МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ
Федеральное государственное бюджетное образовательное учреждение
высшего образования
"МИРЭА - Российский технологический университет"
РТУ МИРЭА

Отчет по выполнению практического задания №6
Тема: Однонаправленный динамический список

Дисциплина: «Структуры и алгоритмы обработки данных»

Выполнил студент Овчаренко М.М.

группа ИКБО-01-20

Москва 2021

Содержание

1. Отчет по разработанной программе

1.1 Постановка задачи

Вариант №3
Тип данных: char
Задача: Даны два линейных однонаправленных списка L1 и L2.

1) Разработать функцию, которая формирует список L, включив в него по одному разу элементы, значения которых входят в список L1 и не входят в список L2.

2) Разработать функцию, которая удаляет подсписок списка L1 заданный диапазоном позиций. Например, со второго три.

3) Разработать функцию, которая упорядочивает значения списка L2, располагая их в порядке возрастания.

1.2 Определение операций над списком

1) Метод, добавляющий новый элемент в конец списка:

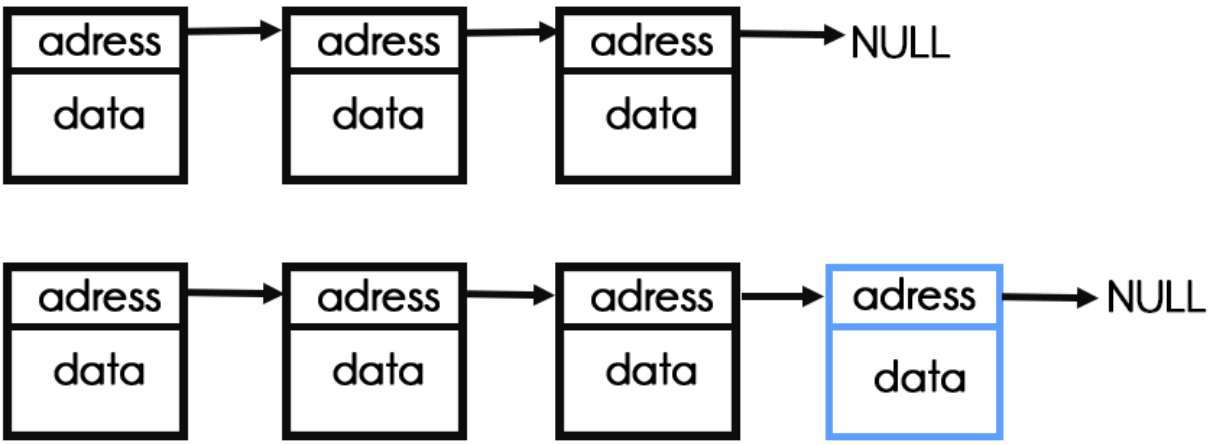

Рис. 1. Метод, добавляющий новое значение в конец списка

2) Метод, удаляющий первый элемент в списке:

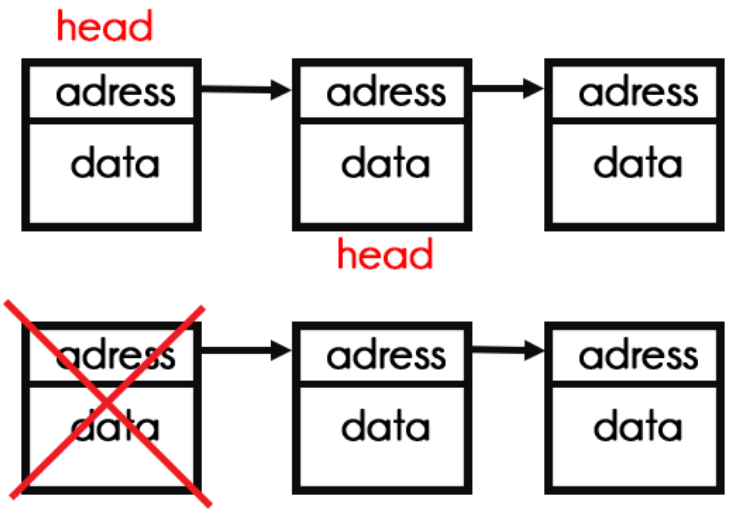

Рис. 2. Метода, удаляющий первый элемент списка

3) Метод, удаляющий элемент по индексу:

