МИНОБРНАУКИ РОССИИ
Федеральное государственное бюджетное образовательное учреждение
высшего образования
«МИРЭА – Российский технологический университет»
РТУ МИРЭА

Институт информационных технологий (ИИТ)
Кафедра математического обеспечения и стандартизации информационных технологий (МОСИТ)

Отчет по выполнению индивидуального задания 2

дисциплина СиАОД

Тема. Линейный двунаправленный список. Операции над списком

Выполнил

Студент Хан А.А.

Группа ИКБО-29-20

Москва 2021

СОДЕРЖАНИЕ

1.Условие задания, требования в соответствии с вариантом. 3
2. Определение списка операций над списком, которые выявлены впроцессе исследования задач
3. Структура узла двунаправленного списка в соответствии с вариантом....3
4. Отчёт по каждой подзадаче....4
4.1. Функция вывода списка в двух направлениях, print....4
4.2. Функция поиска узла с данным значением номера полиса, find_by_policy....5
4.3. Функция вставки в начало списка заболеваний пациента нового узла, insert_disease....5
4.4. Функция формирования нового списка с пациентами, у которых более 5-ти заболеваний, over_five_diseases....7
4.5. Функция удаления болезни с данным кодом у данного пациента, remove_disease 8
5. Оценка сложности первого дополнительного алгоритма 12
6. Выводы по полученным знаниям умениям 13
7. Список информационных источников 13

Отчет по разработанной программе

1. Условие задания, требования в соответствии с вариантом.

Разработать многомодульную программу, которая демонстрирует выполнение всех операций, определенных вариантом, над линейным двунаправленным динамическим списком. Реализовать функции вывода списка в двух направлениях и поиска узла с заданным значением. Протестировать работу программы.

Дополнительные задания варианта: вариант 13 (29 % 17 +1 = 13)

1) Структура узла: Номер мед. полиса, указатель на линейный список (узлы которого содержат: Код заболевания и Дата регистрации заболевания зарегистрировано), фамилия больного
2) Разработать функцию вставки в список заболеваний пациента нового узла
3) Разработать функцию, формирующую новый список, включив в него информацию о пациентах, у которых зарегистрировано более 5 заболеваний
4) Разработать функцию удаления из списка кодов заболеваний указанного пациента, узла с указанным кодом

2. Определение списка операций над списком, которые выявлены в процессе исследования задач.

1) Функция вывода списка в двух направлениях
2) Функция поиска узла с данным значением номера полиса
3) Функция вставки в начало списка заболеваний пациента нового узла
4) Функция формирования нового списка с пациентами, у которых более 5-ти заболеваний
5) Функция удаления болезни с данным кодом у данного пациента

3. Структура узла двунаправленного списка в соответствии с вариантом.

```
struct Diseases {
    string code;
    tm* date;
    Diseases* next;
};

struct Node {
    string policy;
    string name;
    Diseases* diseases;
    Node* next;
    Node* prev;
};
```

Поле policy является полем ключа, поле date типа tm* является полем даты структуры, определённой в модуле <ctime>. Поле prev для головного узла указывает на последний узел списка.

4. Отчёт по каждой подзадаче

4.1. Функция вывода списка в двух направлениях, print

4.1.1. Реализовать функцию, выводящую список, значения поля ключа в консоль двумя разными способами, слева направо и справа налево

4.1.2. Если параметром указан режим работы вывода слева направо, то проходимся по списку слева направо с помощью вспомогательного указателя, выводя поле ключа каждого встретившегося элемента. Если же параметром указан режим работы вывода справа налево, то действия аналогичны, лишь проход по списку осуществляется в обратном направлении (сделать это возможно благодаря тому, что предыдущий элемент для головного – есть последний элемент списка)

