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 Есаев С. А.

группа ИКБО-01-20

Москва 2021

Содержание

1. Ответы на вопросы

1.1. Три уровня представления данных в программной системе

Существует три уровня представления данных: уровень пользователя (предметная область), логический и физический.

1.2. Что определяет тип данных?

Тип определяет возможные значения и их смысл, операции, а также способы хранения значений типа.

1.3. Что определяет структура данных?

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

1.4. Структуры данных в компьютерных технологиях?

Структура данных — 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.

1.5. Определение линейной структуры данных

Линейные структуры — это упорядоченные структуры, в которых адрес элемента однозначно определяется его номером.

1.6. Определение линейного списка, как структуры данных

Линейный однонаправленный список — это структура данных, состоящая из элементов одного типа, связанных между собой последовательно посредством

указателей. Каждый элемент списка имеет указатель на следующий элемент. Последний элемент списка указывает на NULL. Элемент, на который нет указателя, является первым (головным) элементом списка. Здесь ссылка в каждом узле указывает на следующий узел в списке. В односвязном списке можно передвигаться только в сторону конца списка. Узнать адрес предыдущего элемента, опираясь на содержимое текущего узла, невозможно.

1.7. Определение стека, как структуры данных

Логически стек представляет собой последовательность элементов с переменной длиной; включение и исключение элементов из последовательности происходит при этом только с одной стороны, называемой вершиной стека.

1.8. Определение очереди, как структуры данных

Очередь — это структура данных, добавление и удаление элементов в которой происходит путём операций и соответственно. Притом первым из очереди удаляется элемент, который был помещен туда первым, то есть в очереди реализуется принцип «первым вошел — первым вышел»

1.9. Отличие стека и линейного списка

Стек – это линейный список, в котором добавление новых элементов и удаление существующих производится только с одного конца, называемого вершиной стека.

1.10. 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.

1.11. Определение сложности алгоритма операции вставки 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 спика 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.

1.12. Определение сложности алгоритма операции 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 спика 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.

1.13. Трюк Вирта при выполнении операции удаления элемента из списка

```
void remove_elegant(IntList* l, IntListItem* target)
{
    IntListItem** p = &l->head;
    while ((*p) != target) {
        p = &(*p)->next;
    }
    *p = target->next;
}
```

В коде применяется косвенный указатель p, содержащий адрес указателя на элемент списка, начиная с адреса head. В каждой итерации этот указатель расширяется, чтобы включить адрес указателя на следующий элемент списка, то есть адрес элемента next в текущем IntListItem. Когда указатель на элемент списка (*p) равен target, мы выходим из цикла поиска и удаляем элемент из списка.

Косвенный указатель p позволяет интерпретировать связный список таким образом, что указатель head становится неотъемлемой частью структуры данных. Это устраняет необходимость в специальном случае для удаления первого элемента, а также данный указатель 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.

1.14. Определение структуры узла однонаправленного списка

Узел ОЛС можно представить в виде структуры

```
struct Node {
    int data;  // поле данных
    Node* nextNode;// указатель на следующий узел
    Node() {};
    Node(int data, Node* nextNode = nullptr):
        data(data),
        nextNode(nextNode) {}
```

```
};
```

1.15. Алгоритм вывода линейного однонаправленного списка

```
void showList(Node* head) {
    Node* node = head;
    while (node != nullptr)
    {
        std::cout << node->data;
        node = node->nextNode;
    }
}
```

Пока текущей элемент не является указателем на nullptr мы продолжаем переходить от элемента к элементу и осуществлять вывод данных узла списка.

1.16. Перемещение последнего элемента в начало списка

```
void lastToFirst(Node** head)
{
    Node* node = *head;
    while (node->nextNode->nextNode != nullptr) // доходим до предпоследнего элемента
    {
        node = node->nextNode;
    }
    Node* lastNode = node->nextNode; // запоминаем последний элемент
    node->nextNode = *head; // теперь первый идет после предпоследнего
    lastNode->nextNode = (*head)->nextNode; // последний предшествует второму
    (*head)->nextNode = nullptr;  // после первого только  nullptr
    *head = lastNode; // теперь последний элемент стал первым

}
```

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

```
void insertToList(List LL, int x) {
    List q = new Node2; q->info = x; q->next = 0;
    if (LL == nullptr) LL->next = q;
    else
        q->next = LL->next;
    LL->next = q;
}
```

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.

2. Отчет по разработанной программе

Условие задания, требования в соответствии с вариантом

№ варианта	Тип информационной части узла	Дополнительные операции
7	Int	Дан линейный однонаправленный список L 1) Разработать функцию, которая проверяет, есть ли в списке L два одинаковых элемента. 2) Разработать функцию, которая удаляет из списка L максимальное значение. 3) Разработать функцию, которая вставляет в список L новое значение перед каждым узлом в четной позиции.

