**МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ**

Федеральное государственное бюджетное образовательное учреждение   
высшего образования

**«Сибирский государственный университет науки и технологий   
имени академика М.Ф. Решетнева»**

Институт информатики и телекоммуникаций

Кафедра информатики и вычислительной техники

**ОТЧЕТ ПО ЛАБОРАТОРНОЙ РАБОТЕ**

Алгоритмы и структуры данных

|  |
| --- |
| Списки |

Руководитель В. В. Тынченко

подпись, дата инициалы, фамилия

Обучающийся БПИ22-02, 221219040 К.В. Трифонов

номер группы, зачетной книжки подпись, дата инициалы, фамилия

Красноярск 2023 г.

# Цель работы:

Изучение принципов организации и обработки списков как динамических структур данных.

# Постановка задачи.

Разработать шаблон класса «Двусвязный список», включающий в себя необходимый минимум методов, обеспечивающий полноценное функционирование объектов указанного класса при их использовании в программе, а именно:

1) конструкторы (по умолчанию, с параметрами, копирования);

2) деструктор;

3) добавление элемента в начало, конец, заданную (по номеру) позицию списка;

4) удаление элемента из начала, конца, заданной (по номеру) позиции списка;

5) поиск элемента – по значению и по номеру;

6) вывод списка на экран.

Разработать программу, содержащую меню, которое позволяет протестировать функции добавления, удаления, поиска и вывода на экран элементов списка.

В качестве отдельного пункта меню добавить решение задачи в соответствии со своим вариантом. При необходимости в разработанный шаблон класса добавить дополнительные методы, если того требует решение задачи.

# Вариант №21.

Построить два списка из целых чисел, вводимых с клавиатуры: список L1 и список L2. На основе списков L1 и L2 образовать список L3 из чисел, которые входят как в список L1, так и в список L2. Вывести список L3 на экран.

## Код **программы:**

# Описание класса узла *Node*:

*template*<*class* T>

*class* Node

{

        T value;

        Node<T>\* next;

        Node<T>\* prev;

*public:*

        T Value() { return value; }

*void* Set(T *\_value*) {value = *\_value*;}

        Node<T>*\** Next() { return next; }

*void* SetNext(Node<T>*\** *\_next*) {next = *\_next*;}

        Node<T>*\** Prev() { return prev; }

*void* SetPrev(Node<T>*\** *\_prev*) {prev = *\_prev*;}

};

Методы возврата/задания текущего значения и возврата/задания предыдущего и следующего узла.

# Описание класса списка *List*:

*//Класс списка*

*template*<*class* T>

*class* List

{

    Node<T>\* begin\_;

    Node<T>\* end\_;

*int* size\_;

*public:*

*//Конструкторы, деструкторы и операторы*

    List():begin\_(nullptr), end\_(nullptr), size\_(0){}

    List(*int* *S*):size\_(*S*){}

    List(*const* List*&* *obj*) { Copy(*obj*); }

    List<T>*&* operator = (*const* List*&* *obj*) { Copy(*obj*); return \**this*; }

    ~List() { Clear(); }

Поля начального и конечного узлов, конструкторы по умолчанию, с параметрами, копий и оператор присваивания.

*//Доступ к полю размера*

*int* size() { return size\_; }

*//Потоковый ввод*

*void* set(istream*&* *input*, *int* *size*){

*int* value;

        if (&*input* == &cin){

            cout<<"Введите элементы через пробел:\n";

        }

        for (*int* i = 0; i < *size*; i++){

*input* >> value;

            setEnd(value);

        }

    }

Функция, получающая на вход поток и выводит форматирование, если поток является стандартным потоком ввода cin. Заполнение производится посредством считывания строки элементов записанных через пробел.

*//Потоковый вывод*

*void* get(ostream*&* *out*){

        if (size\_ == 0 && &*out* == &cout){

            cout<<"Пустой список.\n";

        }

        else{

            if (&*out* == &cout)

                cout<<"Список длинной " <<size\_<<":\n";

            Node<T>\* t = begin\_;

            for (*int* i = 0; i < size\_; i++){

*out* << t->Value() << " ";

                t = t->Next();

            }

*out* << "\n";

        }

    }

Функция, получающая на вход поток и выводит данные с форматированием, если поток является стандартным потоком вывода cout.

