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 **Абдуллаев Ш. В.**

Группа **ИУ7-34Б**

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

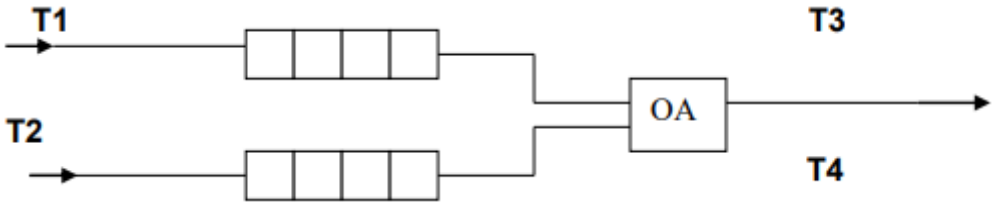
Студент _____ **Абдуллаев Ш. В.**

Преподаватель _____ **Силантьева А. В**

2024

Описание условия задачи

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 памяти.

Описание входных данных

Программа принимает на вход вещественные значения для добавления (удаления) их в различные структуры данных: очередь как статический массив, очередь как динамический массив, очередь как список. Сами эти структуры могут служить входными данными для выполнения различных операций. Также входными данными могут быть параметры моделирования: интервал времени поступления заявки, интервал времени обработки заявки, количество циклов обслуживания.

Описание исходных данных

Программа представляет собой консольное приложение для работы с очередью в виде статического массива, динамического массива и списка. Основные возможности работы со структурами включают:

- Добавить элемент: пользователь вводит число для добавления. Если число уже существует, программа уведомляет об этом.
- Удалить элемент: пользователь вводит число для удаления. Если число отсутствует, программа уведомляет, что удаление невозможно.
- Вывод текущего состояния очереди: элементы очереди можно отобразить в консоли.

Имеются следующие ограничения:

- Размер очереди: программа может иметь ограничение на максимальный размер очереди, как это определено в реализации очереди на основе статического массива. Если пользователь попытается добавить в полный массив, добавление не будет выполнено.
- Удаление из пустой очереди: если пользователь пытается удалить элемент из пустой очереди, удаление не будет выполнено.

Допустимый ввод	**Недопустимый ввод**
Введите число для добавления: 15	Введите число для поиска: 15abc
Введите число для удаления: 12	Введите число для удаления: 2a2
Выберите команду: 1	Выберите команду: q
Какой интервал изменить (1 – 4): 1	Какой интервал изменить (1 – 4): 55
Введите левую и правую границы: 1 3	Введите левую и правую границы: 1 q

Таблица 1. Примеры ввода

Описание цели

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.

Описание команд

1. Выйти из программы: пользователь завершает выполнение программы.
2. Добавление элемента в очередь (статический массив): пользователь добавляет элемент в очередь, реализованную с использованием статического массива.
3. Удаление элемента из очереди (статический массив): удаляется первый элемент из очереди, реализованной с использованием статического массива.
4. Вывод текущего состояния очереди (статический массив): отображение всех элементов очереди, реализованной на основе статического массива.
5. Добавление элемента в очередь (динамический массив): пользователь добавляет элемент в очередь, реализованную с использованием динамического массива.
6. Удаление элемента из очереди (динамический массив): удаляется первый элемент из очереди, реализованной на основе динамического массива.

7. Вывод текущего состояния очереди (динамический массив): отображение всех элементов очереди, реализованной на основе динамического массива.
8. Добавление элемента в очередь (список): пользователь добавляет элемент в очередь, реализованную с использованием односвязного списка.
9. Удаление элемента из очереди (список): удаляется первый элемент из очереди, реализованной на основе списка.
10. Вывод текущего состояния очереди (список): отображение всех элементов очереди, реализованной на основе списка, а также удаленные адреса.
11. Моделирование и характеристика для очереди в виде массива: пользователь может смоделировать система массового обслуживания на основе массива.
12. Моделирование и характеристика для очереди в виде списка: пользователь может смоделировать система массового обслуживания на основе списка.
13. Изменить время обработки заявки: пользователь может изменить параметры обработки заявки в консоли за счет разработанного интерфейса.
14. Вывод сравнения эффективности при выполнении операций: программа выполняет сравнение времени выполнения операций и объема памяти, затраченного при различных способах реализации очереди.

