Name M. Lewis Poly ID: 0528364

CS1124
Spring 2014
Exam One

NOTE:

1. **Write NEATLY. (please)** If you don't we may not know what you meant and won't be able to give you credit.
2. **DO NOT CHEAT**. (They told me I have to say that.)
3. *Do not* tear any pages out of your Blue Book.
4. *Do not* tear any pages from this document. Be sure that you hand in *all 6 pages* of this test, including this cover sheet.
5. Place your answers for questions **1–8** in this document.
6. Place your answer for **the programming questions 9** in your Blue Book.
7. Put your name and ID number on the cover of your Blue Book.
8. Put your name and ID number as indicated on *each* page of this test. Please circle your last name. Thank you.
9. If you need "scratch" paper, use your Blue Book but cross out anything you do not want graded.
10. You are not required to write comments for any code in this test.
11. Do not begin until you are instructed to do so.
12. **Good Luck!**

Name M. Lewis Poly ID: OS28364

1. [Extra Credit] Who created C++?

a) Gallagher
b) Gosling - Java
c) Kildall - PL/M
d) McCarthy - LISP
e) Ritchie - C

f) Stroustrup ← C++
g) Thompson - Unix
h) van Rossum ~ Python
i) Wirth ~ Pascal
j) Wall ~ Perl

0

Questions 2-5 are based on the following classes:

```
class Pet {
public:
    Pet(string aname) { name = aname; }
    void eat() { cout << "Animal eat!\n"; }
    void move() { cout << "Animal move!\n"; }
    void display() { cout << name << ' ' << age << endl; }
private:
    int age;
    string name;
};

class Cat : public Pet {
public:
    virtual void purr() { cout << "purr!"; }
    virtual void eat() { cout << "Cat eat!"; }
private:
};
```

2. [3 pts] Write a constructor for Cat that ensures proper initialization of the Cat's name. If this is not possible, explain why.

0

Cat (string cname): Pet(cname) {}

Name Malcolm Lewis Poly ID: 0S28364

3. [3 pts] Assuming that you have added any necessary code in the previous question, what will be the result of Line A, below:

Pet doesn't have that method. Compile time error.

```
void makePurr(Pet& aPet) {
    aPet.purr();          // Line B
}

int main() {
    Cat felix("Felix");
    makePurr(felix);      // Line A
}
```

a) Outputs: "purr!"
b) Runtime error -
c) Compiler error in Line A calling makePurr because felix is not a Pet
d) Compiler error in Line A calling makePurr because the parameter should have been a constant reference.
e) Compiler error in Line B because purr is not virtual.
f) None of the above

4. [3 pts] Again, assuming that we have a working constructor for Cat, what will be displayed for Line C below?

```
int main() {
    Cat felix("Felix");
    felix.display();      // Line C.
}
```

a) Outputs: "Felix 0"
b) Outputs: "Felix" and then some random (i.e. garbage) int value.
c) Compilation error because age was not initialized.
d) Runtime error because age was not initialized.
e) None of the above

Name Mr. Lewis

int* P = &Address
int* P =

Poly ID: 0528364

5. [3 pts] Again, assuming that we have a working constructor for Cat, what will be displayed for Line D below?

```
int main() {
    Cat felix("Felix");

    Pet* pPtr = &felix;
    pPtr->eat();          // Line D.
}
```

a) Outputs: "Animal eat!"
b) Outputs: "Cat eat!"
c) Fails to compile
d) Runtime error
e) None of the above

Questions 6-8 are based on the definition of the Account type below. The name field identifies the owner of the account. The withdrawals field contains all of the withdrawals that have occurred. Yes, there should be other fields but we want to keep this reasonably short so you will have time to write the necessary code.

```
struct Account {
    string name;
    vector<int> withdrawals;
};
```

Note this is a struct. The functions that you will be asked to write are **not** methods.

