**МИНОБРНАУКИ РОССИИ**

**Санкт-Петербургский государственный**

**электротехнический университет**

**«ЛЭТИ» им. В.И. Ульянова (Ленина)**

**Кафедра МО ЭВМ**

отчет

**по лабораторной работе №1**

**по дисциплине «Объектно-ориентированное программирование»**

Тема: Создание классов

|  |  |  |
| --- | --- | --- |
| Студент гр. 3342 |  | Иванов Д. М. |
| Преподаватель |  | Жангиров Т. Т. |

Санкт-Петербург

2024

## Цель работы

Изучить основы языка C++ и его отличия от C, основы ООП. На этом языке программирования с помощью инструментов ООП реализовать основные классы для игры “Морской бой”: корабль, менеджер кораблей, игровое поле. Затем связать их и протестировать работоспособность данных классов.

## Задание

а) Создать класс корабля, который будет размещаться на игровом поле. Корабль может иметь длину от 1 до 4, а также может быть расположен вертикально или горизонтально. Каждый сегмент корабля может иметь три различных состояния: целый, поврежден, уничтожен. Изначально у корабля все сегменты целые. При нанесении 1 урона по сегменту, он становится поврежденным, а при нанесении 2 урона по сегменту, уничтоженным. Также добавить методы для взаимодействия с кораблем.

б) Создать класс менеджера кораблей, хранящий информацию о кораблях. Данный класс в конструкторе принимает количество кораблей и их размеры, которые нужно расставить на поле.

в) Создать класс игрового поля, которое в конструкторе принимает размеры. У поля должен быть метод, принимающий корабль, координаты, на которые нужно поставить, и его ориентацию на поле. Корабли на поле не могут соприкасаться или пересекаться. Для игрового поля добавить методы для указания того, какая клетка атакуется. При попадании в сегмент корабля изменения должны отображаться в менеджере кораблей.

Каждая клетка игрового поля имеет три статуса:

-неизвестно (изначально вражеское поле полностью неизвестно),

-пустая (если на клетке ничего нет)

-корабль (если в клетке находится один из сегментов корабля).

Для класса игрового поля также необходимо реализовать конструкторы копирования и перемещения, а также соответствующие им операторы присваивания.

## Выполнение работы

1) Для обозначения переменных, которые принимает небольшое ограниченное количество значений, будем использовать enum. Пропишем их в файле Enum\_arguments.h:

Location — ориентация корабля (вертикальная или горизонтальная)

Length\_of\_the\_ship — длина корабля (от 1 до 4)

Statment\_of\_the\_ship — состояние сегмента корабля (нетронутый, поврежденный, уничтоженный)

Statment\_of\_the\_coord — состояние координаты на поле (пустая, неизветсная, сегмент корабля)

Coords — структура координат сегментов на поле (координаты x, y)

2) Создание класса корабля

Создадим заголовочный файл Ship.h. В нем пропишем поля и заголовки методов, определим для них модификаторы доступа.

private:

-Length\_of\_the\_ship length – длина корабля, количество его сегментов

-Statment\_of\_the\_ship\* segments – динамический массив, хранящий состояние каждого сегмента корабля

-Location location\_of\_ship – оентация корабля

public:

Ship(Length\_of\_the\_ship p\_length, Location p\_location\_of\_ship);

Ship(Length\_of\_the\_ship p\_length);

Ship();

Конструкторы класса, которые для объекта устанавливают значения для размера и ориентации. Выделяют место под динамический массив segments и заполняют его.

void shoot\_to\_segment(int coordinate) – метод нанесения удара по определенному сегмента корабля. Через static\_cast переводим значение сегмента в целое число, вычитаем, переводим обратно и присваиваем новое значение.

Location get\_location() – метод вывода значения ориентации корабля.

void set\_location(Location location) – установить новое значение расположения корабля на переданный аргумент.

int get\_length() – вывести размер корабля в виде целого числа.

