Module 1. C++ Classes Exercises

1. The ZooAnimal class definition below is missing a prototype for the Create function. It should have parameters so that a character string and three integer values (in that order) can be provided when it is called for a ZooAnimal object. Like the Destroy function, it should have return type void. Write an appropriate prototype for the ZooAnimal Create function.

```
class ZooAnimal
{
 private:
   char *name;
   int cageNumber;
   int weightDate;
   int weight;
 public:
   void Destroy (); // destroy function
   char* reptName ();
   int daysSinceLastWeighed (int today);
};
```

2. Write a function header for the ZooAnimal class member function daysSinceLastWeighed. This function has a single integer parameter today and returns an integer number of days since the animal was last weighed.

```
void ZooAnimal::Destroy ()
{
 delete [] name;
}

// -------- member function to return the animal's name
char* ZooAnimal::reptName ()
{
 return name;
}

// -------- member function to return the number of days
// -------- since the animal was last weighed

{
 int startday, thisday;
 thisday = today/100*30 + today - today/100*100;
 startday = weightDate/100*30 + weightDate - weightDate/100*100;
 if (thisday < startday)
     thisday += 360;
 return (thisday-startday);
}
```

3. In the main function there is a cout statement that attempts to print the animal's name. However, it is not allowable because it attempts to access the private data member called name. Modify that statement so that it uses a public member function that returns the ZooAnimal's name.

```
class ZooAnimal
{
```

```
 private:
   char *name;
   int cageNumber;
   int weightDate;
   int weight;
 public:
   void Create (char*, int, int, int); // create function
   void Destroy (); // destroy function
   char* reptName ();
   int daysSinceLastWeighed (int today);
};

// -------- member function to return the animal's name
char* ZooAnimal::reptName ()
{
 return name;
}

// ========== an application to use the ZooAnimal class
void main ()
{
 ZooAnimal bozo;
 bozo.Create ("Bozo", 408, 1027, 400);

 cout << "This animal's name is " << bozo.name << endl;

 bozo.Destroy ();
}
```

1. Write the ZooAnimal inline member function reptWeightDate. It should simply return the weightDate data member.

```
enum scale {ounces, kilograms};

class ZooAnimal
{
 private:
   char *name;
   int cageNumber;
   int weightDate;
   int weight;
 public:
   void Create (char*, int, int, int); // create function
   void Destroy (); // destroy function
   void changeWeight (int pounds);
   void changeWeightDate (int today);
   char* reptName ();
   int reptWeight ();
   void reptWeight (scale which);
   inline int reptWeightDate ();
   int daysSinceLastWeighed (int today);
};
```

2. Modify the prototype for changeWeightDate below to make it the appropriate single line inline member function changeWeightDate. The single line needed should set the data member weightDate equal to the parameter today.

```
enum scale {ounces, kilograms};
```

```
class ZooAnimal
{
 private:
   char *name;
   int cageNumber;
   int weightDate;
   int weight;
 public:
   void Create (char*, int, int, int); // create function
   void Destroy (); // destroy function
   void changeWeight (int pounds);
   inline void changeWeightDate (int today);
   char* reptName ();
   int reptWeight ();
   void reptWeight (scale which);
   int reptWeightDate ();
   int daysSinceLastWeighed (int today);
};
```

3. Write the ZooAnimal member function isMotherOf. It needs only a single statement that makes the mother data member of the ZooAnimal object parameter a pointer to the ZooAnimal object for which this function is called.

```
class ZooAnimal
{
 private:
   char *name;
   int cageNumber;
   int weightDate;
   int weight;
   ZooAnimal *mother;
 public:
   void Create (char*, int, int, int); // create function
   void Destroy (); // destroy function
   void isMotherOf (ZooAnimal&);
   void changeWeight (int pounds);
   void changeWeightDate (int today);
   char* reptName ();
   int reptWeight ();
   void reptWeight (scale which);
   inline int reptWeightDate () {return weightDate;};
   int daysSinceLastWeighed (int today);
};
```