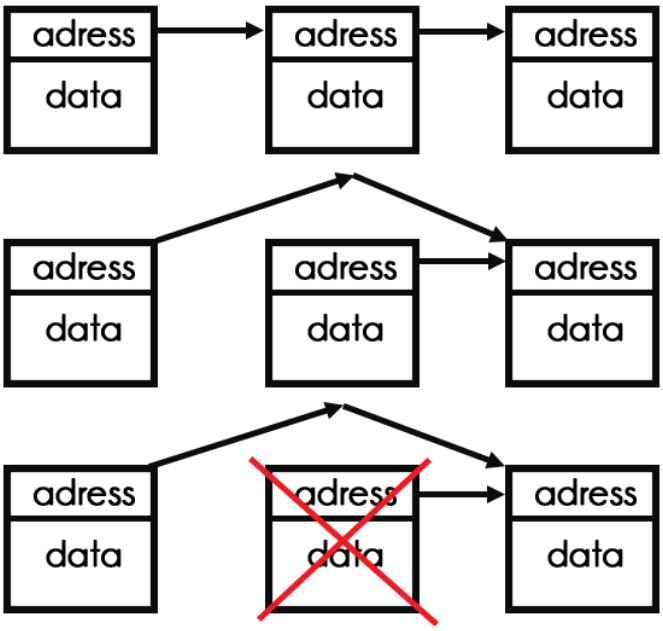

Рис. 3. Метод, удаляющий элемент из списка по индексу

4) Метод, меняющий элементы списка местами:

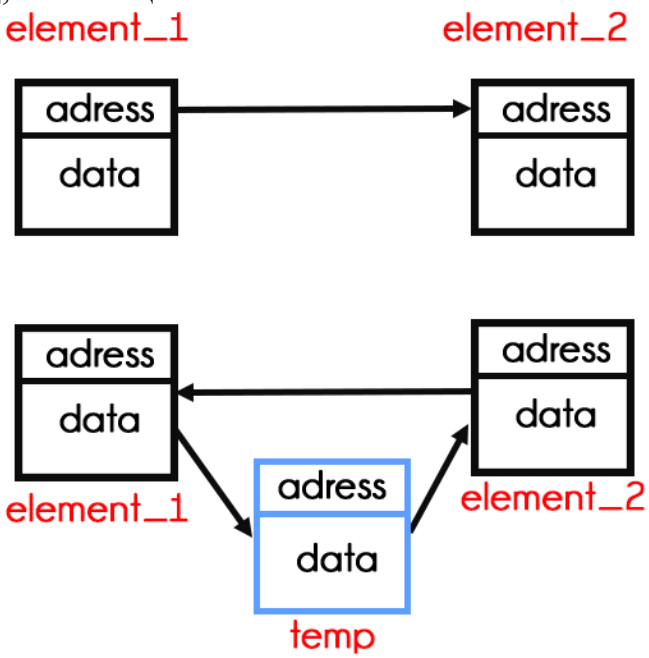

Рис. 4. Метод, меняющий элементы списка местами

Таблица 1. Результаты тестирования метода

№Теста	Входные данные	Ожидаемый результат	Результат работы программы
1	n = 4 index = 2 List_2 = a s d f	a s f d	Amount of elements List_2 = 4 a s d f a s f d
2	n = 3 index = 0 List_2 = c v b	v c b	Amount of elements List_2 = 3 c v b v c b
3	n = 5 index = 6 List_2 = a b c d e	Wrong index	Amount of elements List_2 = 5 a b c d e Wrong index