Statment\_of\_the\_ship\* get\_segments() – вывести массив состояний сегментов.

void print\_statment\_of\_ship() – вывести в консоль состояния сегментов и ориентацию корабля в формате: “{сегмент\_1} {сегмент\_2} {сегмент\_N} | Location={расположение}”

~Ship() – деструктор класса. В нем происходит очистка динамического массива.

3) Создание класса менеджера кораблей.

В этом классе будет хранится множество кораблей. В это множество можно будет добавлять корабли или удалять их оттуда.

private:

int count = 0 – счетчик количества кораблей.

std::vector<Ship\*> ships – вектор, хранящий корабли.

Manager\_of\_ships(int ships\_count, Length\_of\_the\_ship\* sizes) – конструктор. Принимает число кораблей и массив их длин. В вектор ships добавляется каждый корабль. И изменяется значения счетчика кораблей.

std::vector<Ship\*> get\_arr\_of\_ships() – метод, возвращающий вектор, хранящий корабли.

int get\_count() – возвращает число кораблей в менеджере.

void add\_ship(Length\_of\_the\_ship length, Location location);

void add\_ship(Ship\* ship) – методы, добавляющие новые корабли в вектор. В данном случае пользуется перегрузка методов, и вызывается метод в зависимости от переданных аргументов. Если просто передается указатель на корабль, он добавляется в вектор. Если же принимаются длина и расположение, то выделяется память под новый корабль, и только потом он добавляется.

void remove\_ship(int index) – удаление корабля из менеджера по индексу с помощью функции erase.

void print\_ships() – вывод в консоль характеристики каждого корабля.

## Анализ полученных значений

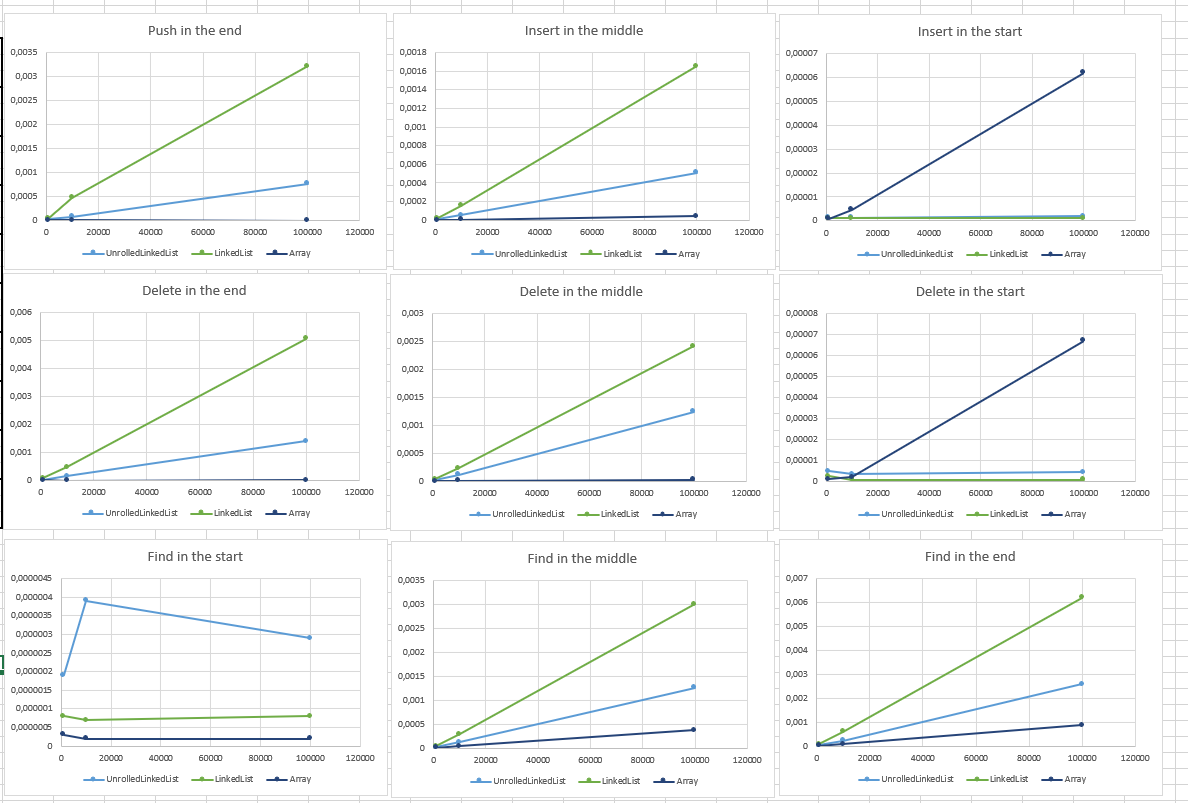
В конце необходимо оценить эффективность данной структуры, сравнив её с односвязанным списком и массивом (list() в Python). Для замерки времени работы использовалась библиотека time. Оценивались такие методы, как нахождение индексов элементов, вставка и удаление чисел в разных частях структур. Проверились операции на маленьких (1000), средних (10000) и больших (100000) наборах данных. Результаты представлены в виде графиков(см. ри. 1). Реализация была написана в файле time.py(код приводится в приложении)

