Министерство науки и высшего образования Российской Федерации Федеральное государственное автономное учреждение высшего образования «Пермский национальный исследовательский политехнический университет»

ПНИПУ

**Лабораторная работа  
“Классы”. №9**

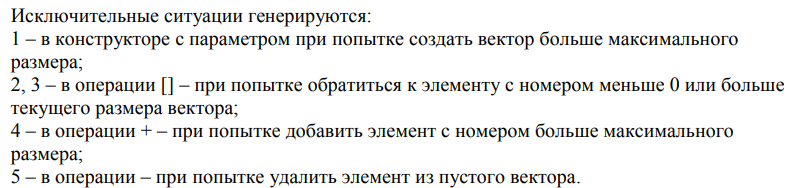
Выполнил:   
студент группы ИВТ-23-2б   
Чудинов Данил Николаевич

Проверила:   
доцент кафедры ИТАС   
О.А. Полякова

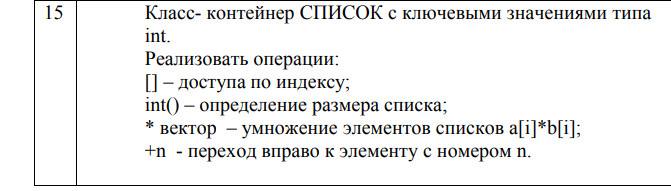
Пермь, 2024 г.

**«Классы и объекты. Принцип подстановки»**

**Анализ задачи:**

****

***Вариант 15:***

****

**Код на языке C++:**

Main cpp:

#include <iostream>

#include <string>

#include "list.h"

using namespace std;

int main() {

try {

system("chcp 1251>NULL");

List list(10, 5);

cout << list;

list.pushBack(3);

cout << list;

cout << "Введите номер элемента: ";

int i;

cin >> i;

cout << list[i] << endl;

}

catch (int) {

cout << "Ошибка!!!";

return 0;

}

}

List.h

#pragma once

#include <iostream>

using namespace std;

struct Node {

int data;

Node\* next = nullptr;

Node\* prev = nullptr;

};

class List {

private:

int size;

Node\* head=nullptr;

Node\* tail=nullptr;

public:

void pushFront(int data);

void pushBack(int data);

List(int size);

List(int size, int data);

List(const List& list);

~List();

int popBack();

int popFront();

int front();

int back();

bool is\_empty();

List& operator =(const List& list);

int& operator [] (int index);

int operator () ();

List operator \* (List& list);

int operator+(int h);

friend istream& operator >> (istream& is, const List& list);

friend ostream& operator << (ostream& os, List& list);

};

List.cpp

#include "list.h"

using namespace std;

void List::pushFront(int data) {

Node\* newNode = new Node;

newNode->data = data;

if (head == nullptr) {

this->head = newNode;

this->tail = newNode;

}

else {

head->prev = newNode;

newNode->next = head;

head = newNode;

}

this->size++;

}

void List::pushBack(int data) {

Node\* newNode = new Node;

newNode->data = data;

newNode->next = nullptr;

if (head == nullptr) {

this->head = newNode;

this->tail = newNode;

}

else {

tail->next = newNode;

newNode->prev = tail;

tail = newNode;

}

this->size++;

}

List::List(int size) {

this->size = size;

if (size < 0) {

throw 1;

}

if (size > 0) {

Node\* node = new Node;

this->head = node;

this->tail = node;

for (int i = 1; i < size; i++) {

Node\* newNode = new Node;

tail->next = newNode;

newNode->prev = tail;

tail = newNode;

}

}

else {

this->head = nullptr;

this->tail = nullptr;

}

}

List:: List(int size, int data) {

this->size = size;

if (size < 0) {

throw 1;

}

if (size > 0) {

Node\* node = new Node;

node->data = data;

this->head = node;

this->tail = node;

for (int i = 1; i < size; i++) {

Node\* newNode = new Node;

newNode->data = data;

tail->next = newNode;

newNode->prev = tail;

tail = newNode;

