CHAPTER 8

STACK

Nor Bahiah Hj Ahmad & Dayang Norhayati A.Jawawi
School of Computing

Objectives

At the end of the lesson students are expected to be able to:

- Understand stack concepts and the structure.
- Understand operations that can be done on stack.
- Understand and know how to implement stack using array and linked list.

Introduction to Stack

- Stack is a collection of items which is organized in a sequential manner.
- Example: stack of books or stack of plates.
- All additions and deletions are restricted at one end, called top.
- LAST IN FIRST OUT (LIFO) data structure.

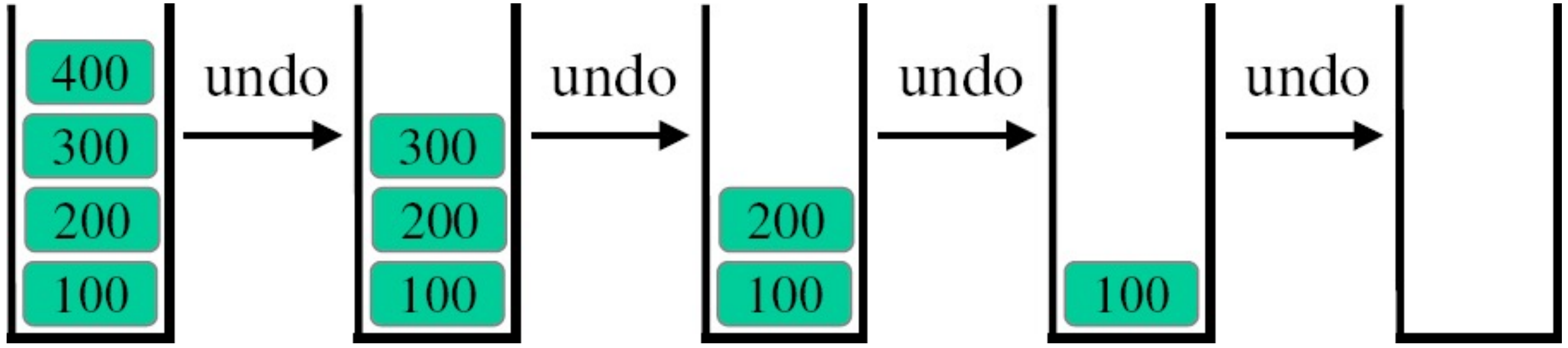
Example of Stack

- Long and narrow driveway
  – BMW comes in first, Lexus follows, Benz enters in last.
  – When the cars come out, Benz comes out first, then Lexus follows, and BMW comes out last.
- Reverse a string
  Example: TOP , Reverse: POT
- Brackets balancing in compiler
- Page-visited history in a Web browser
- Undo operation in many editor tools
- Check nesting structure

More Example: *undo* operation

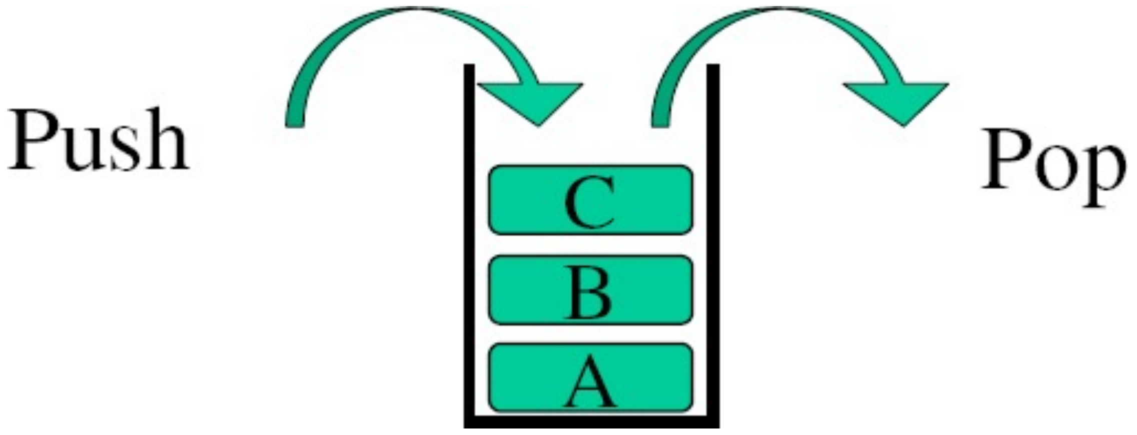
- In an Excel file, input your data in row
  - 100, 200, 300, 400
- Find something wrong, use undo, return to previous state

Operation lists

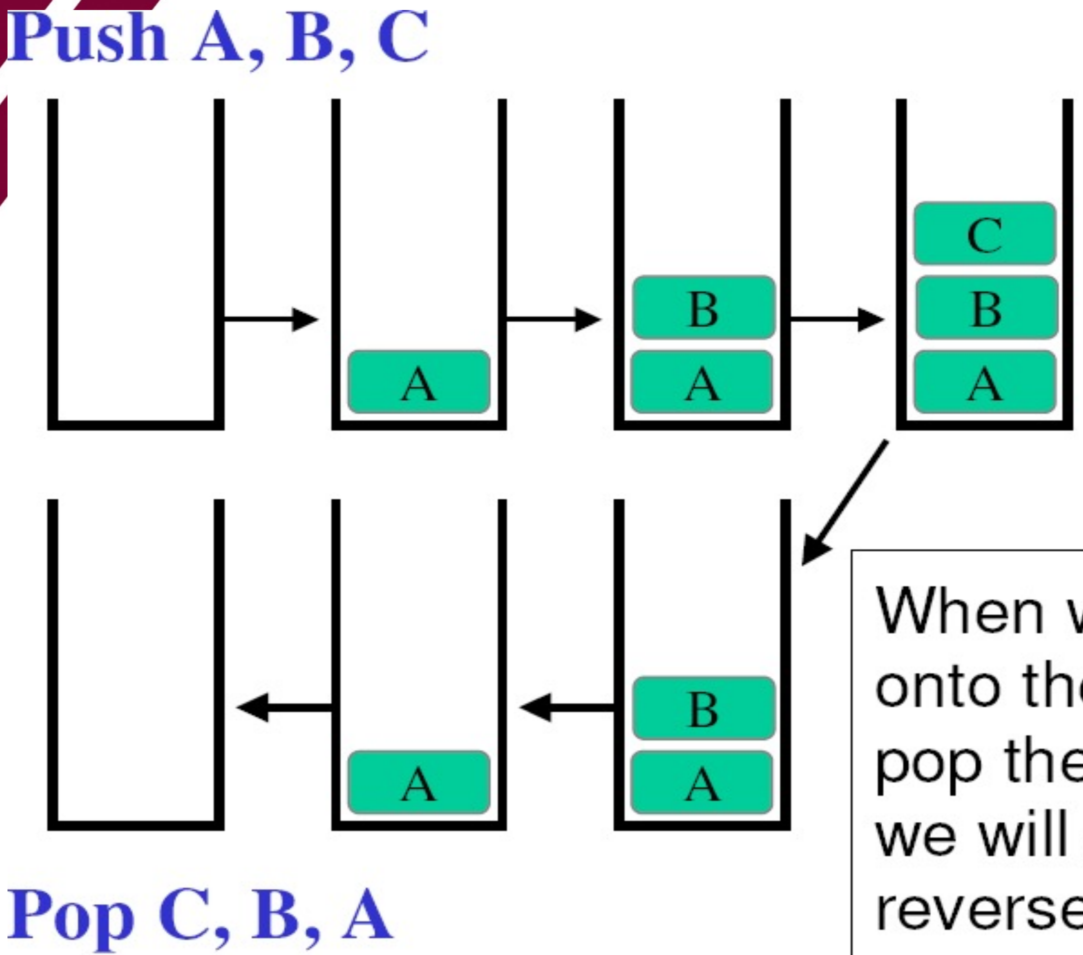


What Is Stack?

