Name Beayna Grigorian

CS 32
Winter 2007
Midterm Exam
February 12, 2007
David Smallberg

Problem #	Possible Points	Actual Points
1	10	10
2	15	15
3	30	30
4	15	15
5	30	22
TOTAL	100	92

STUDENT ID #: 603470655

DISCUSSION SECTION: 2A

SIGNATURE: [illegible]

OPEN BOOK, OPEN NOTES
NO ELECTRONIC DEVICES

ENJOY!

1. **[10 points]**

What is the output of the following program?

```
class Light
{
  public:
    Light() { cout << "L "; }
    ~Light() { cout << "~L "; }
};

class Clock
{
  public:
    Clock() { cout << "C "; }
    ~Clock() { cout << "~C "; }
};

class Signal
{
  public:
    Signal() { cout << "S "; }
    ~Signal() { cout << "~S "; }
  private:
    Light m_lights[3];
};

class TrafficSignal : public Signal
{
  public:
    TrafficSignal() { cout << "T "; }
    ~TrafficSignal() { cout << "~T "; }
  private:
    Clock m_clock;
};

int main()
{
  TrafficSignal ts;
  cout << endl;
  cout << "====" << endl;
  TrafficSignal* pts = &ts;
  cout << "====" << endl;
}
```

```
L L L S C T
====
====
~T ~C ~S ~L ~L ~L
```

10/10

2. [15 points in all]

Consider this excerpt from a rather ordinary class representing a Circle:

```
class Circle
{
  public:
    Circle(double x, double y, double r)
     : m_x(x), m_y(y), m_r(r)
    {}
    // other functions not shown
  private:
    double m_x;
    double m_y;
    double m_r;
};
```

A picture is a collection of circles. We choose to represent a picture as a dynamically allocated array of *pointers* to Circles. Here is an excerpt:

```
class Picture
{
  public:
    Picture(int capacity)
     : m_nItems(0), m_capacity(capacity)
    {
        m_items = new Circle*[capacity];
    }
    void addCircle(double r, double x, double y)
    {
        if (m_nItems < m_capacity)
        {
            m_items[m_nItems] = new Circle(x, y, r);
            m_nItems++;
        }
    }
    // other functions not shown
  private:
    Circle** m_items;
    int m_nItems;
    int m_capacity;
};
```

The first `m_nItems` elements of the `m_items` array contain pointers to dynamically allocated circles; the remaining elements have no particular value.

For parts a, b, and c below, you may implement additional helper functions if you like.

a. [5 points]

Complete the implementation of the destructor for the Picture class:

```
Picture::~Picture()
{
    for (int i=0; i< m_nItems; i++)
            delete m_items[i];
     delete [] m_items;

}
```

✓

b. [5 points]

Implement the copy constructor for the Picture class:

```
Picture:: Picture (const Picture& original)
{
     for (int i=0; i< m_nItems; i++)
            delete m_items[i];
     delete [] m_items;
     m_nItems = original. m_nItems;
     m_capacity = original. m_capacity;
     m_items = new Circle*[m_capacity];
     for(int x=0 ; x< m_nItems; x++)
        m_items[x] = original.m_items[x];
}
```

c. [5 points]

Implement the assignment operator for the Picture class:

```
void Picture::swap(Picture& other)
{
    Circle** temp = m_items;
    m_items = other.m_items;
    other.m_items = temp;

    int nItems = m_nItems;
    m_nItems = other.m_nItems;
    other.m_nItems = nItems;

    int cap = m_capacity;
    m_capacity = other.m_capacity;
    other.m_capacity = cap;
}

Picture& Picture::operator=(const Picture& rhs)
{
    if(this != &rhs)
    {
        Picture temp (rhs);
        swap(temp);
    }
    return *this;
}
```

3. [30 points in all]

Consider the following linked list node structure:

```
struct Node
{
    Node(int v, Node* n) : value(v), next(n) {}
    int value;
    Node* next;
};
```

a. [15 points]

Write a function named `deleteNegative` that accepts a pointer to the head of a non-empty singly-linked list of Nodes. It is guaranteed the linked list will have at least one node, and that the first node of the linked list will ***never*** be negative. Your function should remove/free all nodes in the linked list that have a negative value. *You will receive a score of zero on this problem if the body of your deleteNegative function is more than 20 statements long.*

Hint: Don't forget to check your solution to make sure it works when there are multiple consecutive negative values in the linked list.