*//Добавить в начало списка*

*void* setBegin(T *\_value*){

        Node<T>\* t = new Node<T>;

        t->Set(*\_value*);

        t->SetPrev(nullptr);

        t->SetNext(begin\_);

        if (size\_ > 0){

            begin\_->SetPrev(t);

            begin\_ = t;

        }

        else{

            begin\_ = end\_ = t;

        }

        size\_++;

    }

*//Добавить в конец списка*

*void* setEnd(T *\_value*){

        Node<T>\* t = new Node<T>;

        t->Set(*\_value*);

        t->SetPrev(end\_);

        t->SetNext(nullptr);

        if (end\_ != nullptr)

            end\_->SetNext(t);

        if (size\_ == 0)

            begin\_ = end\_ = t;

        else{

            end\_ = t;

        }

        size\_++;

    }

*//Добавить по индексу*

*void* setIndex(*int* *index*, T *\_value*){

        if (*index* == size\_){

            setEnd(*\_value*);

            return;

        }

        if (*index* == 0){

            setBegin(*\_value*);

            return;

        }

        Node<T>\* itemPrev = getIndex(*index* - 1);

        Node<T>\* item = getIndex(*index*);

        Node<T>\* t = new Node<T>;

        t->Set(*\_value*);

        t->SetNext(item);

        t->SetPrev(itemPrev);

        itemPrev->SetNext(t);

        item->SetPrev(t);

        size\_++;

    }

*//Доступ к элементу по индексу*

    Node<T>*\** getIndex(*int* *index*){

        Node<T>\* t = begin\_;

        for (*int* i = 0; i < *index*; i++)

            t = t->Next();

        return t;

    }

*//Доступ к значению элемента по индексу*

    T GetValue(*int* *index*){

        Node<T>\* t = getIndex(*index*);

        return t->Value();

    }

*//Удалить элемент по индексу*

*void* DelIndex(*int* *index*){

        Node<T>\* item = getIndex(*index*);

        Node<T>\* itemPrev = item->Prev();

        Node<T>\* itemNext = item->Next();

        if ((size\_ > 1) && (itemPrev != nullptr))

            itemPrev->SetNext(itemNext);

        if ((itemNext != nullptr) && (size\_ > 1))

            itemNext->SetPrev(itemPrev);

        if (*index* == 0)

            begin\_ = itemNext;

        if (*index* == size\_ - 1)

            end\_ = itemPrev;

        delete item;

        size\_--;

    }

*//Удалить начальный элемент*

*void* Delete\_start() { return DelIndex(0); }

*//Удалить конечный элемент*

*void* Delete\_end() { return DelIndex(size\_ - 1); }

*//Удалить список*

*void* Clear(){

*int* n = size\_;

        for (*int* i = 0; i < n; i++)

            DelIndex(0);

    }

Функции удаления одного узла из списка и всего списка (очистка для деструктора или перед заполнением из копии).

*//Метод копирования списков*

*void* Copy(*const* List<T>*&* *obj*){

        Clear();

        Node<T>\* t = *obj*.begin;

        while (t != nullptr){

            setBegin(t->Value());

            t = t->Next();

        }

    }

};

Применяется в конструкторе копирования и операторе присваивания.

*int* search(T *value*, *bool* *echo*){

*int* c = 0;

        for (*int* i = 0; i < size\_; i++){

            if (GetValue(i) == *value*){

                if(!*echo*)

                    cout<<"Найден элемент №"<<i<<":"<<" "<<*value*<<"\n";

                c++;

            }

        }

        if (!*echo*){

            if (c)

                cout<<"Всего таких элементов "<<c<<".\n";

            else

                cout<<"Не найдено ни одного такого элемента.\n";

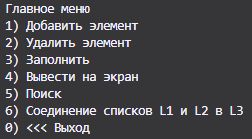
        }

        return c;

    }

Метод поиска значения в списке на вход получает значение поиска и bool переменную, определяющую будут ли выводиться информационные данные в консоль. Возвращает количество найденных элементов (используется для избегания повторных записей уже записанных элементов).

