Study-Unit Assignment

Liam Bugeja Douglas

CPS2004 Object-Oriented Programming

1/20/22

Table of Contents

Task1: DAG..... **2**

Task2: Crypto Exchange..... **4**

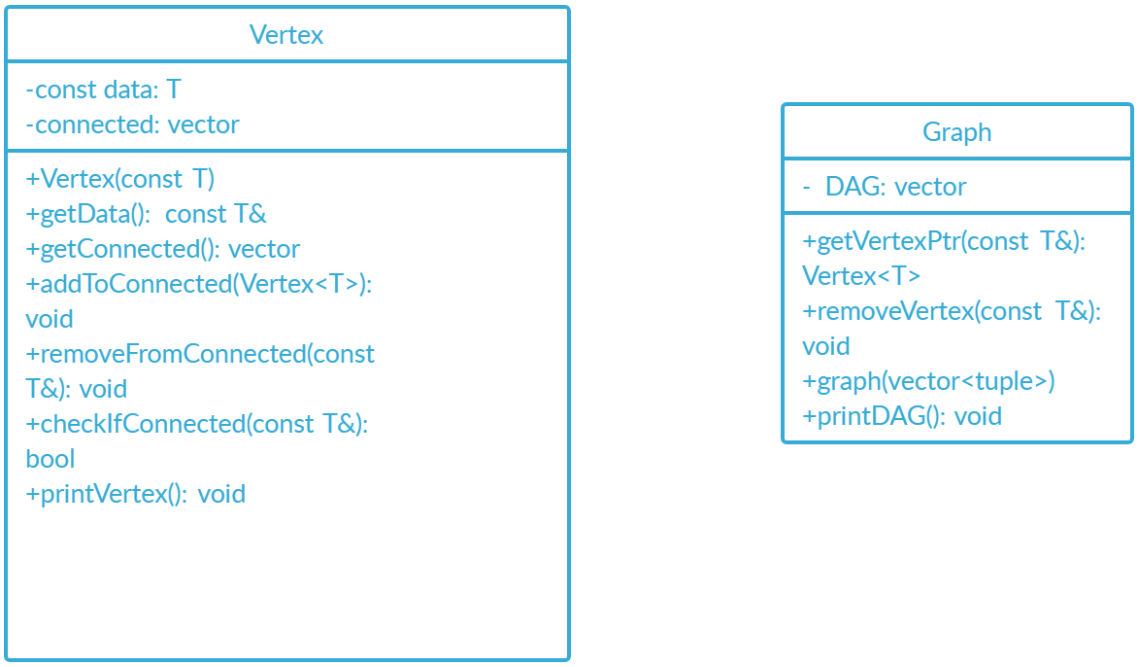
Task3: BigInt…….……**9**

Sources……...…11

Plagarisim…….12

Task1: DAG

UML Diagram:



Approach:

Since a DAG is a graph that should not contain cycles and the links should contain a direction, I assumed each vertex should have a vector of all the vertices it points to. Therefore only two classes were need, graph class and vertex class.

Vertex Class

The vertex class contains two private varaiables: data and connected, data is the value that the vertex would have and connected is a vector wich would contain the pointers of other vertcies. These would only include he vertices that are connected to a specific vertex. The class also contains methods to get the data and vector (getters), to add pointers in the connected vector, to remove pointers from the connected vector and a method to print a specific vertex and vertices it points to. Finally to move ownership of the vertex to the DAG std::move was used.

Graph Class

The graph class contains 1 privaet variable called DAG and is of type vector and data type vertex<T>. The class contains the following methods: the abilit to return a pointer of a specific vertex by finding the same value of data, removal of a specific vertex from the dag and deleting it if found in the connected vertices of other vertices. A graph method is also used in which it acceptsa pair of values, in this method it first moves the ownership to the

DAG then it check if the vertices already exist. If not then it inserts them in the DAG and adds the second value to the connected vector of the first value. Finally a print method is also used in which it loops through the DAG and prints the vertecies by calling the vertex print method of all vertecies.

