Date & Time: Sunday, October 29, 2023, 5:00-6:00 PM
The latest Date/Time to submit the answer: Wednesday, November 1, 2023, 5:00 PM
Total Points: 100 points

Contemporary C++: Learning Modern C++ in Modern way

Final exam

Almas-e Fanavari Abri-e Pasargad (Alpha co.)

Saeed Amrollahi Boyouki

Pre-historical and Historical

1. How do you relate these programming languages to each other: C, C++, and Simula? Explain your answer in detail. **(5 points)**

Basic concepts

2. Which statement is true and which one is false? In the wrong cases, explain why. **(10 points)**
   A. C++ is a *compiled* programming language.
   B. The following program is the *shortest* C++ program:

```
#include <iostream>
int main()
{
   return 0;
}
```

   C. In modern C++ (C++11 and beyond), to write a loop that prints out the values 4, 5, 9, 17, and 12 we must use typical or almost containers like **C-style array**, standard **array**, **vector**, **list,** or alike.
   D. C++ just supports two styles of programming: *object-oriented* (i.e., programming with inheritance) and *generic* programming.

E. We can define a pointer to an abstract class, but we can't define a reference to an abstract class.

Code Rejuvenation

3. Source code rejuvenation is a source-to-source transformation that replaces deprecated or rather old language features and idioms with modern code.

Concerning the above note, rejuvenate the following code snippets. In the 1st line of code, there is a comment which expresses the revision of the rejuvenated target code. **(12 points)**

```
// Rejuvenate to C++98
#define PI  3.14159
#define SQR(X) (X) * (X)

double sqr_of_pi()
{
   return SQR(PI);
}
```

1

```
// Rejuvenate from C++98 to C++11
enum DatabaseState { dbOpen, dbClosed };
enum DoorState { dOpen, dClosed };
enum NetworkConnectionState { netOpen, netClosed };
enum SocketState { sOpen, sClosed };

void enum_user()
{
   DatabaseState dbs =
         static_cast<DatabaseState>(netOpen);
   DoorState ds = dOpen;
   SocketState ss = sClosed;
   // …
}
```

2

```
// Rejuvenate to C++17
// Note: We don't use break statement
// intentionally!
using std::cout;
void print_asterisks(int i)
{
   switch (i) {
   case 1:
      cout << "*\n";
   case 2:
      cout << "**\n";
   case 3:
      cout << "***\n";
   case 4:
      cout << "****\n";
   default:
      cout << "*****\n";
  }
}
```

3

```
// Rejuvenate from C++98 to C++11
class B {
public:
  virtual ~B() =0;
  virtual void mem_fun() =0;
};
class D :public B {
public:
  void mem_fun()
  {
    // …
  }
};
void f(int i)
{
   B* p = new D{};
   // use p
   delete p;
}
```

4

Hint: In snippet code #4, we would like to replace **new/delete** operators with an RAII-based handle class.

General programming

4. Consider the following enumeration:
   enum Season { winter, spring, summer, fall };

Write the *prefix increment operator* (**++**) for Season. Prohibit the *postfix increment operator*. **(5 points)**

5. Compute the exponential function e^x using the following *Maclaurin Series*:

$$e^x = 1 + \frac{x}{1!} + \frac{x^2}{2!} + \frac{x^3}{3!} + \cdots, \qquad -\infty < x < \infty$$

Compute the first 10 terms of the series, i.e. (n = 9). Write the result with six significant digits (1 before the decimal point and 5 after). **(8 points)**

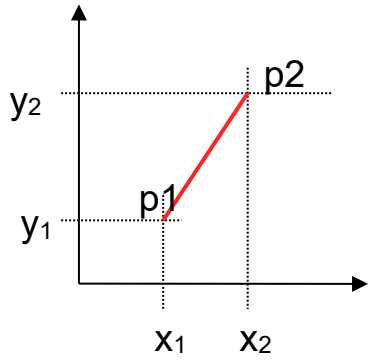
Abstract classes and Virtual functions

6. Consider the problem of representing one's personal assets. The intention here is to find the net worth of a person to plan for future needs and expenses. Think about how you have different assets in your portfolio: Automobiles, jewelry, real estate, securities (shares at Securities exchanges), and bank accounts. There are two kinds of securities: stocks and bonds and there are three kinds of bank accounts: savings, checking, and loan accounts.
   A. Design and implement a class hierarchy for these kinds of assets. Your hierarchy should be polymorphic. Try to implement the derived classes like jewelry or real estate.
   B. Among the -big- interface personal asset class provides; we are interested in net worth computation. declare and define a member function called **compute_net_worth** in all classes in the above class hierarchy.
   C. We want to compute the total worth of our assets. Write a function called **total_net_worth** which accepts a container of all assets and returns the total worth. **(12 points)**

Generic Algorithms

7. Read the coordinates of a two-dimensional point as the center of a circle, then read several -unknown number of – points coordinates and check if the points are at the perimeter of that circle. You have to define a class for **Point** and define an **Input operator** for reading them too. To compute the distance of two points in a 2D Cartesian plane, use the following formula:

$$d = \sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2}$$



Also, you have to use **for_each** to compute the distances and **all_of** generic algorithms to determine all points are at the perimeter of the circle. You can use an **equal** generic algorithm too. **(8 points)**

8. We are asked to rewrite the simplified version of the Unix **sort** utility in modern C++. The **sort** program reads lines from standard input, sorts them, and writes them, sorted, to the standard output stream. **(8 points)**

Hint: To read the lines of text from the standard input stream, use the **getline** function. Use the **sort** generic algorithm and use as many features of modern C++.

9. Compute the grocery Shopping total fee. You have the following typical shopping list:

Item	Quantity (kg)?
Apple	2 Kg
Pear	2 Kg
Tomato	3 Kg
Potato	3.250 Kg
Onion	3.750 Kg
Cucumber	1.250 Kg

There is a parallel list for items prices:

Item	Price Kg (IRR)?
Apple	250’000
Pear	450’000
Tomato	110’000
Potato	130’000
Onion	115’000
Cucumber	180’000

Represent the Shopping Item as a **pair** of Item names and how much to buy (quantity). Use generic algorithms **inner_product** and **accumulate**.

Don’t worry! Both lists are sorted according to items. **(8 points)**

10. Implement **iota_n**. **iota_n** is like iota, but fills just the first n elements of the container:

```
template <class ForwardIterator, class T>
void iota_n(ForwardIterator first, ForwardIterator last, T value, int N);
```

(5 points)

Object construction and destruction & RAII

11. A friend of mine who has used C++ for a few years, told me secretly:

“I don’t know the semantics and logic behind constructors and destructors, and I don’t like to use them! instead, I declare and define two hand-written member functions called *init* and *cleanup* for each class we want to acquire resources. In *init*, I construct a class object using the default constructor, then try to acquire

resources like memory allocation and in *cleanup,* the resources will be released. I call *init* after construction and call *cleanup* before destruction."

Do you support this claim? Argue this claim in detail from the point of C++ idiomatic design. **(6 points)**

Concurrency and parallel programming

12. Design and implementation of a very simple producer/consumer. For this, we have two **thread**s communicating by passing messages through a **queue**. The producer thread produces a couple of integers and at the consumer side, another thread computes the *greatest common divisor (GCD)* of them. You should establish mutual exclusion by **mutex**es and **condition variable**s. **(13 points)**

Hint: 1. You can use the code on *pages 244-245 of A Tour of C++* as a guide.
2. for GCD refer to the *https://en.wikipedia.org/wiki/Greatest_common_divisor#Euclidean_algorithm*