5) Функция, реализующая заполнение списка L согласно поставленное задаче(Вариант№3):

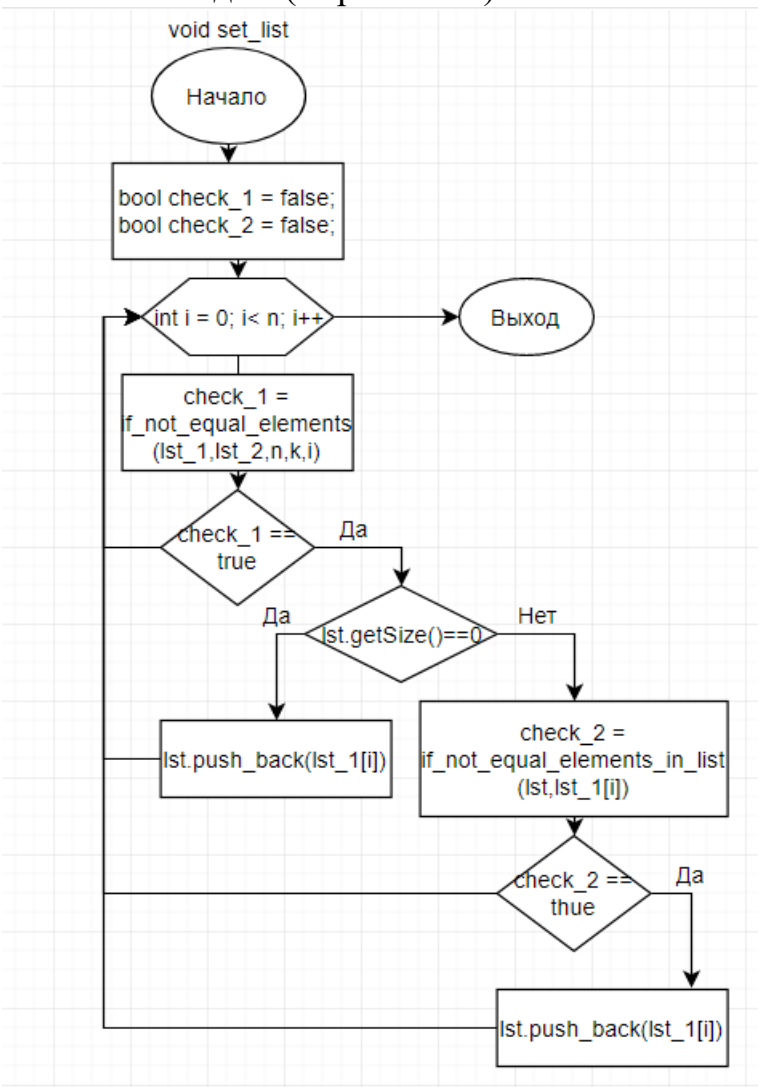

Рис. 5 Блок-схема функции, заполняющей список L

6) Вспомогательные функции для функции set_list:

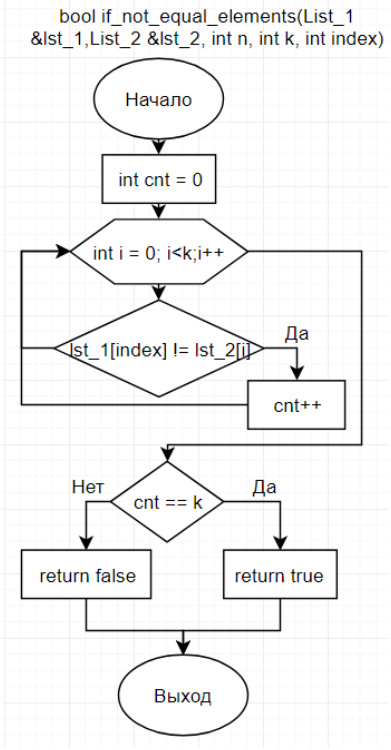

Рис. 6 Блок-схема функции, возвращающей true, если в списке L2 нет заданного элемента списка L1

