UNIVERSITY OF DIYALA

Data Structure

Lecture 3:
Linear Data Structure: The Stack

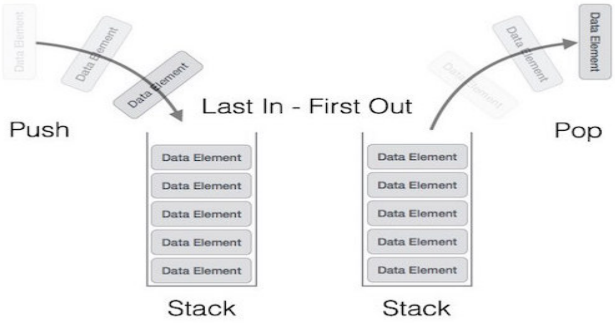
INSTRUCTOR
ALI A. AL-ANI

Department of Computer Science
College of Science

UNIVERSITY OF DIYALA

LDS: Stack

- *A **stack** is a linear data structure which can be accessed only at one of its ends for storing and retrieving data. **For this reason, a stack is called an LIFO structure: last in/first out**.*



Department of Computer Science
College of Science

LDS: Stack Operation

- *The two main operations which can be applied to a stack are given special names, when an item is added to a stack, it is **Pushed** to the stack, and when an item is removed, it is **Popped** from the stack.*

1. ***Push():** Insert element e at the top of the stack.*
2. ***Pop():** Remove the top element from the stack; an error occurs if the stack is empty.*

- *Additionally, these supporting functions:*

1. ***size():** Return the number of elements in the stack.*
2. ***Isempty():** Return true if the stack is empty and false otherwise.*
3. ***Isfull():** Return true if the stack is full and false otherwise.*

LDS: Stack Representation

- *Since a stack is a linear data structure, any linear data structure implementation will do. A stack can be implemented by means of Array and Linked List. Stack can either be a fixed size one or it may have a sense of dynamic resizing*

1. ***Non-linked- structures (The array).***
2. ***Linked structures (Linked list).***

Stack : Array Representation

- *The simplest method to represent a stack is to use an* ***array*** *to be home of the stack.*
- *The stack may therefore be declared and containing two objects: an* ***array*** *with suitable size and with suitable data type (Int, Float,..etc) to hold the elements of the stack, and an* ***integer*** *to indicate the position of the current stack top within the array.*
- ***Ex: The below declaration example in the C++ language***

```
const  SIZE= 10;
Int stack[SIZE];
Int top= -1;      // That is mean the stack empty.
```

Stack : Array Representation

- ***Push process: To perform the Push process to add new element in the stack, we should follow the following steps:***

1. *Verification the stack is not full through the value of index pointer* ***'top'*** *if it is less than the size of 'stack -1' or not. To avoid the case of overflow the size and unable to perform the addition process.*
2. *Update the value of the index pointer* ***'top=top+1'****, To refer to the following empty location.*
3. *Add the new element in the new location* ***'s[top]=the new value*** *'.*

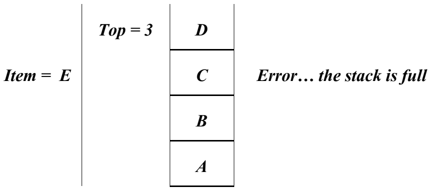
Stack : Array Representation

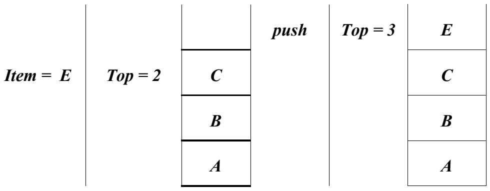
- *Sub program to sure if the stack is full or not*

```
int fullstack()
{    if  (top>=size-1)          return(1);
     else                        return(0);           }
```

- *Sub program to add an element to the Stack*

```
void push(int item)
{  if(fullstack())   {
cout<<"error...the stack is full"<<endl;
cout<<"press any key to exit"<<endl;
getch();   exit(0);  }
Else   {   top=top+1;    stack[top]=item;   }
 }
```





Stack : Array Representation

- ***Pop Process: To perform the properly deletion process, we should follow the following steps:***
- *Verification the stack is not empty through the value of the pointer 'top' it is not equal -1* ***'top!= -1'****. To avoid the case of underflow the size and unable to perform the deletion process.*
- *Take the element from the stack and store it in temporary location* ***'item = s[top]'****.*
- *Update the value of the index pointer* ***'top=top - 1',*** *To refer to the following location after the element deleted.*

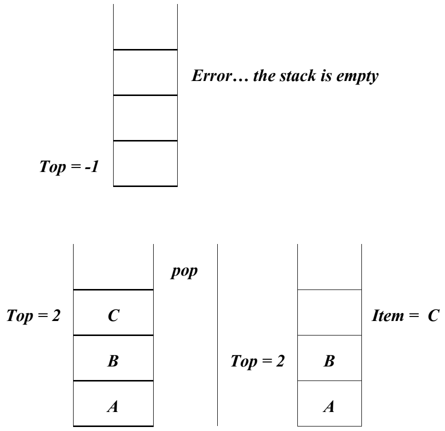
Stack : Array Representation

- *Sub program to sure if the stack is empty or not*

```
int emptystack()
{          if(top==-1)   return(1);
           else          return(0);       }
```

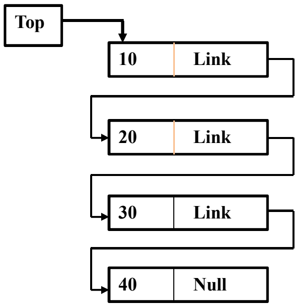
- *Sub program to delete an element from the stack*

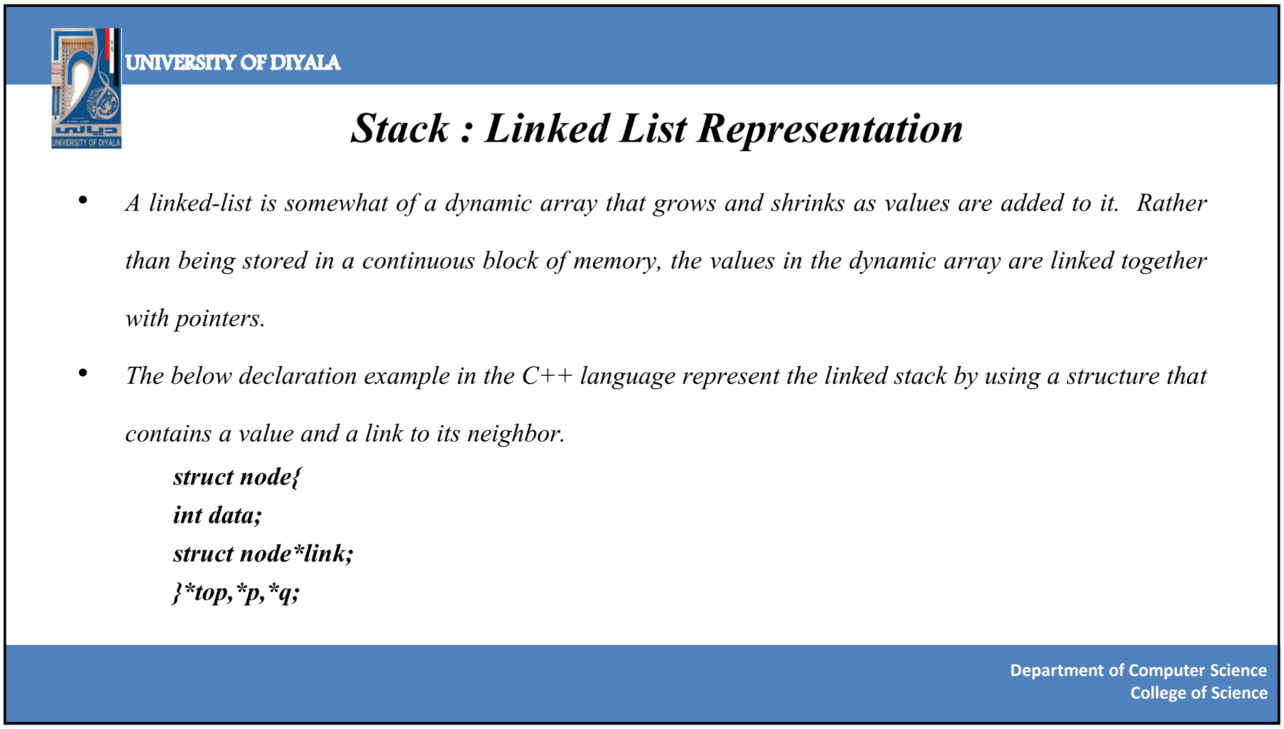
```
void pop()
{          if(emptystack())   {
cout<<"error…the stack is empty"<<endl;
cout<<"press any key to exit"<<endl;          getch();  exit(0);  }
Else {
item=stack[top];
top=top-1;          }}
```