- Stack is an abstract data type
- Adding an entry on the top is called push
- Deleting an entry from the top is called pop
- A stack is open at one end only(the top). You can push entry onto the top, or pop the top entry out of the stack.



Last-in First-out (LIFO)



The last one pushed in is the first one popped out! (LIFO)

When we push entries onto the stack and then pop them out one by one, we will get the entries in reverse order.

Stack implementation

- Stack is an abstract data structure
- Item can be Integer, Double, String, and also can be any data type, such as Employee, Student…
- How to implement a general stack for all those types?
- We can implement stack using array or linked list.

Implementation for Stack

Array

- **Size of stack is fixed during declaration**
- Item can be pushed if there is some space available, need isFull( ) operations.
- Need a variable called, top to keep track the top of a stack.
- Stack is empty when the value of Top is –1.

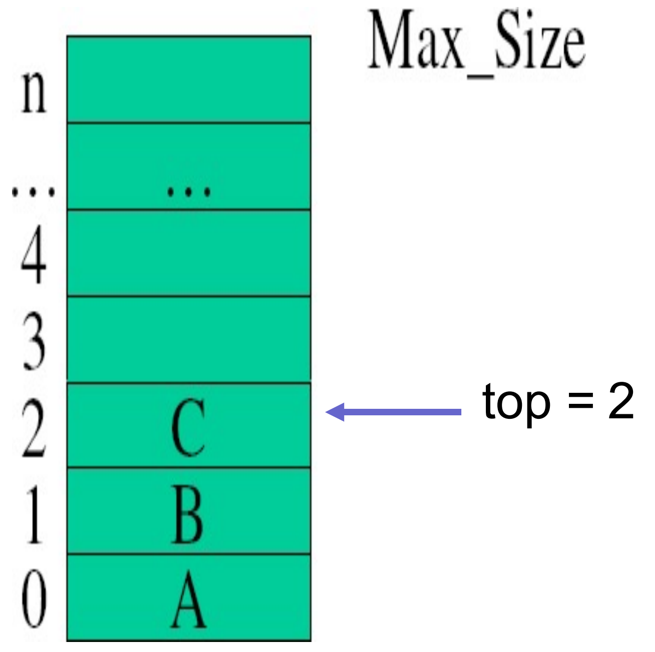
Linked List

- **Size of stack is flexible**. Item can be pushed and popped dynamically.
- Need a pointer, called top to point to top of stack.

Array Implementation of Stack

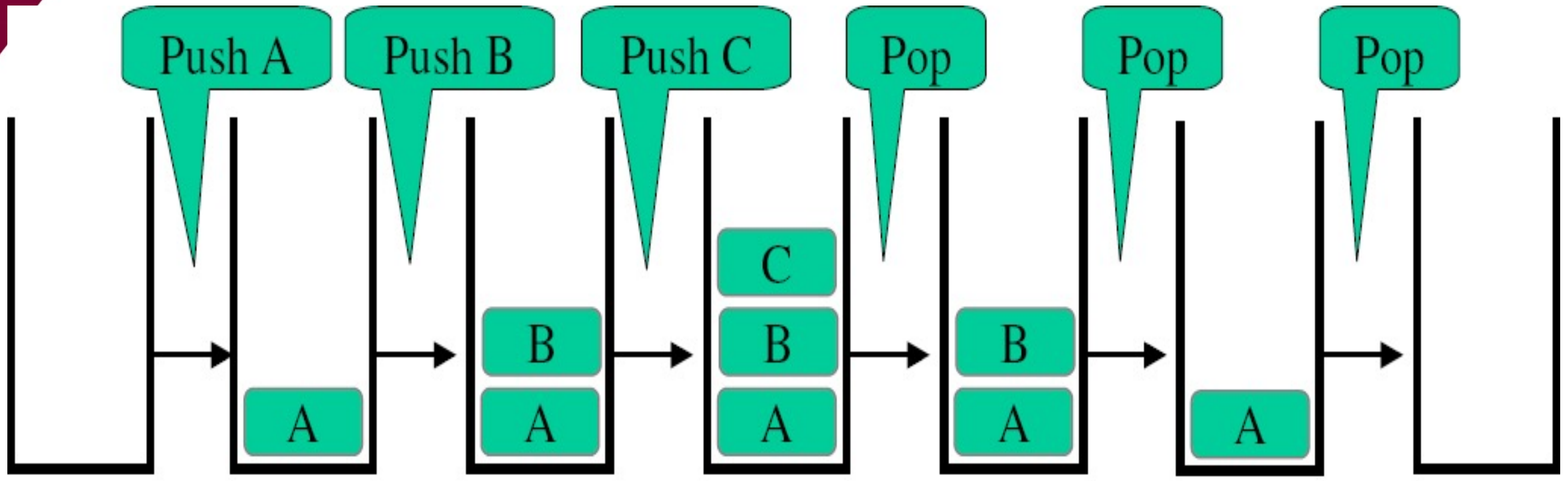
Stack Operations:

- createStack()
- push(item)
- pop( )
- isEmpty( )
- isFull( )
- stackTop()



"Stack can be visualized as array, BUT the operations can be done on top stack only. "

Push() and pop() operations



Top:

-1	0	1	2	1	0	-1
stack empty						**stack empty**

Array Implementation of Stack

3 things to be considered for stack with array

1. **Stack Empty :** when top is -1.
2. **Push operations:** To insert item into stack

   2 statements must be used

   ```
   top = top + 1;
   stack[top] = newitem;
   ```
3. **pop operations:** To delete item from stack.

   2 statements should be used

   Item = stack[top]; or stackTop();
   top = top – 1;

- `Item = stack[top];` statement is needed if we want to check the value to be popped.

Array Implementation of Stack

Stack declaration:

```
const int size = 100;
class stack
{
  private : // data declaration
    int top ;
    char data[size] ;
  public : // function declaration
  void createStack();
  void push(char) ;     // insert operation
  void pop() ;      // delete operation
  char stackTop() ; // get top value
  bool isFull() ; // check if stack is Full
  bool isEmpty(); // check if stack is empty
} ;
```

Array Implementation of Stack

- We need two data attributes for stack:
  1. **Data :** to store item in the stack, in this example data will store char value
  2. **top** : as index for top of stack, integer type
- Size of the array that store component of stack is 100. In this case, stack can store up to 100 char value.
- Declaration of stack instance:

  stack aStack;

Array Implementation of Stack

`createStack()` operation

- Stack will be created by initializing top to -1.
- `createStack()` implementation:

```
void stack:: createStack();
{
        top = -1;
}
```

- Top is **–1 :- means that there is no item being pushed into stack yet.**

Array Implementation of Stack

`isFull()` Operation

- This operation is needed ONLY for implementation of stack using array.
- In an array, size of the array is fixed and to create new item in the array will depend on the space available.
- This operation is needed before any push operation can be implemented on a stack.
- `bool isFull()` implementation

```
bool stack::isFull()
{
      return (top == size-1 );
}
```

- Since the size of the array is 100,

  `bool isFull()` will return ***true***, If top is 99 (100 – 1).

