17-Linked Lists

Instructor: Sridhar Iyer
IIT Bombay
What does this program do?

struct node {    //has two items, as below
    int num;        //the data is an int
    node* next;  //pointer to another node
};

int main() {
    struct node a, b, c;
    struct node* head;

    head = &a;
    a.num = 1; a.next = &b;
    b.num = 2; b.next = &c;
    c.num = 3; c.next = 0;

} //Draw the memory arrangement and their contents
Linked Lists

The structure created by the program is called a Linked List. Its properties are:

- There is a pointer to the head (first node) of the list.
- Each node has some data and a pointer to the next node in the list.
- The pointer in the last node is usually NULL (0).

![Linked List Diagram]
Uses of linked lists

- Items often need to be added or deleted from the “ends” (head or tail).
  - Example: Stack, Queue
- There may be large variation in number of items during program execution
  - Fixed size array may be too small or too large
- Need to insert and delete data at any position
  - Such operations in an array are expensive (Why?)
- Dynamic memory allocation
  - `ptr = new node;` // Creates a new block of memory of size node and assigns its address to `ptr`
  - `delete ptr;` // Gives back the block of memory to OS
Accessing items in a linked list

- The items in a node are accessed using → operator
  - → involves * (dereference) and . (dot)
  - Recall accessing of values in pointers and structs

Example: struct node* head;
- head→num = 5; // sets the value of num in the node pointed to by head, to 5
- cout << head→num; // prints 5
- cout << head→next→num; // prints value in node after head

- Given a pointer to the start of a linked list (head), it is possible to access/modify any node in the list
Traversing a linked list

```cpp
void show(node *head); // prints all the items of a list

node * p = head; // p is a pointer to node, p is initialized to head

while (p != 0) { //iterate till you reach the last node of the list
    cout << "[num:" << p→num << ", next:" << p→next << "] -> ";
    p = p→next; // Move p to the next node in the list
}

cout << "NULL" << endl;
```

Run demo17-linklist.cpp
Inserting after a node

You can get memory for node from OS by `nptr = new node;`

Find the node you want to insert after

*First*, copy the link from the node that's already in the list

*Then*, change the link in the node that's already in the list

See source version for animation
Activity: Think-Pair-Share

Implement a function that inserts an item at the head of a list, and returns a pointer to the new head of list.

```c
node* insert (node *head, int item);
```

**Think (individual):** Write the pseudo-code for insert().

**Pair:** Discuss your pseudo-code with your neighbour. Together, write the C++ code for insert().

**Share:** Compare with demo17-linklist.cpp.
Deleting a node

- To delete the first element, change the link in the header

- To delete some other element, change the link in its predecessor

You can release the memory of deleted node by using `delete nptr;`
Memory of deleted nodes will be reclaimed by OS

See source version for animation
Inserting and deleting in a sorted list

(a) A sorted linked list of integers; b) Insertion; c) Deletion
Commonly used linked list functions

void show(node *head); // prints all the items of a list

node* insert (node *head, int item); // inserts at the head of the list

void append(node *head, int item); // appends an item at the end of the list

node* remove(node* head, int item); // deletes first occurrence of item from the list

int length (node * head); // returns the number of nodes in a list

node* find(node* head, int item); // returns the address of the item

Run demo17-linklist.cpp
Activity: Think-Pair-Share

You have seen the code for show() and insert(). Use the ideas in them to implement append(), a function that inserts an item at the tail (end) of a list.

Think: Write the pseudo-code for append().

Pair: Discuss your pseudo-code with your neighbour. Together, write the C++ code for append().

Share: Compare with demo17-linklist.cpp. + code walk-through of other functions in the file.
In-class Tutorial: Question 1

Show the memory configuration and output of this program:

```c
struct node { int num; node * next; };

int main() {
    node *p, *q, *r; p = new node;  r = p;
    for (int i=0; i<3; i++) { q = new node;
        q→num = i; q→next = 0;
        p→num = i*2;  p→next = q;  p = q;
    } cout << p→num << q→num << r→num;
}
```

Answer: Values output are: 2 2 0
In-class Tutorial: Question 2

Function below to find the smallest 'num' in a linked list, is not working correctly. Identify the bug and fix it.

```c
int findSmallest(node* head) {
    int smallest = head->num;
    node* curr = head->next;
    while (curr != 0) {
        if (curr->num < head->num) smallest = curr->num;
        else curr = curr->next;
    }
    return smallest;
}
```

**Answer:**
This should be `smallest` instead of `head->num`.

Should we keep / remove this else?
In-class Tutorial: Question 3

Write a function that finds a given item in a linked list and returns the address of that item.

Answer:

```c
node* find(node* head, int item) {
    node* curr = head;       // address of the current list node
    while (curr != 0) {       // end not reached
        if (curr->num == item) break;   // item found
        else curr = curr->next;       // move curr to the next node
    }
    return curr;             // If item is not found, curr will be NULL
}
```