Способ обращения к программе

Обращения к программе пользователем происходит с помощью вызова исполняемого файла (app.exe).

Описание возможных аварийных ситуаций и ошибок пользователя

- INPUT_COMMAND_ERR: ошибка ввода команды. Возникает, если пользователь вводит некорректное значение при выборе команды из меню.
- INPUT_FOR_INSERT_ERR: ошибка ввода числа для добавления. Возникает, если пользователь вводит некорректное значение при добавлении элемнта.

Описание внутренних структур данных

Программа содержит в себе структуру, которая используется для хранения очереди в виде статического массива, которая представлена в Листинге 1.

```
#define MAX_QUEUE_SIZE 100


typedef struct
{
    int data[MAX_QUEUE_SIZE];
    int head;
    int tail;
    int size;
} queue_arr_t;
```

Листинг 1. Структура queue_arr_t

Рассмотрим каждое поле структуры:

1. data: массив целых чисел типа int[MAX_QUEUE_SIZE], который хранит элементы очереди.
2. head: целочисленное значение типа int, которое указывает на индекс первого элемента очереди.
3. tail: целочисленное значение типа int, которое указывает на индекс последнего элемента очереди.
4. size: целочисленное значение типа int, которое хранит текущий размер очереди, то есть количество элементов в очереди.

Также программа содержит в себе структуру, которая используется для хранения очереди в виде динамического массива, которая представлена в Листинге 2.

```
typedef struct
{
    int *data;
```

```
    int capacity;
    int size;
    int head;
    int tail;
} queue_dyn_arr_t;
```

Листинг 2. Структура queue_dyn_arr_t

Рассмотрим каждое поле структур:

1. data: указатель на массив целых чисел типа int, который хранит элементы очереди (динамически выделяется).
2. capacity: целочисленное значение типа int, которое указывает на максимально возможное количество элементов в очереди.
3. size: целочисленное значение типа int, которое хранит текущий размер очереди, то есть количество элементов в очереди.
4. head: целочисленное значение типа int, которое указывает на индекс первого элемента очереди.
5. tail: целочисленное значение типа int, которое указывает на индекс последнего элемента очереди.

Также программа содержит в себе структуру, которая используется для хранения очереди в виде списка, которая представлена в Листинге 3.

```
typedef struct list_node_t
{
    int data;
    struct list_node_t *next;
} list_node_t;

typedef struct
{
    list_node_t *head;
```

```
    list_node_t *tail;
} queue_list_t;
```

Листинг 3. Структура queue_list_t

Рассмотрим каждое поле структур:

- list_node_t:

1. data: целочисленное значение типа int, которое хранит данные узла.
2. next: указатель на следующий узел типа list_node_t, который связывает текущий узел с последующим в списке.

- queue_list_t:

1. head: указатель на первый элемент списка типа list_node_t, который указывает на начало очереди.
2. tail: указатель на последний элемент списка типа list_node_t, который указывает на конец очереди.

Описание алгоритма

1. Инициализация системы и выделение памяти.
2. Основной цикл программы начинается. Пользователю предоставляется меню с различными опциями, и программа ожидает ввода выбора пользователя.
3. В зависимости от выбора пользователя, программа выполняет следующие действия:
   - Выйти из программы

   Очередь как статический массив:

   - Добавить элемент

     Если очередь полна, выводится сообщение об ошибке, и операция завершается. В противном случае индекс tail обновляется с использованием циклического перехода, новый элемент записывается в массив, размер очереди увеличивается, и выводится сообщение о добавлении элемента.
   - Удалить элемент

Функция удаляет элемент из очереди, предварительно проверяя, пуста ли она. Если очередь пуста, выводится сообщение об ошибке, и операция завершается. В противном случае считывается элемент из head, индекс head обновляется с использованием циклического перехода, размер очереди уменьшается, и выводится сообщение об удалении элемента.

