CSE100 Programming Principles using C++

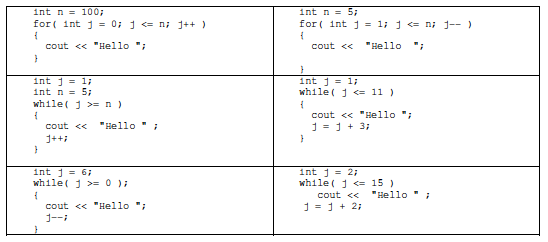
Extra Credit

(Will count for 20 points towards your final score)

1. Which of the following is/are true about the use of functions in programming? (2 points)
   1. Functions allow large problems to be broken into smaller tasks
   2. Functions allow the reuse of existing code
   3. Functions allow the programmer to write complex code that performs simple tasks
   4. Both A and B are true

**Answer: D**

1. How many times do the following loops execute? ( 0, unknown, infinite are perfectly legitimate answers.) (6 points)

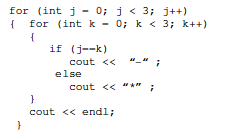


**1: 101 4: Infinite**

**2: 0 5: 4**

**3: 7 6: 7**

3 . Show the output of the following program. (3 points )



**-\*\***

**\*-\***

**\*\*-**

1. The following class declaration or programs contain error(s). Find as many as possible. (3 points)

Class Circle

{

private

double centerX;

double center;

double radius;

public

setCenter(double, double)

setRadius(double)

}

**There is no Constructor statements**

**There is no “:” after private or public**

**Class is capitalized it should be class**

**Both setCenter and setRadius need to define what type of returns they have or set to void.**

**Also the variables that are entered into the function need to be defined.**

**There needs to be a ; after setCenter and setRadius**

**There needs to be a ; after the closing class bracket**

**Re-Written Correctly:**

**class Circle**

**{**

**private:**

**double centerX;**

**double center;**

**double radius;**

**public:**

**Circle();**

**Circle(double, double, double); \\ Optional input constructor**

**void setCenter(double, double);**

**void setRadius(double);**

**};**

1. Draw a UML diagram for a class that has the following properties ( 6 points )
   1. The name of the class is Clock.
   2. The class Clock has the following private attributes: integer hours ( 0 through 24), integer minute (0 through 59) and second (0 through 59).
   3. The class Clock has the following public member functions.
      1. Clock(int, int, int) – constructor takes the hour, minute and seconds for object of type Clock.
      2. getHour – value-returning function to return the hour
      3. tick – adds 1 to seconds and updates the minute and hour as needed
      4. addSeconds(int) – adds a value between 1-60 to the existing value of seconds and adjusts minutes and hours accordingly.

|  |
| --- |
| **Clock** |
| **-hours: int (0-24)**  **-minute: int (0-59)**  **-second: int (0-59)** |
| **+Clock(int h, int m, int s)**  **+getHour()**  **+tick()**  **+addSeconds(int input) (1-60)** |

1. Define the class Clock (the declaration, not the implementation) as described in question 7, which has the three member variables and 2 member functions and the constructor. ( 5 points )

**#include<iostream>**

**using namespace std;**

**class Clock**

**{**

**private:**

**int hours;**

**int minute;**

**int seconds;**

**public:**

**Clock(int h, int m, int s)**

**int getHour();**

**void tick();**

**void addSeconds(int input)**

**};**

1. Write the implementation for the constructor for the Clock class that takes 3 arguments of type integer, and sets the member variables of the object of type Clock. (3 points)

**Clock::Clock(int h, int m, int s){**

**hours = h;**

**minutes = m;**

**seconds = s;**

**}**

1. Write the definition of the function getHour(). This is the implementation of the member function ( 5 points )

**int Clock::getHour(){**

**return hour;**

**}**

1. Define an instance of Clock named alarmClock with an initial time of 6:30. ( 2 points )

**Clock alarmClock(6,30,0);**

1. Write a function implementation for the function addSeconds. (10 points)
   1. Note: Your function should use :: to reference your class.
   2. Note: we are using the 24 hour system instead of AM/PM.
   3. Validate the value. It needs to be between 0 and 60.
   4. If the user enters a number outside the bounds, show an error prompt.
   5. Your code should be able to handle roll over. If say the existing values for hours = 6, minutes = 10, seconds = 20. If we add 50 seconds, the new values would be hours = 6, minutes = 11, seconds = 10. Also account for changes at midnight to next day.

**Honestly, I don’t think that you should have to validate to add seconds…..If they want to add 120 seconds let them. I designed the function to work with the internal tick() function of the Clock so I decided to include that function implementation as well. Since I built in the ability to roll over into the tick function, it would be smarter to implement the code in this way…..In my opinion.**

**void Clock::tick(){**

**if (seconds == 59){**

**if (minutes == 59){**

**if (hour == 24){**

**hour = 0;**

**seconds= 0;**

**}**

**else {**

**hour++;**

**seconds=0;**

**}**

**}**

**else{**

**minutes++;**

**seconds=0;**

**}**

**}**

**else {**

**seconds++;**

**}**

**}**

**void Clock::addSeconds(int input){**

**for (int i=1; i <= input; i++){**

**tick();**

**}**

1. Feedback on the course. (1 point)

**I will re-iterate what I said in the ASU submitted evaluation. This was one of the best programming classes I have taken at ASU so far. The recorded lectures were extremely helpful and appreciated. The instructor and teacher aides were VERY, VERY quick in evaluating my work and responding.**

**Thank you for your time,**

**Jason Pototsky**