Testing:

Entering integer values:

```
//Creating link vector that accepts int values
std::vector<std::tuple<int, int>> link1;

//Inserting int values in link2
link1.push_back( std::make_tuple(1, 2));
link1.push_back( std::make_tuple(1, 3));
link1.push_back( std::make_tuple(1, 4));
link1.push_back( std::make_tuple(3, 5));
link1.push_back( std::make_tuple(3, 6));
link1.push_back( std::make_tuple(6, 7));
link1.push_back( std::make_tuple(6, 8));
link1.push_back( std::make_tuple(5, 1));
```

```
---DAG---
1: [4][3][2]
2:
3: [6][5]
4:
5: [1]
6: [8][7]
7:
8:
---------
```

Entering char values:

```
//Creating link vector that accepts char values
std::vector<std::tuple<char, char>> link2;

//Inserting char values in link1
link2.push_back(std::make_tuple('a','b'));
link2.push_back(std::make_tuple('a','c'));
link2.push_back(std::make_tuple('c','d'));
link2.push_back(std::make_tuple('b','e'));
link2.push_back(std::make_tuple('e','f'));
link1.push_back(std::make_tuple('e','g'));
link2.push_back(std::make_tuple('d','a'));
```

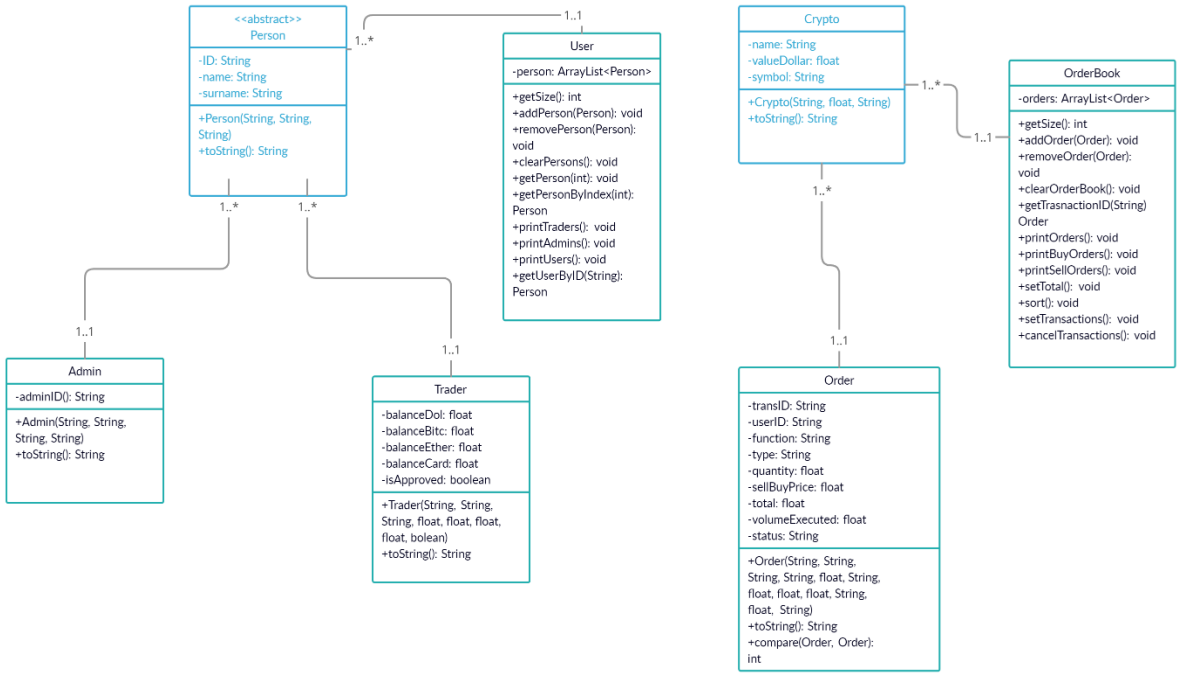
```
---DAG---
a: [c][b]
b: [e]
c: [d]
d: [a]
e: [f]
f:
---------
```

Limitations:

The system contains two limitations: The first one is that it does not contain a separate main.cpp file as well as not having a header file. The second limitation is that it does not check for cycles, I tried different methods but all failed stating that no cycle was found.

