Министерство науки и высшего образования Российской Федерации

Федеральное государственное автономное учреждение высшего образования

«Пермский национальный исследовательский политехнический университет»

ПНИПУ

**Лабораторная работа**

**«№11»**

Выполнил:

студент группы РИС-23-2б

Ившин Максим Сергеевич

Проверила:

доцент кафедры ИТАС

О.А. Полякова

2024 г.

**Разработка алгоритма**

**Постановка задачи:**

Написать программу, в которой создаются динамические структуры и выполнить их обработку в соответствии со своим вариантом.

Записи в линейном списке содержат ключевое поле типа \*char(строка символов). Сформировать двунаправленный список. Удалить элемент с заданным номером. Добавить К элементов в начало списка.

**Однонаправленные списки**

**Анализ задачи:**

1. Структура Node содержит два поля: data - данные узла, и next - указатель на следующий узел в списке.
2. Структура список содержит два поля: head - указатель на головной узел списка, и teil - указатель на хвостовой узел списка.
3. Функция push\_back добавляет новый элемент в конец списка.
4. Если список пустой, новый узел становится головным и хвостовым узлом. Если список не пустой, новый узел связывается с хвостовым узлом и становится новым хвостовым узлом.
5. Функция print выводит текущий список с помощью итерационного цикла. Если список пустой, выводится сообщение "Список пустой!".
6. Функция clear удаляет все элементы из списка. Головной узел списка заменяется следующим узлом, и так до тех пор, пока все узлы не будут удалены. Память освобождается.

**Код на C++:**

#include <iostream>

#include <string>

#include <clocale>

using namespace std;

struct Node

{

    char data;

    Node\* next\_ptr = nullptr;

};

struct List

{

    Node\* head = nullptr;

    Node\* tail = nullptr;

    void print()

    {

        cout << endl << "------------------------------" << endl;

        for (Node\* node = head; node != nullptr; node = node->next\_ptr)

            cout << node->data << ' ';

        cout << endl << "------------------------------" << endl;

    }

    void pop\_front()

    {

        if (head == nullptr)

            return;

        if (head == tail)

        {

            delete tail;

            head = tail = nullptr;

            return;

        }

        Node\* node = head;

        head = node->next\_ptr;

        delete node;

    }

    void push\_back(char data)

    {

        Node\* node = new Node;

        node->data = data;

        if (head == nullptr)

            head = node;

        if (tail != nullptr)

            tail->next\_ptr = node;

        tail = node;

    }

    void push\_front(char data)

    {

        Node\* node = new Node;

        node->data = data;

        node->next\_ptr = head;

        head = node;

        if (tail == nullptr)

            tail = node;

    }

    void pop\_back()

    {

        if (tail == nullptr)

            return;

        if (head == tail)

        {

            delete tail;

            head = tail = nullptr;

            return;

        }

        Node\* node = head;

        for (; node->next\_ptr != tail; node = node->next\_ptr)

            ;

        node->next\_ptr = nullptr;

        delete tail;

        tail = node;

    }

    Node\* getIndex(int k)

    {

        if (k < 0)

            return 0;

        Node\* node = head;

        int n = 0;

        while (node && n != k && node->next\_ptr)

        {

            node = node->next\_ptr;

            n++;

        }

        if (n == k)

            return node;

        else

            return nullptr;

    }

    void erase(int k)

    {

        if (k < 0)

            return;

        if (k == 0)

        {

            pop\_front();

            return;

        }

        Node\* left = getIndex(k - 1);

        Node\* node = left->next\_ptr;

        if (node == nullptr)

            return;

        Node\* right = node->next\_ptr;

        left->next\_ptr = right;

        if (node == tail)

            tail = left;

        delete node;

    }

    void clear()

    {

        while (head != nullptr)

            pop\_front();

    }

};

int main()

{

    setlocale(LC\_ALL, "ru");

    int n;

    int index = 0;

