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

Вариант 2

Студент **Ширяев А.А.**

Группа **ИУ7-33Б**

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

Студент _____ **Ширяев А.А.**

Преподаватель _____**Силантьева. А.В**

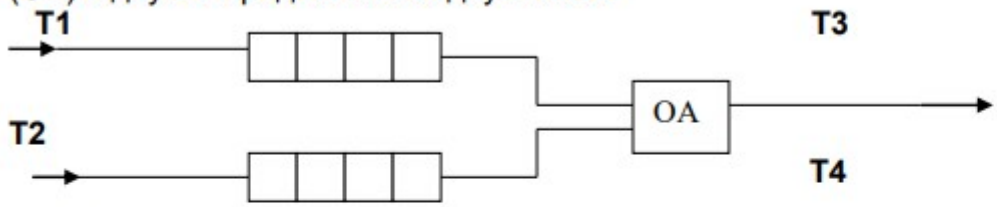
2024

Цель работы — 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.

Условие задачи

Вариант 2

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 ждет первого освобождения ОА и далее поступает на обслуживание (система с **относительным** приоритетом).

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

ПРИМЕЧАНИЕ! В отчёте будут использоваться обозначения для действий:

<init> - Инициализация времён прихода/обработки

<static_service> - Работа СМО с использованием статической очереди

<list_service> - Работа СМО с использованием очереди-списка

<as_output> - Вывод используемых для очереди-списка адресов

<stat>- Вывод статистики

<exit> - Выход из программы

(ПРИМЕЧАНИЕ! Статическая очередь - КОЛЬЦЕВАЯ)

Описание техзадачи

Описание исходных данных

Данные на входе: Меню. Код действия. Далее для каждого действия:

<init> - время прихода/обработки заявки каждой очереди (неотрицательные вещественные числа)

<static_service> - 2 статической очереди, время прихода/обработки заявки каждой очереди

<list_service> - 2 очереди-списка, время прихода/обработки заявки каждой очереди, динамический массив адресов

<as_output> - Динамический массив адресов

<stat>- 2 статической очереди, 2 очереди-списка, время прихода/обработки заявки каждой очереди (2-х очередей)

<exit> - Все очереди в программе, динамический массив адресов

Данные на выходе:

<init> - время прихода/обработки заявки каждой очереди

<static_service> - Статистика по работе СМО с использованием статических очередей

<list_service> - Статистика по работе СМО с использованием очередей-списков

<as_output> - Элементы динамического массива адресов, наличие фрагментации

<stat>- Статистика по реальному времени по работе СМО со статическими очередями и очередями-списками, статистика по реальному времени по выполнению базовых действий со статическими очередями и очередями-списками

<exit> - -

Описание задачи, реализуемой программой

Программа реализует ряд действий:

<init> - Инициализация времён прихода/обработки

<static_service> - Работа СМО с использованием статической очереди

<list_service> - Работа СМО с использованием очереди-списка

<as_output> - Вывод используемых для очереди-списка адресов

<stat>- Вывод статистики

<exit> - Выход из программы

Способ обращения к программе

Для обращения к программе запускается файл app.exe.

Описание возможных аварийных ситуаций и ошибок пользователя

Программа может не вывести результат, а вывести сообщение об ошибке. Данная ситуация может произойти при условии:

Ошибки пользователя

1. Неверный код действия

Аварийные ситуации

1. Программа не смогла выделить необходимую память для работы (с массивом адресов)
2. Программа не смогла выделить необходимую память для работы (для добавления элемента в список-очередь)

Описание внутренних СД

Информация о статической очереди

Информация о статической очереди представляет собой структуру на языке Си, состоящую из:

```
#define MAX_QUEUE_SIZE 2049

typedef struct
{
    double times[MAX_QUEUE_SIZE];
    size_t size;

    double *head;
    double *tail;
} static_queue_t;
```

times - Массив, хранящий элементы статической очереди. Размер stack – текстовая замена MAX_QUEUE_SIZE

size — Текущий размер очереди

*head – указатель на «голову» очереди

*tail – “хвост” очереди (указатель на следующий участок памяти, куда можно добавить элемент)