Stack : Linked List Representation

- *The second implementation of a stack uses a singly linked list. We perform a **Push** by inserting at the **Front** of the list and perform a **Pop** by deleting the element at the **Front** of the list.*

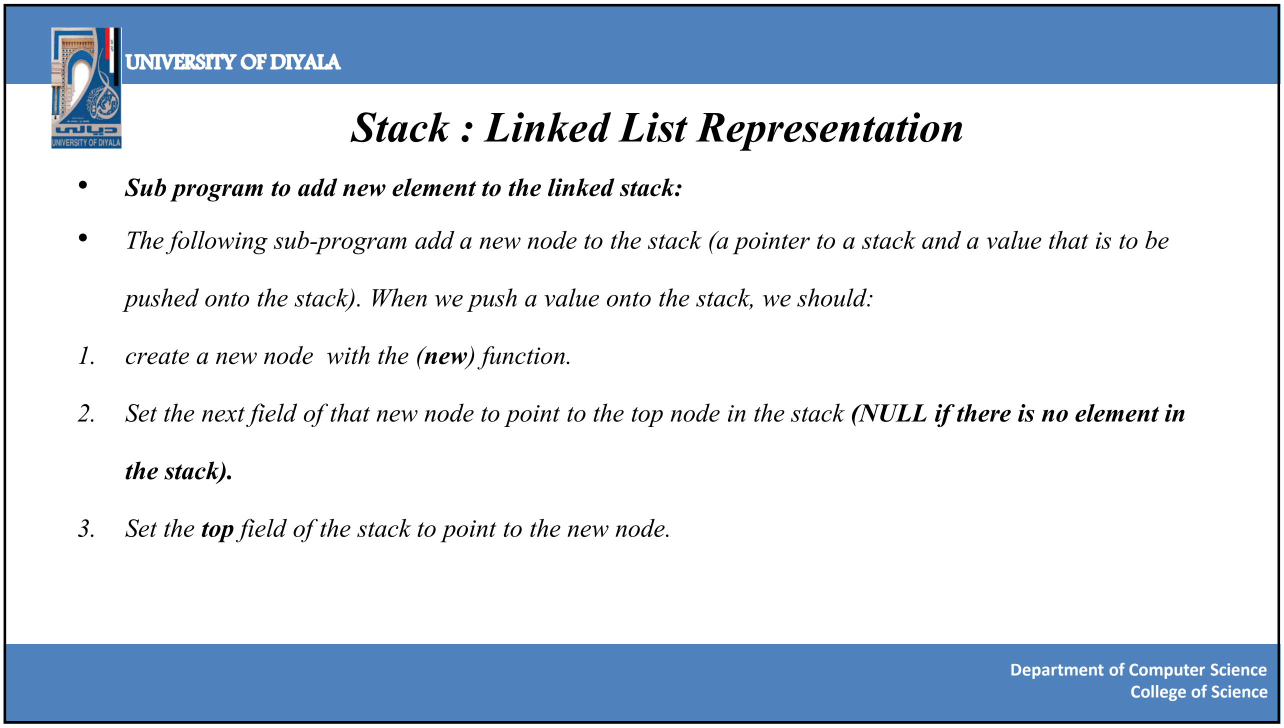




Stack : Linked List Representation

- *A linked-list is somewhat of a dynamic array that grows and shrinks as values are added to it. Rather than being stored in a continuous block of memory, the values in the dynamic array are linked together with pointers.*
- *The below declaration example in the C++ language represent the linked stack by using a structure that contains a value and a link to its neighbor.*

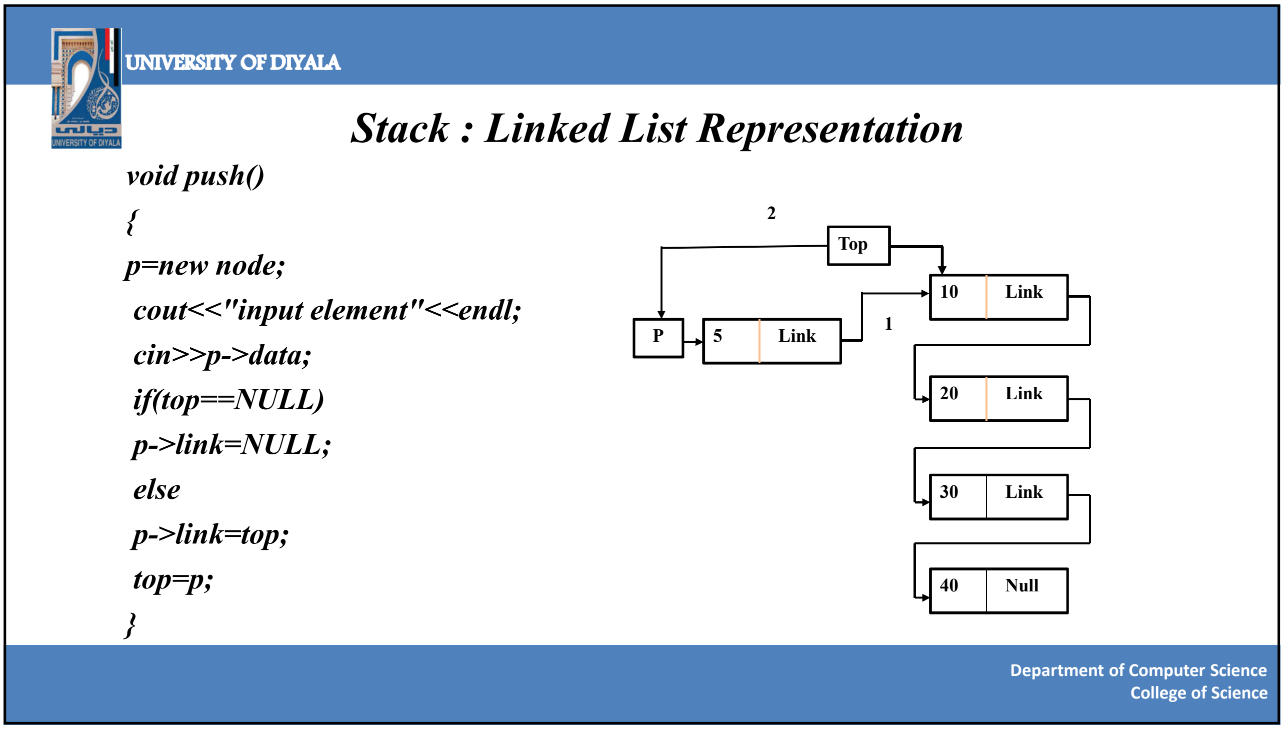
```
struct node{
int data;
struct node*link;
}*top,*p,*q;
```



Stack : Linked List Representation

- ***Sub program to add new element to the linked stack:***
- *The following sub-program add a new node to the stack (a pointer to a stack and a value that is to be pushed onto the stack). When we push a value onto the stack, we should:*

1. *create a new node with the (**new**) function.*
2. *Set the next field of that new node to point to the top node in the stack **(NULL if there is no element in the stack).***
3. *Set the **top** field of the stack to point to the new node.*