    List list;

    cout << "Введите количество элементов списка: ";

    cin >> n;

    char temp;

    for (int i = 0; i < n; ++i)

    {

        cout << "Введите " << i + 1 << " элемент: ";

        cin >> temp;

        list.push\_back(temp);

    }

    cout << endl;

    list.print();

    cout << "Введите номер элемента, который хотите удалить: ";

    cin >> temp;

    index = 0;

    for (Node\* node = list.head; node != nullptr; node = node->next\_ptr)

    {

        if (index == int(temp)- int('0') - 1)

        {

            list.erase(index);

            break;

        }

        index++;

    }

    list.print();

    cout << "Количество элементов для добавления в начало списка: ";

    cin >> n;

    for (int i = 0; i < n; ++i)

    {

        cout << "Введите " << i + 1 << " элемент: ";

        cin >> temp;

        list.push\_front(temp);

    }

    cout << endl;

    cout << "Результат: ";

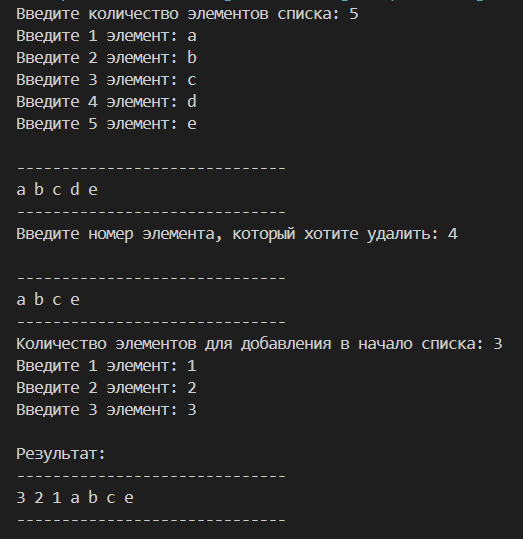
    list.print();

    cout << endl;

    list.clear();

}

Пример работы программы:



**Двунаправленные списки**

**Анализ задачи:**

1. Структура узел содержит данные и два указателя на предыдущий и следующий узлы.
2. Структура список содержит указатели на головной и хвостовой узлы списка.
3. Функция push\_tail добавляет новый элемент в конец списка. Если список пуст, новый узел становится головным и хвостовым узлом. Если список не пуст, новый узел связывается с головным узлом или вставляется в нужное место в списке.
4. Функция erase “связывает” указателями предыдущий и следующий элементы относительно элемента с указанным индексом и стирает его из списка.
5. Функция clear удаляет все элементы из списка. Головной узел списка заменяется следующим узлом, и так до тех пор, пока все узлы не будут удалены. Память освобождается.

**Код на С++:**

#include <iostream>

#include <clocale>

using namespace std;

struct Node

{

    char data;

    Node \*prev = nullptr, \*next = nullptr;

};

struct LinkedList

{

    Node \*head = nullptr, \*tail = nullptr;

    void print()

    {

        cout << "------------------------------" << endl;

        for (Node \*node = head; node != nullptr; node = node->next)

            cout << node->data << ' ';

        cout << endl

             << "------------------------------" << endl;

    }

    void push\_head(char data)

    {

        Node \*ptr = new Node;

        ptr->data = data;

        ptr->next = head;

        if (head != nullptr)

            head->prev = ptr;

        if (tail == nullptr)

            tail = ptr;

        head = ptr;

    }

    void push\_tail(char data)

    {

        Node \*ptr = new Node;

        ptr->data = data;

        ptr->prev = tail;

        if (tail != nullptr)

            tail->next = ptr;

        if (head == nullptr)

            head = ptr;

        tail = ptr;

    }

    void pop\_front()

    {

        if (head == nullptr)

            return;

        Node \*ptr = head->next;

        if (ptr != nullptr)

            ptr->prev = nullptr;

        else

            tail = nullptr;

        delete head;

        head = ptr;

