Министерство образования Республики Беларусь

Учреждение образования

«Брестский государственный технический университет»

Кафедра ИИТ

Лабораторная работа №3

По дисциплине: «Языки программирования»

Вариант №3

Выполнили:

Студенты 2-го курса

Группы ПО-7

Гункевич И.А.

Проверила:

Дряпко А.В.

Брест 2021

**Цель работы:** Изучение правил перегрузки операций и принципов обработки исключений в C++.

**Порядок выполнения работы**

Написать программу, в которой описана иерархия классов: человек («дошкольник», «школьник», «студент», «работающий»). Описать класс для хранения коллекции людей(массива указателей на базовый класс), в котором перегрузить операцию «[ ]». Для базового класса и его потомков перегрузить операции «==», «!=», «=». Продемонстрировать работу операторов.

**Код программы:**

**main.cpp**

#include <iostream>  
#include "human.h"  
#include "schoolkid.h"  
#include "preschooler.h"  
#include "student.h"  
#include "worker.h"  
#include "string"  
#include "HumanArray.h"  
void menu(HumanArray arr);  
  
int main() {  
 setlocale(LC\_ALL, "ru");  
  
 cout << "Вариант 3." << endl  
 << "Написать программу, в которой описана иерархия классов: человек («дошкольник»,«школьник», «студент», «работающий»)."  
 << endl << "Описать класс для хранения коллекции людей (массива указателей на базовый класс), в котором пергрузить операцию [ ],"  
 << endl <<" Для базового класса и его потомков перегрузить"  
 << endl << "операции ==, !=, =. Продеманстрировать работу операторов." << endl;  
  
 HumanArray arr(0);  
 menu(arr);  
  
 return 0;  
}  
  
void menu(HumanArray arr) {  
 bool exit = false;  
  
 while (!exit) {  
 cout << "1 - add preschooler" << endl  
 << "2 - add schoolkid" << endl  
 << "3 - add student" << endl  
 << "4 - add worker" << endl  
 << "5 - Show information about a person by index" << endl  
 << "6 - Show all" << endl  
 << "7 - remove from the end" << endl  
 << "8 - delete by index" << endl  
 << "9 - compare the ages of preschoolers" << endl  
 << "0 - Exit" << endl;  
 try  
 {  
 int num;  
 cin >> num;  
 cout << endl;  
 switch (num)  
 {  
 case 1:  
 {  
 Preschooler \*obj1 = new Preschooler("", 0);  
 obj1->Read();  
 Preschooler \*obj2 = new Preschooler("", 0);  
 obj2 = obj1;  
  
