заявки 2-го типа прерывается и она возвращается в "хвост" своей очереди (система с абсолютным приоритетом и повторным обслуживанием).

Смоделировать процесс обслуживания первых 1000 заявок **1-го типа**, выдавая после обслуживания каждых 100 заявок 1-го типа информацию о текущей и средней длине каждой очереди, а в конце процесса - общее время моделирования и количестве вошедших в систему и вышедших из нее заявок обоих типов, среднем времени пребывания заявок в очереди, количестве «выброшенных» заявок второго типа. Обеспечить по требованию пользователя выдачу на экран адресов элементов очереди при удалении и добавлении элементов. Проследить, возникает ли при этом фрагментация памяти.

Исходные данные:

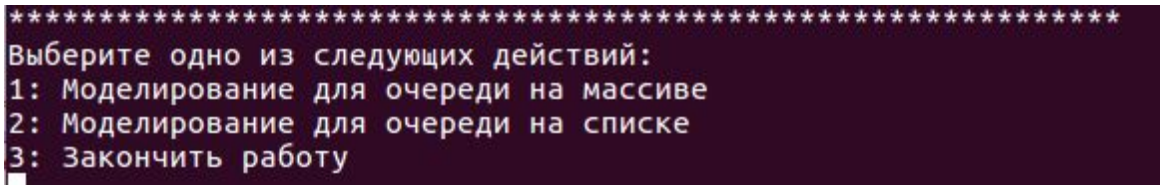
Программа позволяет выбрать структуру данных для хранения очереди.

Выходные данные:

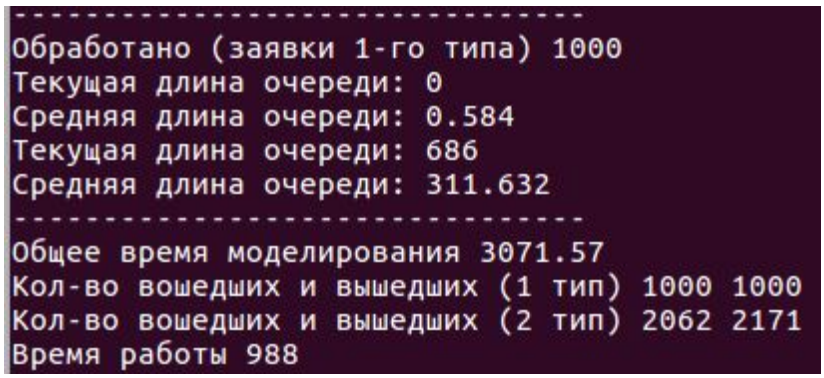
После обслуживания каждых 100 заявок 1-ого типа выводится информация о текущей и средней длине каждой очереди, а в конце процесса - общее время моделирования и количество вошедших в систему и вышедших из нее заявок обоих типов.

Интерфейс программы:

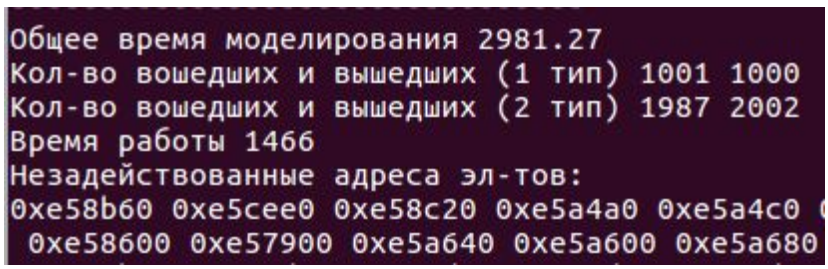
1) Главное меню



2) Моделирование для очереди на массиве



3)Моделирование для очереди на списке



Возможные ошибки пользователя:

1) При превышении допустимого размера очереди пользователь получит следующую ошибку "Переполнение очереди! "

Внутренние структура данных:

Очередь на списке	Очередь на массиве

typedef struct Node {
  int value;
  Node* next;
} list;
 	
list *create_node(int x);
list *add_to_list(list *tmp, list *head);
void free_all(list *head, list* memory, int m);
void print_adr(list *head);
 	
class Q_list {
private:
  list *memory[MAX_MEMORY];
  int m;
  list *head;
  list *tail;
  info_queue info;
public:
  Q_list();
  ~Q_list();
  void show();
  void show_adr();
  int PopFront();
  void PushBack(int x);
  bool Empty_Q();
  bool is_full();
};
	class Q_array {
	private:
	  int * arr = NULL;
	  int back;
	  int front;
	  int max_size;
	  info_queue info;
	
	public:
	  Q_array();
	  Q_array(const Q_array &);
	  ~Q_array();
	  int PopFront();
	  void PushBack(int x);
	  bool Empty_Q();
	  Q_array& operator= (const Q_array &obj);
	  void show();
	  void show_adr() {}
	  bool is_full();
	};

Время моделирования

Среднее время создания заявок:

1-го типа: 3 е.в;

2-го типа: 1.5 е.в.

