INDIAN INSTITUTE OF INFORMATION TECHNOLOGY

Stack - Data Structures

Dr. Animesh Chaturvedi

Assistant Professor: IIIT Dharwad

Young Researcher: Heidelberg Laureate Forum

Postdoc: King's College London & The Alan Turing Institute

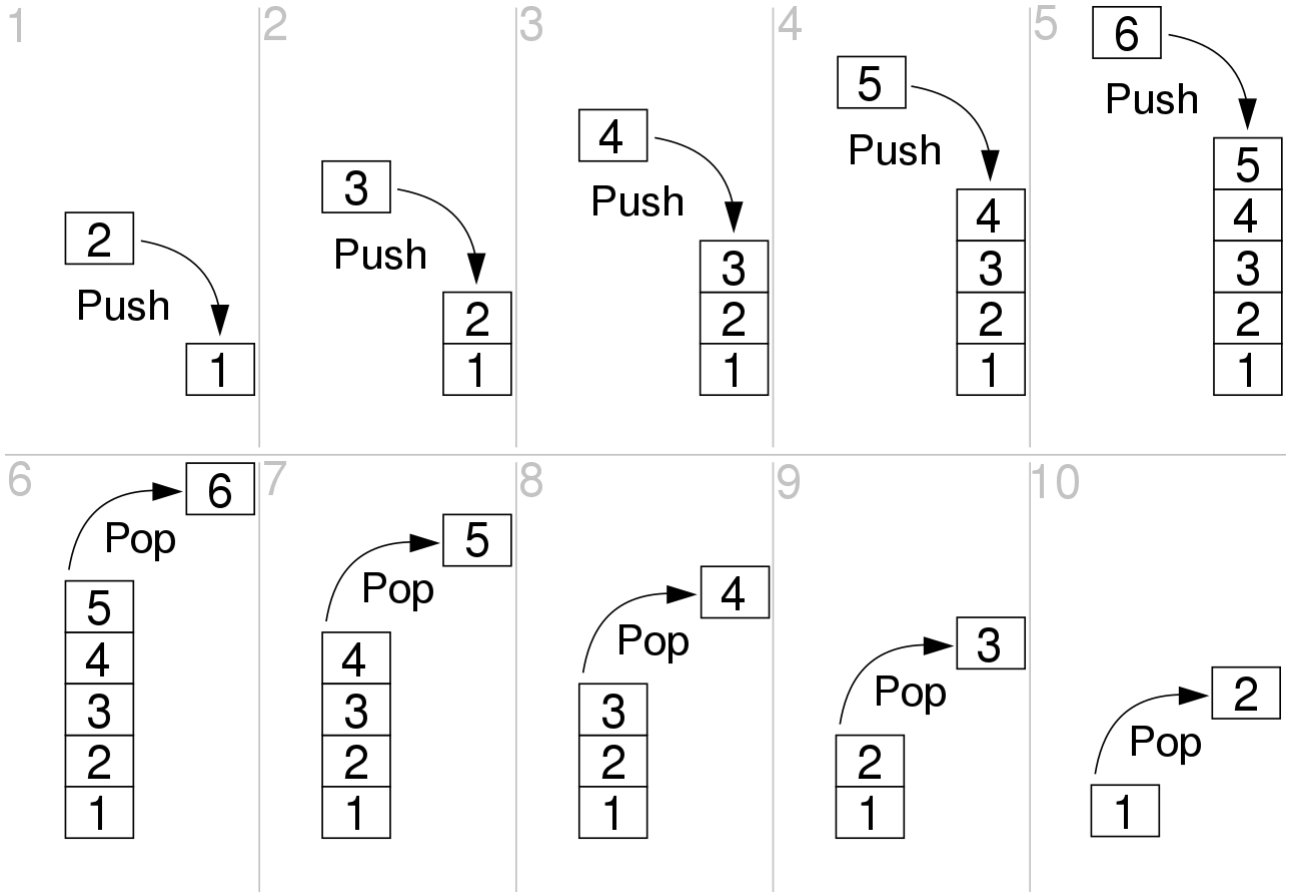
PhD: IIT Indore MTech: IIITDM Jabalpur

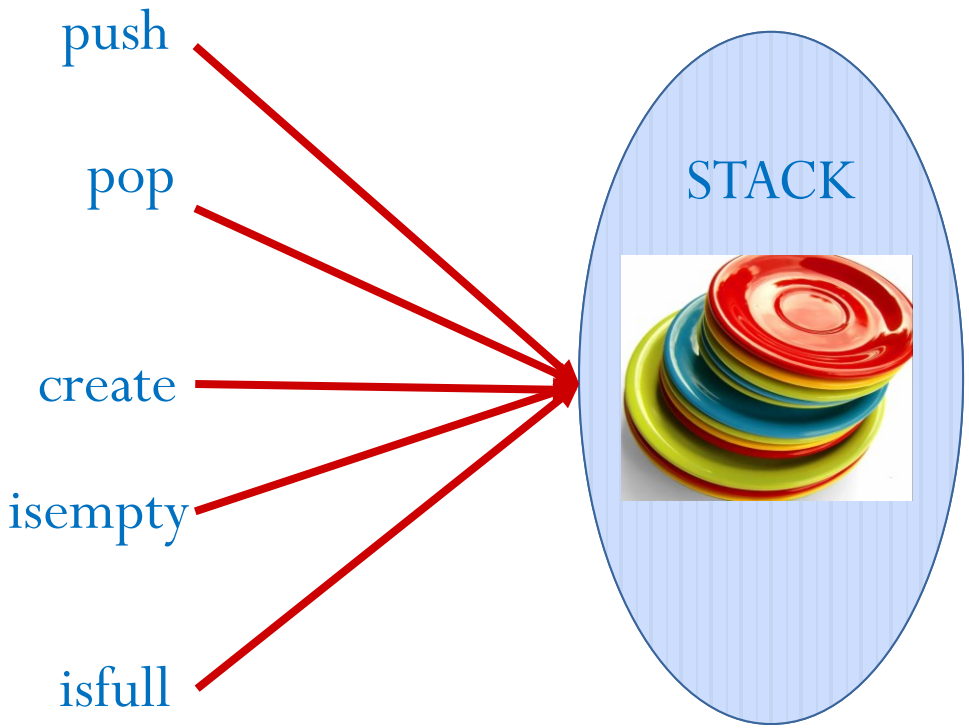
Stack Representation

- 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.