- Вывод текущего состояния очереди

Очередь как динамический массив:

- Добавить элемент

  Функция добавляет элемент в динамическую очередь. Если очередь заполнена, её ёмкость увеличивается в ALLOC_SIZE раз, при необходимости элементы переносятся для сохранения структуры циклической очереди. После этого индекс tail обновляется с использованием циклического перехода, элемент добавляется в массив, размер очереди увеличивается, и выводится сообщение об успешном добавлении. Если память не удалось выделить, выводится ошибка, и операция прекращается.
- Удалить элемент
- Вывод текущего состояния очереди

Очередь как список:

- Добавить элемент

  Функция добавляет элемент в очередь на основе связного списка. Новый узел выделяется с помощью malloc, и, если память не выделена, выводится сообщение об ошибке. Затем новый узел создается с переданным значением. Если очередь пуста, новый узел становится её головой; иначе он добавляется в конец, обновляя указатель tail. После успешного добавления элемента выводится сообщение.
- Удалить элемент

Функция удаляет элемент из очереди на основе связного списка. Если очередь пуста, выводится сообщение, и операция завершается. В противном случае первый элемент (head) удаляется: сохраняется его значение, указатель head обновляется на следующий узел, а если очередь стала пустой, указатель tail обнуляется. Удаляемый узел освобождается с помощью free, и выводится сообщение об удалении элемента.

- Вывод текущего состояния очереди

Общие команды:

- Моделирование и характеристика для очереди в виде массива *
- Моделирование и характеристика для очереди в виде списка *

  * Сначала инициализируются две очереди определенной реализации, переменные для статистики, а также случайное время для поступления заявок 1-го и 2-го типов. Затем запускается основной цикл (пока не обслужено 1000 заявок 1-го типа): определяется, какая заявка (1-го или 2-го типа) будет следующей; выполняется обслуживание заявки, обновляются статистики; каждые 100 обслуженных заявок выводится статистика. После создаются новые заявки с случайными интервалами, и они добавляются в соответствующие очереди. В конце симуляции выводится общее время, количество выполненных заявок, погрешности по времени и количеству заявок.
- Изменить время обработки заявки
- Вывод сравнения эффективности при выполнении операций

4. После выполнения каждой операции программа возвращает пользователя в главное меню, где он может выбрать следующее действие.

Набор тестов

Описание	**Результат**
Добавить элемент в очередь в виде статического массива	Успешное добавление элемента в очередь в виде статического массива

Элемент: 3	Добавленный элемент: 3
Удалить элемент из очереди в виде статического массива Очередь: 1 2 3	Успешное удаление элемента из очереди в виде статического массива Удалённый элемент: 3
Вывести очередь в виде статического массива Очередь: 1 2 3	Успешный вывод очереди в виде статического массива: Состояние очереди: --> 1 2 3 -->
Добавить элемент в очередь в виде списка Элемент: 399	Успешное добавление элемента в очередь в виде списка Добавленный элемент: 399
Удалить элемент из очереди в виде списка Очередь: 12 43 55	Успешное удаление элемента из очереди в виде списка Удалённый элемент: 55

Таблица 2. Набор позитивных тестов

Описание	**Результат**
Попытка ввести символы вместо команды	Возврат ошибки INPUT_COMMAND_ERR
Попытка ввести не целое число	Возврат ошибки INPUT_VALUE_ERR
Неудачная аллокация памяти	Возврат ошибки INIT_LIST_ALLOC_ERR
Попытка ввести не целое число при поиске элемента	Возврат ошибки INPUT_FOR_FIND_ERR
Попытка ввести не целое число при добавлении элемента	Возврат ошибки INPUT_FOR_INSERT_ERR

