Queens College, CUNY,   Department of Computer Science

Object-oriented programming in C++

CSCI 211 / 611

Summer 2018

Instructor: Dr. Sateesh Mane

Midterm 2

Thursday July 26, 2018 (in class)

due date Sunday July 29, 2018, 12:00 noon (take home)

take home version, credit only for Questions 3–10

- **NOTE: It is the policy of the Computer Science Department to issue a failing grade to any student who either gives or receives help on any test.**
- **A student caught cheating on any question in an exam, project or quiz will fail the entire course.**
- This is a **closed-book** test.
- Once you leave the classroom, you cannot come back to the test.
- **Any problem to which you give two or more (different) answers receives the grade of zero automatically.**
- Submit your solution in the envelope provided, with your name and student id on the cover.
- Write your name, student id and the question number at the top of every page of your solution.
- **Answers must be written in legible handwriting.**
- **A failing grade will be awarded if the examiner is unable to decipher your handwriting.**
- **In all questions where you are asked to submit programming code, programs which display any of the following behaviors will receive an automatic F:**
  1. Programs which do not compile successfully (non-fatal compiler warnings are excluded, e.g. use of deprecated features).
  2. Array out of bounds, reading of uninitialized variables (including null pointers).
  3. Operations which yield NAN or infinity, e.g. divide by zero, square root of negative number, etc. *Infinite loops.*
  4. Programs which do NOT implement the public interface stated in the question.
- **In addition, note the following:**
  1. All debugging statements (for your personal testing) should be commented out.
  2. Program performance will be graded solely on the public interface stated in the questions.

1 General information

1.1 Hardware

- The questions in this exam do not involve problems of overflow.
- Solutions involving the writing of programs will not be judged if they work on a 64–bit instead of a 32–bit computer.

1.2 Class `PointXY`

- The class `PointXY` will be used below in questions in this exam.

```
class PointXY {
public:
  PointXY() { x = 0; y = 0; }

  void set(double a, double b) { x = a; y = b; }
  double getx() const { return x; }
  double gety() const { return y; }

private:
  double x, y;
};
```

2 Question 2

- **This question uses the class `PointXY` defined in Sec. 1.2.**
- Declare a class `Polygon` as follows.
- The data in class `Polygon` is an integer `num` and a pointer `PointXY *pxy`, both private.

```
class Polygon {
public:
  Polygon(int n);                          // write code
  // copy constructor                      // write code
  // assignment operator                   // write code
  // destructor                            // write code
  (return type) get_point(int n) const;    // write code
  void set_point(const PointXY &p, int n); // write code
  int get_num() const { return num; }      // nothing to do, code is given to you

private:
  int num;
  PointXY *pxy;
};
```

- **Write a non-default constructor with an input `int n`.**
  1. If $n > 0$, set `num=n` and dynamically allocate `pxy` to an array of length `num`.
  2. Else if $n \leq 0$, set `num = 0` and `pxy = NULL`.
- **Write a copy constructor, assignment operator and destructor to perform appropriate copies and management of dynamic memory.**
- **Write a method `get_point(int n)`.**
  1. Return the address of `pxy[n]` if the value of n is valid.
  2. Else return NULL.
  3. The method is `const`.
  4. Write the function value to have the correct type.
- **Write a method `set_point(const PointXY &p, int n)`.**
  1. Set `pxy[n] = p` if the value of n is valid.
  2. Else do nothing.

3 Question 3

- For two points u and v, the distance d between them is given as follows.

$$d = \sqrt{(u_x - v_x)^2 + (u_y - v_y)^2} \,.$$

- **Write a function `distance` to calculate the distance between two `PointXY` objects.**

```
double distance(const PointXY &u, const PointXY &v);
```

- **Write a function "`area_perimeter`" to calculate the area and perimeter given an input vector of three points.** The three points form a triangle.

- **Your function should enable the program below to compile and run correctly.**

```
// #include relevant headers
using namespace std;

// class declaration and functions

int main()
{
  int n = 3;
  vector<PointXY> v(n);
  // set x,y coordinates
  double area, perimeter;

  area_perimeter(v, area, perimeter);               // function call
  area_perimeter(v, area, perimeter);               // call twice why not

  cout << "area = " << area << endl;
  cout << "perimeter = " << perimeter << endl;
  return 0;
}
```

