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: Inheritance

- **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_inheritance.zip
     StudentId_first_last_CS611_hw_inheritance.zip
     ```
  2. The archive should contain one “text file” named “hw_inheritance.[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

- There is not much time left in the semester now, and inheritance and polymorphism (to come soon) are complicated ideas.
- Hence I shall do a lot of the work for you in this assignment.
- We shall employ the following inheritance tree.

```
C   D
  B
  A
```

- A is the base class, B inherits from A, and C and D both derive from B.
- The classes contain pointers and dynamic memory, so deep copies are required.
- The classes override some methods but not others.
- All the class data members are protected, not private.
- *We shall write print statements in the constructors, destructors and assignment operators, to keep track of the flow of logic.*

Q1 Class A

- Write the following class A. It contains a pointer to int, and dynamic memory.

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

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

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

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

  ~A() {
    delete ip;
    cout << "A destructor" << endl;
  }

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

protected:
  int *ip;
};
```

Q2 Class B

- Write the following class B. It contains a pointer to `string`, and dynamic memory.
- **Observe how the constructors are written, also the assignment operator.**
- B overrides the method `print()` but not the others.
- B has accessor and mutator methods not available in A.

```
class B : public A {
public:
  B() : sp(new string) {
    cout << "B default constructor" << endl;
  }

  B(int x, string s) : A(x), sp(new string)  {
    *sp = s;
    cout << "B non-default constructor" << endl;
  }

  B(const B& orig) : A(orig), sp(new string) {
    *sp = *orig.sp;
    cout << "B copy constructor" << endl;
  }

  B& operator=(const B& rhs) {
    if (this == &rhs) return *this;
    A::operator=(rhs);                  // deep copy A
    *sp = *rhs.sp;
    cout << "B operator=" << endl;
    return *this;
  }

  ~B() {
    delete sp;
    cout << "B destructor" << endl;
  }

  string get_s() const { return *sp; }
  void set_s(string s) { *sp = s; }
  void print() const { cout << "B print: " << *ip << "  " << *sp << endl; }

protected:
  string *sp;
};
```

Q3 Class C

- Write the following class C. The class C has no data members.
- The method `print()` in the class C invokes the method `A::print()` in the class A.
- **Write the function body for the assignment operator. Include a print statement.**

```
class C : public B {
public:
  C() {
    cout << "C default constructor" << endl;
  }

  C(int x, string s) : B(x,s) {
    cout << "C non-default constructor" << endl;
  }

  C(const C& orig) : B(orig) {
    cout << "C copy constructor" << endl;
  }

  C& operator=(const C& rhs) {
    // write the code                              // ** WRITE THE FUNCTION BODY **

    cout << "C operator=" << endl;
    return *this;
  }

  ~C() {
    cout << "C destructor" << endl;
  }

  void print() const {
    cout << "C print invokes A::print" << endl;
    A::print();
  }
};
```

Q4 Class D

- Write the following class D. The class D has a pointer to double, which is an array.
- For simplicity we allocate the array to a length 2, to avoid unnecessary checks if the array is NULL.
- D overrides the method `print()` but not the others.
- D has accessor and mutator methods not available in A or B.
- **Write the function bodies where required below.**
- **Just for practice, write the constructors and destructors and assignment non-inline.**

```
class D : public B {
public:
  D();
  D(int x, string s, double d);
  D(const D& orig);
  D& operator=(const D& rhs);
  ~D();

  double get_d0() const { return dp[0]; }
  double get_d1() const { return dp[1]; }
  void set_dp(double d1, double d2) { dp[0] = d1; dp[1] = d2; }
  void print() const {
    cout << "D print: " << *ip << "  " << *sp << "  " << dp[0] << "  " << dp[1] << endl;
  }

protected:
  double *dp;
};
```

- See next page.

```
D::D() {
  dp = new double[2];                           // INITIALIZE ARRAY ELEMENTS TO ZERO
  cout << "D default constructor" << endl;
}

D::D(int x, string s, double d) : B(x,s), dp(new double[2])  {
                                                // INITIALIZE ARRAY ELEMENTS TO d
  cout << "D non-default constructor" << endl;
}

D::D(const D& orig) : B(orig), dp(new double[2]) {
                                                // WRITE DEEP COPY
  cout << "D copy constructor" << endl;
}

D& D::operator=(const D& rhs) {
  if (this == &rhs) return *this;
                                                // INVOKE OPERATOR= FOR B
                                                // WRITE DEEP COPY
  cout << "D operator=" << endl;
  return *this;
}

D::~D() {
                                                // RELEASE MEMORY CORRECTLY
  cout << "D destructor" << endl;
}
```

Q5 Functions “show”

- **Write the following functions.**

```
void show(const A &aref)
{
 aref.print();
}

void show(const B *bptr)
{
  bptr->print();
}

void show(const D *dptr)
{
  dptr->print();
}
```

Q6 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");
  pa->print();
  delete pa;
  return 0;
}
```

Q7 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);

  return 0;
}
```

Q8 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);

  return 0;
}
```

Q9 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();

  return 0;
}
```

Q10 Class E

- **Write a class E.**

  1. The class E derives from D.
  2. The class E contains a dynamically allocated array of strings.
  3. Write suitable constructors, an assignment operator and a destructor for E.
  4. Write print statements in all of them, to keep track of the flow of logic.
  5. Write suitable accessors and mutators for E.

- **Write a main program to test your code for the class E.**