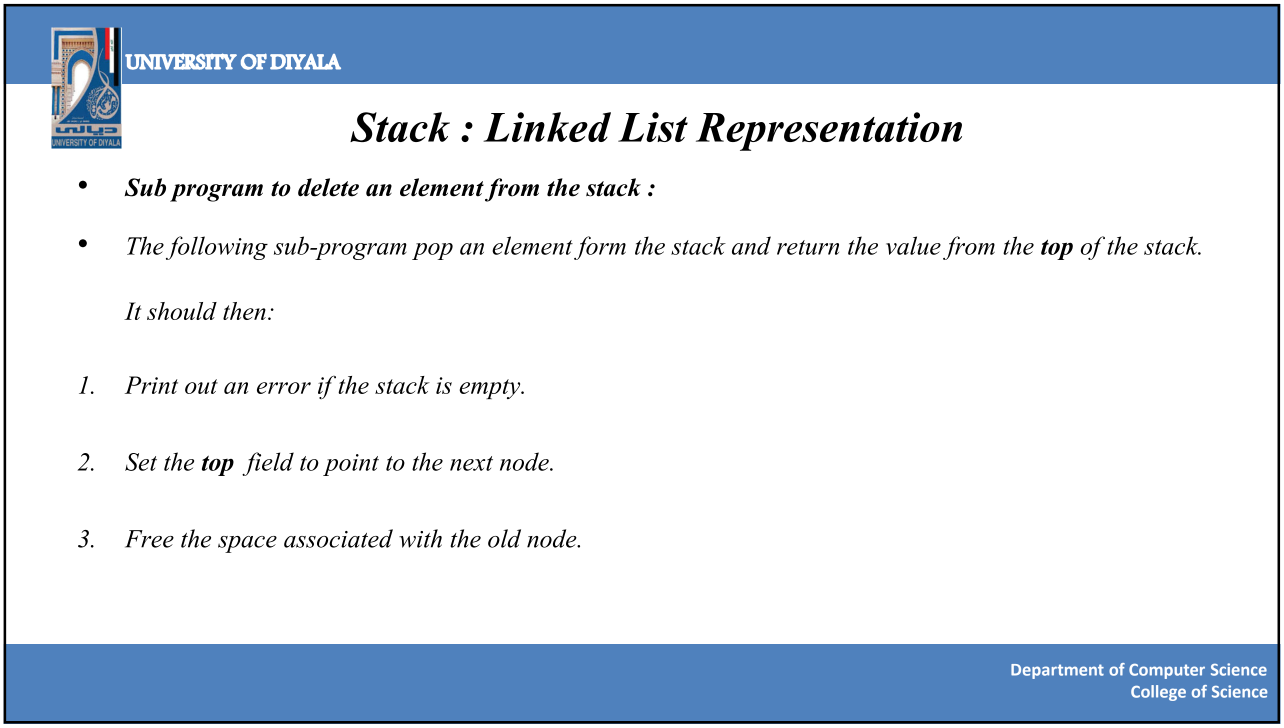
UNIVERSITY OF DIYALA

Stack : Linked List Representation

```
void push()
{
p=new node;
 cout<<"input element"<<endl;
 cin>>p->data;
 if(top==NULL)
 p->link=NULL;
 else
 p->link=top;
 top=p;
}
```

2
Top
10
Link
P
5
Link
1
20
Link
30
Link
40
Null

Department of Computer Science
College of Science



UNIVERSITY OF DIYALA

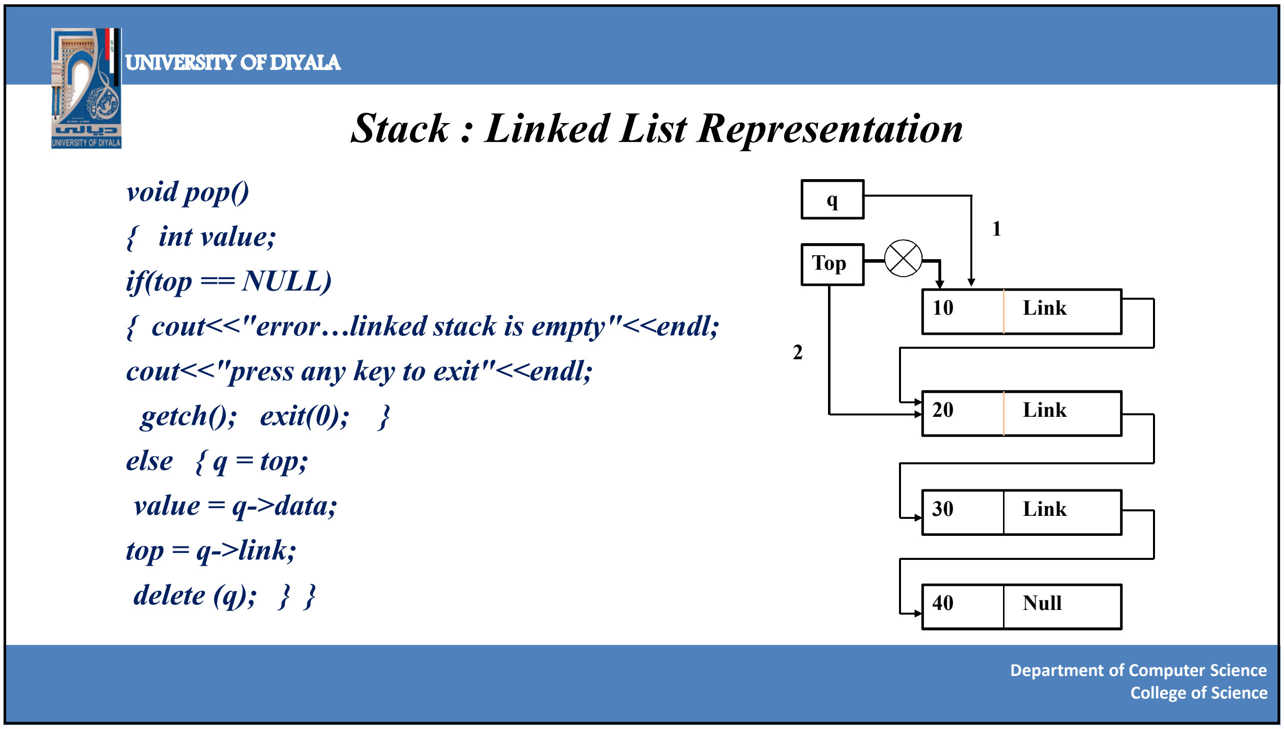
Stack : Linked List Representation

- ***Sub program to delete an element from the stack :***
- *The following sub-program pop an element form the stack and return the value from the* ***top*** *of the stack.*

  It should then:

1. *Print out an error if the stack is empty.*
2. *Set the* ***top*** *field to point to the next node.*
3. *Free the space associated with the old node.*

Department of Computer Science
College of Science



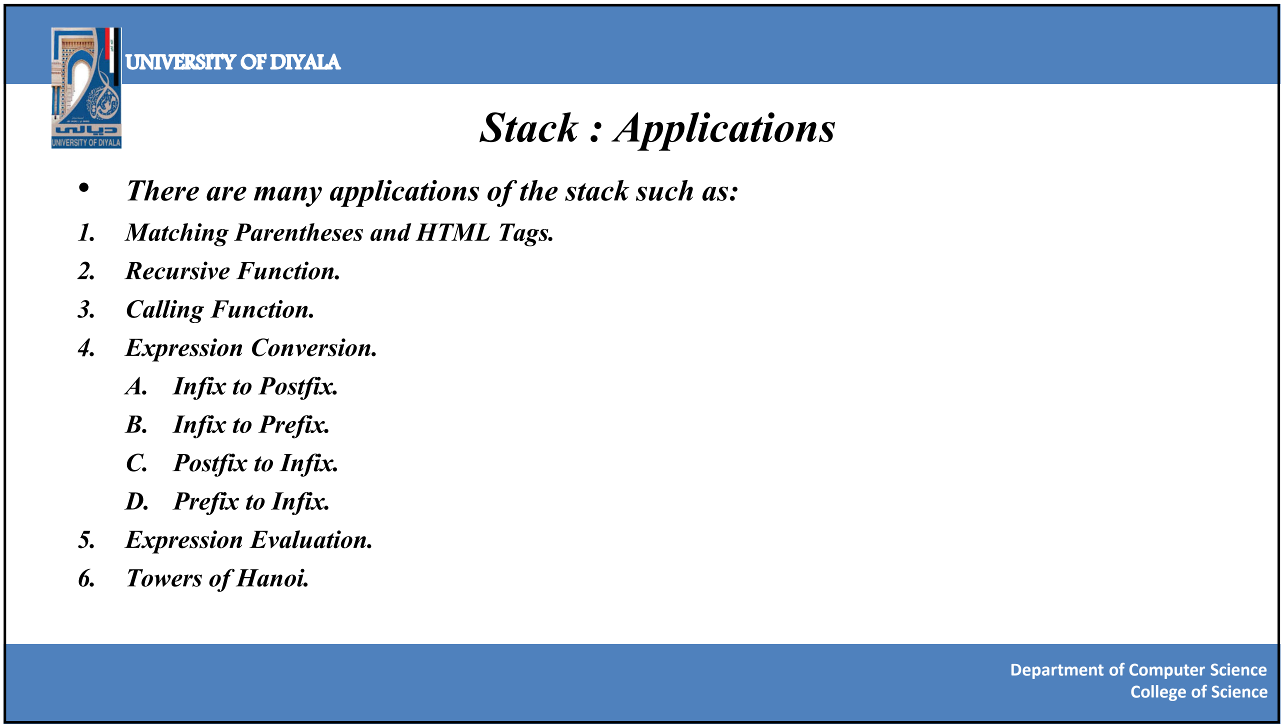
UNIVERSITY OF DIYALA

Stack : Linked List Representation

```
void pop()
{   int value;
if(top == NULL)
{  cout<<"error…linked stack is empty"<<endl;
cout<<"press any key to exit"<<endl;
  getch();   exit(0);    }
else   { q = top;
 value = q->data;
top = q->link;
 delete (q);   } }
```

