HANYANG UNIVERSITY
DIVISION OF COMPUTER SCIENCE

Data Structures

박영준 교수님

Lab3: Stack, Queue

Stack

- An ordered list in which insertions and deletions can be performed at one end of the list
- Push(s, x) : insert x in the list s
- pop(s) : deletes the most recently inserted element from s

Stack ADT

- void InitStack(ArrayStack *stack);
  - Pass the address of the list to be initialized as an argument
  - Must be called first after creating the list
- int IsEmpty(ArrayStack *stack);
  - If stack is empty, return 1
  - If stack is not empty, return 0

Stack ADT

- void Push(ArrayStack *stack, DATATYPE data);
  - Save data into stack
- DATATYPE Pop(ArrayStack *stack);
  - Delete the last data saved in stack
  - Return deleted data
- DATATYPE Peek(ArrayStack *stack);
  - Return the last data saved in stack
  - Do not delete data

Stack

Array

```c
#include <stdio.h>
#include <stdlib.h>

#define TRUE 1
#define FALSE 0
#define STACKLEN 100
#define EMPTY -1

typedef int DATATYPE;

typedef struct
{
    DATATYPE StackArr[STACKLEN];
    int Top;
} ArrayStack;

void InitStack(ArrayStack *stack);
int IsEmpty(ArrayStack *stack);

void Push(ArrayStack *stack, DATATYPE data);
DATATYPE Pop(ArrayStack *stack);
DATATYPE Peek(ArrayStack *stack);
```

```c
void InitStack(ArrayStack *stack)
{
    stack->Top = EMPTY;
}

int IsEmpty(ArrayStack *stack)
{
    if(stack->Top == EMPTY)
    {
        return TRUE;
    }
    else
    {
        return FALSE;
    }
}

void Push(ArrayStack *stack, DATATYPE data)
{
    stack->Top += 1;
    stack->StackArr[stack->Top] = data;
}
```

Stack

Array

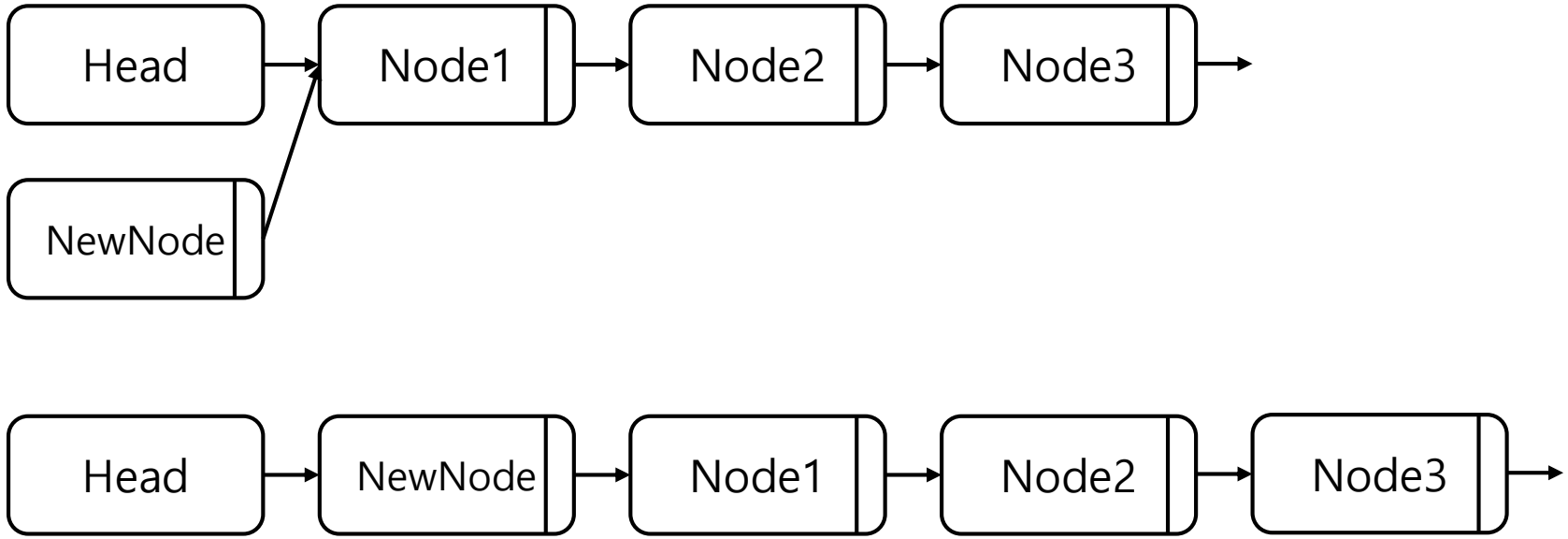
```
73 DATATYPE Pop(ArrayStack *stack)
74 {
75     int tempIdx;
76
77     if(IsEmpty(stack))
78     {
79         printf("Stack is empty\n");
80         exit(1);
81     }
82
83     tempIdx = stack->Top;
84     stack->Top -= 1;
85
86     return stack->StackArr[tempIdx];
87 }
88
89 DATATYPE Peek(ArrayStack *stack)
90 {
91     if(IsEmpty(stack))
92     {
93         printf("Stack is empty\n");
94         exit(1);
95     }
96
97     return stack->StackArr[stack->Top];
98 }
```

```
24 int main(int argc, char *argv[])
25 {
26     ArrayStack stack;
27
28     InitStack(&stack);
29
30     //save 5 data in stack
31     Push(&stack, 1);
32     Push(&stack, 2);
33     Push(&stack, 3);
34     Push(&stack, 4);
35     Push(&stack, 5);
36
37     //peek final data
38     printf("Final data : %d\n", Peek(&stack));
39
40     //pop data
41     while(!IsEmpty(&stack))
42     {
43         printf("%d ", Pop(&stack));
44     }
45     printf("\n");
46
47     return 0;
48 }
```

