SRI AKILANDESWARI WOMEN’S COLLEGE, WANDIWASH

DATA STRUCTURE AND ALGORITHM

Class : I UG Computer Science

Mrs. R.PADMASHREE

Assistant Professor Department of Computer Science

SWAMY ABEDHANADHA EDUCATIONAL TRUST, WANDIWASH

ABSTRACT DATA TYPE

- Data abstraction, or abstract data types, is a programming methodology where one defines not only the data structure to be used, but the processes to manipulate the structure
  - like process abstraction, ADTs can be supported directly by programming languages

- To support it, there needs to be mechanisms for
  - defining data structures
  - encapsulation of data structures and their routines to manipulate the structures into one unit
    - by placing all definitions in one unit, it can be compiled at one time

ADT DESIGN ISSUES

- Encapsulation: it must be possible to *define* a unit that contains a data structure and the subprograms that access (manipulate) it
  - design issues:
    - will ADT access be restricted through pointers?
    - can ADTs be parameterized (size and/or type of data being stored)?
- Information hiding: controlling access to the data structure through some form of interface so that it cannot be directly manipulated by external code

ADT DESIGN ISSUES

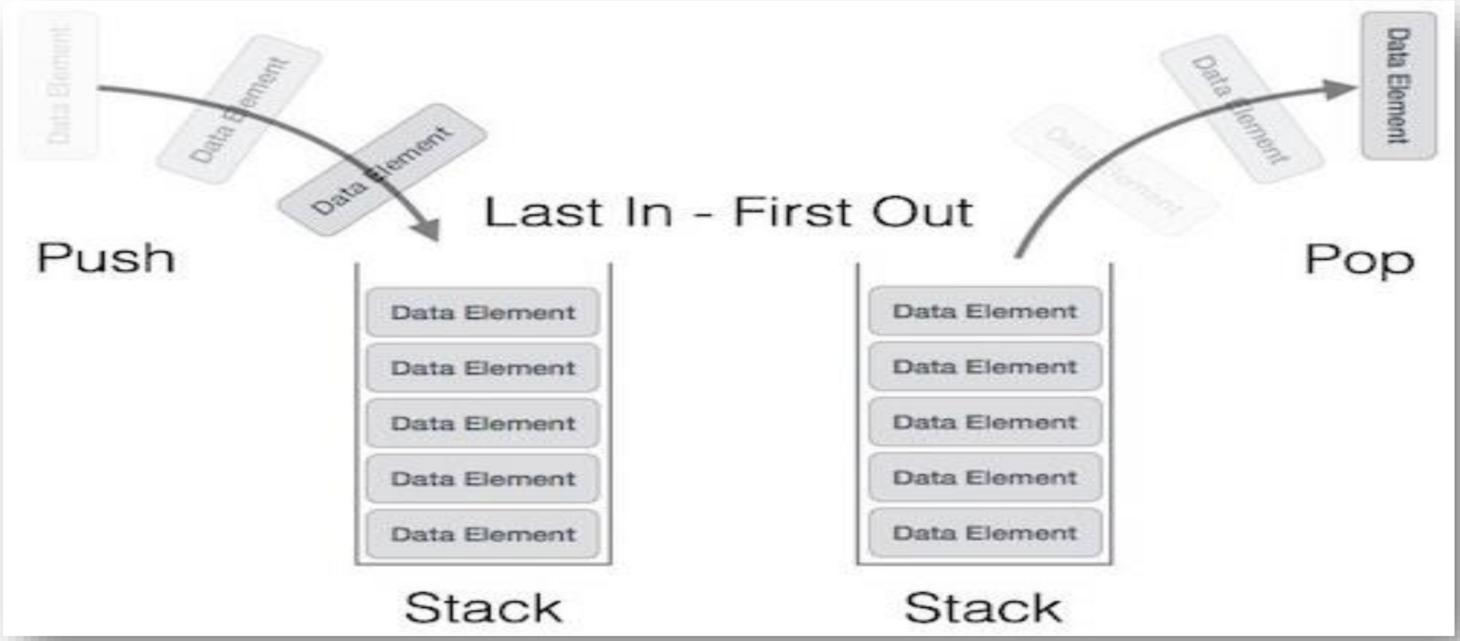
- often implemented via two sections of code
  - public part (interface) constitutes those elements that can be accessed externally (often limited to subprograms and constants)
  - private part, which remains secure because it is only accessible by subprograms of the ADT itself