Рисунок 1 – графики, показывающие время выполнения операций для различный структур

## Выводы

Была реализована такая структура данных, как развернутый связанный список. На языке Python написаны основные методы для нее. Также структура была проверена на работоспособность и эффективность по времени относительно массива и односвязанного списка. По результатам этих проверок можно сделать вывод, что реализованная нами структур в большинстве случаев эффективнее односвязанного списка, также при удалениях и вставках в начале быстрее массива.

# Приложение А Исходный код программы

Название файла: UnrolledLinkedList.py

from Calculate\_size import calculate\_optimal\_node\_size

class Node:

def \_\_init\_\_(self, array=[]):

self.arr = array.copy()

self.next = None

class UnrolledLinkedList:

def \_\_init\_\_(self, arr=[]):

self.head = None

self.length = 0

self.len\_of\_node\_array = calculate\_optimal\_node\_size(len(arr))

self.make\_linked\_list(arr)

def make\_linked\_list(self, arr):

if len(arr) == 0:

return

self.length = 0

k = 0

lis = []

index = 0

flag\_head = False

el = self.head

while index < len(arr):

if k < self.len\_of\_node\_array:

lis.append(arr[index])

k += 1

index += 1

if k >= self.len\_of\_node\_array:

if not flag\_head:

self.head = Node(lis)

self.length += 1

flag\_head = True

el = self.head

else:

el.next = Node(lis)

self.length += 1

el = el.next

lis = []

k = len(lis)

if len(lis) != 0:

if not flag\_head:

if len(lis) > self.len\_of\_node\_array:

self.head = Node(lis[:self.len\_of\_node\_array // 2])

self.head.next = Node(lis[self.len\_of\_node\_array // 2:])

return

self.head = Node(lis)

return

if el != None and len(el.arr) + len(lis) <= self.len\_of\_node\_array:

el.arr += lis

else:

el.next = Node(lis)

def push(self, element):

if self.head == None:

self.head = Node([element])

return True

el = self.head

while el.next != None:

el = el.next

if len(el.arr) + 1 > self.len\_of\_node\_array:

massive = el.arr

el.arr = massive[:self.len\_of\_node\_array // 2]

el.next = Node(massive[self.len\_of\_node\_array // 2:] + [element])

self.length += 1

return True

el.arr.append(element)

return True

def insert(self, element, index):

if index < 0:

return False

el = self.head

start\_index = len(el.arr)

while start\_index <= index:

el = el.next

if el == None:

return False

start\_index += len(el.arr)

if el is None:

return False

el.arr.insert(index - start\_index, element)