Stack

Linked List

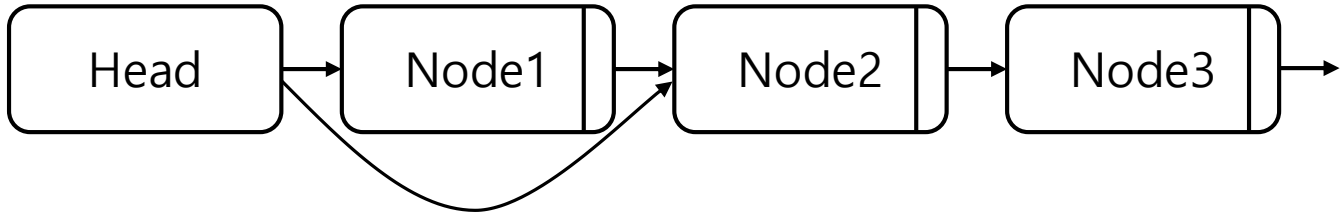
- Push

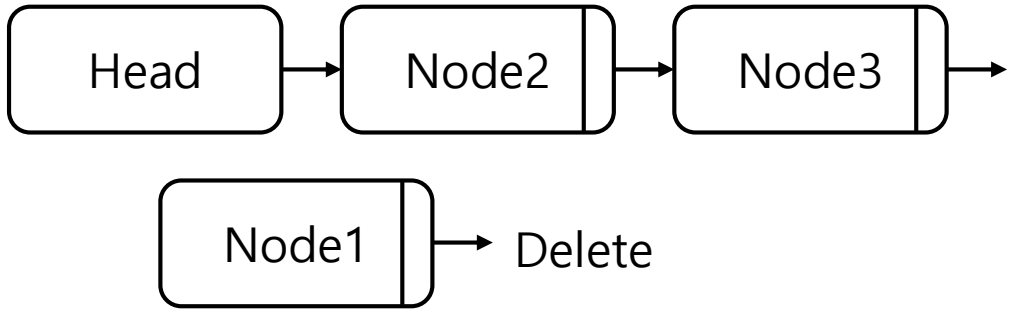


Stack

Linked List

- Pop





Stack

Linked List

- List structure
  - typedef struct
    {
      NODE *head; → Points head node
    } ListStack;