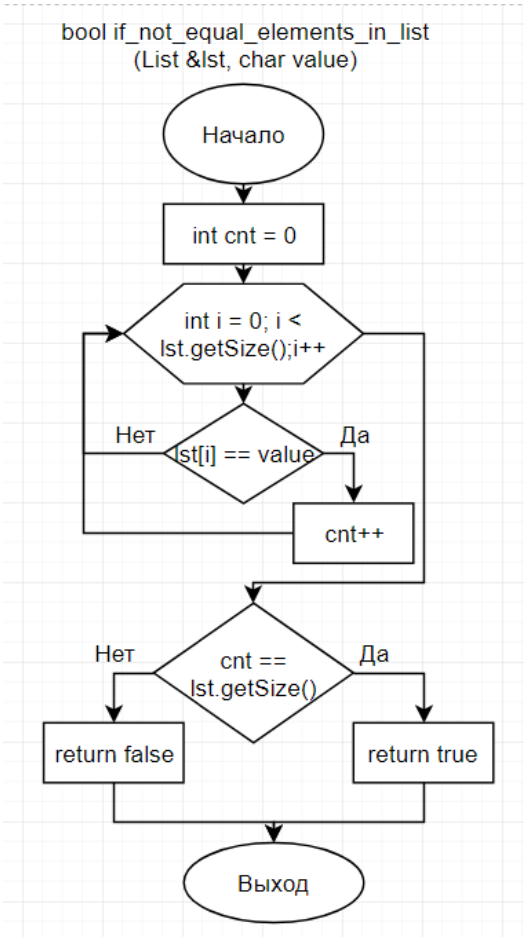

Рис. 7. Блок-схема функции, возвращающей true, если в списке L нет заданного значения value

Таблица 2. Результаты тестирования функции set_list

№Теста	Входные данные	Ожидаемый результат	Результат работы программы
1	n = 6 List_1 = a b c d d c k = 4 List_2 = b a e e	List = c d	Amount of elements List_1 = 6 a b c d d c Amount of elements List_2 = 4 b a e e Amount of elements List = 2 c d
2	n = 3 List_1 = a s d k = 3 List_2 = q w d	List = a s	Amount of elements List_1 = 3 a s d Amount of elements List_2 = 3 q w d Amount of elements List = 2 a s
3	n = 2 List_1 = a f k = 4 List_2 = b v c d	List = a f	Amount of elements List_1 = 2 a f Amount of elements List_2 = 4 b v c d Amount of elements List = 2 a f

7) Функция, выполняющая удаление подсписка из списка L1:

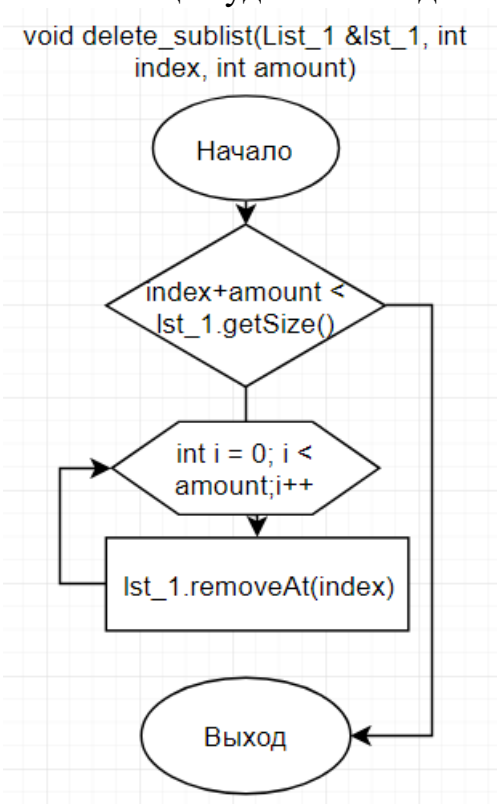

Рис. 8. Блок-схема функции, выполняющей удаление подсписка из списка L1

Таблица 3. Результаты тестирования функции delete_sublist

№Теста	Входные данные	Ожидаемый результат	Результат работы программы
1	n = 6 List_1 = a b c d e f index = 2 amount = 2	List_1 = a b e f	Amount of elements List_1 = 6 a b c d e f Index = 2 Amount = 2 a b e f
2	n = 3 List_1 = a s d index = 4 amount = 2	Wrong index or/and amount	Amount of elements List_1 = 3 a s d Index = 4 Amount = 2 Wrong index or/and amount
3	n = 4 List_1 = z x c v index = 0 amount = 1	List_ = x c v	Amount of elements List_1 = 4 z x c v Index = 0 Amount = 1 x c v

8) Функция, сортирующая список L2 по возрастанию:

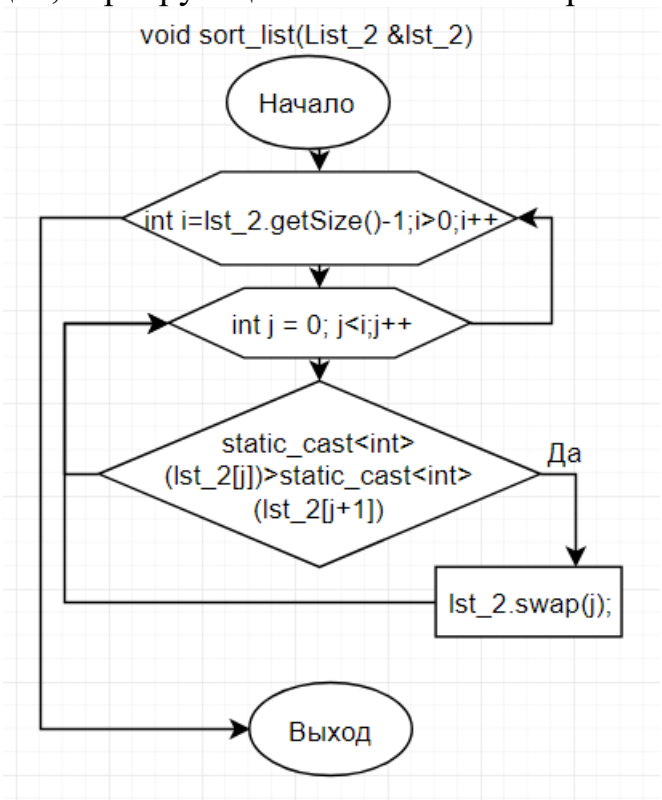

Рис. 9. Блок-схема функции, сортирующей элементы списка L2 по возрастанию

Таблица 4. Результаты тестирования функции sort_list

№Теста	Входные данные	Ожидаемый результат	Результат работы программы
1	n = 6 List_2 = a b c d e f	List_2 = a b c d e f	Amount of elements List_2 = 6 a b c d e f a b c d e f
2	n = 3 List_2 = a s d	List_2 = a d s	Amount of elements List_2 = 3 a s d a d s
3	n = 4 List_2 = z x c v	List_2 = c v x z	Amount of elements List_2 = 4 z x c v c v x z

