Inheritance and Polymorphism

Problem 1. Briefly describe two benefits of using polymorphism. You may use the Shapes example discussed in class as the basis for your answer.

Problem 2. Consider the following classes:

```
class A {
public:
            void m() { cout << 1; }
            void n() { m(); p(); }
            virtual void p() { cout << 2; }
};

class B : public A {
public:
            void f() { n(); }
            void m() { cout << 3; }
            void p() { cout << 4; }
};
```

For each of the following cases determine if the code compiles successfully. If it does, determine the output, and if it does not, describe the error. In both cases, write a brief argument justifying your answer.

Problem 3. In a company, there are two kinds of employees: hourly and salaried. An hourly employee has a fixed wage per hour. The earning of an hourly employee will be wage × hours worked. A salaried employee has a fixed salary for the first 140 hours in month. If a salaried employee works more than 140 hours in a month, he gets 50% more. This means, if the fixed salary of an employee is 2,800,000 Rials, he gets 20,000 Rials an hour. So, if he works 180 hours in a month, he gets his fixed 2,800,000 Rials for the first 140 hours, and 40×20,000=800,000 Rials for the next 40 hours and the total earning will be 3,600,000 Rials.

Write an abstract class Employee with two subclasses Hourly and Salaried. These classes should have a method with the following signature for computing the total earning of the employee in a month if he has worked `hr` hours in that month:

```
    double earnings(int hr)
```

Define required fields, constructors, destructors, and member functions for your classes.

Problem 4. Write a class named Organization that contains a number of employees (of both types), and possibly a number of sub-organizations (which are of type organization themselves). Write a method `double total_earnings(int avgHr)` for Organization class, that computes the total amount of earnings of all employees of that organization and all its sub-organizations, if every employee works `avgHr` in a month.

Problem 5. Define a general list as a list that its elements are either integer numbers or general lists. For example `[2, [3,7], 1]` is a general list: the first and the third elements are integers and the second element is another general list. A more complex example of a general list is `[[2,[36]],4,[[2]],[]]`. Write a class named `Element` and derive two subclasses from it named `NumberElement` and `ListElement.` `NumberElement` is used to represent those elements in a general list that are numbers and `ListElement` represents general lists. In this case, `ListElement` will have a vector of (pointers to) `Elements` in it. Define a pure virtual member function named print for `Element,` and override it in both subclasses such that it either writes the number, or the general list of numbers depending on the overriding subclass. Define other methods required to use these classes such as the ones to add elements to the list, etc.

Problem 6. State whether each of the following is true or false. If false, explain why.

a. All `virtual` functions in an abstract base class must be declared as pure `virtual` functions.
b. Referring to a derived-class object with a base-class handle is dangerous.
c. A class is made abstract by declaring that class `virtual`.
d. If a base class declares a pure `virtual` function, a derived class must implement that function to become a concrete class.
e. Polymorphic programming can eliminate the need for `switch` logic.

Problem 7.

- What are `virtual` functions? Describe a circumstance in which `virtual` functions would be appropriate.
- Distinguish between `virtual` functions and pure `virtual` functions.
- Distinguish between static binding and dynamic binding.

Problem 8. Explain how the following program works without running it.

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

   class base {
   public:
             void Iam()   { cout << "base::Iam()"   << endl; }        /*1*/
     virtual void Iam_v() { cout << "base::Iam_v()" << endl; }        /*2*/
   };

   class drvd : public base {
   public:
     virtual void Iam()   { cout << "drvd::Iam()"   << endl; }        /*3*/
     virtual void Iam_v() { cout << "drvd::Iam_v()" << endl; }        /*4*/
   };

   class drvddrvd : public drvd {
   public:
     virtual void Iam()   { cout << "drvddrvd::Iam()"   << endl; }    /*5*/
     virtual void Iam_v() { cout << "drvddrvd::Iam_v()" << endl; }    /*6*/
   };

   int main() {
     base b;
     drvd d;
     drvddrvd dd;
     base *pb;
     drvd *pd;

     b.Iam();
     b.Iam_v();
     d.Iam();
     d.Iam_v();
     dd.Iam();
     dd.Iam_v();
     pb = &b;
     pb->Iam();
     pb->Iam_v();

     pb = &d;
     pb->Iam();
     pb->Iam_v();

     pb = &dd;
     pb->Iam();
     pb->Iam_v();

     pd = &d;
     pd->Iam();
     pd->Iam_v();

     pd = &dd;
     pd->Iam();
     pd->Iam_v();
   }
```

Problem 9. Explain how the following program works without running it.