Here's how your function might be called:

```
int main()
{
    int a[6] = { 1, 2, -3, -1, -2, 3 };
    Node* h = NULL;
    for (int k = 5; k >= 0; k--)
        h = new Node(a[k], h);
    // h points to the first node in a list whose
    // nodes have the values 1 2 -3 -1 -2 3

    deleteNegative(h);
    // At this point, h points to the first node in a
    // list containing the values 1, 2, and 3.  The
    // other nodes containing -3, -1 and -2 should
    // have been unlinked from the list and deleted.
}
```

Write the `deleteNegative` function on the next page.

15/15

Write the deleteNegative function here.

```
void deleteNegative (Node* n)
{
    Node* p = n;
    Node* l = n->next;
    while (l != NULL)
    {
        if ((l->value) < 0)
        {
            Node* x = l;
            l = l->next;
            p->next = l;
            delete x;
            x = NULL;
        }
        else
        {
            l = l->next;
            p = p->next;
        }
    }
}
```

b. **[15 points]**

Write a **recursive** function named `equals`, which accepts two Node pointers, each pointing to a linked list, and returns a bool. The function returns true if the two linked lists have identical sequences of values (i.e., they have the same number of nodes, and corresponding nodes of each list have the same value) and false otherwise. Two empty lists are identical. *You will receive a score of zero on this problem if the body of your compare function is more than 12 statements long or if it contains any occurrence of the keywords* `while`, `for`, *or* `goto`.

Here's how your function might be called:

```
int main()
{
    Node* h1;
    Node* h2;
    … // Assume this omitted code sets h1 to point to
      // the first node in a list whose nodes have the
      // values 1 2 3, and sets h2 to point to another
      // such list.
    if (equals(h1, h2))   // true in this example
        cout << "Same!" << endl;
}
```

Write the `equals` function on the next page.

15/15

Write the equals function here.

```
bool equals (Node* n1, Node* n2)
{
        if ((n1 != NULL && n2==NULL) || (n1 == NULL && n2 != NULL))
                return false;
        if (n1 == NULL && n2 == NULL)
                return true;
        if (n1->value == n2->value)
                return equals (n1->next, n2->next);
        return false;
}
```

4. [15 points]

Write the *first* digit of your UCLA student ID here: 6

Consider the Person structure and the two functions below:

```
struct Person {
    string name;
    int friends[3];   // 3 best friends
    bool asked;
};

void searchSocialNetwork(Person arr[], int start)
{
    queue<int> yetToAsk;
    yetToAsk.push(start);
    while ( ! yetToAsk.empty() )
    {
        int p = yetToAsk.front();
        yetToAsk.pop();
        if ( ! arr[p].asked )
        {
            arr[p].asked = true;
            cout << arr[p].name << endl;
            for (int k = 0; k < 3; k++)   // 3 friends
                yetToAsk.push(arr[p].friends[k]);
        }
    }
}

int main()
{
    Person people[10] = {
        /* 0 */ { "Lucy",    { 3, 5, 2 }, false },
        /* 1 */ { "Ricky",   { 7, 6, 5 }, false },
        /* 2 */ { "Fred",    { 0, 3, 8 }, false },
        /* 3 */ { "Ethel",   { 9, 0, 2 }, false },
        /* 4 */ { "Jerry",   { 6, 5, 8 }, false },
        /* 5 */ { "George",  { 0, 1, 4 }, false },
        /* 6 */ { "Elaine",  { 1, 9, 4 }, false },
        /* 7 */ { "Cosmo",   { 9, 8, 1 }, false },
        /* 8 */ { "Ralph",   { 4, 2, 7 }, false },
        /* 9 */ { "Ed",      { 3, 6, 7 }, false },
     };
     int s;
     cin >> s;   // Enter the first digit of your student ID number
     searchSocialNetwork(people, s);
}
```

If you enter the first digit of your student ID where indicated, what are the first six lines printed by the above program?

Elaine
Ricky
Ed
Jerry
Cosmo
George

15/15

5. [30 points in all]

Consider the following two classes:

```
class GasTank
{
  public:
    GasTank(double initGallons) : m_gallons(initGallons) {}
    void useGas(double gallons) { m_gallons -= gallons; }
    double getNumGallons() { return m_gallons; }
  private:
    double m_gallons;
};

class Battery
{
  public:
    Battery() : m_joules(10) {}
 void ~~bool~~ useEnergy(double joules) { m_joules -= joules;  }
    double getNumJoules() { return m_joules; }
  private:
    double m_joules;
};
```

For the first part of this problem, you are to write a Car class that is intended to be used as a base class. Here are the characteristics of a Car:

- A Car has a GasTank.
- When a Car is constructed, the user may specify how many gallons of gas a Car starts with in its GasTank. If this parameter is omitted during construction, a Car starts with 0 gallons of gas.
- A Car has a `drive` method that accepts a double specifying the number of miles to drive. The function should return the actual number of miles traveled as a double (which may be less than the parameter value if the Car runs out of gas during the trip). Cars can drive 10 miles per gallon of gas.
- A Car has a `cruiseControl` method that takes no arguments and when called, drives exactly 100 miles or as many miles as it can if there is not enough fuel to go 100 miles. It should return the number of miles actually traveled as a double.
- A Car has a `gallonsLeft` method that takes no arguments and returns a double indicating how many gallons are in its gas tank.
- You may declare a destructor, if appropriate.