1.3 Код программы

1) Код класса List(аналогичный код для классов List_1 и List_2):

```
class List
{
public:
    List();
    ~List();
    void push_back(char data);
    void removeAt(int index);
    void pop_front();
    char& operator[] (const int index);
    int getSize() { return Size; };
    void clear();
private:
    class Node
    {
    public:
        Node *pNext;
        char data;

        Node(char data = 0, Node *pNext = nullptr)
        {
            this->data = data;
            this->pNext = pNext;
        }
    };
    int Size;
    Node *head;
};
```

```
List::List()
{
    Size = 0;
    head = nullptr;
}

List::~List()
{
    clear();
}

void List::push_back(char data)
{
    if (head == nullptr)
    {
        head = new Node(data);
    }
    else
    {
        Node *current = this->head;
        while (current->pNext != nullptr)
        {
            current = current->pNext;
        }
        current->pNext = new Node(data);
    }
    Size++;
}
```

Рис. 10, 11 Реализация класса List

```cpp
void List::removeAt(int index)
{
    if (index == 0)
    {
        pop_front();
    }
    else
    {
        Node *previous = this->head;
        for (int i = 0; i < index - 1; i++)
        {
            previous = previous->pNext;
        }
        Node *toDelete = previous->pNext;
        previous->pNext = toDelete->pNext;
        delete toDelete;
        Size--;
    }
}

void List::pop_front()
{
    Node *temp = head;
    head = head->pNext;
    delete temp;
    Size--;
}
```

```cpp
char & List::operator[](const int index)
{
    int counter = 0;
    Node *current = this->head;
    while (current != nullptr)
    {
        if (counter == index)
        {
            return current->data;
        }
        current = current->pNext;
        counter++;
    }
}

void List::clear()
{
    while (Size != 0)
    {
        pop_front();
    }
}
```

Рис. 12, 13 Реализация класса List

2) Код функции set_list и вспомогательных функций if_not_equal_elements и if_not_equal_elements_in_list:

```cpp
void set_list(List_1 &lst_1, List_2 &lst_2,List &lst, int n, int k)
{
    bool check_1 = false;
    bool check_2 = false;
    for (int i = 0; i < n; i++)
    {
        check_1 = if_not_equal_elements(lst_1, lst_2, n, k, i);
        if (check_1 == true)
        {
            if (lst.getSize() == 0)
            {
                lst.push_back(lst_1[i]);
            }
            else
            {
                check_2 = if_not_equal_elements_in_list(lst, lst_1[i]);
                if (check_2 == true) lst.push_back(lst_1[i]);
            }
        }
    }
}
```

Рис. 14. Функция set_list

```
bool if_not_equal_elements(List_1 &lst_1, List_2 &lst_2,int n,int k, int index)
{
    int cnt = 0;

    for (int i = 0; i < k; i++)
    {
        if (lst_1[index] != lst_2[i]) cnt++;
    }

    if (cnt == k) return true;
    else return false;
}
```

Рис. 15. Функция if_not_equal_elements

```
bool if_not_equal_elements_in_list(List &lst, char value)
{
    int cnt = 0;
    for (int i = 0; i < lst.getSize(); i++)
    {
        if (lst[i] != value) cnt++;
    }
    if (cnt == lst.getSize()) return true;
    else return false;
}
```

Рис. 16. Функция if_not_equal_elements

3) Код функции delete_sublist:

```
void delete_sublist(List_1 &lst_1, int index, int amount)
{
    if ((index + amount) < lst_1.getSize())
    {
        for (int i = 0; i < amount; i++)
        {
            lst_1.removeAt(index);
        }
    }
    else cout << "Wrong index or/and amount" << endl;
}
```

Рис. 17. Функция delete_sublist

4) Код функции sort_list:

```
void sort_list(List_2 &lst_2)
{
    for (int i = lst_2.getSize() - 1; i > 0; i--)
    {
        for (int j = 0; j < i; j++)
        {
            if (static_cast<int>(lst_2[j]) > static_cast<int>(lst_2[j + 1]))
            {
                lst_2.swap(j);
            }
        }
    }
}
```

Рис. 18. Функция sort_list

2. Ответы на вопросы:

2.1 Три уровня представления данных в программе:

Существует три уровня представления данных: уровень пользователя (предметная область), логический и физический. Каждый объект предметной области характеризуется своими атрибутами, каждый атрибут имеет имя и значение.