Module 2. Class Member Functions Exercises

1. Write the ZooAnimal inline member function reptWeightDate. It should simply return the weightDate data member.

```
enum scale {ounces, kilograms};

class ZooAnimal
{
 private:
   char *name;
   int cageNumber;
   int weightDate;
   int weight;
 public:
   void Create (char*, int, int, int); // create function
   void Destroy (); // destroy function
   void changeWeight (int pounds);
   void changeWeightDate (int today);
   char* reptName ();
   int reptWeight ();
   void reptWeight (scale which);
   inline int reptWeightDate ();
   int daysSinceLastWeighed (int today);
};
```

2. Modify the prototype for changeWeightDate below to make it the appropriate single line inline member function changeWeightDate. The single line needed should set the data member weightDate equal to the parameter today.

```
enum scale {ounces, kilograms};

class ZooAnimal
{
 private:
   char *name;
   int cageNumber;
   int weightDate;
   int weight;
 public:
   void Create (char*, int, int, int); // create function
   void Destroy (); // destroy function
   void changeWeight (int pounds);
   inline void changeWeightDate (int today);
   char* reptName ();
   int reptWeight ();
   void reptWeight (scale which);
   int reptWeightDate ();
   int daysSinceLastWeighed (int today);
};
```

3. Write the ZooAnimal member function isMotherOf. It needs only a single statement that makes the mother data member of the ZooAnimal object parameter a pointer to the ZooAnimal object for which this

function is called.

```
class ZooAnimal
{
 private:
   char *name;
   int cageNumber;
   int weightDate;
   int weight;
   ZooAnimal *mother;
 public:
   void Create (char*, int, int, int); // create function
   void Destroy (); // destroy function
   void isMotherOf (ZooAnimal&);
   void changeWeight (int pounds);
   void changeWeightDate (int today);
   char* reptName ();
   int reptWeight ();
   void reptWeight (scale which);
   inline int reptWeightDate () {return weightDate;};
   int daysSinceLastWeighed (int today);
};
```

Module 3. Constructors and Destructors Exercises

1. Create the ZooAnimal constructor function. The function has 4 parameters -- a character string followed by three integer parameters. In the constructor function dynamically allocate the name field (20 characters), copy the character string parameter into the name field, and then assign the three integer parameters to cageNumber, weightDate, and weight respectively.

```
class ZooAnimal
{
 private:
   char *name;
   int cageNumber;
   int weightDate;
   int weight;
 public:
   ZooAnimal (char*, int, int, int); // constructor function
   ~ZooAnimal (); // destructor function
   void changeWeight (int pounds);
   char* reptName ();
   int reptWeight ();
   int daysSinceLastWeighed (int today);
};
```

2. Modify the function header below for the constructor function for ZooAnimal. Provide default values of "Nameless" for who, 9999 for whichCage, 101 (January 1) for weighDay, and 100 for pounds.

```
// ---------- the constructor function
ZooAnimal::ZooAnimal (char* who, int whichCage, int weighDay, int
pounds)
{
 char *name = new char[20];
 strcpy (name, who);
 cageNumber = whichCage;
 weightDate = weighDay;
 weight = pounds;
}
```

3. Write a ZooAnimal member conversion function that converts a type ZooAnimal to produce a value of type int. The function should simply return the cageNumber data member.

```
class ZooAnimal
{
 private:
   char *name;
   int cageNumber;
   int weightDate;
   int weight;
 public:
   ZooAnimal (char*, int, int, int); // constructor function
   operator int (); // member conversion function
   inline ~ZooAnimal () {}; // destructor function
```

```
   void changeWeight (int pounds);
   char* reptName ();
   int reptWeight ();
   int daysSinceLastWeighed (int today);
};
```

