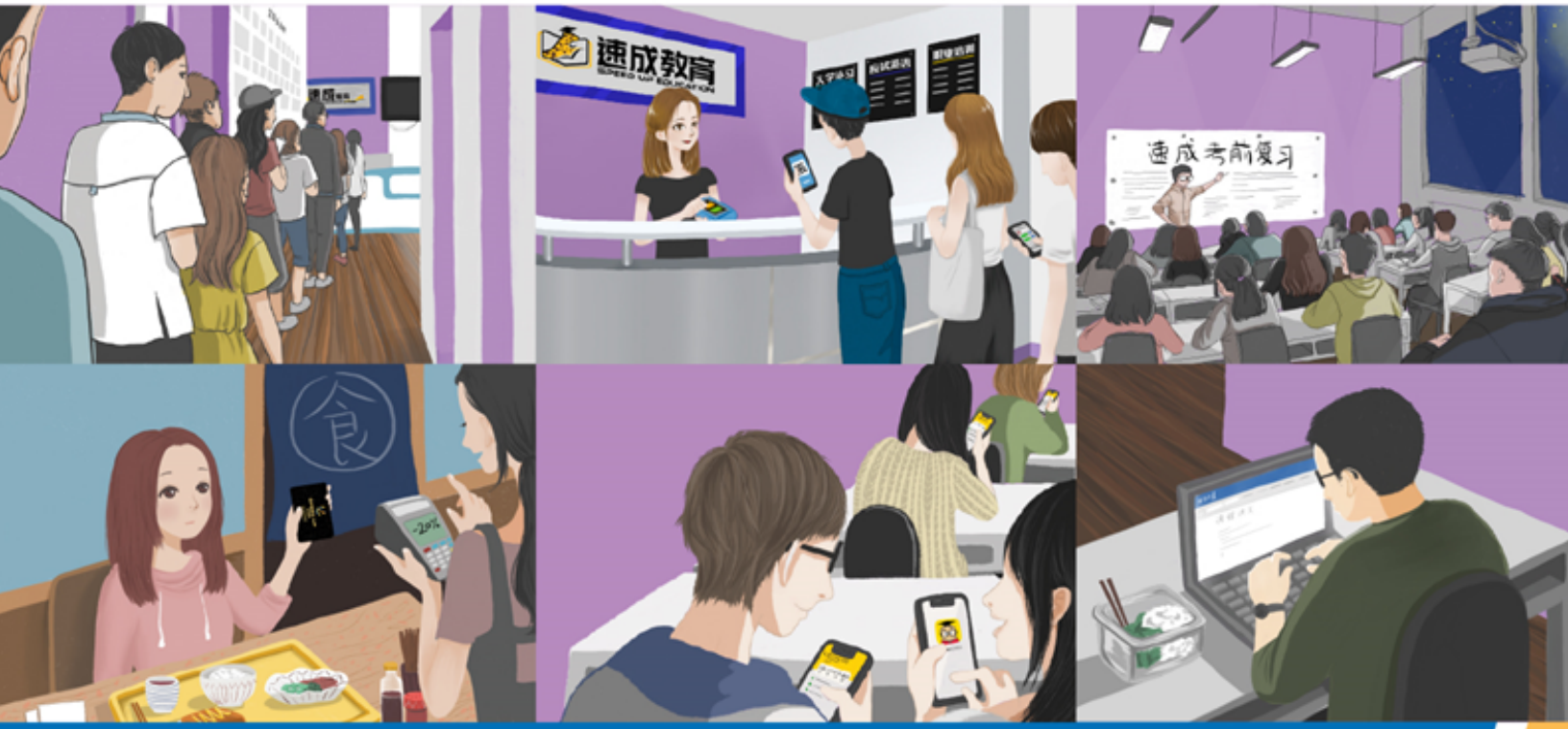
速成教育
SPEED UP EDUCATION



CSCA48

Introduction to Computer Science II

导师： **VC**

UTSC Linked List Practices

```
int detect_cycle(Node *head) {

    if (head == NULL && head->next == NULL)
        return 0;

    Node *slow = head;
    Node *fast = head;
    while (fast && fast->next) {
        slow = slow->next;
        fast = fast->next->next;

        # if fast and catch up slow, there's a cycle
        if (slow == fast)
            return 1;
    }
    return 0;
}



Node *reverse(Node *head) {

    Node *prev = None;
    Node *curr = head;
    Node *next;

    while (curr != NULL) {

        next = curr->next;

        # fix the curr node :)
        curr->next = prev;

        prev = curr;
        curr = next;
    }
    head = prev;

    return head;
}
```

```
Node *remove_duplicates_sorted(Node *head) {

    # empty, do nothing
    if (head == NULL) return;

    # only one item, do nothing
    if (head->next == NULL) return;

    Node *curr = head;
    while (curr != NULL) {
        # if same as the next item
        if (curr->item == curr->next->item)
            # skip the next item
            curr->next = curr->next->next;
        else
            # ONLY move current forward here
            curr = curr->next;
    }
    return head;
}
```

```
Node *merge(Node *head, Node *other) {
    /* Both lists are sorted */
```

```
Node *swap_halves(Node *head) {
    /* Move the nodes in the second half of this list to the front.

    Precondition: head has at least 2 nodes

    For Example:

    > print(head)
    5 -> 10 -> 15 -> 20 -> 25 -> 30
    > Node *new_head = swap_halves(head)
    > print(new_head)
    20 -> 25 -> 30 -> 5 -> 10 -> 15

    */

    // Compute the index of the node that will be the new first node.
    int mid_index = len(self) / 2;        // Assume len() is defined properly

    // Set first_end to refer to the node at the end of the first half

    Node *first_end =

    int pos = 0;
    while (pos < mid_index - 1) {

        first_end =

        pos += 1;
    }

    // Set second_end to refer to the node at the end of the second half

    Node *second_end =

    while (second_end->next != NULL) {

        second_end =

    }
    // Swap the halves

    second_end->          =

    head                  =

    first_end->           =

    return head;
}
```

```
Node *reverse_nodes(Node *head, int i) {
    /* Reverse the nodes at index i and i + 1 by changing their next references
    (not by changing their items).