- **Write an appropriate signature for the "`area_perimeter`" function.**

1. Declare variables a, b, c and compute the edge lengths of the triangle `a = distance(v[0],v[1])`, similarly b for `(v[1],v[2])` and c for `(v[2],v[0])`.
2. The perimeter of the triangle is the sum of the edge lengths `a + b + c`.
3. Define the **semi-perimeter** as `s = perimeter/2.0`.
4. The area of the triangle is given by the **Heron's formula.**

$$\text{area} = \sqrt{s(s-a)(s-b)(s-c)} \,.$$

5. It is named after Hero of Alexandria and I do not know why it is "Heron's" formula.

4 Question 4

- You are given two classes `Chicken` and `Egg`, described below.
- The class `Chicken` contains a vector of `Egg` objects.

```
class Chicken {
public:
  void layEgg();
  int numEggs() const;
  void getEggs(vector<Egg> &v);

private:
  vector<Egg> vec_eggs;
};
```

  1. The method `numEggs` returns the size of the vector `vec_eggs` and is `const`.
  2. Method `layEgg`: instantiate an `Egg` object and push it back onto the vector `vec_eggs`. Do not use dynamic memory. Just instantiate "`Egg e`" and push back e onto the vector.
  3. In the method `getEggs`, do two things:
     (a) Set v to a copy of the vector `vec_eggs`.
     (b) Clear the vector `vec_eggs`.
     (c) In other words, the farmer collects the eggs and the chicken has no more eggs.

- The class `Egg` contains a method `hatch` which returns a pointer to a new `Chicken`.

```
class Egg {
private:
  bool born;

public:
  Egg() { born = false; }
  bool isBorn() const { return born; }
  Chicken* hatch();
};
```

  1. The default constructor initializes `born` to `false`. The code is written for you.
  2. The accessor method `isBorn()` returns the value of `born`. The code is written for you.
  3. In the method `hatch`, do the following.
     (a) If `born` is true, return NULL and exit. An egg cannot hatch more than once.
     (b) If `born` is false, set `born = true`.
     Dynamically create a new `Chicken` and return a pointer to the dynamic memory.
     In other words, a new chicken is born.

- **Write forward class declarations and function bodies for `Chicken` and `Egg`.**
- **Fill the missing code in the following program.**
- **State if dynamic memory must be released at the end of the main program.**
  If yes, write the relevant code in the main program.
- *Write code for the numbered lines only. Do not repeat the entire program.*

```
// #include relevant headers
using namespace std;

// code for classes Chicken and Egg

int main()
{
  Chicken c;
  int n = 3;
  for (int i = 0; i < n; ++i)
    c.layEgg();

  cout << // number of eggs in c                         // #1

  vector<Egg> v;
  c.getEggs(v);
  cout << // number of eggs in c                         // #2

  Chicken *cptr1 = v[0].hatch();
  Chicken *cptr2 = v[2].hatch();

  // cptr2 lays an egg                                   // #3
  cout << // number of eggs in cptr1                     // #4
  cout << // number of eggs in cptr2                     // #5

  for (int i = 0; i < v.size(); ++i) {
    // if v[i] is born (== true) print output            // #6
      cout << "egg is hatched: " << i << endl;
  }

  // chickens are eaten
  // release dynamic memory (if required)                // #7 (if not required, say so)
  return 0;
}
```

- **Write the output printed by the above program.**

5 Question 5

5.1 Int

- **Write code to make the program below work correctly.**
  1. Define two integers:
     `n1 = (first four digits of your student id)`
     `n2 = (last four digits of your student id).`
  2. For example if your student id is 23054611, then `n1 = 2305` and `n2 = 4611`.
  3. Push back `n1` and `n2` onto the vector in the program.
- **State if dynamic memory must be released at the end of the main program.**
  If yes, write the relevant code in the main program.
- *Write code for the numbered lines only. Do not repeat the entire program.*

```
// #include relevant headers
using namespace std;

int main()
{
  int n = 5;
  vector<int> vi(n);
  cout << (size) << "  " << (capacity) << endl;          // #1

  vi.clear();
  cout << (size) << "  " << (capacity) << endl;          // #2

  int n1 = ..., n2 = ...;                                // #3 (write on one line)
  //  push back n1, n2 onto vector vi                    // #4 (write on separate lines)
  for (int i = 0; i < vi.size(); i++)
    cout << vi[i] << endl;

  int *ip = new int;
  // set contents of ip to largest integer in vector vi  // #5 (if equal, use n1)
  cout << (contents of ip) << endl;                      // #6

  // increment contents of ip by 7                       // #7
  cout << (contents of ip) << endl;                      // #8

  // release dynamic memory (if required)                // #9 (if not required, say so)
  return 0;
}
```