2.1. Постановка задачи

Реализовать программу решения задачи варианта по использованию линейного однонаправленного списка. Разработать структуру данных, на основе которой можно реализовать линейный однонаправленный список, и функции для работы с этой структурой:

Обязательные

1) Функция вставки нового узла перед первым узлом

2) Функция создания исходного списка

3) Функция вывода списка

Функции дополнительного задания варианта

4) Функция проверки списка на наличие одинаковых элементов

5) Функция удаления узла с максимальным значением

6) Функция вставки в список L нового значения перед каждым четным узлом

2.2. Определение операций над списком

2.2.1 Определение структуры узла в соответствии с вариантом

Для решения поставленной задачи разработана следующая структура данных, на основе которой можно реализовать однонаправленный список:

Структура Node

- Свойства/поля:
  - Поле, отвечающее за хранение значений:
    - Наименование – data;
    - Тип – целочисленный;
    - Модификатор доступа – открытый;
  - Поле, отвечающее за хранение адреса следующего узла:
    - Наименование – nextNode;
    - Тип – указатель на структуру Node;
    - Модификатор доступа – открытый;
- Методы:
  - Конструктор структуры данных, определяющий значения свойств структуры, по умолчанию устанавливает адрес следующего узла как nullptr.

2.2.2 Иллюстрация процесса операций над списком

1) Функция вставки нового узла перед первым узлом

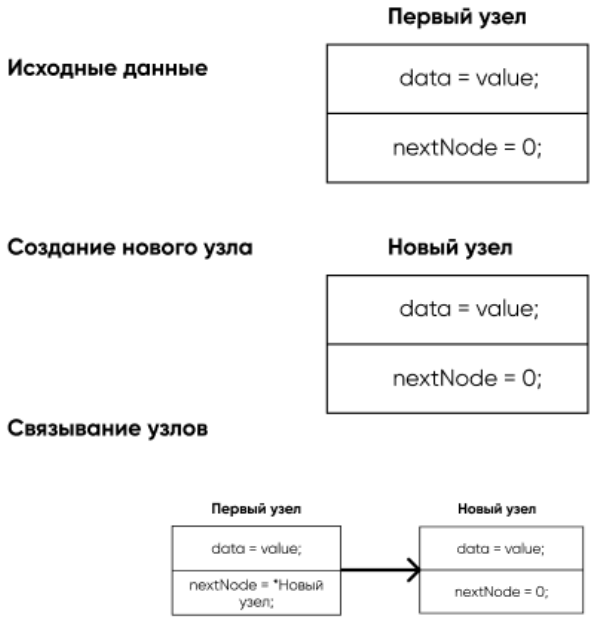

Рис. 1. Иллюстрация функции вставки

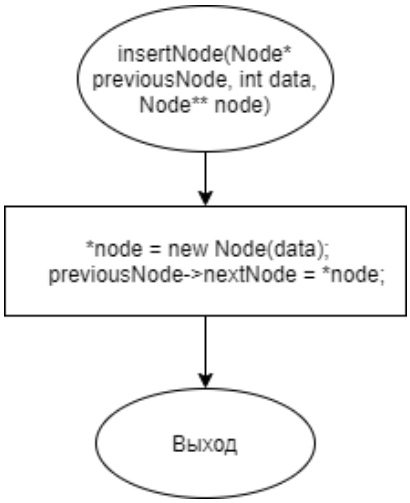

Рис. 2. Алгоритм функции вставки

2) Функция создания исходного списка

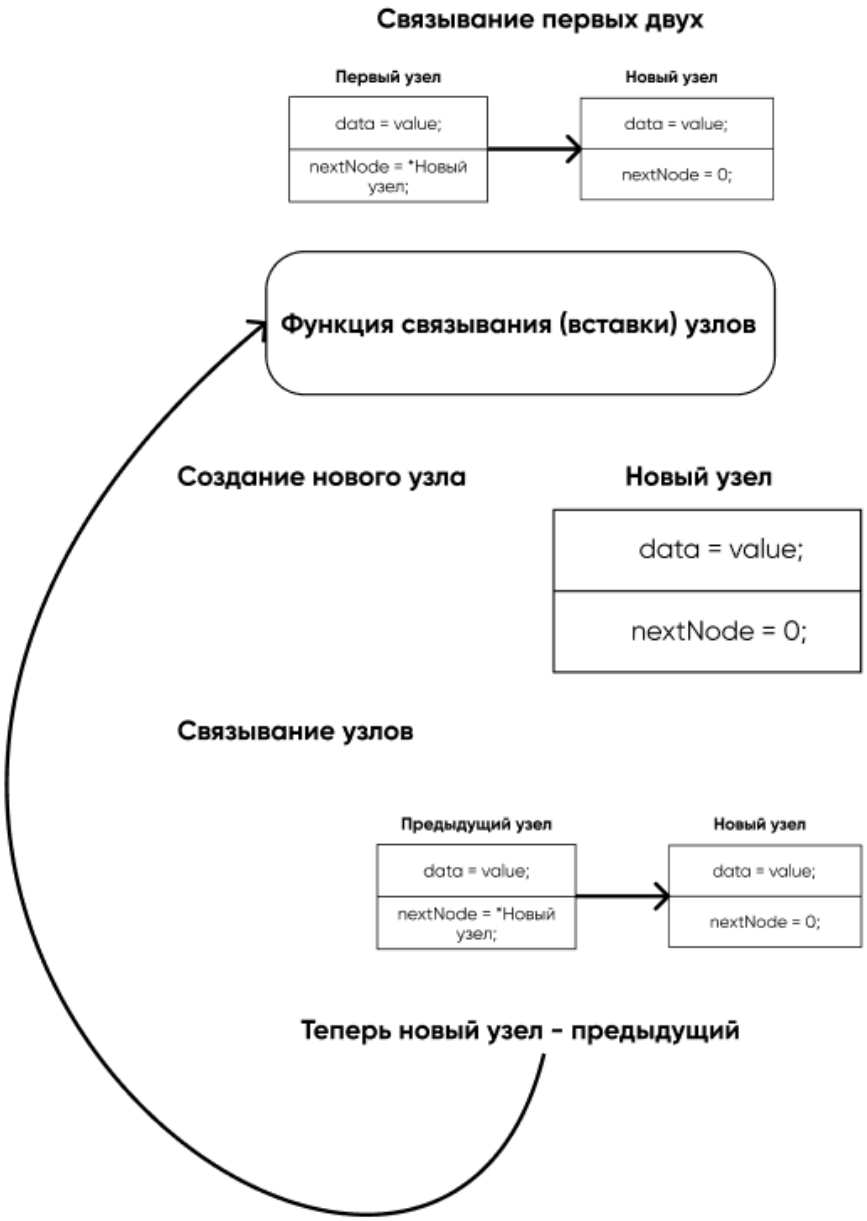

Рис. 3. Иллюстрация создания списка

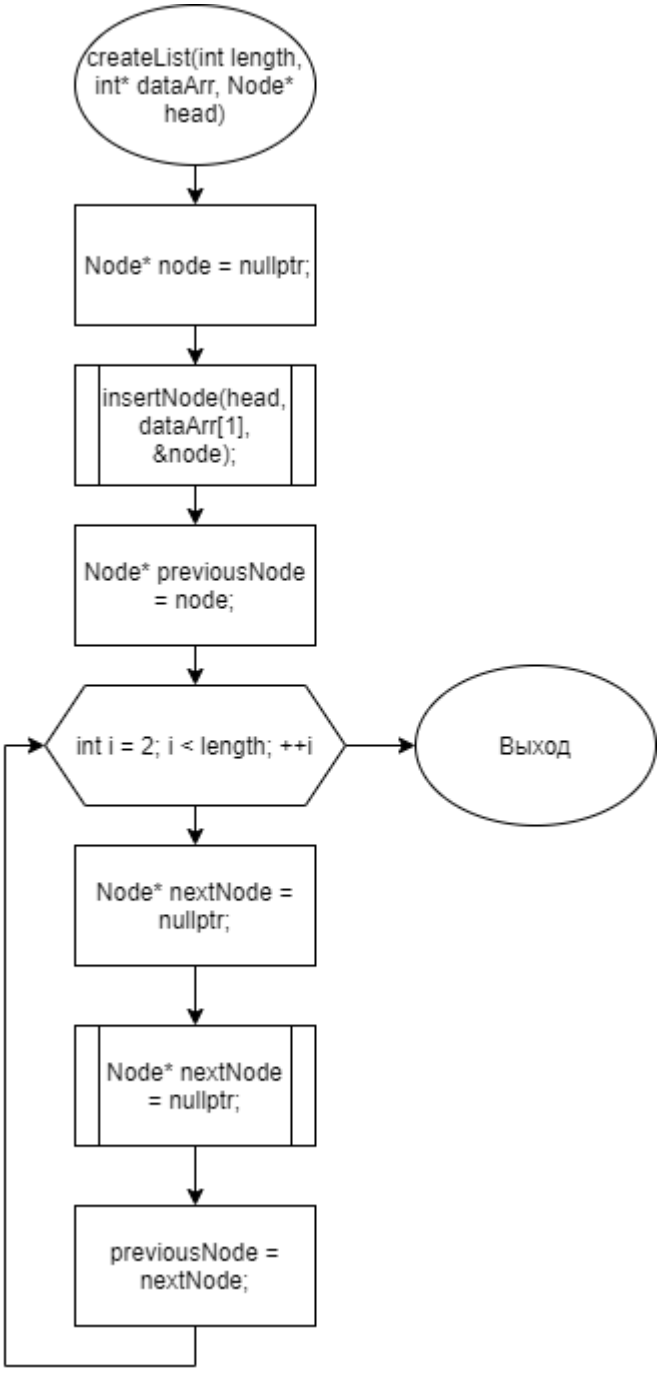

