Министерство науки и высшего образования Российской Федерации
Федеральное государственное бюджетное образовательное учреждение высшего образования
Пермский национальный исследовательский политехнический университет
Электротехнический факультет
Кафедра информационных технологий и автоматизированных систем

Лабораторная работа №11

"двунаправленные списки"

Вариант: 12

Выполнил студент ИВТ-24-2б:
Шишкин Максим Григорьевич

_____
(дата, подпись)

Проверил доцент кафедры ИТАС:

Полякова Ольга Андреевна

_____
(дата, подпись)

Пермь 2025

Содержание

1 Постановка задачи

Записи в линейном списке содержат ключевое поле типа *char(строка символов). Сформировать двунаправленный список. Удалить из него Элементы, с одинаковыми ключевыми полями. Добавить элемент после элемента с заданным ключевым полем.

2 Код на C++

```cpp
#include <iostream>
#include <string>
#include <clocale>

using namespace std;

// Структура узла двунаправленного списка
struct Node {
    string key;
    Node* prev;
    Node* next;
};

// Функция для создания нового узла
Node* createNode(const string& k) {
    Node* newNode = new Node;
    newNode->key = k;
    newNode->prev = nullptr;
    newNode->next = nullptr;
    return newNode;
}


// Функция для добавления элемента в конец двунаправленного списка
void push_back(Node*& head, const string& k) {
    Node* newNode = createNode(k);

    if (head == nullptr) {
        head = newNode;
        return;
    }

    Node* curr = head;
    while (curr->next != nullptr) {
        curr = curr->next;
    }

    curr->next = newNode;
    newNode->prev = curr;
}

// Функция для формирования двунаправленного списка из массива строк
Node* createList(string* strArr, int sz) {
    Node* head = nullptr;
    for (int i = 0; i < sz; ++i) {
        push_back(head, strArr[i]);
    }
```

