Homework Solutions #2

P1: It will print

```
5-1hello 13
```

P2: An implementation is:

```
stack_empty(L) {
    if L.head == NIL
        return true
    else return false
}


push(L, x) {
    x.next = L.head
    L.head = x
}


pop(L, x) {
    if (stack_empty(L)) {
        error "underflow"
    }
    else {
        x = L.head
        L.head = L.head.next
        return x
    }
}
```

P3: Here is an example of implementing in C++. In C++, the STL `std::list` is essentially a doubly linked list. Therefore we can write our codes on it.

```cpp
template<typename T>
class Deque {
private:
    std::list<T> lst; // Internal doubly linked list

public:
    // Add an element to the front of the deque
    void pushFront(const T& value) {
        lst.push_front(value);
    }

    // Add an element to the back of the deque
    void pushBack(const T& value) {
        lst.push_back(value);
    }

    // Remove and return an element from the front of the deque
    T popFront() {
        if (lst.empty()) {
            throw std::out_of_range("Deque is empty");
        }
        T value = lst.front();
        lst.pop_front();
        return value;
    }

    // Remove and return an element from the back of the deque
    T popBack() {
        if (lst.empty()) {
            throw std::out_of_range("Deque is empty");
        }
        T value = lst.back();
        lst.pop_back();
        return value;
    }

    // Return the element at the front of the deque without removing it
    T peekFront() const {
        if (lst.empty()) {
            throw std::out_of_range("Deque is empty");
        }
        return lst.front();
    }
};
```

P4: Unless specified, you can either write a procedure or pseudocode in a quiz or exam. Or you can alternatively write real codes. Just make sure your answer is clear so that the TAs and instructor can understand.

Pseudocode:

```
LIST-REVERSE(L)
    // Let's always maintain two pointers current and previous, and reverse the
       list step by step
    prev_pointer = NIL
    curr_pointer = L.head
    while curr_pointer != NIL
        next_pointer = curr_pointer.next
        curr_pointer.next = prev_pointer // Reverse the order
        prev_pointer = curr_pointer // Move one step further
        curr_pointer = next_pointer
    L.head = prev_pointer // After the while loop, the curr_pointer is NIL, and
       the prev_pointer point to the last element in the orignial list
```

You can alternatively write a new program:

```
// function to reverse the linked list
ListNode* reverse(ListNode* head) {
    ListNode* current = head;
    ListNode* prev = nullptr;
    ListNode* next = nullptr;

    while (current != nullptr) {
        next = current->next; // Store next node
        current->next = prev; // Reverse current node's pointer
        prev = current;      // Move pointers one position ahead
        current = next;
    }
    return prev; // New head of the reversed list
}
```