COMP 151: Object-Oriented Programming
Spring Semester 2005
Midterm Examination
Thursday, March 17, 2005
7:30pm – 10:00pm

This is a **CLOSED-BOOK-CLOSED-NOTES** exam consisting of six (6) problems. Follow the instructions carefully. Please write legibly in the **boxes** provided. *Any space outside the boxes is for sketching and will not be graded.* Keep the exam booklet stapled. Write legibly. You may use pencil to answer the questions.

Name: KEY

E-mail: ____

ID: ____

LAB: ____

Problem		Points	Score
1	OBJECT INITIALIZATION	5	
2	MEMORY MANAGEMENT	7	
3	INHERITANCE: I	4	
4	INHERITANCE: II	7	
5	OBJECT-ORIENTED MINI-PHOTOSHOP	18	
6	INHERITANCE: III	29	
Total		70	

1 Object initialization (5 POINTS)

Consider the following class interface and implementation files, `queue.h` and `queue.cpp`, respectively, for a user-defined class `Queue`.

```
// queue.h

#ifndef QUEUE_H
#define QUEUE_H

struct Node;

class Queue
{
public:
    Queue(int size);        // constructor

private:
    int qsize;              // maximum number of items in Queue
    int nitems;             // current number of items in Queue
    Node* head;             // pointer to head of Queue
    Node* tail;             // pointer to tail of Queue
};

#endif


// queue.cpp

#include "queue.h"

struct Node
{
    double item;
    Node* next;
};

Queue::Queue(int size)
{
    qsize = size;
    nitems = 0;
    head = tail = 0;
}
```

Suppose we want to change the private member variable `qsize` of `Queue` to a constant named `QSIZE`.

(a) (3 POINTS) Show all change(s) that you should make to the file(s) for the class to work properly.

Answer:

The following changes should be made:

i. Change “`int qsize`” in `queue.h` to “`const int QSIZE`”.

ii. Change the constructor definition of `Queue::Queue()` to

```
Queue::Queue(int size) : QSIZE(size)
{
    nitems = 0;
    head = tail = 0;
}
```

(b) (2 POINTS) Explain why initialization of the constant `QSIZE` cannot be done inside the class interface in `queue.h`.

Answer:

It does not make sense to initialize the constant value in the class interface because we have to initialize the constant value for each object which can only be done during its creation via a constructor.

2 Memory management (7 POINTS)

Consider the following program:

```
#include <cstring>
using namespace std;

class Employee
{
public:
    Employee(const char* firstname, const char* lastname);
private:
    char* _firstname;
    char* _lastname;
};

Employee::Employee(const char* firstname, const char* lastname)
{
    _firstname = new char[strlen(firstname)+1];
    strcpy(_firstname, firstname);
    _lastname = new char[strlen(lastname)+1];
    strcpy(_lastname, lastname);
}

void process()
{
    Employee* e;                               // (A)
    Employee f("Arthur", "Li");                // (B)
    e = new Employee("Donald", "Tsang");      // (C)
    delete e;                                  // (D)
}

int main()
{
    // code omitted
    process();                                 // (E)
    // code omitted

    return 0;
}
```

(a) (2.5 POINTS) The program shows a bad practice in memory management. Add some code to the class definition to correct the problem.

Answer:
Add a destructor to the public section of Employee and then give the destructor definition as follows:

```
Employee::~Employee() {
    delete [] _firstname;
    delete [] _lastname;
}
```

(b) (4.5 POINTS) Assuming that the change(s) in (a) has/have already been made. Inside `main()` a void function `process()` is called. During the function call, some variables are created and then possibly destroyed later.

For each variable created, indicate the program statement (statements `(A)`–`(E)`) where it is **created** and possibly **destroyed** and the **memory area** (system stack or system heap) from which memory is allocated and deallocated.

For example, a typical answer is "variable X is created at statement `(Y)` by allocating memory from Z and then destroyed at statement `(T)`". Assume that each checkpoint `(A)-(E)` is right after the corresponding statement.

Answer:

- `e` is created at `(A)` by allocating memory from the system stack and destroyed at `(E)`.
- `f` is created at `(B)` by allocating memory for `_firstname` and `_lastname` of `f` from the system heap and the rest of `f` from the system stack, and it is destroyed at `(E)`.
- `Employee`-type object pointed to by `e` is created at `(C)` by allocating memory from the system heap, and it is destroyed at `(D)`.