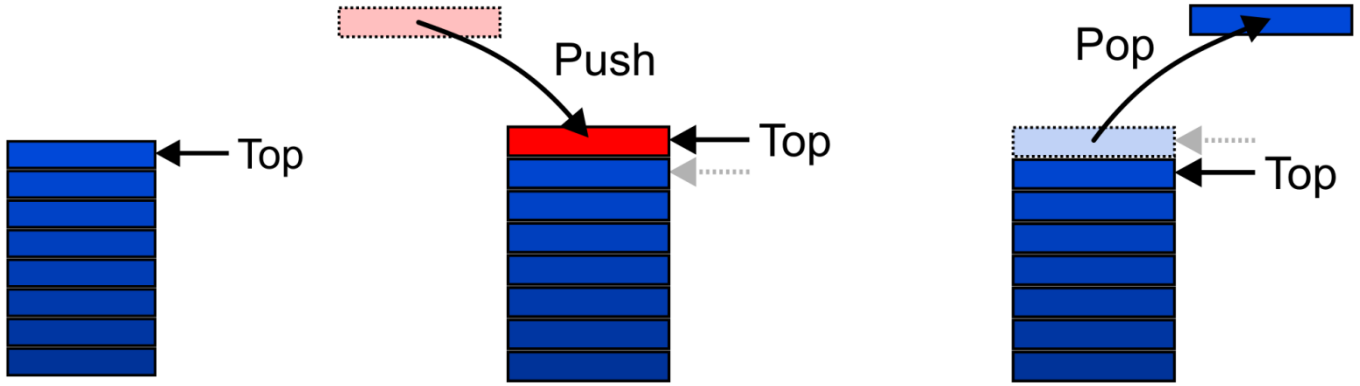


Stack

- Stack is an abstract data type which emphasizes specific operations:
  - Uses a explicit linear ordering
  - Insertions and removals are performed individually
  - Inserted objects are pushed onto the stack
  - The top of the stack is the most recently object pushed onto the stack
  - When an object is popped from the stack, the current top is erased

