Министерство науки и высшего образования Российской Федерации

Федеральное государственное автономное учреждение высшего образования

«Пермский национальный исследовательский политехнический университет»

ПНИПУ

**Лабораторная работа по ООП**

**«№6»**

Выполнил:

студент группы РИС-23-2б

Ившин Максим Сергеевич

Проверила:

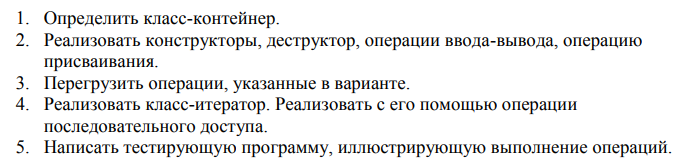
доцент кафедры ИТАС

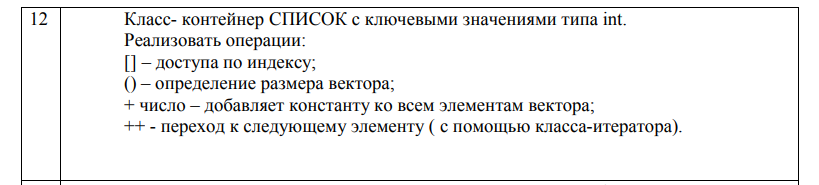
О.А. Полякова

2024 г.

**Разработка алгоритма**

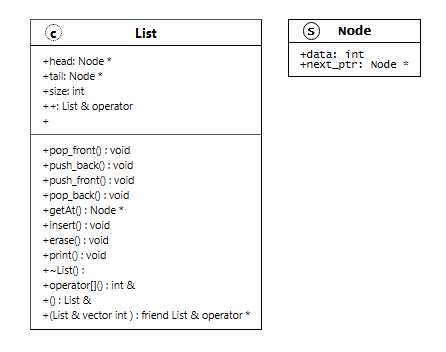
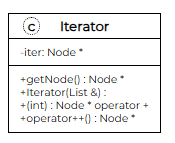
**Постановка задачи:**





**Анализ задачи:**

1. Создадим несколько файлов заголовков и файлов с кодом для удобной работы.
2. Создадим класс-контейнер список List, который будет хранить указатели на головной элемент и конечный элемент списка.
3. Список будет использовать дополнительную структуру Node для реализации узлов списка.
4. Для удобного прохода по списку реализуем итератор Iterator, оперирующий указателями на узлы списка.
5. В главном файле программы создадим список и продемонстрируем все реализованные операции.



Код на C++:

main.cpp:

#include <iostream>

#include <vector>

#include "list.h"

#include "list.cpp"

#include "iterator.h"

#include "iterator.cpp"

using namespace std;

int main()

{

     List list;

     for (int i = 0; i < 5; ++i) // заполнение списка

          list.push\_back(rand() % 100);

     list.print();

     cout << list[2] << endl;

     list(10);

     list += 10;

     list.print();

}

iterator.h:

#pragma once

using namespace std;

#include "List.h"

class Iterator

{

private:

    Node \*iter;

public:

    Node \*getNode();

    Iterator(List &a);

    Node \*operator+(int a);

    Node \*operator++();

    Node \*operator++(int);

};

list.h:

#pragma once

#include <iostream>

#include <vector>

using namespace std;

struct Node

{

    int data;

    Node \*next\_ptr = nullptr;

};

class List

{

public:

    Node \*head = nullptr;

    Node \*tail = nullptr;

    int size = 0;

    void pop\_front();

    void push\_back(int);

    void push\_front(int);

    void pop\_back();

    Node \*getAt(int);

    void insert(int, int);

    void erase(int);

    void print();

    ~List();

    int &operator[](int);

    List &operator()(int);

    List &operator+=(int);

    friend List &operator\*(List &a, vector<int> b);

};

list.cpp:

#pragma once

#include <iostream>

#include <vector>

#include "list.h"

#include "iterator.h"

using namespace std;

void List::pop\_front()

{

    if (head == nullptr)

        return;

    if (head == tail)

    {

        delete tail;

        head = tail = nullptr;

        return;

    }

    Node \*node = head;

    head = node->next\_ptr;

    delete node;

    --size;

}

void List::push\_back(int data)

{

    Node \*node = new Node;

    node->data = data;

    if (head == nullptr)

        head = node;

    if (tail != nullptr)

        tail->next\_ptr = node;

    tail = node;