    Precondition: Both i and i + 1 are valid indexes in the list.

    > print(head)
    5 -> 10 -> 15 -> 20 -> 25 -> 30
    > print(reverse_nodes(head, 1))
    5 -> 15 -> 10 -> 20 -> 25 -> 30

    > print(head)
    5 -> 10 -> 15 -> 20 -> 25 -> 30
    > print(reverse_nodes(head, 0))
    10 -> 5 -> 15 -> 20 -> 25 -> 30

    > print(head)
    5 -> 10 -> 15 -> 20 -> 25 -> 30
    > print(reverse_nodes(head, 4))
    5 -> 10 -> 15 -> 20 -> 30 -> 25
    */

    if (i == 0): // special case of reversing the first two nodes
        Node *temp = head;

        head [                    ] = [                    ]

        temp [                    ] = [                    ]

        head [                    ] = [                    ]

    else { // general case
        // find the node before the pair to reverse

        Node *curr = [                    ]

        // traverse to the node at index i - 1
        for (int unused_ = 0; unused_ < i - 1; unused_++)

            curr = [                    ]

        // reverse the pair of nodes
        temp = curr->next;

        curr [                    ] = [                    ]

        temp [                    ] = [                    ]

        curr [                    ] = [                    ]
    }
    return head
}
```

2019W CSC148 Final Exam Question 7

```
Node *insert_linked_list(Node *head, Node *other, int pos) {
    /* Insert <other> into this linked list immediately before position pos.
    Do not make any new nodes, just link the existing nodes in.
    Preconditions:
        0 <= pos < len(head)
        len(other) >= 1

    > lst1 = LinkedList([0, 1, 2, 3, 4, 5])
    > lst2 = LinkedList([10, 11, 12])
    > lst1 = insert_linked_list(lst1, lst2, 4)
    > print(lst1)
    0 -> 1 -> 2 -> 3 -> 10 -> 11 -> 12 -> 4 -> 5
    > lst3 = LinkedList([99])
    > lst1 = insert_linked_list(lst1, lst3, 0)
    > print(lst1)
    99 -> 0 -> 1 -> 2 -> 3 -> 10 -> 11 -> 12 -> 4 -> 5
    */
```

2020F CSC148 Final Exam q1_a.py

```
Node *insert_nth_a(Nod *head, Node *other, int n) {
    /* Insert one node of <other> after each <nth> node of <head>.
    Do NOT create any new nodes;
    instead, connect nodes from <other> into <self>.

    Precondition: <head> has exactly n times as many nodes as <other>;
                  n >= 1

    > lst = LinkedList([1, 2, 3, 4, 5, 6, 7, 8, 9])
    > lst2 = LinkedList([10, 20, 30])
    > lst = insert_nth(lst, lst2, 3)
    > print(lst)
    1 -> 2 -> 3 -> 10 -> 4 -> 5 -> 6 -> 20 -> 7 -> 8 -> 9 -> 30
    */
```