    }

    void pop\_back()

    {

        if (tail == nullptr)

            return;

        Node \*ptr = tail->prev;

        if (ptr != nullptr)

            ptr->next = nullptr;

        else

            head = nullptr;

        delete tail;

        tail = ptr;

    }

    Node \*getPtr(int index)

    {

        Node \*ptr = head;

        int n = 0;

        while (n != index)

        {

            if (ptr == nullptr)

                return ptr;

            ptr = ptr->next;

            n++;

        }

        return ptr;

    }

    void erase(int index)

    {

        Node \*ptr = getPtr(index);

        if (ptr == nullptr)

            return;

        if (ptr->prev == nullptr)

        {

            pop\_front();

            return;

        }

        if (ptr->next == nullptr)

        {

            pop\_back();

            return;

        }

        Node \*left = ptr->prev;

        Node \*right = ptr->next;

        left->next = right;

        right->prev = left;

        delete ptr;

    }

    void clear()

    {

        while (head != NULL)

            pop\_front();

    }

};

int main()

{

    setlocale(LC\_ALL, "ru");

    int n;

    int index;

    LinkedList list;

    cout << "Введите количество элементов списка: ";

    cin >> n;

    char temp;

    for (int i = 0; i < n; ++i)

    {

        cout << "Введите " << i + 1 << "-ый элемент: ";

        cin >> temp;

        list.push\_tail(temp);

    }

    list.print();

    cout << "Введите номер элемента, который хотите удалить: ";

    cin >> temp;

    index = 0;

    for (Node \*node = list.head; node != nullptr; node = node->next)

    {

        if (index == int(temp) - int('0') - 1)

        {

            list.erase(index);

            break;

        }

        index++;

    }

    cout << endl;

    list.print();

    cout << "Введите сколько элементов необходимо добавить в начало списка: ";

    cin >> n;

    for (int i = 0; i < n; ++i)

    {

        cout << "Введите элемент: ";

        cin >> temp;

        list.push\_head(temp);

    }

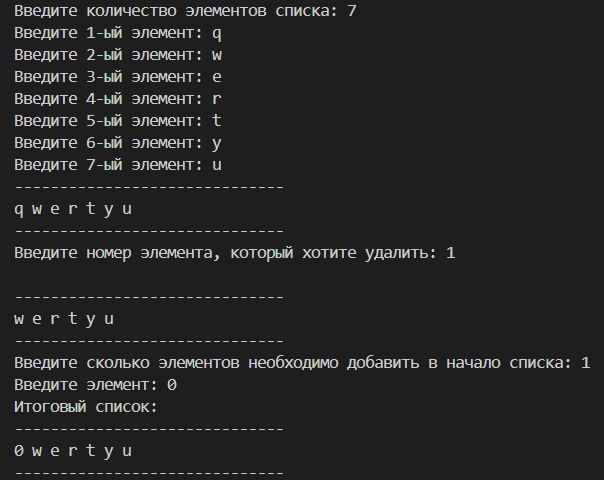
    cout << "Итоговый список: ";

    list.print();

    list.clear();

}

**Пример работы программы:**



**Стек**

**Анализ задачи:**

1) Нужно создать структуру, которая будет содержать поля (data) для данных и (Stack\* next\_ptr;) для адреса следующего элемента.

2) Создам метод для добавления элемента в стек push().

3) Создам функцию для вывода элементов стека print(), которая проходит от головы до хвоста и выводит все элементы.

4) Создам функцию для возвращения верхнего элемента стека top().

7) Чтобы удалить элемент с заданным ключом создадим второй стек, в который будут переноситься нужные элементы главного стека. После чего с помощью цикла и оператора выбора перенесём все нужные элементы во второй стек и посчитаем количество элементов равных ключу. Затем с помощью арифметического цикла перенесём элементы из второго стека в исходный. После чего с помощью ранее написанной функции выведем элементы стека на экран.

