Data Structures and its Applications

Dinesh Singh

Department of Computer Science & Engineering

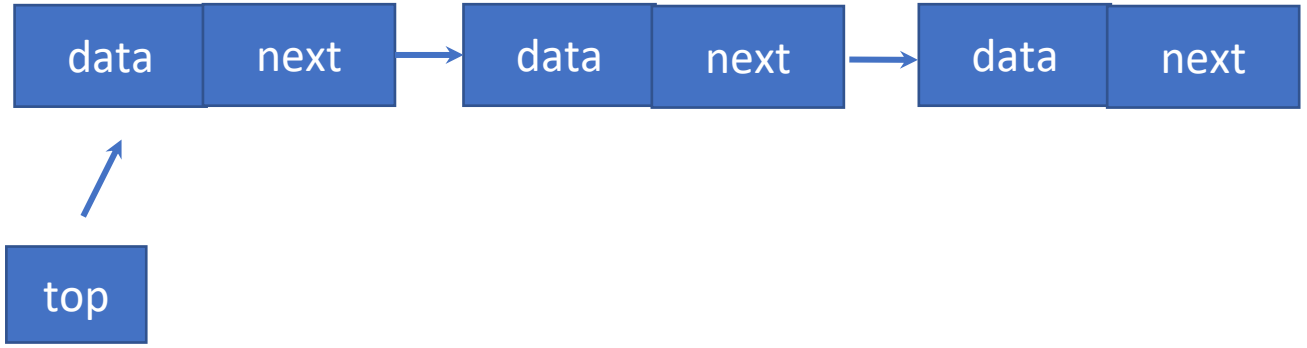
DATA STRUCTURES AND ITS APPLICATIONS

Stacks – Linked List Implementation

Dinesh Singh
Department of Computer Science & Engineering

Stacks – linked list implementation

- A stack can be easily implemented through the linked list. In the Implementation by a linked list, the stack contains a top pointer. which is “head” of the stack. The pushing and popping of items happens at the head of the list.



Structure of the stack

```
struct node
{
   int data;
   struct node *next;
 }
struct node *top
top=NULL;
```

Stacks – linked list implementation

Note :

- Items of the stack represented as the linked list.
- Each item is a node
- Top is a pointer that points to the first node (top of the stack)
- Top is initially NULL (Empty stack)
- Insertion and deletion happens at the front of the list
- stack size is not limited.

Operations on the stack

- Push : Inserting an element at the front of the list
- Pop : delete an element from the front of the list
- Display : displaying the list

Stacks – linked list implementation


```
//implements the push operation

void push(int x, struct node **top)
   {
        struct node *temp;
        temp=(struct node*)malloc(sizeof(struct node));
        temp->data=x;
        temp->next=*top; // insert in front of the list
        *top=temp; // make top points to the new top node
    }
```

Stacks – linked list implementation

```
//implements the pop operation
//returns top element, -1 if stack is empty

int pop(struct node **top)
    {
        int x;
       struct node *q;

       if(*top==NULL)
        {
          printf("Empty Stack\n");
           return -1;
        }
```

Stacks – linked list implementation

```
//implements the pop operation

else
        {
          q=*top;
           x=q->data; // get the top element
          *top=q->next; // make top point to the next top
          free(q); // free the memory of the node
           return(x);
         }
    }
```

Stacks – linked list implementation

```
//implements the display  operation

void display(struct node *top)
    {
       if(top==NULL)
         printf("Empty Stack\n");
       else
        {
          while(top!=NULL)
          {
            printf("%d->“,top->data);
            top=top->next;
          }
        }
     }
```

Stacks – linked list implementation

Another representation of structure of stack

```
struct node
{
  int data;
  struct node *next;
};
struct stack
{
   struct node *top;
}

struct stack s;
s.top=NULL;
```

Stacks – linked list implementation

```
//implements the push operation

void push(int x, struct stack * s)
//s is pointer to structure stack
   {
        struct node *temp;
        temp=(struct node*)malloc(sizeof(struct node));
        temp->data=x;
        temp->next=s->top; // insert in front of the list
        s->top=temp; // make top points to the new top node
    }
```

Stacks – linked list implementation

```
//implements the pop operation
//returns top element, -1 if stack is empty

int pop(struct stack *s)
    {
        int x;
       struct node *q;

       if(s->top==NULL)
        {
          printf("Empty Stack\n");
           return -1;
         }
```

Stacks – linked list implementation

```
//implements the pop operation

else
        {
          q=s->top;
           x=q->data; // get the top element
          s->top=q->next; // make top point to the next top
          free(q); // free the memory of the node
           return(x);
         }
    }
```

Stacks – linked list implementation

//implements the display operation

```
void display(struct stack *s)
   {
      struct node *q;
      if(s->top==NULL)
        printf("Empty Stack\n");
      else
       {
        q=s->top;
        while(q!=NULL)
        {
          printf("%d->“,q->data);
          q=q->next;
         }
       }
   }
```

Stacks – Application

Write an Algorithm to print a string in the reverse order

```
//prints the text in a reverse order
reverse(t)
{
 i=0;
 //push all the characters on to the stack
 while(t[i]!='\0')
 {
    push(s,t[i]);
    i=i+1;
 }
```

pop all the characters from the stack until the stack is empty

```
while(!empty(s))
 {
    x= pop(s);
    print(x)
   }
}
```

1. **In a stack implemented using a linked list, where is the top of the stack maintained?**

a) At the head of the linked list

b) At the tail of the linked list

c) At the middle node

d) It depends on the implementation

1. **In a stack implemented using a linked list, where is the top of the stack maintained?**

a) At the head of the linked list

b) At the tail of the linked list

c) At the middle node

d) It depends on the implementation

2. What is the time required to perform the push() operation in a stack implemented using a linked list?

a) Constant time

b) Logarithmic time

c) Linear time

d) Quadratic time

2. What is the time required to perform the push() operation in a stack implemented using a linked list?

a) Constant time

b) Logarithmic time

c) Linear time

d) Quadratic time

3. What happens when pop() is called on an empty stack implemented using a linked list?

a) It deletes a NULL node

b) It returns garbage value

c) It results in underflow condition

d) It resets the top pointer to head

3. What happens when pop() is called on an empty stack implemented using a linked list?

a) It deletes a NULL node

b) It returns garbage value

c) It results in underflow condition

d) It resets the top pointer to head

4. Which of the following is true about a stack implemented using a linked list vs. an array?

a) Linked list implementation is faster for push() and pop()

b) Array implementation requires dynamic memory allocation

c) Linked list implementation has no fixed size limit (until memory is exhausted)

d) Both a and c

4. Which of the following is true about a stack implemented using a linked list vs. an array?

a) Linked list implementation is faster for push() and pop()

b) Array implementation requires dynamic memory allocation

c) Linked list implementation has no fixed size limit (until memory is exhausted)

d) Both a and c

5. Which of the following is correct for push() operation in a linked list implementation?

a) Create a new node and add it at the end

b) Create a new node and add it at the beginning, change top (pointer)

c) Move the tail pointer backward

d) None of the above

5. Which of the following is correct for push() operation in a linked list implementation?

a) Create a new node and add it at the end

b) Create a new node and add it at the beginning, change top (pointer)

c) Move the tail pointer backward

d) None of the above

THANK YOU

Dinesh Singh
Department of Computer Science & Engineering