Министерство образования и науки Российской Федерации

Федеральное государственное бюджетное образовательное учреждение высшего образования «Московский государственный технический университет имени Н. Э. Баумана (национальный исследовательский университет)

Факультет «Робототехника и комплексная автоматизация»

Кафедра «Системы автоматизированного проектирования»

**Отчет по лабораторной работе**

По курсу «Объектно-ориентированное программирование»

Выполнил: Студент Петраков С.А.

Группа РК6-26Б

Проверил: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Дата \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Подпись\_\_\_\_\_\_\_\_\_\_\_\_\_

Москва, 2020 г.

**ПЯВУ Вариант 16V**

**Задание:**

Разработать объектно-ориентированную программу для числовой игры, где 2 игрока должны поочередно выбирать из кучи заданного размера S любое количество предметов, которое не превышает фиксированного значения N<S. Игра должна продолжаться, пока текущий размер кучи больше 0. Победителем считается игрок, который взял последний предмет кучи. Игра должна быть реализована в варианте, когда двумя противниками являются компьютер и человек, которому предоставляется право первого хода. Человек может делать любые допустимые ходы, сообщая число взятых им предметов через стандартный ввод. При любой ошибке ввода партия игры должна аварийно завершаться с соответствующей диагностикой. Компьютер должен действовать по выигрышной стратегии, согласно которой остаток кучи после каждого его хода должен быть кратен (N+1). Если такой выигрышный ход сделать нельзя, компьютер должен взять из кучи N предметов. В любом случае число предметов, которое взял из кучи компьютер должно отображаться через поток стандартного вывода. Исходными данными для любой партии игры являются размер кучи S и предельное число предметов N, которое можно взять из кучи за 1 ход. Эти параметры должны передаваться программе через аргументы командной строки ее вызова. Каждый ход должен сопровождать информационный запрос, который отображает текущий размер кучи и идентифицирует игрока, чей ход ожидается в данный момент. Игра должна завершаться диагностическим сообщением, которое идентифицирует ее итоговый результат. При разработке программы необходимо реализовать производные классы игроков с виртуальной функцией хода, которые наследуют интерфейс игры и доступ к куче от абстрактного базового класса.

**Алгоритм:**

При запуске программы проверяем корректность ввода 2 чисел. При каждом ходе игрока, выводим число – количество вещей, которые игрок берёт из кучи, число – вещей в куче и имя игрока, который сейчас должен сходить. При неправильном вводе числа, программа аварийно завершается. Игра продолжается пока в куче есть хоть 1 вещь. Выигрывает тот, кто последний возьмёт вещи из кучи; методы запроса (выводит стандартное сообщение перед ходом каждого игрока) и виртуальный метод хода, которые определена в наследниках (проверяет лучший ход для игрока-компьютера и проверяет на корректность у игрока-человека).

В нашем базовом классе хранятся приватные поля: указатель на кучу, имя игрока и максимальное число вещей, которое можно взять из кучи.

**Входные данные:**

2 целых числа, передающиеся через аргументы командной строки: размер кучи и предельное число предметов, которое можно взять из кучи за 1 ход.

**Выходные данные:**

При каждом ходе выводим число предметов, которые взял из кучи игрок (компьютер или человек), текущий размер кучи и имя игрока, который ходит.

В конце игры: итоговый результат.

**Текст программы:**

**main.cpp**

#include <stdlib.h>

#include <iostream>

#include "Man.h"

#include "Pen.h"

#include "Heap.h"

int main(int argc, char\* argv[])

{

//Checking input

if (argc != 3 || atoi(argv[1]) <= 0 || atoi(argv[2]) <= 0 || atoi(argv[1]) <= atoi(argv[2]))

{

std::cout << "Invalid start position. Terminate.\n";

return 0;

}

//init game

Heap heap(atoi(argv[1]));

Gambler\* players[] = {

new Man(heap,atoi(argv[2]),"Man"),

new Pen(heap,atoi(argv[2]),"Pen")

};