if len(el.arr) <= self.len\_of\_node\_array:

return True

half\_length = self.len\_of\_node\_array // 2

new\_array = el.arr[half\_length:]

el.arr = el.arr[:half\_length]

tmp = el.next

el.next = Node(new\_array)

el.next.next = tmp

self.length += 1

return True

def delete\_number(self, index):

el, start\_index, index\_node = self.find\_node\_by\_index\_of\_element(index)

if el == None:

return False

index\_node\_next = index\_node + 1

del el.arr[index - start\_index]

if len(el.arr) == 0:

self.delete\_arr(index\_node)

return True

if el.next != None:

if len(el.next.arr) + len(el.arr) <= self.len\_of\_node\_array:

el.arr += el.next.arr

self.delete\_arr(index\_node\_next)

return True

def find\_node\_by\_index\_of\_element(self, index):

index\_node = 0

if index < 0:

return None, -1, -1

el = self.head

k = len(el.arr)

while k <= index:

el = el.next

index\_node += 1

if el == None:

return None, -1, -1

k += len(el.arr)

return (el, k - len(el.arr), index\_node)

def delete\_arr(self, index):

if index == 0:

self.head = self.head.next

self.length -= 1

return

el = self.head

k = 0

while k < index - 1 and el.next != None:

el = el.next

k += 1

el.next = el.next.next

self.length -= 1

def find\_index\_of\_number(self, number):

k = 0

index = 0

el = self.head

while el != None:

if number in el.arr:

return index + el.arr.index(number)

index += len(el.arr)

el = el.next

k += 1

return None

def print\_list(self):

k = 0

el = self.head

while el != None:

string = ' '.join([str(x) for x in el.arr])

print(f"Node {k}: {string}")

el = el.next

k += 1

Название файла: test.py

from src.main import UnrolledLinkedList

def make\_list():

lis = [x for x in range(1, 40)]

List = UnrolledLinkedList()

for i in lis:

List.push(i)

return List

def test\_find\_start():

ListN = make\_list()

assert ListN.find\_index\_of\_number(1) == 0

def test\_find\_middle():

ListN = make\_list()

assert ListN.find\_index\_of\_number(14) == 13

def test\_find\_end():

ListN = make\_list()

assert ListN.find\_index\_of\_number(39) == 38

def test\_delete\_number\_with\_right\_index\_in\_the\_end():

ListN = make\_list()

assert ListN.delete\_number(38) == True

def test\_delete\_number\_with\_right\_index\_in\_the\_middle():

ListN = make\_list()

assert ListN.delete\_number(14) == True

def test\_delete\_number\_with\_right\_index\_in\_the\_start():

ListN = make\_list()

assert ListN.delete\_number(0) == True

def test\_delete\_number\_with\_wrong\_high\_index():

ListN = make\_list()

assert ListN.delete\_number(55) == False

def test\_delete\_number\_with\_wrong\_low\_index():

ListN = make\_list()

assert ListN.delete\_number(-2) == False

def test\_add\_numbers\_with\_right\_index\_int\_the\_middle():

ListN = make\_list()

assert ListN.insert(99, 3) == True

def test\_add\_numbers\_with\_right\_index\_int\_the\_start():

ListN = make\_list()

assert ListN.insert(66, 0) == True

def test\_add\_numbers\_with\_right\_index\_int\_the\_end():

ListN = make\_list()

assert ListN.push(77) == True

def test\_add\_numbers\_with\_wrong\_low\_index():

ListN = make\_list()

assert ListN.insert(123, -3) == False

def test\_add\_numbers\_with\_wrong\_high\_index():

ListN = make\_list()

assert ListN.insert(70, 61) == False

Название файла: main.py

from UnrolledLinkedList import UnrolledLinkedList

def check(arr1, arr2):

ListN = UnrolledLinkedList(arr1[:47])

print("List node size:", ListN.len\_of\_node\_array)

ListN.print\_list()

print('------------')

for i in arr1[47:50]:

ListN.push(i)

print("Push elements")

ListN.print\_list()

print('------------')

list\_for\_insert = [23, 49, 0]

for i in range(3):

ListN.insert(arr1[50+i], list\_for\_insert[i])

print(f"Insert {arr1[50+i]} to index {list\_for\_insert[i]}")

ListN.print\_list()

print('------------')

for i in arr2:

index = ListN.find\_index\_of\_number(i)

ListN.delete\_number(index)

print(f"Found {i} with index {index} and deleted it")

ListN.print\_list()

print('------------')

arr1 = [54, 22, 60, 68, 31, 5, 17, 37, 32, 46, 49, 6, 62, 77, 16, 19, 20, 14, 3, 56, 56, 11, 58, 77, 95, 44, 11, 21, 32, 88, 58, 23, 30, 41, 89, 48, 89, 29, 76, 45, 37, 99, 97, 17, 46, 59, 91, 21, 61, 36, 101, 102, 103]

arr2 = [68, 21, 54, 36]

check(arr1, arr2)check(arr1, arr2)

Название файла: time.py

from UnrolledLinkedList import UnrolledLinkedList

from LinkedList import LinkedList

import time

NUM = 100000

ListN = UnrolledLinkedList()

LinkedL = LinkedList()

Array = list()

start\_time = time.perf\_counter()

for i in range(NUM):

ListN.push(i)

end\_time = time.perf\_counter()

print(f"UnrolledListNode make: {end\_time - start\_time:.10f} seconds")

start\_time = time.perf\_counter()

for i in range(NUM):

LinkedL.append(i)

end\_time = time.perf\_counter()

print(f"LinkedList make: {end\_time - start\_time:.10f} seconds")

start\_time = time.perf\_counter()

for i in range(NUM):

Array.append(i)

end\_time = time.perf\_counter()

print(f"Array make: {end\_time - start\_time:.10f} seconds")

print('----------------------------')

start\_time = time.perf\_counter()

ListN.push(99)

end\_time = time.perf\_counter()

print(f"UnrolledListNode push: {end\_time - start\_time:.10f} seconds")

start\_time = time.perf\_counter()

LinkedL.append(99)

end\_time = time.perf\_counter()

print(f"LinkedList push: {end\_time - start\_time:.10f} seconds")

start\_time = time.perf\_counter()

Array.append(99)

end\_time = time.perf\_counter()

print(f"Array push: {end\_time - start\_time:.10f} seconds")

print('----------------------------')

start\_time = time.perf\_counter()

ListN.insert(99, NUM // 2)

end\_time = time.perf\_counter()

print(f"UnrolledListNode insert in the middle: {end\_time - start\_time:.10f} seconds")

start\_time = time.perf\_counter()

LinkedL.insert\_at\_index(NUM // 2, 99)

end\_time = time.perf\_counter()

print(f"LinkedList insert in the middle: {end\_time - start\_time:.10f} seconds")

start\_time = time.perf\_counter()

Array.insert(NUM // 2, 99)

end\_time = time.perf\_counter()

print(f"Array insert in the middle: {end\_time - start\_time:.10f} seconds")

print('----------------------------')

start\_time = time.perf\_counter()

ListN.insert(99, 0)

end\_time = time.perf\_counter()

print(f"UnrolledListNode insert in the start: {end\_time - start\_time:.10f} seconds")

start\_time = time.perf\_counter()

LinkedL.insert\_at\_index(0, 99)

end\_time = time.perf\_counter()

print(f"LinkedList insert in the start: {end\_time - start\_time:.10f} seconds")

start\_time = time.perf\_counter()

Array.insert(0, 99)

end\_time = time.perf\_counter()

print(f"Array insert in the start: {end\_time - start\_time:.10f} seconds")

print('----------------------------')

start\_time = time.perf\_counter()