q
Top
1
2
10
Link
20
Link
30
Link
40
Null

Department of Computer Science
College of Science



UNIVERSITY OF DIYALA

Stack : Applications

- *There are many applications of the stack such as:*

1. *Matching Parentheses and HTML Tags.*
2. *Recursive Function.*
3. *Calling Function.*
4. *Expression Conversion.*
   - A. *Infix to Postfix.*
   - B. *Infix to Prefix.*
   - C. *Postfix to Infix.*
   - D. *Prefix to Infix.*
5. *Expression Evaluation.*
6. *Towers of Hanoi.*

Department of Computer Science
College of Science

SA: Matching Parentheses

- *Generally, the stack is very useful in situations when data have to be stored and the retrieved in reverse order.*
- *One application of the stack is in matching delimiters in a program. This is an important example because delimiter matching is part of any compiler: No program is considered correct if the delimiters are mismatched.*
- *In C++ programs. we have the following delimiters parentheses'(' and ')', square brackets '[' and ']', curly brackets '{' and '}', and comment delimiters '/*' and '*/'.*

SA: Matching Parentheses

- ***Example:*** *Let's see what happens on the stack for a typical correct string:* ***a{b(c[d]e)f}***
- *As it's read, each opening delimiter is placed on the stack. Each closing delimiter read from the input is matched with the opening delimiter popped from the top of the stack.*

Char. Read	***Stack***
a	***Empty***
{	*{*
b	*{*
(	*{(*
c	*{(*
[	*{([*
d	*{([*
]	*{(*
e	*{(*
)	*{*
f	*{*
}	

UNIVERSITY OF DIYALA

SA: Function Calls

- *When there is a function call, all the important information that needs to be saved.*
- *Stack is very useful tool used by the compiler in programs that contain sub-programs **(procedures or functions)** to keep the return addresses.*
- *When the main program call the sub-program **(function or procedure)** the main program require to store the next instruction address after the calling instruction. so the main program can execute the sub-program and properly return back to next step after calling instruction.*

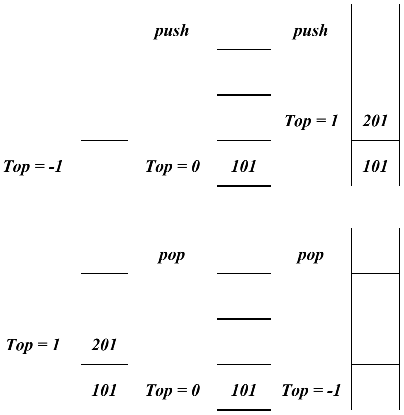
Department of Computer Science
College of Science

UNIVERSITY OF DIYALA

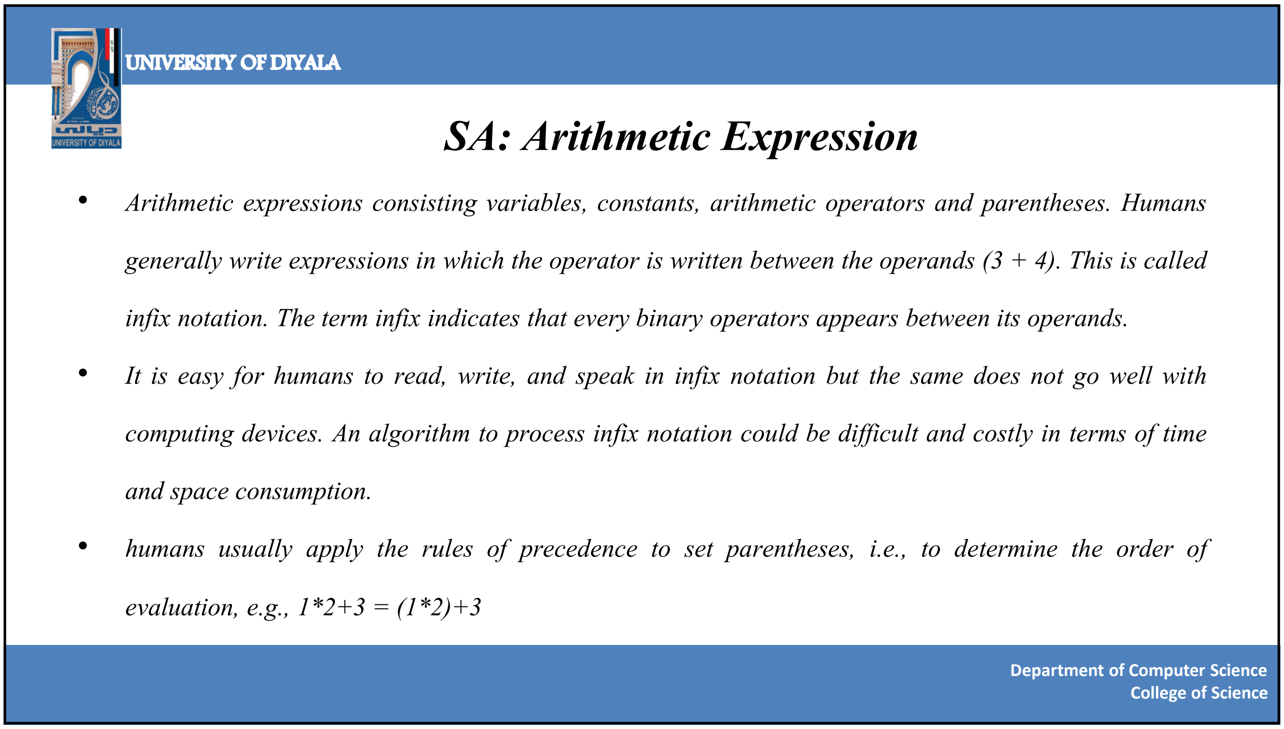
Stack Applications: Function Calls

- ***For example:*** *Suppose the following program call the number of the sub-programs*

```
 Begin     { This is the main program }
100 CALL A ───┐
101           └──→ Function (A)
---                ---
---                200 CALL B──┐
---                201         └──→ Function (B)
---                ---              -------
---                ---              -------
---                ---              --------
End             { The end of main program }
```