Информация об очереди-списке

Информация об очереди-списке представляет собой структуру на языке Си, состоящую из:

```
typedef struct
{
    queue_data_t *head;
    queue_data_t *tail;
} list_queue_t;
```

*head – Указатель на «голову» очереди

*tail – Указатель на «хвост» очереди (Указатель на последний элемент очереди-списка)

Информация о «голове» и «хвосте» очереди-списка

```
struct queue_data
{
    double time;

    queue_data_t *next;
};
```

Информация о «голове» и «хвосте» очереди-списка представляет собой структуру на языке Си, состоящую из:

time – Время обработки заявки в обслуживающем аппарате

*next – Указатель на следующий элемент

Информация о массиве адресов

```
typedef struct
{
    void **addresses;

    size_t size;
    size_t cap;
} addresses_t;
```

Информация о массиве адресов представляет собой структуру на языке Си, состоящую из:

**addresses — Указатель на память, содержащую использованные под элементы очереди-списка адреса

size – Размер массива

cap – Размер выделенной памяти под массив

Информация о времени прихода/обработки

```
double T1_MIN = 1.0, T1_MAX = 5.0;
double T2_MIN = 0.0, T2_MAX = 3.0;
double T3_MIN = 0.0, T3_MAX = 4.0;
double T4_MIN = 0.0, T4_MAX = 1.0;
```

Информация о времени прихода/обработки представляет собой набор переменных вещественного типа (double).

Описание алгоритма

Для начала считывается код действия. Далее в зависимости от выбранного действия:

<init>

```
case CODE_INIT:
    printf("Enter T1_MIN (enter invalid value if you want to skip): ");
    tmp = T1_MIN;
    if (scanf("%lf", &T1_MIN) != 1){}
    if (clear_buf(stdin) || T1_MIN < -EPS)
        T1_MIN = tmp;

    printf("Enter T1_MAX (enter invalid value if you want to skip): ");
    tmp = T1_MAX;
    if (scanf("%lf", &T1_MAX) != 1){}
    if (clear_buf(stdin) || T1_MAX - T1_MIN < -EPS)
        T1_MAX = tmp;

    printf("Enter T2_MIN (enter invalid value if you want to skip): ");
    tmp = T2_MIN;
```

- Инициализируются по порядку: минимальное время прихода заявки 1-го типа, максимальное время прихода заявки 1-го типа, минимальное время прихода заявки 2-го типа, максимальное время прихода заявки 2-го типа, минимальное время обработки заявки 1-го типа, максимальное время обработки заявки 1-го типа, минимальное время обработки заявки 2-го типа, максимальное время обработки заявки 2-го типа
- Полученные данные проверяются на корректность

<static_service>, <list_service>

```
double static_service(static_queue_t *fst_queue, static_queue_t *sec_queue, dou
```

```
double list_service(list_queue_t *fst_queue, list_queue_t *sec_queue, dou
```

- Добавляются заявки за время, пройденное с момента обработки следующей заявки и заканчивая окончанием обработки (дельта)

```
size_t static_push_by_delta(static_queue_t *queue, double *T, double delta, const double T_min, const double T_max, co
```

```
size_t list_push_by_delta_with_addresses(list_queue_t *queue, double *T, double delta, const double T_min, const double T_max, con
```

- Добавляется следующая заявка в обслуживающий аппарат
- Изменяется время в зависимости от наличия заявок: Если заявка помещена в обслуживающий аппарат, время переносится на единицу дельта = времени окончания обработки заявки. В противном случае время переносится на величину дельта = времени добавления ближайшей новой заявки

<as_output>

- Выводится массив адресов

```
void addresses_show(void)
```

<stat>

```
void stack_statistics(static_stack_t *static_stack, list_stack_t **list_stack_head)
```