ListN.delete\_number(NUM + 2)

end\_time = time.perf\_counter()

print(f"UnrolledListNode delete in the end: {end\_time - start\_time:.10f} seconds")

start\_time = time.perf\_counter()

LinkedL.remove\_at\_index(NUM + 2)

end\_time = time.perf\_counter()

print(f"LinkedList delete in the end: {end\_time - start\_time:.10f} seconds")

start\_time = time.perf\_counter()

del Array[NUM + 2]

end\_time = time.perf\_counter()

print(f"Array delete in the end: {end\_time - start\_time:.10f} seconds")

print('----------------------------')

start\_time = time.perf\_counter()

ListN.delete\_number(0)

end\_time = time.perf\_counter()

print(f"UnrolledListNode delete in the start: {end\_time - start\_time:.10f} seconds")

start\_time = time.perf\_counter()

LinkedL.remove\_at\_index(0)

end\_time = time.perf\_counter()

print(f"LinkedList delete in the start: {end\_time - start\_time:.10f} seconds")

start\_time = time.perf\_counter()

del Array[0]

end\_time = time.perf\_counter()

print(f"Array delete in the start: {end\_time - start\_time:.10f} seconds")

print('----------------------------')

start\_time = time.perf\_counter()

ListN.delete\_number(NUM // 2)

end\_time = time.perf\_counter()

print(f"UnrolledListNode delete in the middle: {end\_time - start\_time:.10f} seconds")

start\_time = time.perf\_counter()

LinkedL.remove\_at\_index(NUM // 2)

end\_time = time.perf\_counter()

print(f"LinkedList delete in the middle: {end\_time - start\_time:.10f} seconds")

start\_time = time.perf\_counter()

del Array[NUM // 2]

end\_time = time.perf\_counter()

print(f"Array delete in the middle: {end\_time - start\_time:.10f} seconds")

print('---------------------------')

start\_time = time.perf\_counter()

ListN.find\_index\_of\_number(0)

end\_time = time.perf\_counter()

print(f"UnrolledListNode find in the start: {end\_time - start\_time:.10f} seconds")

start\_time = time.perf\_counter()

LinkedL.find\_index\_by\_value(0)

end\_time = time.perf\_counter()

print(f"LinkedList find in the start: {end\_time - start\_time:.10f} seconds")

start\_time = time.perf\_counter()

Array.index(0)

end\_time = time.perf\_counter()

print(f"Array find in the start: {end\_time - start\_time:.10f} seconds")

print('---------------------------')

start\_time = time.perf\_counter()

ListN.find\_index\_of\_number(NUM // 2)

end\_time = time.perf\_counter()

print(f"UnrolledListNode find the middle: {end\_time - start\_time:.10f} seconds")

start\_time = time.perf\_counter()

LinkedL.find\_index\_by\_value(NUM // 2)

end\_time = time.perf\_counter()

print(f"LinkedList find in the middle: {end\_time - start\_time:.10f} seconds")

start\_time = time.perf\_counter()

Array.index(NUM // 2)

end\_time = time.perf\_counter()

print(f"Array find in the middle: {end\_time - start\_time:.10f} seconds")

print('---------------------------')

start\_time = time.perf\_counter()

ListN.find\_index\_of\_number(NUM - 1)

end\_time = time.perf\_counter()

print(f"UnrolledListNode find the end: {end\_time - start\_time:.10f} seconds")

start\_time = time.perf\_counter()

LinkedL.find\_index\_by\_value(NUM - 1)

end\_time = time.perf\_counter()

print(f"LinkedList find in the end: {end\_time - start\_time:.10f} seconds")

start\_time = time.perf\_counter()

Array.index(NUM - 1)

end\_time = time.perf\_counter()

print(f"Array find in the end: {end\_time - start\_time:.10f} seconds")

print('---------------------------')

Название файла: Calculate\_size.py

from math import ceil

def calculate\_optimal\_node\_size(num\_elements):

memory = num\_elements \* 4

return ceil(memory / 64) + 1