Stack

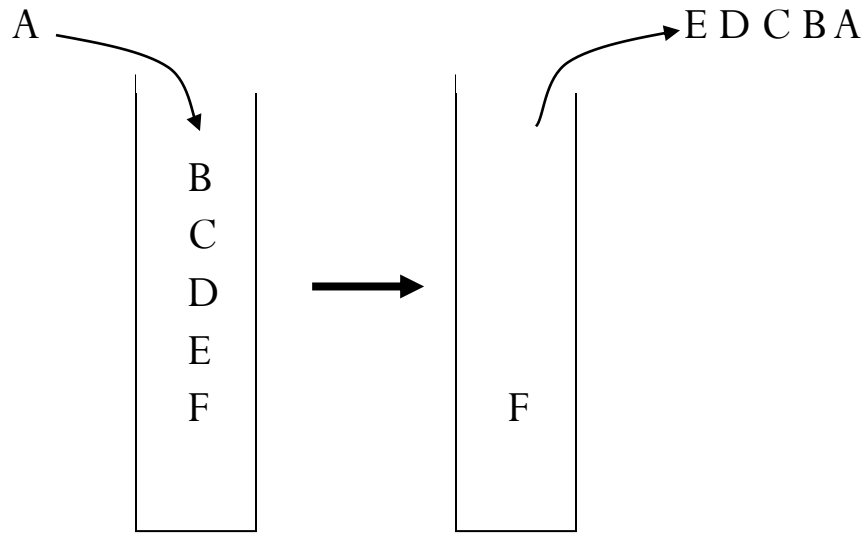
- Also called a last-in–first-out (LIFO) behaviour
  - Graphically, we may view these operations as follows:



- There are two exceptions associated with abstract stacks:
  - It is an undefined operation to call either pop or top on an empty stack

LIFO Stack ADT

- Stack operations
  - create
  - destroy
  - push
  - pop
  - top
  - is_empty



- Stack property: if x is on the stack before y is pushed, then x will be popped after y is popped

  LIFO: Last In First Out

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

- void create (stack *s);

  /* Create a new stack */
- void push (stack *s, int element);

  /* Insert an element in the stack */
- int pop (stack *s);

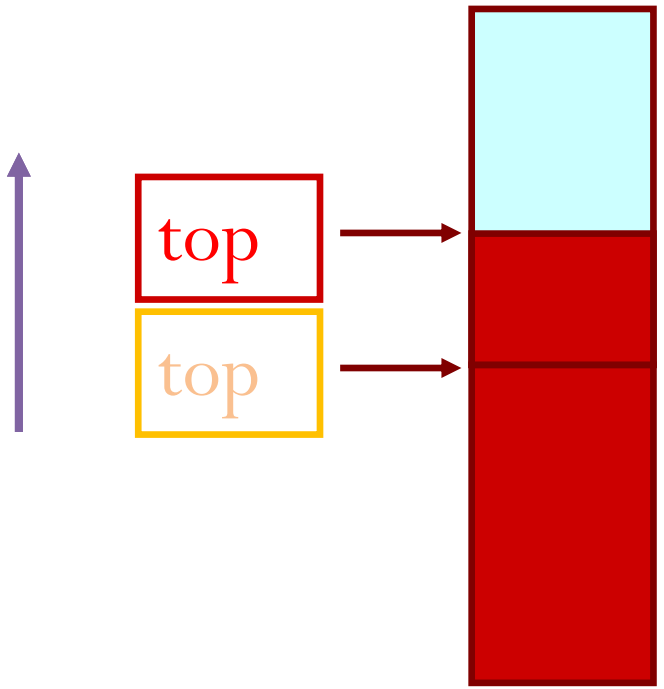
  /* Remove and return the top element */
- int isempty (stack *s);

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

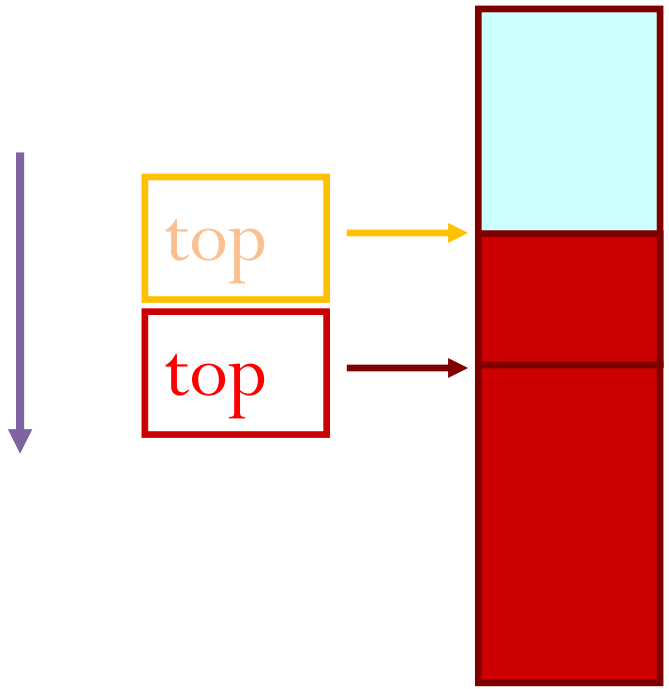
  /* Check if stack is full */

Assumption: stack contains integer elements!