- **Write the output printed by the above program.**

5.2 String

- **Write code to make the program below work correctly.**
  1. Define two strings: `s1 = (your first name)` and `s2 = (your last name)`.
  2. Push back `s1` and `s2` onto the vector in the program.
- **State if dynamic memory must be released at the end of the main program.**
  If yes, write the relevant code in the main program.
- *Write code for the numbered lines only. Do not repeat the entire program.*

```
// #include relevant headers
using namespace std;

void print(const string* ps)
{
  int len = (length of string ps points to)      // #1
  cout << len << endl;
  cout << (contents of ps) << endl;              // #2
}

int main()
{
  vector<string> vs;
  cout << (size) << "  " << (capacity) << endl;  // #3

  string s1(...), s2(...);                       // #4 (write on one line)
  //  push back s1, s2 onto vector vs            // #5 (write on separate lines)
  cout << (size) << "  " << (capacity) << endl;  // #6

  string *ps = NULL;
  for (int i = 0; i < vs.size(); i++) {
    ps = (address of vs[i])                      // #7
    cout << (contents of ps) << endl;            // #8 ** use pointer **
  }
  ps = (address of longest string in vector vs)  // #9 (if same length, use s1)
  print(ps);

  // release dynamic memory (if required)        // #10 (if not required, say so)
  return 0;
}
```

- **Write the output printed by the above program.**

6 Question 6

- **Write a class "`Math`" with the following properties.**

  1. The class has two public data members `pi` and `root2`.
  2. Both are `static` and `const` and have type `double`.
  3. You are given the values `pi = 3.1416` and `root2 = 1.4142` (good enough for this exam).

```
class Math
{
  // etc
};
```

- **Write the class declaration and the initialization statements for the static data.**
- **Write the print statements in the program below, using the `Math` class.**

```
// #include relevant headers
using namespace std;

// declaration of class Math, etc

int main()
{
  cout << (value of pi using Math class) << endl;          // write code
  cout << (value of root2 using Math class) << endl;       // write code
  return 0;
}
```

- **Bonus**
  Formulate the `Math` class so that it is not possible to create or make copies of objects of the class `Math`.

7 Question 7

- We know that it is possible to write assignments of `int` to `double` and vice-versa.
- We also know that the sums `int + double` and `double + int` both return `double`.

```
int i=1, j=2;
double x=3.3, y=4.4;
x = i;                          // double = int
j = y;                          // int = double
x = i + y;                      // int + double is double
y = x + j.                      // double + int is double
```

- **You are given the forward declarations of two classes `PointDbl` and `PointInt`.**
- **State which methods in the classes below are accessors.**
- **State which methods in the classes below are mutators.**

```
class PointDbl;
class PointInt;

class PointDbl {
public:
  PointDbl() { x = 0; y = 0; }
  void set(double a, double b) { x = a; y = b; }
  double getx() const { return x; }
  double gety() const { return y; }

  // operator= ...                              // to be written

private:
  double x, y;
};

class PointInt {
public:
  PointInt() { x = 0; y = 0; }
  void set(int a, int b) { x = a; y = b; }
  int getx() const { return x; }
  int gety() const { return y; }

  // operator= ...                              // to be written

private:
  int x, y;
};
```

- **Overload the following operators as class methods.**
  1. `operator= (const PointInt &rhs)`, method of class `PointDbl`, returns reference `PointDbl&`.
  2. `operator= (const PointDbl &rhs)`, method of class `PointInt`, returns reference `PointInt&`.
- **Explain if a check for self-assignment is necessary.**
  If yes, write the self-assignment test in your code. If no, explain why not.
- *Note: the assignment* `PointInt = PointDbl` *will result in loss of decimal places.*
  Write (x in `PointInt`) = (x in `PointDbl`), etc. (analogous to "`j = y`" on the previous page).
- **Overload two versions of `operator+`, both as external functions.**
  1. `operator+` binary operator with inputs `(PointDbl, PointInt)`, returns `PointDbl`.
  2. `operator+` binary operator with inputs `(PointInt, PointDbl)`, returns `PointDbl`.
- If you do your work correctly, your code will support the following operations.

```
// assignment PointDbl = PointInt
// assignment PointInt = PointDbl            // loss of decimal places
// addition PointDbl =  PointDbl + PointInt
// addition PointDbl =  PointInt + PointDbl
```

- **Write the output printed by the program below.**

```
// #include relevant headers
using namespace std;

// class declarations and functions

int main()
{
  PointDbl dp, pdbl;
  PointInt ip, pint;
  pdbl.set(1.1, 2.2);
  pint.set(3, 4);
  dp = pint;                    // Dbl = Int
  ip = pdbl;                    // Int = Dbl

  PointDbl s1 = pdbl + pint;    // Dbl + Int returns Dbl
  PointDbl s2 = pint + pdbl;    // Int + Dbl returns Dbl

  cout << "dp " << dp.getx() << "  " << dp.gety() << endl;
  cout << "ip " << ip.getx() << "  " << ip.gety() << endl;
  cout << "s1 " << s1.getx() << "  " << s1.gety() << endl;
  cout << "s2 " << s2.getx() << "  " << s2.gety() << endl;
  return 0;
}
```

