Classes: A First Look

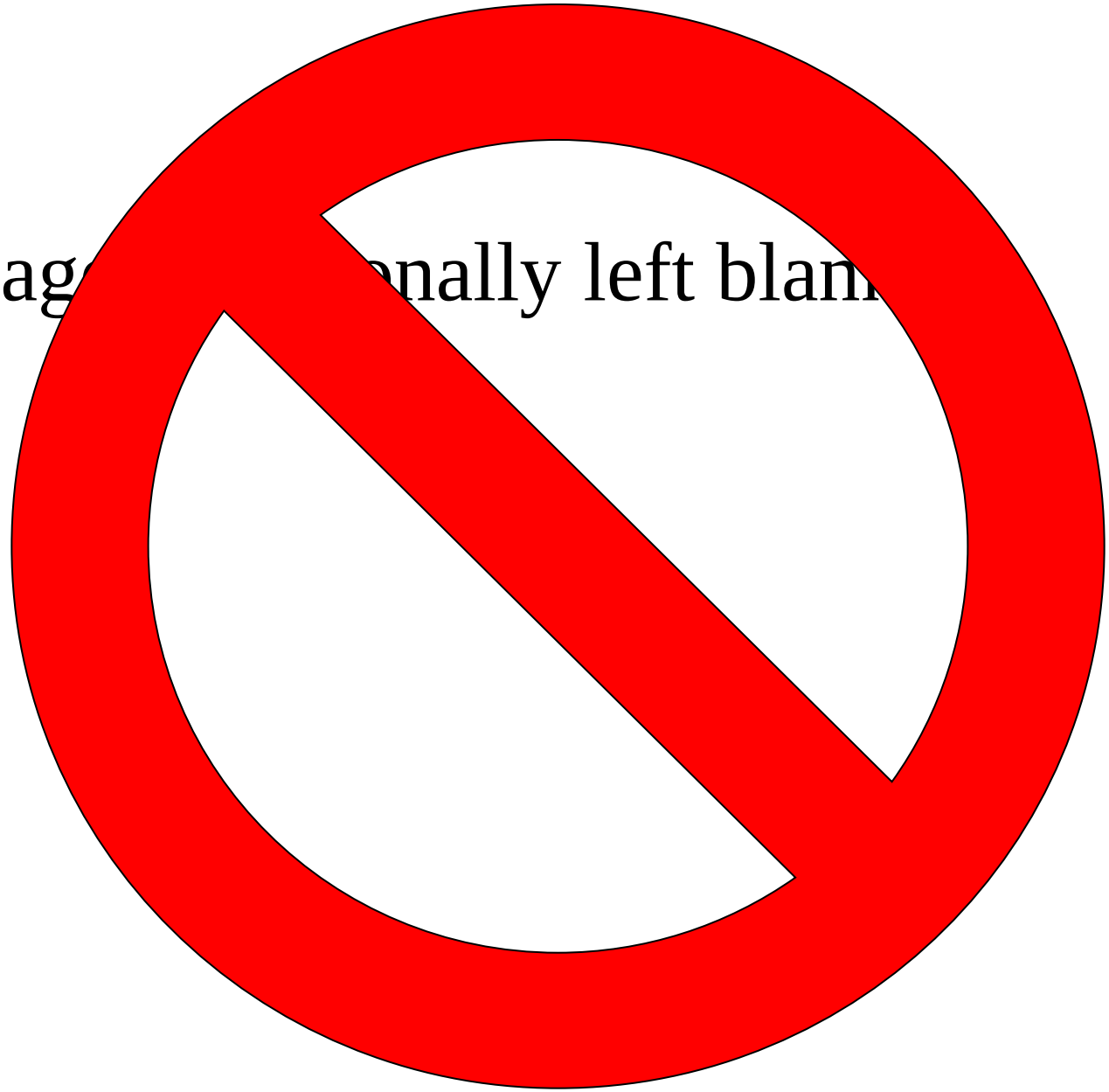
```
#include <iostream.h>

#define SIZE 10

// Declare a stack class for characters
class stack {
    char stck[SIZE]; // holds the stack
    int tos;              // index of top-of-stack

public:
    void init();              // initialize stack
    void push(char ch); // push character on stack
    char pop();               // pop character from stack
}
```

```
// Initialize the stack
void stack::init() { tos = 0; }

// Push a character.
void stack::push(char ch) {
    if (tos==SIZE) { cout << “Stack if full”; return; }
    stck[tos] = ch;
    tos++; }

// Pop a character
char stack::pop() {
    if (tos==0) { cout << “Stack is empty”;
                  return 0; // return null on empty stack
                }
    tos--; return stck[tos]; }
```

```
main() {
    stack s1, s2; // create two stacks
    int i;
    // initialize the stacks
    s1.init();
    s2.init();

    s1.push('a');          s2.push('x');
    s1.push('b');          s2.push('y');
    s1.push('c');          s2.push('z');

    for (i=0; i<3; i++) cout << "Pop s1: " << s1.pop() << "\n";
    for (i=0; i<3; i++) cout << "Pop s2: " << s2.pop() << "\n";

    return 0;
}
```

1. 以後凡是每碰到一個class，都要寫一個 .h和它對應的一個 .cpp。
2. 每個作業因此常會需要寫多個 .h和多個 .cpp。
3. 每次只能寫一個makefile，compile所有的 .cpp (產生它對應的 .o)，並得到最終的a.out。
4. 每次只能寫一個main.cpp，叫做test driver。
5. 批改時，助教會用自己的 (替換掉你的) main.cpp去測試你的作業。

This pag nally left blan



HW #2 (Line & Circle & List & Linked list)

Part A

- Upon completing this assignment, you should be able to implement a simple class, as well as gain a better understanding of the building and use of classes and objects.