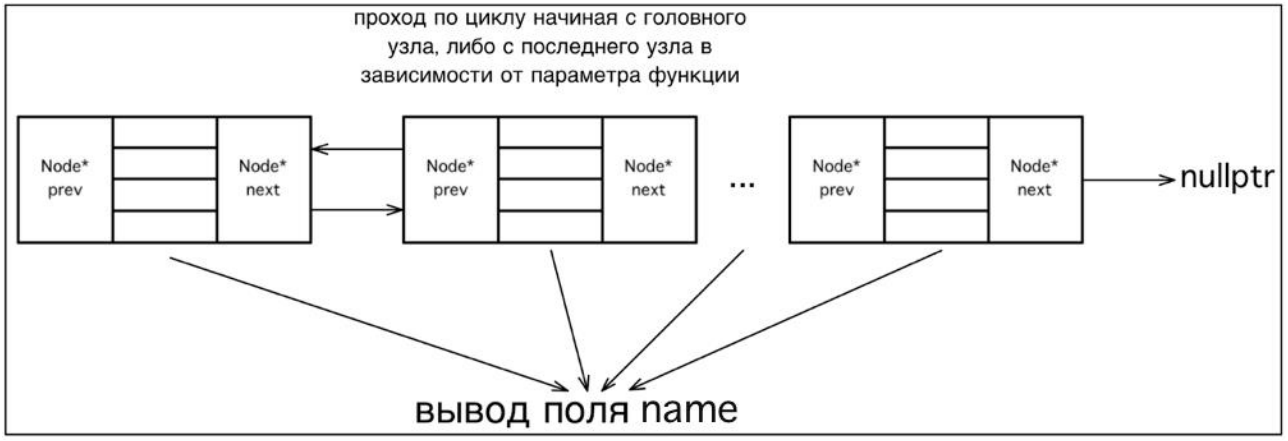

Рис. 1 – Функция вывода списка в двух направлениях

Таблица тестов

№	Исходный список	Выходные данные
1	“0001” “Иванов”, “0002” “Петров”, “0003”, “Сидоров”, “0004” “Калинин”, “0005” “Орлов”	0005 Орлов 0004 Калинин 0003 Сидоров 0002 Петров 0001 Иванов 0001 Иванов 0002 Петров 0003 Сидоров 0004 Калинин 0005 Орлов

4.2. Функция поиска узла с данным значением номера полиса, find_by_policy

4.2.1. Реализовать функцию поиска узла с данным значением поля ключа

4.2.2. Объявить вспомогательный указатель, указывающий на начало списка. Продвигаясь по элементам, пока список не кончится, либо пока не будет достигнут узел с искомым значением policy. Возвращаем итоговое значение указателя

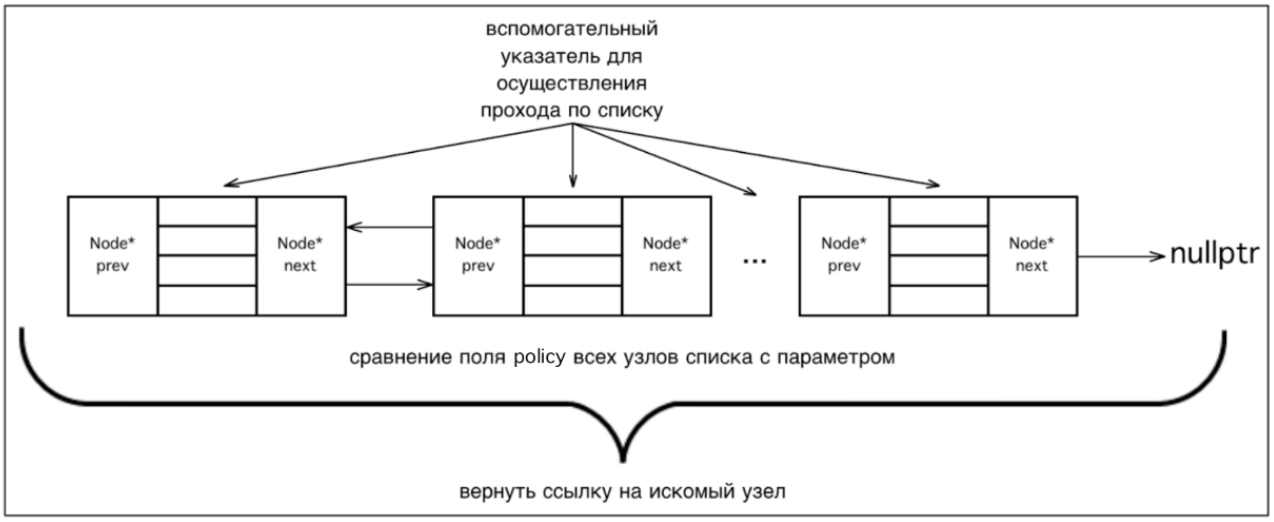

Рис. 2 – Функция поиска узла с данным значением номера полиса

Таблица тестов

№	Исходный список	Входные данные	Выходные данные
1	0001 Иванов 0002 Петров 0003 Сидоров 0004 Калинин 0005 Орлов	0002	0002 Петров

4.3. Функция вставки в начало списка заболеваний пациента нового узла, insert_disease

4.3.1. Реализовать функцию, вставляющую новый узел в список заболеваний данного пациента

