National University of Computer & Emerging Sciences, Karachi
School of Computing (BSCS, BSCS-R, BSCY, BSSE, BSAI)
FINAL EXAMINATION SPRING-2023
22nd May 2023, 12:00 noon – 03:00 pm

Course Code: CS1004	**Course Name: Object Oriented Programming**
Instructors Name: Dr. Farooque Hassan Kumbhar, Dr. Abdul Aziz, Mr. Zain-ul-Hassan, Ms. Abeer Gauher, Mr. Basit Ali, Ms. Sobia Iftikhar, Ms. Aqsa Zahid, Ms. Sumaiyah, Ms. Abeeha Sattar, Ms Javeria Farooq, Mr. Shahroz Bakht, Ms. Eman Shahid	
Student Roll No:	**Section No:**

Instructions:

- Return the question paper and make sure to keep it inside your answer sheet.
- Read questions completely before answering. There are **5 questions, 6 sides on 3 pages**.
- In case of any ambiguity, you may make assumptions. But your assumption should not contradict any statement in the question paper.
- You are **not allowed to write** anything on the question paper (except your ID and section).

Time: 180 minutes. **Max Marks**: 104 Marks

Q1: [30 min, 20 Marks, CLO 1] Write on the answer sheet the output of the following programs, when they are executed. There are no compilation errors in the programs.

A.
```
#include<iostream>
using namespace std;
class Point {
private:
    int x, y;
public:
     Point(){Point(1,1);}
    Point(int i, int j);
    Point(const Point &t);
};
Point::Point(int i, int j)  {
    x = i;
    y = j;
    cout  <<  x<<"  "<<y<<"Normal Constructor called\n";
}
Point::Point(const Point &t) {
   y = t.y;
   cout   <<   y   <<"   "<<"Copy Constructor called\n";
}
int main()
{
   Point *t1, *t2;
   t1 = new Point(10, 15);
   t2 = new Point(*t1);
   Point t3 = *t1;
   Point t4;
   t4 = t3;
   return 0;
}
```

10 15Normal Constructor called
15 Copy Constructor called
15 Copy Constructor called
1 1Normal Constructor called

B.
```
#include<iostream>
using namespace std;
int func(int n){
     if(n==0){
          cout<<"The    value    is zero"<<endl;
          return 10;}
     if(n>0){
          cout<<"The    value    is greater than zero"<<endl;
          throw 'e';}
     if(n<0){
          cout<<"The value is less than zero"<<endl;
          throw 9.9f;}
}
int main(){
    try {
      func(0); func(10); func(-10);
    } catch (int x) {
      cout << "Catching Integer\n";
      func(10);
    } catch (float f) {
      cout << "Catching Float\n";
      func(10);
    } catch (char c) {
      cout << "Catching Character\n";
      func(10);
}
```

C.

```
#include <iostream>
class Base {
public:
    static int count;
    Base() {count++;}
    virtual ~Base() {count--;}
    static void printCount() {
    std::cout << "Count: " << count << std::endl;
    }
};
class Derived : public Base {
public:
    Derived() {count++;}
    ~Derived() {count--;}
};
int Base::count = 0;
int main() {
    Base::printCount();
    {
        Base obj1;
        Derived obj2;
        Base::printCount();
    }
    Base::printCount();
    return 0;
}
```

D.

```
#include<iostream>
using namespace std;
class A
{
public:
    A (){ cout << "\n A's constructor"; }
    A (const A &a) { cout << "\n A's Copy constructor";}
    A& operator = (const A &a)
    {
       if(this == &a) return *this;
       cout << "\n A's Assignment Operator"; return *this;
    }
};
class B
{
    A a;
public:
    B(A &a) { this->a = a; cout << "\n B's constructor"; }
};
int main()
{
    A a1;
    B b(a1);
    return 0;
}
```

Q2: [30 min, 20 Marks, CLO 2] Considering the output given, complete the following code snippets.

A.

```
#include <iostream>
class Base {
private:
    int data;
public:
    Base(int value) : data(value) {}
    void printData() {
        std::cout << "Data: " << data << std::endl;
    }
_____________________
_____________________
_____________________
};
int main() {
    Base obj1(10);
    Base obj2(20);
    Base result = obj1 + obj2;
    result.printData();
    return 0;}
```

```
Output: 30
```