- Считается затраченная память на очереди всех типов

```
#define TIME_COMPARING_ITER 1000
```

- Программа циклически выполняет N элементов добавляются и удаляются (порядок важен) TIME_COUNTING_ITER раз.
- Высчитывается среднее значение времени
- Выводится строка с подсчитанными данными

<exit>

```
case CODE_EXIT:
    list_free(&first_list_queue);
    list_free(&second_list_queue);
    addresses_free();

    flag = false;

    break;
```

- Освобождается память, выделенная под динамически выделенные данные программы (массив освобождённых элементов и очередь-список)
- Программа завершает свою работу

Набор тестов

В ходе выполнения лабораторной работы были написаны тесты для проверки работы программы

Позитивные тесты

Номер теста	Входные данные	Ожидаемые выходные данные
01 - <init> Инициализация времён прихода/обслуживания	1 1.1 1.2 1.1 1.2 1.1 1.2 1.1 1.5	T1: 1.1 - 1.2 T2: 1.1 - 1.2 T3: 1.1 - 1.2 T4: 1.1 - 1.5
02 - <static_service> Работа СМО (статическая очередь)	1	*Информация о работе СМО*
03 - <list_service> Работа СМО (очередь-список)	2	*Информация о работе СМО*
04 - <as_output> Вывод массива, имеющего адреса	2 3	ALL ADDRESSES *адреса, используемые для очереди-списка* FRAGMENTATION: *Состояние фрагментации*
05 - <stat> Вывод статистики	4	*Статистика по времени работы с очередями 2^{-x} типов, а также времени работы СМО*

06 - <exit> Выход из программы	5	-

Негативные тесты

Номер теста	Входные данные	Выходные данные
01 - Код неправильный	15	INVALID CODE
02 - Код с иными символами	1.1	INVALID CODE
03 - <as_output> Адресов нет	3	ALL ADDRESSES FRAGMENTATION: NO
04 - <init> Инициализация отрицательного времени	1 1.1 -1 1.1 1.2 1.1 1.2 1.1 1.5	T1: 1.1 - 1.2 T2: 0 - 1.2 T3: 1.1 - 1.2 T4: 1.1 - 1.5
05 - <init> Минимальное значение больше максимального	1 1.1 -1 1.1 1.2	T1: 1.1 - 1.2 T2: 0 - 1.2 T3: 1.1 - 1.2 T4: 0.0 - 1.0

	1.1 1.2 5 0	
06 - <init> 0.0 - 0.0	1 1.1 1.2 0 0 1.1 1.2 0 1	T1: 1.1 - 1.2 T2: 0 - 1.2 T3: 1.1 - 1.2 T4: 0.0 - 1.0

Моделирование

В начале работы программы (и по условию) даны следующие параметры моделирования:
- T1 1 – 5 (Время прихода заявок 1-го типа)
- T2 0 – 3 (Время прихода заявок 2-го типа)
- T3 0 – 4 (Время обработки заявок 1-го типа)
- T4 0 – 1 (Время обработки заявок 2-го типа)

(units – единицы времени)

Статический массив

T1	T2	T3	T4	Ожидаемое время моделирования (units)	Полученное время моделирования (units)	Полученная погрешность (%)
1 - 5	0 - 3	0 - 4	0 - 1	3000	2990.619354	0.758646
5 - 9	0 - 1	0 - 1	0 - 1	7000	7049.294717	0.555957
1 - 2	0 - 3	4 - 7	0 - 1	5500	5500.419912	0.007634
1 - 9	0 - 3	0 - 1	4 - 5	5000	4968.618083	0.631603

