Содержание

1.Ответы на вопросы

1.1. Расскажите о трех уровнях представления данных в программной системе

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.

1.2. Что определяет тип данных?

Тип данных определяет:

- внутреннее представление данных в памяти компьютера;
- множество значений, которые могут принимать величины этого типа;
- операции и функции, которые можно применять к величинам этого типа.

1.3. Что определяет структура данных?

Структура данных — это контейнер, который хранит данные в определенном макете. Этот «макет» позволяет структуре данных быть эффективной в некоторых операциях и неэффективной в других

1.4. Расскажите о структуры хранения данных в компьютерных технологиях.

Структуры хранения данных могут быть статическими или динамическими. Статические структуры создаются при компиляции программы в области данных и их размер изменить нельзя во время работы программы. Они освобождают память, выделенную им, только по завершении программы. Динамические структуры создаются во время работы программы в динамической памяти, их размер может быть изменен во время работы программы. Память этой структуры хранения может и ее отдельных элементов быть освобождена во время работы программы.

1.5. Дайте определение линейной структуре данных.

Линейные структуры данных представляют список элементов, упорядоченных по положению. Доступ к элементу прямой по номеру.

1.6. Дайте определение структуре данных линейный список.

Структура данных задачи могут представлять линейный список значений – т.е. список значений, следующих друг за другом. Для таких данных можно использовать векторную структуру данных. Вектор представляет линейную структуру, состоящую из нескольких последовательно расположенных ячеек. Ячейка предназначена только для хранения значения и связи со следующей за ней ячейкой она не хранит.

1.7. Дайте определение структуре данных стек.

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

1.8. Дайте определение структуре данных очередь.

Очередь – это структура данных, представляющая собой последовательность элементов, образованная в порядке их поступления.

1.9. Чем стек отличается от структуры данных линейный список?

Список — это линейная цепочка элементов с последовательным доступом и вставкой / удалением элементов в любом месте списка. Стек — это доступ по схеме LIFO: "последним пришёл, первым ушёл", добавление и удаление элементов происходит с одного конца, а не в любом месте. Список - один из способов реализации стека в программе. Но это далеко не единственный и не самый эффективный способ.

1.10. Какой из видов линейных списков лучше использовать, если нужно введенную последовательность вывести наоборот?

Для вывода введённой последовательности, наоборот, лучше использовать линейную структуру Дек, так как в ней доступ к элементам осуществляется как с начала, так и с конца.

1.11. Определите сложность алгоритма операции вставки элемента в i-ую позицию: а) массива; б) линейного списка.

а) Сложность вставки в массив O(n)

б) Сложность вставки в структуру O(n)

1.12. Определите сложность алгоритма операции удаления элемента из i-ой позиции: а) массива; б) линейного списка.

а) Сложность удаления элемента из массива O(n)

б) Сложность удаления элемента из структуры O(n)

1.13. В чем суть трюка Вирта при выполнении операции удаления элемента из списка?

Трюк Вирта заключается в том, чтобы указатель на удаляемый элемент перенаправить на элемент, следующий за удаляемым.

1.14. Определите структур узла однонаправленного списка.

Односвязный список состоит из узлов. Узел содержит значение и указатель на следующий узел.

1.15. Реализуйте алгоритм вывода линейного однонаправленного списка.

```
void out_list(Node* L) {
Node* q = L;
while (q) {
cout << q->data << endl;
q = q->next;
}
}
```

1.16. Приведите фрагмент кода программы на языке C++ выполнения операции перемещения последнего элемента в начало списка.

```
void lasttofirst(Node*& L)
{
Node* LastOne = nullptr;
Node* FirstOne = L;
while (FirstOne->next)
{
LastOne = FirstOne;
FirstOne = FirstOne->next;
}

LastOne->next = 0;
FirstOne->next = L;

L = FirstOne;
}
```

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

Лишнее действие является условный оператор с оператором иначе (if … else):

```
if (LL==nullptr) LL->next=q;
else
```

Тогда код станет рабочим при условии, если вставлять в НЕ ПУСТОЙ список. Новый узел будет вставляться в конец списка.

2.Отчет по разработанной программе

2.1. Условие задания, требования в соответствии с вариантом.

Реализуйте программу решения задачи варианта по использованию линейного однонаправленного списка.