 bool temp = false;  
 cout << "Add to the end (0) or by index (1)?";  
 cin >> temp;  
 if (temp) {  
 int index;  
 cout << "Enter the index: ";  
 cin >> index;  
 arr.add(index, obj2);  
 }  
 else {  
 arr.addToTheEnd(obj2);  
 }  
 break;  
 }  
 case 2:  
 {  
 Schoolkid \*obj = new Schoolkid("", 0);  
 obj->Read();  
 bool temp = false;  
 cout << "Add to the end (0) or by index (1)?";  
 cin >> temp;  
 if (temp) {  
 int index;  
 cout << "Enter the index: ";  
 cin >> index;  
 arr.add(index, obj);  
 }  
 else {  
 arr.addToTheEnd(obj);  
 }  
 break;  
 }  
 case 3:  
 {  
 Student \*obj = new Student("");  
 obj->Read();  
 bool temp = false;  
 cout << "Add to the end (0) or by index (1)?";  
 cin >> temp;  
 if (temp) {  
 int index;  
 cout << "Enter the index: ";  
 cin >> index;  
 arr.add(index, obj);  
 }  
 else {  
 arr.addToTheEnd(obj);  
 }  
 break;  
 }  
 case 4:  
 {  
 Worker \*obj = new Worker("");  
 obj->Read();  
 bool temp = false;  
 cout << "Add to the end (0) or by index (1)?";  
 cin >> temp;  
 if (temp) {  
 int index;  
 cout << "Enter the index: ";  
 cin >> index;  
 arr.add(index, obj);  
 }  
 else {  
 arr.addToTheEnd(obj);  
 }  
 break;  
 }  
 case 5:  
 {  
 int index;  
 cout << "Enter the index: ";  
 cin >> index;  
 if (arr[index] != 0) {  
 arr[index]->Print();  
 }  
 break;  
 }  
 case 6:  
 {  
 cout << "Number of people = " << arr.get\_count() << endl;  
 for (int i = 0; i < arr.get\_count(); i++) {  
 arr[i]->Print();  
 }  
 break;  
 }  
 case 7:  
 {  
 arr.deleteFromTheEnd();  
 break;  
 }  
 case 8:  
 {  
 int index;  
 cout << "Enter the index: ";  
 cin >> index;  
 arr.del(index);  
 break;  
 }  
 case 0:  
 exit = true;  
 case 9: {  
 Preschooler obj1("", 0);  
 obj1.Read();  
 Preschooler obj2("", 0);  
 obj2.Read();  
 if (obj1 == obj2) {  
 cout << "Age equal" << endl;  
 } else if (obj1 != obj2) {  
 cout << "different age" << endl;  
 }  
 break;  
 }  
 }  
 cout << endl;  
 }  
 catch (IndexError &e)  
 {  
 cout << "Index error: " << e.get\_message() << endl;  
 }  
 catch (...)  
 {  
 cout << "Unknown error" << endl;  
 }  
 }  
}

**error.h**

#ifndef UNTITLED\_ERROR\_H  
#define UNTITLED\_ERROR\_H  
#include <iostream>  
#include "cstring"  
class IndexError {  
protected:  
 char \*message;  
public:  
 IndexError(const char \*message);  
 ~IndexError();  
 char \*get\_message();  
};  
#endif //UNTITLED\_ERROR\_H

**error.cpp**

#include "error.h"  
IndexError::IndexError(const char \*message) {  
 this->message = \_strdup(message);  
}  
IndexError::~IndexError() {  
 delete this->message;  
}  
char \*IndexError::get\_message() {  
 return message;  
}

**human.h**

#ifndef UNTITLED\_HUMAN\_H  
#define UNTITLED\_HUMAN\_H  
#include <iostream>  
using namespace std;  
  
class Human {  
public:  
 virtual void Print() = 0;  
 virtual void Read() = 0;  
};  
  
#endif //UNTITLED\_HUMAN\_H

**HumanArray.h**

#ifndef UNTITLED\_HUMANARRAY\_H  
#define UNTITLED\_HUMANARRAY\_H  
#include "human.h"  
#include "error.h"  
class HumanArray {  
private:  
 Human \*\*arr;  
 int count;  
public:  
 HumanArray(int count);  
 ~HumanArray();  
 Human\* operator[] (int n) const;  
 Human\*& operator[] (int n);  
 int get\_count();  
 void addToTheEnd(Human \*person);  
 void add(int index, Human \*person);  
 void deleteFromTheEnd();  
 void del(int index);  
};  
  
#endif //UNTITLED\_HUMANARRAY\_H

**HumanArray.cpp**

#include "HumanArray.h"  
  
HumanArray::HumanArray(int count) {  
 arr = new Human\*[count];  
 for (int i = 0; i < count; i++) {  
 arr[i] = nullptr;  
 }  
 this->count = count;  
}  
HumanArray::~HumanArray() {  
 for (int i = 0; i < count; i++) {  
 if (arr[i] != nullptr) {  
 delete arr[i];  
 }  
 delete[]arr;  
 }  
}  
  
