1. (50 points) For the following program (you may not use auto):

   (a) (10 pts.) Write an overloaded output operator so that the output statement on line 28 prints the Pokemon's number.

   (b) (10 pts.) Rewrite the ranged for loop, lines 27–29, as a while loop using iterators.

   (c) (20 pts.) Write a function object, NumberLess, so that the sort statement on line 38 sorts the Pokemon by their number from **high to low**.

   (d) (10 pts.) Write a lambda function and rewrite line 38 to use the lambda function so that it sorts the Pokemon from **low to high** by their number.

```
1  #include <iostream>
2  #include <list>
3  #include <cstdlib>
4  #include <algorithm>
5  const int MAX = 20;
6  const int MAX_NUMBER = 100;
7
8  class Pokemon {
9  public:
10   Pokemon() : number(0) { }
11   Pokemon(int n) : number(n) {
12   }
13   Pokemon(const Pokemon& a) : number(a.number) { }
14   int getPokemon() const { return number; }
15   bool operator<(const Pokemon& rhs) const { return number < rhs.number; }
16 private:
17   int number;
18 };
19
20 void init(std::list<Pokemon*> & pokeList) {
21   for (unsigned int i = 0; i < MAX; ++i) {
22     pokeList.push_back( new Pokemon(rand() % MAX_NUMBER) );
23   }
24 }
25
26 void print(const std::list<Pokemon*> & pokeList) {
27   for ( const Pokemon* p : pokeList ) {
28     std::cout << p  << ", ";
29   }
30   std::cout << std::endl;
31 }
32
33
34 int main() {
35   std::list<Pokemon*> pokeList;
36   init(pokeList);
37   print(pokeList);
38   pokeList.sort(PokemonLess());
39   print(pokeList);
40 }
```

```
#include <iostream>
#include <list>
#include <cstdlib>
#include <algorithm>
const int MAX = 20;
const int MAX_NUMBER = 100;

class Pokemon {
public:
  Pokemon() : number(0) { }
  Pokemon(int n) : number(n) {
  }
  Pokemon(const Pokemon& a) : number(a.number) { }
  int getPokemon() const { return number; }
  bool operator<(const Pokemon& rhs) const { return number < rhs.number; }
private:
  int number;
};
std::ostream& operator<<(std::ostream& out, const Pokemon* number) {
  return out << number->getPokemon();
}

class PokemonLess{
public:
  bool operator()(const Pokemon* lhs, const Pokemon* rhs) const {
    return lhs->getPokemon() > rhs->getPokemon();
  }
};


void init(std::list<Pokemon*> & pokeList) {
  for (unsigned int i = 0; i < MAX; ++i) {
    pokeList.push_back( new Pokemon(rand() % MAX_NUMBER) );
  }
}

void print(const std::list<Pokemon*> & pokeList) {
  std::list<Pokemon*>::const_iterator it = pokeList.begin();
  while ( it != pokeList.end() ) {
    std::cout << (*it)  << ", ";
    ++it;
  }
  std::cout << std::endl;
}


int main() {
  std::list<Pokemon*> pokeList;
  init(pokeList);
  print(pokeList);
  pokeList.sort(PokemonLess());
  print(pokeList);
  pokeList.sort(
    [](const Pokemon* a, const Pokemon* b)->bool{return (*a) < (*b);}
  );
  print(pokeList);
}
```

2. (30 points) For the program below (you may use auto):

   (a) There is a ternary operator used on line 10. Write a lambda function, fixSpeed, used on line 17, so that fixSpeed does the same thing that the ternary operator does.

   (b) Write a lambda function, isEven, so that line 26 prints 0 if number is odd, 1 if number is even.

```
1  #include <iostream>
2  #include <functional>
3  #include <ctime>
4  // [capture clause] (parameters) -> return-type {body}
5
6  int main() {
7    srand( time(0) );
8    int speed = rand() % 100;
9
10   speed = speed * (rand()%2?-1:1);
11   std::cout << "speed " << speed << std::endl;
12
13   auto fixSpeed =  [](){ if(rand()%2) return -1; else return 1; };
14   speed = speed * fixSpeed();
15   std::cout << "speed " << speed << std::endl;
16
17   int number = rand() % 100;
18   std::cout << "number " << number << std::endl;
19   auto isEven = [](int x) { return x%2==0; };
20   std::cout << "isEven(number) = " << isEven(number) << std::endl;
21 }
```

3. (20 points) Write an overloaded assignment operator for class Derived.

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

class Base {
};

class Derived : Base {
public:
  Derived() : name(new char[1]) {
    name[0] = '\0';
  }
  Derived(const char* n) : name(new char[strlen(n)+1]) {
    strcpy(name, n);
  }
  Derived& operator=(const Derived& rhs) {
    if ( this == &rhs ) return * this;
    // Unfortunately, this is optional in this particular example
    // But you may need it!
    Base::operator=(rhs);
    delete [] name;
    name = new char[strlen(rhs.name)+1];
    strcpy(name, rhs.name);
    return *this;
  }
  const char* getName() const { return name; }
private:
  char * name;
};

int main() {
  Derived d("bill"), e;
  e = d;
  std::cout << e.getName() << std::endl;
}
```