Таблица 3. Набор негативных тестов

Оценка моделирования

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

1 случай – когда работа ОА зависит от среднего времени прихода первой заявки:

Время прихода заявок 1-го и 2-го типа	Время обработки заявок 1-го и 2-го типа
T1: 1...5	T3: 0...4
T2: 0...3	T4: 0...1

Чтобы найти время моделирования, мы должны найти максимальное время между обработкой и приходом заявки первого типа (потому что у нее относительный приоритет). В данном случае время прихода больше времени обработки, и среднее время прихода равно 3 единиц условного времени. Так как обрабатывается заявка первого типа быстрее, чем приходит, то у нас почти каждая заявка первого типа будет обработана.

Время моделирования будет равно 1000 * (ср. время прихода первой заявки) = 3000 у. е. в. За это время успеет прийти 2000 заявок второго типа. Это находится из выражения (Время Моделирования / max(время обр, время прих)). В данном случае время прихода в среднем больше времени обработки для второй заявки и равно 1.5. То есть 2000 заявок = 3000 / 1.5. Так как они обрабатываются быстрее, чем приходят, то почти все заявки второго типа будут обработаны во время простоя ОА. Простой ОА гарантирован, так как заявки первого типа обрабатываются быстрее, чем приходят.

Расчет время моделирования заявок:

Число заявок 1 типа, вошедших: 1000, вышедших = 1000
Число заявок 2 типа, вошедших: 2000

```
Вышедших = (время моделирования / max(приход, обработки)) =
= 3000 / max(1.5, 0.5) = 2000
Время моделирования: 2000
```

Листинг 5. Практические результаты для очереди в виде массива

```
Общее время моделирования: 2979.661165
Ожидаемое время моделирования: 3000.000000
Погрешность работы ОА: 0.677961%
Время простоя ОА: 30.962249
Время работы (мкс): 268
Среднее время обработки заявки 1 очереди: 3.000000
Среднее время обработки заявки 2 очереди: 1.500000
Число вошедших в 1 очередь: 1001
Число вышедших из 1 очереди: 1000
Число вошедших во 2 очередь: 1964
Число вышедших из 2 очереди: 1871
Погрешность ввода 1 очереди: 0.783271%
Погрешность ввода 2 очереди: 1.129698%
```

Листинг 6. Практические результаты для очереди в виде списка

```
Общее время моделирования: 3027.076412
Ожидаемое время моделирования: 3000.000000
Погрешность работы ОА: 0.902547%
Время простоя ОА: 53.202544
Время работы (мкс): 6401
Среднее время обработки заявки 1 очереди: 3.000000
Среднее время обработки заявки 2 очереди: 1.500000
Число вошедших в 1 очередь: 1001
```

Число вышедших из 1 очереди: 1000
Число вошедших во 2 очередь: 2035
Число вышедших из 2 очереди: 2010
Погрешность ввода 1 очереди: 0.795369%
Погрешность ввода 2 очереди: 0.839873%

2 случай – когда работа ОА зависит от среднего времени обработки первой заявки:

Время прихода заявок 1-го и 2-го типа	Время обработки заявок 1-го и 2-го типа
T1: 1...5	T3: 2…6 !(изменено)
T2: 0...2 !(изменено)	T4: 0...1

Чтобы найти время моделирования, мы должны найти максимальное время между обработкой и приходом заявки первого типа (потому что у нее относительный приоритет). В данном случае время обработки больше времени прихода, и среднее время обработки равно 4 единиц условного времени. Так как обрабатывается заявка первого типа дольше, чем приходит, то будут заявки 1-го типа, которые не успеют обработаться, а также время простоя будет низким.