4.3.2. Объявить указатель на новый узел заболеваний Diseases*. Инициализировать его, обозначив поля date, code. Вставить новый узел в уже имеющийся список заболеваний пациента

Рис. 3 – Функция вставки в начало списка заболеваний пациента нового узла

Таблица тестов

№	Исходный список	Входные данные	Выходные данные
1	CANCER 15.10.2019 SARS 5.1.2021 POX 16.2.2021 FLU 30.3.2021 ARI 10.2.2021	COVID19 28.4.2021	COVID19 28.4.2021 CANCER 15.10.2019 SARS 5.1.2021 POX 16.2.2021 FLU 30.3.2021 ARI 10.2.2021

4.4. Функция формирования нового списка с пациентами, у которых более 5-ти заболеваний, over_five_diseases

4.4.1. Реализовать функцию, формирующую новый список с пациентами, у которых больше 5-ти заболеваний

4.4.2. Инициализируем новый пустой список. Инициализируем вспомогательный указатель типа Node* который указывает на начало исходного списка. Проходясь по каждому элементу, также проходимся по списку заболеваний и с помощью счётчика оцениваем количество узлов в списке заболеваний, если их больше 5-ти, то добавляем данный узел исходного списка в новый список, с помощью копирования данных и вызова вспомогательной функции copy_and_insert

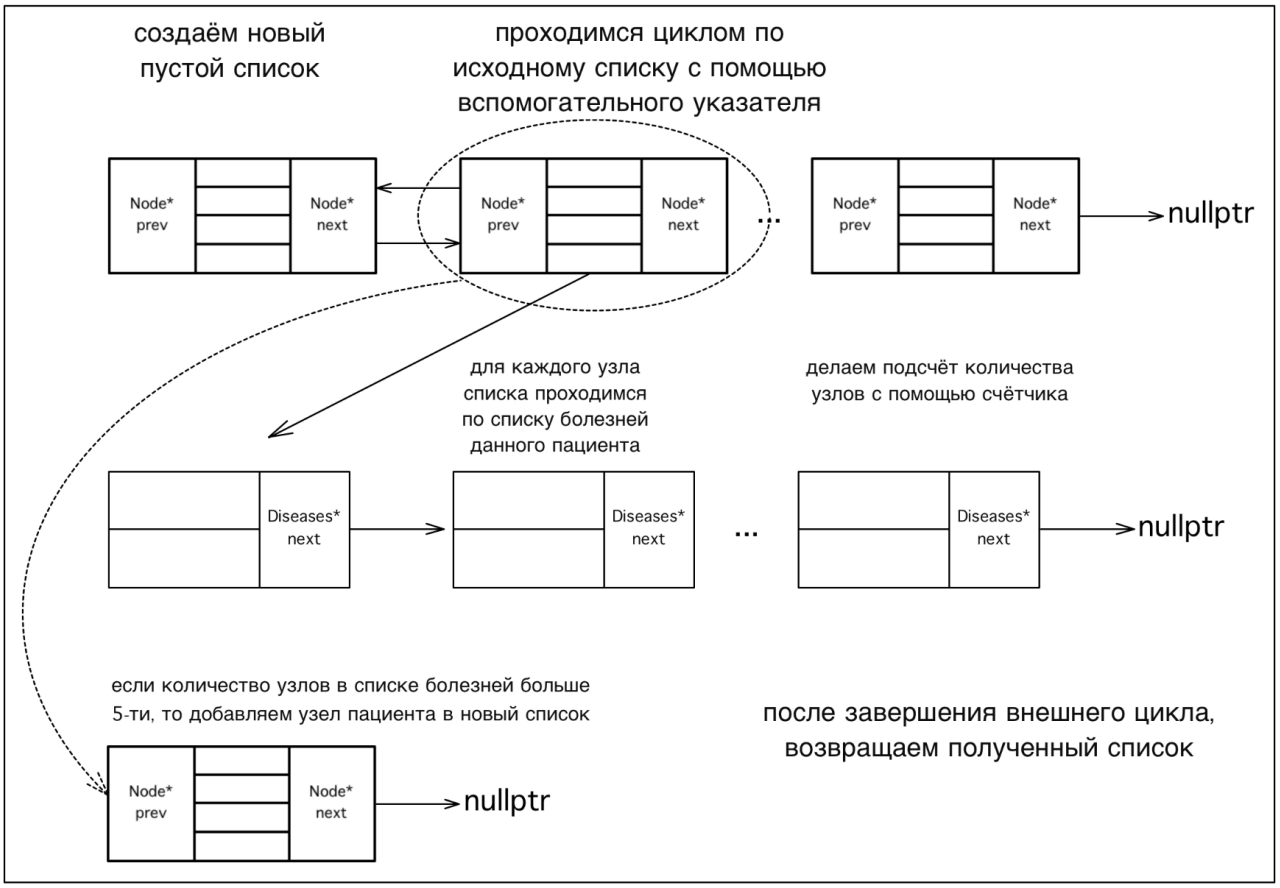

Рис. 4 – Функция формирования нового списка с пациентами, у которых более 5-ти заболеваний