T1	T2	T3	T4	Кол-во пришедших заявок (1-й тип)	Кол-во пришедших заявок (2-й тип)	Кол-во обработанных заявок (1-й тип)	Кол-во обработанных заявок (2-й тип)	Среднее время жизни заявки (1-й тип) (units)	Среднее время жизни заявки (2-й тип) (units)	Требуемая память (байт)	Время моделирования (нсек)
1 - 5	0 - 3	0 - 4	0 - 1	1000	1922	1000	1872	0.866784	164.561231	16416	185049.971000
5 - 9	0 - 1	0 - 1	0 - 1	1000	14073	1000	13062	0.335677	238.111517	16416	841179.061000
1 - 2	0 - 3	4 - 7	0 - 1	3049	2049	1000	0	1890.906797	2249.244444	16416	349798.694000
1 - 9	0 - 3	0 - 1	4 - 5	1000	3041	1000	992	2.274503	1747.502697	16416	280235.219000

Список

T1	T2	T3	T4	Ожидаемое время моделирования (units)	Полученное время моделирования (units)	Полученная погрешность (%)
1 - 5	0 - 3	0 - 4	0 - 1	3000	2958.664121	1.397113
5 - 9	0 - 1	0 - 1	0 - 1	7000	6940.500370	0.857282
1 - 2	0 - 3	4 - 7	0 - 1	5500	5511.677548	0.211869
1 - 9	0 - 3	0 - 1	4 - 5	5000	5014.392349	0.287020

T1	T2	T3	T4	Кол-во пришедших заявок (1-й тип)	Кол-во пришедших заявок (2-й тип)	Кол-во обработанных заявок (1-й тип)	Кол-во обработанных заявок (2-й тип)	Среднее время жизни заявки (1-й тип) (units)	Среднее время жизни заявки (2-й тип) (units)	Требуемая память (байт)	Время моделирования (нсек)
1 - 5	0 - 3	0 - 4	0 - 1	1000	2034	1000	1826	1.035515	126.181231	3536	202658.325000
5 - 9	0 - 1	0 - 1	0 - 1	1000	13805	1000	12931	0.338438	220.001557	16256	998788.560000
1 - 2	0 - 3	4 - 7	0 - 1	3667	3643	1000	0	1983.513732	2769.010351	101872	556907.378000
1 - 9	0 - 3	0 - 1	4 - 5	1001	3330	1000	1003	2.257046	1735.455022	37168	336583.926000

Пример вычисления

Если время обработки больше времени прихода, то очередь будет расти, и время моделирования будет определяться временем обработки. В противном случае определяется временем прихода заявок.

В данном случае время моделирования определяется временем прихода заявок 1-го типа, так как среднее время прихода заявки 1-го типа БОЛЬШЕ среднего времени обработки заявки 1-го типа.
T1_AVG = (1 + 5) / 2 = 3
t_ожидаемое = T1_AVG * кол-во пришедших заявок 1-го типа = 3 * 1000 = 3000 units.

При работе программы получили время t_полученное = 2990.619354 units

Погрешность = (t_полученное — t_ожидаемое) / t_полученное * 100% ~= 0.758646%

Таблицы с результатами измерений времени и памяти при выполнении различных операций со статической очередью и очередью-списком

Ниже представлены результаты статистики при различных количествах элементов (для получения значения использовалось 1000 итераций (Значение TIME_COMPARING_ITER))

Таблица для очереди

STATISTICS (time in nsec) (memory in bytes) (STATIC QUEUE SIZE IS 2049)

N count	PUSH t (static)	POP t (static)	PUSH t (list)	POP t (list)	Memory (static)	Memory (list)	best t (PUSH)	best t (POP)	best memory
2	21	22	30	32	16416	72	STATIC	STATIC	LIST
4	32	29	48	47	16416	104	STATIC	STATIC	LIST
8	43	43	84	81	16416	168	STATIC	STATIC	LIST
16	69	71	165	141	16416	296	STATIC	STATIC	LIST
32	120	133	308	281	16416	552	STATIC	STATIC	LIST
64	234	246	593	538	16416	1064	STATIC	STATIC	LIST
128	431	469	1168	1048	16416	2088	STATIC	STATIC	LIST
256	837	919	2331	2080	16416	4136	STATIC	STATIC	LIST
512	1653	1815	4600	4138	16416	8232	STATIC	STATIC	LIST
1024	3316	3638	9291	8399	16416	16424	STATIC	STATIC	STATIC
2048	6560	7194	20515	18827	16416	32808	STATIC	STATIC	STATIC

```
Average service time (static): 105277.371000
Average push time (static): 3.252565
Average pop time (static):  3.561065

Average service time (list): 143215.074000
Average push time (list):  9.558622
Average pop time (list): 8.698583

Static PUSH time is better than list on: 293.879543%
Static POP time is better than list on: 244.269154%

--------------------------------
```