Department of Computer Science
College of Science

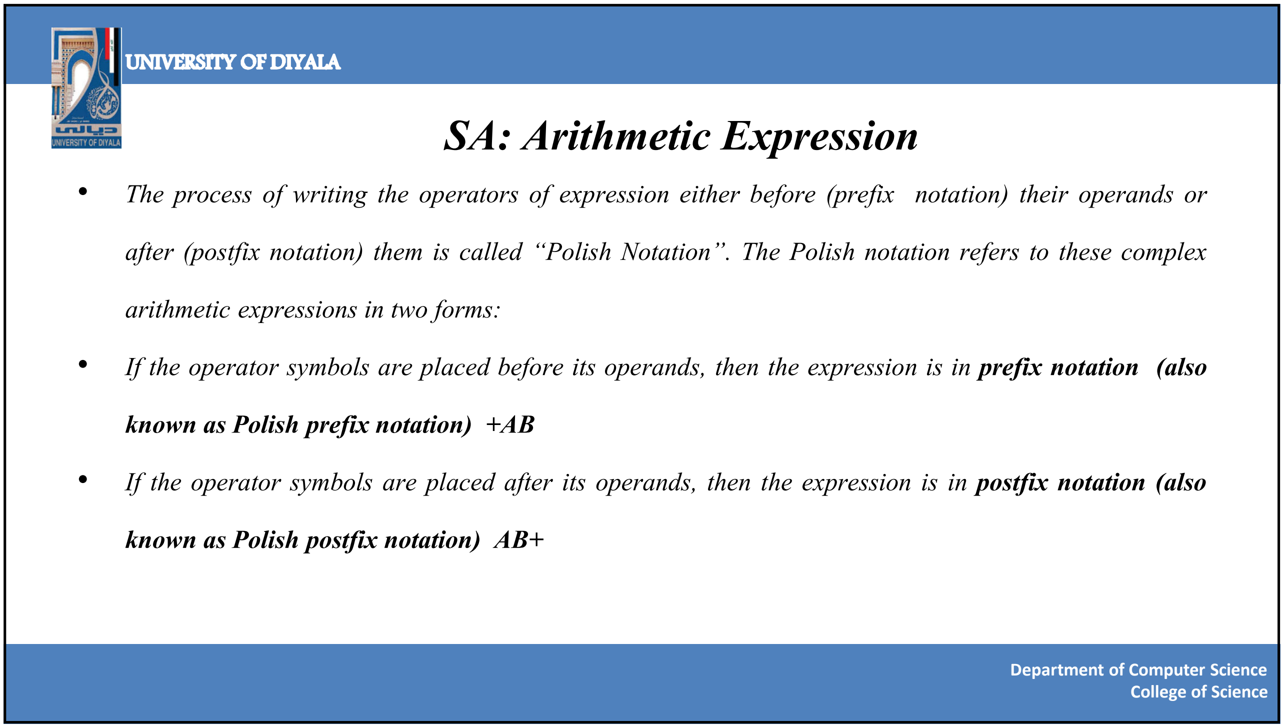


UNIVERSITY OF DIYALA

SA: Arithmetic Expression

- *Arithmetic expressions consisting variables, constants, arithmetic operators and parentheses. Humans generally write expressions in which the operator is written between the operands (3 + 4). This is called infix notation. The term infix indicates that every binary operators appears between its operands.*
- *It is easy for humans to read, write, and speak in infix notation but the same does not go well with computing devices. An algorithm to process infix notation could be difficult and costly in terms of time and space consumption.*
- *humans usually apply the rules of precedence to set parentheses, i.e., to determine the order of evaluation, e.g., 1*2+3 = (1*2)+3*

Department of Computer Science
College of Science



UNIVERSITY OF DIYALA

SA: Arithmetic Expression

- *The process of writing the operators of expression either before (prefix notation) their operands or after (postfix notation) them is called "Polish Notation". The Polish notation refers to these complex arithmetic expressions in two forms:*
- *If the operator symbols are placed before its operands, then the expression is in* ***prefix notation (also known as Polish prefix notation) +AB***
- *If the operator symbols are placed after its operands, then the expression is in* ***postfix notation (also known as Polish postfix notation) AB+***

Department of Computer Science
College of Science

UNIVERSITY OF DIYALA

SA: Arithmetic Expression

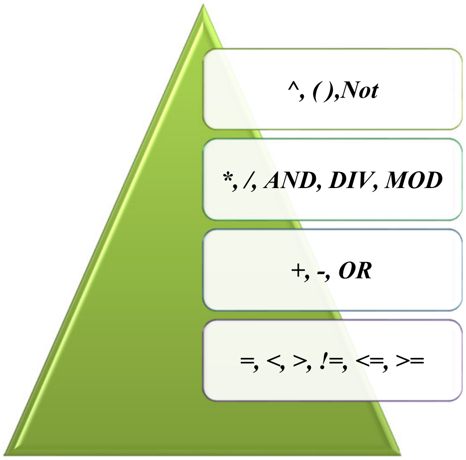
- *One of the compiler's task is to **Convert and evaluate** arithmetic expression. Example of assignment statement:* $y = x + z * (w / x + z * (7 + 6))$
- *Compiler must determine whether the right expression is a syntactically legal arithmetic expression before evaluation can be done on the expression.*
- ***Stack** is used by compilers to help in the process of converting infix to postfix arithmetic expressions and also evaluating arithmetic expressions*
- *The advantage of using prefix and postfix is that we don't need to use precedence rules, associative rules and parentheses when evaluating an expression.*

Department of Computer Science
College of Science

UNIVERSITY OF DIYALA

SA: Convert infix to Postfix

- *Generally postfix expressions are free from operator precedence that why they are preferred in computer system. Computer system uses postfix to represent expression.*
- *This process **(infix to postfix)** can be done by using one stack and output list. The following graph show the Order Of Precedence For Operators*



