1. (10 points) Give the output for the following program.

```
1  #include <iostream>
2  #include <functional>
3  #include <ctime>
4
5  // [capture clause] (parameters) -> return-type {body}
6
7  int main() {
8    std::function<void(const std::string&, unsigned int)> back;
9    back = [&back](const std::string& s, unsigned int index)->void {
10     if ( index < s.size()-1 ) back(s, index+1);
11     std::cout << s[index];
12   };
13   back("front", 0);
14   std::cout << std::endl;
15 }
```

tnorf

---

2. (10 points) The code listed below illustrates an interface for class Drawable. However, one of the constructors violates a rule for initializing data in a class. What is the violation and how would you fix the constructor?

```
1  #include <iostream>
2  #include <string>
3  #include "vector2f.h"
4
5  class Drawable {
6  public:
7    Drawable(const std::string& n, const Vector2f& pos, const Vector2f& vel):
8      name(n), position(pos), velocity(vel), scale(1.0) {}
9
10   Drawable(const Drawable& s) : name(s.name), position(s.position),
11     velocity(s.velocity), scale(s.scale) { }
12
13   virtual void draw() const = 0;
14   virtual void update(Uint32 ticks) = 0;
15 private:
16   std::string name;
17   Vector2f position;
18   Vector2f velocity;
19   float scale;
20 };
21 #endif
```

The constructor listed on lines #7 and #8 is a conversion constructor, which should receive parameters to initialize all non-static data attributes of the calss. On line #8 scale is initialized to 1.0 rather than a value passed in to the constructor. The fix is to pass a value in for scale.

3. (50 points) For the program below that contains a *list* of pointers to Number:

   (a) Write function display that uses a *ranged for loop* to print the itmes in the list games. Be sure to use the output operator on line #12.

   (b) Write a *function object* to be used to sort the list of games

   (c) Write a *lambda function* to be used to sort the list of games

   (d) Write the code on line #26 to sort the list.

```
1  #include <iostream>
2  #include <list>
3  #include <algorithm>
4
5  class Game {
6  public:
7    Game(int n) : number(n) { }
8    int getNumber() const { return number; }
9  private:
10   int number;
11 };
12 std::ostream& operator<<(std::ostream& out, const Game* g) {
13   return out << g->getNumber();
14 }
15 class GameLess {
16 public:
17   bool operator()(const Game* lhs, const Game* rhs) const {
18     return lhs->getNumber() < rhs->getNumber();
19   }
20 };
21
22 void display( const std::list<Game*>& games ) {
23   for (const Game* g : games) {
24     std::cout << g << ", ";
25   }
26   std::cout << std::endl;
27 }
28
29 void init( std::list<Game*>& games ) {
30   for (int i = 0; i < 10; ++i) {
31     games.push_back( new Game(rand() % 100) );
32   }
33 }
34
35 int main() {
36   std::list<Game*> games;
37   init(games);
38   display(games);
39   games.sort( GameLess() );
40   games.sort(
41    [](const Game* a, const Game* b){return a->getNumber() < b->getNumber();}
42   );
43   display(games);
44 }
```

4. (30 points)

Write a *function object* to be used to search the list, games, for the number generated on line #25. Then print a message that shows if the number was found or not found.

I'm showing two possible solutions here:

```
1  #include <iostream>
2  #include <list>
3  #include <algorithm>
4  #include <ctime>
5
6  class Game {
7  public:
8    Game(int n) : number(n) { }
9    int getNumber() const { return number; }
10 private:
11   int number;
12 };
13 std::ostream& operator<<(std::ostream& out, const Game* g) {
14   return out << g->getNumber();
15 }
16
17 void display( const std::list<Game*>& games ) {
18   for (const Game* g : games) {
19     std::cout << g << ", ";
20   }
21   std::cout << std::endl;
22 }
23
24 void init( std::list<Game*>& games ) {
25   for (int i = 0; i < 20; ++i) {
26     games.push_back( new Game(rand() % 40) );
27   }
28 }
29
30 class GameObj2 {
31 public:
32   GameObj2(const Game& g) : game(g) {}
33   bool operator()(const Game* g) const {
34     return g->getNumber() == game.getNumber();
35   }
36 private:
37   Game game;
38 };
39
40 class GameObj {
41 public:
42   GameObj(int n) : number(n) {}
43   bool operator()(const Game* g) const {
44     return g->getNumber() == number;
45   }
46 private:
47   int number;
48 };
49
50 int main() {
51   srand( time(0) );
52   std::list<Game*> games;
53   init(games);
```

```
54    display(games);
55    int number( rand()%100 );
56    std::cout << number;
57
58    auto itr = find_if(games.begin(), games.end(), GameObj(number));
59    if ( itr == games.end() ) std::cout << " not found" << std::endl;
60    else std::cout << " found." << std::endl;
61
62    itr = find_if(games.begin(), games.end(), GameObj2(Game(number)));
63    if ( itr == games.end() ) std::cout << " not found" << std::endl;
64    else std::cout << " found." << std::endl;
65 }
```