На основе полученных данных можно сделать следующие выводы:

- При работе с маленьким количеством элементов стека очередь-список будет выгоднее по памяти, но при большом количестве элементов очередь-список будет проигрывать статической очереди по памяти.
- Вне зависимости от количества элементов PUSH и POP статической очереди работают быстрее PUSH и POP очереди-списка: среднее время PUSH статической очереди быстрее среднего времени PUSH очереди-списка в 3 раза или в 300% (9.558622 nsec / 3.252565 nsec ~= 3); среднее время POP статической очереди быстрее среднего времени POP очереди-списка в 2 раза или в 245% (8.698583 nsec / 3.561065 nsec ~= 2.45).
- С возрастанием количества добавленных/удалённых элементов скорость операций увеличивается, но с ~512 элементов скорость нормализуется и колеблется в окрестности среднего значения.
- Система массового обслуживания при статической очереди работает быстрее системы массового обслуживания при очереди-списке
- Несмотря на такую разницу в PUSH и POP время моделирование не особо различается, так как функция псевдрандома потребляет значительно больше ресурсов

Сравнение FIFO и LIFO

1 Ниже представлены таблицы измерения статистики для стека и очереди (размер статического(ой) стека(очереди) = 2049 элементов)

Таблица для очереди

```
STATISTICS (time in nsec) (memory in bytes) (STATIC QUEUE SIZE IS 2049)
N count |PUSH t (static) | POP t (static) |  PUSH t (list) |   POP t (list) |Memory (static) | Memory (list)  |  best t (PUSH) |  best t (POP)  |  best memory   |
2       |21              |22              |30              |32              |16416           |72              |     STATIC     |     STATIC     |      LIST      |
4       |32              |29              |48              |47              |16416           |104             |     STATIC     |     STATIC     |      LIST      |
8       |43              |43              |84              |81              |16416           |168             |     STATIC     |     STATIC     |      LIST      |
16      |69              |71              |165             |141             |16416           |296             |     STATIC     |     STATIC     |      LIST      |
32      |120             |133             |308             |281             |16416           |552             |     STATIC     |     STATIC     |      LIST      |
64      |234             |246             |593             |538             |16416           |1064            |     STATIC     |     STATIC     |      LIST      |
128     |431             |469             |1168            |1048            |16416           |2088            |     STATIC     |     STATIC     |      LIST      |
256     |837             |919             |2331            |2080            |16416           |4136            |     STATIC     |     STATIC     |      LIST      |
512     |1653            |1815            |4600            |4138            |16416           |8232            |     STATIC     |     STATIC     |      LIST      |
1024    |3316            |3638            |9291            |8399            |16416           |16424           |     STATIC     |     STATIC     |     STATIC     |
2048    |6560            |7194            |20515           |18827           |16416           |32808           |     STATIC     |     STATIC     |     STATIC     |

Average service time (static): 105277.371000
Average push time (static): 3.252565
Average pop time (static):  3.561065

Average service time (list): 143215.074000
Average push time (list):  9.558622
Average pop time (list): 8.698583

Static PUSH time is better than list on: 293.879543%
Static POP time is better than list on: 244.269154%

--------------------------------
```