  `bool isFull()` will return *false*, if there is some space available, top is less than 100.

Array Implementation of Stack

`bool isEmpty()` operation

- This operation will check whether the array for stack is empty.
- This operation is needed before ANY pop operation can be done. If the stack is empty, then pop operation cannot be done.
- `bool isEmpty()` will return *true* if top –1 and return *false* if top is not equal to -1, showing that the stack has element in it.
- `bool isEmpty()` implementation :

```
bool stack::isEmpty()
{
      return ( top == -1 );
}
```

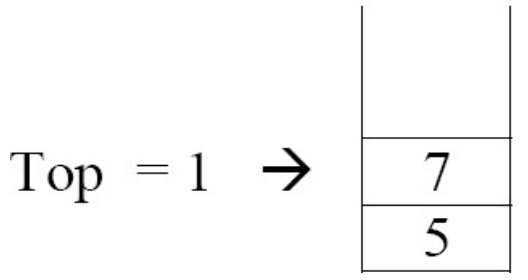
Array Implementation of Stack

`push(newItem)` **operation : Insert item onto stack**

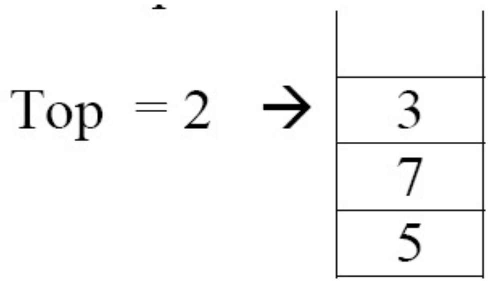
- **push()** operation will insert an item at the top of stack. This operation can be done only if there is space availbale in the array
- Before any item can be inserted into a stack, isFull() operation must be called first.
- Insertion operation involve the following steps:
  - Top will be increased by 1.

    top = top + 1;
  - New item will be inserted at the top

    data[Top] = newItem;



before push()



after push()

Array Implementation of Stack

```
void stack::push(char newitem)
{
  if (isFull()) // check whether stack is full
     cout << "Sorry,Cannot push item.
              Stack is now full!"<< endl;
  else
  {  top = top + 1       // Top point to next index
     data[top] = newitem; //assign new item at top
   }//end else
}//end push()
```

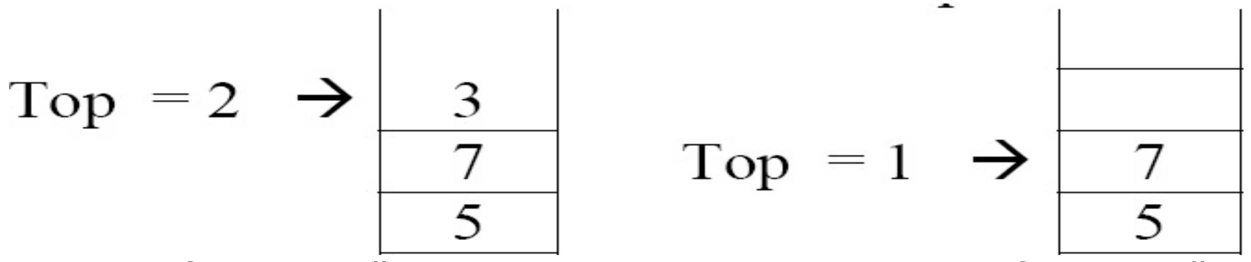
Top will be increased first before item is inserted in order to avoid inserting item at the current top value.

Array Implementation of Stack

pop() Operation

- This operation will delete an item at top of scak.
- Function isEmpty() will be called first in order to ensure that there is item in a stack to be deleted.
- pop() operation will decrease the value of top by 1:

```
top = top - 1;
```



Before pop()   after pop()

Array Implementation of Stack

```
void stack::pop()
{
   char item;
   if ( isEmpty() )
        cout << "Sorry, Cannot pop item.
                        Stack is empty!" << endl;
  else
   { //display value at top to be deleted
        cout << "Popped value :" << data[top];
        top = top - 1;
         // top will hold to new index
   }// end if
}//end pop
```

Array Implementation of Stack

`stackTop()` **operation : to get value at the top**

```
char stackTop()
{ //function to get top value
   if (isEmpty())
      cout <<"Sorry, stack is empty!"<< endl;
   else
      return data[top];
} // end stackTop
```

Linked List Implementation of Stack

- Stack implemented using linked list – number of elements in stack is not restricted to any size.
- Dynamic memory creation, memory will be assigned to stack when a new node is pushed into stack, and memory will be released when an element being popped from the stack.
- Stack using linked list implementation can be empty or contains a series of nodes.
- Each node in a stack must contain at least 2 attributes:
- i) **data** – to store information in the stack.
- ii) pointer next (store address of the next node in the stack

