

Name Henry Cheung

3

CS 32
Spring 2009
Midterm Exam
May 6, 2009
David Smallberg

Problem #	Possible Points	Actual Points
1	10	4 3
2	15	9
3	35	24 27
4	15	15
5	25	14 17
TOTAL	100	66 67 71

STUDENT ID #: 303 690 121

SIGNATURE: Henry Cheung

OPEN BOOK, OPEN NOTES
NO ELECTRONIC DEVICES

ENJOY!

Thank You!

1. [10 points]

What is the output of the following program?

Consider this excerpt from a rather arbitrary class representing a construction worker:

```
class Vehicle
{
public:
    Vehicle() { cout << "V "; }
    ~Vehicle() { cout << "~V "; }
};

class Tire
{
public:
    Tire() { cout << "T "; }
    ~Tire() { cout << "~T "; }
};

class Wheel
{
public:
    Wheel() { cout << "W "; }
    ~Wheel() { cout << "~W "; }
private:
    Tire m_tire;
};

class Motorcycle : public Vehicle
{
public:
    Motorcycle() { cout << "M "; }
    ~Motorcycle() { cout << "~M "; }
private:
    Wheel m_wheels[2];
};

int main()
{
    Motorcycle m;
    cout << endl;
    cout << "====" << endl;
    Motorcycle* pm = &m;
    cout << "====" << endl;
}
```

V M + 2
==== +2
====

2. [15 points in all]

Consider this excerpt from a rather ordinary class representing a construction worker:

```
class Worker
{
public:
    Worker(string nm, string s)
        : m_name(nm), m_skill(s)
    {}
    string name() const { return m_name; }
    string skill() const { return m_skill; }
private:
    string m_name;
    string m_skill;
};
```

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

A construction crew is a collection of workers. We choose to represent a crew as a dynamically allocated array of *pointers* to Workers. Here is an excerpt:

```
class Crew
{
public:
    Crew(int lim)
        : m_size(0), m_maxCrewSize(lim)
    {
        m_crew = new Worker*[lim];
    }
    void hire(string nm, string s)
    {
        if (m_size < m_maxCrewSize)
        {
            m_crew[m_size] = new Worker(nm, s);
            m_size++;
        }
    }
    // other functions not shown
private:
    Worker** m_crew; // pointer to array of pointers
    int m_size;
    int m_maxCrewSize;
};
```

The first `m_size` elements of the `m_crew` array contain pointers to dynamically allocated workers; the remaining elements have no particular value.

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

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

a. [5 points]

Complete the implementation of the destructor for the Crew class:

```
Crew::~Crew()
{
    for(int i=0; i<m_size; i++) {
        delete m_crew[i];
    }
}
```

b. [5 points]

Implement the copy constructor for the Crew class:

```
Crew::Crew(const Crew& original)
: m_size(original.m_size), m_maxCrewSize(original.m_maxCrewSize) {
    m_crew = new Worker* [m_maxCrewSize];
    for(int i=0; i<m_size; i++) {
        m_crew[i] = original.m_crew[i];
    }
}
```



c. [5 points]

Implement the assignment operator for the Crew class:

```

Crew& Crew::operator=(const Crew& other) {
    if (&this != &other) {
        Crew temp(*this);
        m_crew = other.m_crew;
        m_size = other.m_size;
        m_maxCrewSize = other.m_maxCrewSize;
        other.m_crew = temp.m_crew;
        other.m_maxCrewSize = temp.m_maxCrewSize;
        other.m_size = temp.m_size;
    }
    return this;
}

```

For this problem, we will ask you to write some function implementations. Be sure your code is syntactically correct as well as functionally correct. 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* n = new Node(v, NULL);
    m_tail->m_next = n;
    m_tail = n;
}

```

(m_head ok?)
 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 uses no dummy node, and the list end is indicated 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. 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) {
    { Null? } m_tail->m_next = new Node(v, NULL);
    m_tail = m_tail->m_next;
}
```

-(m_head set?