Рис. 4. Алгоритм функции создания списка

3) Функция вывода списка

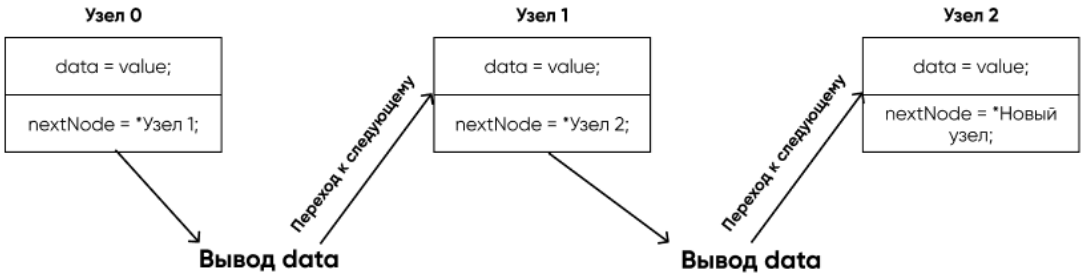

Рис. 5. Иллюстрация вывода списка

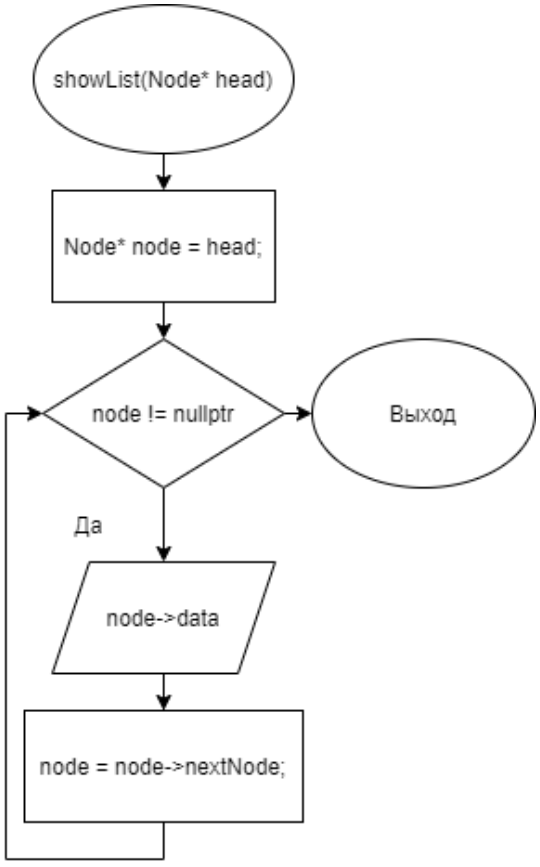

Рис. 6. Алгоритм вывода списка

Тестировать обязательные функции удобнее вместе, т.к. при создании и выводе нового списка используется каждая из обязательных функций. Для тестирования композиции этих функций была разработана следующая таблица тестов.

Таблица 1. Тестирование обязательных функций

Номер теста	Входные данные	Ожидаемый результат	Вывод
1	5 1 2 3 4 5	1 2 3 4 5	Length of list = 5 Data of nodes: 1 2 3 4 5 1 2 3 4 5
2	12 7 8 9 10 11 12 3 4 5 1 2 5	7 8 9 10 11 12 3 4 5 1 2 5	Length of list = 12 Data of nodes: 7 8 9 10 11 12 3 4 5 1 2 5 7 8 9 10 11 12 3 4 5 1 2 5
3	8 8 7 6 5 4 3 2 1	8 7 6 5 4 3 2 1	Length of list = 8 Data of nodes: 8 7 6 5 4 3 2 1 8 7 6 5 4 3 2 1