}

}

else {

this->head = nullptr;

this->tail = nullptr;

}

}

List::List(const List& list) {

this->head = nullptr;

this->tail = nullptr;

this->size = 0;

Node\* current\_node = list.head;

while (current\_node != nullptr) {

pushBack(current\_node->data);

current\_node = current\_node->next;

}

}

List::~List() {

Node\* node = head;

while (node != nullptr) {

Node\* newNode = node->next;

delete node;

node = newNode;

}

head = nullptr;

}

int List::popBack() {

int temp = 0;

if (this->tail != nullptr) {

Node\* current\_node = this->tail;

tail = current\_node->prev;

temp = current\_node->data;

tail->next = nullptr;

this->size = size - 1;

delete current\_node;

}

return temp;

}

int List::popFront() {

int temp = 0;

if (this->head != nullptr) {

Node\* current\_node = this->head;

head = current\_node->next;

temp = current\_node->data;

if (head) {

head->prev = nullptr;

}

delete current\_node;

this->size = size - 1;

}

return temp;

}

int List::front() {

return this->head->data;

}

int List::back() {

return this->tail->data;

}

bool List::is\_empty() {

return this->size == 0;

}

List& List:: operator =(const List& list) {

cout << "Вызвана операция присваивания: " << endl;

if (this == &list) {

return \*this;

}

while (head != nullptr) {

Node\* temp = head;

head = head->next;

delete temp;

}

this->size = 0;

Node\* current = list.head;

while (current != nullptr) {

pushBack(current->data);

current = current->next;

}

return \*this;

}

int& List:: operator [] (int index) {

cout << "Вызвана операция поиска по индексу: " << endl;

if (index < 0) {

throw 2;

}

if (index > size) {

throw 3;

}

if (index < this->size && index >= 0) {

Node\* current = head;

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

current = current->next;

}

return current->data;

}

else {

cout << "Индекс не найден " << endl;

}

}

int List:: operator () () {

return this->size;

}

List List:: operator \* (List& list) {

int temp;

if (this->size > list.size) {

temp = list.size;

}

else {

temp = this->size;

}

List templ(temp, 0);

for (int i = 0; i < temp; i++) {

templ[i] = (\*this)[i] \* list[i];

}

return templ;

}

int List:: operator+(int h){

if (h > +size) {

throw 4;

}

Node\* current = head;

for (int i = 0; i < h; i++) {

current = current->next;

}

return (current->data);

}

istream& operator >> (istream& is, const List& list) {

cout << "Вызван оператор перегрузки »" << endl << endl;

Node\* current\_node = list.head;

while (current\_node != nullptr) {

is >> current\_node->data;

current\_node = current\_node->next;

}

cout << "Ввод закончен" << endl << endl;

return is;

}

ostream& operator << (ostream& os, List& list) {

cout << "Вызван оператор перегрузки « " << endl << endl;

Node\* current\_node = list.head;

while (current\_node != nullptr) {

os << current\_node->data << " ";

current\_node = current\_node->next;

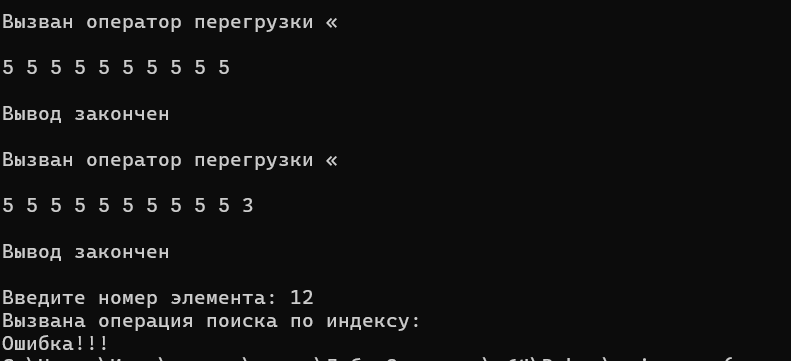
}

cout << endl << endl << "Вывод закончен" << endl << endl;

return os;

}

**Работа программы:**



**Ответ на вопросы:**

1. Механизм наследования используется в объектно-ориентированном программировании для создания иерархий классов, когда один класс (производный) может наследовать свойства и методы другого класса (базового).