//Start output

std::cout << "Start position: " << heap.get() << std::endl;

std::cout << "Limit for move:" << atoi(argv[2]) << std::endl;

//Start game

int i = 0;

while (heap.get() != 0)

{

players[i]->query();

if (!players[i]->move())

{

std::cout << "Invalid input. Terminate.\n";

return 0;

}

if (++i > 1)

i = 0;

}

std::cout << "Winner!";

//Clearing memory

delete players[0];

delete players[1];

return 0;

}

**Heap.h**

#pragma once

#ifndef HEAPH

#define HEAPH

class Heap

{

private:

int size;

public:

Heap(int);

int get();

void set(int);

};

#endif

**Heap.cpp**

#include "Heap.h"

Heap::Heap(int s) : size(s)

{

}

int Heap::get()

{

return size;

}

void Heap::set(int s)

{

size = s;

return;

}

**Gambler.h**

#pragma once

#ifndef GAMBLERH

#define GAMBLERH

#include "Heap.h"

#include <iostream>

class Gambler

{

protected:

Heap\* heap;

const int limit; //Max input

const char\* name;

public:

Gambler(Heap&, int);

void query();

virtual bool move() = 0;//True if all good, false if game is broken

virtual ~Gambler();

};

#endif

**Gambler.cpp**

#include "Gambler.h"

Gambler::Gambler(Heap& h, int l) :limit(l)

{

heap = &h;

}

void Gambler::query()

{

std::cout << "Heap=" << heap->get() << ". " << name << " > ";

return;

}

Gambler::~Gambler()

{

}

**Man.h**

#pragma once

#ifndef MANH

#define MANH

#include "Gambler.h"

#include <iostream>

class Man : public Gambler

{

public:

Man(Heap&, int, const char\*);

virtual bool move();

};

#endif

**Man.cpp**

#include "Man.h"

Man::Man(Heap& h, int l, const char\* n) :Gambler(h, l)

{

name = n;

}

bool Man::move()

{

int m;

std::cin >> m;

if (m<1 || m>limit || m > heap->get() || std::cin.fail())

return false;

heap->set((heap->get() - m));

return true;

}

**Pen.h**

#pragma once

#ifndef PENH

#define PENH

#include "Gambler.h"

class Pen : public Gambler

{

public:

Pen(Heap&, int, const char\*);

virtual bool move();

};

#endif

**Pen.cpp**

#include "Pen.h"

Pen::Pen(Heap& h, int l, const char\* n) :Gambler(h, l)

{

name = n;

}

bool Pen::move()

{

int m;

for (m = 1; m <= limit; m++)

if ((heap->get() - m) % (limit + 1) == 0)

{

heap->set(heap->get() - m);

std::cout << m << std::endl;

return true;

}

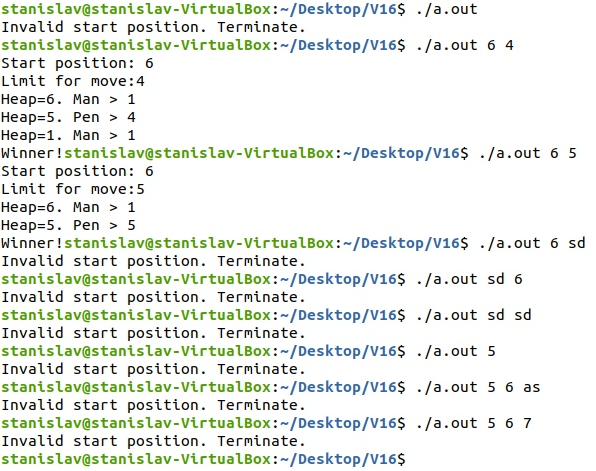
heap->set(heap->get() - limit);

std::cout << limit << std::endl;

return true;

}

**Тесты:**

****

**Список использованной литературы:**

* Волосатова Т.М., Родионов С.В. Лекции по курсу «Объектно-ориентированное программирование»
* bigor.bmstu.ru