```
#include <iostream>
using namespace std;
#include "Complex.h"

class Base {
public:
  virtual void f( int ) {
    cout << "Base::f(int)" << endl;
  }

  virtual void f( double ) {
    cout << "Base::f(double)" << endl;
  }

  virtual void g( int i = 10 ) {
    cout << "Base::g(int = " << i << ')' << endl;
  }
};

class Derived: public Base {
public:
  void f( Complex ) {
    cout << "Derived::f(Complex)" << endl;
  }

  void g( int i = 20 ) {
    cout << "Derived::g(int = " << i << ')' << endl;
  }
};
```

```
int main() {
  Base  b;
  Derived d;
  Base*   pb = new Derived;

  b.f(1.0);    // Base::f(double)

  d.f(1.0);    // Derived::f(Complex)

               // Derived::f(Complex) doesn't overload Base::f -
               // it hides them!! (Base::f(int) and Base::f(double) are not
               // visible in Derived.
               // Complex has an implicit conversion constructor for double
               // ==> Derived::f(Complex(1.0))


  pb->f(1.0);  // Base::f(double)

               // when called through pointer, no hiding. But as compiler
               // has to determine which overloaded function to call
               // at compile time, this decision is based on static type
               // of pointer (here Base)

  b.g();       // Base::g(int = 10)

  d.g();       // Derived::g(int = 20)

  pb->g();     // Derived::g(int = 10)

               // default values are "inserted" by the compiler.
               // again, at compile time only the static type is known
               // (here: Base), so compiler uses this default value .
               // However, the function actually called is determined at
               // runtime (as it is virtual), so Derived::g() is called

               // [RULE] Never change the default parameters of
               //        overridden inherited functions

  delete pb;   // ERROR: compiler generated Base::~Base(pb) is called
               // even if bp is point to a Derived object because
               // destructor is not declared virtual

               // [RULE] Make base class destructors virtual!
}
```

Exception Handling

Problem 10. Answer the following questions:

- If no exceptions are thrown in a `try` block, where does control proceed to after the `try` block completes execution?
- What happens if an exception is thrown outside a `try` block?

- Give a key advantage and a key disadvantage of using `catch(...)`.
- What happens if no `catch` handler matches the type of a thrown object?
- What happens if several handlers match the type of the thrown object?
- Why would a programmer specify a base-class type as the type of a `catch` handler, then `throw` objects of derived-class types?
- Suppose a `catch` handler with a precise match to an exception object type is available. Under what circumstances might a different handler be executed for exception objects of that type?
- What happens when a `catch` handler `throws` an exception?

Problem 11. A program contains the statement