Task 2: Crypto Exchange

UML Diagram:



Approach:

The system was split into two main categories: Users and Orders. The Users are the traders which can trade in the crypto exchange and the administrators which approve traders to trade in the exchange. The Orders contains orders made by traders which can be either Buy or Sell orders having a market or limit price, the limit price can be bid or ask respectively. The system was designed in such a way to facilitate future changes by including a number of methods that are not needed currently. These include all the getter and setter methods as well as many Array List functions such as printing of specific types and removal methods.

Person

This class is of type abstract so that it cannot be accidently instantiated, it contains the details needed for the two subclasses “Trader” and “Admin”. The class contains 3 private variables: ID, name and surname all of which are all strings. It also contains the following methods: the default constructor, an instance of the class Person, getters and setters and finally the toString method.

Admin

The class is a subclass of the abstract class Person, therefore it contains the values from the person class and another value called adminID which is of type string. To get the person variables the super keyword is used. The class contains the following methods: the default constructor, an instance of the admin class, getter and setter and toString method. The toString method also uses the super keyword to get the values from the parent class Person.

Trader

The class trader is a subclass of the abstract class Person, so it also contains variables from the Person class. The class also contains the following private variables: balanceDol, balanceBitc, balanceEther, balanceCard all of type float and finally isApproved of type Boolean. the float values are used to save the balance information of the user and the isApproved variable is a check to see if the user is eligible to trade in the crypto exchange. The class them contains the following methods: the default constructor, an instance of the class, the getters and setters and the toString method. Like the Amdin class to get the variables from the Person class the super keyword is used, it is also used in the toString method.

Crypto

The class crypto is a parent class and is used as a blueprint to create crypto currencies, the class contains the following private variables: name and symbol which are of type string and valueInDol which is of type float. The class contains the following methods: default constructor, instance of the class, getter and setters and the toString method.

Order

The class order is a subclass of the crypto class, since an order must contain a crypto exchange, it made sense to do this. The class contains the following private variables: transID, userID, function, type, status all of type string and quantity, buySellPrice, total, volumeExecuted of type float. The super class variables were used using the super keyword. The class contained the following methods: default constructor, instance of the class, getters and setter and toString method using the super keyword to get the parent class toString method as well.

User

This class contains the one private variable called person which is an Array List of type Person this was done so that bought admins and traders can be stored in the same structure. With this Array List the system can control the information of the users which can be done using the following methods: adding and removing Users, finding Users by ID and printing specific users.

OrderBook

This class contains one private variable called orders which is an Array List of type Order, with this, orders which can be either buy or sell can be stored inside a single structure. The class has the following methods allowing the system to use the Array List: adding and removing orders, printing orders or specific types of orders, clearing the Array List, soring the Array list by ascending price and a method called setTransactions. The setTransactions method is the matching engine of the system, it check the following criteria before accepting the transaction: one order must be buy and the other one sell, both having the same coin, both having a quantity larger than 0> and the buy price must be greater or equal to the sell price.

Testing:

Creating a user and inserting him inside an Array List of type Users:

```
Trader t1 = new Trader("1324TR", "Jack", "Galea",1900.12f, 12f, 21f, 78f, true);
```

```
Users us = new Users();
us.addPerson(t1);
```

```
ID: 1324TR
Name: Jack
Surname: Galea
Balance Dollar: $1900.12
Balance Bitcoin: B12.0
Balance Etherum: E21.0
Balance Cardano: C78.0
Approved by Admin: true
```

Creating an order and inserting it inside an Array List of type OrderBook:

```
Order o1 = new Order("OB001", t1.getID(), "Buy", "Market", 5f, c1.getName(), c1.getValDollar(), c1.getValDollar(), 0f, c1.getSymbol(), 0f, "Active");
```

```
OrderBook or = new OrderBook();
or.addOrder(o1);
```

```
TransID: OB001
Function: Buy
Type: Market
Symbol and name of coin: B Bitcoin
Market value of coin in Dollar: $35000.0
Quantity: 5.0
Sell/Buy Price of 1 coin: $35000.0
Total Price in Dollar: $175000.0
Volume Executed: $0.0
Status: Active
```