- Write the following four classes using C++, or Java or Python if you prefer, called **Line**, **Line2**, **Circle**, and **Circle2**, respectively.

HW #2 (2)

- Definition of class **Line** that represents lines on the real plane:

  private:

  ```
  double x0, y0, x1, y1;
  ```

  // (x0, y0) and (x1, y1) are two distinct points on the line.

- Implement the following (public) member functions:

1. A constructor that initializes this line (default values are x0=0.0, y0=0.0, x1=1.0, and y1=0.0).
2. Constant member functions **get_x0()**, **get_y0()**, **get_x1()**, and **get_y1()** to return individually the x0, y0, x1, and y1.

HW #2 (3)

3. A constant member function **slope()** to return slope of this line.
4. A constant member function **y_intercept()** to return y-intercept.
5. A member function **vshift()** to set the line shifted vertically by given amount.

HW #2 (4)

6. A friend function to output a line to an output stream, overloaded on the operator <<, i.e., outputting line in format “(x0, y0), (x1, y1)”.

```
std::ostream & operator << (std::ostream & os, const Line & l) {
    os << “(“ << l.get_x0() << “, “ << l.get_y0() << “) “;
    os << “(“ << l.get_x1() << “, “ << l.get_y1() << “)”;
    return os;
}
```

HW #2 (5)

7. A friend function to input a line from an input stream, overloaded on the operator >>, i.e., reading x0, y0, x1, y1 from input stream.

```
std::istream & operator >> (std::istream & is, Line & l) {
        double a, b, c, d;
        is >> a >> b >> c >> d;
        l = Line(a, b, c, d);
        return is;
}
```

HW #2 (6)

- A line can have another representation, i.e., definition of class **Line2**:

  private:

  ```
  double x, y, dx, dy;
  // (x, y) is a point on this line; (dx, dy) is the line vector;
  // in other words, slope is written as (run, rise).
  ```

- Re-implement the following (public) member functions:

8. A constructor that takes an initial quadruple of x, y, dx, and dy.

HW #2 (7)

9. A friend function to output a line to an output stream, overloaded on the operator <<.
10. A friend function to input a line from an input stream, overloaded on the operator >>.
11. A member function **normal()** that takes a perpendicular line to this current one and returns true iff successful.

- Following is a sample test program starter for **Line** & **Line2** class.

```
int main() {
    Line l(0, 0, 1, 1);
    Line k;
    Line n(1, 2, 3, 3);

    cout << "slope of l = " << l.slope() << endl;
    cout << "slope of k = " << k.slope() << endl;
    cout << "slope of n = " << n.slope() << endl;

    n.vshift(2.0);

    cout << n << k << l << endl;

    cout << "Enter a line (4 doubles): ";
    Line x;
    cin >> x;

    cout << "Input line is: " << x << endl;
```

```
    cout << "Enter a line2 (4 doubles): ";
    Line2  y;
    cin >> y;

    cout << "Input line2 is: " << y << endl;

    Line2 z;
    y.normal(z);
    cout << “Normal of line2 is: " << z << endl;

    // test Circle and Circle2 here…

    return 0;
}
```

HW #2 (8)

- A circle can be represented by its radius and center:

```
class Circle {
    private:
        double cx, cy, // (cx, cy) is the center of this circle
        radius;        //  radius is the radius of this circle
```

- Implement the following (public) member functions:

1. A constructor that initializes this circle to the unit circle.
2. A constant member function **radius_of()** to return this circle's radius.

HW #2 (9)

3. A member function **set_center()** to set this circle's origin to a given point on the plane (given as two *double* values).
4. A constant member function **is_inside()** to determine if a given point on the plane (given as two *double* values) is inside this circle.
5. A friend function to output a circle to an output stream, overloaded on the operator <<.

HW #2 (10)

- Another representation of a circle by the two end points of a diameter:

class **Circle2** {

private:

double x0, y0, x1, y1;

// the line segment (x0, y0) – (x1, y1) is a diameter of

// this circle.