Время моделирования будет равно 1000 * (ср. время обработки первой заявки) = 4000 у. е. в. За это время успеет прийти 4000 заявок второго типа. Это находится из выражения (Время Моделирования / max(время обр, время прих)). В данном случае время прихода в среднем больше времени обработки для второй заявки и равно 1 е. в. То есть 4000 заявок = 4000 / 1. Заявки второго типа обрабатываются быстрее, чем приходят, но из-за того, что время обработки заявки 1-го типа больше времени её прихода, то с основном в ОА будут обрабатываться заявки 1-го типа, и заявок 2-го типа будет обработано мало. Т.е. к концу выполнения моделирования очередь из заявок 2-го типа будет примерно равна 4000 заявок.

Расчет время моделирования заявок:

```
Число заявок 1 типа, вошедших: (время моделирования / время прихода)
4000 / 3 = 1333;
Вышедшие заявки первого типа = 1000
Число заявок 2 типа, вошедших: 4000, вышедших = 0
Время моделирования: 4000
```

Листинг 7. Практические результаты для очереди в виде массива

```
Общее время моделирования: 3991.586529
Ожидаемое время моделирования: 4000.000000
Погрешность работы ОА: 0.210337%
Время простоя ОА: 0.410131
Время работы (мкс): 535
Среднее время обработки заявки 1 очереди: 3.000000
Среднее время обработки заявки 2 очереди: 1.000000
Число вошедших в 1 очередь: 1331
Число вышедших из 1 очереди: 1000
Число вошедших во 2 очередь: 4036
Число вышедших из 2 очереди: 2
Погрешность ввода 1 очереди: 0.035411%
Погрешность ввода 2 очереди: 1.112677%
```

Листинг 8. Практические результаты для очереди в виде списка

```
Общее время моделирования: 4125.390263
Ожидаемое время моделирования: 4000.000000
Погрешность работы ОА: 3.134757%
Время простоя ОА: 0.613014
Время работы (мкс): 11931
```

Среднее время обработки заявки 1 очереди: 3.000000 Среднее время обработки заявки 2 очереди: 1.000000 Число вошедших в 1 очередь: 1343 Число вышедших из 1 очереди: 1000 Число вошедших во 2 очередь: 4027 Число вышедших из 2 очереди: 4 Погрешность ввода 1 очереди: 0.781513% Погрешность ввода 2 очереди: 0.799120%

Оценка эффективности

При запуске программы N = 500 раз, были получены следующие данные:

Таблица 4. Эффективность добавления положительно возрастающей последовательности целых чисел в различные структуры очередей

Размер очереди	Статический массив		Динам. массив		Список	
	Время, нс	Память, б	Время, нс	Память, б	Время, нс	Память, б
10	123	4012	473	80	558	176
50	459	4012	1194	272	1759	816
100	1035	4012	1975	528	3811	1616
500	4551	4012	8927	2064	16704	8016
1000	8159	4012	15669	4112	32951	16016

Таблица 5. Эффективность удаления всех элементов из различных структур очередей

Размер очереди	Статический массив		Динам. массив		Список	
	Время, нс	Память, б	Время, нс	Память, б	Время, нс	Память, б
10	157	4012	199	80	348	176
50	453	4012	550	272	1487	816

100	905	4012	1120	528	2834	1616
500	4336	4012	5246	2064	15431	8016
1000	7427	4012	11314	4112	30870	16016

Выводы

В данной лабораторной работе было проведено равнение производительности очереди на основе статического, динамического массива и связанного списка. Были рассмотрены операции на разных размера очереди: 10, 50, 100, 500, 1000 элементов. Результаты сравнения были представлены в виде таблицы, в которой указаны размер очереди, объем занимаемой памяти для статического, динамического массива и связанного списка, а также время выполнения операций для всех типов очереди.

В результате анализа, мы видим, что очередь на основе статического массива имеет лучшую эффективность по скорости добавления на 56% относительно динамического массива и на 78% относительно списка, также по скорости удаления на 21% относительно динамического массива и на 68% относительно списка. Однако динамический массив выигрывает по памяти статический массив на 64% и список на 67% список до 500 элементов в очереди, на больших выигрывает статический массив.