Module 4. Static Members, Friends Exercises

1. Add the following below: Declare the integer static data member oldestWeightDate. Create the static member function changeOldestWeightDate with void return type and a single integer parameter. Set oldestWeightDate to that parameter. Do not forget a function prototype for changeOldestWeightDate. Also, be sure to define the static member at global scope.

```
class ZooAnimal
{
 private:
   char *name;
   int cageNumber;
   int weightDate;
   int weight;
 public:
   ZooAnimal (char*, int, int, int); // constructor function
   void changeWeight (int pounds);
   char* reptName ();
   int reptWeight ();
   int daysSinceLastWeighed (int today);
};
```

2. Write a non-member function reptOldestZooAnimalWeightDate that returns the value of the ZooAnimal class static data member oldestWeightDate.

```
class ZooAnimal
{
 private:
   char *name;
   int cageNumber;
 public:
   static int oldestWeightDate;
 private:
   int weightDate;
   int weight;
 public:
   ZooAnimal (char*, int, int, int); // constructor function
   void changeWeight (int pounds);
   static changeOldestWeightDate (int date);
   char* reptName ();
   int reptWeight ();
   static int reptOldestWeightDate ();
   int daysSinceLastWeighed (int today);
};

int ZooAnimal::oldestWeightDate;  //define static member at global scope
```

3. Write the ZooAnimal member function reptNutrition that returns the result of the member function reptMealNutrition from the nested class nutrition invoked on the ZooAnimal class data member nutr.

```
class ZooAnimal
```

```
{
 public:
   class nutrition
   {
    private:
      int numberMeals;
      int maxPoundsFed;
      float avgNutrValue;
    public:
      inline nutrition () {
        numberMeals = 4;
        maxPoundsFed = 75;
        avgNutrValue = 38.21;}; // constructor function
      inline float reptMealNutrition ()
       {return numberMeals * avgNutrValue;};
    };

 private:
   nutrition nutr;
 public:
   ZooAnimal (char*, int, int, int); // constructor function
   float reptNutrition ();
};
```

4. Declare that the class cageAssignment is a friend class to ZooAnimal.

```
class ZooAnimal
{
 private:
   char *name;
   int cageNumber;
   int weightDate;
   int weight;
 public:
   ZooAnimal (char*, int, int, int); // constructor function
   ~ZooAnimal (); // destructor function
   void changeWeight (int pounds);
   char* reptName ();
   int reptWeight ();
   int daysSinceLastWeighed (int today);
};
```

5. Write the changeCage member function of class cageAssignment. It should set the data member cageNumber of the ZooAnimal object parameter to the value of the cageAssignment data member enclosure and then increment enclosure.

```
class ZooAnimal;

class cageAssignment
{
 private:
   int enclosure;
 public:
   inline cageAssignment (int num) {enclosure = num;}; // constructor
   inline ~cageAssignment () {}; // destructor function
   void changeCage (ZooAnimal&);
   int reptCage (ZooAnimal&);
```

```
};

class ZooAnimal  { int cageNumber; };
```

6. Write a function prototype for the friend function declaration which is the cageAssignment class member function reptCage. This function has a single ZooAnimal parameter (called by reference) and returns an integer value.

```
class ZooAnimal;

class cageAssignment
{
   int reptCage (ZooAnimal&);
};

class ZooAnimal
{
 private:
   char *name;
   int cageNumber;
   int weightDate;
   int weight;
 public:
   ZooAnimal (char*, int, int, int); // constructor function
   ~ZooAnimal (); // destructor function
   void changeWeight (int pounds);
   char* reptName ();
   int reptWeight ();
   int daysSinceLastWeighed (int today);
};
```

Module 5. Linked Lists, Stacks, Queues Exercises

1. Complete the ZooAnimal constructor function. It should insert the new ZooAnimal class object in the linked list at the end after the object currently pointed to by the static data member current. Make the nextAnml pointer of the current object point to the object being constructed. Be sure to update the pointer current.

```
class ZooAnimal
{
 private:
   char *name;
   ZooAnimal* prevAnml;                // previous link in the list
   ZooAnimal* nextAnml;                // next link in the list
 public:
   static ZooAnimal *start;            // first element of linked list
   static ZooAnimal *current;          // last element of linked list
 public:
   ZooAnimal (char*);                  // constructor function
   ~ZooAnimal ();                      // destructor function
   char* reptName () { return name; }
   ZooAnimal* prevAnimal ();
};

ZooAnimal* ZooAnimal::start = NULL;
ZooAnimal* ZooAnimal::current = NULL;

ZooAnimal::ZooAnimal (char* theName)
{
   char *name = new char[20];
   strcpy(name, theName);
}
```