Требования для всех вариантов:

1. Информационная часть узла определена вариантом
2. Разработать функцию для создания исходного списка, используя функцию вставки нового узла перед первым узлом.
3. Разработать функцию вывода списка.
4. Разработать функции дополнительного задания варианта. При необходимости можно добавлять функции, декомпозируя задачу.
5. В основной программе выполните только тестирование каждой функции. Меню можно не создавать. Тесты обязательны.
6. Составить отчет по выполненному заданию (формат отчета после вариантов)

Выбор варианта: 11%17+1=12

2.2. Определение списка операций над списком, которые выявлены в процессе исследования задач дополнительного задания

Node* findByPos(Node* L, int pos); - поиск узла в списке L по позиции pos

void create_list(Node*& L, int n); - добавление в список L n узлов / создание списка L с n узлами

void sort_list(Node*& L); - сортировка списка L по возрастанию младшей цифры

int findMinLowestDigitPos(Node* L, int startFrom = 1); - поиск узла с наименьшей младшей цифрой в списке L, начиная искать с startFrom позиции

void print(Node* L); - вывод списка L

bool isSorted(Node* L); - проверка, отсортирован ли список L по возрастанию младшей цифры

2.3. Определить структуру узла однонаправленного списка в соответствии с вариантом

struct Node {

short number; - однозначное/двузначное число

Node* next; - указатель на последующий узел

};

2.4. Для каждой задачи варианта определить

2.4.1. Постановку задачи

Дан линейный однонаправленный список L1, информационная часть которого содержит однозначные и двузначные числа, упорядоченные в порядке возрастания старшей цифры.

1) Разработать функцию, которая удаляет узел в заданной позиции списка L1.

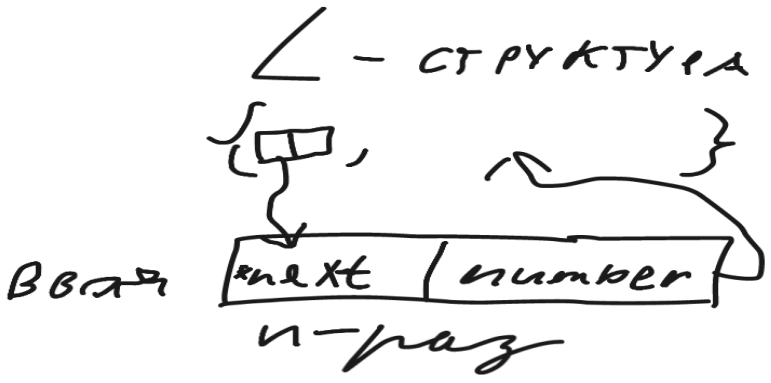
2) Разработать функцию, которая формирует новый список L2 вставляя в него элементы списка L1, располагая их в порядке возрастания младшей цифры. Удаляя из списка L1 перемещенный узел.

3) Разработать функцию, которая определяет, что список L2 упорядочен по возрастанию

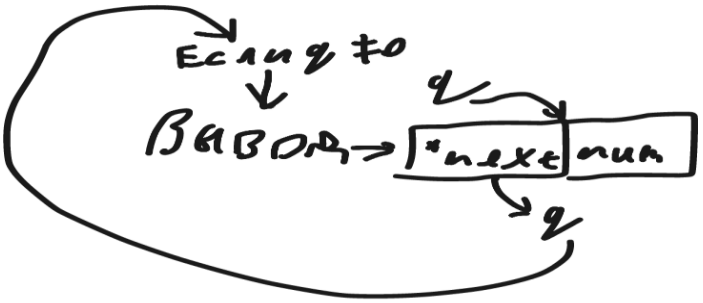
2.4.2. Разработать алгоритм выполнения операции, для этого:

1. **Изобразить (рисунок) для каждой операции процесс выполнения операции на существующем однонаправленном списке.**