You may assume (i.e., you do not have to check) that the doubles passed to the constructor and the `drive` method are nonnegative.

a. [15 points]

Declare and implement a `Car` class that provides the functionality above. Avoid needless redundancy in your implementation.

13

```
class Car
{
  public:
      Car(double initGallons = 0.0);
-2 virtual ~Car();
  virtual double drive(double miles);
      double cruiseControl();
      double gallonsLeft() const;
  private:
      GasTank* GT;
}

Car::Car(double initGallons)
{
    GT = new GasTank(initGallons);
}

Car::~Car()
{
    delete GT;
}
virtual double Car::drive(double miles)
{
     if(GT->getNumGallons() >= (miles/10))
     {
         GT->useGas(miles/10);
         return miles;
     }
     else
     {   double numMiles = 10 * GT->getNumGallons();
         GT->useGas(GT->getNumGallons());
         return numMiles;
     }
}
```

```
double Car::cruiseControl()
{
    return drive(100.0);
}

double Car::gallonsLeft() const
{
    return GT-> getNumGallons();
}
```

b. [15 points]

Next, define a class named `HybridCar`. A HybridCar is a kind of Car, but it also has multiple internal batteries. Define your HybridCar class with the following characteristics. (Do not redefine any member functions unless it is absolutely necessary!):

- The user must provide an int specifying how many batteries a HybridCar starts out with when it is constructed. A HybridCar's gas tank starts with 5 gallons of gas for each battery it has. So, for example, a HybridCar with 30 batteries would start out with 150 gallons of gas.
- Every HybridCar has a `drive` method which accepts a double specifying the number of miles to drive. The drive method should try to run entirely off the vehicle's batteries (using their power first); if all battery power runs out before the trip is complete, the car's gas should then be used. A HybridCar can travel 1 mile per Joule of electricity and 10 miles per gallon of gas. The function should return the actual number of miles traveled (which may be less than the parameter value if theHybridCar runs out of battery power and gas during the trip).
- A HybridCar may have a destructor, if necessary.

You may assume (i.e., you do not have to check) that the int passed to the constructor and the double passed to the `drive` method are nonnegative.

On the next page, declare and implement a Hybrid`Car` class that provides the required functionality. Avoid needless redundancy in your implementation.

Your class must be defined so that the following compiles, and when run, prints `Success`:

```
void getOutOfMyDreamsGetIntoMyCar(Car& c)
{
    c.cruiseControl();
}

int main()
{
    HybridCar myPrius(3);
    getOutOfMyDreamsGetIntoMyCar(myPrius);
    if (myPrius.gallonsLeft() == 8)   // not 5
        cout << "Success" << endl;
}
```

Write your declaration and implementation of HybridCar here. Avoid needless redundancy in your implementation.

9

```
class HybridCar: public Car
{
 public:
      HybridCar(int numBatteries);
      ~HybridCar();
      virtual double drive(double drive);
 private:
      Battery** batteries;
      int  m_nB;
}

Hybrid Car:: HybridCar(int numBatteries)
  : Car(5.0 * numBatteries) , m_nB(numBatteries)
{
      batteries = new Battery*[m_nB];
      for(int i=0 ; i<m_nB; i++)
            batteries[i] = new Battery();
}

HybridCar:: ~HybridCar()
{
    for (int i=0, i<m_nB; i++)
          delete batteries[i];
    delete [] batteries;
}

virtual double HybridCar:: drive(double miles)
{
                        Fraction? -2
          if(miles ≤ (10* m_nB))
          {
             for(int i= 0 ; i< miles/10; i++)
                   batteries[i]-> useEnergy(10.0);
             batteries[static_cast<int>(miles/10)]-> useEnergy (miles % 10);
             return miles;
          }

          for (int i=0; i<m_nB; i++)
                batteries[i]->useEnergy(10.0);
          miles = miles-(10*m_nB);          Fractional -2
```

→

```
    if(miles ≤ gallonsLeft())
    {
     return Car::drive(miles) + 10*m_nB;
    }
    else
    {
        miles = miles - Car::drive(10*gallonsLeft());
        return miles;
    }
}
```

some used

-2