2. Компоненты класса, описанные со спецификатором `public`, наследуются производным классом также со спецификатором `public`. Это значит, что эти компоненты будут доступны и снаружи производного класса.

3. Компоненты класса, описанные как `private`, не наследуются производным классом и остаются закрытыми для доступа извне.

4. Компоненты класса, описанные как `protected`, наследуются производным классом с таким же спецификатором `protected`.

5. Производный класс описывается с использованием ключевого слова `class`, за которым идет имя производного класса, двоеточие и ключевое слово `public` или `private`, затем имя базового класса.

6. Конструкторы наследуются, но они должны быть вызваны явно в конструкторе производного класса.

7. Деструкторы также наследуются, но если они виртуальные, то вызывается сначала деструктор производного класса, а затем базового.

8. Объекты производных классов конструируются в следующем порядке: сначала конструируется базовый класс, затем производный класс.

9. Объекты производных классов уничтожаются в обратном порядке: сначала вызывается деструктор производного, затем базового класса.

10. Виртуальные функции обеспечивают механизм позднего связывания, что позволяет использовать методы производного класса через указатель или ссылку на базовый класс.

11. Виртуальные конструкторы не могут быть, но деструкторы могут быть виртуальными. Наследуемый лицензионный спецификатор `virtual`.

12. Открытое наследование устанавливает "есть-как" отношение, где производный класс является расширением базового класса.

13. Закрытое наследование устанавливает "есть-часть" отношение, где производный класс является частным случаем базового класса.

14. Принцип подстановки заключается в том, что объект базового класса может заменить объект производного класса, если поведение базового класса соответствует поведению производного.

15. Объект `x` класса `Teacher` будет иметь следующие компоненты данных: `age` (унаследовано от `Student`), `name` (унаследовано от `Student`), `post` (унаследовано от `Employee`) и `stage`.

16. Конструкторы без параметров:

class Student {

public:

Student() : age(0), name("") {}

};

class Employee : public Student {

protected:

Employee() : post("") {}

};

class Teacher : public Employee {

protected:

Teacher() : stage(0) {}

};

17. Конструкторы с параметрами:

class Student {

public:

Student(int a, string n) : age(a), name(n) {}

};

class Employee : public Student {

protected:

Employee(int a, string n, string p) : post(p) {}

};

class Teacher : public Employee {

protected:

Teacher(int a, string n, string p, int s) : stage(s) {}

};

18. Конструкторы копирования:

class Student {

public:

Student(const Student& other) : age(other.age), name(other.name) {}

};

class Employee : public Student {

protected:

Employee(const Employee& other) : post(other.post) {}

};

class Teacher : public Employee {

protected:

Teacher(const Teacher& other) : stage(other.stage) {}

};

19. Операция присваивания:

class Student {

public:

Student& operator=(const Student& other) {

if (this != &other) {

age = other.age;

name = other.name;

}

return \*this;

}

};

class Employee : public Student {

protected:

Employee& operator=(const Employee& other) {

if (this != &other) {

static\_cast<Student&>(\*this) = other;

post = other.post;

}

return \*this;

}

};

class Teacher : public Employee {

protected:

Teacher& operator=(const Teacher& other) {

if (this != &other) {

static\_cast<Employee&>(\*this) = other;

stage = other.stage;

}

return \*this;

}

};

**Вывод:** Задача была выполнена. Всё получилось.