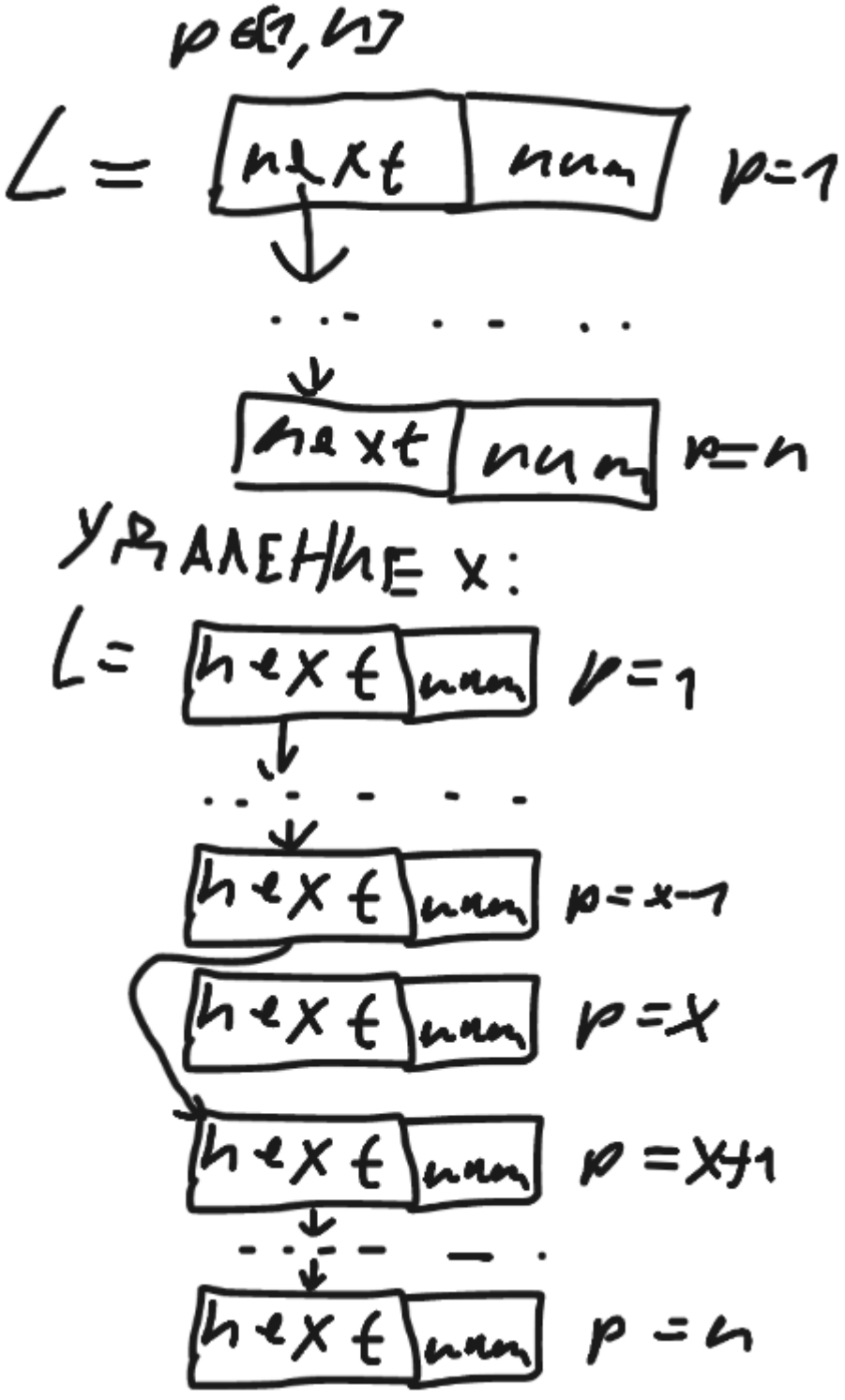
Операция создания нового списка



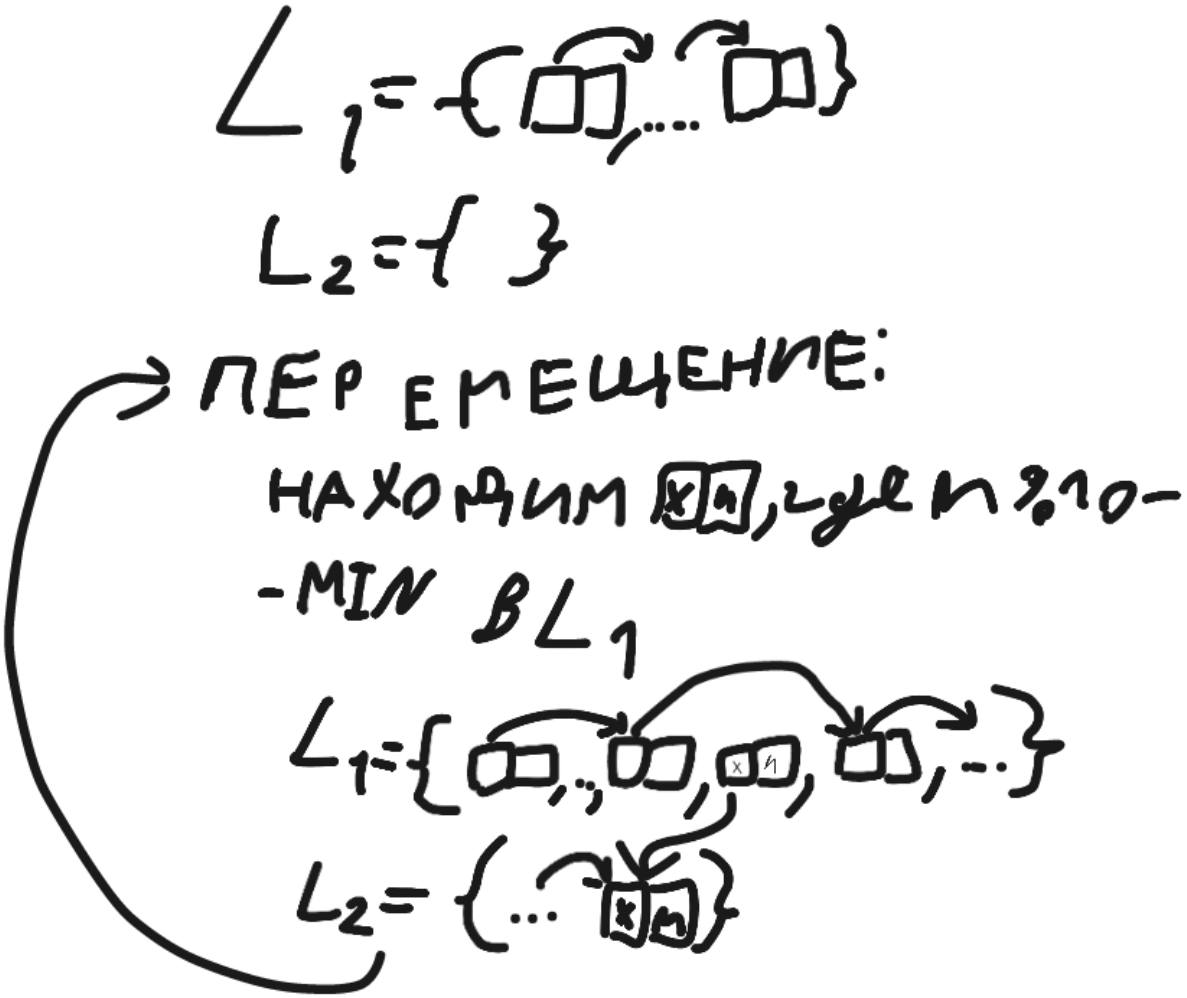
Операция вывода



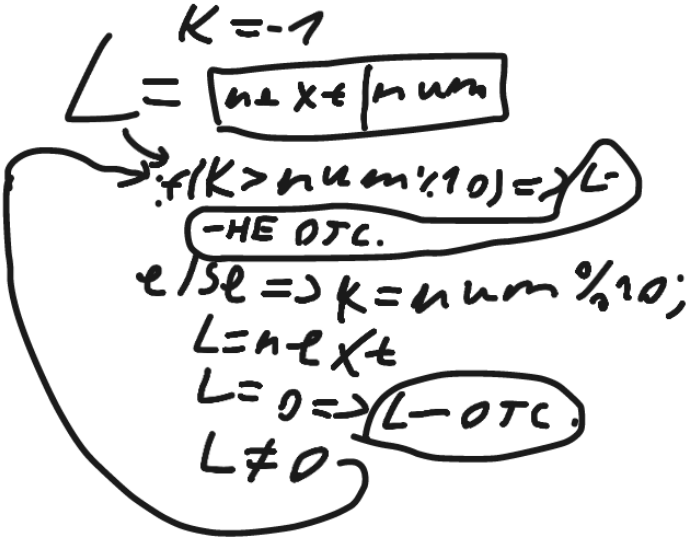
Операция, которая удаляет узел в заданной позиции списка L1



Операция, которая формирует новый список L2 вставляя в него элементы списка L1, располагая их в порядке возрастания младшей цифры. Удаляя из списка L1 перемещенный узел



Операция , которая определяет, что список L2 упорядочен по возрастанию



2. Привести алгоритм выполнения операции

Операция создания нового списка:

1) Инициализация q, q1 = 0 – указателей на узлы
2) Ввод информации для каждого узла
3)Добавление в список узла

Операция вывода:

1) Вывод информации узлов

Операция, которая удаляет узел в заданной позиции списка L1:

1) Инициализация q = 0 – указатель на узел
2) Если удаляемый узел – голова списка, то теперь голова списка – следующий узел и удаление узла. Вернуть 0
3) Если родитель удаляемого узла существует, то ребёнок родителя = ребёнок удаляемого узла и удаление узла. Вернуть 0

   Иначе вернуть -1

Операция, которая формирует новый список L2 вставляя в него элементы списка L1, располагая их в порядке возрастания младшей цифры. Удаляя из списка L1 перемещенный узел:

1) Находим узел с минимальным значением младшей цифры из списка L1
2) Удаляем этот узел из списка L1
3) Добавляем этот в конец списка L2
4) Переход на п.1, если L1 – не пустой список

Операция, которая определяет, что список L2 упорядочен по возрастанию:

1) Цикл (int i = 1, int p = минимальное значение младшей цифры из L списка; p > 0; пока выполняется условие i++; действие в конце итерации: p = минимальное значение младшей цифры)

   Тело цикла п.2

   После цикла п.3
2) Если i != p, то L – не отсортирован. Вернуть false.

   Иначе следующая итерация цикла(п.1)
3) L – отсортирован. Вернуть true

3. Разработать функцию реализации алгоритма

Операция создания нового списка

```
void create_list(Node*& L, int n) {
    Node* q, * q1 = nullptr;
    cout << "Значения чисел для " << n << " узлов:\n";
    for (int i = 1; i <= n; i++) {
        q = new Node;
        q->number = 100/i*pow(-1, i+i/3);
        q->next = nullptr;
        if (L == NULL) L = q;
        else {
            q1->next = q;
        }
        q1 = q;
    }
}
```

Операция вывода

```
void print(Node* L) {
    if (!L) {
        cout << "Список пуст!\n";
        return;
    }
    Node* q = L;
    for (int i = 1; q; i++, q = q->next) {
        if (i > 1) cout << " ";
        cout << q->number;
    }
    cout << endl;
}
```

Операция, которая удаляет узел в заданной позиции списка L1

```
int deleteByPos(Node*& L, int p) {
    Node* q = 0;
    if (p == 1) {
        q = L;
        L = L->next;
        delete q;
        return 0;
    }
    Node* q1 = findByPos(L, p-1);
    if (q1 != 0) {
        q = q1->next;
        if (q != 0) {//есть удаляемый объект
            q1->next = q->next;
            delete q;
            return 0;
        }
    }
    return -1;
}
```

Операция, которая формирует новый список L2 вставляя в него элементы списка L1, располагая их в порядке возрастания младшей цифры. Удаляя из списка L1 перемещенный узел

```
void sort_list(Node*& L) {
    if (!L) return;
    int p = findMinLowestDigitPos(L);
    Node* node = findByPos(L, p);
    if (p > 1) findByPos(L, p - 1)->next = node->next;
    else L = node->next;
    Node* L2 = node;
    Node* pasteTo = L2;
    while (true) {
        p = findMinLowestDigitPos(L);
        node = findByPos(L, p);
        if (!node) break;
        if (p > 1) findByPos(L, p - 1)->next = node->next;
        else L = node->next;
        pasteTo->next = node;
        pasteTo = node;
    }
    L = L2;
}
```

Операция , которая определяет, что список L2 упорядочен по возрастанию

```
bool isSorted(Node* L) {
    for (int i = 1, p = findMinLowestDigitPos(L, i); p > 0; i++, p = findMinLowestDigitPos(L, i)) {
        if (i != p) return false;
    }
    return true;
}
```

2.4.3. Привести таблицу тестов для тестирования каждой операции

Операция создания нового списка

Номер теста	Список, Число новых узлов, информация для узлов	Полученный список	Ожидаемый список
1	L = {}, 1 99	{99}	{99}
2	L = {99}, 2 35 13	{99,35,13}	{99,35,13}

Операция вывода

Номер теста	Список	Полученный вывод	Ожидаемый вывод
1	L = {-100,50,20,33,-25}	Список: -100 50 33 -25 20	Список: -100 50 33 -25 20