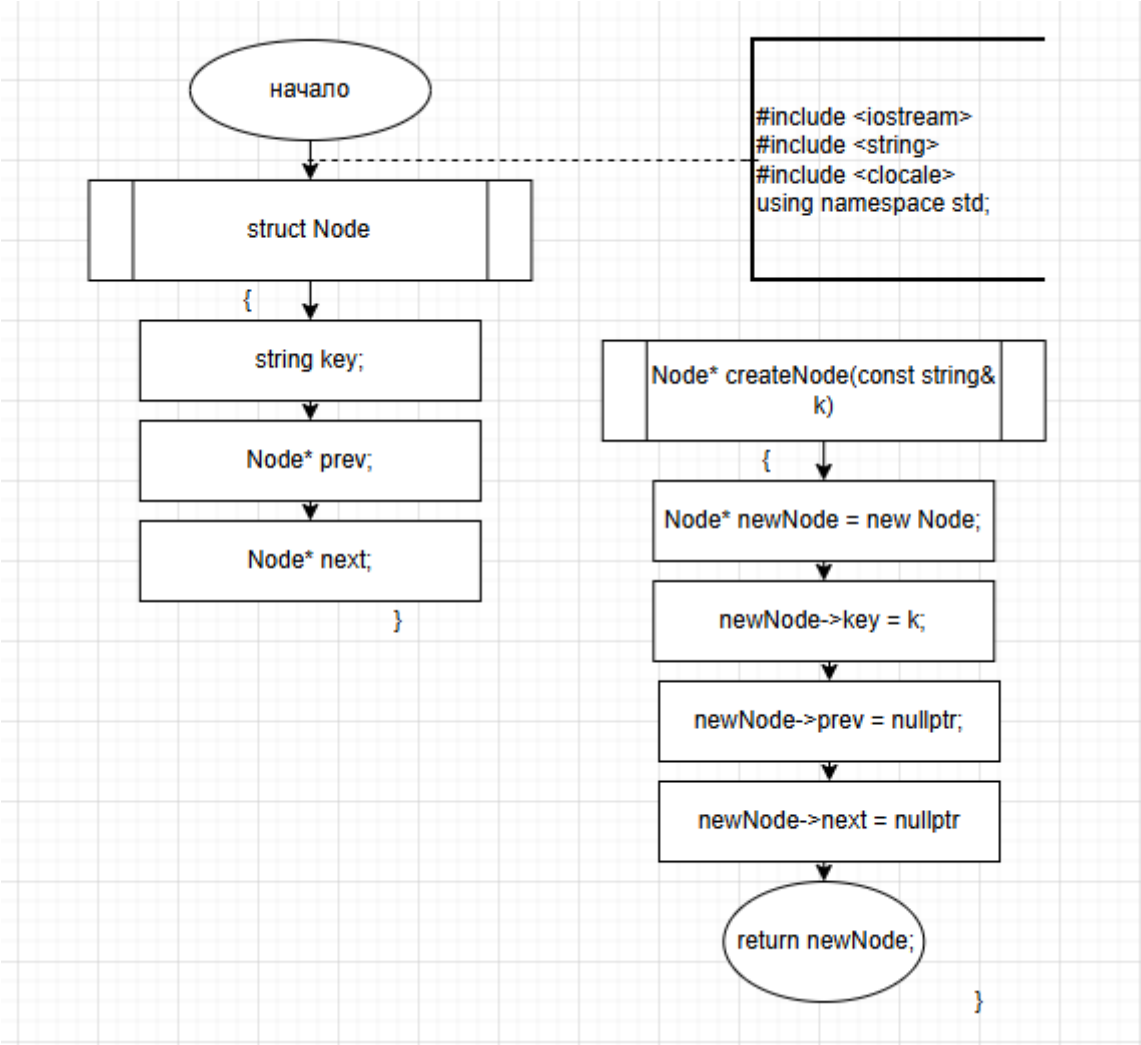
```
48      return head;
49  }
50
51  // Функция для вывода двунаправленного списка
52  void printList(Node* head) {
53      Node* curr = head;
54      while (curr != nullptr) {
55          cout << curr->key << " ";
56          curr = curr->next;
57      }
58      cout << endl;
59  }
60
61  // Функция для удаления элементов с одинаковыми ключевыми полями
62  void removeDuplicates(Node*& head) {
63      if (head == nullptr) {
64          return;
65      }
66
67      Node* curr = head;
68      while (curr != nullptr) {
69          Node* runner = curr->next;
70          Node* prevRunner = curr; // Добавили переменную для хранения предыдущего элемента runner'a
71          while (runner != nullptr) {
72              if (curr->key == runner->key) {
73                  // Нашли дубликат
74                  if (runner->next != nullptr) {
75                      runner->next->prev = runner->prev;
76                  }
77
78                  if (runner->prev != nullptr) {  // Проверяем, что runner не является head
79                      runner->prev->next = runner->next;
80                  }
81                  else {
82                      head = runner->next; // Если runner == head
83                      if (head != nullptr) {
84                          head->prev = nullptr; //Устанавливаем prev нового head в nullptr
85                      }
86                  }
87
88                  Node* temp = runner;
89                  runner = runner->next;
90
91                  delete temp;
92
93              }
94              else {
```