    ++size;

}

void List::push\_front(int data)

{

    Node \*node = new Node;

    node->data = data;

    node->next\_ptr = head;

    head = node;

    if (tail == nullptr)

        tail = node;

    ++size;

}

void List::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;

    --size;

}

Node \*List::getAt(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 List::insert(int k, int data)

{

    Node \*left = getAt(k);

    if (left == nullptr)

        return;

    Node \*right = left->next\_ptr;

    Node \*node = new Node;

    node->data = data;

    left->next\_ptr = node;

    node->next\_ptr = right;

    if (right == nullptr)

        tail = node;

    size++;

}

void List::erase(int k)

{

    if (k < 0)

        return;

    if (k == 0)

    {

        pop\_front();

        return;

    }

    Node \*left = getAt(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;

    --size;

}

void List::print()

{

    for (Iterator iterator(\*this); iterator.getNode() != nullptr; iterator++) // вывод

        cout << iterator.getNode()->data << ' ';

    cout << endl

         << endl;

}

List &List::operator+=(int k)

{

    for (Iterator iterator(\*this); iterator.getNode() != nullptr; iterator++) // вывод

        iterator.getNode()->data += k;

    return \*this;

}

List::~List()

{

    while (head != nullptr)

        pop\_front();

}

int &List::operator[](int a)

{

    return getAt(a)->data;

}

List &List::operator()(int k)

{

    while (size > k)

    {

        this->pop\_back();

        size--;

    }

    while (size < k)

    {

        this->push\_back(0);

    }

    return \*this;

}

iterator.cpp:

#pragma once

#include <iostream>

#include <string>

#include "iterator.h"

#include "list.h"

using namespace std;

Node \*Iterator::getNode()

{

    return this->iter;

}

Iterator::Iterator(List &a)

{

    this->iter = a.head;

}

Node \*Iterator::operator+(int a)

{

    Node \*iter = this->iter;

    for (int i = 0; i < a; ++i)

        iter = iter->next\_ptr;

    return iter;

}

Node \*Iterator::operator++()

{

    iter = iter->next\_ptr;

    return iter;

}

Node \*Iterator::operator++(int)

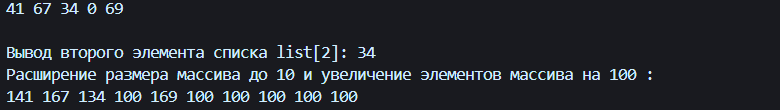
{

    iter = iter->next\_ptr;

    return iter;

}

Пример работы программы:



**Контрольные вопросы**

1. **Что такое абстрактный тип данных? Привести примеры**

**АТД.**

АТД - тип данных, определяемый только через операции, которые могут выполняться над соответствующими объектами безотносительно к способу представления этих объектов. Примеры АТД включают строки, списки, деревья, множества, стеки очереди.

1. **Привести примеры абстракции через параметризацию.**

Абстракция через параметризацию достигается путем создания шаблонов классов или функций, которые могут работать с различными типами данных. Например, шаблонный класс vector<T> в C++ является примером абстракции через параметризацию, так как он может хранить элементы любого типа T.

Абстракция через параметризацию позволяет, используя параметры, представить фактически неограниченный набор различных вычислений одной программой, которая есть абстракция этих наборов.

Например у нас существует процедура сортировки массива:

Procedure Sort(Var A: TArray; N: Word);

Если эта процедура получает сортируемый массив в качестве параметра, то может быть вызвана любое количество раз, причем для различных массивов. Код процедуры абстрагируется от конкретики данных, которые она обрабатывает. Абстракция через параметризацию - весьма полезное средство.

1. **Привести примеры абстракции через спецификацию.**

Абстракция через спецификацию означает определение интерфейса для класса или функции без раскрытия деталей реализации. В C++ это часто достигается с помощью чисто виртуальных методов в абстрактных базовых классах.

