Consider the following complete program:

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

struct List;

class Link {
public:
    string value;

    Link(const string& v, Link* s = 0)
        : value(v), succ(s) { }

    Link* insert(List& l, Link* n);                          // insert n before this object
    Link* add(Link* n);                                      // insert n after this object
    Link* erase(List& l);                                    // remove this object from the list
    Link* find(const List& l, const string& s);              // find s in list
    const Link* find(const List& l, const string& s) const; // find s in const list

    Link* advance(int n) const; // move n positions in list

    Link* next() const { return succ; }
private:
    Link* succ;
};

struct List {
    List() : first_link(0) { }
    List(Link* l) : first_link(l) { }
    Link* first_link;
};

Link* Link::insert(List& l, Link* n) {
    if (n==0) return this;
    if (this==0) return n;
    n->succ = this;
    if (l.first_link == this) {
        l.first_link = n;
        return n;
    }
    Link* p = l.first_link;
    while (p->succ != this)
        p = p->succ;
    p->succ = n;
    return n;
}

Link* Link::add(Link* n) {
    if (n==0) return this;
    if (this==0) return n;
    n->succ = succ;
    succ = n;
    return n;
}

Link* Link::erase(List& l) {
    if (this==0) return 0;
    if (l.first_link == this) {
        l.first_link = succ;
    }
    Link* p = l.first_link;
    while (p->succ != this)
        p = p->succ;
    p->succ = succ;
```

```
    return succ;
}

Link* Link::find(const List& l, const string& s) {
    Link* p = l.first_link;
    while (p) {
        if (p->value==s) return p;
        p = p->succ;
    }
    return 0;
}

const Link* Link::find(const List& l, const string& s) const {
    const Link* p = l.first_link;
    while (p) {
        if (p->value==s) return p;
        p = p->succ;
    }
    return 0;
}

Link* Link::advance(int n) const {
    if (this==0) return 0;
    Link* p = const_cast<Link*>(this);  // UGLY
    if (0 <= n) {
        while (n--) {
            if (p->succ==0) return 0;
            p = p->succ;
        }
    }
    else cerr << "must advance by a positive number";
    return p;
}

void print_all(const List& l)
{
    cout << "{ ";
    Link* p = l.first_link;
    while (p) {
        cout << p->value;
        if (p=p->next()) cout << ", ";
    }
    cout << " }";
}

int main() {
    Link* swiss_mathematicians = new Link("LEuler");
    List s_mathematicians(swiss_mathematicians);
    swiss_mathematicians = swiss_mathematicians->insert(s_mathematicians, new Link("Lambert"));
    swiss_mathematicians = swiss_mathematicians->insert(s_mathematicians, new Link("JBernoulli"));
    swiss_mathematicians = swiss_mathematicians->insert(s_mathematicians, new Link("DBernoulli"));

    Link* german_mathematicians = new Link("Gauss");
    List g_mathemticians(german_mathematicians);
    german_mathematicians = german_mathematicians->insert(g_mathemticians,new Link("Riemann"));
    german_mathematicians = german_mathematicians->insert(g_mathemticians,new Link("Klein"));
    german_mathematicians = german_mathematicians->insert(g_mathemticians,new Link("Frege"));

    Link* p = german_mathematicians->find(g_mathemticians,"Klein");
    if (p) p->value = "Einstein";

    Link* p2 = swiss_mathematicians->find(s_mathematicians,"Frege");
    if (p2) {
        if (p2==swiss_mathematicians) swiss_mathematicians = p2->next();
        p2->erase(s_mathematicians);
        german_mathematicians->add(p2);
        german_mathematicians = g_mathemticians.first_link;
    }
```

```
133     print_all(s_mathematicians);
        cout << "\n";
135
        print_all(g_mathemticians);
137     cout << "\n";
}
```

1. What is `List` and how does it work here?// ANS: `List` is a `struct` declared on line 4 and defined on lines 26-30 to have a single member variable which is a pointer to a `Link` called `first_link` which is the `nullptr` if the default constructor is called or can be set if the other constructor is called.

   If a `List` is constructed with a pointer to a `Link` as on line 111, then that link will always be the first `Link` in the `List`, though other `Link`s can be `inserted` or `added`.

   We could write a `while` loop to experiment with this like so:

```
1  Link* swiss_mathematicians = new Link("Euler");
   List s_mathematicians(swiss_mathematicians);
3  while(cin>>mathematician) {
           cout << "\nWould you like to insert(i) or add(a) a Swiss mathematician?: ";
5          cin >> choice;
           switch (choice) {
7          case 'a' : {
               swiss_mathematicians = swiss_mathematicians->add(new Link(mathematician));
9              break;
           }
11         case 'i' : {
               swiss_mathematicians = swiss_mathematicians->insert(s_mathematicians, new Link(
       mathematician));
13             break;
           }
15         }
           cout << "\nYou now have " << endl;
17         print_all(s_mathematicians);
           cout << endl;
19     }
```

   This leads to the following instance of interaction and output:

```
 A

Would you like to insert(i) or add(a) a Swiss mathematician?: a

You now have
{ Euler, A }
B

Would you like to insert(i) or add(a) a Swiss mathematician?: i

You now have
{ Euler, B, A }
C

Would you like to insert(i) or add(a) a Swiss mathematician?: a

You now have
```

```
{ Euler, B, C, A }
D

Would you like to insert(i) or add(a) a Swiss mathematician?: i

You now have
{ Euler, B, D, C, A }
E

Would you like to insert(i) or add(a) a Swiss mathematician?: a

You now have
{ Euler, B, D, E, C, A }
F

Would you like to insert(i) or add(a) a Swiss mathematician?: i

You now have
{ Euler, B, D, F, E, C, A }
G

Would you like to insert(i) or add(a) a Swiss mathematician?: a

You now have
{ Euler, B, D, F, G, E, C, A }
```