Testing if two orders cancel out:

```
Order o1 = new Order("OB001", t1.getID(), "Buy", "Market", 5f, c1.getName(), c1.getValDollar(), c1.getValDollar(), 0f, c1.getSymbol(), 0f, "Active");
Order o2 = new Order("OB002", t2.getID(), "Sell", "Market", 5f, c1.getName(), c1.getValDollar(), c1.getValDollar(), 0f, c1.getSymbol(), 0f, "Active");
```

```
TransID: OB001
Function: Buy
Type: Market
Symbol and name of coin: B Bitcoin
Market value of coin in Dollar: $35000.0
Quantity: 0.0
Sell/Buy Price of 1 coin: $35000.0
Total Price in Dollar: $175000.0
Volume Executed: $175000.0
Status: Complete

TransID: OB002
Function: Sell
Type: Market
Symbol and name of coin: B Bitcoin
Market value of coin in Dollar: $35000.0
Quantity: 0.0
Sell/Buy Price of 1 coin: $35000.0
Total Price in Dollar: $175000.0
Volume Executed: $175000.0
Status: Complete
```

Testing partial Orders:

```
Order o3 = new Order("OB003", t2.getID(), "Sell", "Limit", 20f, c3.getName(), c3.getValDollar(), 600f, 0f, c3.getSymbol(), 0f, "Active");
Order o4 = new Order("OB004", t1.getID(),"Buy", "Limit", 10f, c3.getName(), c3.getValDollar(), 700f, 0f, c3.getSymbol(), 0f, "Active");
```

```
TransID: OB003
Function: Sell
Type: Limit
Symbol and name of coin: C Cardano
Market value of coin in Dollar: $500.0
Quantity: 10.0
Sell/Buy Price of 1 coin: $600.0
Total Price in Dollar: $12000.0
Volume Executed: $7000.0
Status: Partial
```

```
TransID: OB004
Function: Buy
Type: Limit
Symbol and name of coin: C Cardano
Market value of coin in Dollar: $500.0
Quantity: 0.0
Sell/Buy Price of 1 coin: $700.0
Total Price in Dollar: $7000.0
Volume Executed: $7000.0
Status: Complete
```

Cancelling incomplete orders:

```
TransID: OB003
Function: Sell
Type: Limit
Symbol and name of coin: C Cardano
Market value of coin in Dollar: $500.0
Quantity: 5.0
Sell/Buy Price of 1 coin: $600.0
Total Price in Dollar: $12000.0
Volume Executed: $7000.0
Status: Cancelled
```

Limitations and extras:

As previously stated, many functions were created with the intent for future use such as printing specific types of orders (Buy and Sell) or persons (Traders and Users) and clearing the Array Lists. Moreover, each order has much more detail than currently needed such as: market of value of coin in dollar, total transaction price. These were added so that when creating each log for the FATF the transaction can have as much detail as possible.