4) Функция поиска двух одинаковых

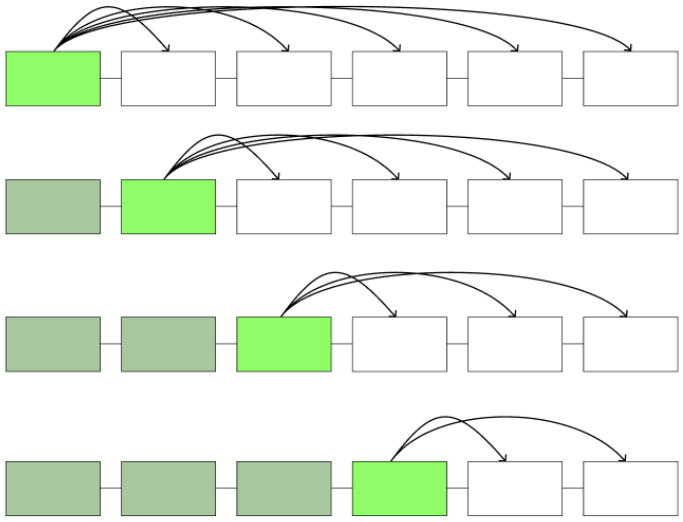

Рис. 7. Иллюстрация поиска двух одинаковых

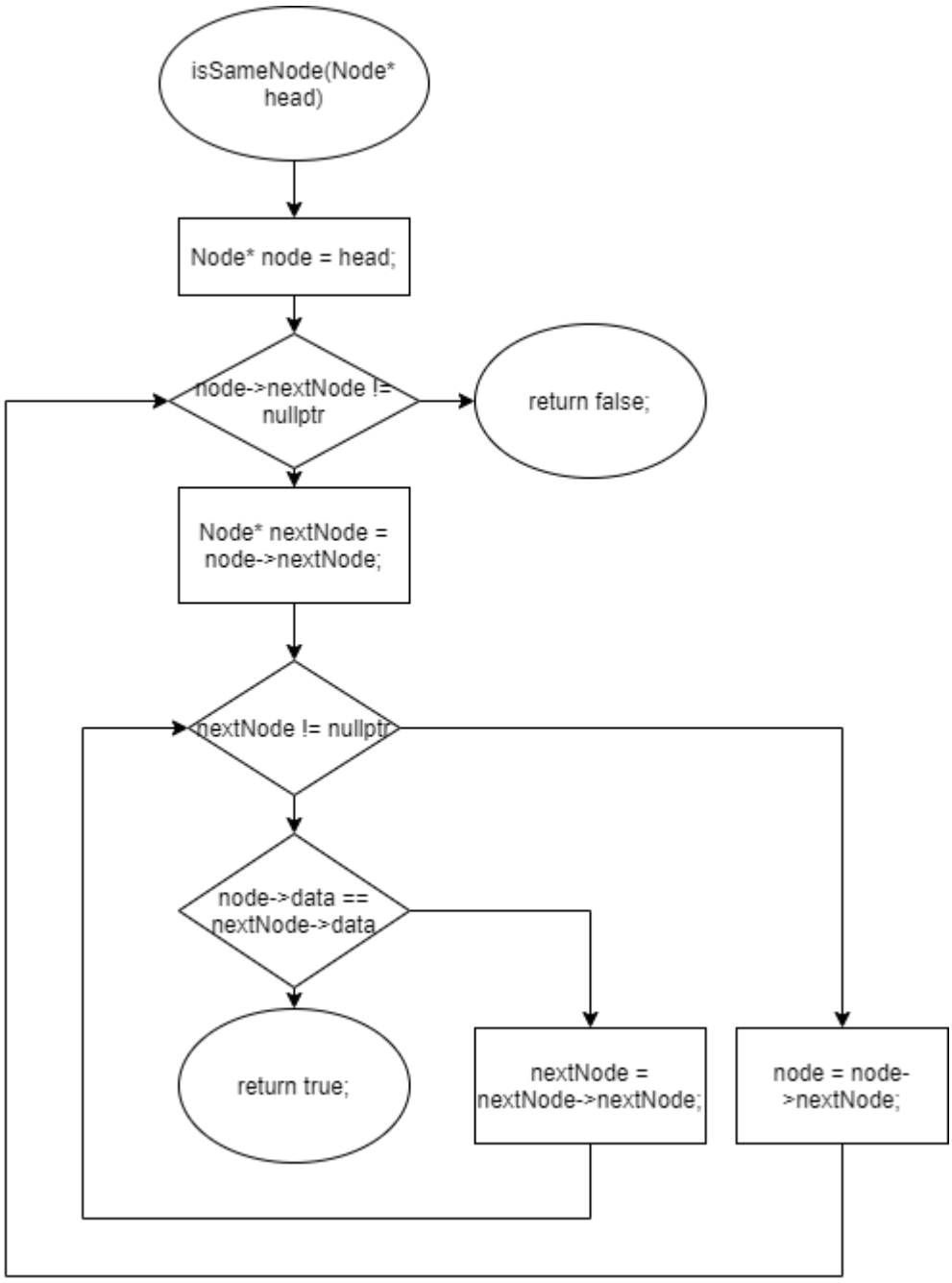

Рис. 8. Алгоритм поиска двух одинаковых значений

Таблица 2. Тестирование функции поиска двух одинаковых

Номер теста	Входные данные	Ожидаемый результат	Вывод
1	5 1 2 3 4 5	0	Length of list = 5 Data of nodes: 1 2 3 4 5 0
2	3 1 2 2	1	Length of list = 3 Data of nodes: 1 2 2 1
3	5 1 2 3 3 5	1	Length of list = 5 Data of nodes: 1 2 3 3 5 1

