 *Софийски университет „Св. Климент Охридски“  
 Факултет по математика и информатика*

*Курсова работа по Обектно-ориентирано програмиране*

*специалност Компютърни науки*

Тема №8 Джурасик парк

Съдържание

1. Увод……………………………………………………………………………………..…....3
   1. Описание и идея на проекта
   2. Цели и задачи на разработката
   3. Връзка към хранилището в Github
2. Общ поглед върху проекта……………………………………………………………….....4
   1. Основни дефиниции, концепции и алгоритми, използвани в проекта
   2. Подходи и методи за решаване на поставените проблеми
3. Проектиране
   1. Архитектура……………………………………………………………………………..5
   2. Диаграми (най-важните извадки от кода)……………………………………………..8
4. Реализация и тестване……………………………………………………………………...10  
   4.1. Управление на паметта и алгоритми

4.2. Тестове

1. Заключение………………………………………………………………………………….11
   1. Бъдещо развитие и усъвършенстване
2. **Увод**
   1. **Описание и идея на проекта**

Проектът „Джурасик парк“ реализира виртуален зоопарк за динозаври, в който потребителят може да създаде различни по брой и вид клетки за виртуалните праисторически влечуги. Идеята е да се напише система за управление на така наречения зоопарк, която да строи клетки, добавя животни, наема хора и зарежда храна за динозаврите.

* 1. **Цели и задачи на разработката**

Целта на проекта е да се създаде оптимална програма с максимално разнообразна функционалност, която да отговаря на поставената задача.   
Проектът е изграден съгласно добрите принципи на ООП – проектиран е така, че да работи с „абстрактни данни“ – данни с неясно представяне, което дава възможност за по-лесно описание и модификация на програмата.

* 1. **Връзка към хранилището в Github**: <https://github.com/Spacepanda21/Jurassic_Park>

1. **Общ поглед върху проекта**
   1. **Основни дефиниции, концепции и алгоритми, използвани в проекта**

Данните в проекта са разпределени в съответни класове. За по-голяма достъпност, улеснение и подредба проектът е съставен от *header* и *cpp* *файлове*, като във всеки *header* се съдържа отделен клас, обединяващ необохидимите за неговата имплементация член-данни и член-функци, които са разпределени в *public* и *private* секциите в зависимост от необходимия достъп до тях, а в *cpp файла* се намират дефинициите на декларираните в *header* член-функции. Също така са използвани външни функции, които не са методи на класовете. Използвана е конвенцията за именуване на функции и променливи *snake\_case*, като функциите и променливите започват с малка буква, а класовете с главна.

* 1. **Подходи и методи за решаване на поставените проблеми**

Класовете са изградени по така нареченото *Rule of Three*,т.е. съдържат член-функциите *Destructor*, *Copy Constructor* и *Copy Assignment Operator* и *Constructor*, за да може програмата да работи правилно. В конструктора са инициализирани член-данните чрез обобщена синтактична конструкция (в така наречените инициализиращи списъци), т.е. в заглавието преди изпълнението на тялото на конструктора. Понеже имаме обекти, които не могат да се създадат чрез директно присвояване на член-данните, сме дефинирали копи конструктор. Тъй като при изготвянето на проекта не са създавани класове с общи компоненти и поведение като тези на вече дефиниран клас, то не се е налагало използването на идеята на наследяването.

В програмата е използван *enum class* за член-данните на класа динозавър (Dinosaur) ера (era\_t ::= TRIASSIC|JURASSIC|CRETACEOUS\_PERIODS), категория (category\_t ::= HERBIVORE|CARNIVORE|PTEROSAUR|SPINOSAUR) и пол (sex\_t ::= FEMALE|MALE) и за член-данните на класа храна (Food) тип храна (food\_t ::= PLANTS|MEAT|FISH) понеже според заданието те могат да бъдат само тези няколко определени вида и нищо друго.