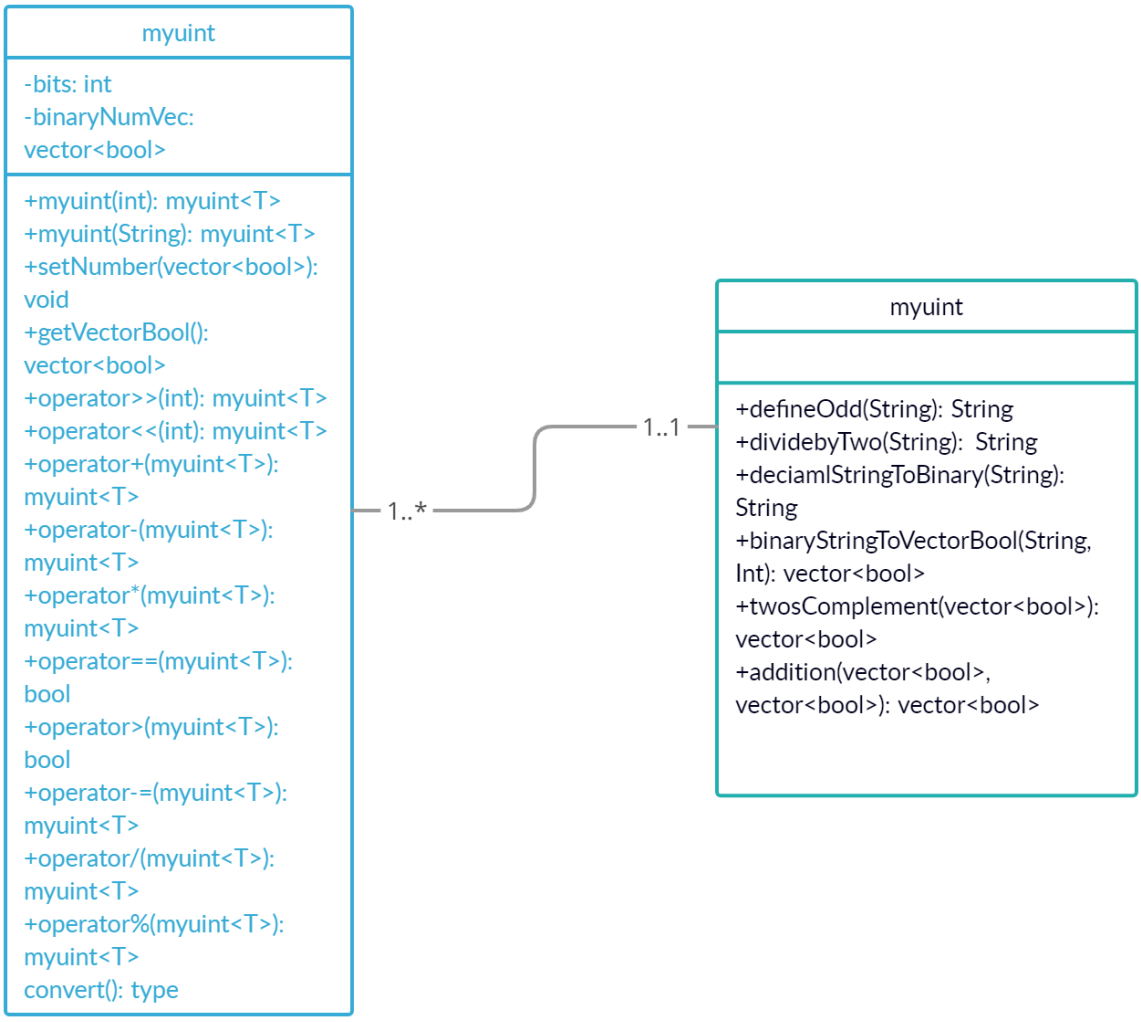
The system has two main limitations, the first one is that administrators do not approve users to trade in the exchange and secondly the system does not add or subtract FIAT and crypto currencies when transactions are complete or partially complete from the users account.

Response to the FATF new regulation:

Since each user action must produce a log that cannot in any way be tampered with, we can create another private Array List that can store each transaction that is done. This structure would not be visible to Users by not creating an instance of it. Due to adding the extra features this new regulation would not take too much time to implement in the system.

Task 3: Big Integers

UML Diagram:



Approach:

Since we could not use bitsets or strings the next best approach in my opinion was to use a vector Boolean structure due to two main reasons. The first being that the space complexity would always be O(n) (n being the size given by the user) and secondly all decimal operations can be easily done in Boolean form. Even though strings cannot be used, it was clarified that if an input is too large to store in int or long data types, strings could be used.

Myuint

So, the first thing to be done was to overload a method that could accept strings and integers and transform the number from decimal to Boolean vector. This overloading method saved a lot of space since there was no need for multiple operator methods to be done for different data types. Inside each method the astatic_assert function was used to check if the size given by the user is 2^n, this function would check at the compiling stage and would not let the user run the code if the number given for is not 2^n. Next it converts the number into a bool string by inserting

a 1 or 0 at the end of the string if the number odd or even respectively and divides it by two. Finally, the string is converted into a vector Boolean by loop through the vector and inserting the 1's and 0's. With this the methods can accept vector Booleans and could be able to work out any number.

