Федеральное государственное автономное образовательное учреждение высшего образования «Пермский национальный исследовательский политехнический университет»

Лабораторная работа №6
«АТД. Контейнеры»

Выполнил:
студент первого курса
ЭТФ группы РИС-23-3б
Акбашева Софья Руслановна

Проверила:
Доцент кафедры ИТАС О. А. Полякова

Пермь, 2024

АТД. Контейнеры

Цель задания

1) Создание консольного приложения, состоящего из нескольких файлов в системе программирования Visual Studio.

2) Реализация класса-контейнера.

Постановка задачи

1. Определить класс-контейнер.
2. Реализовать конструкторы, деструктор, операции ввода-вывода, операцию присваивания.
3. Перегрузить операции, указанные в варианте.
4. Реализовать класс-итератор. Реализовать с его помощью операции последовательного доступа.
5. Написать тестирующую программу, иллюстрирующую выполнение операций.

Задание

Класс- контейнер СПИСОК с ключевыми значениями типа int. Реализовать операции: [] – доступа по индексу;

int() – определение размера списка;

* вектор – умножение элементов списков a[i]*b[i];

+n - переход вправо к элементу с номером n (с помощью класса-итератора).

Анализ задачи

1) Для реализации класса список необходимо создать структуру Node.

```
struct Node {
	int data; //данные
	Node* ptr_to_prev_node = nullptr; //указатель на предыдущий элемент
	Node* ptr_to_next_node = nullptr; //указатель на следующий элемент
};
```

2) В классе список необходимо реализовать конструкторы, деструктор, функции вставки/ удаления первого/последнего элемента, а также операторы перегрузки.

```
class List {
```

```
private:
	int size; //размер
	Node* head; //указатель на голову
	Node* tail; //указатель на хвост
	Iterator begin;//указатель на начальный элемент
	Iterator end;//указатель на последний элемент
public:
	List(int size); //конструктор с длиной
	List(int size, int data);//конструктор с длиной и данными
	List(const List& list);//конструктор копирования
	~List();//деструктор

	void push_bask(int data);//вставка элемента в конец
	void push_front(int data);//вставка элемента в НАЧАЛО

	Iterator first_el();//последний элемент
	Iterator last_el();//первый элемент

	int pop_bask();//удалить последний элемент
	int pop_front();//удалить первый элемент

	bool is_empty();//список пустой иль нет

	int operator () ();//размер списка
	int& operator[](int index);//данные по индексу
	List& operator = (const List& list);//оператор присваивания
	List operator *(List& list);

	friend ostream& operator << (ostream& stream, const List& list);//ВЫВОД
	friend istream& operator >> (istream& stream, const List& list);//ВВОД
};
```

3) Операции последовательного доступа можно реализовать посредством создания класса итератора. В классе должны быть конструкторы, деструктор, а также перегрузки операций.

[] – доступа по индексу;

int() – определение размера списка;