- Re-implement the following (public) member function:

6. A constant member function to determine whether a given point is inside the circle.

HW #2 (11)

Part B

- Given a **List** class definition (**List.h**), you are required to implement (using **C++**, or **Java** or **Python** if you prefer) its member functions (in red color) in **List.cpp**. (Should you find mistakes in original definitions, fix them.)

```
// List.h
class List {
public:
  List();   // constructor
  ~List(); // destructor
```

HW #2 (12)

```
// inserts at index.
// E.g., insert_at(2, 4.2) into [9.5, 3.0, 0.6, 12.5]
// would produce [9.5, 3.0, 4.2, 0.6, 12.5]
  void insert_at(int index, float value);
  float value_at(int index);
  void remove_at(int index);

private:
  float *dynamic_array;
  int size;          // the number of elements in the array
  int capacity;      // the capacity of the array
  void expand(); // used to expand the array when size==capacity
};                   // To double in size in expansion
```

HW #2 (13)

Part C

- Given a **LnList** class definition (**LnList.h**), you are required to implement its member functions (in red color) in **LnList.cpp**. (Should you find mistakes in original definitions, fix them.)

```
// LnList.h
struct Node {
  int data;
  Node *next;
};
```

HW #2 (14)

```
class LnList {
public:
  LnList();   // constructor
  ~LnList(); // destructor
  void insert(int value);
  bool find(int value);
  bool remove(int value); // returns true if successfully removed
                          // false otherwise
private:
  Node *head;
};
```

- Consider a **List** implemented as a dynamic array (as in **Part B**) and implemented as a linked list (as in **Part C**), what is the fundamental difference in term of layout in memory?

HW #2 (15)

Part D

- Fill in the recursive function **same_tree()** to solve the following problem: Given two binary trees, return true if they are structurally identical – they are made of nodes with the same values arranged in the same way. (Should you find mistakes in original definitions, fix them.)

```
// Node.h
class Node {
  int data;
  Node *left;
  Node *right;
  friend bool same_tree(Node* a, Node* b);
};
```

HW #2 (16)

Part E

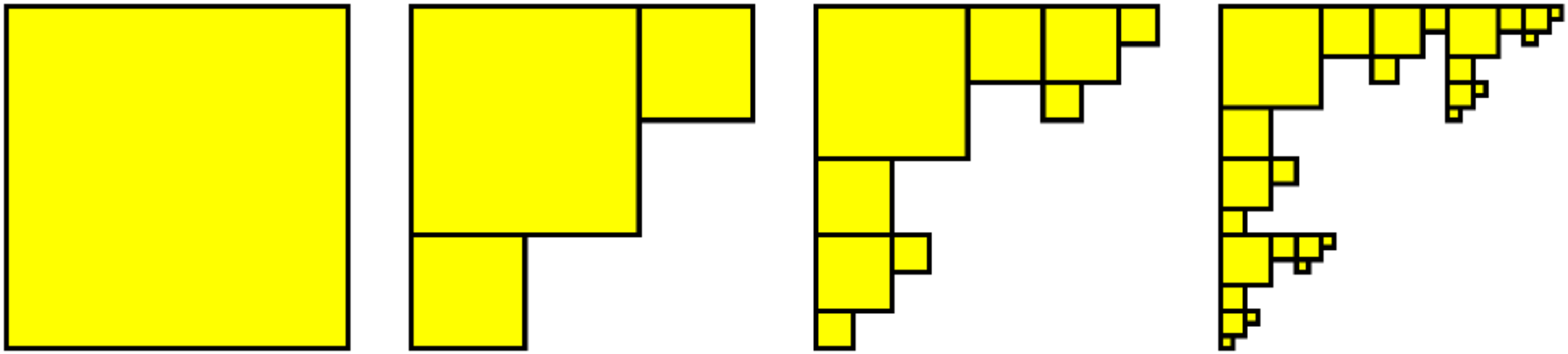
- The Box Trio三件套 fractal is a self-similar pattern that is defined recursively:

- An order-0 Box Trio is a single filled yellow square.

- An order-n Box Trio consists of three Box Trios of order n – 1 arranged in an inverted L shape. The larger middle Box Trio has a side length equal to 2/3 of the original side length. It is flanked側翼 by two smaller Box Trios with a side length equal to 1/3 of the original side length.

HW #2 (17)

- Here are the first few orders of the Box Trio fractal:



HW #2 (18)

- We provide the function **double drawYellowBox (double upperLeft, double length)** that draws a single filled yellow square of size **length** at position **upperLeft**. As a courtesy, **drawYellowBox** returns the area of the box that was drawn. // -- assume already implemented for you --

- Write the function **drawBoxTrio**:

  double drawBoxTrio (double upperLeft, double length, int order) // ---------- to be filled by you ----------

- Here are the specifications for **drawBoxTrio:**

HW #2 (19)

- The three parameters are the upper left corner, the side length, and the order.

- The function draws a Box Trio fractal of **order** with upper left corner positioned at **upperLeft** and side length equal to **length**. You may assume that **order >= 0**.

- The return value from **drawBoxTrio** is the area of the largest yellow box drawn in the fractal. You should not calculate this via a formula but instead use a recursive approach.

HW #2 (20)

- The provided test below confirms the function result for some simple cases: (pt can be any point)

1. drawBoxTrio(pt, 9.0, 0) returns 81.0
2. drawBoxTrio(pt, 9.0, 1) returns 36.0
3. drawBoxTrio(pt, 9.0, 2) and drawBoxTrio(pt, 6.0, 1) both return the same value