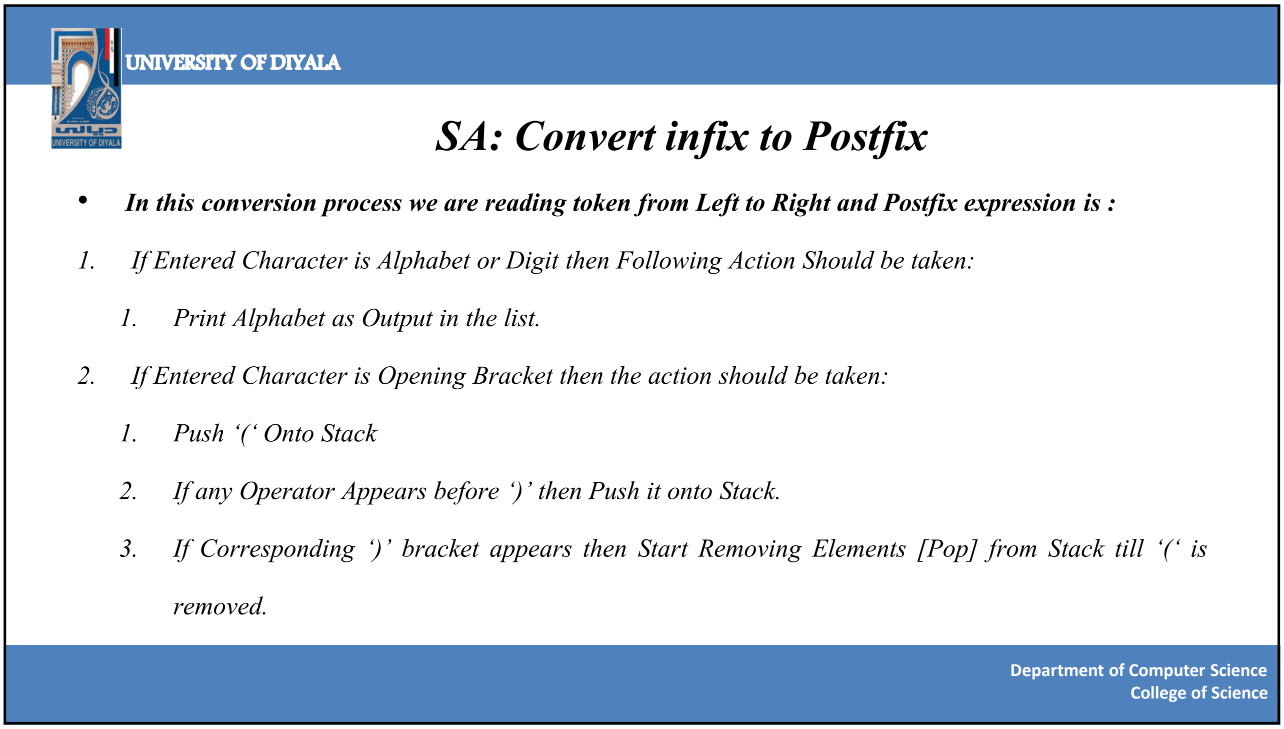
^, ( ),Not

****, /, AND, DIV, MOD***

+, -, OR

=, <, >, !=, <=, >=

Department of Computer Science
College of Science



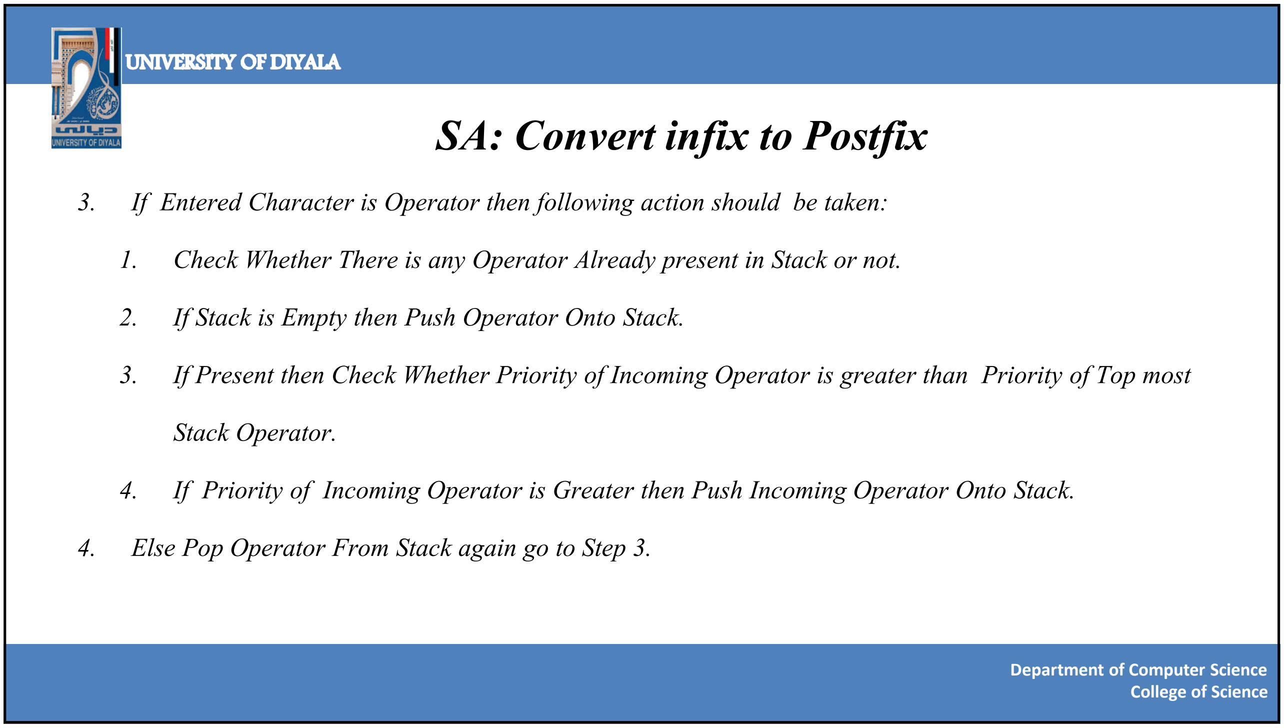
UNIVERSITY OF DIYALA

SA: Convert infix to Postfix

- ***In this conversion process we are reading token from Left to Right and Postfix expression is :***

1. *If Entered Character is Alphabet or Digit then Following Action Should be taken:*
    1. *Print Alphabet as Output in the list.*
2. *If Entered Character is Opening Bracket then the action should be taken:*
    1. *Push '(' Onto Stack*
    2. *If any Operator Appears before ')' then Push it onto Stack.*
    3. *If Corresponding ')' bracket appears then Start Removing Elements [Pop] from Stack till '(' is removed.*

Department of Computer Science
College of Science

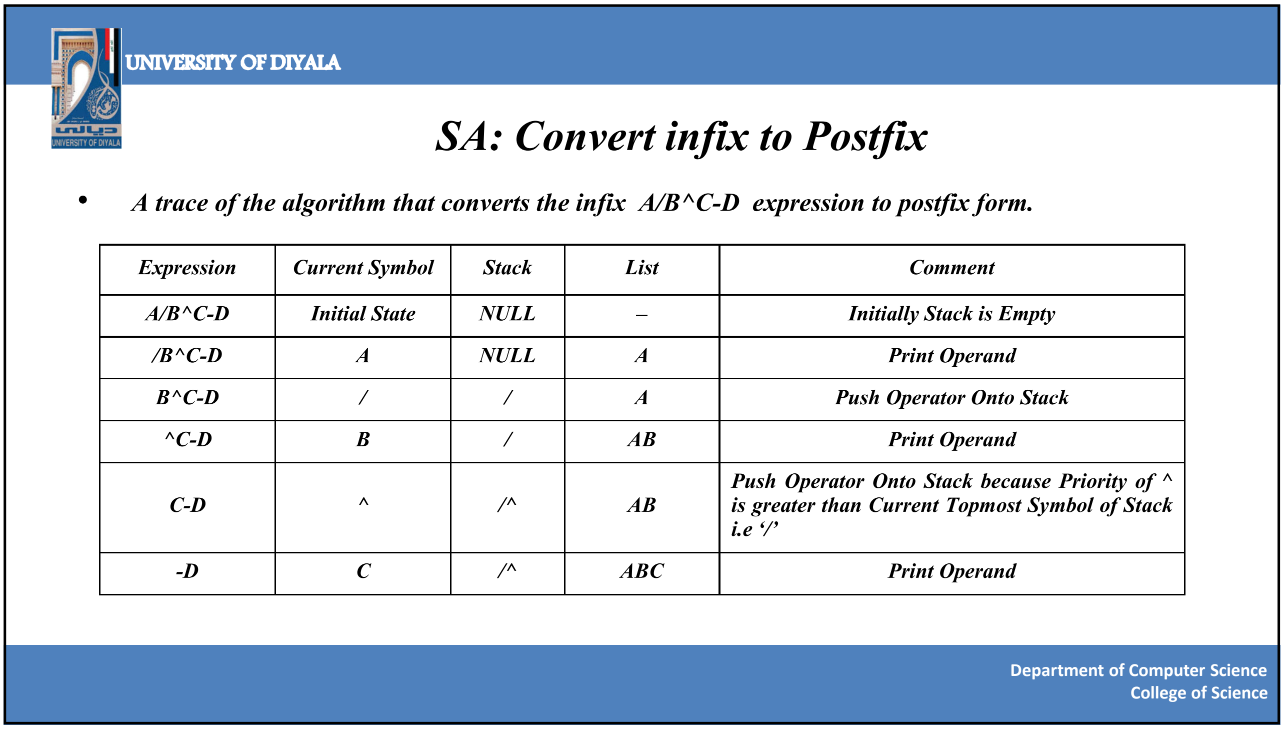


UNIVERSITY OF DIYALA

SA: Convert infix to Postfix

3. *If Entered Character is Operator then following action should be taken:*
    1. *Check Whether There is any Operator Already present in Stack or not.*
    2. *If Stack is Empty then Push Operator Onto Stack.*
    3. *If Present then Check Whether Priority of Incoming Operator is greater than Priority of Top most Stack Operator.*
    4. *If Priority of Incoming Operator is Greater then Push Incoming Operator Onto Stack.*
4. *Else Pop Operator From Stack again go to Step 3.*

Department of Computer Science
College of Science



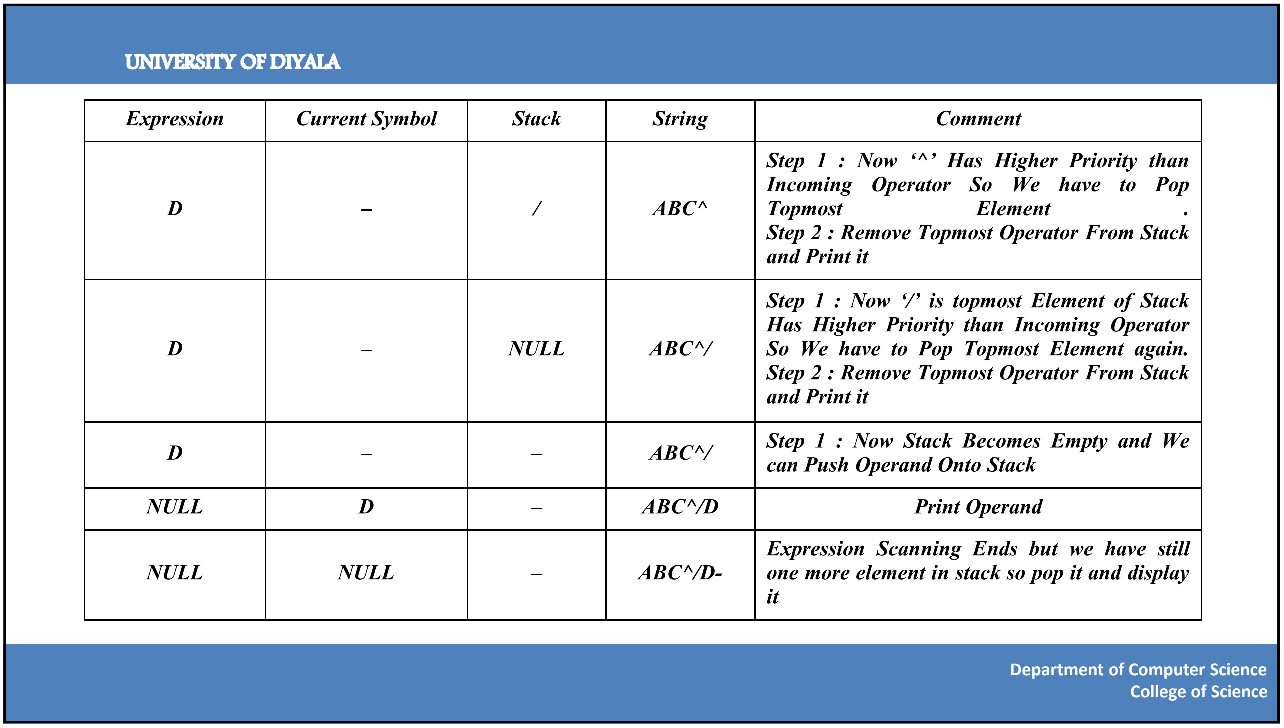
UNIVERSITY OF DIYALA

SA: Convert infix to Postfix

- *A trace of the algorithm that converts the infix A/B^C-D expression to postfix form.*