3 Inheritance: I (4 Points)

Explain the problem of line 3 and 8 in the main function.

```
#include <string>
using namespace std;

enum Department { accounting, business, engineering, mathematics, unknown };
enum Rank { instructor, assistant_prof, associate_prof, professor, dean };

class Person{
public:
    Person(string n = "", string a = "") { name = n; address = a; }

    void set_name(string n) { name = n; }
    void set_address(string a) { address = a; }

protected:
    string name;
    string address;
};

class Student : public Person{
public:
    Student(Department d) { department = d; }

    void set_name(string n) { name = n; }

private:
    Department department;
};

class Teacher : private Person{
public:
    Teacher() {};

    void set_name(string n) { name = n; }

private:
    Rank rank;
};

1  int main(){
2      Person person;
3      Student student;
4      Teacher teacher;
5
6      person.set_address("Person Address");
7      student.set_address("Student Address");
8      teacher.set_address("Teacher Address");
9
10     return 0;
11 };
```

```
Line 3: Since a constructor is defined in the class Student,
        no default constructor is provided by the compiler.
Line 8: Since class Teacher uses private inheritance, the function
        set_address inherited from class Person is changed to private.
        Therefore, the main function cannot access it.
```

4 Inheritance: II (7 POINTS)

Write the output of the following program in the space provided.

```
#include <iostream>
using namespace std;

class Person{
public:
    Person() { cout << "Person default constructor called" << endl; }
    Person(const Person& person) { cout << "Person copy constructor called" << endl; }
};

class Student : public Person{
public:
    Student() { cout << "Student default constructor called" << endl; }
    Student(const Student& person) { cout << "Student copy constructor called" << endl; }
};

class Teacher : public Person{
public:
    Teacher() { cout << "Teacher default constructor called" << endl; }
    Teacher(const Teacher& person) { cout << "Teacher copy constructor called" << endl; }
};

void func(Person person){ }

int main(){
    cout << "Line 1" << endl;
    Teacher teacher;
    cout << "Line 2" << endl;
    Student student;
    cout << "Line 3" << endl;
    Person person = student;
    cout << "Line 4" << endl;
    func(person);
    cout << "Line 5" << endl;
    func(teacher);

    return 0;
}
```

```
Line 1
Person default constructor called
Teacher default constructor called
Line 2
Person default constructor called
Student default constructor called
Line 3
Person copy constructor called
Line 4
Person copy constructor called
Line 5
Person copy constructor called
```

5 Object Oriented Mini-PhotoShop (18 Points)

In this question, you are required to extend the Mini-Photoshop you implemented in programming assignment 1. The following gives part of the header files in the assignment.

```
// --- ImageStruct.h ---
#ifndef _GLOBAL_H_
#define _GLOBAL_H_

typedef unsigned int BMP_DWORD;

...

typedef struct{
    BMP_DWORD red;
    BMP_DWORD green;
    BMP_DWORD blue;
} COLORREF;

#endif

// --- CImage.h ---
#ifndef _CIMAGE_H_
#define _CIMAGE_H_

#include "ImageStruct.h"

using namespace std;

class CImage{
public:
    CImage();                                    // default constructor
    CImage(const CImage& copy);                  // copy constructor
    CImage(const char* strFilePath);             // constructor
    virtual ˜CImage();                           // virtual destructor

    // create a empty image that width = iWidth, height = iHeight and
    // number of channel = iNumChannel
    bool CreateImage(int iWidth, int iHeight, int iNumChannel);

    // retrieve the Color value of point (x, y)
    bool RetrievePixel(int x, int y, COLORREF& color) const;

    // set the color value of point (x, y)
    bool SetPixel(int x, int y, COLORREF& color);

    ...

    // Flip the image upside down, to be implemented
    CImage* CreateInvertImage() const;

    // generate a subimage of the original image, to be implemented
    CImage* CreateSubImage(int x, int y, int iWidth, int iHeight) const;

    // ---------- [ get functions ] ----------
    inline int GetWidth() const;
    inline int GetHeight() const;
    inline int GetNumChannel() const;
    inline unsigned char* GetImageData() const;

private:
    unsigned char* m_ucImage;  // image data in RGB
    int m_iWidth;              // image width
    int m_iHeight;             // image height
    int m_iNumChannel;         // number of channel
};

#endif


// --- Photo.h ---
```

```
#ifndef _PHOTO_H_
#define _PHOTO_H_

#include "CImage.h"

using namespace std;

class Photo{
public:
    Photo();                                                // default constructor
    Photo(const Photo& copy);                               // copy constructor
    Photo(const char* filename, const char* description);  // constructor
    virtual ˜Photo();                                       // virtual destructor

    // --------- [ get functions ] ----------
    inline CImage* GetImage() const;
    inline char* GetFilename() const;
    inline char* GetDescription() const;

private:
    CImage* m_image;      // pointer to an image
    char* m_filename;     // image filename
    char* m_description;  // photo description
};

#endif

// --- PhotoAlbum.h ---

#ifndef _PHOTOALBUM_H_
#define _PHOTOALBUM_H_

#include "Photo.h"

using namespace std;

class PhotoAlbum{
public:
    PhotoAlbum();                       // default constructor
    PhotoAlbum(const char* filename);  // constructor
    virtual ˜PhotoAlbum();             // virtual destructor

    ...

    // add a photo to the list
    bool AddPhoto(const char* filename, const char* description);

    // delete the photo at current position
    bool DeleteCurPhoto();

    // clear all photo album data
    bool Clear();

    // Add all photos in another album to current album, to be implemented
    void Merge(PhotoAlbum& album);

    // ---------- [ get functions ] ----------
    inline int GetNumPhotos() const;
    inline Photo* GetCurPhoto() const;
    inline bool PrevPhoto();
    inline bool NextPhoto();
    inline void ResetCurPtr();

private:
    ...
};

#endif
```