Linked List Implementation of Stack

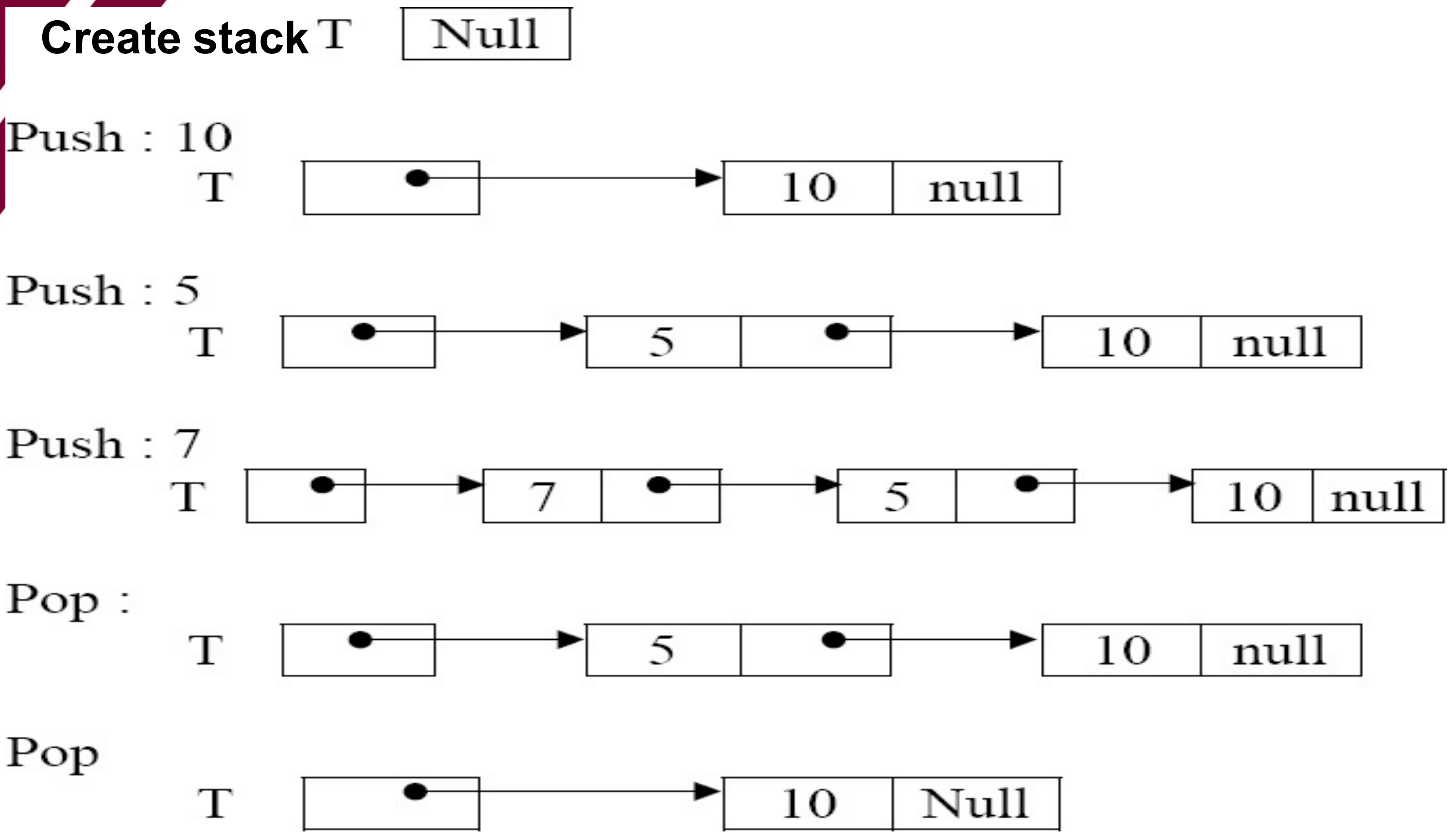
Basic operations for a stack implemented using linked list:

- `createStack()` – initialize top
- `push(char)` – insert item onto stack
- `pop()` – delete item from stack
- `isEmpty()` – check whether a stack is empty.
- `stackTop()` – get item at top

`isFull()` operation is not needed since elements can be inserted into stack without limitation to the stack size.

- Push and pop operations can only be done at the top ~ similar to add and delete in front of the linked list.

Linked List Implementation of Stack: push() and pop() operations



Linked List Implementation of Stack

```
class nodeStack
{
   int data;
   nodeStack *next;
};
class stack
{
    private: // pengisytiharan ahli data
       nodeStack *top;
    public : // pengisytiharan ahli fungsi
       void createStack(); // set Top to NULL
       void push(int) ; // insert item into stack
       void pop() ; // delete item from stack
       int stackTop() ; // get content at top stack
       bool isEmpty(); // check whether stack is empty
};
```

Create Stack and isEmpty()

- Creating a stack will initialize top to NULL - showing that currently, there is no node in the stack.

```
void stack::createStack()
{
     top = NULL;
}
```

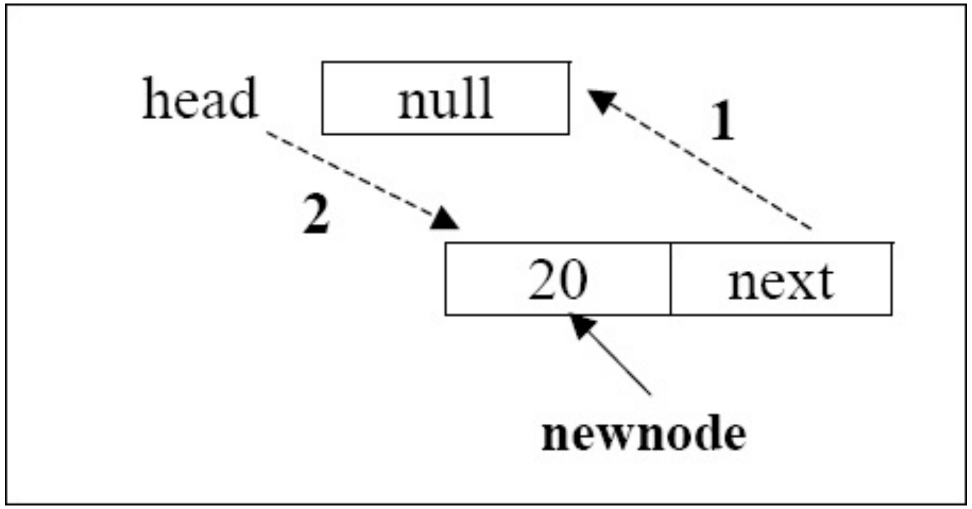
- Is Empty() stack will return true if stack is empty, top is NULL.

```
  bool stack::isEmpty()
  {
   return (top == NULL);
  }
```

`push()` operations

- 2 conditions for inserting element in stack:
  - Insert to empty stack.
  - Insert item to non empty stack : stack with value.

`push()` to empty stack



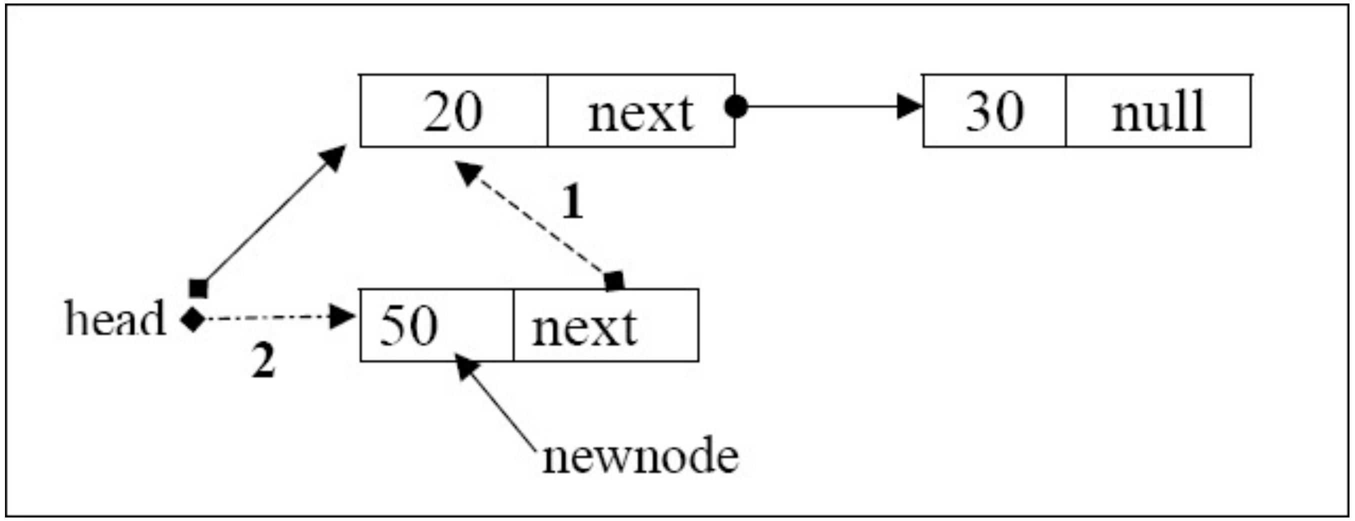
In this situation the new node being inserted, will become the first item in stack.

