

1. (25 points) Write an output operator for class `Game` and then write a global function, `display`, that accepts the vector of games and uses a *ranged for loop* to display the games. Then, overload a less than operator for class `Game` and add code on line 22 to sort the vector of `Games`. Finally, call the `display` function to display the vector. For example, your output might look like:

15, 21, 35, 49, 77, 83, 86, 86, 92, 93

```
1  #include <iostream>
2  #include <vector>
3  #include <algorithm>
4
5  class Game {
6  public:
7      Game(int n) : number(n) { }
8      int getNumber() const { return number; }
9      bool operator<(const Game& g) const { return number < g.number; }
10 private:
11     int number;
12 };
13 std::ostream& operator<<(std::ostream& out, const Game& game) {
14     return out << game.getNumber();
15 }
16
17 void init( std::vector<Game>& games ) {
18     for (int i = 0; i < 10; ++i) {
19         games.push_back( rand() % 100 );
20     }
21 }
22
23 void display( const std::vector<Game>& games ) {
24     for (const Game& g : games) {
25         std::cout << g << ", ";
26     }
27     std::cout << std::endl;
28 }
29
30 int main() {
31     std::vector<Game> games;
32     init(games);
33     std::sort(games.begin(), games.end());
34     display(games);
35 }
```

2. (25 points) Write a *function object* needed to sort the list of Games in the program below. Then, write the code needed on line 22 to sort the list of Games.

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

(10 points) Write a lamda function to replace the function object used in the previous question.

```
games.sort(
    [](const Game* a, const Game* b)->bool{return (*a) < (*b);}
);
```

(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<int(int)> lie;
9     lie = [&lie](int x) {
10         if(x==1 || x ==2) return 1;
11         else return lie(x-1) + lie(x-2);
12     };
13     std::cout << "lie(5) = " << lie(5) << std::endl;
14 }
```

malloy@aramis: /pubgit/malloymalloy/4160-2018fall/quiz/3/code/fib

lie(5) = 5

(10 points) Write a lamda function, `inrange`, that accepts two parameters and returns a random number between the two parameters, including the end points. For example, your program might generate the following random numbers in the range 1 to 10:

4941531695

```
1 #include <iostream>
2 #include <functional>
3 #include <ctime>
4
5 // [capture clause] (parameters) -> return-type {body}
6
7 int main() {
8     srand( time(0) );
9
10     auto inrange=[](int x, int y)->int{
11         return rand()%(y-x) + x;
12     };
13     for (int i = 0; i < 10; ++i) {
14         std::cout << inrange(1, 10) << " ";
15     }
16     std::cout << std::endl;
17 }
```

(20 points) Class `std::string` contains a function, `find`, that returns the position of the first occurrence of a sequence in a string, and returns `std::npos` if the sequence is not in the string. Give the output for the following program:

```
1  #include <iostream>
2  #include <list>
3  #include <string>
4
5  class Observer {
6  public:
7      void observeEvent(int position) const {
8          std::cout << "duck found at position: " << position << std::endl;
9      }
10 };
11
12 class Subject {
13 public:
14     Subject() : observers() {}
15     void attach(const Observer& o) {
16         observers.push_back( o );
17     }
18     void notify(int position) const {
19         for ( const auto& o : observers ) {
20             o.observeEvent( position );
21         }
22     }
23 private:
24     std::list<Observer> observers;
25 };
26
27 class DuckDetector : public Subject {
28 public:
29     void detect(const std::string& line) const {
30         size_t position = line.find("duck");
31         if ( position != std::string::npos ) {
32             notify(position);
33         }
34     }
35 };
36
37 int main() {
38     DuckDetector duck;
39     Observer x, y;
40     duck.attach(x);
41     duck.attach(y);
42     duck.detect("This sentence has a duck in it");
43     duck.detect("This sentence has a goat in it");
44     duck.detect("No duck here");
45 }
```