Tutorial : Abstract Data Types - Stack

An abstract data type (ADT) is a specification of a set of data and the set of operations that
can be performed on the data. Such data type is **abstract** in the sense that it is independent of various concrete implementations.

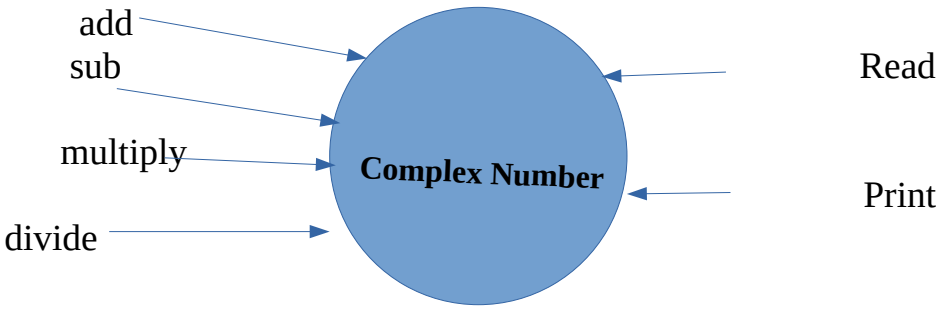
Example 1: List of size N: $A_0, A_1, \ldots, A_{N-1}$

1. Each element A_k has a unique position k in the list

2. Elements can be arbitrarily complex operations
   `insert(X,k), remove(k),find(X),findKth(k),printList()`

Some more examples :

Example 2 : Complex Numbers




```
struct cplx {
             float re;
             float im;
           }
typedef struct cplx complex;
```

Structure Definition

```
complex *add (complex a, complex b);
complex *sub (complex a, complex b);
complex *mul (complex a, complex b);
complex *div (complex a, complex b);
complex *read();
void print (complex a);
```

Function Prototypes

Example 3 : Set Manipulation

```
struct node {
             int element;
             struct node *next;
           }
typedef struct node set;
```

Structure Definition

```
set *union (set a, set b);
set *intersect (set a, set b);
set *minus (set a, set b);
void insert (set a, int x);
void delete (set a, int x);
int size (set a);
```

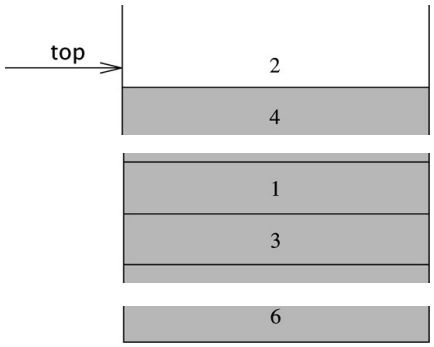
Function Prototypes

Example 4 : Last In First Out Stack

Stack = a list where insert and remove take place only at the "top"

Operations

Push (insert) element on top of stack
Pop (remove) element from top of stack
Top: return element at top of stack



Assume:: stack contains integer elements

```
void push (stack s, int element); /* Insert an element in
the stack */

int pop (stack s);                 /* Remove and return the
top element */

void create (stack s);            /* Create a new stack */

int isempty (stack s);            /* Check if stack is
empty */

int isfull (stack s);             /* Check if stack is full
*/
```

Stack Implementation :

1. Using Arrays
2. Using Linked List

Basic Idea :

In the array implementation, we would:
– Declare an array of fixed size (which determines the maximum size of the stack).
– Keep a variable top which always points to the "top" of the stack.
Contains the array index of the "top" element.

In the linked list implementation, we would:
– Maintain the stack as a linked list.
– A pointer variable top points to the start of the list.
– The first element of the linked list is considered as the stack top

Declaration

Array :

```
#define MAXSIZE 100

struct lifo
{
    int st[MAXSIZE];
    int top;
};
```

Linked List :

```
struct lifo
{
    int value;
    struct lifo *next;
};

typedef struct lifo stack;

typedef struct lifo stack;
```

Stack Creation

Array :

```
void create (stack *s)
{
    (*s).top = -1;

// s.top points to last element pushed in; initially -
1 //
}
```

Linked List :

```
void create (stack **top)
{
    *top = NULL;

// top points to NULL, indicating empty stack //


}
```

Pushing element into stack :

Array :

```
void push (stack *s, int element)
{
    if ((*s).top == (MAXSIZE-1))
    {
        printf (”\n Stack overflow”);
        exit(-1);
    }

else
{
    (*s).top++;
    (*s).st[(*s).top] = element;
}
```

Linked List :

```
void push (stack **top, int element)
{
    stack *new;

    new = (stack *) malloc(sizeof(stack));

    if (new == NULL)
    {
        printf (”\n Stack is full”);

        exit(-1);
    }

    new->value = element;

    new->next = *top;

    *top = new;
}
```

Popping element into stack :

Array :

```
int pop (stack *s)
{
    if ((*s).top == -1)
    {
        printf (”\n Stack underflow”);
        exit(-1);
    }

    else
    {
        return ((*s).st[(*s).top--]);
    }
}
```

Linked List :

```
int pop (stack **top)
{
    int t;
    stack *p;
    if (*top == NULL)
    {
        printf (”\n Stack is empty”);
        exit(-1);

    }
    else
    {
        t = (*top)->value;
        p = *top;
        *top = (*top)->next;
        free (p);
        return t;
    }
```

Checking for stack empty:

Array :

```
int isempty (stack s)
{
    if (s.top == -1)
        return (1);
    else
        return (0);
}
```

Linked List :

```
int isempty (stack *top)
{
    if (top == NULL)
        return (1);
    else
        return (0);
}
```

Checking for stack full:

Array :

```
int isfull (stack s)
{
    if (s.top == (MAXSIZE-1))
        return (1);
    else
        return (0);
}
```

Linked List :

Not required for linked list implementation. In the push() function, we can check the return value of malloc().
– If -1, then memory cannot be allocated

Main function :

Array :

```
#include <stdio.h>
#define MAXSIZE 100

struct lifo
{
    int st[MAXSIZE];
    int top;
};

typedef struct lifo stack;

main()
{
    stack A, B;
    create(&A);
    create(&B);

    push(&A,10);
    push(&A,20);
    push(&A,30);
    push(&B,100);
    push(&B,5);

    printf (”%d %d”, pop(&A),

    pop(&B));

    push (&A, pop(&B));

    if (isempty(B))
        printf (”\nB is empty”);
}
```

Linked List :

```
#include <stdio.h>

struct lifo
{
    int value;
    struct lifo *next;
};

typedef struct lifo stack;

main()
{
    stack *A, *B;
    create(&A);
    create(&B);

    push(&A,10);
    push(&A,20);
    push(&A,30);
    push(&B,100);
    push(&B,5);
    printf (”%d %d”,

    pop(&A), pop(&B));

    push (&A, pop(&B));

     if (isempty(B))
         printf (”\nB is empty”);



}
```