1.) Advantage of Linked List vs Arrays

- Dynamic size; they can grow and shrink as needed. Eliminates the need to allocate fixed size like arrays
- Inserting and deleting elements can be done in mostly O(1). Arrays require shifting elements, O(n)
- LL allocate memory as needed. Arrays may over allocate memory if resize needed
- LL better for large data. Don't require contigous memory block
- Memory overflow w/ LL can only occur with memory exhaustion. Arrays can overflow if preallocated size exceeded

Advantage of Arrays vs Linked Lists

- Arrays allow O(1) access to elements bc indexing, ideal for frequent access to elems
- Arrays are stored in contigous mem, faster iteration/access
- Arrays are pure data, no extra mem needed for ptrs. LL's require mem for pts.
- Simpler data structure to implement and manage. Require fewer operations vs LL's which involve managing pointers
- Faster element access vs LL's which require traversal to access elems.

2.) a.) ArrayList is backed by a dynamic array.
Accessing an element by index is O(1).
Removing front element requires shifting all other elements one to the left, O(n), n as number of elements in list.
Adding element to end, O(1) normally, O(n) worst case if resize needed.

b.) LinkedList is a doubly linked list.
Removal is O(1), no shifting needed
Adding is O(1) at eol, ref to last node

ArrayList = $O(n^2)$
LinkedList = $O(n)$

3.
```
typedef struct list {
    int item;
    struct list *next;
}list;

void reverse_list(list **head){
    list *prev = nullptr
    list *current = *head
    list *next = nullptr

    while (current != nullptr){
        next = current -> next;
        current -> next = prev;
        prev = current;
        current = next;
    }
    *head = prev;
}
```

4.) a + b * c + (d - e)

Stack = [ ] | PF =

a) a | [ ] | a

b.) + | [+] | a

c) b | [+] | ab

d) * | [+, *] | ab

e) c | [+, *] | abc

f) + | [+] | abc*+

g) (| [+, (] | abc*+

h) d | [+, (] | abc*+d

i) - | [+, (, -] | abc*+d

j) e | [+, (, -] | abc*+de

k)) | [+] | abc*+de-

l) EOF | [ ] | abc*+de-+

Infix

a+b*c+(d-e)

Postfix

abc*+de-+

5.)
```
bool isBalanced(const std::string& expr){
  std::stack<char> s;

  for (char ch : expr){
    if (ch == '(' || ch == '{' || ch == '['){
      s.push(ch);
    }
    else if (ch == ')' || ch == '}' || ch == ']'){
      if (s.empty()){
        return false;
      }
      char top = s.top();
      if((ch == ')' && top == '(') ||
         (ch == '}' && top == '{') ||
         (ch == ']' && top == '[') {
        s.pop();
      }
      else {
        return false;
      }
    }
  }
  return s.empty();
}
```