Таблица тестов

№	Исходный список	Выходные данные
1	0005 Орлов 0004 Калинин 0003 Сидоров 0002 Петров 0001 Иванов	0002 Петров

4.5. Функция удаления болезни с данным кодом у данного пациента, remove_disease

4.5.1. Реализовать функцию удаления болезни с данным кодом у данного пациента

4.5.2. Инициализируем вспомогательный указатель Diseases*, указывающий на головной элемент списка болезней данного пациента. Отдельно проверяем первый элемент списка, является ли этот узел искомым, если да, то удаляем его путём переопределения начала списка на второй узел. Далее проходимся по списку, сравнивая поле code следующего узла с введённым в качестве параметра значением. Если встретился узел с искомым значением поля code, удаляем его путём переопределения значения поля next. Замечание: при проходе по списку проверяется следующий элемент, а не текущий, для удобства работы с однонаправленным списком заболеваний.

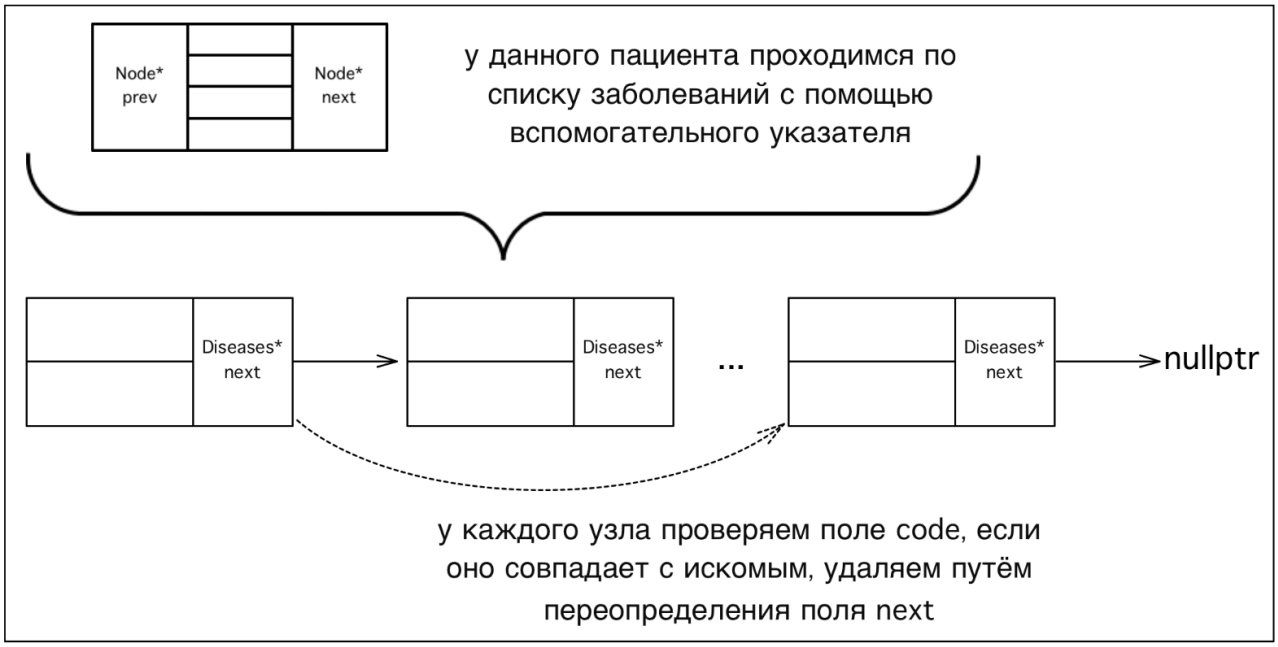

Рис. 5 – Функция удаления болезни с данным кодом у данного пациента

Таблица тестов

№	Исходный список	Входные данные	Выходные данные
1	COVID19 28.4.2021 CANCER 15.10.2019 SARS 5.1.2021 POX 16.2.2021 FLU 30.3.2021 ARI 10.2.2021	CANCER	COVID19 28.4.2021 SARS 5.1.2021 POX 16.2.2021 FLU 30.3.2021 ARI 10.2.2021