# Меню файла main.cpp:



**Первый режим работы программы:**

case 1:{

                cout<<"1) Добавить в начало\n"

                <<"2) Добавить в конец\n"

                <<"3) Добавить по индексу\n";

*int* sw1;

                cin>>sw1;

                switch (sw1){

                    case 1:{

*int* temp;

                        cout<<"Укажите значение:\n";

                        cin>>temp;

                        L.setBegin(temp);

                        break;

                    }

                    case 2:{

*int* temp;

                        cout<<"Укажите значение:\n";

                        cin>>temp;

                        L.setEnd(temp);

                        break;

                    }

                    case 3:{

*int* temp, ind;

                        cout<<"Укажите значение и индекс:\n";

                        cin>>temp>>ind;

                        L.setIndex(temp,ind);

                        break;

                    }

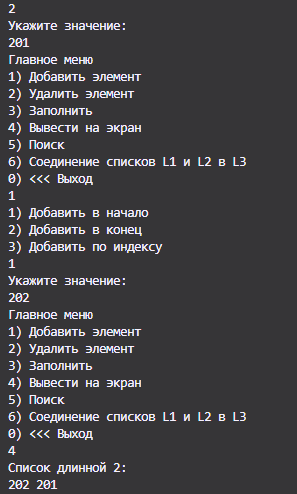
                }

                break;

            }

По выбору пользователя выполняется нужная операция с добавлением элемента к списку.

Пример работы режима с заполнением списка двумя числами:



**Второй режим работы программы:**

case 2:{

                cout<<"1) Удалить в начале\n"

                <<"2) Удалить в конеце\n"

                <<"3) Удалить по индексу\n";

*int* sw2;

                cin>>sw2;

                switch (sw2){

                    case 1:{

                        L.Delete\_start();

                        break;

                    }

                    case 2:{

                        L.Delete\_end();

                        break;

                    }

                    case 3:{

*int* ind;

                        cout<<"Укажите индекс:\n";

                        cin>>ind;

                        L.DelIndex(ind);

                        break;

                    }

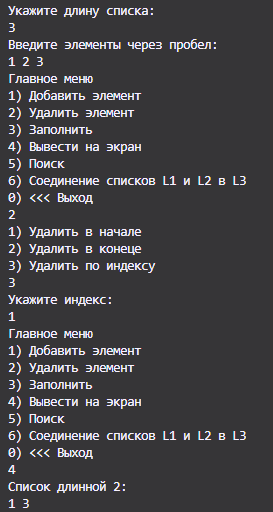
                }

                break;

            }

По выбору пользователя выполняется нужная операция с удалением элемента в списке.

Пример работы режима с удалением второго элемента в списке из трёх элементов.



**Третий режим работы программы:**

case 3:{

*int* size;

                L.Clear();

                cout<<"Укажите длину списка:\n";

                cin>>size;

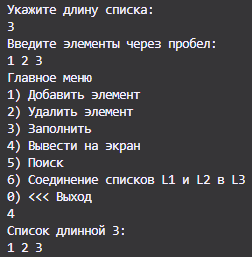
                L.set(cin, size);

                break;

            }

Список очищается и пользователю предлагается ввести длину списка, затем через метод set происходит заполнение списка.

Пример работы режима при заполнении списка трема элементами:



**Четвертый режим работы программы:**

case 4:{

                L.get(cout);

                break;

            }

Вызов метода вывода списка на экран.

**Пятый режим работы программы:**

            case 5:{

*int* temp;

                cout<<"Введите элемент поиска:\n";

                cin>>temp;

                L.search(temp,0);

                break;

            }

Считывание элемента поиска и вызов метода поиска этого элемента в списке.