8) Добавить К элементам в начало стека. После этого перенесём с помощью арифметического цикла нужные элементы во второй стек. Затем с помощью арифметического цикла добавим k элементов в начало исходного стека. После чего с помощью арифметического цикла перенесём элементы из второго стека в исходный.

**Код на С++:**

#include <iostream>

using namespace std;

struct Node

{

    char data;

    Node \*next\_ptr = nullptr;

};

struct Stack

{

    Node \*head = nullptr, \*tail = nullptr;

    int size = 0;

    void print()

    {

        Node \*node = head;

        while (node != nullptr)

        {

            cout << node->data << ' ';

            node = node->next\_ptr;

        }

    }

    void push(char data)

    {

        Node \*node = new Node;

        node->data = data;

        if (head == nullptr)

            head = node;

        else if (tail != nullptr)

            tail->next\_ptr = node;

        tail = node;

        size++;

    }

    void pop()

    {

        if (tail == nullptr)

            return;

        if (head == tail)

        {

            delete tail;

            head = tail = nullptr;

            --size;

            return;

        }

        Node \*node = head;

        for (; node->next\_ptr != tail; node = node->next\_ptr)

            ;

        node->next\_ptr = nullptr;

        delete tail;

        tail = node;

        --size;

    }

    char top()

    {

        return tail->data;

    }

    void clear()

    {

        while (head != nullptr)

            pop();

    }

};

int main()

{

    setlocale(LC\_ALL, "ru");

    Stack stack;

    Stack stack2;

    int n;

    cout << "Введите количество элементов: ";

    cin >> n;

    char temp;

    for (int i = 0; i < n; ++i)

    {

        cout << "Введите " << i + 1 << "-ый элемент: ";

        cin >> temp;

        stack.push(temp);

    }

    do

    {

        cout << "Введите номер элемента, который необходимо удалить: ";

        cin >> temp;

    } while (int(temp) - int('0') - 1 < 0);

    int index = n - 1;

    while (index != int(temp) - int('0') - 1)

    {

        stack2.push(stack.top());

        stack.pop();

        index--;

    }

    stack.pop();

    n--;

    while (index != n)

    {

        stack.push(stack2.top());

        stack2.pop();

        index++;

    }

    stack.print();

    cout << endl;

    cout << "Введите сколько элементов добавить: ";

    cin >> temp;

    index = int(temp) - int('0') - 1;

    for (int i = 0; i < n; i++)

    {

        stack2.push(stack.top());

        stack.pop();

    }

    for (int i = 0; i <= index;i++)

    {

        cout << "Введите " << i+1 <<"-ый элемент: ";

        cin >> temp;

        stack.push(temp);

    }

    for (int i = 0; i < n; i++)

    {

        stack.push(stack2.top());

        stack2.pop();

    }

    stack.print();

    cout << endl;

    while (stack.size != 0)

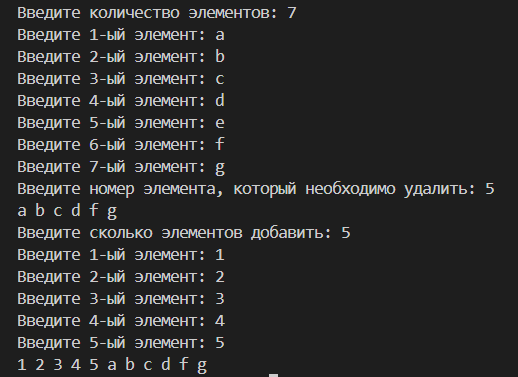
    {

        stack.pop();

    }

}

**Пример работы программы:**



**Очереди**

**Анализ задачи:**

1. Очередь работает по принципу «первый зашел – первый вышел», значит будем работать с «головой» списка (head).
2. Для добавления элемента в очередь используется метод push(), который создает новый элемент и привязывает конечный элемент списка (если такой есть) с созданным.
3. Метод front() возвращает самый первый элемент списка.
4. Удаление элемент реализовано методом pop(), который удаляет самый первый «головной» элемент списка.
5. Для удаления из середины очереди потребуется дополнительная очередь, куда будем записывать все промежуточные значения главной очереди.

