Name_____

CS 32
Spring 2011
Midterm Exam
May 4, 2011

Problem #	Possible Points	Actual Points
1	10	6
2	15	3
3	35	(3) (7) (10) 20
4	15	8
5	25	4
TOTAL	100	41

STUDENT ID #: _____

SIGNATURE: _____

OPEN BOOK, OPEN NOTES
NO ELECTRONIC DEVICES

ENJOY !

1. [10 points]

What is the output of the following program?

```
class Icon
{
  public:
    Icon() { cout << "I "; }
    ~Icon() { cout << "~I "; }
};

class File
{
  public:
    File() { cout << "F "; }
    ~File() { cout << "~F "; }
  private:
    Icon m_icons[2];
};

class Rectangle
{
  public:
    Rectangle() { cout << "R "; }
    ~Rectangle() { cout << "~R "; }
};

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

class Picture : public File
{
  public:
    Picture() { cout << "P "; }
    ~Picture() { cout << "~P "; }
  private:
    Caption    m_caption;
    Rectangle* m_boundary;
};

int main()
{
  Picture p;
}
```

I I F C R P ~P ~R ~C ~F ~I ~I

2. [15 points in all]

Here is a Circle class. Every Circle contains a pointer to the Picture it is in:

```
class Picture;

class Circle
{
  public:
    Circle(Picture* pp, double x, double y, double r)
     : m_picture(pp), m_x(x), m_y(y), m_radius(r)
    {}
    void setPicture(Picture* pp) { m_picture = pp; }
    double centerX() const { return m_x; }
    double centerY() const { return m_y; }
    double radius() const  { return m_radius; }
  private:
    Picture* m_picture;
    double   m_x;
    double   m_y;
    double   m_radius;
};
```

Because we did not declare a destructor, copy constructor, or assignment operator for the Circle class, the compiler writes those functions for us.

Here is a Picture class. Every Picture contains a collection of pointers to the dynamically allocated Circles in that Picture:

```
class Picture
{
  public:
    Picture()
     : m_nCircles(0)
    {}
    // other functions not shown
  private:
    Circle* m_Circles[100];
    int     m_nCircles;
};
```

The first `m_nCircles` elements of the `m_Circles` array contain pointers to dynamically allocated Circles; the remaining elements have no particular value.

The users of the Picture class will need to copy Picture objects and assign one Picture object to another.

For parts a, b, and c below, you may implement additional Picture class helper functions if you like. Make no changes or additions to the Circle class.

a. [3 points]

Complete the implementation of the destructor for the Picture class:

```
Picture::~Picture()
{
   for (int i=0; i<m_nCircles; i++)
      delete (m_Circles+i)->Circle;
}
```

-2

b. [6 points]

Implement the copy constructor for the Picture class. (Be careful: Circles must believe that they're in the Picture that they're actually in.)

```
Picture::Picture(const Picture& pic)
   : m_nCircles(pic.m_nCircles)
{
   m_Circles = new Circle[100];

   for (int i=0; i<m_nCircles; i++)
      m_Circles[i] = pic.m_Circles[i];
}
```

-5

c. [6 points]

Implement the assignment operator for the Picture class:

```
Picture &  Picture :: operator = (const Picture& rhs)
{
    if (this != rhs)
    {
        for ( int i=0; i< m_nCircles; i++)
            delete (m_Circles + i) -> Circle;

        m_nCircles = rhs. m_nCircles;

        for (int i=0; i < m_nCircles; i++)
            m_Circles [i] = rhs. m_Circles [i];
    }

    return *this;
}
```

/2

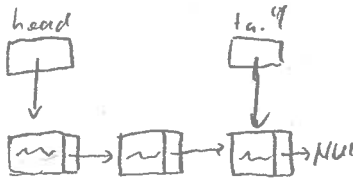
/3

3

3. [35 points in all]

Here is an excerpt from the definition of a singly-linked list class. A LinkedList object represents a singly-linked list of integers. The implementation dynamically allocates the nodes it needs, uses no dummy node, and indicates the end of the list by the last node having NULL as its `m_next` data member. The empty list is represented by `m_head` and `m_tail` being NULL.

```
class LinkedList
{
  public:
    LinkedList();      // creates an empty list
    ...
    void push_back(int v);
    void unique();
    bool dominates(LinkedList& other) const
    { return dom(m_head, other.m_head); }
  private:
    struct Node
    {
        Node(int v, Node* n) : m_value(v), m_next(n) {}
        int   m_value;
        Node* m_next;
    };
    Node* m_head;   // points to first Node in the list
    Node* m_tail;   // points to last Node in the list
    bool dom(const Node* p1, const Node* p2) const;
};
```



For this problem, we will ask you to write some function implementations. Be sure your code is syntactically correct as well as functionally correct, and leaks no memory. Notice that the Node type has no default constructor.

a. [5 points]

The `push_back` function appends to the end of the list a node whose value is `v`. Write an implementation of this function in the space below.

```
void LinkedList::push_back(int v)
{
    Node* temp = new Node;
    temp->m_value = v;
    temp->m_next = NULL;
    tail->m_next = temp;
    tail = temp;
}
```

Empty -1

b. [15 points]

For each group of consecutive nodes with identical values, the `unique` member function deletes all but the first node of that group, leaving the list containing nodes where each node's value is different from the value of the following node. For example, if the LinkedList a contained nodes with values 5 3 3 3 8 2 2 5 5 5 9 7 7 2 2 2, then after the call `a.unique()`, the list a would contain 5 3 8 2 5 9 7 2.