**Шестой режим работы программы (по варианту):**

            case 6:{

*int* s1, s2;

                List<*int*> L1, L2, L3;

                cout<<"Укажите длину списков L1 и L2:\n";

                cin>>s1>>s2;

                cout<<"Заполните список L1:\n";

                L1.set(cin, s1);

                cout<<"Заполните список L2:\n";

                L2.set(cin, s2);

*int* c = 0;

*//Матрица сравнений*

                for (*int* j = 0; j < L2.size(); j++){

                    for (*int* i = 0; i < L1.size(); i++){

                        if (L1.GetValue(i) == L2.GetValue(j) && !L3.search(L1.GetValue(i), 1)){

                            L3.setEnd(L1.GetValue(i));

                        }

                    }

                }

                cout<<"Список чисел, которые входят как в L1, так и в L2:\n";

                L3.get(cout);

                break;

            }

Создаются три объекта типа список L1, L2, L3, далее L1 и L2 заполняются с консоли и выполняется поиск одинаковых в них элементов через матрицу сравнений. Дубликаты отсеиваются с помощью поиска элемента в списке ответов L3.

# Ответы на контрольные вопросы

1. *Опишите принцип организации списковых структур. В чем заключается преимущество подобной организации данных?*

Списковые структуры представляют собой набор элементов, связанных ссылками, что обеспечивает динамичность и более эффективные операции добавления и удаления по сравнению с массивами.

1. *Схематично изобразите порядок добавления нового узла в начало, в конец и в середину односвязного списка.*

В начало - создание нового узла с указанием следующего элемента; в конец - обновление указателя последнего узла; в середину - изменение указателей соседних узлов.

1. *Схематично изобразите порядок удаления узла из начала, из конца и из середины односвязного списка.*

Из начала - обновление указателя на первый элемент; из конца - обновление указателя последнего узла и освобождение памяти; из середины - изменение указателей соседних узлов.

1. *Опишите принцип организации и дисциплину обслуживания стека, реализованного на основе односвязного списка.*

Стек использует принцип "последний вошел, первый вышел" (LIFO). Операция push добавляет элемент в начало списка, а операция pop удаляет элемент с начала.

1. *Опишите принцип организации и дисциплину обслуживания очереди, реализованной на основе односвязного списка.*

Очередь использует принцип "первый вошел, первый вышел" (FIFO). Операция enqueue добавляет элемент в конец списка, а операция dequeue удаляет элемент с начала.

1. *В чем состоит отличие организации двусвязного списка от односвязного?*

Двусвязный список содержит узлы, каждый из которых имеет две ссылки: на предыдущий и следующий узлы, что обеспечивает более эффективные операции вставки и удаления в середине списка.

1. *Схематично изобразите порядок добавления нового узла в начало, в конец и в середину двусвязного списка.*

*Ответ:* В начало - создание нового узла и обновление указателя на следующий узел; в конец - обновление указателя последнего узла; в середину - изменение указателей соседних узлов.

1. **Удаление узла из двусвязного списка:**
   * *Вопрос:* Каким образом происходит удаление узла из начала, из конца и из середины двусвязного списка?
   * *Ответ:* Из начала - обновление указателя на следующий узел и освобождение памяти; из конца - обновление указателя последнего узла и освобождение памяти; из середины - изменение указателей соседних узлов и освобождение памяти.
2. **Проход по двусвязному списку:**
   * *Вопрос:* Как осуществляется проход по двусвязному списку справа налево и слева направо?
   * *Ответ:* Справа налево - начиная с последнего узла, идем к началу по указателям prev; слева направо - начиная с первого узла, идем к концу по указателям next.
3. **Двусторонняя очередь (дек) на основе двусвязного списка:**

* *Вопрос:* Как организуется двусторонняя очередь (дек) на основе двусвязного списка, и как выполняются операции добавления и удаления с обоих концов?
* *Ответ:* Дек использует двусвязный список. Операции добавления элемента могут выполняться и в начало, и в конец, а операции удаления - как с начала, так и с конца списка.

# Вывод

Несмотря на родство shaker sort и insert sort, второй является более эффективным, за счёт меньшего числа перестановок элементов массива (на длине 20000 эл у shaker sort около 150 миллионов сравнений, у insert sort около 85 миллионов). Также insert sort имеет большую степень естественности по сравнению с shaker sort.