CS32: Introduction to Computer Science II

Discussion Week 4

Yichao (Joey)

Jan. 31, 2019

Credit: Carey Nachenberg, Junheng Hao

Announcements

- Homework 2 is due on 11PM Tuesday, February 4
- (from your LA, Matthew) Please fill out this feedback form for me. I know we've only had one discussion together so far, but it will help a lot!

  https://tinyurl.com/StudentsToLAsW20

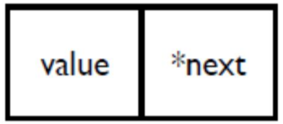
Outline Today

- Linked List
- Stack
- Queue
- Homework 2: Guide

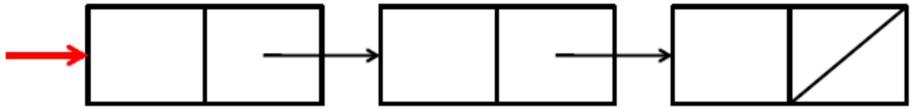
Linked List: Review

Basis

- Minimum Requirement
  - Key component as unit: Node (with `value` and `pointer` to next node)
  - Head pointer → points to the first term
- Regular operations
  - Insertion
  - Search
  - Removal
- Pros and cons
  - Efficient insertion, flexible memory allocation, simple implementation
  - High complexity of search



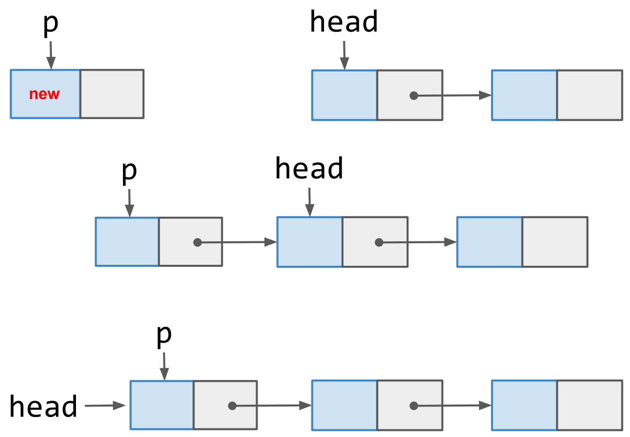
```
typedef int ItemType;
Struct Node
{
        ItemType value;
        Node *next;
};
```



Linked List

Insertion: Add a new node to a list

- Example: Insert as head in a list
- Steps
  a) Create a new node and call the pointer p
  b) Make its `next` pointer point to the first item
  c) Make the head pointer to the new node



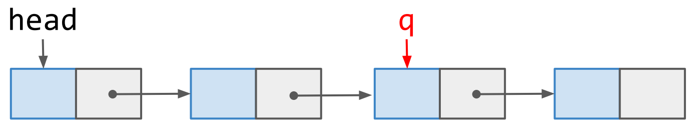
```
//Skeleton: Linked list insertion
//=======================================

//insert as head
p->next = head;
Head = p;

//insert after end: End node: q
q->next = p;
p->next = nullptr;

//insert in the middle: node q
p->next = q->next;
q->next = p;
```

Linked List

Search

UCLA Samueli Computer Science

- Steps
  a) Find matched node and return
  b) If no match, return NULL

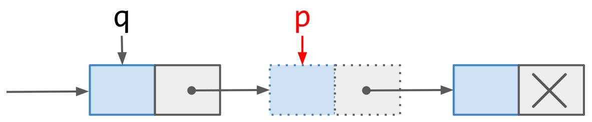


```
// Skeleton Code: Linked list search
// ========================================


Node* Search(int key, Node* head){
  Node *q = head;
  while(q != NULL)
  {
    if(q -> value != key) q = q -> next;
    else return q;
  }
  return NULL;
}
```

Linked List

Removal

- Remember to set the previous node q's `next` pointer to point the next node of p
  `q->next = p->next;`
  `delete p`
- What if `p == head`? What if `p` prints to the last node in the linked list?



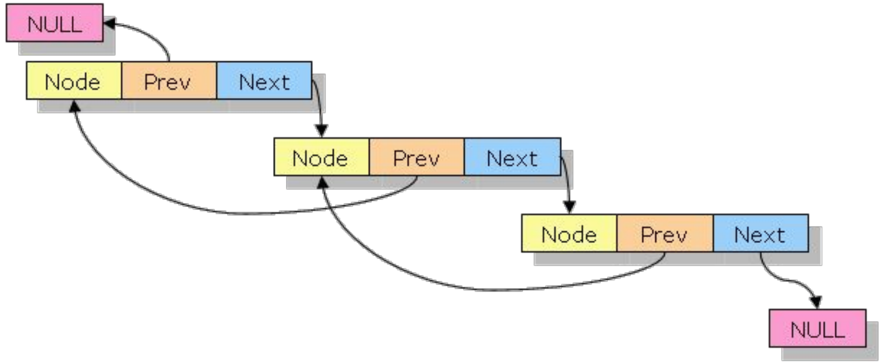
```
// Skeleton Code: Linked list removal
// =====================================

void remove(int valToRemove, Node* head) {
     Node *p = head, *q = NULL;
     while (p != NULL) {
          if (p->value == valToRemove)
               break;
          q = p;
          p = p->next;}
     if (p == NULL) return;
     if (p == head) //special case
          head = p->next;
     else
          q->next = p->next;
     delete p;
}
```

Linked List
Conclusion

- Pros:
    - Efficient insertion (add new data items)
    - Flexible memory allocation
- Cons:
    - Slow search (search is more important than insertion and removal in real situations)
        - e.g. retrieve the fifth value of the list.
        - e.g. a list of values is sorted, find 10 in the linked list
- Many variations
    - Doubly linked lists
    - Sorted linked lists
    - Circularly linked lists

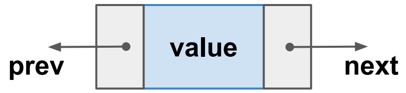
Double Linked List

Data structures and properties



- A linked list where each node has two pointers:
  - Next – pointing to the next node
  - Prev – pointing to the previous node
- Features
  - head, tail pointers
  - `head->prev = NULL; tail->next = NLL;`
  - `head == tail == NULL` when doubly linked list is empty

```
typedef int ItemType;
Struct Node
{
        ItemType value;
        Node *next;
        Node *prev;
};
```



Double Linked List

Insertion: How many cases to consider?