5) Удаление максимального значения

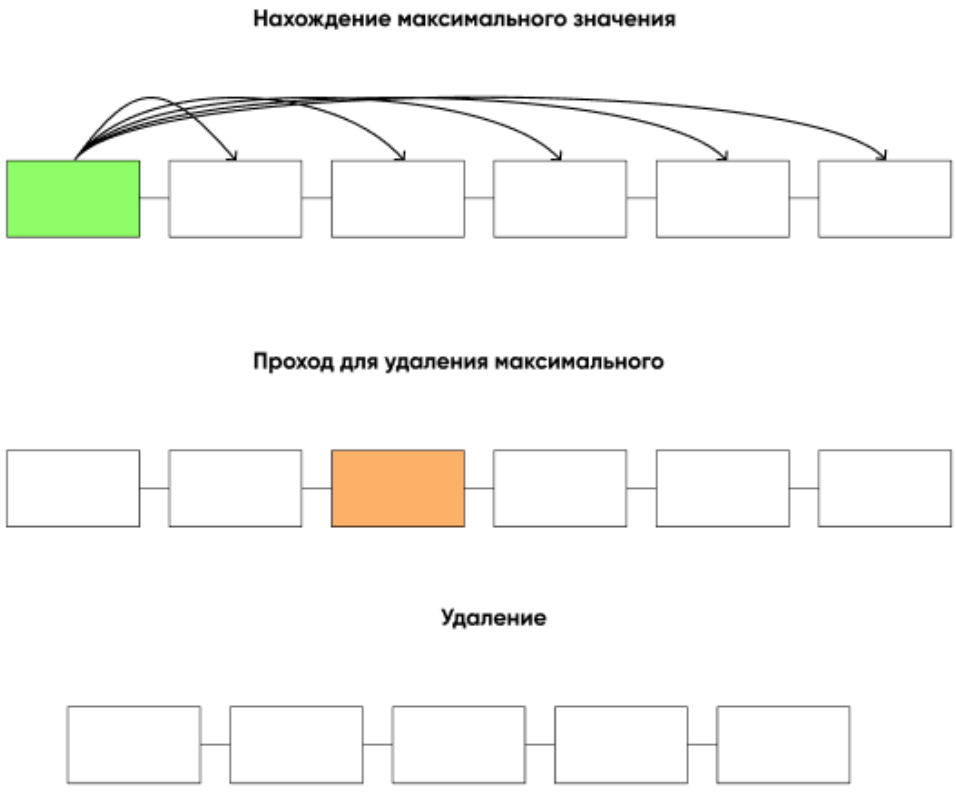

Рис. 9. Иллюстрация удаления максимального элемента

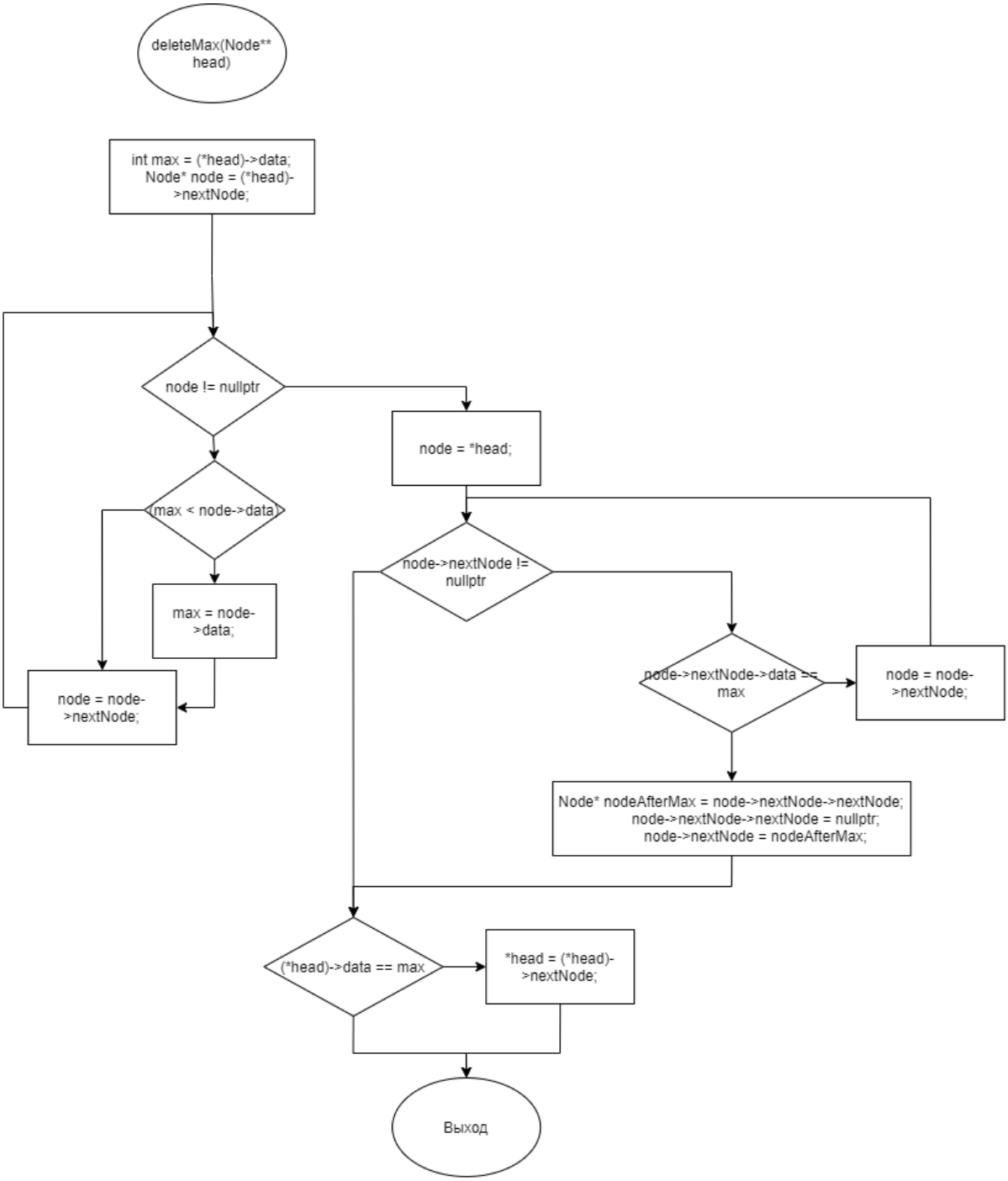

Рис. 10. Алгоритм удаления максимального элемента

Таблица 3. Тестирование функции удаления максимального элемента

Номер теста	Входные данные	Ожидаемый результат	Вывод
1	5 1 2 3 4 5	1 2 3 4	Length of list = 5 Data of nodes: 1 2 3 4 5 1 2 3 4
2	3 1 2 2	1 2	Length of list = 3 Data of nodes: 1 2 2 1 2
3	5 1 2 3 3 5	1 2 3 3	Length of list = 5 Data of nodes: 1 2 3 3 5 1 2 3 3