The following are the mathematical operators included in the system: bit shift left, bit shift right, addition, subtraction, multiplication, division and modulus.

Testing:

Testing bit shift left and right:

```
myuint<8> a(10);
myuint<8> b = a << 1;
myuint<8> c = a >> 1;
std::cout << b.convert<int>() << std::endl;
std::cout << c.convert<int>() << std::endl;
```

```
liam@DESKTOP-3904MQ9:~/OOP$ ./a.out
20
5
```

Testing subtraction and addition:

```
myuint<8> a(25);
myuint<8> b = a + 30;
myuint<8> c = a - 10;
std::cout << b.convert<int>() << std::endl;
std::cout << c.convert<int>() << std::endl;
```

```
liam@DESKTOP-3904MQ9:~/OOP$ ./a.out
55
15
```

Testing division and modulus:

```
std::string num1 = "10";
std::string num2 = "3";

myuint<1024> a(num1);
myuint<1024> b = a / num2;
myuint<1024> c = a % num2;
std::cout << b.convert<int>() << std::endl;
std::cout << c.convert<int>() << std::endl;
```

```
liam@DESKTOP-3904MQ9:~/OOP$ ./a.out
3
1
```

Testing multiplication:

```
std::string num1 = "10";
std::string num2 = "3";

myuint<128> a (num1);
myuint<128> b = a * (num2);
std::cout << b.convert<int>() << std::endl;
```

```
liam@DESKTOP-3904MQ9:~/OOP$ ./a.out
30
```

Testing for non 2^n numbers:

```
//Checking if static_assert works
myuint<17> e(30);
```

```
myuint.cpp:261:34: error: static assertion failed: myuint only accepts unsigned integers or strings, and must be 2^n bits
  261 |     static_assert(floor(log2(T)) == log2(T), "myuint only accepts unsigned integers or strings, and must be 2^n bits");
      |                   ~~~~~~~~~~~~~~~~~~~~~~~^~~~~~~~~~~~
```

Limitations:

Current limitations of the system are that the implementation of a main file is not included, also I believe that the methods to turn a string or int into a binary vector should also have their own .cpp file. Lastly, when trying to using different sizes of myuint<T> in calculations the compiler inputs an error stating different sizes cannot be worked out.

Sources

- https://www.creative-proteomics.com/services/directed-acyclic-graph-dag-service.htm - Used to understand the concept of a DAG.
- https://www.geeksforgeeks.org/smart-pointers-cpp/ - Used to understand smart pointers, and how to implement unique pointers.
- https://www.javatpoint.com/java-arraylist#:~:text=The%20ArrayList%20maintains%20the%20insertion,class%20can%20contain%20duplicate%20elements. -Used to find useful Array List functions.
- https://www.javatpoint.com/Comparator-interface-in-collection-framework - Used to sort the orders by price.
- https://stackoverflow.com/questions/25460192/convert-vectorbool-to-decimal-as-string - Used to understand how to transform a string decimal number to a Boolean vector.

GitHub Link

https://github.com/LiamBugejaDouglas/CPS2004

FACULTY OF INFORMATION AND

COMMUNICATION TECHNOLOGY

Declaration

Plagiarism is defined as “the unacknowledged use, as one’s own work, of work of another person, whether or not such work has been published” (Regulations Governing Conduct at Examinations, 1997, Regulation 1 (viii), University of Malta).

I the undersigned, declare that the assignment submitted is my work, except where acknowledged and referenced.

I understand that the penalties for making a false declaration may include, but are not limited to, loss of marks; cancellation of examination results; enforced suspension of studies; or expulsion from the degree programme.

Work submitted without this signed declaration will not be corrected, and will be given zero marks.

Liam Bugeja Douglas

Student Name	Signature
CPS2004	CPS2004 Object-Oriented Programming
Course Code	Title of work submitted

20/01/2022

_____

Date