**Код на С++:**

#include <iostream>

#include <clocale>

using namespace std;

struct Node

{

    char data;

    Node\* next\_ptr = nullptr;

};

struct Queue

{

    Node\* head = nullptr;

    Node\* tail = nullptr;

    int size = 0;

    void print()

    {

        Node\* node = head;

        while (node != nullptr)

        {

            cout << node->data << ' ';

            node = node->next\_ptr;

        }

        cout << endl;

    }

    void pop()

    {

        if (head == nullptr)

            return;

        if (head == tail)

        {

            delete tail;

            size--;

            head = tail = nullptr;

            return;

        }

        Node\* node = head;

        head = head->next\_ptr;

        delete node;

        size--;

    }

    void push(char data)

    {

        Node\* node = new Node;

        node->data = data;

        if (head == nullptr)

            head = node;

        if (tail != nullptr)

            tail->next\_ptr = node;

        tail = node;

        size++;

    }

    char front()

    {

        return head->data;

    }

    void clear()

    {

        while (head != nullptr)

            pop();

    }

};

int main()

{

    setlocale(LC\_ALL, "ru");

    Queue queue;

    cout << "Введите количество элементов: ";

    int temp\_int;

    cin >> temp\_int;

    char temp;

    for (int i = 0; i < temp\_int; ++i)

    {

        cout << "Введите " << i + 1 << "-ый элемент: ";

        cin >> temp;

        queue.push(temp);

    }

    cout << "Введите номер элемента, который необходимо удалить: ";

    cin >> temp\_int;

    temp\_int--; // в индексы

    Queue queue2;

    int temp\_size = queue.size;

    for (int i = 0; i < temp\_size; i++)

    {

        if (i == temp\_int)

        {

            queue.pop();

            continue;

        }

        queue2.push(queue.front());

        queue.pop();

    }

    for (int i = queue2.size; i > 0; i--)

    {

        queue.push(queue2.front());

        queue2.pop();

    }

    queue.print();

    cout << "Введите количество элементов для добавления в начало очереди: ";

    cin >> temp\_int;

    for (int i = queue.size; i > 0; i--)

    {

        queue2.push(queue.front());

        queue.pop();

    }

    for (int i = 0; i < temp\_int; i++)

    {

        cout << "Введите " << i + 1 << "-ый элемент: ";

        cin >> temp;

        queue.push(temp);

    }

    for (int i = queue2.size; i > 0; i--)

    {

        queue.push(queue2.front());

        queue2.pop();

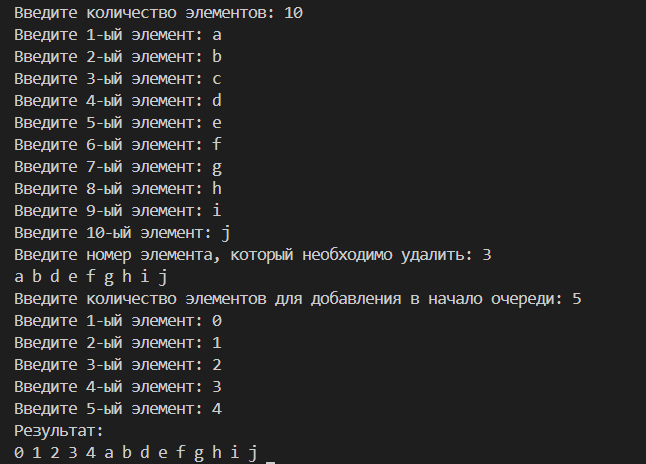
    }

    cout << "Результат: " << endl;

    queue.print();

}

**Пример работы программы:**



**GitHub:** *https://github.com/geroineee/PNRPU*