2. The destructor function is called to destroy the last item in the linked list. The pointer to the object being destroyed is the this pointer. The destructor function updates the static data member current. Write the destructor function. Make current the same as the prevAnml pointer of the object being destroyed.

```
class ZooAnimal
{
 private:
   char *name;
   ZooAnimal* prevAnml;                // previous link in the list
   ZooAnimal* nextAnml;                // next link in the list
 public:
   static ZooAnimal *start;            // first element of linked list
   static ZooAnimal *current;          // last element of linked list
 public:
   ZooAnimal (char*);                  // constructor function
   ~ZooAnimal ();                      // destructor function
   char* reptName () { return name; }
   ZooAnimal* prevAnimal ();
};

ZooAnimal* ZooAnimal::start = NULL;
```

```
ZooAnimal* ZooAnimal::current = NULL;
```

3. Modify the cout statement below so that it invokes the ZooAnimal member function reptName to report the name of the creature object denoted by subscript whichOne.

```
class ZooAnimal
{
 private:
   char *name;
   int cageNumber;
   int weightDate;
   int weight;
 public:
   ZooAnimal (char*, int, int, int); // constructor function
   ~ZooAnimal (); // destructor function
   void changeWeight (int pounds);
   char* reptName ();
   int reptWeight ();
   int daysSinceLastWeighed (int today);
};

// ========== an application to use the ZooAnimal class
void main ()
{
 ZooAnimal creature [200];
 int whichOne = 173;

 cout << "This animal's name is " << _____ << endl;
}
```

Module 6. Inheritance Exercises

1. Make the necessary changes below to indicate that the class LargeAnimal is derived from the base class ZooAnimal. Specify that the base class is public.

```
class ZooAnimal
{
 private:
   char* name;
   int cageNumber;
   int weightDate;
   int weight;
 public:
   ZooAnimal (char*, int, int, int); // constructor function
   inline ~ZooAnimal () { delete [] name; }; // destructor function
   void changeWeight (int pounds);
   char* reptName ();
   int reptWeight ();
   int daysSinceLastWeighed (int today);
};

class LargeAnimal
{
 private:
   char* species;
   float cageMinimumVolume;
 public:
   LargeAnimal (char*, int, int, int, float); // constructor function
   inline ~LargeAnimal () { delete [] species; }; // destructor function
   float reptCageMinimumVolume ();
};
```

2. On the cout statement for gonzo (last statement below), since the gonzo object is of type class LargeAnimal, gonzo's species will be returned by the LargeAnimal class' reptName member function. Make the necessary changes so that gonzo's name will be returned instead by the ZooAnimal class' reptName member function.

```
class ZooAnimal
{
 private:
   char* name;
   int cageNumber;
   int weightDate;
   int weight;
 public:
   ZooAnimal (char*, int, int, int); // constructor function
   inline ~ZooAnimal () { delete [] name; }; // destructor function
   void changeWeight (int pounds);
   char* reptName ();
   int reptWeight ();
   int daysSinceLastWeighed (int today);
};

// -------- member function to return the animal's name
char* ZooAnimal::reptName ()
{
 return name;
```

```
}

class LargeAnimal : public ZooAnimal
{
 private:
   char* species;
   float cageMinimumVolume;
 public:
   LargeAnimal (char*, int, int, int, char*, float); // constructor
function
   inline ~LargeAnimal () { delete [] species; }; // destructor function
   float reptCageMinimumVolume ();
   char* reptName ();
};

// -------- member function to return the large animal's species
char* LargeAnimal::reptName ()
{
 return species;
}

// ========== an application to use the LargeAnimal class
void main ()
{
 ZooAnimal bozo;
 LargeAnimal gonzo;
 ...
 cout << bozo.reptName () << endl;
 cout << gonzo.reptName () << endl;
}
```