1.5 Код программы

Файл main.cpp

```
#include "main.h"

void insert_first (Node*& L, string policy, string name) {

    Node* q = new Node{policy, name, nullptr, nullptr, nullptr};

    if (!L) {
        L = q;
        L->prev = L;
        L->next = nullptr;
    } else {
        q->prev = L->prev;
        L->prev = q;
        q->next = L;
        L = q;
    }
}

void copy_and_insert (Node*& L, Node*& node) {
    Node* q = new Node{};
    q->name = node->name;
    q->policy = node->policy;
    q->diseases = node->diseases;

    if (!L) {
        L = q;
        L->prev = L;
        L->next = nullptr;
    } else {
        q->prev = L->prev;
        L->prev = q;
        q->next = L;
        L = q;
    }
}


void print (Node*& L, bool reverse) { //вывод списка в двух направлениях
    Node* q = L;

    if (!reverse) {
        while (q != nullptr) {
```

```
            cout << q->policy << " " << q->name << "\n";
            q = q->next;
        }
    } else {
        q = q->prev;
        while (q != L) {
            cout << q->policy << " " << q->name << "\n";
            q = q->prev;
        }
        cout << q->policy << " " << q->name << "\n";
    }
}

void print_diseases (Node* patient) {
    Diseases* q = patient->diseases;

    while (q) {
        cout << q->code << " " << q->date->tm_mday << "." << q->date->tm_mon <<
"." << q->date->tm_year << "\n";
        q = q->next;
    }
}

Node* find_by_policy (Node*& L, string policy) {
    Node* q = L;
    while (q != nullptr and q->policy != policy) {
        q = q->next;
    }
    return q;
}


void insert_disease (Node*& patient, string code, int day, int month, int year) {
    Diseases* q = new Diseases{};
    tm* date = new tm;
    date->tm_year = year;
    date->tm_mday = day;
    date->tm_mon = month;

    q->code = code;
    q->date = date;

    q->next = patient->diseases;
    patient->diseases = q;
```

```
}

Node* over_five_diseases (Node* L) {
   Node* list = new Node{};

   Node* q = L;
   Diseases* q1;
   int count;

   while (q) {
      count = 0;
      q1 = q->diseases;

      while (q1) {
         count++;
         q1 = q1->next;
      }

      if (count > 5) {
         copy_and_insert(list, q);
      }
      q = q->next;
   }
   return list;
}

void remove_disease (Node* patient, string code) {
   Diseases* q = patient->diseases;

   if (q->code == code) {
      patient->diseases = patient->diseases->next;
   }

   while (q->next) {

      if (q->next->code == code) {
         q->next = q->next->next;
      }

      q = q->next;
   }
}
```

Файл main.h

```
#ifndef main_h
#define main_h

#include <iostream>
#include <ctime>
using namespace std;

struct Diseases {
   string code;
   tm* date;
   Diseases* next;
};

struct Node {
   string policy;
   string name;
   Diseases* diseases;
   Node* next;
   Node* prev;
};

void insert_first (Node*& L, string policy, string name); //вставка узла на первую позицию

void copy_and_insert (Node*& L, Node*& node);

void print (Node*& L, bool reverse); //вывод списка в двух направлениях

Node* find_by_policy (Node*& L, string policy);

void insert_disease (Node*& patient, string code, int day, int month, int year);

Node* over_five_diseases (Node* L);

void remove_disease (Node* patient, string code);

#endif
```

5. Оценка сложности первого дополнительного алгоритма

Алгоритм вставки элемента в начало массива имеет сложность O(n), так как придется передвинуть все элементы для реализации вставки. Сложность вставки в начало списка будет O(1), так как для вставки требуется только поменять указатель на первый элемент (и указатель на следующий у теперь уже первого)

6. Выводы по полученным знаниям умениям

   Двунаправленный список – структура данных, позволяющая хранить упорядоченный набор данных, где каждый элемент имеет ссылку как на предыдущий, так и на следующий. Ссылка на предыдущий элемент первого равна null, как и ссылка следующего у последнего элемента.
   Двунаправленный список проигрывает массиву по скорости поиска элемента по индексу, однако явным плюсом является то, что разработчик сам определяет структуру узла.

7. Список информационных источников

1) Лекции по Структурам и алгоритмам обработки данных Скворцова Л. А. – МИРЭА – Российский технологический университет 2021 год