Operations in Stack: Push and Pop

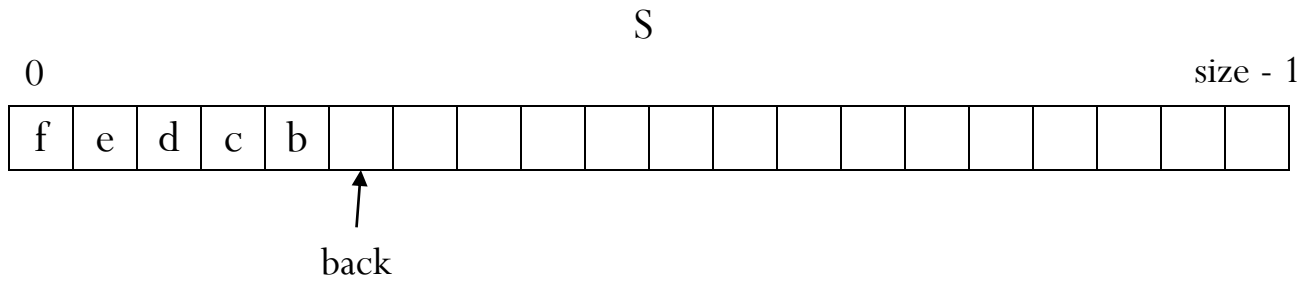


PUSH



POP

Array Stack Data Structure

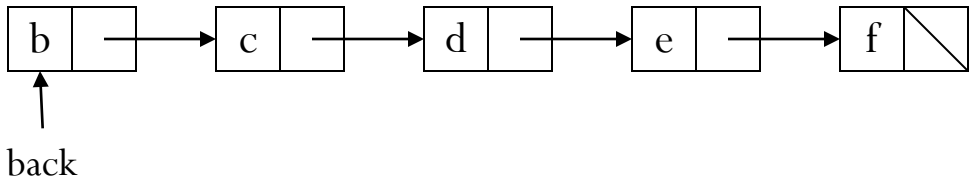



```
void push(Object x) {
   assert(!is_full())
   S[back] = x
   back++
}
Object top() {
   assert(!is_empty())
   return S[back - 1]
}
```

```
Object pop() {
   assert(!is_empty())
   back--
   return S[back]
}

bool is_full() {
   return back == size
}
```

Linked List Stack Data Structure

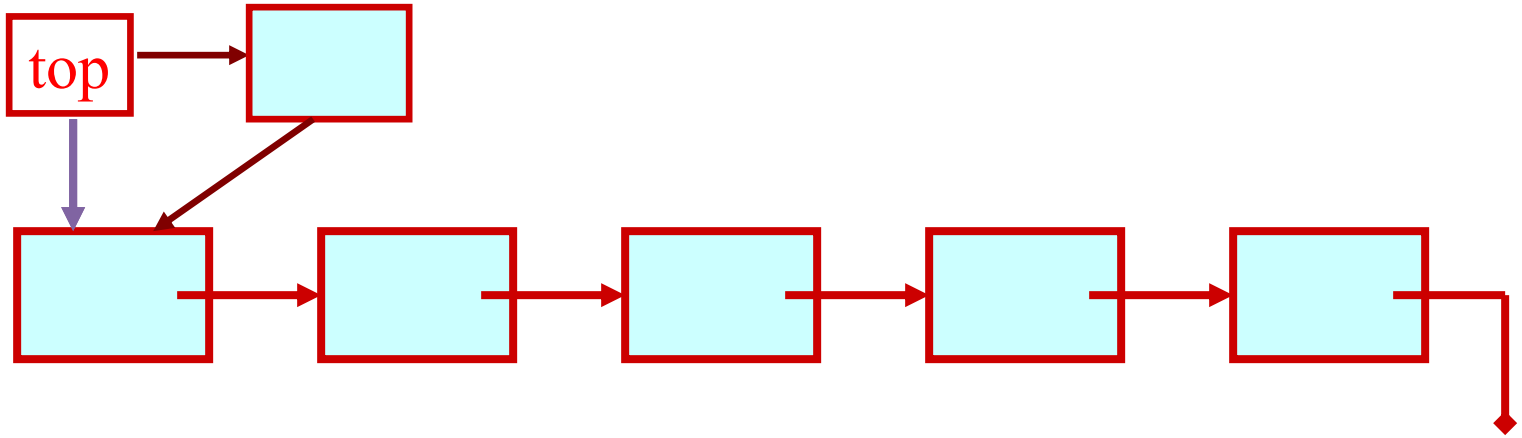



```
void push(Object x) {
   temp = back
   back = new Node(x)
   back->next = temp
}
Object top() {
   assert(!is_empty())
   return back->data
}
```

```
Object pop() {
   assert(!is_empty())
   return_data = back->data
   temp = back
   back = back->next
   free(temp)
   return return_data
}
```