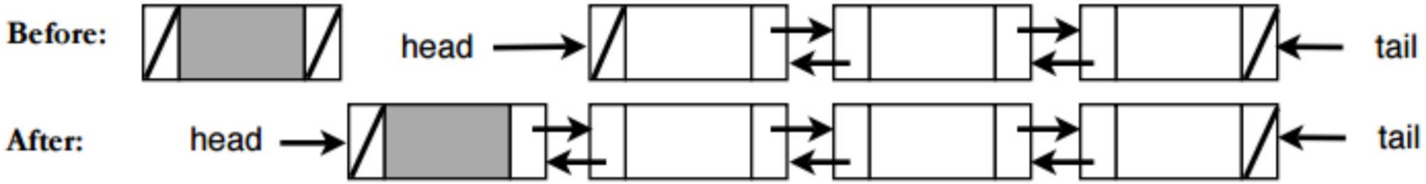
- Four cases:
  - Insert before the head
  - Insert after the tail
  - Insert somewhere in the middle
  - When list is empty ⬅

Double Linked List

Insertion: Before head / After tail

UCLA Samueli Computer Science

- Steps for insertion before head:
  - Set the `prev` of head to the new node p
  - Set the `next` of p to `head`
  - p becomes the new `head`
  - `head->prev = NULL;`
- Steps for insertion after tail:
  - Similar to insertion before head (try it yourself!)



Double Linked List

Insertion: In the middle of the list

- Steps for insertion in the middle (after node q):
  - Fix the next node of q first: `Node *r = q->next;`
  - Point both next of q and prev of r to p: `q->next = r->prev = p;`
  - Point both sides of p to q and r respectively: `p->prev = q; p->next = r;`

- You can do that without the help of pointer `r`

```
p->prev = q;
p->next = q->next;
q->next = q->next->prev = p;
```

Double Linked List

Insertion to empty list / Search

- Insertion to an empty list

```
head = tail = p;
p->next = p->prev = NULL;
```

- Search in doubly linked list
    - Similar to standard linked list
    - Can be done either from head or tail

Double Linked List

Removal

- Removal is more complex!
- Consider the following cases:
  - Check if the node p is the head `(p == head)`. Let this boolean be A.
  - Check if the node p is the tail `(p == tail)`. Let this boolean be B.
- Different cases:
  - Case 1 (A, but not B): P is the head of the list and there is more than one node.
  - Case 2 (B, but not A): P is the tail of the list, and there is more than one node.
  - Case 3 (A and B): P is the only node.
  - Case 4 (not A and not B): P is in the middle of the list.

Double Linked List

Removal

```
void removeNodeInDLL(Node *p, Node& *head, Node& *tail)
{
      if (p == head && p == tail) //case 3
            head = tail = NULL;
      else if (p == head) {
            //case 1
            head = head -> next;
            head -> prev = NULL; }
      else if (p == tail) {
            //case 2
            tail = tail -> prev;
            tail -> next = NULL; }
      else {
            //case 4
            p -> prev -> next = p -> next;
            p -> next -> prev = p -> prev; }
      delete p;
}
```

Double Linked List

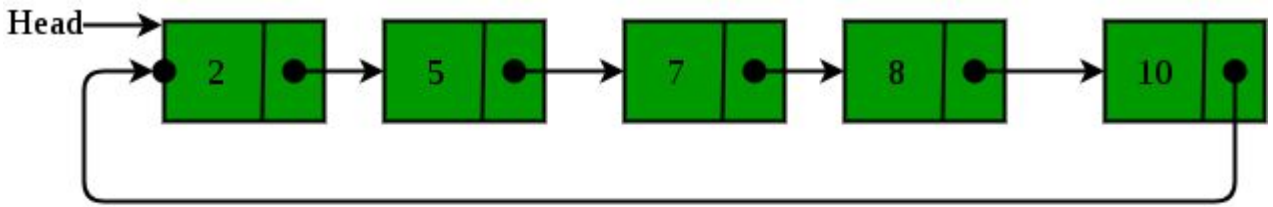
Copy a doubly linked list (and more)

- Steps
  - Create head and tail for the new list
  - Iterate through the old list. For each node, copy its value to a new node.
  - Insert the new node to the tail of the new list.
  - Repeat until we have iterated the entire old list.
  - Set `NULL` before head and next of tail.

- Tips for linked list problems
  - To draw diagrams of nodes and pointers will be extremely helpful.
  - When copying a linked list, only copy stored values to new nodes. Do not copy pointers.
  - You need to check **edge cases**!

Circular Linked List

Motivation and properties



- Linked list where all nodes are connected to form a circle.
  - There is no NULL at the end.
  - Can be a singly circular linked list or doubly circular linked list.
- Pros:
  - Any points can be head (starting point).
  - Implementation for queue.
  - Fit to repeatedly go around the list.

We can maintain a pointer to the last inserted node and front can always be obtained as next of last.

Problem: Reverse Linked List

Leetcode questions #206

UCLA Samueli Computer Science

Question: How to reverse a (single) linked list?