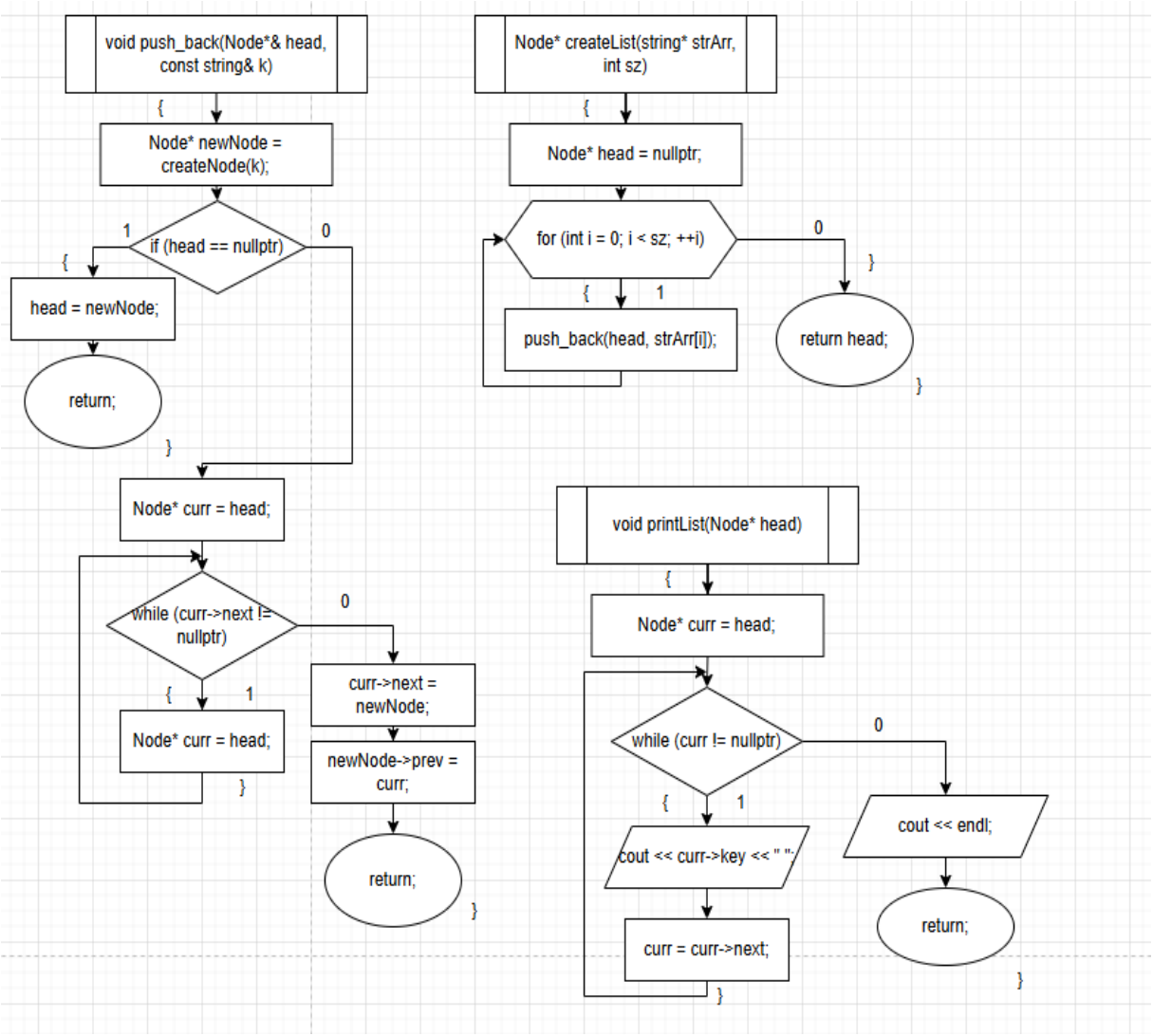
```
                    prevRunner = runner;
                    runner = runner->next;
                }

            }
            curr = curr->next;
        }
    }
    // Функция для добавления элемента после элемента с заданным ключевым полем
    void addAfter(Node* head, const string& targetKey, const string& newKey) {
        if (head == nullptr) {
            return;
        }

        Node* curr = head;
        while (curr != nullptr) {
            if (curr->key == targetKey) {
                // Нашли элемент после которого нужно добавить
                Node* newNode = createNode(newKey);

                newNode->next = curr->next;
                newNode->prev = curr;

                if (curr->next != nullptr) {
                    curr->next->prev = newNode;
                }

                curr->next = newNode;
                return; // Завершаем, т.к. добавили один элемент
            }
            curr = curr->next;
        }

        // Если targetKey не найден, ничего не делаем
        cout << "Элемент с ключом '" << targetKey << "' не найден." << endl;
    }
    // Функция для освобождения памяти, занимаемой списком
    void deleteList(Node* head) {
        Node* curr = head;
        while (curr != nullptr) {
            Node* next = curr->next;
            delete curr;
            curr = next;
        }
    }
    int main() {
        // Пример использования
      setlocale(LC_ALL, "ru");
      string strArr[] = { "sos", "acer", "dog", "sos", "date", "acer", "cat" }; // Изменили тип массива на string
      int sz = sizeof(strArr) / sizeof(strArr[0]);

      cout << "Исходный линейный список: ";
      for (int i = 0; i < sz; ++i) {
          cout << strArr[i] << " ";
      }
      cout << endl;

      // 1. Формирование двунаправленного списка
      Node* head = createList(strArr, sz);
      cout << "Двунаправленный список: ";
      printList(head);

      // 2. Удаление элементов с одинаковыми ключевыми полями
      removeDuplicates(head);
      cout << "Двунаправленный список после удаления дубликатов: ";
      printList(head);

      // 3. Добавление элемента после элемента с заданным ключевым полем
      addAfter(head, "dog", "mouse");
      cout << "Двунаправленный список после добавления элемента: ";
      printList(head);

      // Освобождаем память
      deleteList(head);

      return 0;
  }
```