Write an implementation of the `unique` member function in the space below.

```
void LinkedList::unique()
{
    for (Node* pos = m_head; pos->m_value != NULL; pos = pos->m_next)
    {
        if (pos->m_value == pos->m_next->m_value)
        {
            Node* next = pos->m_next;
            pos->m_next = next->m_next;
            delete next;
        }
    }
}
```

-2 -2 -2 -2 tail =

c. [15 points]

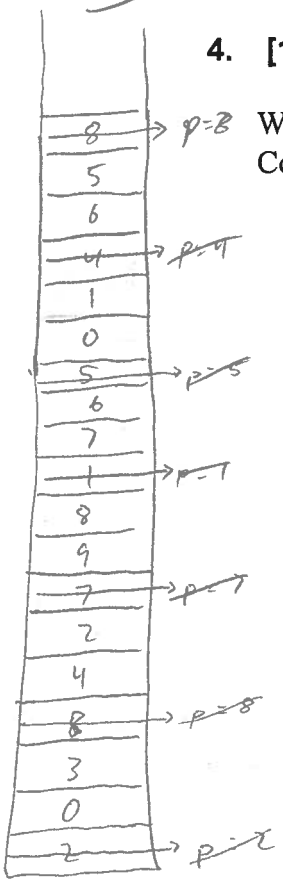
A sequence s *dominates* a sequence t if s has at least as many elements as t, and for all k, the kth element of s is greater than or equal to the kth element of t (if t has a kth element). Every sequence (including the empty sequence) dominates the empty sequence. As examples,

the sequence 2 5 3 5 7		the sequence	2 5 3 5 7
dominates	1 3 3 4 7	does not dominate	1 4 4 4 (because 3 < 4)

Write a **recursive** implementation of the member function named dom, which accepts two Node pointers, each pointing to a linked list of Nodes, and returns a bool. The function returns true if the first linked list dominates the second. *You will receive a score of zero on this problem if the body of your* `dom` *function is more than 15 statements long or if it contains any occurrence of the keywords* `while`, `for`, `goto`, *or* `static`.

Write the dom member function here:

```
bool dom(const Node* p1, const Node* p2) const
{
    if (p2->m_value == NULL)
        return true;
    if (p1->m_value == NULL)
        return false;
    if (p2->m_value < p1->m_value)
        return false;
    return dom(p1->m_next, p2->m_next)
}
```



4. [15 points]

Write the *ninth* digit of your UCLA student ID here: 2

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)
{
    stack<int> yetToAsk;
    yetToAsk.push(start);
    while ( ! yetToAsk.empty() )
    {
        int p = yetToAsk.top();
        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 */ { "Kramer", { 9, 8, 1 }, false },
        /* 8 */ { "Ralph",  { 4, 2, 7 }, false },
        /* 9 */ { "Ed",     { 3, 6, 7 }, false },
    };
    int s;
    cin >> s; // Enter the ninth digit of your student ID number
    searchSocialNetwork(people, s);
}
```

If you enter the ninth digit of your student ID where indicated, what is printed by the above program?

Fred
Ralph
Kramer
Ricky
George
Jerry

4

5. [25 points in all]

a. [12 points]

You have been hired by Wally Wall's Wallet World to create a set of C++ classes for their new electronic wallet device. For the first phase, you are to write a C++ class named *Wallet*. Wally Wall expects that you will eventually be deriving other classes from your Wallet class, so you should design it appropriately. Here is what you know about the Wallet class:

- When a Wallet object is constructed, the user must provide a double representing the initial number of dollars in the wallet. (If the amount specified is negative, treat it as 0.)
- All wallets allow the user to determine whether the wallet has any money in it with an isEmpty() method. This method should return a bool that is true iff the amount of money in the wallet is zero. No derived class will ever determine the emptiness of the wallet in a different manner.
- All wallets have a deposit() method. This method allows the user to add more dollars to the wallet. It accepts a double parameter and adds that number of dollars to the amount already in the wallet. (If the amount is negative, treat it as 0.)
- The amount in a wallet can be determined by a value() method. This method takes no arguments and returns a double, the number of dollars currently in the wallet.

(Notice there's no way to get your money out of the wallet. That's for Release 2.0…)

Using your best style, write the class definition and method implementations for your Wallet. Be sure to use `public`, `private`, and `const` appropriately. Your answer must not contain the word `protected`. (You may continue onto the next page.)

```
class Wallet
{
  public:
    Wallet(double money);
    bool isEmpty(), const;
    virtual void deposit(double amount);
    virtual double value() const;
  private:
    double m_money;

};
```

v-dest -7

no virt -6

(You may continue your answer to part a on this page.)

b. [13 points]

Since you did such a great job building your Wallet class, Wally Wall wants you to build a derived wallet class for U.S. travelers' use in the U.K.: a class named UKWallet. The interface to this class lets users deposit and query pounds sterling instead of dollars. Here is what you know about UKWallet:

1. When a UKWallet object is constructed, the user must specify a conversion factor, the number of dollars per pound. (If this amount is not positive, treat it as 1.65) The initial amount of money in a UKWallet is always 0.
2. The UKWallet has a deposit() method that allows the user to add a number of pounds to the wallet. It accepts a double parameter representing the number of pounds (which should be treated as 0 if negative). All amounts are first converted to dollars before they are stored in the wallet, using the conversion factor supplied at construction time.
3. The UKWallet has a value() method. This method takes no arguments and returns a double, the number of pounds in the wallet, using the conversion factor provided during construction to convert from the stored amount (which is in dollars) to the amount to return (which is in pounds).

Using your best style, write the class definition and method implementations for your UKWallet class. Avoid needless redundancy in your implementation. (You may continue onto the next page.)

(You may continue your answer to part b on this page.)