Example:

```
Input: 1->2->3->4->5->NULL
Output: 5->4->3->2->1->NULL
```

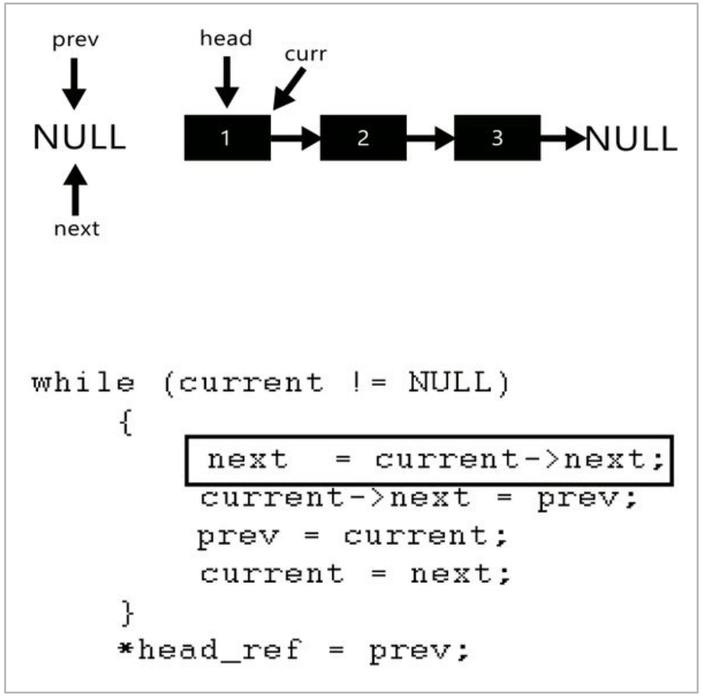
```
// One possible solution
Node* reverseList(struct ListNode* head)
{
     Node *prev=NULL,*current=head,*next;
     while(current) {
          next = current->next;
          current->next = prev;
          prev = current;
          current = next;
     }
     return prev;
}
```

Problem: Reverse Linked List

Leetcode questions #206

UCLA Samueli Computer Science

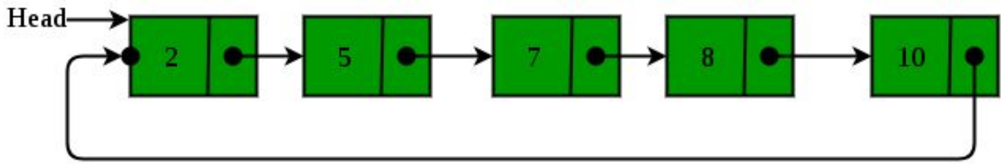
Let's see what happens in these lines of codes! [Link]

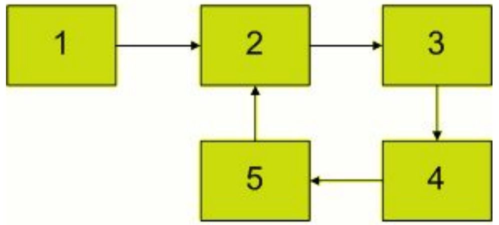


```
while (current != NULL)
    {
        next  = current->next;
        current->next = prev;
        prev = current;
        current = next;
    }
    *head_ref = prev;
```

Circular vs Linked List with Loop

- Two different tasks:
    - Tell whether the linked list is circular
    - Tell whether there is a loop in the linked list (this is much harder!)



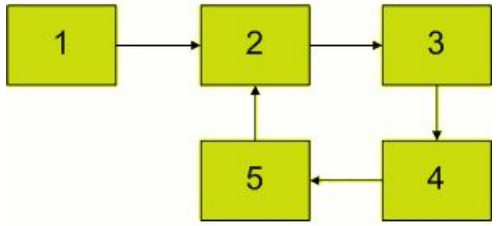


Circular vs Linked List with Loop

UCLA Samueli Computer Science

- Two different tasks:
  - Tell whether the linked list is circular
  - Tell whether there is a loop in the linked list (this is much harder!)

```
/* This function returns true if given linked
   list is circular, else false. */
bool isCircular(struct Node *head)
{
    // An empty linked list is circular
    if (head == NULL)
       return true;

    // Next of head
    struct Node *node = head->next;

    // This loop would stope in both cases (1) If
    // Circular (2) Not circular
    while (node != NULL && node != head)
       node = node->next;

    // If loop stopped because of circular
    // condition
    return (node == head);
}
```



Suggestions on Linked List

- Drawing pictures and carefully tracing through your code, updating the picture with each statement, can help you find bugs in your code.
- Check any list operations for these:
  - In a typical, middle-of-the-list case?
  - At the beginning of the list?
  - At the end of the list?
  - For the empty list?
  - For a one-element list?
- Another validation technique is for every expression of the form **`p->something`**, prove that you are sure that **`p`** has a well-defined, non-null value at that point.

Linked List Problem: Reverse

Leetcode questions #206

UCLA Samueli Computer Science

Question: How to reverse a (single) linked list?