6) Функция вставки нового элемента перед четным узлом

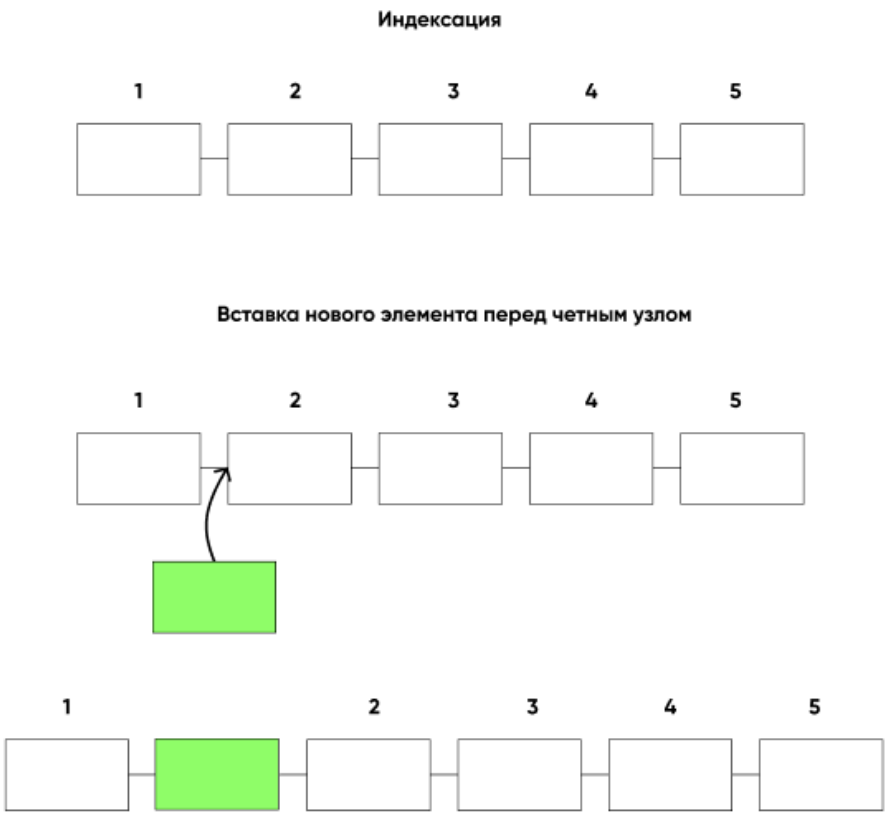

Рис. 11. Иллюстрация вставки нового элемента перед четным узлом

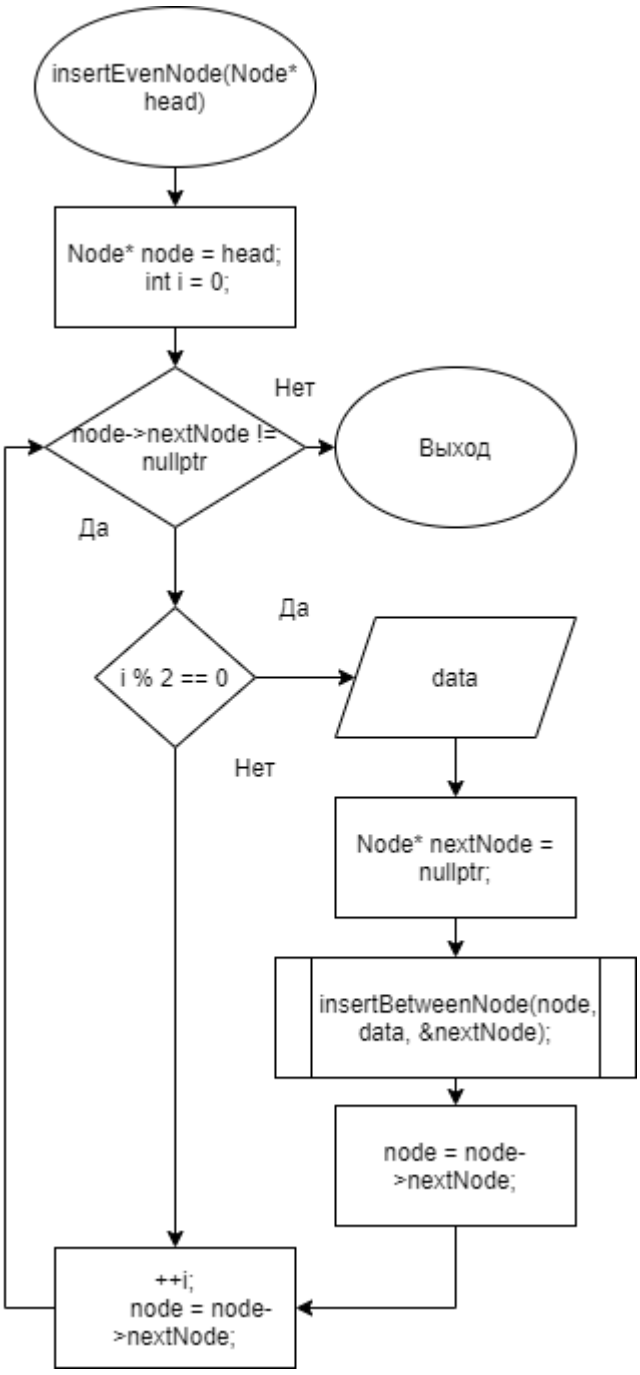

Рис. 12. Алгоритм вставки нового элемента перед четным узлом

Таблица 4. Тестирование вставки нового элемента перед четным узлом

Номер теста	Входные данные	Ожидаемый результат	Вывод
1	5 1 2 3 4 5	1 -1 2 -1 3 -1 4 5	Length of list = 5 Data of nodes: 1 2 3 4 5 1 -1 2 3 -1 4 5
2	3 1 2 2	1 -1 2 2	Length of list = 3 Data of nodes: 1 2 2 1 -1 2 2
3	8 0 0 0 0 0 0 0 0	0 -1 0 0 -1 0 0 -1 0 0 -1 0	Length of list = 8 Data of nodes: 0 0 0 0 0 0 0 0 0 -1 0 0 -1 0 0 -1 0 0 -1 0

2.3. Код программы

```
#include <iostream>
struct Node {
    int data;
    Node* nextNode;
    Node() {};
    Node(int data, Node* nextNode = nullptr):
        data(data),
        nextNode(nextNode) {}
};


void showList(Node* head) {
    Node* node = head;
    while (node != nullptr)
    {
        std::cout << node->data;
        node = node->nextNode;
    }
}

void insertNode(Node* previousNode, int data, Node** node) {
    *node = new Node(data);
    previousNode->nextNode = *node;
}

void createList(int length, int* dataArr, Node* head) {
    Node* node = nullptr;
    insertNode(head, dataArr[1], &node);
    Node* previousNode = node;
```

```
    for (int i = 2; i < length; ++i)
    {
        Node* nextNode = nullptr;
        insertNode(previousNode, dataArr[i], &nextNode);
        previousNode = nextNode;
    }
}



void insertBetweenNode(Node* previousNode, int data, Node** node) {
    *node = new Node(data, previousNode->nextNode);
    previousNode->nextNode = *node;
}

void insertEvenNode(Node* head) {
    Node* node = head;
    int i = 0;
    while (node != nullptr) {
        if (i % 2 == 0)
        {
            int data = -1;
            //std::cin >> data;
            Node* nextNode = nullptr;
