ПРАКТИЧЕСКАЯ РАБОТА №3

Создание динамического списка объектов на основе структур типа: вектор, очередь, список, словарь.

Цель работы: изучить новые способы хранения структур в динамической памяти и связывание их. Реализовать структуру и методы для хранения ранее созданных структур в динамической памяти. Возможности их чтения и поиска, добавления и удаления, а также перемещения внутри новой структуры.

Задание: реализовать структуру контейнер (***struct***) в соответствии с вариантом. Структура должна обеспечивать набор методов для работы с данными размещая объекты в динамической памяти компьютера. Структура должна быть описана с использованием ***typedef***. Создать методы: конструкторы и деструктор. Реализовать указанные структуры с динамическим выделением памяти для хранения ранее созданной по варианту структуры. Реализовать структуру отдельным модулем (**cont**struct.h, **cont**struct.c), где struct название структуры.

Изменить демонстрационную программу, которая не имеет никаких операторов консольного ввода. Для сборки демонстрационной программы использовать ***makefile.***

Вариант 1. **Создайте структуру словарь** *dict*

Задание на 3. Разработать структуру типа словарь (он же ассоциативный массив), по сути, это массив, но в качестве массивов используются не только цифры, но и любые другие типы данных, при этом зависимости между ключами никакой нет. Каждый элемент словаря — это ключ – значение. Сам список состоит из элемента и количества элементов. **Методы**: создание пустого словаря с инициализацией его пары ключ-значение, добавление/обновление элемента (***put***), получение значения по ключу (***get***), удаление элемента по ключу(***remove***), проверка наличие ключа и элемента по ключу (***cont_key***), проверка на пустоту(***is_empty***), освобождение памяти словаря. **Правила**: Обработка ошибок не требуется, поиск только линейный.

Задание на 4. Выполнить все задания на 3. Реализовать **методы**: очистка словаря (***clear***), возврат текущего размера (***size***), заполнение массива всеми имеющимися в словаре ключами (***dkey***), заполнение массива всеми имеющимися в словаре значениями (***dval***). В методах ***get***, ***remove***, ***cont_key*** проверять существует ли ключ, в ***put*** проверять успешность выделения памяти. Отдельно реализовать простое хеширование ключей `hash(key) = (uintptr_t)key % TABLE_SIZE`.

Задание на 5. Выполнить все задания на 4. Разработать методы копирование словаря(***copy***), слиянии двух словарей (***merge***), перебор элементов (***feach***), словаря над каждым из которых выполняется какая-либо пользовательская функция, ***например*** увеличения на час каждой даты. Разработать ***итераторы*** возвращающие позиции в словаре (начало, следующий элемент, ключ, значение). Реализовать расширение хэш таблицы в зависимости от расширения словаря. Доработать хеширование тем, что доработать функцию перехэширования при изменении размера количества значений в словаре.

Вариант 2. **Создайте структуру очередь** *queue*

Задание на 3. Разработать структуру, которая представляет собой линейную структуру данных в которой каждый элемент хранится последовательно. В очереди реализован метод FIFO (англ. *first in, first out*, «первым пришёл — первым ушёл»). Структура содержит указатель на первый элемент, указатель на последний элемент, количество элементов. **Методы:** Создание пустой очереди, добавление элемента в конце очереди (***enqueue***), извлечение первого элемента и удаление его (***dequeue***), проверка на пустоту очереди, освобождение памяти. **Правила**: Элементы должны быть лишь связаны в одну сторону. Обрабатывать какие-либо ошибки не требуется.
Задание на 4. Выполнить все задания на 3. Реализовать **методы** возврата текущего размера(***size***), просмотр первого элемента(***peek***), очистка очереди (***clear***). В методах ***dequeue*** и ***peek*** добавить проверку на пустоту, а в также в методе ***enqueue*** проверять успешность выделения памяти. Увеличивать ёмкость выделяемой памяти на очередь в 2 раза, для более эффективной работы.
Задание на 5. Выполнить все задания на 4. Реализовать метод добавление элемента в начало(***fenqueue***), копирование очереди (***copy***), слияние двух очередей (***merge***). разработать итераторы возвращающие позиции в очереди (начало, конце, следующий элемент), сравнение итераторов (равны ли итераторы и указывает ли итератор на указанную очередь), реализовать резервирование памяти для очереди, но не с размером в 2 раза больше ёмкости очереди, а в sqrt(2).