Expression	Current Symbol	Stack	List	Comment
A/B^C-D	Initial State	NULL	–	Initially Stack is Empty
/B^C-D	A	NULL	A	Print Operand
B^C-D	/	/	A	Push Operator Onto Stack
^C-D	B	/	AB	Print Operand
C-D	^	/^	AB	Push Operator Onto Stack because Priority of ^ is greater than Current Topmost Symbol of Stack i.e '/'
-D	C	/^	ABC	Print Operand

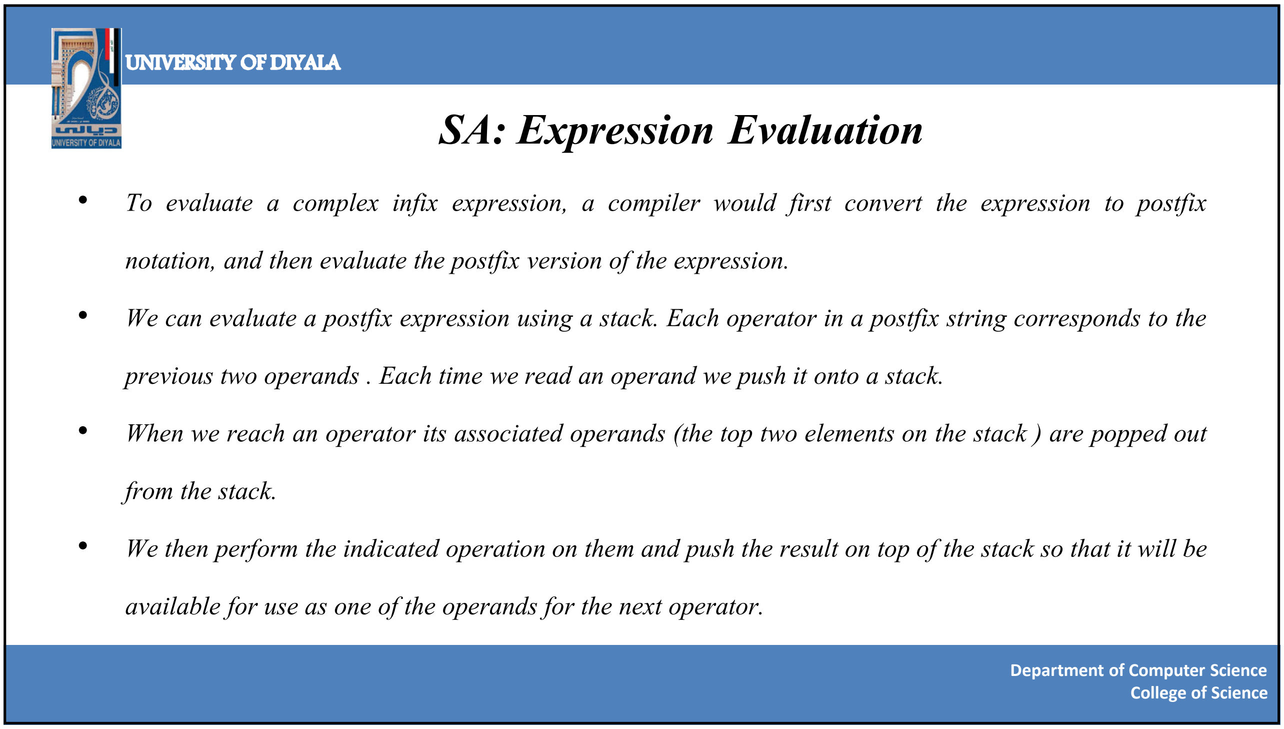
Department of Computer Science
College of Science



UNIVERSITY OF DIYALA

Expression	Current Symbol	Stack	String	Comment
D	–	/	ABC^	Step 1 : Now '^' Has Higher Priority than Incoming Operator So We have to Pop Topmost Element . Step 2 : Remove Topmost Operator From Stack and Print it
D	–	NULL	ABC^/	Step 1 : Now '/' is topmost Element of Stack Has Higher Priority than Incoming Operator So We have to Pop Topmost Element again. Step 2 : Remove Topmost Operator From Stack and Print it
D	–	–	ABC^/	Step 1 : Now Stack Becomes Empty and We can Push Operand Onto Stack
NULL	D	–	ABC^/D	Print Operand
NULL	NULL	–	ABC^/D-	Expression Scanning Ends but we have still one more element in stack so pop it and display it

Department of Computer Science
College of Science

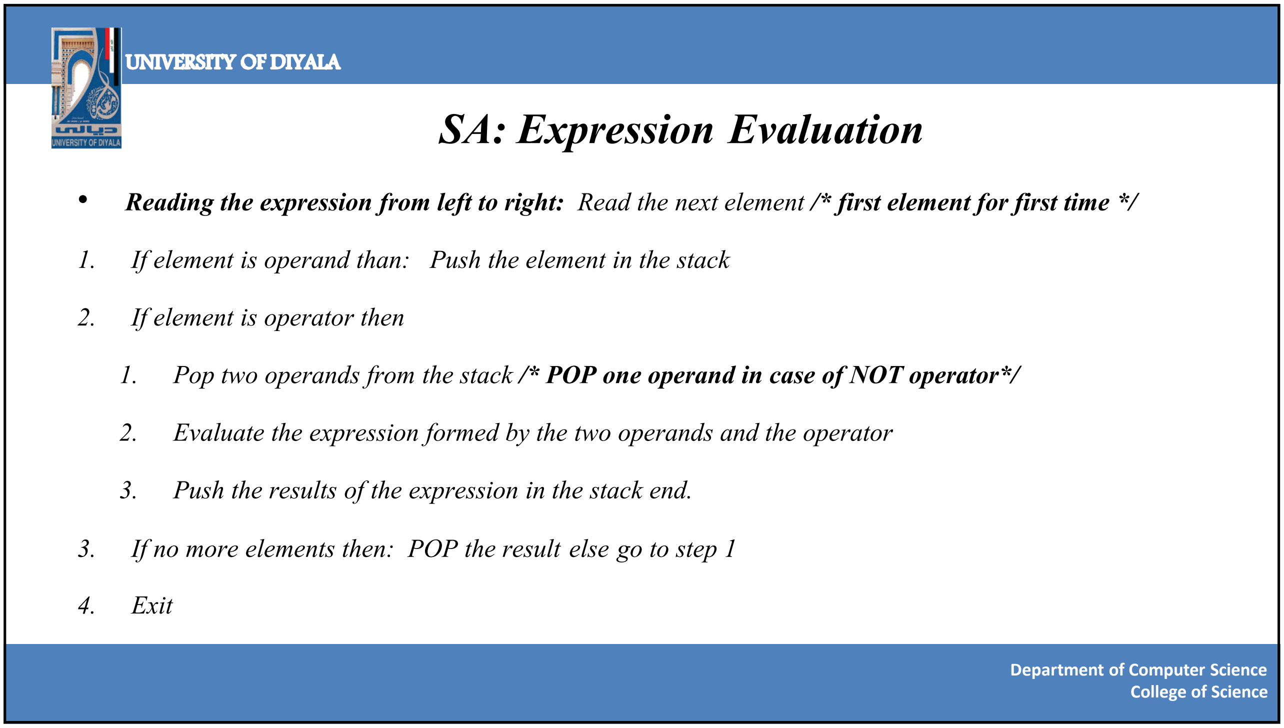


UNIVERSITY OF DIYALA

SA: Expression Evaluation

- *To evaluate a complex infix expression, a compiler would first convert the expression to postfix notation, and then evaluate the postfix version of the expression.*
- *We can evaluate a postfix expression using a stack. Each operator in a postfix string corresponds to the previous two operands . Each time we read an operand we push it onto a stack.*
- *When we reach an operator its associated operands (the top two elements on the stack) are popped out from the stack.*
- *We then perform the indicated operation on them and push the result on top of the stack so that it will be available for use as one of the operands for the next operator.*