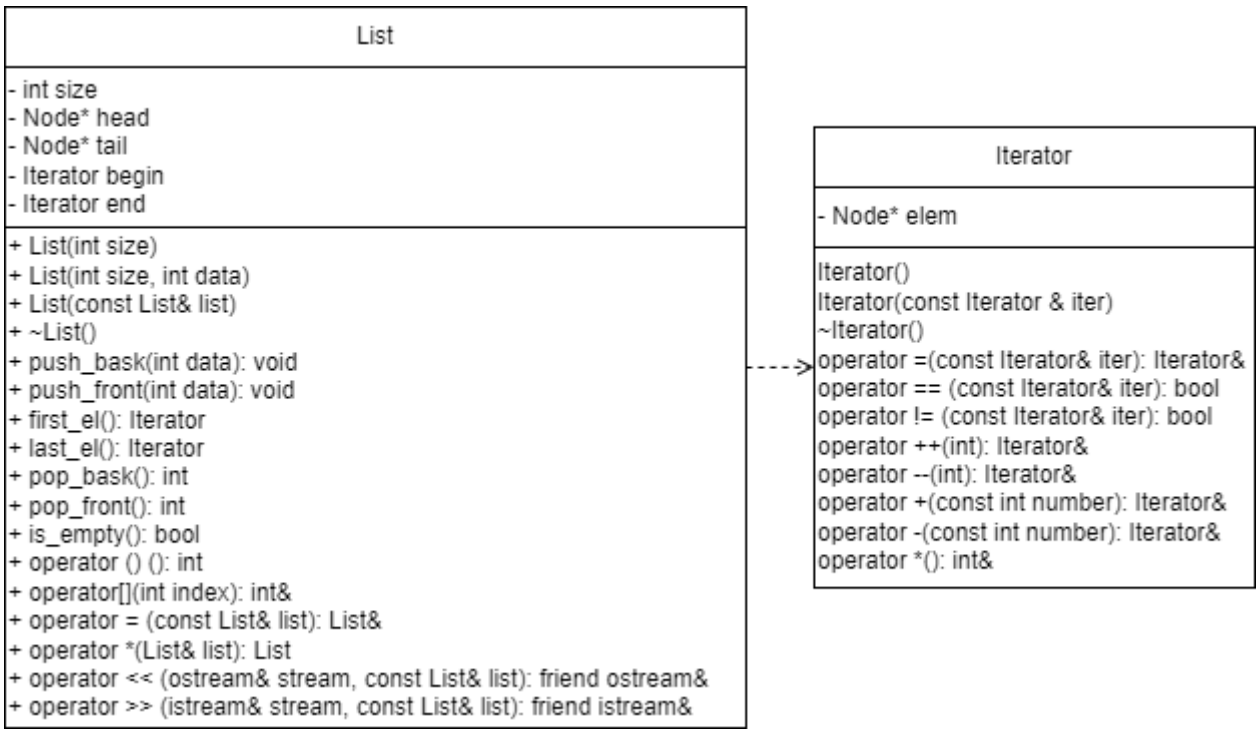
* вектор – умножение элементов списков a[i]*b[i];

+n - переход вправо к элементу с номером n (с помощью класса-итератора).

4) В классе Iterator необходимо указать, что класс List – дружественный.

5) В главной функции main необходимо продемонстрировать все операции и функции.

UML диаграмма



Код

Файл Class_6.cpp

```
#include "list.h"
#include "iterator.h"
#include <iostream>
#include <string>

using namespace std;

void creature_list_rand(List& list);
void print_list_iterator_start(List& list);
void print_list_iterator_end(List& list);

int main(){
   system("chcp 1251 > Null");
   srand(time(0));

   List list_1(10, 0);
   cout << "list_1" << endl;
   cout << list_1 << endl << endl;

   creature_list_rand(list_1);
   cout << "list_1" << endl;
   cout << list_1 << endl << endl;
   cout << "Первый элемент: " << *(list_1.first_el()) << endl;
   cout << "Последний элемент: " << *(list_1.last_el()) << endl << endl;

   List list_2(list_1);
   cout << "list_2" << endl;
   cout << list_2 << endl << endl;

   list_2.push_front(100);
   list_2.pop_bask();
   cout << list_2 << endl << endl;

   cout << "Первый элемент: " << *(list_2.first_el()) << endl;
```

```
    cout << "Последний элемент: " << *(list_2.last_el()) << endl << endl;

    List list_3(5);
    cout << "list_3" << endl;
    cin >> list_3;
    cout << list_3 << endl << endl;

    cout << "Первый элемент: " << *(list_3.first_el()) << endl;
    cout << "Последний элемент: " << *(list_3.last_el()) << endl << endl;

    List list_4 = list_2 * list_3;
    cout << "list_4" << endl;
    cout << list_4 << endl << endl;

    cout << "Первый элемент: " << *(list_4.first_el()) << endl;
    cout << "Последний элемент: " << *(list_4.last_el()) << endl << endl;

    cout << "Вывод элементов list_1 с помощью Iterator с НАЧАЛА" << endl;
    print_list_iterator_start(list_1);
    cout << endl;

    cout << "Вывод элементов list_1 с помощью Iterator с КОНЦА" << endl;
    print_list_iterator_end(list_1);
    cout << endl << endl;

    cout << "Вывод элемента 2 list_3 с помощью Iterator с НАЧАЛА" << endl;
    Iterator iter = list_3.first_el();
    cout << *(iter + 1);
    cout << endl;

    cout << "Вывод элемента 2 list_3 с помощью Iterator с КОНЦА" << endl;
    iter = list_3.last_el();
    cout << *(iter - 3);
    cout << endl << endl;

    cout << "Сравнение элемента 2 и 3 list_3 с помощью Iterator" << endl;
    iter = list_3.first_el();
    if (*(iter + 1) == *(iter + 2)) {
        cout << *(iter + 1) << " == " << *(iter + 2) <<" элементы равны";
    }
    else {
        cout << *(iter + 1) << " != " << *(iter + 2) << " элементы не равны";
    }
    cout << endl;

    return 0;
}

void creature_list_rand(List& list) {
    for (int i = 0; i < list(); i++) {
        list[i] = rand() % (100 + 1) - 50;
    }
}

void print_list_iterator_start(List& list) {
    for (Iterator iter = list.first_el(); iter != list.last_el() + 1; iter++) {
        cout << *iter << ' ';
    }
}

void print_list_iterator_end(List& list) {
    for (Iterator iter = list.last_el(); iter != list.first_el() - 1; iter--) {
```

```
        cout << *iter << ' ';
    }
}
```