STEP 1 : newnode-> next = head;

STEP 2 : head = newnode;

`push()` to non-empty stack

This operation is similar to inserting element in front of a linked list. The next value for the new element will point to the top of stack and head will point to the new element.



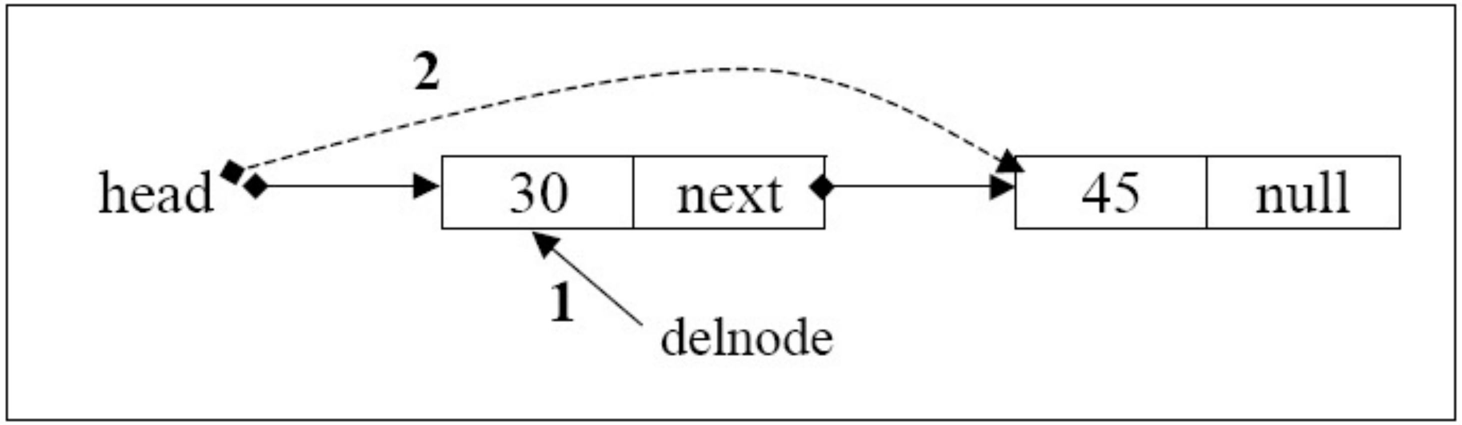
STEP 1 : newnode-> next = head;
STEP 2 : head = newnode;

Push() operations

```
void stack::push(int newitem)
{ // create newnode
  nodeStack *newnode;
  newnode = new (nodeStack);
  if( newnode == NULL)
     cout << “Cannot allocate memory…” << endl;
  else // add to empty stack, or to front stack
  { newnode->data = newitem;
    newnode->next = head;
    head = newnode;
  }// end if
} //end push operation
```

Delete item from stack (pop)

- Pop operation can only be done to non-empty stack. Before pop() operation can be done, operation must be called in order to check whether the stack is empty or there is item in the stack. If `isEmpty()` function return true, pop() operation cannot be done.
- During pop() operation, an external pointer is needed to point to the delete node. In the figure below, `delnode` is the pointer variable to point to the node that is going to be deleted.



STEP 1 : delnode = head;
STEP 2 : head = delnode -> next; or **head = head->next;**
STEP 3 : delete(delnode);

Pop() operation

```
void stack::pop()
{   nodeStack *delnode;
    if (isEmpty())
      cout <<"Sorry, Cannot pop item from
          stack.Stack is still empty!" <<endl;
    else
    { delnode = head;
      cout << "Item to be popped from stack
                is: " << stackTop()<<endl;
      head = delnode->next;
      delete(delnode);
     }// end else
} // end pop
```

Check item at top stack

```
int stack::stackTop()
{  if (isEmpty())
       cout <<"Sorry,Stack is still empty!"
            <<endl;
   else
        return head->data;
} // end check item at top
```

End of Stack Implementation

What we have learned so far….

- Stack is a LIFO data structure
- Can be implemented using array and link list
- Structure of a stack using array and link list
- Basic Operation for a stack
  - `createStack(),push(),pop()`
  - `stackTop(),isEmpty(),isFull()`

Next…..Stack Application

Stack Application Examples

- Check whether parantheses are balanced (open and closed parantheses are properly paired)
- Evaluate Algebraic expressions.
- Creating simple Calculator
- Backtracking (example. Find the way out when lost in a place)

Example1: Parantheses Balance

- Stack can be used to recognize a balanced parentheses.
- Examples of balanced parentheses.

  (a+b), (a/b+c), a/((b-c)*d)

  Open and closed parentheses are properly paired.
- Examples of not balance parentheses.

  ((a+b)*2 and m*(n+(k/2)))

  Open and closed parentheses are not properly paired.

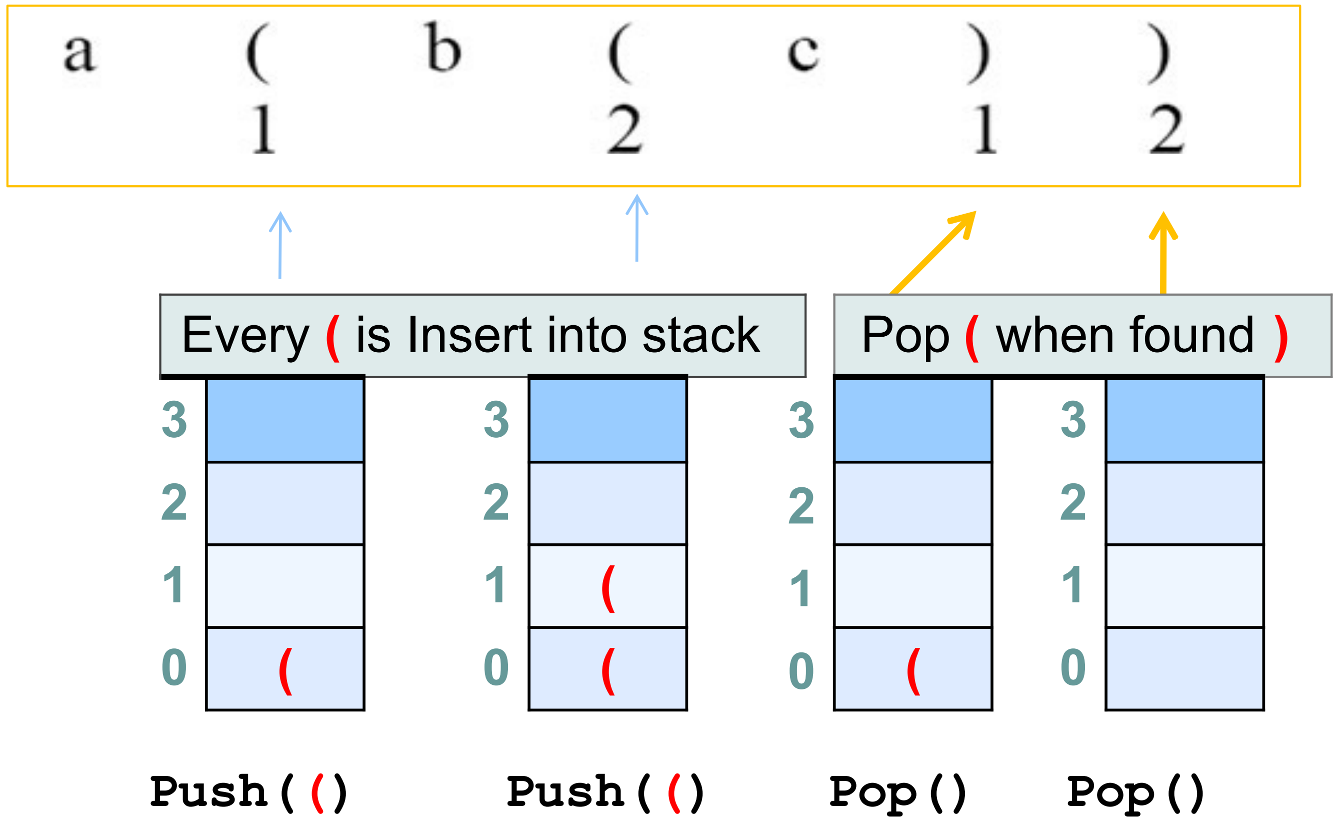
Check for Balanced Parantheses Algorithm

```
create (stack);
continue = true;
while ( not end of input string) && continue
{     ch = getch( );
      if ch = '(' || ch = ')'
        { if ch = '('
              Push(stack, '(');
          else if IsEmpty(stack)
            continue = false;
          else
             Pop(s);
       } // end if
} // end while
if ( end of input && isEmpty(stack);
    cout << "Balanced.." << endl;
else
    cout << "Not Balanced.. " << endl;
```

