**LAB 6: CLASSES AND OBJECTS MANIPULATION (ANSWERS & CODES)**

1. Program 6.1

**Output**:

1

2

3

Program 6.2

**Output**:

1

1

1

1. This program demonstrates the difference between static member variable and normal member variable.
   1. Program 6.3

**Output**:

t1's static i=1

and instance j=1

t2's static i=2

t2's instance j=1

t3's static i=3

t3's instance j=1

**Explanation**: getI() member function has access to static variable i. It will increment the variable i, then return i. getJ() member function has access to variable j. It will increment the variable j, then return j.

getI() is called three times, each time static variable i is incremented once. This is because t1, t2, t3 objects share one static variable i. getJ() is called three times, however, each time, each variable j of each object is incremented once. This is because t1, t2, t3 objects have their own variable j.

* 1. Program 6.4

**Output**:

The max between 5 and 2 is 5

**Explanation**: the max function called to determine the maximum between variable i and j is the default c++ function. It is not related to the static member function max() of class Test5. The static member function max() of class Test5 can be access by int Test5::max(i,j);.

1. #include <iostream>
2. **using** **namespace** std;
4. **class** Accumulator
5. {
6. **private**:
7. **int** m\_nValue;
8. **public**:
9. Accumulator() { m\_nValue = 0; }
10. **void** Add(**int** nValue) { m\_nValue += nValue; }
12. // Make the Reset() function a friend of this class
13. **friend** **void** Reset(Accumulator &cAccumulator);
15. **int** getValue() { **return** m\_nValue; }
16. };
18. **void** Reset(Accumulator &cAccumulator)
19. {
20. cAccumulator.m\_nValue = 0;
21. }
23. **int** main()
24. {
25. // object of class Accumulator is created
26. // default constructor is called, and m\_nValue is set to 0
27. Accumulator testAccumulator;
29. // add 200 to the m\_nValue of object
30. testAccumulator.Add(200);
32. // print the m\_nValue after adding and before reset
33. cout << testAccumulator.getValue() << endl;
35. // reset the m\_nValue of object
36. Reset(testAccumulator);
38. // print the m\_nValue after reset
39. cout << testAccumulator.getValue() << endl;
41. **return** 0;
42. }

**Output:**

200

0

**Explanation:**

The Reset() standalone function is the friend of Accumulator class, so it can access the private member variable m\_nValue of Accumulator class. It sets the m\_nValue back to 0.

1. friend allows a class to access private member functions of another class. This is done by putting friend keyword in front of the class Display as in line 15.
2. No question.
3. The corrected is as follows:
   1. #include <iostream>
   2. **using** **namespace** std;
   4. **class** Number
   5. {
   6. **private**:
   7. **int** m\_num;
   8. **public**:
   9. Number();
   10. Number(**int** n);
   11. **int** GetNumber();
   12. **void** SetNumber(**int** n);
   13. Number operator++();
   14. };
   16. Number::Number()
   17. {
   18. }
   20. Number::Number(**int** n)
   21. {
   22. m\_num = n;
   23. }
   25. **int** Number::GetNumber()
   26. {
   27. **return** m\_num;
   28. }
   30. **void** Number::SetNumber(**int** n)
   31. {
   32. m\_num = n;
   33. }
   35. Number Number::operator++ ()
   36. {
   37. m\_num += 1;
   38. **return** \***this**;
   39. }
   41. **int** main()
   42. {
   43. Number num1(5);
   44. ++num1;
   45. cout << num1.GetNumber() << endl;
   46. **return** 0;
   47. }

**Explanation:**

Line 44 used a prefix increment operator. It is overloaded at line 35. A declaration is added at line 13. Prefix overloading does not require any argument. The value is incremented first, then only current object is returned.

**EXERCISE 2: STRUCTURED PROBLEMS**

1. Line 15 sets the static variable, z of class Thing to 4. Static variable of class does not need any objects to be created to be set.
2. **Program 6.10**

**Output:**

Value before function tukarGanti() 100 50

Value after function tukarGanti() 50 100

**Explanation:**

The function tukarGanti is the friend of class kawan1 and kawan2, so it can access the private member variable of class kawan1 and kawan2. tukarGanti() swaps the value of object of class kawan1 and object of class kawan2.