Example:

```
Input: 1->2->3->4->5->NULL
Output: 5->4->3->2->1->NULL
```

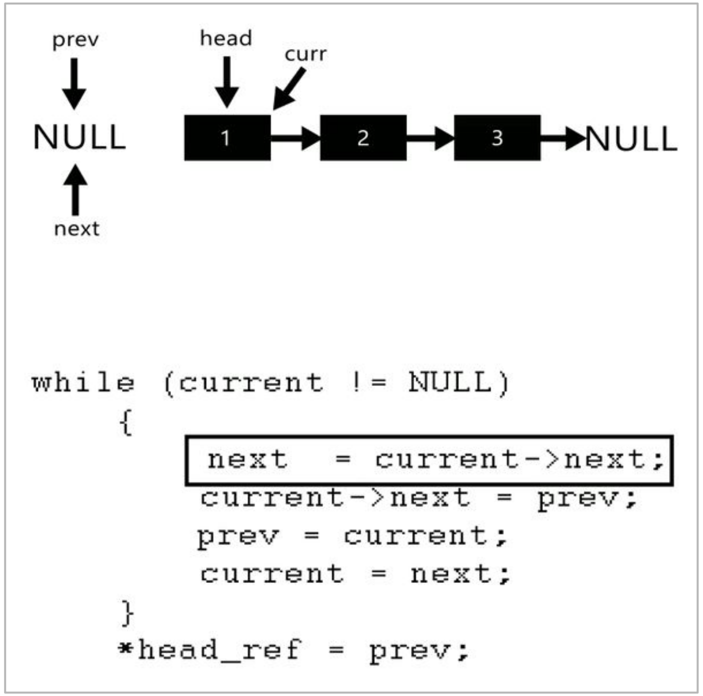
```
// One possible solution
Node* reverseList(struct ListNode* head)
{
     Node *prev=NULL,*current=head,*next;
     while(current) {
          next = current->next;
          current->next = prev;
          prev = current;
          current = next;
     }
     return prev;
}
```

Problem: Reverse Linked List

Leetcode questions #206

UCLA Samueli Computer Science

Let's see what happens in these lines of codes! [Link]



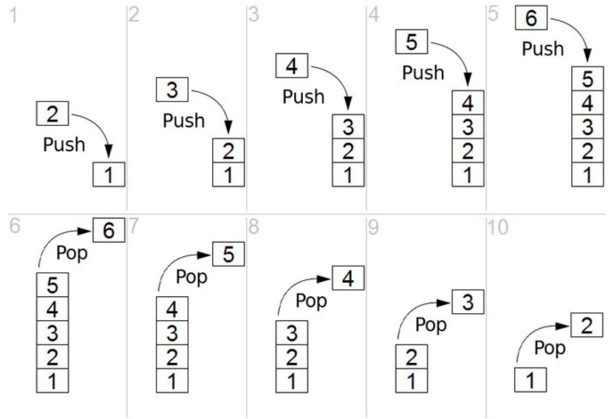
```
while (current != NULL)
    {
        next  = current->next;
        current->next = prev;
        prev = current;
        current = next;
    }
    *head_ref = prev;
```

Stack: FILO

Review: Basics

- FILO: First In, Last Out
- A standard stack implementation
  - `push()` and `pop()`
  - Other methods: `top(), count()`
- Applications:
  - Stack memory: function call
  - Check expressions: matching brackets
  - Depth-first graph search
- Question: How do you implement stack with linked list?

```
class Stack
{
public:
        bool push(const ItemType& item);
        ItemType pop();
        bool empty() const;
        int count() const;
private:
        // some features
};
```



Stack

Implement stack with linked list

- Container: linked list
- Functions:
  - `push()`: Insert node before head.
  - `pop()`: Remove head and return the head value.
  - `top()`: Read head node.
  - `count()`: Maintain a private `int` member.

Stack

Examples

- Given a math expression or text sequence:
  - 6+((5+2)*3-(7+11)*5)*6 → Consider calculation of Reverse Polish Notation (**postfix notation**)
  - Latex: f^{\text{DNN}}(X,\mathbf{W}) = \sigma \left(\mathbf{W} \cdot X + \mathbf{b} \right)

$$f^{\text{DNN}}(X,\mathbf{W}) = \sigma \left(\mathbf{W} \cdot X + \mathbf{b} \right)$$

- How to check the brackets of all types are valid in the sequence? How to calculate expression in Reverse Polish Notation (RPN)?

Regular Expression: 2 + 3 * (5 - 1)

Reverse Polish Notation (RPN): [2] [3] [5] [1] [-] [*] [+]

Stack
Application: Algebraic Expressions

- **Infix Notation**
  - Operators are written in between their operands → X + Y
  - Ambiguous - needs extra rules built in about operator precedence and associativity and parentheses
- **Postfix Notation**
  - Operators are written after their operands → X Y +
  - Operators are evaluated left-to-right. They act on the two nearest values on the left.
- **Prefix Notation**
- **Tasks**
  - Evaluating Postfix Expressions
  - Converting Infix to Postfix Expressions