(i) (5 POINTS) Implement the member function CImage::CreateInvertImage().

```
CImage* CImage::CreateInvertImage() const{
    if (m_ucImage == NULL)
        return NULL;

    CImage* pImage = new CImage();
    pImage->CreateImage(m_iWidth, m_iHeight, m_iNumChannel);

    // flip the image
    int i, j;
    for (j = 0; j < m_iHeight; j++){
        for (i = 0; i < m_iWidth; i++){
            COLORREF color;
            this->RetrievePixel(i, j, color);
            pImage->SetPixel(i, m_iHeight-j-1, color);
        }
    }

    return pImage;
}
```

(ii) (6 POINTS) Implement the function CreateSubImage() in the class CImage. You should check that: (1) (x, y) must be a valid position; (2) the subimage must be completely inside the original image. Return NULL if the condition is not satisfied.

```
CImage* CImage::CreateSubImage(int x, int y, int iWidth, int iHeight) const{
    // check if (x, y) is valid
    if (x >= m_iWidth || x < 0)
        return NULL;
    if (y >= m_iHeight || y < 0)
        return NULL;

    // check if the subimage is inside the original image
    if (x + iWidth > m_iWidth)
        return NULL;
    if (y + iHeight > m_iHeight)
        return NULL;

    // create the subimage
    CImage* pImage = new CImage();
    pImage->CreateImage(iWidth, iHeight, m_iNumChannel);

    // copy pixel color
    int i, j;
    for (j = 0; j < iHeight; j++){
        for (i = 0; i < iWidth; i++){
            COLORREF color;
            this->RetrievePixel(x+i, y+j, color);
            pImage->SetPixel(i, j, color);
        }
    }

    return pImage;
}
```

(iii) (7 POINTS) Implement the function Merge() in the class PhotoAlbum.

```
void PhotoAlbum::Merge(PhotoAlbum& album){
    // point to the first photo
    album.ResetCurPtr();

    // add photo one by one
    for (int i = 0; i < album.GetNumPhotos(); i++){
        Photo* newPhoto = album.GetCurPhoto();
        this->AddPhoto(newPhoto->GetFilename(), newPhoto->GetDescription());
        album.NextPhoto();
    }
}
```

6 Inheritance: III (29 POINTS)

Consider the following header file (`account.h`) for a class named `Account` for bank accounts.

```
#ifndef ACCOUNT_H
#define ACCOUNT_H

class Account
{
public:
    Account();                                  // constructor
    double account_balance();                   // return the balance
    double withdraw(double money);              // withdraw from account
    void deposit(double money);                 // deposit into account
    void set_min_balance(double money);         // set minimum balance
private:
    double balance;                             // current balance
    double min_balance;                         // minimum balance
};

#endif
```

It consists of a constructor and four other member functions in the `public` section and two private data members in the `private` section.

(a) (7 POINTS) Write a complete class implementation for `Account` in a file named `account.cpp`. The constructor and member functions should do the following:

- constructor: initialize `balance` and `min_balance` to 0;
- `account_balance`: return the current account balance;
- `withdraw`: if there is enough money in the account, then withdraw the requested amount and return this amount as the function return value; otherwise, deny the withdrawal request by returning 0;
- `deposit`: always accept the deposit;
- `set_min_balance`: set the value of the minimum balance.