Stack

Linked List

```c
#include <stdio.h>
#include <stdlib.h>

#define TRUE 1
#define FALSE 0
#define EMPTY -1

typedef int DATATYPE;

typedef struct NODE
{
    DATATYPE data;
    struct NODE *next;
} NODE;

typedef struct
{
    NODE *head;
} ListStack;

void InitStack(ListStack *stack);
int IsEmpty(ListStack *stack);

void Push(ListStack *stack, DATATYPE data);
DATATYPE Pop(ListStack *stack);
DATATYPE Peek(ListStack *stack);

void InitStack(ListStack *stack)
{
    stack->head = NULL;
}

int IsEmpty(ListStack *stack)
{
    if(stack->head == NULL)
    {
        return TRUE;
    }
    else
    {
        return FALSE;
    }
}

void Push(ListStack *stack, DATATYPE data)
{
    NODE *temp = (NODE*)malloc(sizeof(NODE));

    temp->data = data;
    temp->next = stack->head;

    stack->head = temp;
}
```

Stack

Linked List

```
81  DATATYPE Pop(ListStack *stack)
82  {
83      DATATYPE tempData;
84      NODE *temp;
85
86      if(IsEmpty(stack))
87      {
88          printf("Stack is empty\n");
89          exit(1);
90      }
91
92      tempData = stack->head->data;
93      temp = stack->head;
94
95      stack->head = stack->head->next;
96      free(temp);
97
98      return tempData;
99  }
100
```

```
101 DATATYPE Peek(ListStack *stack)
102 {
103     if(IsEmpty(stack))
104     {
105         printf("Stack is empty\n");
106         exit(1);
107     }
108
109     return stack->head->data;
110 }
111
```

Stack

Linked List

```
28 int main(int argc, char *argv[])
29 {
30     ListStack stack;
31
32     InitStack(&stack);
33
34     //save 5 data in stack
35     Push(&stack, 1);
36     Push(&stack, 2);
37     Push(&stack, 3);
38     Push(&stack, 4);
39     Push(&stack, 5);
40
41     //peek final data
42     printf("Final data : %d\n", Peek(&stack));
43
44     //pop data
45     while(!IsEmpty(&stack))
46     {
47         printf("%d ", Pop(&stack));
48     }
49     printf("\n");
50
51     return 0;
52 }
53
```

Queue

Linked List

- A list that insertion is done at one end, whereas deletion is performed at the other end.
- Enqueu(q, x) : inserts x at the end of q
- Dequeue(q) : deletes the element at the start of the q

Queue

Linked List

- List structure
  - typedef struct
    {
      NODE *front; → pointing the first data
      NODE *rear; → pointing the last inserted data
    } QueueLinkedList;

Queue ADT

Linked List

- void InitQueue(QueueLinkedList *queue);
  - Pass the address of the list to be initialized as an argument
  - Must be called first after creating the list
- int IsEmpty(QueueLinkedList *queue);
  - If stack is empty, return 1
  - If stack is not empty, return 0

Queue ADT

Linked List

- void Enqueue(QueueLinkedList *queue, DATATYPE data);
  - Save data into queue
- DATATYPE Dequeue(QueueLinkedList *queue);
  - Delete the data at the start of the queue
  - Return deleted data
- DATATYPE Peek(QueueLinkedList *queue);
  - Return the data at the start of the queue
  - Do not delete data

Queue ADT

Linked List

```c
#include <stdio.h>
#include <stdlib.h>

#define TRUE 1
#define FALSE 0

typedef int DATATYPE;

typedef struct NODE
{
    DATATYPE data;
    struct NODE *next;
} NODE;

typedef struct
{
    NODE *front;
    NODE *rear;
} QueueLinkedList;

void InitQueue(QueueLinkedList *queue);
int IsEmpty(QueueLinkedList *queue);

void Enqueue(QueueLinkedList *queue, DATATYPE data);
DATATYPE Dequeue(QueueLinkedList *queue);
DATATYPE Peek(QueueLinkedList *queue);
```

```c
void InitQueue(QueueLinkedList *queue)
{
    queue->front = NULL;
    queue->rear = NULL;
}

int IsEmpty(QueueLinkedList *queue)
{
    if(queue->front == NULL)
    {
        return TRUE;
    }
    else
    {
        return FALSE;
    }
}
```