**Program 6.11**

**Code:**

1. // write the functions here
2. // member function, gossip of kawan1 will take an object of type kawan2 as input
3. // it access the private member variable value and girlF of kawan2 object
4. **void** kawan1::gossip(kawan2 examplekawan2)
5. {
6. cout << value << " gossips on " << examplekawan2.value << endl;
7. cout << value << " says " << examplekawan2.value << " girl friend is " << examplekawan2.girlF << endl;
8. }
9. // member function, gossip of kawan2 will take an object of type kawan1 as input
10. // it access the private member variable value and age of kawan1 object
11. **void** kawan2::gossip(kawan1 examplekawan1)
12. {
13. cout << value << " gossips on " << examplekawan1.value << endl;
14. cout << value << " says " << examplekawan1.value << " age is " << examplekawan1.age << endl;
15. This question is the same as assignment 4, so I copy my own code here:
16. #include <iostream>
17. **using** **namespace** std;
19. **class** Rectangle
20. {
21. **int** width, height;
22. **public**:
23. // default constructor does not accept any input
24. Rectangle();
25. // constructor
26. Rectangle(**int**, **int**);
27. **int** calculateArea();
28. **friend** Rectangle duplicate(Rectangle);
29. };
31. // returns the area value of a Rectangle object
32. **int** Rectangle::calculateArea()
33. {
34. **int** area;
35. area = width \* height;
36. **return** area;
37. }
39. // This is a standalone function that has access to class Rectangle
40. // It will accepts a Rectangle object and multiply by 2 both the width and height of the object
41. // Then it will return the object
42. Rectangle duplicate(Rectangle exampleRectangle)
43. {
44. exampleRectangle.width \*= 2;
45. exampleRectangle.height \*= 2;
46. **return** exampleRectangle;
47. }
49. // Default constructor sets width and height to 0
50. Rectangle::Rectangle()
51. {
52. width = 0;
53. height = 0;
54. }
56. // constructor that accepts integers
57. // sets the width and height
58. Rectangle::Rectangle(**int** a, **int** b)
59. {
60. width = a;
61. height = b;
62. }
64. **int** main()
65. {
66. // initialize the two objects of class Rectangle
67. // call default constructor when rectb is created
68. Rectangle rect(10,30), rectb;
69. // double the width and length of rect
70. // pass the object returned to rectb
71. rectb = duplicate(rect);
72. // calculate the arae of rectb
73. cout << rectb.calculateArea();
75. **return** 0;
76. }

**Output:**

1200

1. No question.
2. A copy constructor and a member function, multiply() is added

**Code:**

1. #include <iostream>
2. **using** **namespace** std;
4. **class** FeetInches {
5. **private**:
6. **int** feet;
7. **int** inches;
8. **public**:
9. FeetInches(**int** f=0, **int** i=0){
10. feet = f;
11. inches = i;
12. }
14. // copy constructor
15. FeetInches(**const** FeetInches &objToBeCopied)
16. {
17. feet = objToBeCopied.feet;
18. inches = objToBeCopied.feet;
19. }
21. **void** setFeet(**int** f)
22. { feet = f; }
24. **void** setInches(**int** i)
25. { inches = i; }
27. **int** getFeet() **const**
28. { **return** feet; }
30. **int** getInches() **const**
31. { **return** inches; }
33. // multiply the calling object's feet by the argument's object's feet
34. // same for inches
35. FeetInches multiply(**const** FeetInches &obj)
36. {
37. FeetInches tempObj;
38. tempObj.feet = feet \* obj.feet;
39. tempObj.inches = inches \* obj.inches;
40. **return** tempObj;
41. }
42. };
44. **int** main(){
46. FeetInches one(20,50);
47. FeetInches two(one);
48. FeetInches three;
50. three = one.multiply(two);
52. cout << one.getFeet() << endl
53. << two.getFeet() << endl
54. << three.getFeet() << endl;
56. }

**Output:**

20

20

400

**EXERCISE 3: PROBLEM SOLVING**

1. No question.