Файл list.h

```
#pragma once //предотвращает повторную загрузку заголовочного файла, если он уже был включен
#include <iostream> //стандартные потоки ввода и вывода
#include <string>
#include "iterator.h"
using namespace std;

struct Node {
        int data; //данные
        Node* ptr_to_prev_node = nullptr; //указатель на предыдущий элемент
        Node* ptr_to_next_node = nullptr; //указатель на следующий элемент
};

class List {
private:
        int size; //размер
        Node* head; //указатель на голову
        Node* tail; //указатель на хвост
        Iterator begin;//указатель на начальный элемент
        Iterator end;//указатель на последний элемент
public:
        List(int size); //конструктор с длиной
        List(int size, int data);//конструктор с длиной и данными
        List(const List& list);//конструктор копирования
        ~List();//деструктор

        void push_bask(int data);//вставка элемента в конец
        void push_front(int data);//вставка элемента в НАЧАЛО

        Iterator first_el();//последний элемент
        Iterator last_el();//первый элемент

        int pop_bask();//удалить последний элемент
        int pop_front();//удалить первый элемент

        bool is_empty();//список пустой иль нет

        int operator () ();//размер списка
        int& operator[](int index);//данные по индексу
        List& operator = (const List& list);//оператор присваивания
        List operator *(List& list);

        friend ostream& operator << (ostream& stream, const List& list);//ВЫВОД
        friend istream& operator >> (istream& stream, const List& list);//ВВОД
};
```