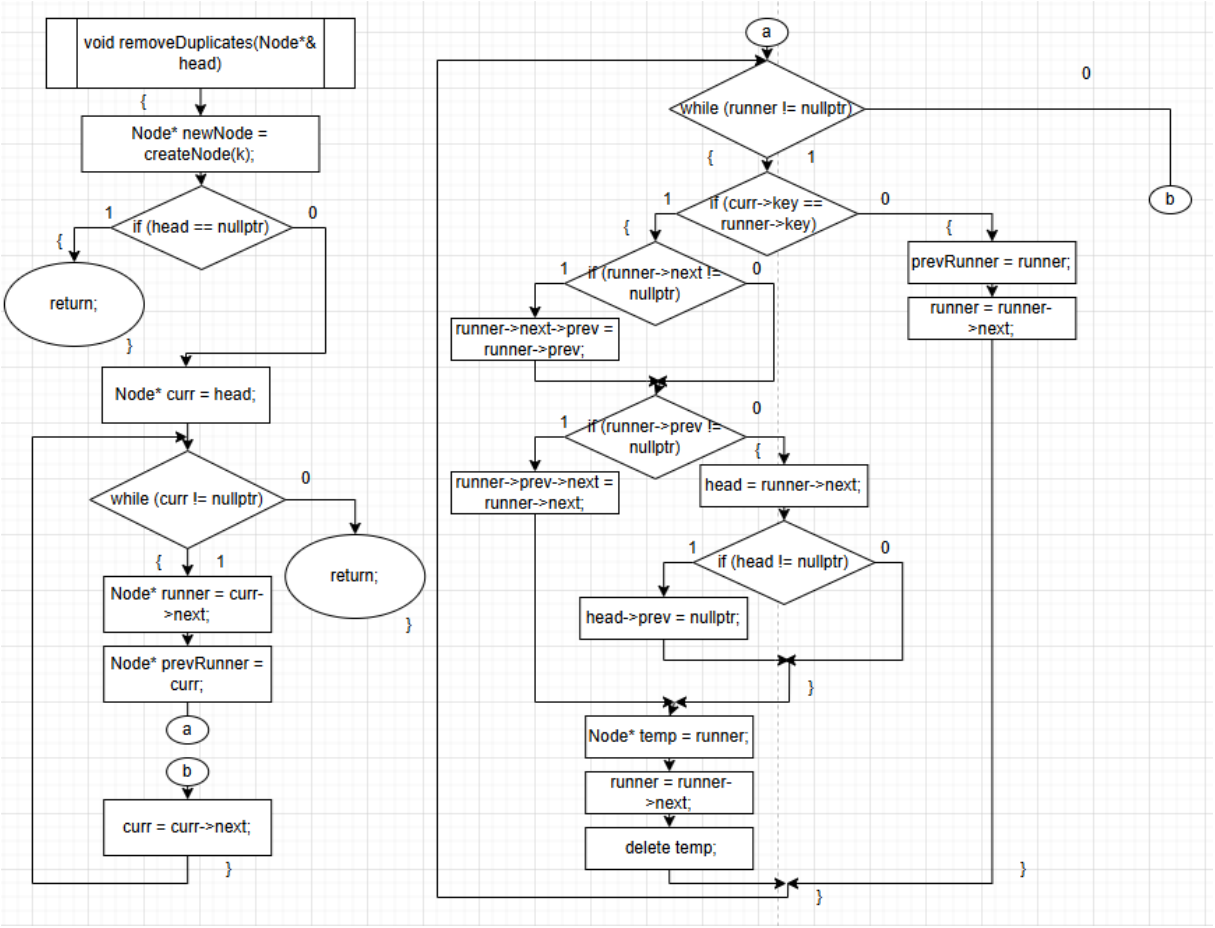
3 Результат работы кода

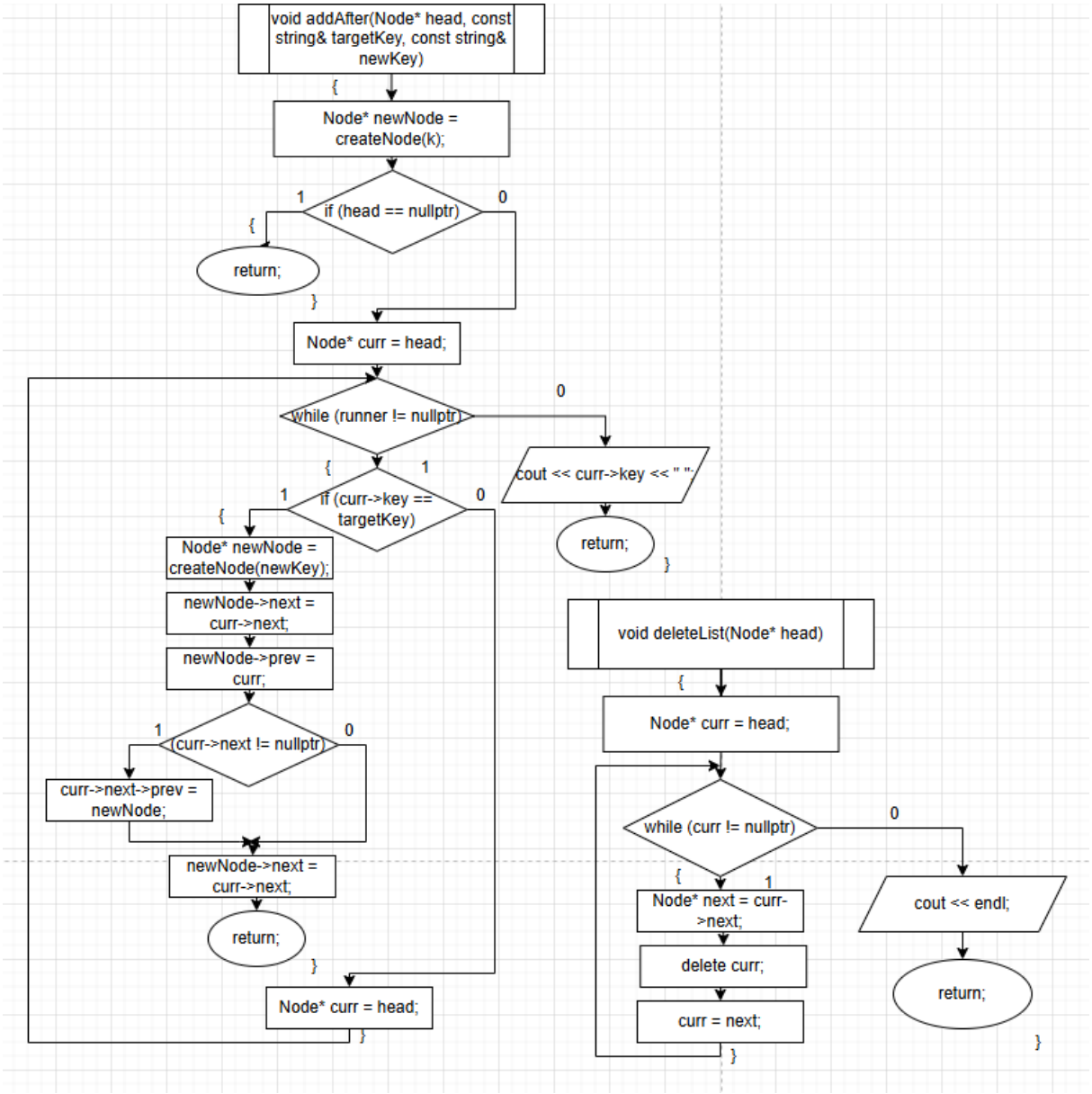
```
Исходный линейный список: sos acer dog sos date acer cat
Двунаправленный список: sos acer dog sos date acer cat
Двунаправленный список после удаления дубликатов: sos acer dog date cat
Двунаправленный список после добавления элемента: sos acer dog mouse date cat
```