2.2 Что определяет тип данных?

Тип данных однозначно определяет: внутреннее представление данных, а, следовательно, и диапазон их возможных значений; допустимые действия над данными (операции и функции). Например, целые и вещественные числа, даже если они занимают одинаковый объем памяти, имеют совершенно разный диапазон возможных значений; целые числа можно умножать друг на друга, а, к примеру, символы – нельзя.

2.3 Что определяет структура данных?

Под структурой данных программ в общем случае понимают множество элементов данных, множество связей между ними, а также характер их организованности.

2.4 Структура данных в компьютерных технологиях

Структура данных (англ. data structure) — программная единица, позволяющая хранить и обрабатывать множество однотипных и/или логически связанных данных в вычислительной технике. Для добавления, поиска, изменения и удаления данных структура данных предоставляет некоторый набор функций, составляющих её интерфейс.

2.5 Определение линейной структуры данных

Линейные структуры — это упорядоченные структуры, в которых адрес элемента однозначно определяется его номером. Линейных структуры данных обладают следующими свойствами: Каждый элемент имеет не более 1 предшественника Два разных элемента не могут иметь одинакового последователя

2.6 Определение линейного списка, как структуры данных

Линейный список — это динамическая структура данных, каждый элемент которой посредством указателя связывается со следующим элементом. Из определения следует, что каждый элемент списка содержит поле данных (Data) (оно может иметь сложную структуру) и поле ссылки на следующий элемент (next).

2.7 Определение стека, как структуры данных

Стек (от англ. stack — стопка) — структура данных, представляющая из себя упорядоченный набор элементов, в которой добавление новых элементов и удаление существующих производится с одного конца, называемого вершиной стека.

2.8 Определение очереди, как структуры данных

Очередью (англ. – queue) называется структура данных, из которой удаляется первым тот элемент, который был первым в очередь добавлен. То есть очередь в программировании соответствует «бытовому» понятию очереди. Очередь также называют структурой типа FIFO (first in, first out — первым пришел, первым ушел).

2.9 Отличие стека и линейного списка

Стек — это частный случай однонаправленного списка, добавление элементов в который и выборка из которого выполняются с одного конца, называемого вершиной стека. При выборке элемент исключается из стека.

2.10 Какой из видов линейных списков лучше использовать, если нужно введенную последовательность вывести наоборот

Для решения такой задачи удобнее использовать двунаправленный список, потому что в нем есть возможность пройти список как с начала, так и с конца.

2.11 Определение сложности алгоритма вставками элемента в i-ую позицию

Вставку элемента в i-ую позицию можно разделить на две операции: поиск i-го элемента и саму вставку. Для массива поиск элемента по индексу не составляет труда, алгоритмическая сложность составляет O(1). В случае односвязного списка придётся последовательно перебрать все элементы, пока не доберёмся до нужного элемента. Сложность будет O(n). Вставка в массив связана со сдвигом всех элементов, находящихся после точки вставки, поэтому алгоритмическая сложность этой операции O(n). В односвязном списке вставка заключается в создании нового связующего объекта и установки ссылок на него у соседних элементов. Сложность O(1). В сумме сложность вставки i-го элемента у массива и у списка получается одинаковая — O(n). Но поскольку операция чтения по сути быстрее операции записи, односвязный список работает быстрее.

2.12 Определение сложности алгоритма удаления элемента из i-ой позиции

Удаление элемента из i-ой позиции можно разделить на две операции: поиск i-го элемента и удаление. Для массива поиск элемента по индексу не составляет труда, алгоритмическая сложность составляет O(1). В случае односвязного списка придётся последовательно перебрать все элементы, пока не доберёмся до нужного элемента. Сложность будет O(n). Удаление из массива связана со сдвигом всех элементов, находящихся после точки вставки, поэтому алгоритмическая сложность
этой операции O(n). В односвязном списке удаление заключается в переустановке ссылок соседних элементов. Сложность O(1). В сумме сложность удаление i-го элемента у массива и у списка получается одинаковая — O(n). Но поскольку операция чтения по сути быстрее операции записи, односвязный список работает быстрее.