Абстракция через спецификацию позволяет абстрагироваться от процесса вычислений описанных в теле процедуры, до уровня знания того, что данная процедура делает. Это достигается путем задания спецификации, описывающей эффект ее работы, после чего смысл обращения к данной процедуре становится ясным через анализ этой спецификации, а не самого тела процедуры. Мы пользуемся абстракцией через спецификацию всякий раз, когда связываем с процедурой некий комментарий, достаточно информативный для того, чтобы иметь возможность работать без анализа

тела процедуры. Спецификация описывает соглашение между разработчиками и пользователями. Разработчик берется написать модуль, а пользователь соглашается не полагаться на знания о том, как именно этот модуль реализован, т.е. не предполагать ничего такого, что не было бы указано в спецификации.

1. **Что такое контейнер? Привести примеры.**

Контейнер – набор однотипных элементов. Встроенные массивы в С++

* частный случай контейнера

1. **Какие группы операций выделяют в контейнерах?**

Среди всех операций контейнера можно выделить несколько типовых групп:

* Операции доступа к элементам, которые обеспечивают и операцию замены значений элементов;
* Операции добавления и удаления элементов или групп

элементов;

* Операции поиска элементов и групп элементов;
* Операции объединения контейнеров;
* Специальные операции, которые зависят от вида контейнера.

1. **Какие виды доступа к элементам контейнера существуют?**

**Привести примеры.**

Доступ к элементам контейнера бывает: последовательный, прямой и ассоциативный.

Прямой доступ – это доступ по индексу. Например, a[10] – требуется найти элемент контейнера с номером 10.

Ассоциативный доступ также выполняется по индексу, но индексом будет являться не номер элемента, а его содержимое. Пусть имеется контейнер –словарь, в котором хранится информация, состоящая, как минимум из двух полей: слово и его перевод. Индексом может служить слово, например, a[“word”].

При последовательном доступе осуществляется перемещение от элемента к элементу контейнера. Набор операций последовательного доступа включает следующие:

* Перейти к первому элементу;
* Перейти к последнему элементу;
* Перейти к следующему элементу;
* Перейти к предыдущему элементу;
* Перейти на n элементов вперед;
* Перейти на n элементов назад;
* Получить текущий элемент.

1. **Что такое итератор?**

Итератор – это объект, который обеспечивает последовательный доступ к элементам контейнера.

1. **Каким образом может быть реализован итератор?**

Итератор может быть реализован как часть класса-контейнера в виде набора методов:

v.first() перейти к первому элементу

v.last() перейти к последнему элементу

v.next() перейти к следующему элементу

v.prev() перейти к предыдущему элементу v.skip(n) перейти на n элементов вперед v.skip(-n) перейти на n элементов назад

v.current() получить текущий элемент

1. **Каким образом можно организовать объединение контейнеров?**

* Простое сцепление двух контейнеров: в новый контейнер попадают сначала элементы первого контейнера, потом второго, операция не коммутативна.
* Объединение упорядоченных контейнеров, новый контейнер тоже будет упорядочен, операция коммутативна.
* Объединение контейнеров как объединение множеств, в новый контейнер попадают только те элементы, которые есть хотя бы в одном контейнере, операция коммутативна.
* Объединение контейнеров как пересечение множеств, в новый контейнер попадают только те элементы, которые есть в обоих контейнерах,

операция коммутативна.

* Для контейнеров-множеств может быть еще реализована операция вычитания, в контейнер попадают только те элементы первого контейнера, которых нет во втором, операция не коммутативна.
* Извлечение части элементов из контейнера и создание нового контейнера. Эта операция может быть выполнена с помощью конструктора, а часть контейнера задается двумя итераторами.

1. **Какой доступ к элементам предоставляет контейнер,**

**состоящий из элементов «ключ-значение»?**

Ассоциативный доступ.

1. **Как называется контейнер, в котором вставка и удаление элементов выполняется на одном конце контейнера?**

Стек.

1. **Какой из объектов (a, b, c, d) является контейнером?**
2. **int mas=10;**
3. **2. int mas;**
4. **3. struct {char name[30]; int age;} mas;**
5. **4. int mas[100];**

Ответ: d

1. **Какой из объектов (a,b,c,d) не является контейнером?**
2. **int a[]={1,2,3,4,5};**
3. **2. int mas[30];**
4. **3. struct {char name[30]; int age;} mas[30];**
5. **4. int mas;**

Ответ: d

1. **Контейнер реализован как динамический массив, в нем определена операция доступ по индексу. Каким будет доступ к элементам контейнера?**

Прямой доступ.

1. **Контейнер реализован как линейный список. Каким будет доступ к элементам контейнера?**

Последовательный доступ.

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