Human \*HumanArray::operator[] (int n) const {  
 if (n < 0 || n >= count) {  
 throw IndexError("You are out of bounds of an array.");  
 }  
 return arr[n];  
}  
Human \*&HumanArray::operator[] (int n) {  
 if (n < 0 || n >= count) {  
 throw IndexError("You are out of bounds of an array.");  
 }  
 return arr[n];  
}  
  
int HumanArray::get\_count() {  
 return count;  
}  
  
void HumanArray::addToTheEnd(Human \*person) {  
 Human \*\*temp = new Human\*[count + 1];  
 for (int i = 0; i < count; i++) {  
 temp[i] = arr[i];  
 }  
 arr = temp;  
 temp[count] = person;  
 count++;  
}  
  
void HumanArray::add(int index, Human \*person) {  
 if (index < 0 || index > count) {  
 addToTheEnd(person);  
 throw IndexError("You are out of bounds of an array. The person will be added to the end of the array");  
 }  
 Human \*\*temp = new Human\*[count + 1];  
 for (int i = 0; i < index; i++) {  
 temp[i] = arr[i];  
 }  
 temp[index] = person;  
 for (int i = index; i < count; i++) {  
 temp[i + 1] = arr[i];  
 }  
 arr = temp;  
 count++;  
}  
  
void HumanArray::deleteFromTheEnd() {  
 Human \*\*temp = new Human\*[count - 1];  
 for (int i = 0; i < count - 1; i++) {  
 temp[i] = arr[i];  
 }  
 arr = temp;  
 count--;  
}  
  
void HumanArray::del(int index) {  
 Human \*\*temp = new Human\*[count - 1];  
 for (int i = 0; i < index; i++) {  
 temp[i] = arr[i];  
 }  
 for (int i = index + 1; i < count; i++) {  
 temp[i - 1] = arr[i];  
 }  
 arr = temp;  
 count--;

}

**preschooler.h**

#ifndef UNTITLED\_PRESCHOOLER\_H  
#define UNTITLED\_PRESCHOOLER\_H  
#include "human.h"  
#include "string"  
class Preschooler :public Human {  
private:  
 string name;  
 int age;  
public:  
 Preschooler();  
 Preschooler(string name, int age);  
 Preschooler(const Preschooler &other);  
 ~Preschooler();  
  
 bool operator == (const Preschooler &right);  
 bool operator != (const Preschooler &right);  
 Preschooler &operator = (const Preschooler &right);  
  
 void Print();  
 void Read();  
};  
  
#endif //UNTITLED\_PRESCHOOLER\_H

**preschooler.cpp**

#include "preschooler.h"  
  
Preschooler::Preschooler() {}  
Preschooler::Preschooler(string name, int age) {  
 this->name = name;  
 this->age = age;  
}  
Preschooler::Preschooler(const Preschooler &other) {  
 name = other.name;  
 age = other.age;  
}  
Preschooler::~Preschooler() {}  
  
bool Preschooler::operator == (const Preschooler &right) {  
 return age == right.age;  
}  
bool Preschooler::operator != (const Preschooler &right) {  
 return !(\*this == right);  
}  
Preschooler &Preschooler::operator = (const Preschooler &right) {  
 age = right.age;  
 return \*this;  
}  
  
void Preschooler::Print() {  
 cout << endl << "Name of preschooler: " << name << endl;  
 cout << endl << "Age of preschooler: " << age << endl;  
}  
void Preschooler::Read() {  
 cout << endl << "Enter the name of preschooler: ";  
 cin >> name;  
 cout << endl << "Enter the age of preschooler: ";  
 cin >> age;  
}

**schoolkid.h**

#ifndef UNTITLED\_SCHOOLKID\_H  
#define UNTITLED\_SCHOOLKID\_H  
#include "human.h"  
#include "string"  
class Schoolkid :public Human {  
private:  
 string name;  
 int age;  
public:  
 Schoolkid();  
 Schoolkid(string name, int age);  
 Schoolkid(const Schoolkid &other);  
 ~Schoolkid();  
  
 bool operator == (const Schoolkid &right);  
 bool operator != (const Schoolkid &right);  
 Schoolkid &operator = (const Schoolkid &right);  
  
 void Print();  
 void Read();  
};  
#endif //UNTITLED\_SCHOOLKID\_H

**schoolkid.cpp**

#include "schoolkid.h"  
  