Среднее время обработки заявок

1-го типа: 2 е.в.

2-го типа: 0.5 е.в.

На создание 1000 заявок 1 типа будет потрачено 3*1000 =3000 е.в.
За это время будет создано около 3000е.в /1.5 =2000 заявок 2 типа.
1000 заявок первого типа будет обработано за 1000*2=2000 е.в
2000 заявок второго типа будет обработано за 2000*0,5=1000 е.в

Время создания заявок обоих типов больше чем время их обработки=> Обе очереди не должны сильно расти со временем=> на время моделирования влияет только Время создания заявок 1-го типа.
Среднее время моделирования 3000 е.в.

Сравнение эффективности:

На массиве

#	Число заявок 1-го типа	Число заявок 2-го типа	Время моделирования (е.в.)	Время работы, мкс
1	1000	1923	3022.43	1086
2	1000	1850	2984.15	1211
3	1000	2102	3038.35	992
5	1000	1910	2986.93	1118
5	1000	2056	2997.44	790
6	1000	2083	3029.1	538
7	1000	2045	3010.6	767
8	1000	2037	2960.66	805
9	1000	2236	3088.07	961
10	1000	2204	3061.89	913
среднее	1000	2044	3017,96	918.1

На списке

#	Число заявок 1-го типа	Число заявок 2-го типа	Время моделирования (е.в.)	Время работы, мкс
1	1000	1834	2966.53	1384
2	1000	1952	2977.51	1667
3	1000	1902	3032.58	1333
5	1000	2129	3052	1652
5	1000	1916	2940.84	1581
6	1000	1963	3015.96	1008
7	1000	1904	2969.56	1221
8	1000	1995	3006.12	1521
9	1000	2150	3078.88	1468
10	1000	2047	2992.64	1498
среднее	1000	1979.2	3003.262	1433

Погрешность времени моделирования: 0,6%
Время выполнения программы с использованием очереди на основе списка в среднем на 150% больше реализации с помощью массива.
При работе программы возникает фрагментация. Одной из причин возникновения которой является существование двух очередей.

Сравнение реализаций:

При представлении очереди в виде списка используется большее кол-во памяти для хранения указателей. К недостаткам очереди-списка можно отнести возникновение фрагментации памяти. Очередь-список позволяет воспользоваться памятью, ограниченной лишь объемом оперативной памяти компьютера.
Эффективность реализации работы со списком зависит от кол-ва операций добавления и извлечения элемента из очереди. Недостатком очереди-списка является то, что при совершении этих операций он обязательно выделяет или

освобождает память. Очередь-массив производит операции выделения/освобождения намного реже.

Теоретическая часть:

1. Что такое очередь?

Очередь – это последовательный список переменной длины, включение элементов в который идет с одной стороны , а исключение – с другой стороны.

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

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.

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

3. 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 эл-та из очереди происходит только смещение указателя.

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

4. Что происходит с элементами очереди при ее просмотре?

При просмотре очереди, хвостовой элемент из нее удаляется

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

Если необходимо избежать фрагментации памяти, то следует использовать очередь на массиве. Также выбор способа реализации зависит от того, известно ли заранее количество эл- тов в очереди и насколько оно велико.

6. В каком случае лучше реализовать очередь посредством указателей, а в каком – массивом?

Очередь лучше реализовывать с помощью указателей, если новые элементы в среднем появляются реже, чем происходит полное очищение очереди – в общем случае фрагментация не возникает.

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

При реализации очереди массивом не возникает фрагментации памяти, однако может произойти переполнение очереди, а также затрачивается дополнительное время на сдвиг элементов. При реализации очереди списка затрачивается большее кол-во времени при добавлении нового элемента

8. Что такое фрагментация памяти?

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

9. На что необходимо обратить внимание при тестировании программы?

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

10. 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 может привести к непредвиденным последствиям, поскольку они могут быть уже изменены.