Check for Balanced Parantheses Algorithm

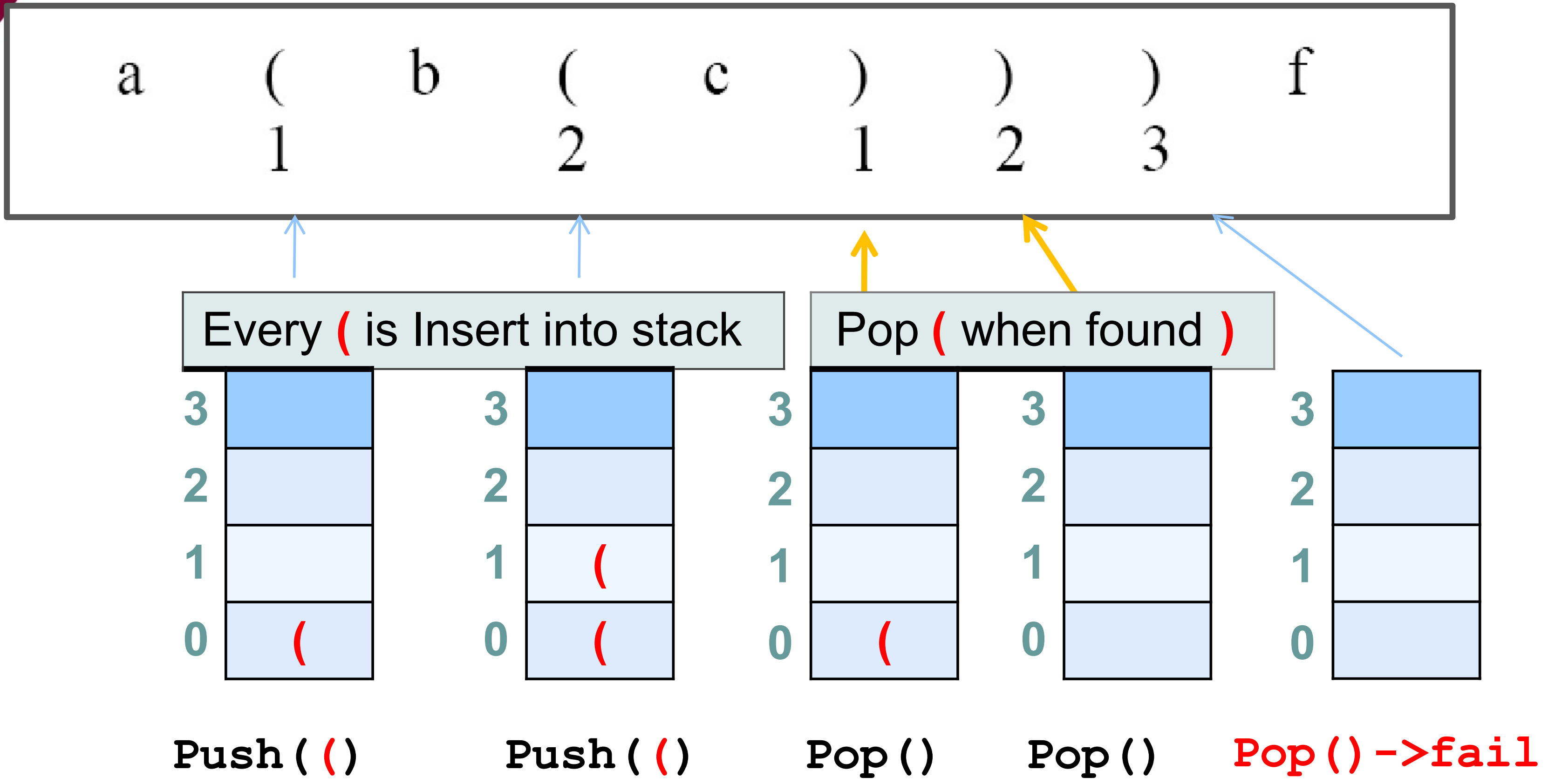
- Every ‘(’ read from a string will be pushed into stack.
- The open parentheses ‘(’ will be popped from a stack whenever the closed parentheses ‘)’ is read from string.
- An expression have balanced parentheses if :
  - Each time a “)” is encountered it matches a previously encountered “(“.
  - When reaching the end of the string, every “(“ is matched and stack is finally empty.
- An expression does NOT have balanced parentheses if :
  - When there is still ‘)’ in input string, the stack is already empty.
  - When end of string is reached, there is still ‘(‘ in stack.

Example for Balance Parantheses



Expression **a(b(c))** have balance parentheses since when end of string is found the stack is empty.

Example for ImBalanced Parantheses



Expression a(b(c))) f does not have balance parentheses => the third) encountered does not has its match, the stack is empty.