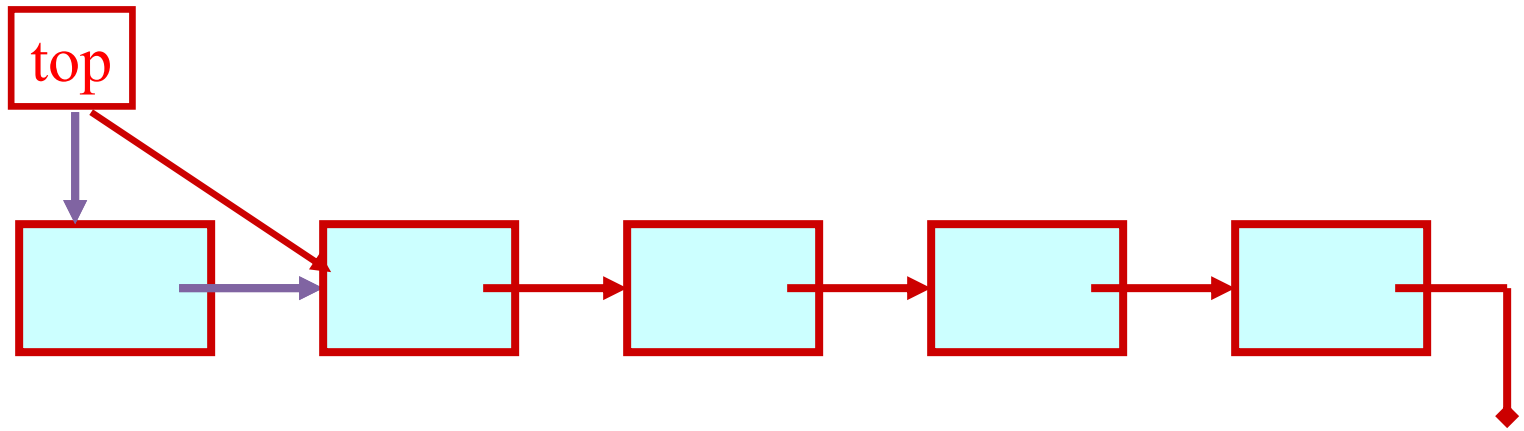
Push using Linked List

PUSH OPERATION



Pop using Linked List

POP OPERATION



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 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

Checking for stack empty

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

ARRAY

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

LINKED LIST

Example: Stack in C using 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 ("\n B is empty");
  return;
}
```

References

- Donald E. Knuth, The Art of Computer Programming, Volume 1: Fundamental Algorithms, 3rd Ed., Addison Wesley, 1997, §2.2.1, p.238.
- Cormen, Leiserson, and Rivest, Introduction to Algorithms, McGraw Hill, 1990, §11.1, p.200.
- Weiss, Data Structures and Algorithm Analysis in C++, 3rd Ed., Addison Wesley, §3.3.1, p.75.
- Koffman and Wolfgang, "Objects, Abstraction, Data Structures and Design using C++", John Wiley & Sons, Inc., Ch. 6.
- Wikipedia, http://en.wikipedia.org/wiki/Double-ended_queue
- CSE326: Data Structure, Department of Computer Science and Engineering, University of Washington https://courses.cs.washington.edu/courses/cse326
- Mike Scott, CS 307 Fundamentals of Computer Science, https://www.cs.utexas.edu/~scottm/cs307/
- Debasis Samanta, Computer Science & Engineering, Indian Institute of Technology Kharagpur, Spring-2017, Programming and Data Structures. https://cse.iitkgp.ac.in/~dsamanta/courses/pds/index.html

ขอบคุณ
Thai

Grazie
Italian

תודה רבה
Hebrew

धन्यवादः
Sanskrit

ಧನ್ಯವಾದಗಳು
Kannada

Ευχαριστώ
Greek

Thank You
English

Gracias
Spanish

Спасибо
Russian

Obrigado
Portuguese

شكراً
Arabic

https://sites.google.com/site/animeshchaturvedi07

Merci
French

多謝
Traditional
Chinese

धन्यवाद
Hindi

Danke
German

多谢
Simplified
Chinese

நன்றி
Tamil
Tamil

ありがとうございました
Japanese

감사합니다
Korean