В современных системах тип данных "очередь" активно используется, и выбор реализации зависит от конкретных требований. Если важна высокая скорость доступа к элементам, лучше использовать статический массив, обеспечивающий мгновенный доступ, но с риском переполнения. Если приоритетом являются гибкость и отсутствие ограничений на размер, целесообразнее выбрать реализацию на основе связного списка, которая позволяет динамически изменять размер очереди и эффективно использовать память. Таким образом, разработчику необходимо балансировать между производительностью и удобством при выборе реализации.

Контрольные вопросы

1. Что такое FIFO и LIFO?

Очередь (FIFO) — структура данных, где добавление элементов происходит с одной стороны ("хвост"), а удаление — с другой ("голова"). Работает по принципу "первым пришел — первым вышел" (First In — First Out).

Стек (LIFO) — структура данных, где добавление и удаление элементов выполняются с одной стороны — с вершины. Принцип работы: "последним пришел — первым ушел" (Last In — First Out).

2. Как выделяется память для очереди при разных реализациях?

Реализация очереди в виде массива:

Для линейной очереди используется одномерный массив с выделением области памяти из m элементов по L байт каждый (L — размер одного элемента). Область памяти может быть полностью свободной, частично заполненной или полностью занятой.

Реализация очереди в виде линейного списка:

В статической памяти хранятся указатели на начало и конец очереди и возможно количество элементов. Динамически выделяется память для каждого нового элемента.

3. Как освобождается память при удалении элемента?

При реализации статическим массивом память из-под элемента не освобождается, так как она выделена на стеке. В такой реализации просто 13 сдвигается указатель выхода очереди на следующий элемент. При реализации списком при удалении память освобождается из-под узла элемента.

4. Что происходит с элементами очереди при ее просмотре?

В классической реализации при просмотре очереди ее элементы из нее удаляются.

5. От чего зависит эффективность физической реализации очереди?

Эффективность реализации зависит от вида выделения памяти, так как динамическое выделение памяти как правило замедляет программу, а также от размера дополнительных данных, таких как указатель на следующий элемент при реализации списком.

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

Особенностями реализации очереди статическим массивом является большая скорость выполнения операций добавления и удаления и эффективность по памяти, однако при данной реализации может возникнуть переполнение массива и такая реализация сложнее для написания, из-за кольцевой структуры. Однако если структура статического массива не кольцевая, то удаление элемента может быть сильно замедлено из-за сдвига.

Реализация списком, как правило, работает медленнее из-за динамического выделения памяти и занимает больше памяти, однако размер такой очереди зависит только от объема оперативной памяти.

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

Фрагментация памяти - явление, при котором данные некоторой структуры данных разбросаны по разным участкам памяти, а не лежат друг за другом. Такое может возникать в куче при динамическом выделении/освобождении памяти.

8. Для чего нужен алгоритм «близнецов»

Алгоритм близнецов используется для быстрого (O(log n)) выделения памяти по запросу. Суть алгоритма в том, чтобы при запросе памяти разбивать её на блоки кратные степени двойки, доходя до минимального размера, необходимого по запросу.

9. Какие дисциплины выделения памяти вы знаете?

Выделения памяти может быть статическим или динамическим. 14 Статическое выделение памяти происходит для данных, определенных до начала программы, то есть литералы. Динамически выделяются все остальные, при этом память для объектов может быть зарезервирована, если их размер известен до начала программы или выделятся во время выполнения.

10. На что необходимо обратить внимание при тестировании программы?

При реализации списком важно следить, что все узлы при удалении освобождаются, и при завершении программы все ресурсы освобождены.

При реализации массивом важно следить за тем, чтобы данные не начали записываться в области вне массива.

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

При динамическом запросе программа запрашивает у ОС блок памяти необходимого размера. ОС находит блок памяти такого размера и заносит его адрес и размер в таблицу адресов и затем передает адрес в программу.