Algebraic expression

- One of the compiler's task is to evaluate algebraic expression.
- Example of assignment statement:

  y = x + z * (w / x + z * (7 + 6))

- Compiler must determine whether the right expression is a syntactically legal algebraic expression before evaluation can be done on the expression.
- 3 algebraic expressions are :

  Infix, prefix and postfix

Infix Expression

- The algebraic expression commonly used is infix.
- The term infix indicates that every binary operators appears between its operands.
- Example

A	+	B
operan	operator	operan

- Example 2: A + B * C $\rightarrow$

A + (B * C)
(a + b) * c

- To evaluate infix expression,

  the following rules were applied:

1. Precedence rules.
2. Association rules (associate from *left to right*)
3. Parentheses rules.

Prefix and Postfix Expressions

- Alternatives to infix expression
- Prefix : Operator appears before its operand.
- Example:

```
+ a b
+ a * b c
* + a b c
```

- Postfix : Operator appears after its operand.
- Example:

```
a b +
a b c * +
a b + c *
```

Infix, prefix and postfix

Infix	Prefix	Postfix
a + b	+ a b	a b +
a + (b * c)	+ a * b c	a b c * +
(a + b) * c	* + a b c	a b + c *

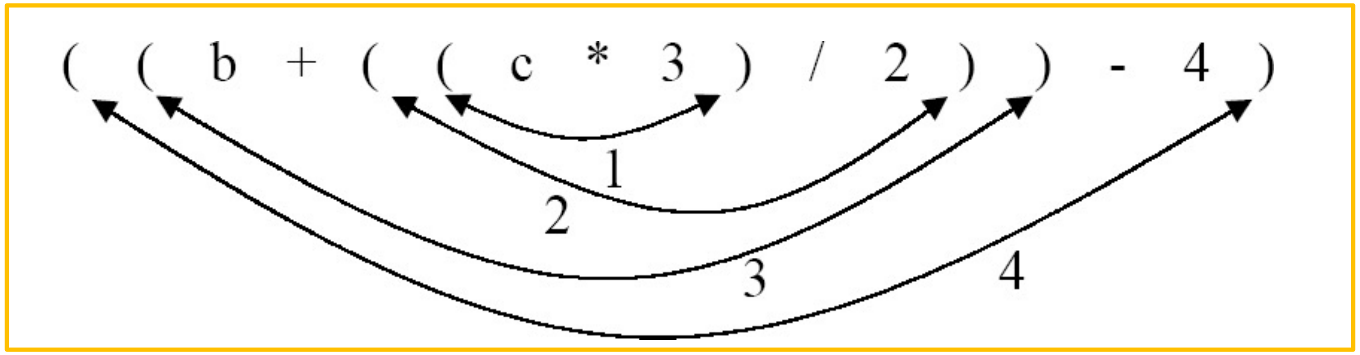
The advantage of using prefix and postfix is that we don’t need to use precedence rules, associative rules and parentheses when evaluating an expression.

Converting Infix to Prefix

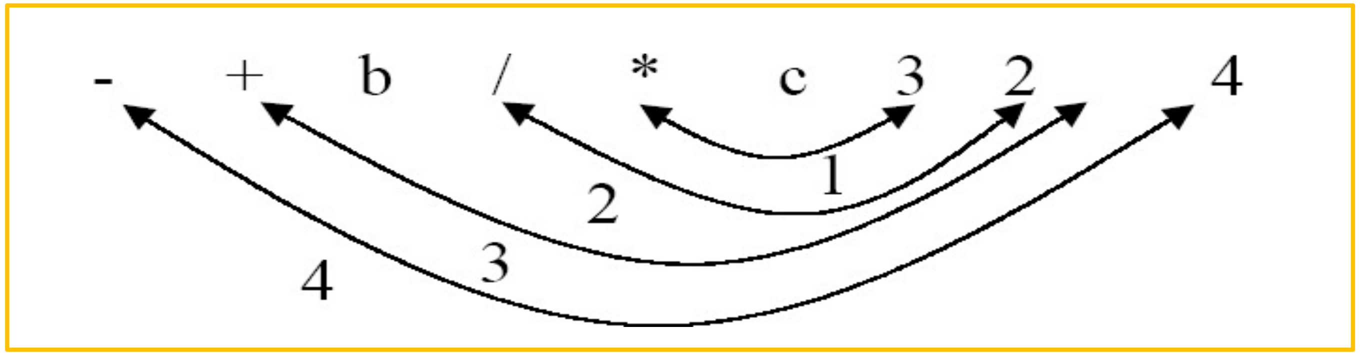
Steps to convert infix expression to prefix:

b + c * 3 / 2 - 4

STEP 1 : Determine the precedence.



STEP 2



Convertin

Another Example:

1. $a + b \quad = \quad (a + b)$
   $\quad = \quad + a\, b$

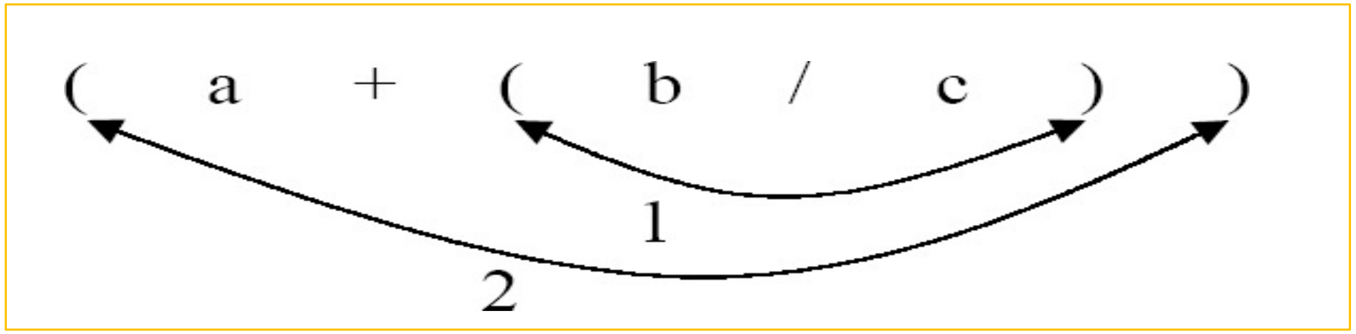
2. $a + (b * c) \quad = \quad (a + (b * c))$
   $\quad = \quad + a * b\, c$

Converting Infix to Postfix

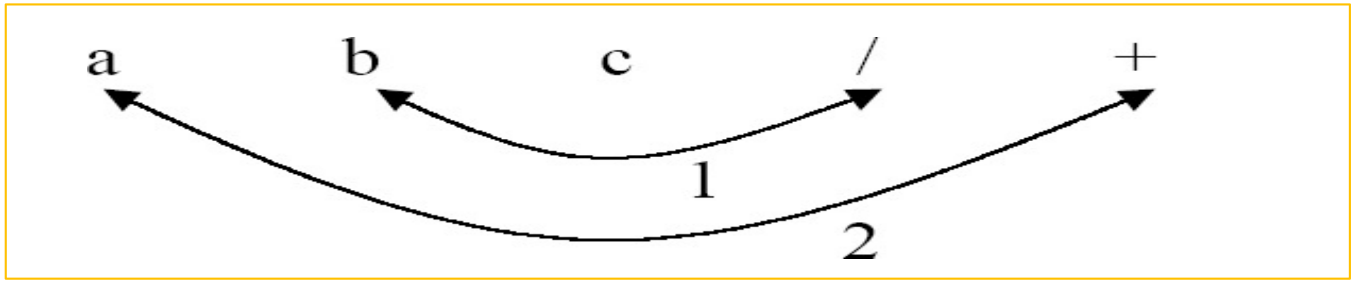
Steps to convert infix expression to postfix:

a + b / c

STEP 1



STEP 2



Converting Infix to Postfix

1. $a + b \quad = \quad (a + b)$

   $\quad = \quad a\ b\ +$

2. $a + b * c \quad = \quad (a + (b * c))$

   $\quad = \quad a\ b\ c\ *\ +$

3. $a + b * (c - d)\ /\ (p - r)$

   $= \quad a + (b * (c - d))\ /\ (p - r)$

   $= \quad (a + ((b * (c - d))\ /\ (p - r)))$

   $= \quad a\ b\ c\ d\ -\ *\ p\ r\ -\ /\ +$

Evaluating Postfix Expression

- Postfix expression can be evaluated easily using stack.
- Stack operations, such as push(), pop() and isEmpty() will be used to solve this problem.
- Steps to evaluate postfix expression :

