Курс «Объектно-ориентированное программирование на C++»

Встреча №15

Тема: *динамические структуры данных — односвязный список*

Задание 1.
Реализуйте стек в виде односвязного списка. Требуется реализовать типичные операции по работе со стеком. При переполнении стека нужно увеличивать его размер.

Задание 2.
В существующий класс односвязного списка добавить: операцию клонирования списка (функция должна возвращать адрес головы клонированного списка), перегрузить оператор + (оператор должен возвращать адрес головы нового списка, содержащего элементы обоих списков для которых вызывался оператор), перегрузить оператор * (оператор должен возвращать адрес головы нового списка, содержащего только общие элементы обоих списков для которых вызывался оператор).

Встреча №16

Тема: *динамические структуры данных — двусвязный список*

Задание 1.
Реализовать шаблонный класс "Очередь" на основе двусвязного списка.

Задание 2.
В существующий класс двусвязного списка добавить: операцию клонирования списка (функция должна возвращать адрес головы клонированного списка), перегрузить оператор + (оператор должен возвращать адрес головы нового списка, содержащего элементы обоих списков для которых вызывался оператор), перегрузить оператор * (оператор должен возвращать адрес головы нового списка, содержащего только общие элементы обоих списков для которых вызывался оператор).

Задание 3.
Создать шаблонный класс-контейнер Array, который представляет собой массив, позволяющий хранить объекты заданного типа. Класс должен быть реализован с помощью двусвязного списка. Класс должен реализовывать следующие функции:

a. *GetSize* — получение размера массива (количество элементов, под которые выделена память).
b. *SetSize*(*int size, int grow = 1*) — установка размера массива (если параметр size больше предыдущего размера массива, то выделяется дополнительный блок памяти, если нет, то "лишние" элементы теряются и память

освобождается); параметр grow определяет для какого количества элементов необходимо выделить память, если количество элементов превосходит текущий размер массива. Например, SetSize(5, 5); означает, что при добавлении 6-го элемента размер массива становится равным 10, при добавлении 11-го — 15 и т. д.

c. *GetUpperBound* — получение последнего допустимого индекса в массиве. Например, если при размере массива 10, вы добавляете в него 4 элемента, то функция вернет 3.
d. *IsEmpty* — массив пуст?
e. *FreeExtra* — удалить "лишнюю" память (выше последнего допустимого индекса);
f. *RemoveAll* — удалить все;
g. *GetAt* — получение определенного элемента (по индексу);
h. *SetAt* — установка нового значения для определенного элемента (индекс элемента должен быть меньше текущего размера массива);
i. *operator []* — для реализации двух предыдущих функций;
j. *Add* — добавление элемента в массив (при необходимости массив увеличивается на значение grow функции SetSize);
k. *Append* — "сложение" двух массивов;
l. *operator =;*
m. *GetData* — получения адреса массива с данными;
n. *InsertAt* — вставка элемента(-ов) в заданную позицию;
o. *RemoveAt* — удаление элемента(-ов) с заданной позиции.

Встреча №17

Тема: *Бинарное дерево*

Задание.

Реализовать базу данных ГАИ по штрафным квитанциям с помощью бинарного дерева. Ключом будет служить номер автомашины, значением узла — список правонарушений. Если квитанция добавляется в первый раз, то в дереве появляется новый узел, а в списке данные по правонарушению; если нет, то данные заносятся в существующий список. Необходимо также реализовать следующие операции:

- Полная распечатка базы данных (по номерам машин и списку правонарушений, числящихся за ними);
- Распечатка данных по заданному номеру;
- Распечатка данных по диапазону номеров.