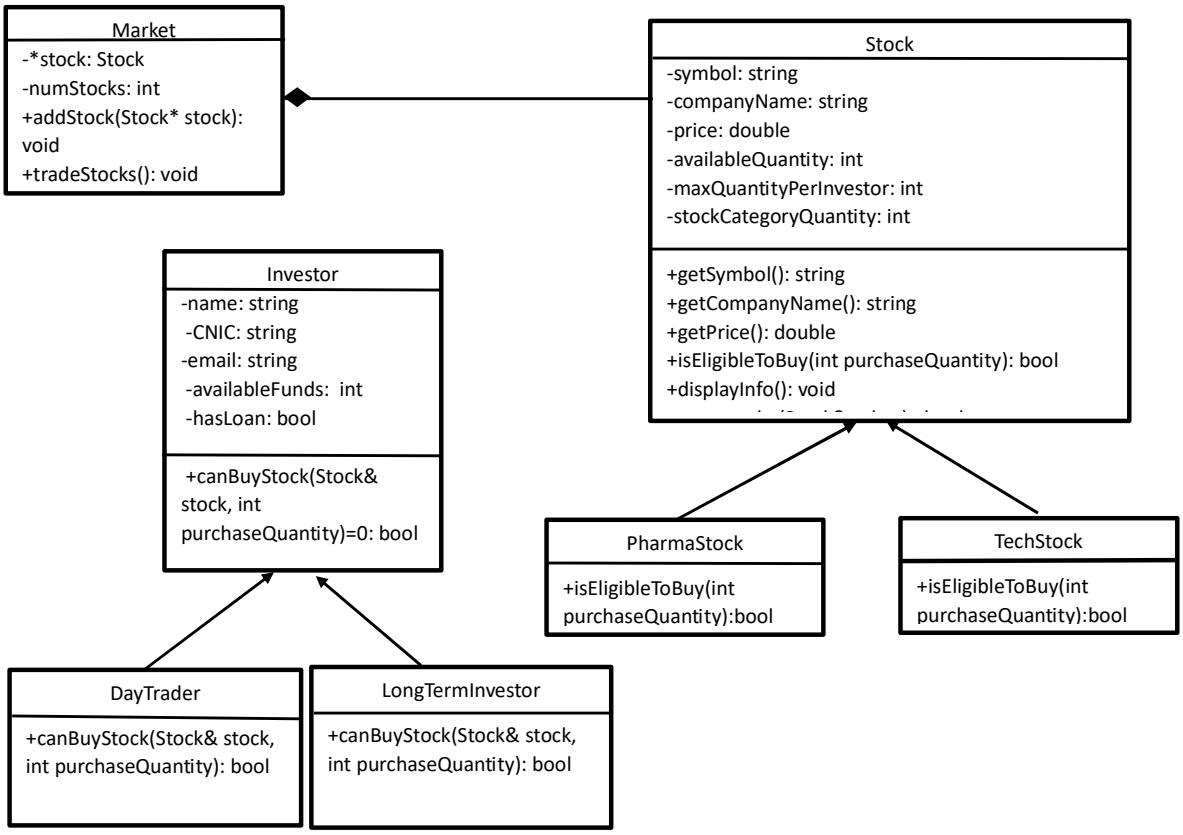
B. #include<iostream>
_____
_____
_____
int main() {
int intMax = maximum<int>(5, 10);
std::cout << "Max of 5 and 10: " << intMax << std::endl;
double doubleMax = maximum<float>(3.14, 2.71);
std::cout << "Max of 3.14 and 2.71: " << doubleMax << std::endl;
char charMax = maximum<char>('a', 'z');
std::cout << "Max of 'a' and 'z': " << charMax << std::endl;
return 0;
}
	Output:
	Maximum of 5 and 10: 10
	Maximum of 3.14 and 2.71: 3.14
	Maximum of 'a' and 'z': z

C. #include <iostream>
#include <fstream>
#include <string>
 	
class Person {
protected:
std::string name;
int age;
public:
Person(const std::string& name, int age) : name(name), age(age) {}
virtual void saveToFile(const std::string& filename) const = 0;
};
class Student : public Person {
private:
std::string university;
public:
_____
_____
_____
_____
_____
_____
_____
};
int main() {
Student student("Ali", 21, "FAST NUCES");
student.saveToFile("student.txt");
return 0;
}
	Output:
	Data saved to file with details:
	Ali
	21
	FAST NUCES

D.

```
#include <iostream>
template<typename T>
class Pair {
private:
    T first;
    T second;
public:
    Pair(T f, T s) : first(f), second(s) {}
    T sum() const {
        return first + second;
    }
_____________________
_____________________
_____________________
_____________________
_____________________
int main() {
    Pair<int> p(5, 7);
    std::cout << "Sum: " << p.sum() << std::endl;
    std::cout<<"Average: "<<average(p)<< std::endl;
}
```

```
Output:
Sum: 12
Average: 6
```



Q3: [25 min, 10 Marks, CLO 3] Consider the above given class diagram that demonstrates a stock market scenario. You are required to write the skeleton of the classes (i.e., member variables and function signatures) as shown in the above given class diagram. The parameterized constructor in all inherited classes must call parent's constructor to initialize variables. Create getter methods for all the attributes in all the classes. You are also required to define following functions:

- Stock::displayInfo() const: Displays the information about the stock. Prints the symbol, company name, price, available quantity, max quantity per investor, and category quantity of the stock.
- Market::getNumStocks() const: Gets the number of stocks currently in the market. Returns An integer representing the number of stocks in the market.