            insertBetweenNode(node, data, &nextNode);
        }
        ++i;
        node = node->nextNode;
    }

}



void deleteMax(Node** head) {
    int max = (*head)->data;
    Node* node = (*head)->nextNode;

    while (node != nullptr)
    {
        if (max < node->data) max = node->data;
        node = node->nextNode;
    }

    node = *head;
    while (node->nextNode != nullptr)
    {
        if (node->nextNode->data == max)
        {
            Node* nodeAfterMax = node->nextNode->nextNode;
            node->nextNode->nextNode = nullptr;
            node->nextNode = nodeAfterMax;

            break;
        }
        node = node->nextNode;
    }
```

```
    if ((*head)->data == max) *head = (*head)->nextNode;

}


bool isSameNode(Node* head) {
    Node* node = head;
    while (node->nextNode != nullptr)
    {
        Node* nextNode = node->nextNode;
        while (nextNode != nullptr)
        {
            if (node->data == nextNode->data) {
                std::cout << node->data;
                return true;
            }

            nextNode = nextNode->nextNode;
        }
        node = node->nextNode;
    }

    return false;
}



int main()
{
    int length;
    std::cout << "Length of list = ";
    length = 10;
    //std::cin >> length;
    std::cout << "Data of nodes: ";
    int* dataArr = new int[length] {9, 5, 7, 8, 1, 6, 4, 3, 2, 0};
    /*for (int i = 0; i < length; ++i)
    {
        std::cin >> dataArr[i];
    }*/
    Node* head = new Node(dataArr[0]);
    createList(length, dataArr, head);
    showList(head);
    isSameNode(head);
    insertEvenNode(head);
    showList(head);
    deleteMax(&head);
    showList(head);


}
```

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.

СПИСОК ИНФОРМАЦИОННЫХ ИСТОЧНИКОВ

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)

4. Прата, Стивен Язык программирования C++. Лекции и упражнения [Текст] / Стивен Прата. - М.: Вильямс, 2015. - 445 с.