Schoolkid::Schoolkid() {}  
Schoolkid::Schoolkid(string name, int age) {  
 this->name = name;  
 this->age = age;  
}  
Schoolkid::Schoolkid(const Schoolkid &other) {  
 name = other.name;  
 age = other.age;  
}  
Schoolkid::~Schoolkid() {}  
  
bool Schoolkid::operator == (const Schoolkid &right) {  
 return age == right.age;  
}  
bool Schoolkid::operator != (const Schoolkid &right) {  
 return !(\*this == right);  
}  
Schoolkid &Schoolkid::operator = (const Schoolkid &right) {  
 age = right.age;  
 return \*this;  
}  
  
void Schoolkid::Print() {  
 cout << endl << "Name of schoolkid: " << name << endl;  
 cout << endl << "Age of schoolkid: " << age << endl;  
}  
void Schoolkid::Read() {  
 cout << endl << "Enter the name of schoolkid: ";  
 cin >> name;  
 cout << endl << "Enter the age of schoolkid: ";  
 cin >> age;  
}

**student.h**

#ifndef UNTITLED\_STUDENT\_H  
#define UNTITLED\_STUDENT\_H  
#include "human.h"  
#include "string"  
class Student :public Human {  
private:  
 string name;  
public:  
 Student();  
 Student(string name);  
 Student(const Student &other);  
 ~Student();  
  
 bool operator == (const Student &right);  
 bool operator != (const Student &right);  
 Student &operator = (const Student &right);  
  
 void Print();  
 void Read();  
};  
#endif //UNTITLED\_STUDENT\_H

**student.cpp**

#include "student.h"  
  
Student::Student() {}  
Student::Student(string name) {  
 this->name = name;  
}  
Student::Student(const Student &other) {  
 name = other.name;  
}  
Student::~Student() {}  
  
bool Student::operator == (const Student &right) {  
 return name == right.name;  
}  
bool Student::operator != (const Student &right) {  
 return !(\*this == right);  
}  
Student &Student::operator = (const Student &right) {  
 name = right.name;  
 return \*this;  
}  
  
void Student::Print() {  
 cout << endl << "Name of student: " << name << endl;  
}  
void Student::Read() {  
 cout << endl << "Enter the name of student: ";  
 cin >> name;  
}

**worker.h**

#ifndef UNTITLED\_WORKER\_H  
#define UNTITLED\_WORKER\_H  
#include "human.h"  
#include "string"  
class Worker :public Human {  
private:  
 string name;  
public:  
 Worker();  
 Worker(string name);  
 Worker(const Worker &other);  
 ~Worker();  
  
 bool operator == (const Worker &right);  
 bool operator != (const Worker &right);  
 Worker &operator = (const Worker &right);  
  
 void Print();  
 void Read();  
};  
#endif //UNTITLED\_WORKER\_H

**worker.cpp**

#include "worker.h"  
Worker::Worker() {}  
Worker::Worker(string name) {  
 this->name = name;  
}  
Worker::Worker(const Worker &other) {  
 name = other.name;  
}  
Worker::~Worker() {}  
  
bool Worker::operator == (const Worker &right) {  
 return name == right.name;  
}  
bool Worker::operator != (const Worker &right) {  
 return !(\*this == right);  
}  
Worker &Worker::operator = (const Worker &right) {  
 name = right.name;  
 return \*this;  
}  
  
void Worker::Print() {  
 cout << endl << "Name of worker: " << name << endl;  
}  
void Worker::Read() {  
 cout << endl << "Enter the name of worker: ";  
 cin >> name;  
}

**Вывод:** я изучил правила перегрузки операций и принципов обработки исключений в C++.