Вариант 3. **Создайте структуру вектор** *vector*

Задание на 3. Разработать структуру, которая представляет собой одномерный массив, который представляет собой набор элементов с произвольным доступом по числовому индексу. Структура имеет поле для данных произвольного формата, текущее количество элементов и максимальную ёмкость.
Методы: Создание вектора с начальной ёмкостью, добавление элемента в конец вектора (***push***), извлечение последнего элемента и удаление его из вектора (***pop***), доступ элемента по индексу (***get***), изменение ёмкость вектора, освобождение памяти вектора.
Условия: инициализировать вектор не нулевым значением, если требуется внести элемент в вектор, а объём вектора не позволяет это, увеличивать ёмкость вектора в 2 раза. Обрабатывать какие-либо ошибки не требуется.
Задание на 4. Выполнить все задания на 3. Доработать обработку ошибок в методах ***get*** и ***pop.*** Предусмотреть все типичные ошибки. Разработать **методы:** возврат текущий размер вектора, текущую ёмкость вектора, вставка элемента по индексу, удаление элемента по индексу. При удалении элементов, если количество элементов в массиве ровно четверти ёмкости вектора, уменьшить ёмкость вектора.
Задание на 5. Выполнить все задания на 4. Разработать **методы**: замена элемента по индексу(change), копирование вектора(copy), слияние двух векторов (merge), разработать итераторы возвращающие позиции в векторе (начало, конце, следующий элемент), сравнение итераторов (равны ли итераторы и указывает ли итератор на указанный вектор), реализовать резервирование памяти для вектора, но не с размером в 2 раза больше ёмкости вектора, а в sqrt(2). **Переключать** вектора, при превышении размерности, на резервированную память и резервировать вновь, тем самым оптимизировать работу с памятью.

Вариант 4. **Создайте структуру список** *list*

Задание на 3. Разработать структуру, реализующую двунаправленный список. Суть в том, что данные расположены в памяти в разброс, из-за этого теряется возможность получить элемент по индексу, а также не сможем быстро скопировать весь список. Однако вставка и удаление элемента более эффективны. Элементы структуры содержат данные, указатель на следующий узел и на предыдущей. Сама структуру хранит указатель на первый элемент и на последний элемент, а также количество элементов. **Методы**: создание пустого списка, добавление элемента в начало (***fpush***), добавление элемента в конец (***bpush***), извлечение первого элемента и удаление из списка (***fpop***), извлечение последнего элемента и удаление из списка (***bpop***), проверка на пустоту (***is_empty***), освобождение памяти. **Правила**: обязательно реализовать двусвязный список, где каждый элемент знает где находится следующий и предыдущей элемент. Обрабатывать какие-либо ошибки не требуется.

Задание на 4. Реализовать ***метод*** доступа по индексу (***gbind***), возврата текущего размера списка (***size***), очистка списка (***clear***), вставка по индексу (***insert***), удаление по индексу (***remove***). Доработать методы ***fpop***, ***bpop***, ***gbind*** проверяя наличие элементов в списке. А в метода ***fpush*** и ***bpush*** проверять успешное выделение памяти.

Задание на 5. Реализовать **методы** копирования списка (***copy***), слияния двух списков (***merge***), реверсирование списка (***reverse***), перебор элементов (***feach***), списка над каждым из которых выполняется какая-либо пользовательская функция, ***например*** увеличения баланса на 1000. Разработать **итераторы** возвращающие позиции в списке (начало, конце, следующий элемент). Проверить и переработать функции вставки и удаления на двусвязном списке, также реализовать **метод** вставки списка в список по индексу (splice).