Файл list.cpp

```
#include "list.h"
#include "iterator.h"
#include <iostream>
#include <string>
using namespace std;

List::List(int size) {//конструктор с длиной
        this->size = size;
        if (size > 0) {//если длина больше 0
```

```
                Node* node = new Node;//создаю узел
                this->head = node; //указатель на голову
                this->tail = node; //указатель на хвост
                for (int i = 1; i < size; i++) {//заполняю со второго номера
                        Node* New_Node = new Node;//новый узел
                        tail->ptr_to_next_node = New_Node;//хвост указывает на новый узел
                        New_Node->ptr_to_prev_node = tail;//новый узел ук-т на хвост как на пред-й
                        tail = New_Node;//новый узел становится хвостом
                }
                tail->ptr_to_next_node = nullptr;//до хвоста элементов нет
        }
        else {
                this->head = nullptr; //указатель на голову
                this->tail = nullptr; //указатель на хвост
        }
        this->begin.elem = this->head;
        this->end.elem = this->tail;
}

List::List(int size, int data) {//конструктор с длиной и данными
        this->size = size;
        if (size > 0) {//если длина больше 0
                Node* node = new Node;//создаю узел
                node->data = data; //данные для узла
                this->head = node; //указатель на голову
                this->tail = node; //указатель на хвост
                for (int i = 1; i < size; i++) {//заполняю со второго номера
                        Node* New_Node = new Node;//новый узел
                        New_Node->data = data; //данные для нового узла
                        tail->ptr_to_next_node = New_Node;//хвост указывает на новый узел
                        New_Node->ptr_to_prev_node = tail;//новый узел ук-т на хвост как на пред-й
                        tail = New_Node;//новый узел становится хвостом
                }
                tail->ptr_to_next_node = nullptr;//до хвоста элементов нет
        }
        else {
                this->head = nullptr; //указатель на голову
                this->tail = nullptr; //указатель на хвост
        }
        this->begin.elem = this->head;
        this->end.elem = this->tail;
}

List::List(const List& list) {//конструктор копирования
        this->head = nullptr; //указатель на голову
        this->tail = nullptr; //указатель на хвост
        this->size = 0;

        Node* current_node = list.head;//создаю узел
        while (current_node != nullptr) {//пока не пройду весь список
                push_bask(current_node->data);//добавляю в конец эл-т
                current_node = current_node->ptr_to_next_node;//беру следующий элемент
        }
        this->begin.elem = this->head;
        this->end.elem = this->tail;
}

List::~List() {//деструктор
        Node* current_node = head;//голова
        while (current_node != nullptr) {//пока не пройду весь список
                Node* next_node = current_node->ptr_to_next_node;//следующий элемент
                delete current_node;//удаляю текущий элемент
```

```
                current_node = next_node;//беру новый элемент
        }
        head = nullptr; //голова пустая
}

void List::push_bask(int data) {//вставка элемента в конец
        Node* New_Node = new Node;//новый узел
        New_Node->data = data; //данные для нового узла
        New_Node->ptr_to_next_node = nullptr;//новый узел пока никуда не указывает
        if (this->head == nullptr) {//если список пустой
                this->head = New_Node;//новый узел - голова
                this->tail = New_Node;//новый узел - хвост
                this->begin.elem = this->head;
                this->end.elem = this->tail;
        }
        else {
                tail->ptr_to_next_node = New_Node;//хвост указывает на узел
                New_Node->ptr_to_prev_node = tail;
                tail = New_Node; ////новый узел - хвост
                this->end.elem = this->tail;
        }
        this->size++;//увеличиваю длину
}

void List::push_front(int data) {//вставка элемента в НАЧАЛО
        Node* New_Node = new Node;//новый узел
        New_Node->data = data; //данные для нового узла

        if (this->head == nullptr) {//если список пустой
                this->head = New_Node;//новый узел - голова
                this->tail = New_Node;//новый узел - хвост
                this->begin.elem = this->head;
                this->end.elem = this->tail;
        }
        else {
                head->ptr_to_prev_node = New_Node;//голова указывает на узел
                New_Node->ptr_to_next_node = head;
                head = New_Node; //новый узел - голова
                this->begin.elem = this->head;
        }
        this->size++;//увеличиваю длину
}

Iterator List::first_el() {//первый элемент
        return this->begin;
}

Iterator List::last_el() {//первый элемент
        return this->end;
}

int List::pop_bask() {//удаление последнего
        int temp;
        if (this->tail != nullptr) {//если список не пустой
                Node* current_node = this->tail;//узел предыдущий
                tail = current_node->ptr_to_prev_node;//меняю хвост
                temp = current_node->data;
                tail->ptr_to_next_node = nullptr;
                this->size--;//уменьшаю размер
                this->end.elem = tail;
        }
        return temp;//значение из удаленной переменной
```

```
}

int List::pop_front() {//удаление первого
	int temp;
	if (this->tail != nullptr) {//если список не пустой
		Node* current_node = this->head;//узел предыдущий
		head = current_node->ptr_to_next_node;//меняю хвост
		temp = current_node->data;
		head->ptr_to_prev_node = nullptr;
		this->size--;//уменьшаю размер
		this->begin.elem = head;
	}
	return temp;//значение из удаленной переменной
}

bool List::is_empty() {
	return this->size == 0;//если пустой, то вернет 1 иначе 0
}

int List::operator () () {//размер списка
	return this->size;
}

int& List::operator[](int index) {
	if (index < this->size && index >= 0) {//если индекс меньше размера и больше -
		Node* current_node = this->head;
		for (int i = 0; i != index; i++) {
			current_node = current_node->ptr_to_next_node;//беру следующий элемент
		}
		return current_node->data;
	}
	else {
		cout << "Индекс вне цикла";
		exit(0);
	}
}

List& List::operator = (const List& list) {
	if (this == &list) {//если элементы и так уже равны
		return *this;
	}
	while (head != nullptr) {//сначала очищаю список полностью
		Node* temp = head;
		head = head->ptr_to_next_node;
		delete temp;
	}
	size = 0;

	Node* current_node = list.head;//голова
	while (current_node != nullptr) {//пока не рпойду весь спиок
		push_bask(current_node->data);//добавляю очередной элемент в список
		current_node = current_node->ptr_to_next_node;//беру следующий элемент
	}
	this->begin = list.begin;
	this->end = list.end;
	return *this;
}

List List::operator *(List& list) {
	int temp_size;
	if (this->size > list.size) {//если первый размер списка больше размера 2 списка
		temp_size = list.size;//наименьший размер из двух
```

```
	}
	else {
		temp_size = this->size;//наименьший размер из двух
	}

	List temp(temp_size, 0);//список с наименьшей длиной
	for (int i = 0; i < temp_size; i++) {//пока не пройду по списку наименьшей длины
		temp[i] = (*this)[i] * list[i];
	}
	return temp;
}

ostream& operator <<(ostream& stream, const List& list) {//вывод
	cout << "Вывод элементов списка:" << endl;
	Node* current_node = list.head;
	while (current_node != nullptr) {//пока не пройду весь список
		stream << current_node->data << ' ';
		current_node = current_node->ptr_to_next_node;//беру следующий элемент
	}
	return stream;
}

istream& operator >>(istream& stream, const List& list) {//ввод
	cout << "Введите элементы списка" << endl;
	Node* current_node = list.head;
	while (current_node != nullptr) {//пока не пройду весь список
		stream >> current_node->data;
		current_node = current_node->ptr_to_next_node;//беру следующий элемент
	}
	return stream;
}
```