3. Make the necessary changes below so that all calls to the ZooAnimal member function reptName are passed to the matching function in the derived type when called for an object of the derived type.

```
class ZooAnimal
{
 private:
   char* name;
   int cageNumber;
   int weightDate;
   int weight;
 public:
   ZooAnimal (char*, int, int, int); // constructor function
   inline ~ZooAnimal () { delete [] name; }; // destructor function
   void changeWeight (int pounds);
   char* reptName ();
   int reptWeight ();
   int daysSinceLastWeighed (int today);
};

// -------- member function to return the animal's name
char* ZooAnimal::reptName ()
{
 return name;
}

class LargeAnimal : public ZooAnimal
{
 private:
```

```
      char* species;
      float cageMinimumVolume;
    public:
      LargeAnimal (char*, int, int, int, char*, float); // constructor
function
      inline ~LargeAnimal () { delete [] species; }; // destructor function
      float reptCageMinimumVolume ();
      char* reptName ();
   };

   // -------- member function to return the large animal's species
   char* LargeAnimal::reptName ()
   {
    return species;
   }
```

4. Make the necessary changes to indicate that the class LargeAnimal is a derived class of the base classes ZooAnimal and Mammal (in that order). Specify that the base classes are public.

```
   class ZooAnimal
   {
    protected:
      char* name;
      int cageNumber;
      int weightDate;
      int weight;
    public:
      ZooAnimal (char*, int, int, int); // constructor function
      inline ~ZooAnimal () { delete [] name; }; // destructor function
      void changeWeight (int pounds);
      char* reptName ();
      int reptWeight ();
      int daysSinceLastWeighed (int today);
   };

   class Mammal
   {
    protected:
      float minimumVolume;
      int minimumWeight;
    public:
      Mammal (float, int); // constructor function
      inline ~Mammal () {}; // destructor function
      float reptminimumVolume ();
      int reptminimumWeight ();
   };

   class LargeAnimal
   {
    protected:
      char* species;
      float cageMinimumVolume;
    public:
      LargeAnimal (char*, int, int, int, float, float, int); // constructor
      inline ~LargeAnimal () { delete [] species; }; // destructor function
      float reptCageMinimumVolume ();
   };
```

5. In the reptCageMinimumVolume function below, the reference to the data name weight is ambiguous. Make the necessary changes to indicate that the weight to be used is the one from the Mammal base class.

```
class ZooAnimal
{
 protected:
   char* name;
   int cageNumber;
   int weightDate;
   int weight;
 public:
   ZooAnimal (char*, int, int, int); // constructor function
   inline ~ZooAnimal () { delete [] name; }; // destructor function
   void changeWeight (int pounds);
   char* reptName ();
   int reptWeight ();
   int daysSinceLastWeighed (int today);
};

class Mammal
{
 protected:
   float minimumVolume;
   int weight;
 public:
   Mammal (float, int); // constructor function
   inline ~Mammal () {}; // destructor function
   float reptminimumVolume ();
   int reptWeight ();
};

class LargeAnimal : public ZooAnimal, public Mammal
{
 protected:
   char* species;
   float cageMinimumVolume;
 public:
   LargeAnimal (char*, int, int, int, float, float, int); // constructor
   inline ~LargeAnimal () { delete [] species; }; // destructor function
   float reptCageMinimumVolume ();
};

// -------- member function to return the minimum cage volume
// -------- needed for this large animal
float LargeAnimal::reptCageMinimumVolume ()
{
 if (weight < 500)
    return cageMinimumVolume;
 else
    return reptminimumVolume ();
}
```

