1. (15 points) Give the output for the following program.

```
#include <iostream>
#include <string>

class A {
public:
  A(int n) : number(n) {}
  ~A() { std::cout << "deleting A" << std::endl; }
  int getNumber() const { return number; }
  virtual std::string getName() const { return "I'm A"; }
private:
  int number;
};

class B : public A {
public:
  B(int m, int  n) : A(m), number(n) {}
  ~B() { std::cout << "deleting B" << std::endl; }
  int getNumber() const { return number; }
  virtual std::string getName() const { return "I'm B"; }
private:
  int number;
};

int main() {
  A* x = new B(12, 13);
  B* y = new B(66, 77);
  A* z = new B(1, 2);
  std::cout << x->getNumber() << std::endl;
  std::cout << x->getName() << std::endl;
  std::cout << y->getNumber() << std::endl;
  std::cout << y->getName() << std::endl;
  delete z;
}
```

---

2. (5 points) Explain the purpose of the Frame class in the tracker framework? Does the Frame class relate to a design pattern? If so, which one and why?

3. (10 points) Give the output for the following program.

```
#include <string>
#include <vector>
#include <iostream>
class Pokemon {
public:
  Pokemon( ) : name() { std::cout << "default" << std::endl; }
  Pokemon(const std::string& n) : name(n) {
    std::cout << "conversion" << std::endl;
  }
  Pokemon(const Pokemon& p ) : name(p.name) {
    std::cout << "copy" << std::endl;
  }
  Pokemon& operator=(const Pokemon&) {
    std::cout << "copy" << std::endl;
    return *this;
  }
private:
  std::string name;
};
int main() {
  std::vector<Pokemon> pokes;
  pokes.reserve(2);
  pokes.push_back(std::string("Larvitar"));
  pokes.push_back(Pokemon("Steelix"));
  pokes.push_back(Pokemon("Dragonite"));
  std::cout << "size: " << pokes.size() << std::endl;
  std::cout << "capacity: " << pokes.capacity() << std::endl;
}
```

4. (10 points) Convert the following program so that Clock is a *Meyer's* singleton.

```
#include <iostream>

class Clock {
public:
  Clock() : ticks(0) {}
  int getTicks() const { return ticks; }
  void update() { ++ticks; }
private:
  int ticks;
};


int main( ) {
  Clock clock;
  clock.update();
  std::cout << clock.getTicks() << std::endl;
}
```

---

5. (10 points) Convert the following program so that Clock is a *GoF* singleton.

```
#include <iostream>

class Clock {
public:
  Clock() : ticks(0) {}
  int getTicks() const { return ticks; }
  void update() { ++ticks; }
private:
  int ticks;
};


int main( ) {
  Clock clock;
  clock.update();
  std::cout << clock.getTicks() << std::endl;
}
```

6. (10 points) Give the output for the following program.

```
#include <iostream>
class Number {
public:
  Number()               { std::cout << "default" << std::endl;     }
  Number(float)          { std::cout << "convert" << std::endl;     }
  Number(const Number&) { std::cout << "copy" << std::endl;        }
  ~Number()              { std::cout << "destructor" << std::endl; }
  Number& operator=(const Number&) {
    std::cout << "assign" << std::endl;
    return *this;
  }
};
class Student {
public:
  Student(float g) {
    gpa = g;
  }
private:
  Number gpa;
};
int main() {
  Student* npc = new Student(3.4);
  delete npc;
}
```

---

7. (10 points) Function display uses a ranged **for** loop to print the numbers stored in the vector. Modify display so that it uses a **while** loop and a const_iterator to print the numbers.

```
#include <cstdlib>
#include <vector>
#include <iostream>

void display( const std::vector<int>& numbers ) {
  for ( auto n : numbers ) {
    std::cout << n << std::endl;
  }
}
```

```
int main() {
  std::vector<int> numbers;
  numbers.reserve(5);
  for (unsigned int i = 0; i < 5; ++i) {
    numbers.push_back( rand()%100 );
  }
  display( numbers );
}
```

8. (15 points) Write a copy constructor and an assignment operator for class Student.

```
#include <iostream>
#include <cstring>
#include <vector>

class Student : public Person {
public:
  Student() : Person(), study(new char[1]);
  Student(const char* n, const char* m);
  virtual ~Student() { delete [] study; }
  virtual void display() const {
    std::cout << Person::getName() << ", " << study << std::endl;
  }
private:
  char* study;
  Student(const Student& s) = delete;
};

int main( ) {
  std::vector<Person*> people;
  people.push_back(new Student("Peter Quill", "Math"));
  people.push_back(new Person("Peter Parker"));
}
```

9. (10 points) Write function getName and an output operator, operator<< that uses getName, so that Pokemon prints the name of the pokemon. Be attentive to const correctness.

```
#include <cstring>
#include <iostream>

class Pokemon {
public:
  Pokemon(const char* n) : name(new char[strlen(n)+1]) {
    strcpy(name, n);
  }
private:
  char* name;
};

int main() {
  Pokemon poke("Snorlax");
  std::cout << poke << std::endl;
}
```

---

10. (5 points) There is a class, RenderContext, in the tracker framework that has a function to initialize an SDL window. However, initWindow, shown below, has a string constant, "Hello World", as the title of the window. Modify initWindow so that it reads the title from an XML element named title.

```
#include "renderContext.h"

SDL_Window* RenderContext::initWindow( ) {
        window = SDL_CreateWindow( "Hello World", SDL_WINDOWPOS_CENTERED,
            SDL_WINDOWPOS_CENTERED, WIDTH, HEIGHT, SDL_WINDOW_SHOWN );
  if( window == NULL ) {
    throw (std::string("Couldn't make a window: ")+SDL_GetError());
  }
  return window;
}
```