Файл iterator.h

```
#pragma once //предотвращает повторную загрузку заголовочного файла, если он уже был включен
#include "list.h"
#include <iostream> //стандартные потоки ввода и вывода
using namespace std;

class Iterator {
private:
	friend struct Node;
	friend class List;
	Node* elem;
public:
	Iterator(); //конструктор по умолчанию
	Iterator(const Iterator & iter);//конструктор копирования
	~Iterator() {}
	Iterator& operator =(const Iterator& iter);//текущий итератор
	bool operator == (const Iterator& iter);//элементы равны?
	bool operator != (const Iterator& iter);//элементы НЕ равны?
	Iterator& operator ++(int);//следующий элемент
	Iterator& operator --(int);//предыдущий элемент
	Iterator& operator +(const int number);//элемент по индексу
	Iterator& operator -(const int number);//элемент по индексу
	int& operator *();//доступ к данным элемента по итератору
};
```

Файл iterator.cpp

```
#include "list.h"
#include "iterator.h"
```

```
#include <iostream>
#include <string>
using namespace std;

Iterator::Iterator() {//конструктор по умолчанию
	this->elem = nullptr;
}

Iterator::Iterator(const Iterator& iter) {//конструктор копирования
	this->elem = iter.elem;
}

Iterator& Iterator:: operator =(const Iterator& iter) {//текущий итератор
	this->elem = iter.elem;
	return *this;
}

bool Iterator::operator == (const Iterator& iter) {//элементы равны?
	return this->elem == iter.elem;
}

bool Iterator::operator != (const Iterator& iter) {//элементы НЕ равны?
	return this->elem != iter.elem;
}

Iterator& Iterator::operator ++(int) {//следующий элемент
	this->elem = this->elem->ptr_to_next_node;
	return *this;
}

Iterator& Iterator::operator --(int) {
	this->elem = this->elem->ptr_to_prev_node;//предыдущий элемент
	return *this;
}

Iterator& Iterator::operator +(const int number) {//элемент по индексу
	Iterator temp(*this);
	for (int i = 0; i < number; i++) {
		temp.elem = temp.elem->ptr_to_next_node;
	}
	return temp;
}

Iterator& Iterator::operator -(const int number) {//элемент по индексу
	Iterator temp(*this);
	for (int i = 0; i < number; i++) {
		temp.elem = temp.elem->ptr_to_prev_node;
	}
	return temp;
}

int& Iterator::operator *() {//доступ к данным элемента по итератору
	return this->elem->data;
}
```

