Polytechnic Tutoring Center

CS 1124 REVIEW EXAM II Answer Key, Spring 2013

Disclaimer: This mock exam is only for practice. It was made by tutors in the Polytechnic Tutoring Center and is not representative of the actual exam given by the CS Department.

1. For the following questions write whether it will compile or not, and the side effects if there are any.

a) Derived* x = new Base;	**Will not compile**
b) Base* x = new Derived;	**Will compile**
c) Base x; Derived y; x = y;	**Will compile, slicing will occur**
d) Base x; Derived y; y = x;	**Will not compile**

2. Circle all that apply to the following code:

```
int main()
{
	vector<int*> v;
	for(int j = 0; j < 10000; j++){
		for (int i = 0; i < 10; i++)
		{
			v.push_back(new int(i+2));
		}
		v.clear();
		for (int i = 0; i < v.size(); i++)
		{
			delete v[i];
		}
	}
}
```

a) **Run to Completion**
b) Runtime Error
c) Compilation Error
d) Dangling Pointers
e) **Memory Leak**

3. Given the Base and Derived classes below, write a constructor for Derived.

```
class Base {
public:
	Base(int x):baseX(x){}
private:
	int baseX;
};

class Derived : public Base {
public:
private:
	int derivedY;
};
```

Code:

```
Derived(int x, int y): Base(x), derivedY(y) {}
```

4. Write a Cookie class which has a constructor, a getDiameter method, overloaded friend output operator (just displays the diameter), and overloaded comparison (<, ==, !=, >) operators. Cookie contains one member variable: diameter (double). Assume everything you need has already been included.

Code:

```
class Cookie
{
	friend ostream& operator<<(ostream& os, const Cookie& p);
public:
	Cookie(double d) : diameter(d) {}
	double getDiameter() const { return diameter; }
private:
	double diameter;
};

ostream& operator << (ostream& os, const Cookie& c){
	os << c.diameter << endl;
	return os;
}

bool operator == (const Cookie& c1, const Cookie& c2){
	return c1.getDiameter() == c2.getDiameter();
}

bool operator != (const Cookie& c1, const Cookie& c2){
	return !(c1 == c2);
}

bool operator < (const Cookie& c1, const Cookie& c2){
	return c1.getDiameter() < c2.getDiameter();
}

bool operator > (const Cookie& c1, const Cookie& c2){
	return !(c1 < c2 || c1 == c2);
}
```

5. Write a class, CookieJar, which has a vector of Cookie pointers. Next, write an addCookie method which takes a diameter (double), creates a Cookie on the heap, and adds it to the vector. Finally, write the Big 3 for CookieJar.

Code:

```
class CookieJar {
public:
	CookieJar() {}

	void addCookie(double diameter){
		cookies.push_back(new Cookie(diameter));
	}

	~CookieJar(){
		for(size_t i = 0; i < cookies.size(); i++){
			delete cookies[i];
		}
		cookies.clear();
	}

	CookieJar(const CookieJar& cj){
		for(size_t i = 0; i < cj.cookies.size(); i++){
			Cookie * cp = new Cookie(*cj.cookies[i]);
			cookies.push_back(cp);
		}
	}

	CookieJar& operator = (const CookieJar& cj){
		//Self-Assignment?
		if(this != &cj){
			//Delete my data
			for(size_t i = 0; i < cookies.size(); i++){
				delete cookies[i];
			}
			cookies.clear();
			//Copy over new data
			for(size_t i = 0; i < cj.cookies.size(); i++){
				Cookie * cp = new Cookie(*cj.cookies[i]);
				cookies.push_back(cp);
			}
		}
		return *this;
	}

private:
	vector<Cookie*> cookies;
};
```

6. Suppose you wanted to write the addCookie function, from problem 5, outside of the class itself. Write the code to do this.

Inside CookieJar class:

```
void addCookie(double diameter);
```

Outside CookieJar class:

```
void CookieJar::addCookie(double diameter){
	cookies.push_back(new Cookie(diameter));
}
```

7. You are modeling a world of users and computers. Users can buy and own more than one computer, and computers can be Macs or PCs. Users have a work() method, which calls the work() method of all the user's computers. PCs work by printing "*BLUE SCREEN OF DEATH*" and Macs work by printing "*WAITING FOR A YIELD*".

Specifications:

- Write an abstract class Computer.
- Computer only has a work method.
- Write the classes PC and Mac which inherit from Computer.
- Write the User class.
- The User class should have a buy method and a work method.
- Additionally, the User class should have a name attribute and a vector of Computer pointers to all of its CPUs.

Sample Main and Program Execution:

```
int main(){
	PC pc1;
	PC pc2;
	MAC mac1;

	User tom("Tom");
	User and("And");
	User jerry("Jerry");

	tom.buy(pc1);
	jerry.buy(pc2);
	jerry.buy(mac1);

	tom.work();
	and.work();
	jerry.work();
}
```

```
User: Tom
    *BLUE SCREEN OF DEATH*
User: And
User: Jerry
    *BLUE SCREEN OF DEATH*
    *WAITING FOR A YIELD*
Press any key to continue . . .
```

Code:

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


class Computer {
public:
	Computer() {}

	virtual void work() const = 0;
};

class PC : public Computer {
public:
	PC() {}

	void work() const {
		cout << "*BLUE SCREEN OF DEATH*" << endl;
	}
};

class MAC : public Computer {
public:
	MAC() {}

	void work() const {
		cout << "*WAITING FOR A YIELD*" << endl;
	}
};

class User{
public:
	User(const string& userName): name(userName) {}

	void buy(Computer& cp){
		CPUs.push_back(&cp);
	}

	void work() const {
		cout << "User: " << name << endl;
		for(int i = 0; i < CPUs.size(); i++){
			cout << "\t";
			CPUs[i]->work();
		}
	}

private:
	string name;
	vector<Computer *> CPUs;
};
```