Използвани са следните библиотеки:

* <fstream> // библиотека за работа с файлове
* <iostream> //стандартантната библиотека за вход и изход
* <sstream> //stringstream за функцията split\_string();
* <string>
* <time.h> //std::rand(); за генериране на произволно число
* <vector>

1. **Проектиране**
   1. **Архитектура**

За имплементацията на проекта са създадени следните класове:

* Dinosaur
* Cage
* Dimensions
* Zoо

и структури:

* Food
* Parser
* Клас **Dinosaur** се характеризира с член-данните в *private* секцията :
  + string name (името на динозавъра)
  + sex\_t sex (полът на динозавъра)
  + category\_t category (разред на динозавъра)
  + era\_t era (ерата, към която принадлежи динозавърът)
  + string climate (климатът, който обитава)
  + string type (видът на динозавъра)
  + food\_t food (храна, с която се храни)

В *public* секцията се намират необходимите конструктор, деструктор, копи конструктор, оператор=, съответните селектори, които да позволят достъп до член-данните в *private* секцията на класа, и метода print();, който с подходящи *log* извежда информацията за динозаврите в конзолата.

* Клас **Cage** се характеризира със следните член-данни в секцията със спецификатор за достъп *private*:
* size\_t capacity // максималният капацитет на клетката
* Dimensions dimensions //размерите на клетката са обект от клас

Dimensions

* vector<Dinosaur> dinosaurs //вектор от динозаврите в клетката

}

* category\_t category /\*тези три член-данни служат за
* era\_t era свързване на динозавъра с
* string climate подходяща клетка\*/

Методите на този клас са съответно „Голямата 4-ка“, съответните селектори за достъп до category, era и climate, print(); и list\_dinosaurs();, извеждащи съответната информация за клетката и динозаврите в нея. Също така са дефинирани функциите void add\_new\_dinosaur(Dinosaur); и void remove\_dinosaur(string name);, които ще помогнат при реализирането на void add\_dinosaur(Dinosaur); и void remove\_dinosaur\_by\_name(); в класа Zoo. Освен това има и помощни функции като bool has\_free\_space(); и int dinosaur\_count();, които ще бъдат извикани в други функции.

* Клас **Dimensions** е имплементиран, за да можем да създадем обект от този тип в клас Cage, тъй като разглеждаме клетката в тримерното пространство. Съответно член-данните му x, y, z са от тип double и служат за означаване на размерите на клетката в пространството. В *public* секцията има съответните конструктор, деструктор, копи конструктор и оператор= както и void print(); метода.
* Клас **Zoo** се състои от член-данните в *private* секцията:
* vector<Cage> cages //вектор от клетки в зоопарка
* Food plants
* Food meat
* Food fish
* int staff\_count //имаме нужда от екип от хора, който да се

грижи за динозаврите в зоопарка

и методите:

* bool is\_food\_category\_compatible(food\_t, category\_t); използва се за проверка при подаване на храна на динозавъра, понеже растителноядните динозаври не могат да ядат месо; ако е *false* се извежда подходящо съобщение при създаването на динозавър
* Dinosaur load\_dinosaur(ifstream&); помощна функция, която ще се използва при четене на информация от файл

Функции в *public* секцията са:

* Конструктор Zoo();, в който се извиква функцията void load\_from\_file(string filename);, която със създаване на обект от класа Zoo започва четене на информацията от файла със съответното име, подадено като аргумент на функцията, както и по-долу описаната функция void create\_random\_cages();
* Деструктор ~Zoo(); , в който се извиква функцията void save\_in\_file(string filename); , която с приключването на програмата запаметява информацията във файла, подаден като аргумент на метода
* void load\_command(); и void save\_command(); са аналогични на функциите, извиквани в конструктора и деструктора, но позволяват на потребителя да запази информацията във въведен от него файл, защото все пак не искаме въведената от потребителя информация да се губи, каквото е и едно от условията на задачата
* void create\_cage(); и void create\_random\_cages();, чрез които се създават клетки в зоопарка, като разликата е, че методът void create\_random\_cages(); се използва за генериране на прозволен брой клетки при стартирането на програмата, каквото е и изискването на задачата, а void create\_cage(); позволява на потребителя да създаде клетка с въведени от него в конзолата параметри
* void list(ostream&) const; изкарва в конзолата информация за всички налични клетки в зоопарка – техните параметри и динозаврите вътре
* void hire\_staff(); служи за добавяне на нов служител, което е необходимо, когато броят на динозаврите спрямо хората се увеличи с повече от 3, т.е. 1 човек се грижи за не повече от 3 динозавъра; нуждата от тази функция се подсказва на потребителя при добавяне на динозавър в зоопарка
* void deliver\_food\_in\_storage(); се използва, за да се зареди определено количество храна в хранилището, като потребителят избира от кой тип и какво количество храна да въведе; необходимостта от тази функция се подсказва на потребителя с подходящо съобщение при добавяне на динозавър в зоопарка
* Структура **Food** пази храната и количеството ѝ от тип double като член-данни и съдържа метода void print();
* Структура **Parser** е необходима при интерпретирането на данните от тип string, които са въведени от потребителя, към съответно тип category\_t, era\_t, sex\_t, food\_t за правилното функциониране на програмата, а ако въведеният от потребителя низ не съотвтства на никоя от възможните стойности, които приемат съответните enum class, се извежда подходящо съобщение; данните ѝ са статични, за да може да има достъп до тях отвсякъде
* **Main.cpp** съдържа външните функции void static print\_help();, която функция извежда на конзолата помощно меню, показващо на потребителя кои функции се поддържат от програмата и как трябва да бъдат въведени от него със съответните параметри, и функцията void run\_program(Zoo& zoo);, приемаща аргумент по референция обект от клас Zoo и състояща се от последователност от условни оператори *if*, които проверяват дали въведената от потребителя команда съответства с поддържаните от програмата функции; ако съвпадат, се извежда подходящо съобщение и се извиква съответната функция. Освен това има и направени тестове, за да се покаже функциалността на програмата.
  1. **Диаграми**

Фигура 3.2.1 представлява блок схема на функцията void create\_dinosaur() в класа Zoo.

Начало

Въведете name, sex\_str, category\_str, era\_str, climate, type, food\_str

is\_food\_catogory\_compatible()

Не

Изведете „Enter another food type: “  
Въведете food\_str

Да

Dinosaur dinosaur (name, sex,  
 category, era, climate, type,  
 food);

Намали food

add\_dinosaur(dinosaur);

Край

Фигура 3.2.1

На фигура 3.2.2 може да се проследи обобщеното изпълнение на функциите, свързани със създаването на динозавър в различните класове.

**Zoo**

void create\_dinosaur();

**Zoo**

void add\_dinosaur(Dinosaur);

**Cage**category\_t get\_category();  
era\_t get\_era();  
string get\_climate();  
bool has\_free\_space();  
add\_new\_dinosaur(Dinosaur);

**Dinosaur**

category\_t get\_category();  
era\_t get\_era();  
string get\_climate();

**Zoo**

int get\_dinosaur\_count();