Результаты работы

```
list_1
Вывод элементов списка:
0 0 0 0 0 0 0 0 0 0

list_1
Вывод элементов списка:
50 22 50 48 16 34 -19 -14 12 33

Первый элемент: 50
Последний элемент: 33

list_2
Вывод элементов списка:
50 22 50 48 16 34 -19 -14 12 33

Вывод элементов списка:
100 50 22 50 48 16 34 -19 -14 12

Первый элемент: 100
Последний элемент: 12

list_3
Введите элементы списка
23
-9
0
3232
34563456
Вывод элементов списка:
23 -9 0 3232 34563456

Первый элемент: 23
Последний элемент: 34563456

list_4
Вывод элементов списка:
2300 -450 0 161600 1659045888

Первый элемент: 2300
Последний элемент: 1659045888

Вывод элементов list_1 с помощью Iterator с НАЧАЛА
50 22 50 48 16 34 -19 -14 12 33
Вывод элементов list_1 с помощью Iterator с КОНЦА
33 12 -14 -19 34 16 48 50 22 50

Вывод элемента 2 list_3 с помощью Iterator с НАЧАЛА
-9
Вывод элемента 2 list_3 с помощью Iterator с КОНЦА
-9

Сравнение элемента 2 и 3 list_3 с помощью Iterator
-9 != 0 элементы не равны
```

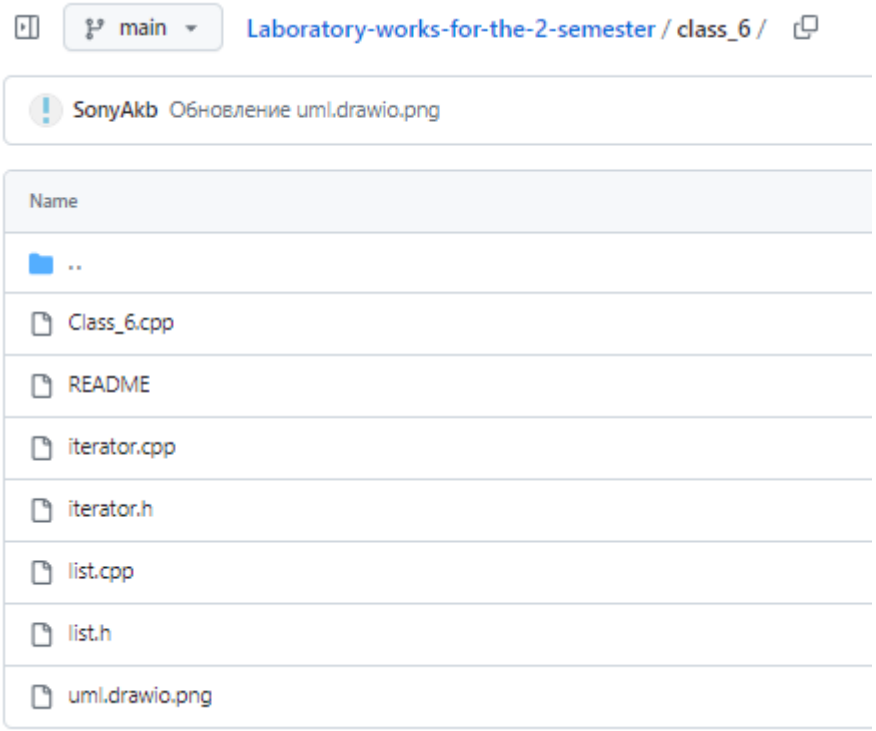
Вывод

В ходе работы я применила знания о работе с классами. В частности, об абстрактных типах данных, контейнерах. В ходе работы были реализованы

класс контейнер – список, конструкторы, деструктор, операции ввода-вывода, операцию присваивания. Также был создан класс-итератор, который был создан с помощью операции последовательного доступа. В основной функции были продемонстрированы операции со списокм.

GitHub

Ссылка: https://github.com/SonyAkb/Laboratory-works-for-the-2-semester/tree/main/class_6



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.

2. 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.

3. 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.

4. Что такое контейнер? Привести примеры.

Контейнер – набор однотипных элементов. Встроенные массивы в C++ - частный случай контейнера

5. 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.

6. 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.

7. Что такое итератор?

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

8. 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

9. 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.

10. Какой доступ к элементам предоставляет контейнер, состоящий из элементов «ключ-значение»?

Ассоциативный доступ.

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

Стек.

12. Какой из объектов (a, b, c, d) является контейнером?

a. int mas=10;

b. 2. int mas;

c. 3. struct {char name[30]; int age;} mas;

d. 4. int mas[100];

Ответ: d

13. Какой из объектов (a,b,c,d) не является контейнером?

a. int a[]={1,2,3,4,5};

b. 2. int mas[30];

c. 3. struct {char name[30]; int age;} mas[30];

d. 4. int mas;

Ответ: d

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

Прямой доступ.

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

Последовательный доступ.