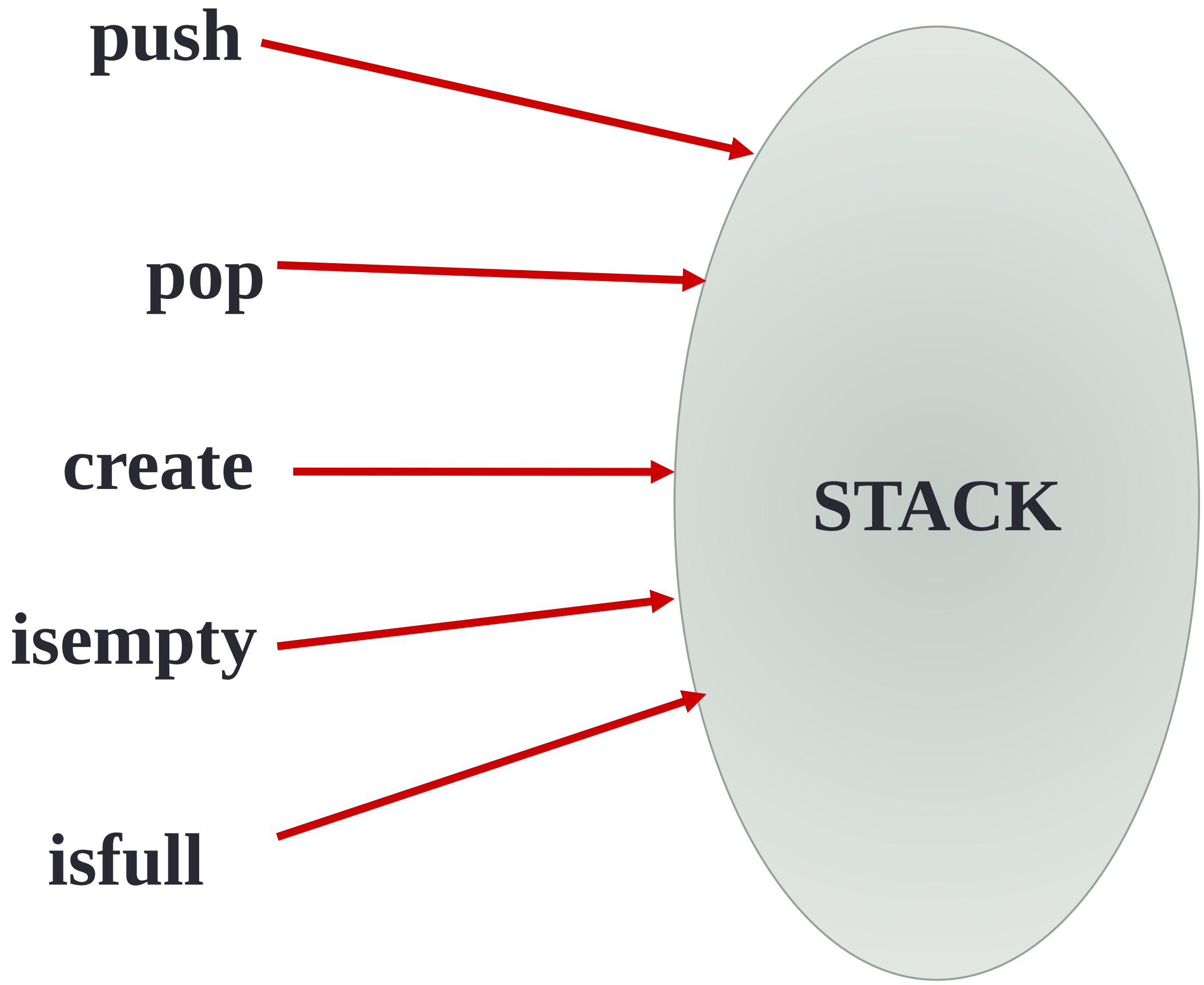
STACK

- A stack is an Abstract Data Type (ADT), commonly used in most programming languages. It is named stack as it behaves like a real-world stack, for example – a deck of cards or a pile of plates, etc.