6. In the reptCageMinimumVolume function below, the reference to the function name reptWeight is ambiguous. Make the necessary changes to indicate that the reptWeight function to be used is the one from the ZooAnimal base class.

```
class ZooAnimal
{
```

```
 protected:
   char* name;
   int cageNumber;
   int weightDate;
   int weight;
 public:
   ZooAnimal (char*, int, int, int); // constructor function
   inline ~ZooAnimal () { delete [] name; }; // destructor function
   void changeWeight (int pounds);
   char* reptName ();
   int reptWeight ();
   int daysSinceLastWeighed (int today);
};

class Mammal
{
 protected:
   float minimumVolume;
   int weight;
 public:
   Mammal (float, int); // constructor function
   inline ~Mammal () {}; // destructor function
   float reptminimumVolume ();
   int reptWeight ();
};

class LargeAnimal : public ZooAnimal, public Mammal
{
 protected:
   char* species;
   float cageMinimumVolume;
 public:
   LargeAnimal (char*, int, int, int, float, float, int); // constructor
   inline ~LargeAnimal () { delete [] species; }; // destructor function
   float reptCageMinimumVolume ();
};

// -------- member function to return the minimum cage volume
// -------- needed for this large animal
float LargeAnimal::reptCageMinimumVolume ()
{
 if (Mammal::weight < 500)
    return cageMinimumVolume;
 else
    return reptWeight ();
}
```

Module 7. Overloading Operators Exercises

1. Write the member function to overload the == operator. Use the standard == operator to compare the weightDate data member of the ZooAnimal object to the integer parameter.

```
class ZooAnimal
{
 private:
   char *name;
   int cageNumber;
   int weightDate;
   int weight;
 public:
   ...
   int operator== (int);
   ...
};
```

2. Write the member function to overload the auto-decrement operator. Use the standard -= operator to decrease the ZooAnimal object's weight by 1 (pound).

```
class ZooAnimal
{
 private:
   char *name;
   int cageNumber;
   int weightDate;
   int weight;
 public:
   ...
   void operator-- ();
   ...
};
```

3. Write the member function to overload the function call operator. Give the parameter the value of the data member cageNumber.

```
class ZooAnimal
{
 private:
   char *name;
   int cageNumber;
   int weightDate;
   int weight;
 public:
   ...
   void operator() (int&);
   ...
};

// ========== an application to use the ZooAnimal class
void main ()
{
 ZooAnimal bozo ("Bozo", 408, 1027, 400);
 int cage;
```

```
 bozo(cage);
 cout << cage << endl;
}
```

Module 8. Class Templates Exercises

1. The class ZooAnimal is going to become a Class Template. Declare a class template with a single template type parameter weightType. Declare the data name weight to be of whatever type is used for weightType. Make the last parameter to the constructor function to be of weightType. The constructor function is not in the body of class template ZooAnimal. Modify it so that it shows that this is the constructor function for the class template ZooAnimal with template type parameter weightType.

```
class ZooAnimal
{
 private:
   char *name;
   int cageNumber;
   int weightDate;
   int weight;
 public:
   ZooAnimal (char*, int, int, int); // constructor function
   inline ~ZooAnimal () { delete [] name; }; // destructor function
   void changeWeight (int pounds);
   char* reptName ();
   int reptWeight ();
   int daysSinceLastWeighed (int today);
};

// ---------- the constructor function
  ZooAnimal::ZooAnimal (char* who, int whichCage, int weighDay, int
pounds)
{
 char *name = new char[20];
 strcpy (name, who);
 cageNumber = whichCage;
 weightDate = weighDay;
 weight = pounds;
}
```

2. The class template ZooAnimal has been modified to include another template type parameter. Write a statement that declares a class template with two template type parameters: first weightType and second cageType.

```
template <class weightType, class cageType>
class ZooAnimal
{
 private:
   char *name;
   cageType cageNumber;
   int weightDate;
   weightType weight;
 public:
   ZooAnimal (char*, cageType, int, weightType); // constructor function
   inline ~ZooAnimal () { delete [] name; }; // destructor function
   void changeWeight (weightType pounds);
   char* reptName ();
   weightType reptWeight ();
   int daysSinceLastWeighed (int today);
};
```