6. [7 pts] Your boss asks you to write a function `displayName` that will be passed just one argument, an Account. (Nice guy he even points out that it should of course be passed by constant reference!) He says the body of the function should contain just **two lines** of code. The first to define a local variable p and store the address of the Account there. The second to print the name field, using the variable p. You think your boss is silly to make these requirements, but you value your job, so you write it as specified.

a) The first line:

int* P = &anAccount; X

const Account * P = & theAccount;

b) The second line:

cout << P->name << endl;

Name Malcolm Lewis Poly ID: 0528364

7. [16 pts] Write a function `readAccounts` that is passed in input file stream and a vector of accounts.

The input file stream is already open – you do not have to check.

Your function will fill the vector from the information in the file. Each input line consists of:

name_of_account num_of_withdrawals first_withdrawal second_withdrawal …

For example, if the following were a line in the file:

```
Fred 3 100 200 300
```

it would be indicating that the account for Fred had 3 withdrawals, the first for 100, the second for 200 and the third for 300.

This function will likely require about seven lines of code, plus whatever lines you use for open/close braces and local variable definitions. Write your answer below:

```
#include <vector>
#include <iostream>
#include <string>
#include <fstream>

void readAccounts (ifstream& accounts, vector<Account>& accountCollection) {
string name; int withdrawals; int withdrawals;
while (accounts >> name >> int) { Account name;
Account.name = name;
Account.withdrawals(int); withdrawals.push_back(int);
accountCollection.push_back(name); } }
```

warning: don't use keywords as variable names

8. [15 pts] Write a function `displayAccounts`, that is passed a vector of Accounts and prints out, for each account, the name of each Account and the total of the withdrawals for the account. This will likely require about six lines of code, plus whatever lines you use for open/close braces.

Oh, your boss tends to make unreasonable coding requirements. In this case he says you are **not** permitted to use the "ranged for", aka the foreach loop. Write your answer below:

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

void displayAccounts (const vector<Account>& accounts) {
for (size_t i; i < accounts.size(); ++i) {
cout << accounts[i].name << " " << withdrawals.size() << endl;
}
}
```

Blue Book

- Place the answers to the following question in your Blue Book.
- **Comments** and **#includes** are **not** required in the blue book!
- **Do not use iterators!!!** (Sorry for the shouting. If you don't know what they are, don't worry. We did not cover them.)

9. [50 pts] In the land of Nyew, people have evolved so that if they want children, they go to a Person store and adopt them. They cannot adopt any Person who already has a parent. Nor can they adopt their own parents or themselves. If you try to adopt someone that you can't, nothing happens.

   It is very important to note that **names are not unique**. Comparing the names of two people does not tell you if they are the same person.

   Your job is simply to write the class Person.

   Below is a sample test program with comments indicating what happens. Lines with the comment starting with "Outputs: " indicate what the display function will output (without the "Output: ")

   So, you need a constructor, an adopt method, a runaway method and a display method, as shown. Pay attention to what arguments are passed to the functions!

```
int main() {
    Person moe("Moe");
    Person larry("Larry");
    Person curly("Curly");
    Person curly2("Curly");

    moe.adopt(larry);
    larry.display();     // Outputs: Person: Larry; Parent: Moe; Children: none.
    moe.adopt(curly);
    moe.adopt(curly2);   // Now we have two children named Curly
    moe.display();       // Outputs: Person: Moe; Parent: none; Children: Larry Curly Curly.
    larry.adopt(moe);    // No effect
    moe.adopt(moe);      // No effect
    moe.adopt(larry);    // No effect
    moe.display();       // Outputs: Person: Moe; Parent: none; Children: Larry Curly Curly.

    larry.runaway();
    larry.adopt(moe);    // Larry is now Moe's parent, not his child
    moe.display();       // Outputs: Person: Moe; Parent: Larry; Children: Curly Curly.
    larry.display();     // Outputs: Person: Larry; Parent: none; Children: Moe.
}
```