8 Question 8

- In C++ we can write code such as the following.

```
int a = ...         // value
double d = ...      // value
if (a == d)         // compare int and double
```

- **This question uses the classes `PointDbl` and `PointInt` defined in Question 7.**
- We shall overload `operator==` as a binary operator (not a class method).
  1. Let the operands be objects u and v.
  2. Then $u == v$ if and only if (i) the x values of u and v are equal and (ii) the y values of u and v are equal.
- There are four versions of `operator==` as follows.

```
(return type) operator== (const PointDbl &u, const PointDbl &v);
(return type) operator== (const PointInt &u, const PointInt &v);
(return type) operator== (const PointDbl &u, const PointInt &v);
(return type) operator== (const PointInt &u, const PointDbl &v);
```

- **Write the function body for each overload above of `operator==`.**
- **State the return type of `operator==` for each case.**
- **Bonus**
  Overload `operator!=` for each case above.

```
(return type) operator!= (const PointDbl &u, const PointDbl &v);
(return type) operator!= (const PointInt &u, const PointInt &v);
(return type) operator!= (const PointDbl &u, const PointInt &v);
(return type) operator!= (const PointInt &u, const PointDbl &v);
```

- *Hint: this should be easy.*

9 Question 9 (bonus)

- **This question uses the class `PointXY` defined in Sec. 1.2 and the class `Polygon` defined in Question 2.**
- **Overload `operator+` to add two `PointXY` objects**

```
PointXY operator+ (const PointXY &u, const PointXY &v);
```

- **Write a function `shift` with the following signature.**

```
Polygon shift(const Polygon &p, const PointXY &s);
```

- If the points in p are v_i, the points in the output object are $v_i + s$.
- Hence if `q = shift(p,s)`, then the polygon q is the polygon p shifted by the point s.
- **Note:**
  1. You may assume the polygon p is not empty.
  2. Do not waste time on validation checks to test if `num <= 0` in p.

10 Question 10 (bonus)

- **This question uses the class `PointXY` defined in Sec. 1.2 and the class `Polygon` defined in Question 2.**
- **Write a function `rotate90` with the following signature.**

```
PointXY rotate90(const PointXY &u);
```

- *Math formula:*
  1. If the coordinates in u are (x, y), the coordinates in the output object are $(-y, x)$.
  2. This is the mathematical operation of rotating a point counterclockwise through $90°$.
- **Write a function `rotate90` with the following signature.**

```
Polygon rotate90(const Polygon &p);
```

  1. If the points in the polygon p are v_i, the points in the output object are `rotate90(v[i])`.
  2. This is the mathematical operation of rotating a polygon counterclockwise through $90°$.

- Hence if `q = rotate90(p)`, then the polygon q is the polygon p rotated counterclockwise through $90°$.
- **Note:**
  1. You may assume the polygon p is not empty.
  2. Do not waste time on validation checks to test if `num <= 0` in p.

2a Question 2a

- Declare a class SNAP ("static, number, array, pointer") as follows.
- The data in class SNAP is static string st, number int n, array int *a, pointer int *p.
- A non-default constructor is written for you.
  1. The pointer p is dynamically allocated to a single object.
  2. The pointer a is dynamically allocated to an array of length n.
  3. You are given $n > 0$.

```
class SNAP {
public:
  SNAP(int x, int m) {                              // given to you
    // you are given that m > 0
    n = m;
    a = new int[n];
    p = new int;
    *p = x;
    for (int i = 0; i < n; ++i)
      a[i] = 0;
  }

  int get_n() const { return n; }                   // given to you

  // get_st
  // set_st

  // default constructor
  // copy constructor
  // assignment operator
  // destructor

private:
  static string st;
  int n;
  int *a;
  int *p;
};
```

- **See next page.**

- **Set `st = "SNAP"` using an appropriate initialization.**
- **Write a default constructor which sets $n = 0$ and the pointers to NULL. (You may write the code inline.)**
- **Write a copy constructor, assignment operator and destructor to perform appropriate copies and management of dynamic memory. (You may write the code inline.)**
- **Write public accessor and mutator methods `get_st`, `set_st` with appropriate signatures.**
  Write the function bodies non-inline.
- *Do not write any other class methods. Assume they are all written for you.*