3. The class template ZooAnimal once again has only the single template type parameter weightType. Modify the declaration of object bozo so that it is the declaration of a ZooAnimal object with template type parameter weightType to be of type integer.

```
template <class weightType>
class ZooAnimal
{
 private:
   char *name;
   int cageNumber;
   int weightDate;
   weightType weight;
 public:
   ZooAnimal (char*, int, int, weightType); // constructor function
   inline ~ZooAnimal () { delete [] name; }; // destructor function
   void changeWeight (weightType pounds);
   char* reptName ();
   weightType reptWeight ();
   int daysSinceLastWeighed (int today);
};

// ---------- the constructor function
template <class weightType>
ZooAnimal<weightType>::ZooAnimal (char* who, int whichCage, int
weighDay,
                          weightType pounds)
{
 char *name = new char[20];
 strcpy (name, who);
 cageNumber = whichCage;
 weightDate = weighDay;
 weight = pounds;
}

// ========== an application to use the ZooAnimal class
void main ()
{
 ZooAnimal bozo ("Bozo", 408, 1027, 400);
}
```

Module 9. I/O Stream Libraries Exercises

1. The cout statement below outputs a spectrum value and a team character string. Modify this line using the setw function so that each spectrum value is output in a field 10 characters wide and each team string is output in a field 8 characters wide.

```
#include <iostream.h>

void main()
{
 float spectrum[] = { 6.24, 3.87, 5.06, 2.19, 4.38 };
 char *team[] = {"Bozo", "Gonzo", "Rollo", "Marco"};
 for (int level = 0; level < 4; level++)
   {
    cout << spectrum[level] << team[level] << endl;
   }
}
```

2. Modify the cout statement below to specify that 2 digits of precision are to be used when printing spectrum's floating point values.

```
#include <iostream.h>

void main()
{
 float spectrum[] = { 6.24, 3.87, 5.06, 2.19, 4.38 };
 for (int level = 0; level < 5; level++)
   {
    cout << spectrum[level] << endl;
   }
}
```

3. Write the function that defines the overloaded >> operator for ZooAnimal objects to input values for the name, cage number, weight date, and weight (respectively) of the ZooAnimal object parameter.

```
class ZooAnimal
{
 private:
   char *name;
   int cageNumber;
   int weightDate;
   int weight;
 public:
   ZooAnimal (char*, int, int, int); // constructor function
   inline ~ZooAnimal () { delete [] name; }; // destructor function
   void changeWeight (int pounds);
   char* reptName ();
   int reptWeight ();
   int daysSinceLastWeighed (int today);
   friend ostream& operator<< (ostream& stream, ZooAnimal& Z);
   friend istream& operator>> (istream& stream, ZooAnimal& Z);
};
```

Module 10. Exception Handling Exercises

1. On the blank line below write a statement that throws an InvalidInput object (with an empty parameter list) if the variable number is greater than 99.

```
class InvalidInput {...};

void factorial (int& facto, int number)
{
if (number > 99)
  _____________________________________________________
else
 {
  facto = 1;
  int n = 1;
  while (++n <= number)
    facto *= n;
 }
}
```

2. Modify the last line below so that the block of code represented by { /*...*/ } is a Catch Handler for exceptions of type InvalidInput.

```
class InvalidInput {...};

void factorial (int& facto, int number)
{
if (number > 99)
  throw InvalidInput ();
else
 {
  facto = 1;
  int n = 1;
  while (++n <= number)
    facto *= n;
 }
}

void main()
{
 int east, eastfact;
 int west, westfact;
 ...
 cin >> east;
 ...
 cin >> west;
 ...
 try {
 ...
 factorial (eastfact, east);
 ...
 factorial (westfact, west);
 ...
 } // end of try block
 { /*...*/ }
}
```

3. Modify the changeWeight prototype so that it specifies that function changeWeight may only throw an exception of type UnderWeight.

```
class UnderWeight {...}; //exception class

class ZooAnimal
{
 private:
   static int minWeight;
   int weight;
 public:
   void changeMinWeight (int);
   void changeWeight (int);
};
```