Exam-M0313 模板

一、请举一些实例，说明在什么情况下使用模板比继承更合适、什么时候可以结合模板和继承。

二、请将下面的 shell_sort 函数参数化（定义成函数模板）

```
void shell_sort(std::vector<int>& vec) {
  const size_t n = vec.size();
  for (int gap = n / 2; 0 < gap; gap /= 2)
    for (int i = gap; i < n; ++i)
      for (int j = i - gap; 0 <= j; j -= gap)
        if (vec[j + gap] < vec[j]) {
          int tmp = vec[j + gap];
          vec[j + gap] = vec[j];
          vec[j] = tmp;
        }
}
```

请注意：

1. vector 中容纳的是指针类型的情形
2. vector 中容纳的是 char*类型的情形

三、模板实例化

```
struct A {
  A() {
  }
  void print() {
  }
};

struct B {
  B(int n) {
  }
  void print() {
  }
};

class C {
  void print() {
  }
```

```
};

template<typename T, int n = 8>
class D {
  T ar[n];
public:
  void printAll() {
    for (int i = 0; i < n; ++i)
      ar[i].print();
  }
};

int main() {
  D<A> da;
  da.printAll();
  D<B> db;
  db.printAll();
  D<C> dc;
  dc.printAll();
}
```

提示：通过这段代码可以比较继承和模板所体现的多态机制的约束。

四、名字解析

1. 请分析下面代码中可能存在的错误，并修正之：

```
// #1
template<typename T>
int find(const vector<T>& items) {
  for (vector<T>::iterator i = items.end(); i < items.end(); ++i) {
    // ...
  }

  // ...
}


// #2
struct A {
  template<typename T> T* getPtr();
  template<typename T> void setValue(const T& v);
};

template<typename T>
void testA(T a) {
```

```
  int* ptr = a.getPtr<int> ();
  a.setValue(12);
}
```

2. 请判断下面代码的输出结果，并解释为什么会出现这种情况。

```
// #3
template<typename T> struct B {
  double x;
};

int x = 16;

template<typename T>
struct D: B<T> {
  int getX() const {
    return x;
  }
};

int main() {
  cout << D<char>().getX() << endl;
}
```

五、模板特化（此题作一般了解即可）

请看看下列代码究竟要做什么，并推导其输出

```
struct empty {
};

template<typename H, typename T>
struct node {
  typedef H head;
  typedef T tail;
};

template<typename T1 = empty, typename T2 = empty, typename T3 =
empty, typename T4 = empty, typename T5 = empty, typename T6 =
empty, typename T7 = empty, typename T8 = empty, typename T9 =
empty,  typename T10 = empty, typename T11 = empty, typename T12 =
empty>
struct list {
  typedef node<T1, node<T2, node<T3, node<T4,
     node<T5, node<T6, node<T7, node<T8, node<T9, node<T10,
node<T11, node<T12, empty> > > > > > > > > > > > type;
```

```
};

template<typename L>
struct length {
  enum {
    value = 1 + length<typename L::tail>::value
  };
};

template<>
struct length<empty> {
  enum {
    value = 0
  };
};

template<typename L>
struct is_empty {
  enum {
    value = false
  };
};

template<>
struct is_empty<empty> {
  enum {
    value = true
  };
};

template<typename T, typename U>
struct is_same_type {
  enum {
    value = false
  };
};

template<typename T>
struct is_same_type<T, T> {
  enum {
    value = true
  };
};

template<typename T, typename L>
struct is_member {
```

```
  enum {
    value = is_same_type<T, typename L::head>::value || is_member<T,
        typename L::tail>::value
  };
};

template<typename T>
struct is_member<T, empty> {
  enum {
    value = false
  };
};

// test
typedef list<bool, char, unsigned char, signed char, int, short,
long,
    unsigned, unsigned long, unsigned short>::type int_types;

typedef list<float, double, long double>::type real_types;

int main() {
  cout << is_same_type<int, int>::value << '\n';
  cout << is_same_type<int, signed int>::value << '\n';
  cout << is_same_type<int, unsigned int>::value << '\n';
  cout << is_member<int, int_types>::value << '\n';
  cout << is_member<float, int_types>::value << '\n';
  cout << is_member<ostream, int_types>::value << '\n';
}
```