Stack

Examples

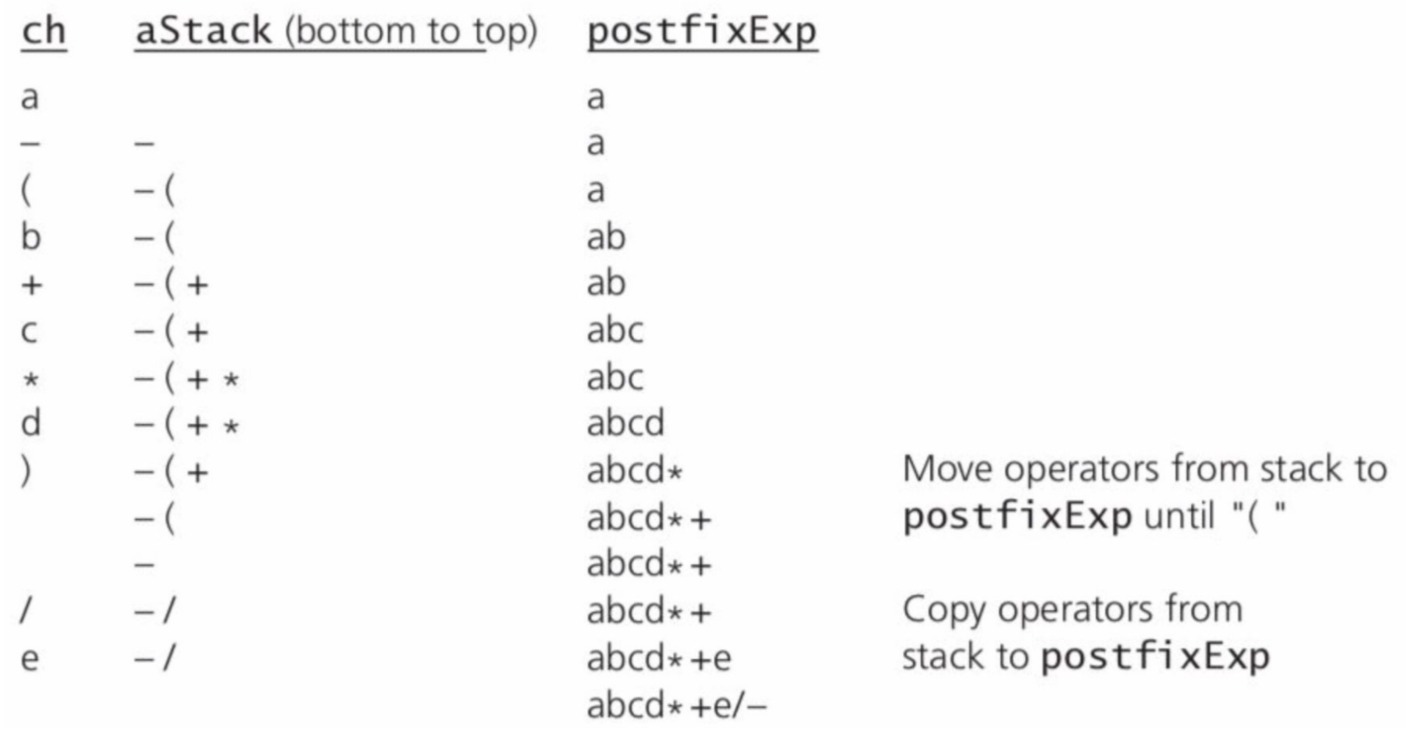
Algorithm
1. Scan the infix expression from left to right.
2. If the scanned character is an operand, output it.
3. Else,
…..**3.1** If the precedence of the scanned operator is greater than the precedence of the operator in the stack(or the stack is empty or the stack contains a ‘(‘), push it.
…..**3.2** Else, Pop all the operators from the stack which are greater than or equal to in precedence than that of the scanned operator. After doing that Push the scanned operator to the stack. (If you encounter parenthesis while popping then stop there and push the scanned operator in the stack.)
4. If the scanned character is an ‘(‘, push it to the stack.
5. If the scanned character is an ‘)’, pop the stack and and output it until a ‘(‘ is encountered, and discard both the parenthesis.
6. Repeat steps 2-6 until infix expression is scanned.
7. Print the output
8. Pop and output from the stack until it is not empty.

Stack

Application: Converting Infix to Postfix Expressions

UCLA Samueli Computer Science

Infix expression: `a - ( b + c * d ) / e`



ch	aStack (bottom to top)	postfixExp	
a		a	
–	–	a	
(	– (	a	
b	– (	ab	
+	– (+	ab	
c	– (+	abc	
*	– (+ *	abc	
d	– (+ *	abcd	
)	– (+	abcd*	Move operators from stack to
	– (	abcd*+	postfixExp until "("
	–	abcd*+	
/	– /	abcd*+	Copy operators from
e	– /	abcd*+e	stack to postfixExp
		abcd*+e/–	

Stack

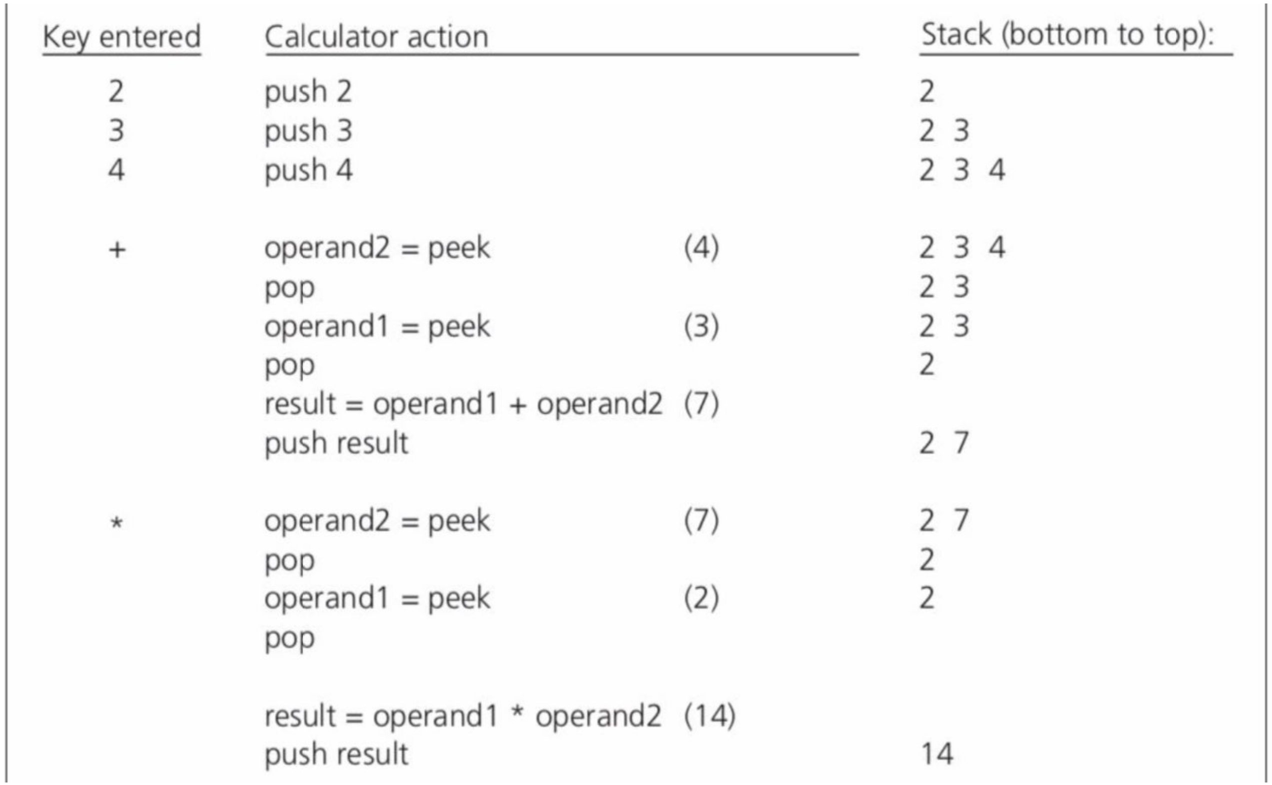
Application: Evaluating Postfix Expressions