You are required to use the member initialization list for initialization if at all possible.

Answer:

```
#include "account.h"

Account::Account() : balance(0.0), min_balance(0.0)
{
}

double Account::account_balance()
{
    return balance;
}

double Account::withdraw(double money)
{
    if (balance - money >= min_balance)  // OK to replace min_balance by 0
    {
        balance -= money;
        return money;
    } else {
        return 0.0;
    }
}

void Account::deposit(double money)
{
    balance += money;
}

void Account::set_min_balance(double money)
{
    min_balance = money;
}
```

(b) (7.5 POINTS + 10 POINTS) Using `Account` as the base class, define a derived class `SavingsAccount` with private data members:

- `account_name`: name of bank account (of `string` type);
- `RATE`: a constant (of `double` type) that represents the interest rate (for simplicity, assumed to be fixed once initialized);
- `accumulated_interest`: accumulated interest (of `double` type) that has not been credited to the account;

as well as constructor and public member functions:

- constructor: initialize `account_name` by a `string`-type parameter with the empty string as default value, `RATE` by a `double`-type parameter with a constant `DEFAULT_RATE` (defined and initialized to 0.0002 in `savingsaccount.h`) as default value, and `accumulated_interest` to 0;
- `name`: return the `string`-type account name;
- `end_of_day`: update the accumulated interest at the end of a day based on the current account balance and interest rate;
- `interest_credit`: update the current account balance by adding the accumulated interest to it.

The class definition should be separated into two files, class interface (`savingsaccount.h`) and class implementation (`savingsaccount.cpp`), as usual. You are required to use the member initialization list for initialization if at all possible.

Answer: `savingsaccount.h`

```
#ifndef SAVINGSACCOUNT_H
#define SAVINGSACCOUNT_H

#include <string>
using namespace std;

#include "account.h"

const double DEFAULT_RATE = 0.0002;           // default interest rate

class SavingsAccount : public Account
{
public:
    SavingsAccount(const string name = "", double rate = DEFAULT_RATE);
                                      // constructor
    string name();                    // return account name
    void end_of_day();                // end-of-day interest accumulation
    void interest_credit();           // add interest to account
private:
    string account_name;              // account name
    const double RATE;                // interest rate
    double accumulated_interest;      // interest gained
};

#endif
```

Answer: `savingsaccount.cpp`

```
#include "savingsaccount.h"

SavingsAccount::SavingsAccount(const string name, double rate)
    : account_name(name), RATE(rate), accumulated_interest(0.0)
{
}

string SavingsAccount::name()
{
    return account_name;
}

void SavingsAccount::end_of_day()
{
    accumulated_interest += account_balance() * RATE;
}

void SavingsAccount::interest_credit()
{
    deposit(accumulated_interest);
    accumulated_interest = 0.0;
}
```

(c) (4.5 POINTS) Assuming that the `Account` and `SavingsAccount` classes have been defined properly in four separate files. The following application program is then written:

```
#include <iostream>
using namespace std;

#include "savingsaccount.h"

int main()
{
    SavingsAccount john("John");
    double obtained;

    john.deposit(1000.0);
    cout << "Account balance of " << john.name() << " = "
         << john.account_balance() << endl;

    obtained = john.withdraw(1500.0);
    cout << "Money tried to withdraw = " << 1500.0 << endl;
    cout << "Money withdrawn = " << obtained << endl;
    cout << "Account balance of " << john.name() << " = "
         << john.account_balance() << endl;

    obtained = john.withdraw(200.0);
    cout << "Money tried to withdraw = " << 200.0 << endl;
    cout << "Money withdrawn = " << obtained << endl;
    cout << "Account balance of " << john.name() << " = "
         << john.account_balance() << endl;

    john.deposit(50.0);
    cout << "Account balance of " << john.name() << " = "
         << john.account_balance() << endl;

    john.end_of_day();
    john.interest_credit();
    cout << "Account balance of " << john.name() << " = "
         << john.account_balance() << endl;

    return 0;
}
```

After proper compilation and linking, give the program output when the program is run.

Answer:

```
Account balance of John = 1000
Money tried to withdraw = 1500
Money withdrawn = 0
Account balance of John = 1000
Money tried to withdraw = 200
Money withdrawn = 200
Account balance of John = 800
Account balance of John = 850
Account balance of John = 850.17
```