Q4: [50 min, 10+10+4 Marks, CLO 4] Use the class diagram and your answer of Q3 as reference and write programs for the following.

A. Implement functions as described below.

Stock::isEligibleToBuy() const: Following checks must be confirmed to check the buying eligibility of a stock. A stock is eligible for purchase if all conditions are met.

- Purchase quantity should not exceed maximum quantity limit per investor. Also, purchase quantity should not exceed the available quantity of the stock. If the purchaseQuantity is less than or equal to zero, then display an error message stating that the purchase quantity is invalid and return false.

TechStock::isEligibleToBuy(): Along with all parent class conditions, eligibility to buy of TechStock includes following checks.

- The purchase quantity must be a multiple of 10 for TechStock. If not, it displays an error message and returns false. Also, if the purchaseQuantity is greater than 100, it will display an error message stating that the maximum purchase quantity for TechStock is 100 and return false.

PharmaStock::isEligibleToBuy(): Along with all parent class conditions, eligibility to buy of PharmaStock includes following checks.:

- The purchase quantity must be at least 50 for PharmaStock. If not, it displays an error message and returns false. If the purchaseQuantity is not a multiple of 5, it will display an error message indicating that the quantity must be a multiple of 5 for PharmaStock and return false.

B. Implement functions as described below.

Investor::canBuyStock() is pure virtual function.

DayTrader::canBuyStock(): Checks if the investor can buy a given stock based on their financial status. If the investor has availed a loan, it displays an error message and returns false. The function should also calculate the total price of the stock purchase and check if it exceeds the available funds of the day trader. If so, it displays an error message and returns false. Finally, uses isEligibleToBuy() method of the stock and returns its response.

LongTermInvestor::canBuyStock(): Checks if the investor can buy a given stock based on their financial status. If the LongTermInvestor has availed a loan and has availableFunds less than 50000, it displays an error message and returns false. The function should also check that the purchaseQuantity does not exceed the maxQuantityPerInvestor limit. If it does, then it displays an error message and returns false. Finally, uses isEligibleToBuy() method of the stock and returns its response.

C. Overloaded inequality operator to compare stocks based on their symbols, i.e. the following statement should work: bool isNotEqual = stockObject1 != stockObject 2.

A stock is not equal to another stock if either the name of the stock company or the stock symbol is different from the other stock.

Q5: [45 min, 10+10+10 Marks, CLO 5] Consider a chatbot system designed to provide responses to users' queries. The system consists of four chatbot variants tailored for medical, technology, legal, and general queries. Your task is to implement an object-oriented program that fulfills the following requirements:

A. Design a User class to store user data, including attributes such as username, country, interest, and age. User have a method Ask() which takes a string as a query and generates a specific response.

- The medical chatbot should respond if the query's prefix (first word) is "doc". The legal chatbot should respond if the query begins with "attorney". The technology chatbot should respond only if the query starts with "guru".

- If a query begins with "special", check the user's interest, and forward the query to the relevant chatbot variant based on their interest.
- If a query does not match the relevant prefix, it must throw a custom exception object of type "Bot_Exception" with an error message.

B. Implement a Chatbot class as the base class for all chatbot variants. Each chatbot variant (MedicalChatbot, TechnologyChatbot, LegalChatbot, GeneralChatbot) should be inherited from the Chatbot class. Make sure that each chatbot class should keep track of the number of instances created throughout the program.
   - Each chatbot variant should have a method **string generate_response(string query, User u)** to generate responses based on user queries. It should store the name of the most recent user that interacted with it and maintain a total user count, tracking the number of users who have ever interacted with it.
   - Provide functionality to access the total user count for each chatbot variant at any given time.

C. As specified in part A, when a chatbot variant throws a "Bot_Exception", an error message should notify the user that the query is invalid. Capture the username and query of the user causing the exception and write it to the "error_log.txt" file. The "error_log.txt" file should contain a list of usernames + queries for users who caused exceptions to be thrown. Also, you need to write a function “Analysis()” that will open the file “error_log.txt” in read mode and will perform the following analysis:
   - It will print the username who caused the maximum number of exceptions.
   - It will print the total count of words from each query stored.

Note: Ensure that your program demonstrates proper usage of object-oriented principles such as inheritance, encapsulation, exception handling, generics, and file handling. Implement appropriate methods and attributes in each class to fulfill the requirements outlined above.

BEST OF LUCK!