(3)

b. [15 points]

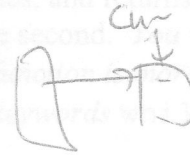
For each group of consecutive nodes with identical values, the unique member function deletes all but the first node of that group, leaving the list containing nodes where each node's value is different from the value of the following node. For example, if the LinkedList a contained nodes with values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

```

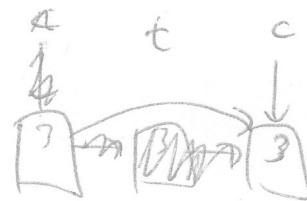
void LinkedList::unique() {
    if (m_head == NULL) return;
    Node* curr = m_head;
    int currValue = m_head->m_value;
    Node* temp;
    while (curr != m_tail) {
        if (curr->m_next->m_value == currValue) {
            temp = curr->m_next;
            curr->m_next = curr->m_next->m_next;
            delete temp;
        }
        else {
            currValue = curr->m_next->m_value;
            curr = curr->m_next;
        }
    }
}

```



might set curr to NULL and not reset temp

-3 m_tail not set



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 k th element of s is greater than or equal to the k th element of t (if t has a k th element). Every sequence (including the empty sequence) dominates the empty sequence. As examples,

the sequence 2 5 3 5 7 dominates 1 3 3 4 7 the sequence 2 5 3 5 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`, or `goto`.*

Write the `dom` member function here:

```
bool dom(const Node* p1, const Node* p2) const {
    if (p2 == NULL) return true; // all lists dominate empty list
    if (p1 == NULL) return false; // list 1 shorter than list 2
    if (p1->m_value < p2->m_value) return false;
    return dom(p1->m_next, p2->m_next);
}
```

15

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

- Rocky
- George
- Jerry
- Ralph
- Kramer
- Ed
- Elaine
- Shei
- Fred

15

4. [15 points]

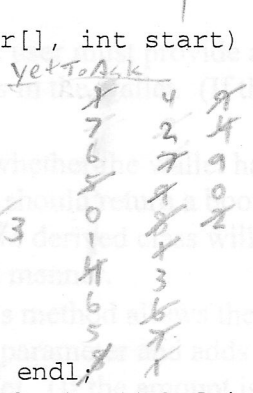
Write the *seventh* digit of your UCLA student ID here: 1
 Consider the Person structure and the two functions below:

```

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 seventh digit of your student ID number
    searchSocialNetwork(people, s);
}
    
```



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

- Ricky
 - George
 - Jerry
 - Ralph
 - Kramer
 - Ed
 - Elaine
 - Ethel
 - Fred
- Lucy

15

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 initial);
    bool isEmpty() const;
    virtual void deposit(double add);
    virtual double value() const;
private:
    double m_value;
};
```

→ virtual destructor

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

b. [13 points]

```
Wallet::Wallet(double initial) {
    if (initial > 0.0) m_value = initial;
    else m_value = 0;
}

bool Wallet::isEmpty() const {
    if (m_value > 0.0) return false;
    return true;
}

void Wallet::deposit(double add) {
    if (add < 0.0) return;
    m_value += add;
}

double Wallet::value() const {
    return m_value;
}

private:
    double m_value;
};
```

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 1.50) 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.)

```
class UKWallet public public Wallet {
public:
    UKWallet(double conversion);
    virtual void deposit(double add);
    virtual double value() const;

private:
    double m_value; X -2
    double m_convert;
};
```

Name Henry Cheung

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

```
UKWallet::UKWallet(double conversion) {
    :m_value(0)
    {
        if (conversion > 0.0) m_convert = conversion;
        else m_convert = 1.50;
    }
}
```

//wrong because m_value
should be private.
use Wallet constructor instead

```
void UKWallet::deposit(double add) {
    if (add < 0.0) return;
    m_value += add/m_convert;
}

double UKWallet::value() {
    return m_value * m_convert;
}
```

Problem #	Possible Points	Actual Points
1	10	10
2	15	15
3	35	35
	15	15
	25	25
TOTAL	100	100

STUDENT ID #: 303 690 121

SIGNATURE: Henry Cheung

OPEN BOOK, OPEN NOTES
NO ELECTRONIC DEVICES

ENJOY!

Thank You!

7