Queens College, CUNY, Department of Computer Science

Object-Oriented Programming in C++

CSCI 211/611

Summer 2018

Instructor: Dr. Sateesh Mane

due date Friday, August 3, 2018, 11.59 pm

Homework: Polymorphism #1

- **Experience with other classes has demonstrated that in many cases the source of difficulty is not the mathematics or the programming.**
- **The source of difficulty is the English (understanding the text).**
- **If you do not understand the words in the lectures or homework, THEN ASK.**
- **If you do not understand the concepts in the lectures or homework, THEN ASK.**
- **Send me an email, explain what you do not understand.**
- **Do not just keep quiet and then produce nonsense in exams.**

- **Consult your lab instructor for assistance.**
- **You may also contact me directly, but I cannot promise a prompt response.**
- Please submit your inquiry via email, as a file attachment, to `Sateesh.Mane@qc.cuny.edu`.
- Please submit one zip archive with all your files in it.
  1. The zip archive should have either of the names (CS211 or CS611):

     `StudentId_first_last_CS211_hw_polymorphism1.zip`
     `StudentId_first_last_CS611_hw_polymorphism1.zip`
  2. The archive should contain one "text file" named "hw_polymorphism1.[txt/docx/pdf]" (if required) and cpp files named "Q1.cpp" and"Q2.cpp" etc.
  3. Note that a text file is not always required for every homework assignment.
  4. Note that not all questions may require a cpp file.

General information

- You should include the following header files, to run the programs below.

```
#include <iostream>
#include <fstream>
#include <sstream>
#include <iomanip>
#include <string>
#include <cmath>
```

- If you require additional header files to do your work, feel free to include them.
- **Include the list of all header files you use, in your solution for each question.**
- The questions below do not require complicated mathematical calculations.
- If for any reason you require help with mathematical calculations, **ask the lab instructor or the lecturer.**

Inheritance tree

- This homework is a carbon copy of the homework on inheritance.
- The classes are renamed “AA” etc.
- We shall employ the following inheritance tree.

CC DD

BB

AA

- AA is the base class and has a **virtual destructor.**
- All the classes override the method `print()`.
- **The method `print()` is virtual.**
- *We shall write print statements in the constructors, destructors and assignment operators, to keep track of the flow of logic.*
- *You must observe how the results are different from inheritance without polymorphism.*

Q1 Base class AA

- Write the following class AA.
- It has a virtual destructor, and the method `print()` is virtual.

```
class AA {
public:
  AA() {
    ip = new int;
    *ip = 0;
    cout << "AA default constructor" << endl;
  }

  AA(int x) : ip(new int) {
    *ip = x;
    cout << "AA non-default constructor" << endl;
  }

  AA(const AA& orig) {
    ip = new int;
    *ip = *orig.ip;
    cout << "AA copy constructor" << endl;
  }

  AA& operator=(const AA& rhs) {
    if (this == &rhs) return *this;
    *ip = *rhs.ip;
    cout << "AA operator=" << endl;
    return *this;
  }

  virtual ~AA() {                                     // virtual
    delete ip;
    cout << "AA virtual destructor" << endl;
  }

  int get_i() const { return *ip; }
  void set_i(int x) { *ip = x; }
  virtual void print() const {                        // virtual
    cout << "AA print: " << *ip << endl;
  }

protected:
  int *ip;
};
```

Q2 Classes BB, CC, DD

- Write classes BB, CC and DD.
- **They are the same as B, C and D, just rename to BB, CC and DD.**
- Change all the debugging print statements to “BB” etc.

```
class BB : public AA;
class CC : public BB;
class DD : public BB;
```

Q3 Functions “show”

- **Write the following functions.**

```
void show(const AA &aaref)
{
  aaref.print();
}

void show(const BB *bbptr)
{
  bbptr->print();
}

void show(const DD *ddptr)
{
  ddptr->print();
}
```

Q4 Main program #1

- **Run the following main program.**
- Make sure you understand it and can explain all the print statements.

```
// include relevant headers and class declarations
using namespace std;

int main()
{
  A *pa = new B(5, "xyz");                // from homework on inheritance
  pa->print();
  delete pa;

  AA *paa = new BB(5, "xyz");
  paa->print();
  delete paa;

  return 0;
}
```

- **Observe how `paa->print()` invokes `BB::print()`.**
- **Observe how the call to the virtual destructor releases the memory correctly.**
- This is the effect of polymorphism.
- The pointer is of type `AA`, but it knows it is pointing to an object of type `BB`.

Q5 Main program #2

- **Run the following main program.**
- Make sure you understand it and can explain all the print statements.

```
// include relevant headers and class declarations
using namespace std;

int main()
{
  A a(2);
  B b1;
  B b2(3, "abc");
  b1 = b2;

  show(a);
  show(b1);
  show(&b2);

  AA aa(2);
  BB bb1;
  BB bb2(3, "abc");
  bb1 = bb2;

  show(aa);
  show(bb1);
  show(&bb2);

  return 0;
}
```

Q6 Main program #3

- **Run the following main program.**
- Make sure you understand it and can explain all the print statements.

```
// include relevant headers and class declarations
using namespace std;

int main()
{
  C c(4, "alpha");
  A &ra = c;
  B *pb = &c;
  c.print();
  show(c);
  show(ra);
  show(pb);

  CC cc(4, "alpha");
  AA &raa = cc;
  BB *pbb = &cc;
  cc.print();
  show(cc);
  show(raa);
  show(pbb);

  return 0;
}
```

Q7 Main program #4

- **Run the following main program.**
- Make sure you understand it and can explain all the print statements.
- Note that D has access to all the accessors and mutators in A and B.

```
// include relevant headers and class declarations
using namespace std;

int main()
{
  D *darray = new D[2];
  darray[0].set_i(7);
  darray[0].set_s("pdstring");
  darray[0].set_dp(8.2, 9.3);

  show(darray[0]);
  show(&darray[0]);

  darray[1] = darray[0];
  cout << darray[1].get_i() << endl;
  cout << darray[1].get_s() << endl;
  cout << darray[1].get_d0() << endl;
  cout << darray[1].get_d1() << endl;

  D dcopy(darray[1]);

  delete [] darray;

  dcopy.print();

  DD *ddarray = new DD[2];
  ddarray[0].set_i(7);
  ddarray[0].set_s("pdstring");
  ddarray[0].set_dp(8.2, 9.3);

  show(ddarray[0]);
  show(&ddarray[0]);

  ddarray[1] = ddarray[0];
  cout << ddarray[1].get_i() << endl;
  cout << ddarray[1].get_s() << endl;
  cout << ddarray[1].get_d0() << endl;
  cout << ddarray[1].get_d1() << endl;
  DD ddcopy(ddarray[1]);

  delete [] ddarray;

  ddcopy.print();

  return 0;
}
```

Q8 Class EE, etc.

- You do not need to write a class EE, but you can if you wish.
- It should behave the same as the class E, with polymorphism added.
- **Write a main program to perform other tests for polymorphism. Use your imagination.**