Postfix Expression

2 3 4 + *

Infix Expression

2 * (3 + 4)

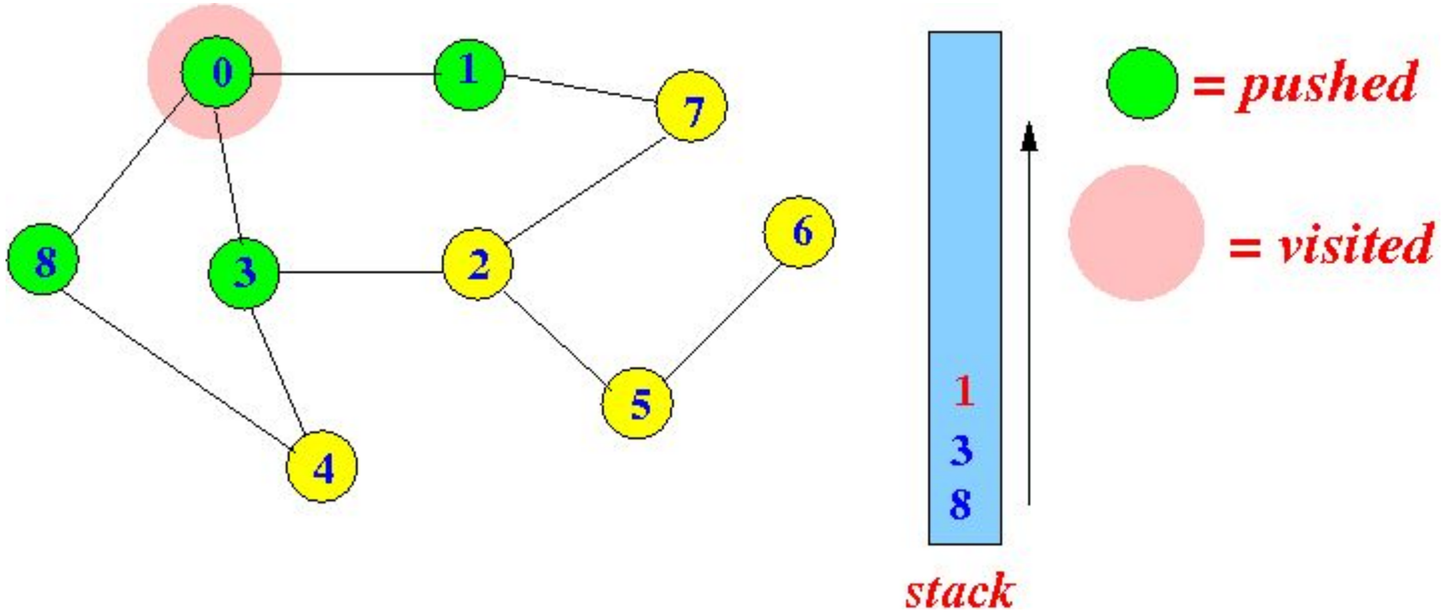


Key entered	Calculator action		Stack (bottom to top):
2	push 2		2
3	push 3		2 3
4	push 4		2 3 4
+	operand2 = peek	(4)	2 3 4
	pop		2 3
	operand1 = peek	(3)	2 3
	pop		2
	result = operand1 + operand2	(7)	
	push result		2 7
*	operand2 = peek	(7)	2 7
	pop		2
	operand1 = peek	(2)	2
	pop		
	result = operand1 * operand2	(14)	
	push result		14

Stack

Example: stack and depth-first search

- Depth-first Search (DFS) on graph (will be later lectures or CS180)



Stack*

Example: Use stack to implement DFS [Link]

UCLA Samueli Computer Science

```
void Graph::DFS(int s)
{
    vector<bool> visited(V, false);    // Initially mark all vertices as not visited
    stack<int> stack; // Create a stack for DFS
    stack.push(s);         // Push the current source node
    while (!stack.empty())
    {
        s = stack.top(); // Pop a vertex from stack and print it
        stack.pop();
        // Print the popped item only if it is not visited.
        if (!visited[s])  { cout << s << " "; visited[s] = true;  }
        for (auto i = adj[s].begin(); i != adj[s].end(); ++i)
            if (!visited[*i])
                stack.push(*i);
    }
}
```

Note: Get all adjacent vertices of the popped vertex s.
If the adjacent has not been visited, then push it to stack .

Stack*

Problem: Find largest rectangle all-1sub-matrix

Question:

Given a binary matrix, find the area of maximum size rectangle binary-sub-matrix with all 1’s.