Таблица для стека

```
STATISTICS (time in nsec) (memory in bytes) (STATIC STACK SIZE IS 2049)
N count |PUSH t (static) | POP t (static) |  PUSH t (list) |   POP t (list) |Memory (static) | Memory (list)  |  best t (PUSH) |  best t (POP)  |  best memory   |
2       |26              |29              |28              |30              |2064            |40              |     STATIC     |     STATIC     |      LIST      |
4       |27              |31              |33              |39              |2064            |72              |     STATIC     |     STATIC     |      LIST      |
8       |40              |49              |70              |67              |2064            |136             |     STATIC     |     STATIC     |      LIST      |
16      |64              |81              |120             |125             |2064            |264             |     STATIC     |     STATIC     |      LIST      |
32      |116             |147             |223             |243             |2064            |520             |     STATIC     |     STATIC     |      LIST      |
64      |232             |283             |416             |464             |2064            |1032            |     STATIC     |     STATIC     |      LIST      |
128     |445             |554             |811             |896             |2064            |2056            |     STATIC     |     STATIC     |      LIST      |
256     |877             |1127            |1581            |1790            |2064            |4104            |     STATIC     |     STATIC     |     STATIC     |
512     |1767            |2221            |3106            |3586            |2064            |8200            |     STATIC     |     STATIC     |     STATIC     |
1024    |3580            |4437            |6180            |7231            |2064            |16392           |     STATIC     |     STATIC     |     STATIC     |
2048    |7148            |8894            |12304           |14494           |2064            |32776           |     STATIC     |     STATIC     |     STATIC     |

Average push time (static): 3.498290
Average pop time (static):  4.360772
Average push time (list):  6.075232
Average pop time (list): 7.074988

Static PUSH time is better than list on: 173.662896%
Static POP time is better than list on: 162.241640%

--------------------------------
```

На основе полученных данных можно сделать следующие выводы:

- Размер статического стека примерно в 8 раз меньше размера очереди. Это связано с различием данных, хранящихся в этих типах данных (double у статической очереди и char у статического стека). Если же предположить, что данные одного типа, статическая очередь требует больше памяти, нежели статический стек из-за наличия указателей (дополнительные расходы памяти), на основе которых работает очередь.
- При работе с очередью количество элементов, при котором выгода по памяти у очереди-списка, больше количества элементов при работе со стеком.

- PUSH и POP статической очереди работают быстрее PUSH и POP статического стека. Это связано с тем, что работа с указателями (статическая очередь) быстрее работы с индексами (статический стек). В это же время операции с очередью-списком медленнее, чем операции со списком-стеком.
- Для работы с очередью-списком при любом количестве элементов требуется больше памяти, чем для работы со списком-стеком
- И в очереди, и в стеке операции со статическим типом данных быстрее операций с динамическим типом данных (список).

Выводы по проделанной работе

Если нельзя предположить, с каким объёмом данных будет работать программа, стоит реализовать очередь как связный список: данный вариант ограничивается лишь аппаратными возможностями компьютера. В противном случае стоит реализовать очередь через статический массив: скорость операций при статическом массиве в разы быстрее, чем операции при связном списке.

При сравнении статических реализаций типа данных, работающих по принципу LIFO и FIFO среднее время работы операций немного отличалась, но это было связано с особенностью реализации (например, скорость изменения данных при помощи индексации и указателей).

При сравнении динамических реализаций типа данных, работающих по принципу LIFO и FIFO среднее время работы операций немного отличалась.

(ПРИМЕЧАНИЕ! В ходе работы программы фрагментация не наблюдалась, но при работе с утилитой Valgrind фрагментация наблюдалась)

Контрольные вопросы

1. Что такое FIFO и LIFO?

FIFO – принцип работы структуры данных, имеющий расшифровку «First In – First Out», означающая «Первый зашёл — первый вышел». По такому принципу работает тип данных «очередь»

LIFO – принцип работы структуры данных, имеющий расшифровку «Last In – First Out», означающая «Последний зашёл — первый вышел». По такому принципу работает тип данных «стек»

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

РЕАЛИЗАЦИЯ СО СТАТИЧЕСКИМ МАССИВОМ

Структура данных содержит в себе массив, содержащий информацию, указатели на «хвост» и «голову», а также переменную, при помощи которой можно различить пустую очередь от переполненной (если статическая очередь кольцевая).