Department of Computer Science
College of Science



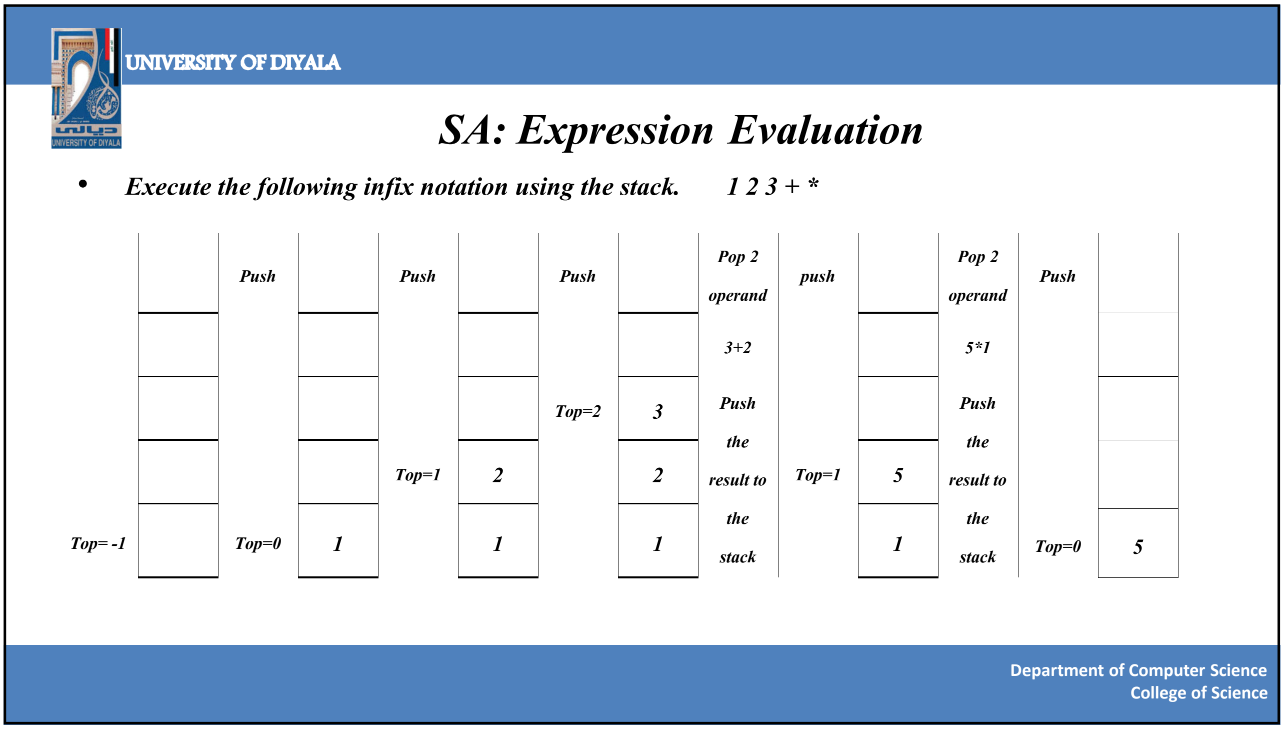
UNIVERSITY OF DIYALA

SA: Expression Evaluation

- ***Reading the expression from left to right:*** *Read the next element /* ***first element for first time*** **/*

1. *If element is operand than: Push the element in the stack*
2. *If element is operator then*
    1. *Pop two operands from the stack /** ***POP one operand in case of NOT operator*/***
    2. *Evaluate the expression formed by the two operands and the operator*
    3. *Push the results of the expression in the stack end.*
3. *If no more elements then: POP the result else go to step 1*
4. *Exit*

Department of Computer Science
College of Science



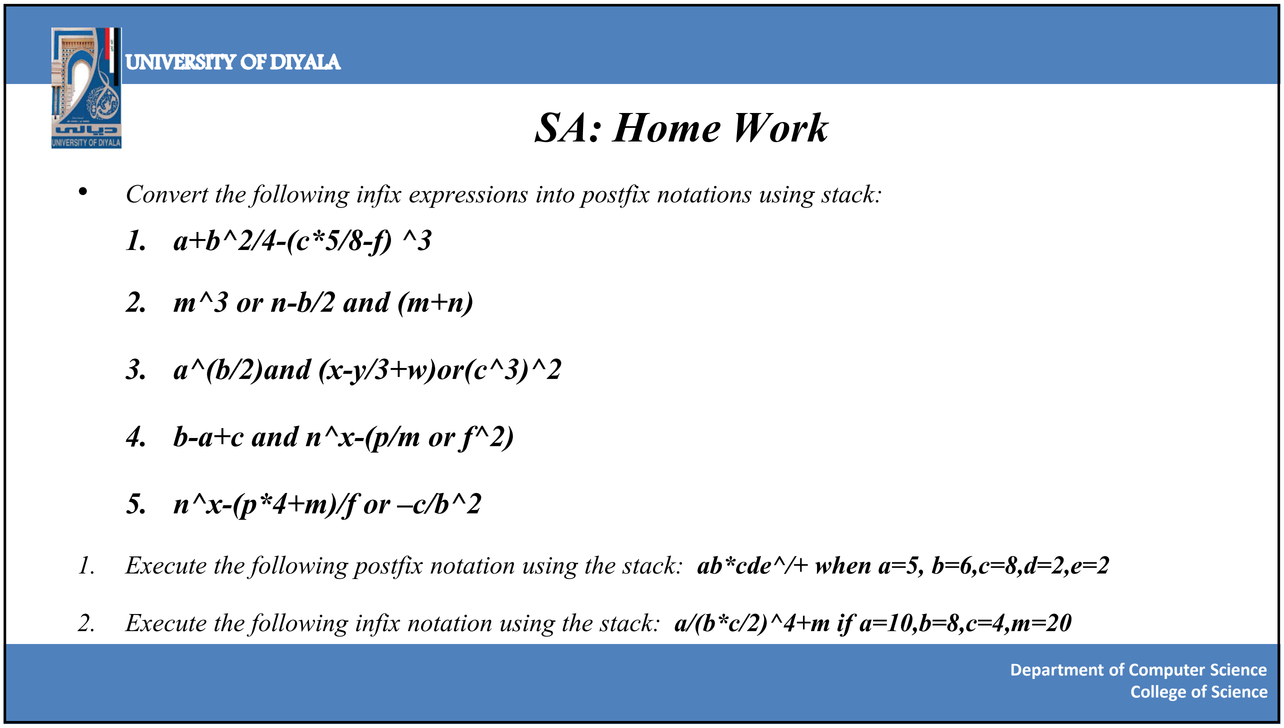
UNIVERSITY OF DIYALA

SA: Expression Evaluation

- *Execute the following infix notation using the stack.* *1 2 3 + **

		Push		*Push*		*Push*		*Pop 2* *operand*	*push*		*Pop 2* *operand*	*Push*	
								3+2			*5*1*		
						Top=2	*3*	*Push* *the*			*Push* *the*		
				Top=1	*2*		*2*	*result to*	*Top=1*	*5*	*result to*		
Top= -1		*Top=0*	*1*		*1*		*1*	*the* *stack*		*1*	*the* *stack*	*Top=0*	*5*

Department of Computer Science
College of Science



UNIVERSITY OF DIYALA

SA: Home Work

- *Convert the following infix expressions into postfix notations using stack:*
  1. ***a+b^2/4-(c*5/8-f) ^3***
  2. ***m^3 or n-b/2 and (m+n)***
  3. ***a^(b/2)and (x-y/3+w)or(c^3)^2***
  4. ***b-a+c and n^x-(p/m or f^2)***
  5. ***n^x-(p*4+m)/f or –c/b^2***

1. *Execute the following postfix notation using the stack:* ***ab*cde^/+ when a=5, b=6,c=8,d=2,e=2***
2. *Execute the following infix notation using the stack:* ***a/(b*c/2)^4+m if a=10,b=8,c=4,m=20***

Department of Computer Science
College of Science

UNIVERSITY OF DIYALA

The End

Department of Computer Science
College of Science