Level: (Super) Difficult

Example 1:

```
Input:     0 1 1 0
           1 1 1 1
           1 1 1 1
           1 1 0 0

Output:    8
```

Example 2:

```
Input:     0 1 1 0
           1 1 1 1
           1 1 1 1
           1 1 0 0
           1 1 0 1
           1 1 1 0

Output:    10
```

```
Step 1: Find maximum area for row[0]
Step 2:
    for each row in 1 to N - 1
        for each column in that row
            if A[row][column] == 1
              update A[row][column] with
                A[row][column] += A[row - 1][column]
    find area for that row and update maximum area so far
```

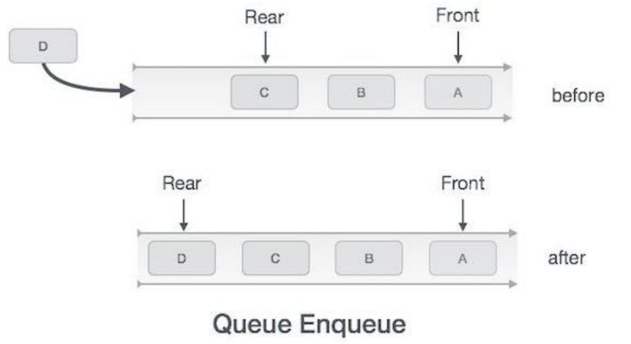
```
// Your solution here
// Think about how you can use stack to solve this problem and compare with brute-forth methods
// Tips: Think about the problem of Largest Rectangular Area in a Histogram solved by using stack
// One possible solution: [Link]
```

Queue: FIFO

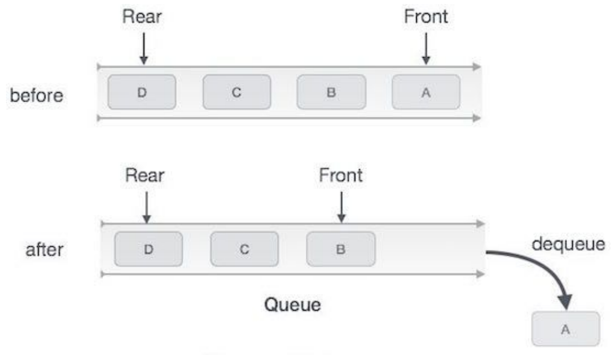
Review: Basics

UCLA Samueli Computer Science

- FIFO: First In, First Out
- Basic methods:
  - `enqueue(), dequeue()`
  - `front(), back()`
  - `count()`
- Applications
  - Data streams
  - Process scheduling (DMV service request)
  - Breadth-first graph search
- How to implement queue with linked lists or dynamic arrays?



Queue Enqueue



Queue Dequeue

Queue

Extension: Deque (double-ended queue)

```
class Deque
{
    public:
        bool push_front(const ItemType& item);
        bool push_back(const ItemType& item);
        bool pop_front(const ItemType& item);
        bool pop_back(const ItemType& item);
        bool empty() const; // true if empty
        int count() const; // number of items
    private:
        int size; // Some data structure that keeps the items.
};
```

Question: How to implement `class Deque` with linked lists?

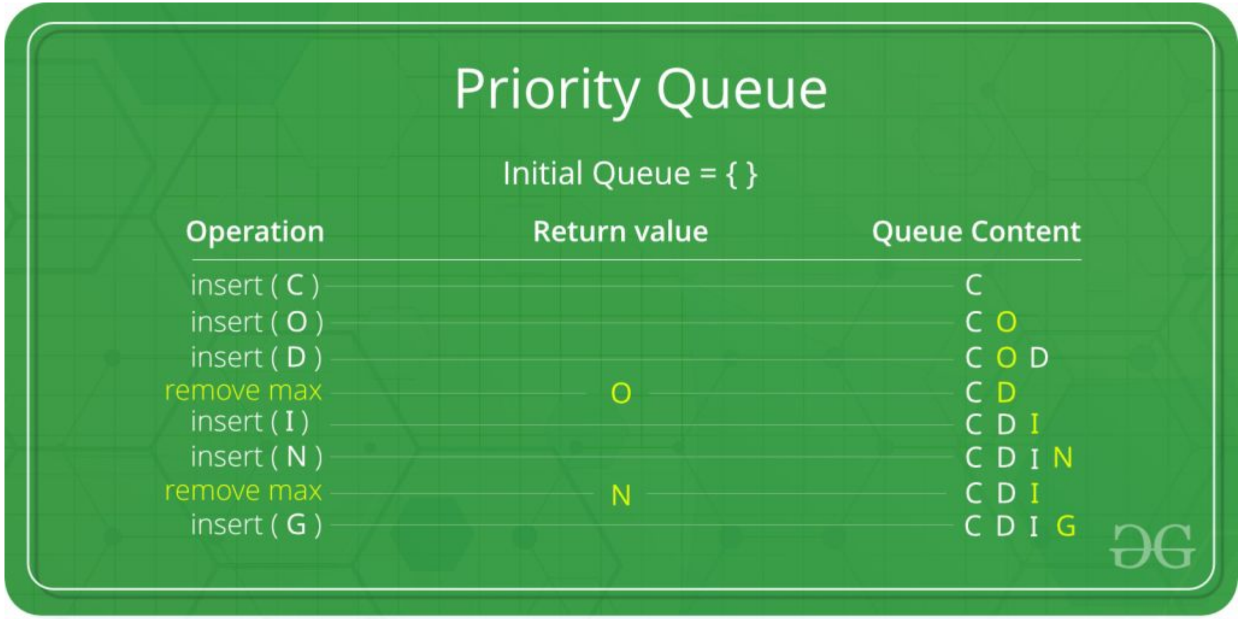
Queue

Extension: Priority queue

- Data: A finite number of objects, not necessarily distinct, having the same data type and ordered by priority
- Operations:
  - Add a new entry to the queue based on priority
  - Remove the entry with the highest priority from the queue
- We will learn priority queue (and heap) later this quarter after tree!