РЕАЛИЗАЦИЯ С ОЧЕРЕДЬЮ-СПИСКОМ

Структура данных содержит в себе указатели на «голову» и «хвост» очереди. Указатели указывают на структуру данных, содержащую информацию и указатель на следующий элемент очереди-списка

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

РЕАЛИЗАЦИЯ СО СТАТИЧЕСКИМ МАССИВОМ

Во время работы программы память остаётся неизменной и уничтожается при завершении работы программы. Для условного удаления «удалённый» элемент зануляется

РЕАЛИЗАЦИЯ С ОЧЕРЕДЬЮ-СПИСКОМ

При удалении память освобождается динамически

4. Что происходит с элементами очереди при ее просмотре?

При просмотре элементов очереди данные вытягиваются (POP), а потом снова добавляются в очередь в конец (PUSH), что приводит к тому, что после просмотра ВСЕХ элементов очередь не изменилась.

5. От чего зависит эффективность физической реализации очереди?

Эффективность физической реализации очереди зависит от:

- Памяти, выделенной под очередь
- Скорости операций над очередью

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

РЕАЛИЗАЦИЯ СО СТАТИЧЕСКИМ МАССИВОМ

+ Высокая скорость работы

+ При большом количестве элементов выгоден по памяти по сравнению с реализацией с очередью-списком.

- Размер ограничен

- При небольшом количестве элементов невыгоден по памяти по сравнению с реализацией с очередью-списком.

РЕАЛИЗАЦИЯ С ОЧЕРЕДЬЮ-СПИСКОМ

+ Количество элементов зависит от аппаратных возможностей машины

+ При небольшом количестве элементов требует меньше памяти в отличие от реализации со статическим массивом

- Операции требуют больше времени, чем операции со статическим массивом

- При большом количестве элементов требует больше памяти в отличие от реализации со статическим массивом

7. Что такое фрагментация памяти, и в какой части ОП она возникает?

Фрагментации — это явление деления памяти на отдельные независимые блоки, необходимые для хранения данных. В оперативной памяти она возникает в куче.

8. Для чего нужен алгоритм «близнецов».

Алгоритм «близнецов» выделения памяти — алгоритм, при котором буфер делится на блоки памяти степени 2^n. Это сделано для эффективного выделения памяти и снижения фрагментации, жертвуя при этом дополнительной памятью.

9. Какие дисциплины выделения памяти вы знаете?

- Дисциплина выделение памяти «Первый подходящий»
- Дисциплина выделение памяти «Наиболее подходящий»

10. На что необходимо обратить внимание при тестировании программы?

При тестировании программы необходимо обратить внимание на:

- Диапазон числовых данных, при котором программа работает корректно
- Аппаратные возможности машины
- Корректность входных данных
- Логическое соотношение взаимосвязных данных

В данном случае при тестировании программы необходимо:

- проверить правильность работы программы при различном заполнении очередей, т.е., когда время моделирования определяется временем обработки заявок и когда определяется временем прихода заявок;
- отследить переполнение очереди, если очередь в программе ограничена.

При реализации очереди списком необходимо тщательно следить за освобождением памяти при удалении элемента из очереди. При этом по запросу пользователя должны выдаваться на экран адреса памяти, содержащие элементы очереди при добавлении или удалении элемента очереди. Следует проверять, происходит ли при этом фрагментация памяти (выделение непоследовательных адресов память)

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

Программа отправляет запрос ОС на выделение памяти точного размера. ОС (при нахождении подходящей) выделяет программе кусок памяти в куче (часть ОП) запрашиваемого размера (помечает память занятой).

При запросе на удаление по указанному логическому адресу ОС освобождает кусок памяти в куче (помечает память свободной).