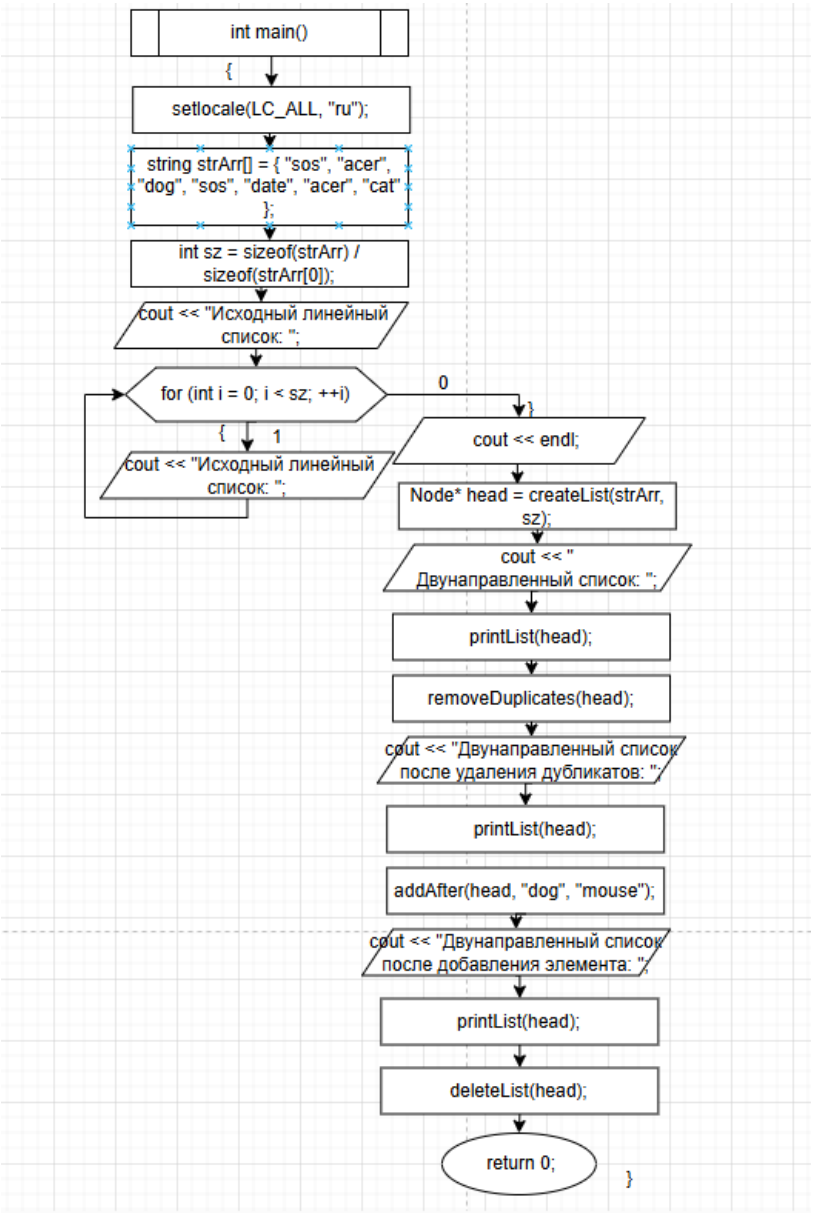
4 Блок-схема




void push_back(Node*& head, const string& k)
{
Node* newNode = createNode(k);
if (head == nullptr)
1
0
{
head = newNode;
return;
}
Node* curr = head;
while (curr->next != nullptr)
0
{
1
Node* curr = head;
}
curr->next = newNode;
newNode->prev = curr;
return;
}
Node* createList(string* strArr, int sz)
{
Node* head = nullptr;
for (int i = 0; i < sz; ++i)
0
}
{
1
push_back(head, strArr[i]);
return head;
}
void printList(Node* head)
{
Node* curr = head;
while (curr != nullptr)
0
{
1
cout << curr->key << " ";
curr = curr->next;
}
cout << endl;
return;
}


void removeDuplicates(Node*& head)
{
Node* newNode = createNode(k);
if (head == nullptr)
1
0
{
return;
}
Node* curr = head;
while (curr != nullptr)
0
{
1
Node* runner = curr->next;
Node* prevRunner = curr;
a
b
curr = curr->next;
}
return;
}
a
while (runner != nullptr)
0
{
1
b
if (curr->key == runner->key)
1
0
{
{
prevRunner = runner;
runner = runner->next;
if (runner->next != nullptr)
1
0
runner->next->prev = runner->prev;
if (runner->prev != nullptr)
1
0
{
runner->prev->next = runner->next;
head = runner->next;
if (head != nullptr)
1
0
head->prev = nullptr;
}
Node* temp = runner;
runner = runner->next;
delete temp;
}
}


void addAfter(Node* head, const string& targetKey, const string& newKey)
{
Node* newNode = createNode(k);
if (head == nullptr)
1
0
{
return;
}
Node* curr = head;
while (runner != nullptr)
0
{
if (curr->key == targetKey)
1
0
1
{
cout << curr->key << " "-
return;
}
Node* newNode = createNode(newKey);
newNode->next = curr->next;
newNode->prev = curr;
(curr->next != nullptr)
1
0
curr->next->prev = newNode;
newNode->next = curr->next;
return;
}
Node* curr = head;
}
void deleteList(Node* head)
{
Node* curr = head;
while (curr != nullptr)
0
{
1
Node* next = curr->next;
delete curr;
curr = next;
}
cout << endl;
return;
}



5 Ссылка на github

ссылка на github - https://github.com/MAKSPOWERO/mas1