Операция, которая удаляет узел в заданной позиции списка L1

Номер теста	Список, Удаляемая позиция	Полученный список	Ожидаемый список
1	L = {-100,50,20,33,-25}, 1	{50,20,33,-25}	{50,20,33,-25}
2	L = {50 20 33 -25}, 2	{50,33,-25}	{50,33,-25}

Операция, которая формирует новый список L2 вставляя в него элементы списка L1, располагая их в порядке возрастания младшей цифры. Удаляя из списка L1 перемещенный узел

Номер теста	Список L1	Полученный список L2	Ожидаемый список L2
1	L1 = {-100,50,33, -25,20}	L2 = {-100,50,20, 33,-25}	L2 = {-100,50,20, 33,-25}

Операция, которая определяет, что список L2 упорядочен по возрастанию

Номер теста	Список L	Полученный результат	Ожидаемый результат
1	L = {-100,50,33, -25,20}	false	false
2	L = {-100,50,20, 33,-25}	true	true

2.5. Представить код программы

```
//1.12
#include <iostream>
#include <time.h>

using namespace std;
```

```
struct Node {
short number;
Node* next;
};

Node* findByPos(Node* L, int pos);
void create_list(Node*& L, int n);
void sort_list(Node*& L);
int findMinLowestDigitPos(Node* L, int startFrom = 1);
void print(Node* L);
bool isSorted(Node* L);
int deleteByPos(Node*&, int);

int main() {
setlocale(LC_ALL, "RUS");
srand(time(0));
Node* L = NULL;
int n;
//cout << "Размер списка:\n";
//cin >> n;
n = 5;
create_list(L, n);
cout << "Список:\n";
print(L);
cout << "Отсортирован ли список по возрастанию?\n";
cout << (isSorted(L) ? "ДА" : "НЕТ") << endl;
int pos = 2;
cout << "Значение в позиции " << pos << ":\n";
cout << findByPos(L, 2)->number << endl;
```

```
pos = findMinLowestDigitPos(L);
cout << "Элемент списка с минимальным значением младшей цифры:\n";
cout << pos << " " << findByPos(L, pos)->number << endl;
sort_list(L);
cout << "Отсортированный список по возрастанию значения младшей цифры:\n";
print(L);
cout << "Отсортирован ли список по возрастанию?\n";
cout << (isSorted(L) ? "ДА" : "НЕТ") << endl;
cout << "Удаление узла с позицией 1\n";
deleteByPos(L, 1);
print(L);
cout << "Удаление узла с позицией 2\n";
deleteByPos(L, 2);
print(L);
system("pause");
return 0;
}

//Предусловие: L - указатель на узел не пустого списка, pos - позиция получаемого узла - 1 <= pos <= размера узла
//Постусловие: Указатель на элемент с позицией pos относительно верхушки списка L
Node* findByPos(Node* L, int pos) {
if (pos < 1) return 0;
Node* q = L;
for (int i = 1; q != 0 && i < pos; i++) {
q = q->next;
}
return q;
```

```
}
```

//Предусловие: L - указатель на узел не пустого/пустого списка, n - количество добавляемых элементов - 1 <= n

//Постусловие: Замена указателя L на начало списка, если L было nullptr. Добавлены n элементов в список. Значения заполняемых элементов определяются функцией 100/i*pow(-1, i+i/3)

```
void create_list(Node*& L, int n) {
Node* q, * q1 = nullptr;
cout << "Значения чисел для " << n << " узлов:\n";
for (int i = 1; i <= n; i++) {
q = new Node;
q->number = 100 / i * pow(-1, i + i / 3);
q->next = nullptr;
if (L == NULL) L = q;
else {
q1->next = q;
}
q1 = q;
}
}
```

//Предусловие: L - указатель на узел не пустого списка

//Постусловие: Замена указателя L на начало списка L2 - отсортированного списка по возрастанию, созданного из узлов списка L

```
void sort_list(Node*& L) {
if (!L) return;
int p = findMinLowestDigitPos(L);
Node* node = findByPos(L, p);
if (p > 1) findByPos(L, p - 1)->next = node->next;
```

```
else L = node->next;
Node* L2 = node;
Node* pasteTo = L2;
while (true) {
p = findMinLowestDigitPos(L);
node = findByPos(L, p);
if (!node) break;
if (p > 1) findByPos(L, p - 1)->next = node->next;
else L = node->next;
pasteTo->next = node;
pasteTo = node;
}
L = L2;
}
```