Queue

Extension: Priority queue



Priority Queue

Initial Queue = { }

Operation	Return value	Queue Content
insert (C)		C
insert (O)		C O
insert (D)		C O D
remove max	O	C D
insert (I)		C D I
insert (N)		C D I N
remove max	N	C D I
insert (G)		C D I G

A typical priority queue supports following operations.

insert(item, priority): Inserts an item with given priority.

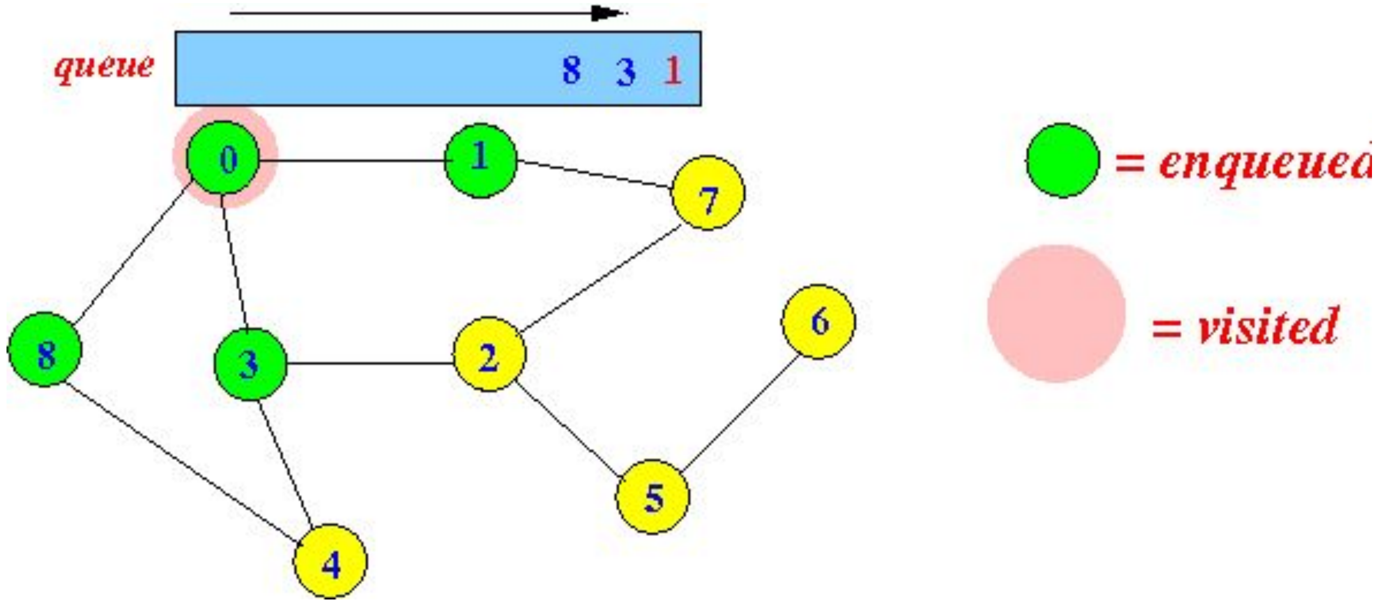
getHighestPriority(): Returns the highest priority item.

deleteHighestPriority(): Removes the highest priority item.

Queue*

Example: queue and breadth-first search

- Breadth-first Search (BFS) on graph (will be later lectures or CS180)



Queue*

Example: Use queue to implement BFS [Link]

UCLA Samueli Computer Science

```
void Graph::BFS(int s)
{
    bool *visited = new bool[V];  // Mark all the vertices as not visited
    for(int i = 0; i < V; i++) { visited[i] = false; }
    list<int> queue; // Create a queue for BFS
    visited[s] = true; // Mark the current node as visited and enqueue it
    queue.push_back(s);
    list<int>::iterator i;
    while(!queue.empty()) {
        s = queue.front(); // Dequeue a vertex from queue and print it
        queue.pop_front();
        for (i = adj[s].begin(); i != adj[s].end(); ++i) {
            if (!visited[*i]) {
                visited[*i] = true;
                queue.push_back(*i);
            }
        }
    }
}
```

Note: Get all adjacent vertices of the dequeued vertex s. If a adjacent has not been visited, then mark it visited and enqueue it.

Suggestions on Stack and Queue

UCLA Samueli Computer Science

- ❖ Drawing pictures and carefully tracing the current status through your code, updating the picture with each statement, can help you find bugs in your code.
- ❖ **Infix to postfix conversion is very important!**
- ❖ We have shown that stack and queue can be used for traversal on graphs. They can also be applied on Trees (topics later in this quarter). Understand different data structures you use will result in different traversal order.
- ❖ You can use the given **Standard Template Library (STL)** to implement stack and queue.

```
#include <stack>
#include <queue>
std::stack<type> variableName;
std::queue<type> variableName;
```

Hints for Homework 2

UCLA Samueli Computer Science

Task: Use stack or queue to implement the `pathExists()` function in the maze.

```
// Homework 2 (P1-4): One possible solution

bool pathExists(string maze[], int sr, int sc,
int er, int ec){
  // Some basic case to return false??
  // initialize stack or queue
  // Push starting coordinate (sr,sc)
  // Loop & check (when stack/queue is non-empty)
  // Return true or false
}
int main(){ … } //test your code
```

Note & Reminders:

1. Follow the pseudocode if you still have no idea. The question itself is somewhat similar to BFS and DFS.
2. How to mark the `(r,c)` as visited?
3. Using stack and queue will result **different Coord visit order** in your `hw.txt` to be submitted.
4. Reminder: **You may either write your own queue class, or use the queue type from the C++ Standard Library.**

Group Exercises: Worksheet

- Exercise problems from **Worksheet 2** (see “LA worksheet” tab in CS32 website). Answers will be posted after all discussions.
- Questions for today:
    - 1-4 from the stacks and queues portion