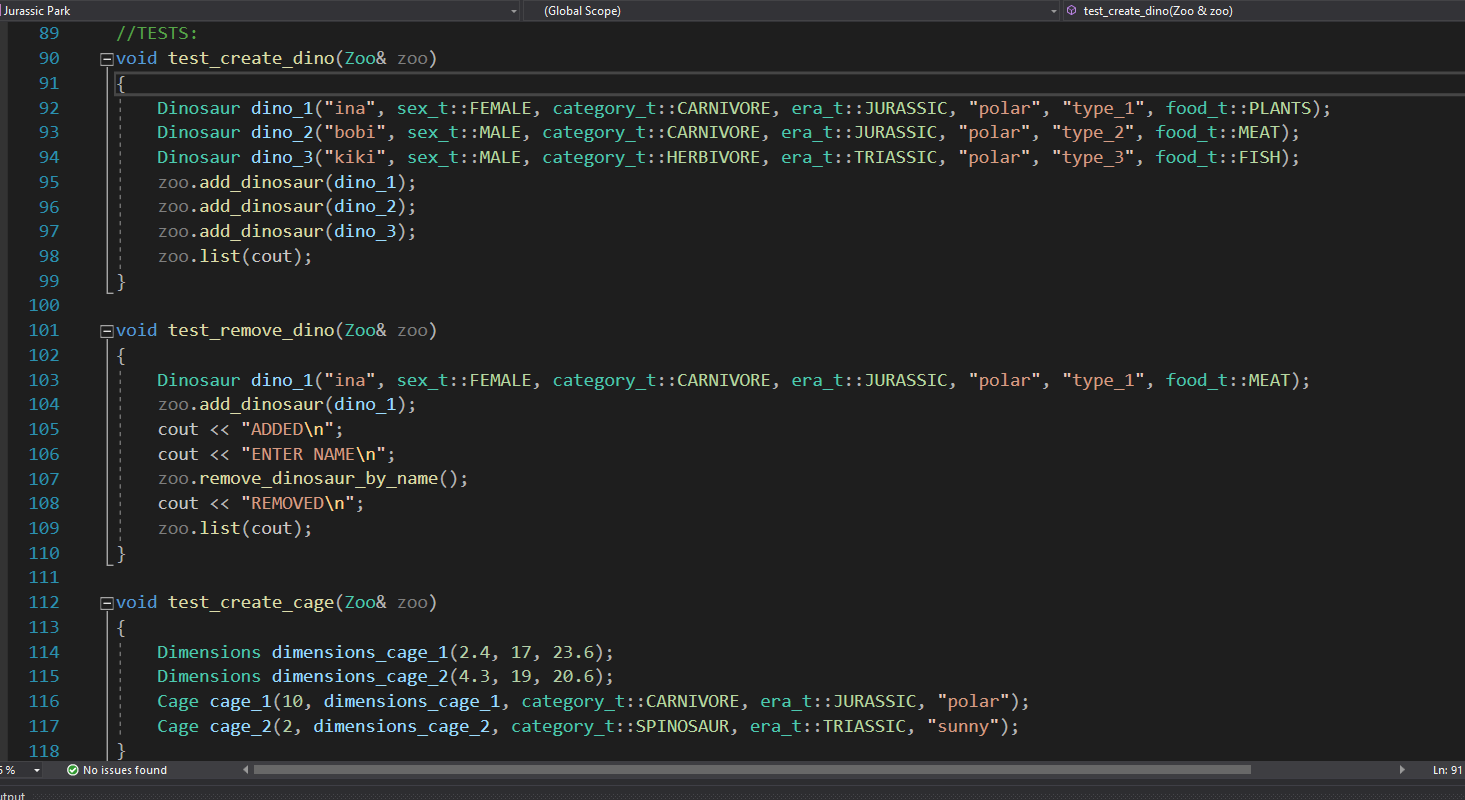
Фигура 3.2.2

1. **Реализация и тестване**
   1. **Управление на паметта и алгоритми**

Навсякъде паметта автоматично се заделя и унищожава от класа *vector* в стандартната библиотека, т.е. работим с динамична памет. Използваните алгоритми са с линейна сложност.

* 1. **Тестване**

При изготвянето на проекта в Main.cpp са направени и необходимите тестове (фигура 4.2.1), които да проверят правилното функциониране на функциите и на цялостната програма.



Фигура 4.2.1

1. **Заключение**
   1. **Бъдещо развитие и усъвършенстване**

С подходяща графика, звукови ефекти и разширение на функциалността този проект може да се разрасне и да се превърне в забавна онлайн игра за деца. В последствие може да бъде направено и приложение за телефон, за да бъде още по-достъпна.