Queue ADT

Linked List

```
71 void Enqueue(QueueLinkedList *queue, DATATYPE data)
72 {
73     NODE *temp = (NODE*)malloc(sizeof(NODE));
74     temp->data = data;
75     temp->next = NULL;
76
77     if(IsEmpty(queue))
78     {
79         queue->front = temp;
80         queue->rear = temp;
81     }
82     else
83     {
84         queue->rear->next = temp;
85         queue->rear = temp;
86     }
87 }
88
```

```
89 DATATYPE Dequeue(QueueLinkedList *queue)
90 {
91     NODE *temp;
92     DATATYPE tempData;
93
94     if(IsEmpty(queue))
95     {
96         printf("Queue is empty\n");
97         exit(1);
98     }
99
100    temp = queue->front;
101    tempData = temp->data;
102    queue->front = queue->front->next;
103
104    free(temp);
105    return tempData;
106 }
107
108 DATATYPE Peek(QueueLinkedList *queue)
109 {
110     if(IsEmpty(queue))
111     {
112         printf("Queue is empty\n");
113         exit(1);
114     }
115
116     return queue->front->data;
117 }
```

Queue ADT

Linked List

```
28 int main(int argc, char *argv[])
29 {
30     QueueLinkedList queue;
31     InitQueue(&queue);
32
33     //save 5 data
34     Enqueue(&queue, 1);
35     Enqueue(&queue, 2);
36     Enqueue(&queue, 3);
37     Enqueue(&queue, 4);
38     Enqueue(&queue, 5);
39
40     //print peek data
41     printf("%d\n", Peek(&queue));
42
43     //dequeue
44     while(!IsEmpty(&queue))
45     {
46         printf("%d ", Dequeue(&queue));
47     }
48     printf("\n");
49
50     return 0;
51 }
```

Lab3: Calculator

- Submit on GitLab
- Infix to postfix converter & postfix calculator using linked list based stack
- Create Lab3 directory on your own GitLab project
- Submit file : source code(c only, run on linux)
- Filename : StudentID_lab3.c
- Input : no

Lab3: Calculator

- Infix to postfix converter & postfix calculator using linked list based stack
- ex) Input infix equation : 1+2*3
  Converted postfix equation : 123*+
  Answer : 7
- ex) Input infix equation : (1+2)*3
  Converted postfix equation : 12+3*
  Answer : 9

Lab3: Calculator

Infix to postfix converter ADT

- void ConvertExp(char exp[]);
  - Pass the infix equation as an argument
  - Convert infix to postfix
  - Return postfix equation
  - Using linked list based stack for converting

Lab3: Calculator

Infix to postfix converter ADT

- int GetOperatorPriority(char op);
  - Return priority of operator
  - *, / → return 5 (highest priority)
  - +, - → return 3
  - (→ return 1
  - else → -1

Lab3: Calculator

Infix to postfix converter ADT

- int CompareOperatorPriority(char op1, char op2);
  - Compare operators
  - if op1 > op2 → return 1
  - if op1 < op2 → return -1
  - if op1 == op2 → return 0

Lab3: Calculator

Postfix Calculator ADT

- int EvalExp(char exp[]);
  - Pass the postfix equation as an argument
  - Calculate post fix equation
  - Return answer
  - Using linked list based stack

Lab3: Calculator

- Infix equation format
  - Using single integer only(0~9 integer available, no other numbers allowed)
  - Can use (), consider priorities in () in converters and calculators
  - No any space
  - ex) Infix eq : (1+2)*3
    Result : 9
  - ex) Infix eq : (1 + 2) * 3 → has space, not available

Lab3: Calculator

- Output format
  - Postfix eq = result

```
123*+ = 7
12+3* = 9
12-3+52-* = 6
```

Lab3: Calculator

- Note
- in stdlib.h
  - strlen(const char *str);
    - Return length of str between the beginning of the string and the null character
  - strcpy(char *destination, char *source);
    - Copy source array to destination array

Lab3: Calculator

- Note
- in ctype.h
  - isdigit(int arg);
    - Task a single integer argument and return the value of type int
    - Character is passed as an argument, and convert to its ASCII value for the check
    - Character 0 to 9 available

Lab3: Calculator

- Note
- in string.h
  - memset(void *ptr, int value, size_t num);
    - Sets the first num bytes of the block of memory pointed by ptr to the specified value
    - ptr → pointer to the block of memory to fill
    - value → value to be set
    - num → number of bytes to be set to the values
    - ex) memset(str, 0, sizeof(len)); → fill the str with 0 as len size from the beginning