SOLUTIONS:

1.

a.
```
bool isCyclic(Node* head) {
    if (!head)
        return false;

    Node* slow = head;
    Node* fast = head;
    while(fast->next != NULL && fast->next->next != NULL){
        slow = slow->next;
        fast = fast->next->next;
        if (slow == fast)
            return true;
    }
    return false;
}
```

2.

a.
```
int sumList(Node* head) {
    int sum = 0;
    Node* temp = head;
    while (temp) {
        sum += temp->value;
        temp = temp->next;
    }
    return sum;
}
```

b.
```
int sumList(Node* head) {
    if (head)
        return (head->value + sumList(head->next));
    else {
        return 0;
    }
}
```

3. Mistakes:

1) `this.name` should be `this->name` because `this` is a pointer.

2) Because `ID_number` is const, it should be initialized with an initializer list. The reason for this is that initializing it without an initializer list causes it to be assigned a junk value before reassignment in the constructor body, but the attempt to reassign a const variable produces an error. On the other hand, when an initializer list is used, no junk values are stored in the variable prior to `ID` being assigned to `ID_number`.
3) “const ID_number” appears not to have a type; replace with “const int ID_number”

```
Student(string name, int ID, double gpa): ID_number(ID){
      this->name = name;
      GPA = gpa;
}
```

4.

```
string StudentList::getValedictorianName() const{
Node* n = head;
Node* max = head;     // Node containing student with max gpa
while(n){
      if(n->data.getGPA() > max->data.getGPA()){
            max = n;
      }
      n = n->next;
}
return max->data.getName()
}

double StudentList::getAvgGPA(){
Node*n = head;
double sum = 0;
int count = 0;
while(n){
      sum += n->data.getGPA;
      count++;
      n = n->next;
}
if(count == 0){
      return 0;
}
return sum/count;
}
```

5.

```
OutFit::OutFit(ClothingItem Top, ClothingItem Bottoms){
      tShirt.setBrand(Top.getBrand());
      tShirt.setColor(Top.getColor());
```

```
        tShirt.setSize(Top.getSize());

        Pants.setBrand(Bottom.getBrand());
        Pants.setColor(Bottom.getColor());
        Pants.setSize(Bottom.getSize());
}
```

6.

```
    bool operator==(const OutFit& FirstOutfit,const OutFit& SecondOutfit){
        if((FirstOutfit.tShirt.getBrand()==SecondOutfit.tShirt.getBrand()) &&
        (FirstOutfit.tShirt.getColor()==SecondOutfit.tShirt.getColor()) &&
        (FirstOutfit.tShirt.getSize()==SecondOutfit.tShirt.getSize())&&
        (FirstOutfit.pants.getBrand()==SecondOutfit.pants.getBrand()) &&
        (FirstOutfit.pants.getColor()==SecondOutfit.pants.getColor()) &&
        (FirstOutfit.pants.getSize()==SecondOutfit.pants.getSize())){
                return true;
        }else{
                return false;
        }
}
```

7.

```
void IntList::reverse() {

  Node* itr = first;
  Node* next = NULL;
  Node* prev = NULL;
  while(itr != NULL){
    next = itr->next;
    itr->next = prev;
    prev = itr;
    itr = next;
  }
  first = prev;
}
```

8.

```
void IntList::reverse() {
   if( !first ) return;
   if( !( first->next ) ) return;
   Node *newFirst = reverse( first );
   first->next = NULL;
   first = newFirst;
}

Node* IntList::reverse(Node *node) {
   if( !( node->next->next ) ) {
      node->next->next = node;
      return node->next;
   }
   Node *newFirst = reverse( node->next );
   node->next->next = node;
   return newFirst;
}
```

9.

```
void LinkedList::clear(Node *h){
        if(!h) return;
        clear(h->next);
        delete h;
}
```

10.

The default copy constructor will be called. l2 will just have the head and tail of l1, and we'd end up with both linked lists pointing to the same nodes on the heap.