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

1. Program 6.1
   Output:
   1
   2
   3

   Program 6.2
   Output:
   1
   1
   1

2. This program demonstrates the difference between static member variable and normal member variable.
   i. 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.

   ii. 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);.

3.

```
1.  #include <iostream>
2.  using namespace std;
3.
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; }
11.
12.         // Make the Reset() function a friend of this class
13.         friend void Reset(Accumulator &cAccumulator);
14.
15.         int getValue() { return m_nValue; }
16. };
17.
18. void Reset(Accumulator &cAccumulator)
19. {
20.     cAccumulator.m_nValue = 0;
21. }
22.
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;
28.
29.     // add 200 to the m_nValue of object
30.     testAccumulator.Add(200);
31.
32.     // print the m_nValue after adding and before reset
33.     cout << testAccumulator.getValue() << endl;
34.
35.     // reset the m_nValue of object
36.     Reset(testAccumulator);
37.
38.     // print the m_nValue after reset
39.     cout << testAccumulator.getValue() << endl;
40.
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.

4. 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.

5. No question.
6. The corrected is as follows:

```
1.  #include <iostream>
2.  using namespace std;
3.
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. };
15.
16. Number::Number()
17. {
18. }
19.
20. Number::Number(int n)
21. {
22.     m_num = n;
23. }
24.
25. int Number::GetNumber()
26. {
27.     return m_num;
28. }
29.
30. void Number::SetNumber(int n)
31. {
32.     m_num = n;
33. }
34.
35. Number Number::operator++ ()
36. {
37.     m_num += 1;
38.     return *this;
39. }
40.
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 " << examp
    lekawan2.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;
```

3. This question is the same as assignment 4, so I copy my own code here:

```
1.  #include <iostream>
2.  using namespace std;
3.
4.  class Rectangle
5.  {
6.      int width, height;
7.      public:
8.          // default constructor does not accept any input
9.          Rectangle();
10.         // constructor
11.         Rectangle(int, int);
12.         int calculateArea();
13.         friend Rectangle duplicate(Rectangle);
14. };
15.
16. // returns the area value of a Rectangle object
```

```
int Rectangle::calculateArea()
{
    int area;
    area = width * height;
    return area;
}

// This is a standalone function that has access to class Rectangle
// It will accepts a Rectangle object and multiply by 2 both the width and height of the object
// Then it will return the object
Rectangle duplicate(Rectangle exampleRectangle)
{
    exampleRectangle.width *= 2;
    exampleRectangle.height *= 2;
    return exampleRectangle;
}

// Default constructor sets width and height to 0
Rectangle::Rectangle()
{
    width = 0;
    height = 0;
}

// constructor that accepts integers
// sets the width and height
Rectangle::Rectangle(int a, int b)
{
    width = a;
    height = b;
}

int main()
{
    // initialize the two objects of class Rectangle
    // call default constructor when rectb is created
    Rectangle rect(10,30), rectb;
    // double the width and length of rect
    // pass the object returned to rectb
    rectb = duplicate(rect);
    // calculate the arae of rectb
    cout << rectb.calculateArea();

    return 0;
}
```

Output:

1200

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

Code:

```
#include <iostream>
using namespace std;

class FeetInches {
    private:
       int feet;
```

```
        int inches;
    public:
        FeetInches(int f=0, int i=0){
            feet = f;
            inches = i;
        }

        // copy constructor
        FeetInches(const FeetInches &objToBeCopied)
        {
            feet = objToBeCopied.feet;
            inches = objToBeCopied.feet;
        }

        void setFeet(int f)
        { feet = f; }

        void setInches(int i)
        { inches = i; }

        int getFeet() const
        { return feet; }

        int getInches() const
        { return inches; }

        // multiply the calling object's feet by the argument's object's feet
        // same for inches
        FeetInches multiply(const FeetInches &obj)
        {
            FeetInches tempObj;
            tempObj.feet = feet * obj.feet;
            tempObj.inches = inches * obj.inches;
            return tempObj;
        }
};

int main(){

   FeetInches one(20,50);
   FeetInches two(one);
   FeetInches three;

   three = one.multiply(two);

   cout << one.getFeet() << endl
    << two.getFeet() << endl
    << three.getFeet() << endl;

}
```

Output:
20
20
400

EXERCISE 3: PROBLEM SOLVING

1. No question.