```
throw;
```

Where would you normally expect to find such a statement? What if that statement appeared in a different part of the program?

Problem 12. Dealing with errors detected by constructors can be awkward. Exception handling gives us a better means of handling such errors. Consider a constructor for a `stack` class. The constructor uses `new` to obtain space from the free store. Suppose `new` fails. Show how you would deal with this without exception handling. Discuss the key issues. Show how you would deal with such memory exhaustion with exception handling. Explain why the exception-handling approach is superior.

Problem 13. Suppose a program `throws` an exception and the appropriate exception handler begins executing. Now suppose that the exception handler itself `throws` the same exception. Does this create infinite recursion? Write a program to check your observation.

Dynamic Memory Allocation and Linked Lists

سؤال ۱۴ - هدف این سؤال تعریف کلاسی به نام Poly است که یک چندجمله‌ای به شکل $a_nx^n + \ldots + a_1x + a_0$ را ذخیره کند. در انجام این تمرین از vector استفاده نکنید. کلاس Poly را به طور کامل بنویسید طوری که خواص زیر را داشته باشد:

- سازنده‌ها، عملگرها یا سایر متدهای ضروری برای این که هنگام کار با کلاس دچار مشکل حافظه نشویم تعریف شده باشند
- روی حداکثر درجه‌ی چندجمله‌ای محدودیتی وجود نداشته باشد
- در مواردی که عملی روی دو چندجمله‌ای انجام می‌شود فرض بر این است که درجه‌ی آنها لزوماً مساوی نیست
- اعضای زیر برای کلاس تعریف‌شده باشند:

اعضای رده	**مثال**

سازنده‌ای که ضرایب و درجه‌ی چندجمله‌ای را به عنوان پارامتر می‌گیرد. در مثال مقابل، P معادل چندجمله ای x^2+1 است.	**double** c[] = {2,0,1};
	Polynomial P(c, 2);

عملگر += برای جمع دو چندجمله‌ای و ذخیره‌ی حاصل در چندجمله‌ای	P += Q;

	سمت چپ
R = P + Q;	عملگر + برای جمع دو چندجمله‌ای و برگرداندن چندجمله‌ای حاصل
R = 2 * P;	عملگر * برای ضرب یک عدد ثابت در یک چندجمله‌ای (از سمت چپ) و برگرداندن چندجمله‌ای حاصل

cout << P[0];
P[0] = 2;
	عملگر [ ] برای دسترسی (<u>خواندن و نوشتن</u>) به ضریب i ام چندجمله‌ای

cout << P;	عملگر << برای نوشتن چندجمله‌ای در یک جویبار خروجی
cout << P.degree();	متد degree برای برگرداندن درجه‌ی چندجمله‌ای

In problems 15-19, use the LinkedList class as found on the course homepage.

Problem 15. Add a member function to `LinkedList` that concatenates another linked list to the end of the list.

Problem 16. Write a function that merges two ordered lists of integers into a single ordered list object of integers. Function `merge` should receive references to each of the lists to be merged and return a list of merged elements.

Problem 17. Write a member function `printListBackward` that recursively outputs the items in a linked list object in reverse order.

Problem 18. Write a member function `searchList` that recursively searches a linked list object for a specified value. The function should return a pointer to the node if it is found; otherwise, null should be returned.

Problem 19. Rewrite the `LinkedList` class so that every loop in the function bodies are converted into recursion.

Templates

Problem 20. Make the classes in Problem 5 generic by defining those required as templates. The type parameter determines the type of the (atomic) elements. For example, one can have a general list of characters instead of numbers.

STL and Function Objects

سؤال ۲۱ - هدف این سؤال نوشتن تابعی است با امضای زیر که برداری از تاریخ‌ها را می‌گیرد و اولین تاریخی را که در فصل تابستان است برمی‌گرداند.

```
Date first_in_summer(const vector<Date>& v);
```

الف) این تابع را به کمک `iterator` روی بردار پیاده‌سازی کنید.

ب) این تابع را به کمک تابع `find_if` و اشاره‌گر به تابع پیاده‌سازی کنید (مانند بخش ۲۱.۳ کتاب).

پ) این تابع را به کمک تابع `find_if` و `function object` پیاده‌سازی کنید (مانند ساختار `Odd` تعریف شده در اسلاید ۳۳ `Container`ها).

ت) تابع تعمیم‌یافته‌ای بنویسید با امضای زیر که شماره‌ی فصل (۱ تا ۴) را به عنوان پارامتر دوم می‌گیرد و اولین تاریخی را که در آن فصل است برمی‌گرداند. این قسمت را به شیوه‌ی بخش ۲۱.۴ کتاب یا اسلاید ۳۴ `Container`ها پیاده‌سازی کنید.

```
Date first_in_season(const vector<Date>& v, int season);
```