//Предусловие: L - указатель на узел не пустого списка, startFrom - целочисленное значение индекса, с которого начинать поиск - 1 <= startFrom <= размера списка

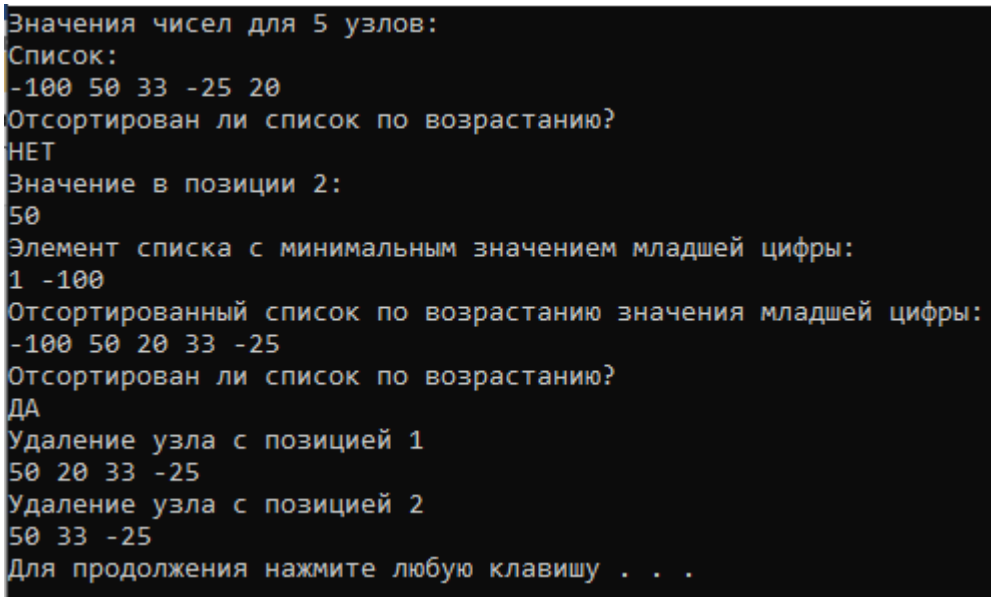
//Постусловие: Получение относительной позиции узла с минимальной младшей цифрой относительно списка узла L

```
int findMinLowestDigitPos(Node* L, int startFrom) {
if (startFrom < 1) startFrom = 1;
Node* q = L;
int min = INT_MAX;
int p = -1;
for (int i = 1; i < startFrom; i++) {
q = q->next;
}
for (int i = startFrom; q != 0; i++, q = q->next) {
if (abs(q->number % 10) < min || i == 1) {
```

```
min = abs(q->number % 10);
p = i;
}
}
return p;
}

//Предусловие: L - указатель на узел не пустого списка
//Постусловие: Вывод значений узлов списка L
void print(Node* L) {
if (!L) {
cout << "Список пуст!\n";
return;
}
Node* q = L;
for (int i = 1; q; i++, q = q->next) {//пока не найдём NULL в последнем узле
if (i > 1) cout << " ";
cout << q->number;
}
cout << endl;
}

//Предусловие: L - указатель на узел не пустого списка
//Постусловие: Получение истинности, что список узла L - отсортирован
bool isSorted(Node* L) {
for (int i = 1, p = findMinLowestDigitPos(L, i); p > 0; i++, p = findMinLowestDigitPos(L, i)) {
if (i != p) return false;
}
return true;
```

```
}

//Предусловие: L - указатель на узел не пустого списка, p - позиция удаляемого узла, 1 <= p <= размера списка L
//Постусловие: Удаление узла на позиции p
int deleteByPos(Node*& L, int p) {
Node* q = 0;
if (p == 1) {
q = L;
L = L->next;
delete q;
return 0;
}
Node* q1 = findByPos(L, p-1);
if (q1 != 0) {
q = q1->next;
if (q != 0) {//есть удаляемый объект
q1->next = q->next;
delete q;
return 0;
}
}
return -1;
}
```

2.6. Представить результат тестирования программы: скриншоты выполнения каждой операции.



2.7. Привести выводы по полученным знания и умениям

При работе над программой были получены и усвоены знания об однонаправленных списках. Также были получены умения по выполнению различных операциях над списками

2.8. Список информационных источников, которые были использованы при выполнении задания.

1. Лекции по дисциплине «Структуры и алгоритмы обработки данных» (Лектор: Скворцова Л.А.) (2021).