Virtual Functions

```
class Animal {
   // no "bark" is defined
};
class Dog: public Animal {
 public:
   virtual void bark();
};
class KDog: public Dog {
 public:
   void bark();
};
class GDog: public KDog {
 public:
   void bark();
};
```

```
int main() {
   Animal *a = new KDog;
   a->bark();

   Dog *b = new KDog;
   b->bark();

   Dog *c = new GDog;
   c->bark();

   Kdog *d = new Gdog;
   d->bark();
}
```

➔Any compilation error?
➔Which bark() is called?

Practice #3 (for lecture note #3)

C++ Review Part III:

Overloading and Polymorphism

Due: 9pm, Friday, 2018-10-11

Notes and Advices:

1. Submission of the practice is NOT mandatory. The main purpose of these problems is for you to practice and get refreshed on the contents of the lecture note for the coming class meeting. Practice of the problems before class is HIGHLY recommended.
2. However, in case you need to "bargain" your term grade while you are below the next/higher letter grade for a small enough margin, you can talk to me if you have been consistently submitting your practices in time.
3. Note that your submission of the practice will NOT be graded or checked (by me or TA). Again, this is for your practice only.
4. The submissions on NTU CEIBA have strict deadline. Usually it is before midnight of the class meeting.
5. Please put all the practice problems in a directory and name it as <yourID>_pr3 (replace yourID with your ID, and omit <>) and compress it as "tar zcvf <yourID>_pr3.tgz <yourID>_pr3. Submit the .tgz file.
6. Practice problems will be, hopefully, uploaded to CEIBA no later than the previous weekend of the class.

Problems // "(pn)" refers to the page #n in the lecture note

1. (p10) Define a base `class Base` and its derived `class Derived`
   - For `class Base`, define two public functions: `virtual void f();  void g();`
   - For `class Derived`, define two public functions: `void f();  void g();`
   - In the above functions, print out message showing that the function is called (e.g. "`Base::f() is called`").

   In main, instantiate two objects "`Base b`" and "`Derived d`". Use them to call `f()` and `g()`
   - Which functions are called?
   - What does "`virtual`" keyword do in this case?
     What if we do NOT declare "`virtual`"?
   - What if we do NOT declare inheritance?

2. (p13) Define a base `class Base` and its derived `class Derived`
   - For `class Base`, define three public functions: `virtual void f(); void g(); virtual void h();`
   - For `class Derived`, define two public functions: `void f();  void g();`
   - In the above functions, print out message showing that the function is called (e.g. "`Base::f() is called`").

   In main, instantiate three objects "`Base *p = new Derived`", "`Base *q = new Base`" and "`Derived *r = new Derived`". Use them to call `f()`, `g()` and `h()`

   - Is it OK NOT to define "`Derived::h()`"?

   - Which functions are called?

3. (p25) Define a base `class Base` and its derived `class Derived`
   - For `class Base`, define three public functions: `virtual void f(); void g(); virtual void h();`
   - For `class Derived`, define two public functions: `void f(); void g();`
   - In the above functions, print out message showing that the function is called (e.g. "`Base::f() is called`").

   In main, instantiate an object "`Base *p = new Derived`". Use it to call `f()`, `g()` and `h()`
   - Any compilation error?
   - Which ones are called?

   Follow the modifications below, see if there is any compilation error for each of the steps? Try to read the error message and understand why.
   1. Add one more public function `void h()` for class Derived without function body
   2. Make "`Base::h()`" pure virtual
   3. Comment out "`Derived::h()`"
   4. Comment out the call "`p->h()`" in "`main()`"
   5. Uncomment "`Derived::h()`" and write a function body for it; uncomment out the call "`p->h()`";
   6. add a "Base *q = new Base" in "main()".

4. (p48) Define a `class A` and with data member "`int _d`" and a constructor to initialize this data member (with default = 0).
   - Implement all the overloaded operators in the p45 (maybe except "`[]`")
   - Play with the combinations of the operators, such as "`a + b - c`", "`a++ + b`", "`++a * c`", "`a += b + c`"… Check if the behavior matches your expectation.
   - Can you overload operators "`()`", "`{}`", "`->`", "`.`"? Why and why not?

5. (p50) In the following example:

```
template<class T> class Array
{
  public:
     Array(size_t i = 0) { _data = new T[i]; }
     T& operator[] (size_t i) { return _data[i]; }
     const T& operator[] (size_t i) const {
        return _data[i]; }
  private:
     T  *_data;
};
template<class T>
void f(const Array<T>& arr) {
   arr[1] = 20;
   int a = arr[0];
```

```
}
int main()
{
   Array<int> arr(10);  // size = 10
   f(arr);
}
```

- Compile it and run. Any error?
- Comment out `const` version of `operator []`. Compile again. Any error?
- Change the `non-const` version to "`T& operator[] (size_t i) const`…" and compile again. Any error? If not, does this make sense?

6. (p56) Define a `class A` and with data member "`int _d`" and a constructor to initialize this data member (with default = 0). Instantiate "`A a1(10), a2(20)`" and call "`a1 + a2`".
   - Overload `operator +` as its member function.
   - Change it to a public function with two `class A` objects as its parameters.
   - What happens if both of the above exist?

   Change the behavior of the "operator +" to subtraction. Any compilation error?

7. (p66) Figure out why the following code has compilation errors.

```
#include <algorithm>
#include <iostream>
using namespace std;
struct Compare
{
   virtual bool operator() (int, int) const = 0;
};
struct Less : public Compare
{
   bool operator() (int i, int j) const { return (i < j); }
};
struct Greater : public Compare
{
   bool operator() (int i, int j) const { return (i > j); }
};
void f(const Compare& c)
{
   int arr[10] = { 4,2,6,7,1,3,5,9,8,0 };
   ::sort<int*>(arr, arr+10, c);
   for (int i = 0; i < 10; ++i)
```

```
        cout << arr[i] << endl;
}
int main()
{
   f(Less());
   f(Greater());
}
```