STACK REPRESENTATION

- Can be implemented by means of Array, Structure, Pointers and Linked List.
- Stack can either be a fixed size or dynamic.



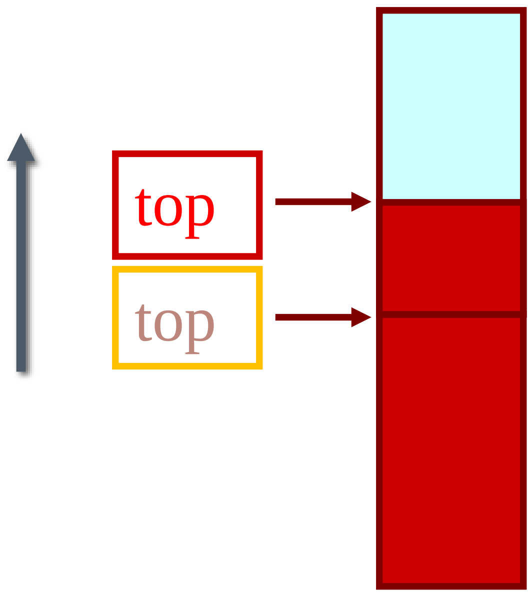

push
pop
create
isempty
isfull
STACK

STACK: Last-In-First-Out (LIFO)

- **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 */

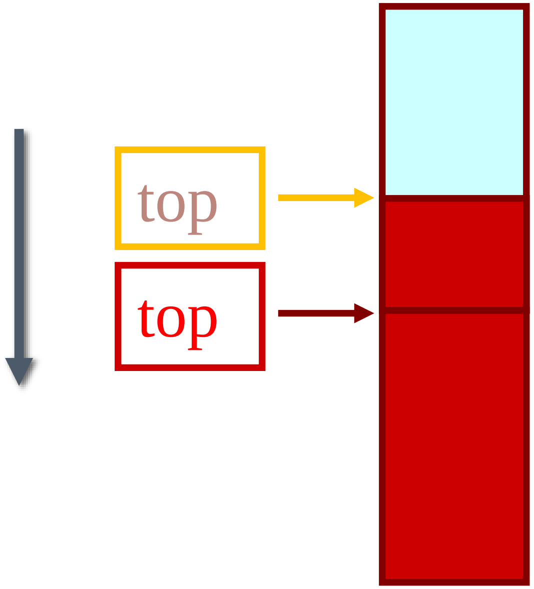
STACK USING ARRAY

PUSH USING STACK



PUSH

POP USING STACK



POP

STACK USING LINKED LIST

- In the array implementation, we would:
  - Declare an array of fixed size (which determines the maximum size of the stack).

  - Keep a variable 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

```
#define MAXSIZE 100

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

ARRAY

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

stack *top;
```

LINKED LIST

STACK CREATION

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

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

ARRAY

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

    /* top points to NULL,
       indicating empty
       stack             */
}
```

LINKED LIST

PUSHING AN ELEMENT INTO STACK

```
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;
     }
  }
```

ARRAY

```
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;
}
```

LINKED LIST

POPPING AN ELEMENT FROM STACK

```
int pop (stack *s)
  {
     if (s->top == -1)
     {
        printf ("\n Stack
underflow");
        exit(-1);
     }
     else
     {
        return (s->st[s->top--]);
     }
  }
```

ARRAY

```
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;
   }
}
```

LINKED LIST

BASIC IDEA

- Queue is an abstract data structure, somewhat similar to Stacks. Unlike stacks, a queue is open at both its ends. One end is always used to insert data (enqueue) and the other is used to remove data (dequeue).

QUEUE: First-In-First-Out (LIFO)

```
void enqueue (queue *q, int element);
                        /* Insert an element in the queue */
int dequeue (queue *q);
                        /* Remove an element from the queue */
queue *create();
                        /* Create a new queue */
int isempty (queue *q);
                        /* Check if queue is empty */
int size (queue *q);
                        /* Return the no. of elements in queue */
```


Thank you