2.13 Трюк Вирта при выполнении операции удаления элемента из списка

Косвенный указатель p даёт два концептуальных преимущества:

1. Позволяет интерпретировать связный список таким образом, что указатель head становится неотъемлемой частью структуры данных. Это устраняет необходимость в специальном случае для удаления первого элемента.
2. Также позволяет оценить состояние цикла while без необходимости отпускать указатель, указывающий на target. Это позволяет изменять указатель на target и обходиться одним итератором, в отличие от prev и cur.

2.14 Определение структуры узла однонаправленного списка

Структуру узла однонаправленного списка можно представить в виде класса:

```
class Node
{
public:
    Node *pNext;
    char data;

    Node(char data = 0, Node *pNext = nullptr)
    {
        this->data = data;
        this->pNext = pNext;
    }
};
```

Рис. 19 Структура узла списка

2.15 Алгоритм вывода линейного однонаправленного списка

Для того, чтобы вывести линейный список необходимо переопределить оператор, а затем написать функцию вывода:

```
char & List::operator[](const int index)
{
    int counter = 0;
    Node *current = this->head;
    while (current != nullptr)
    {
        if (counter == index)
        {
            return current->data;
        }
        current = current->pNext;
        counter++;
    }
}
```

Рис. 20 Переопределенный оператор для списка

```
for (int i = 0; i < lst.getSize(); i++)
{
    cout << lst[i] << " ";
}
```

Рис. 21 Функция вывода списка

2.16 Перемещение последнего элемента в начало списка

```
void List::last_to_first(Node **head)
{
    Node *node = *head;
    while (node->pNext->pNext != nullptr)
    {
        node = node->pNext;
    }
    Node *lastNode = node->pNext;
    node->pNext = *head;
    lastNode->pNext = nullptr;
    *head = lastNode;
}
```

Рис. 22 Метод перемещения последнего элемента в начало списка

2.17 Какое из действий лишнее в следующем фрагменте кода? Куда вставляется новый узел

Лишней является строчка, содержащая if (LL == nullptr) LL->next = q; т.к. по ней – если текущей указатель нулевой, то следующим за ним вставляется новый узел. Получается ошибка, т.к. мы обращаемся к неинициализированному участку памяти.

ВЫВОДЫ

В ходе данной практической работы разработали структуру данных – однонаправленный динамический список, получили знания и практические навыки управления динамическим однонаправленным списком. Разработали обязательные функции по управлению списком и функции согласно варианту индивидуального задания. Провели тестирование функций, результат тестирования подтвердил правильность работы функций.

ИНФОРМАЦИОННЫЕ ИСТОЧНИКИ

1. Лекционный материал по структурам и алгоритмам обработки данных Гданского Н.И.
2. Односвязный список // Википедия [Электронный ресурс]. https://ru.wikipedia.org/wiki/%D0%94%D0%B8%D0%BD%D0%B0%D0%BC%D0%B8%D1%87%D0%B5%D1%81%D0%BA%D0%B8%D0%B9_%D0%BC%D0%B0%D1%81%D1%81%D0%B8%D0%B2. – (дата обращения: 12.04.2021)
3. Трюк Вирта // Википедия [Электронный ресурс]. - https://ru.wikipedia.org/wiki/%D0%92%D1%8B%D1%87%D0%B8%D1%81%D0%BB%D0%B8%D1%82%D0%B5%D0%BB%D1%8C%D0%BD%D0%B0%D1%8F_%D1%81%D0%BB%D0%BE%D0%B6%D0%BD%D0%BE%D1%81%D1%82%D1%8C. –(дата обращения: 12.04.2021)