1. Convert infix to postfix expression.
2. Evaluate postfix using stack.

Converting Infix to Postfix Algorithm

```
create(s);  push(s, '#');
while (not end of infix input)
    {  ch = getch  // get next input character
       if (ch is an operan)
         add ch to postfix notation;
       if (ch = '(')
         push(s, ch)
       if (ch = ')')
       { ch = pop(s);
         while (ch != '(')
         {    add ch to postfix  notation;
              ch = pop(s);
         }
       }
```

Converting Infix to Postfix Algorithm

```
if (ch is an operator)
{    while (!isEmpty(s) &&
           (precedence(stacktop()) >= precedence(ch)))
      {    chpop = pop(s);
           add chpop to postfix notation;
       }
      push(s, ch);
      }
} // end while
while (stacktop() != '#')
{    ch = pop(s);
     add ch to postfix notation;
}// end while
```

Converting Infix to Postfix

A + B * C – D / E

infix	stack	postfix
A + B * C - D / E	#	
+ B * C - D / E	#	A
B * C - D / E	# +	A
* C - D / E	# +	A B
C - D / E	# + *	A B C
- D / E	# + *	A B C * +
D / E	# -	A B C * + D
/ E	# -	A B C * + D
E	# - /	A B C * + D E
	# - /	A B C * + D E
	#	A B C * + D E / -

Converting Infix to Postfix :

A * B – (C + D) + E

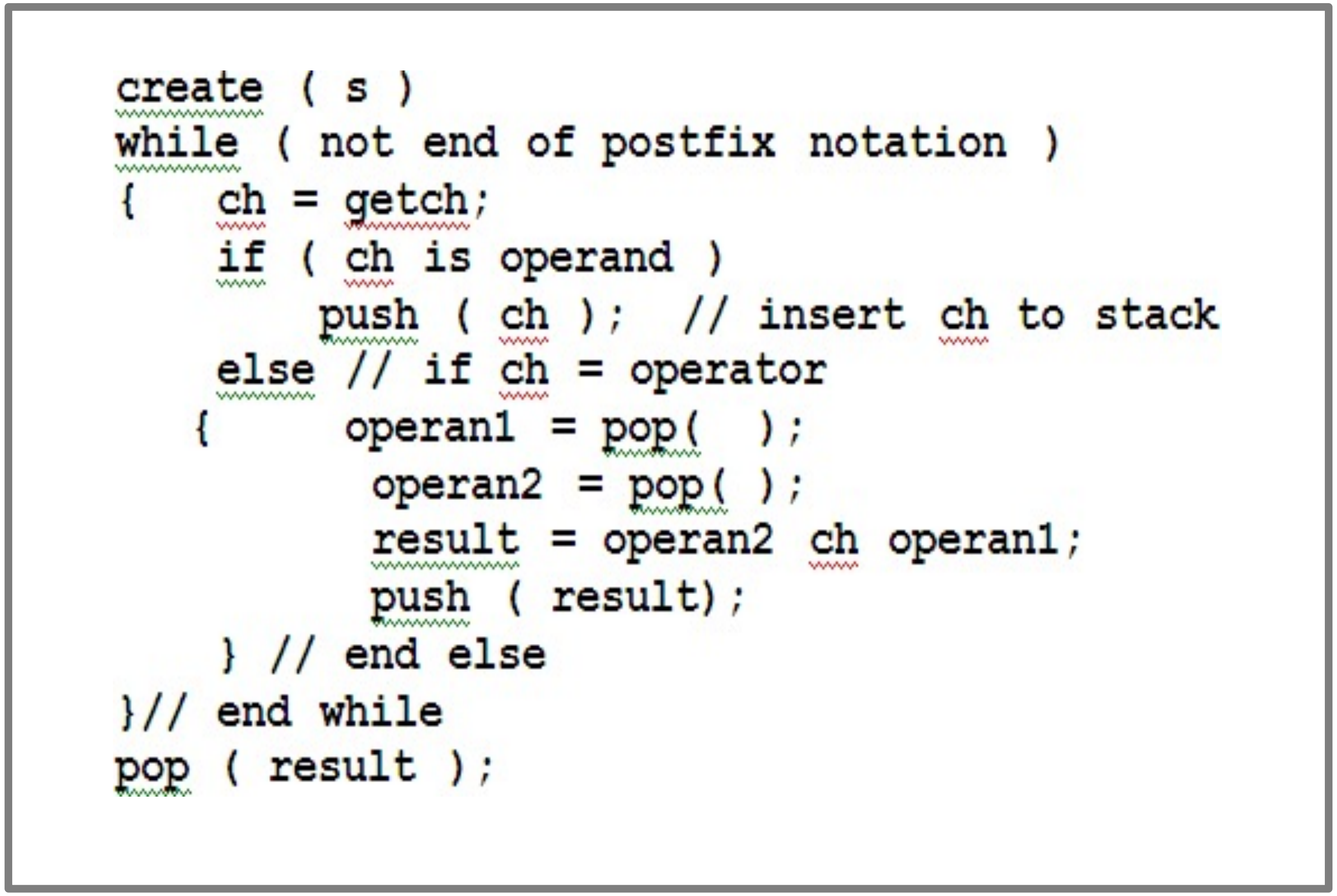
INFIX	STACK	POSTFIX
A * B – (C + D) + E	#	
* B – (C + D) + E	#	A
B – (C + D) + E	# *	A
– (C + D) + E	# *	A B
(C + D) + E	# -	A B *
C + D) + E	# - (	A B *
+ D) + E	# - (	A B * C
D) + E	# - (+	A B * C
) + E	# - (+	A B * C D
+ E	# -	A B * C D +
E	# +	A B * C D + -
	# +	A B * C D + - E
	#	A B * C D + - E +

Steps to Evaluate Postfix Expression

1. If char read from postfix expression is an operand, push operand to stack.
2. If char read from postfix expression is an operator, pop the first 2 operand in stack and implement the expression using the following operations:
   - **pop(opr1) dan pop(opr2)**
   - **result = opr2 operator opr1**
3. Push the result of the evaluation to stack.
4. Repeat steps 1 to steps 3 until end of postfix expression

- Finally, At the end of the operation, only one value left in the stack. The value is the result of postfix evaluation.

Evaluating Postfix Expression

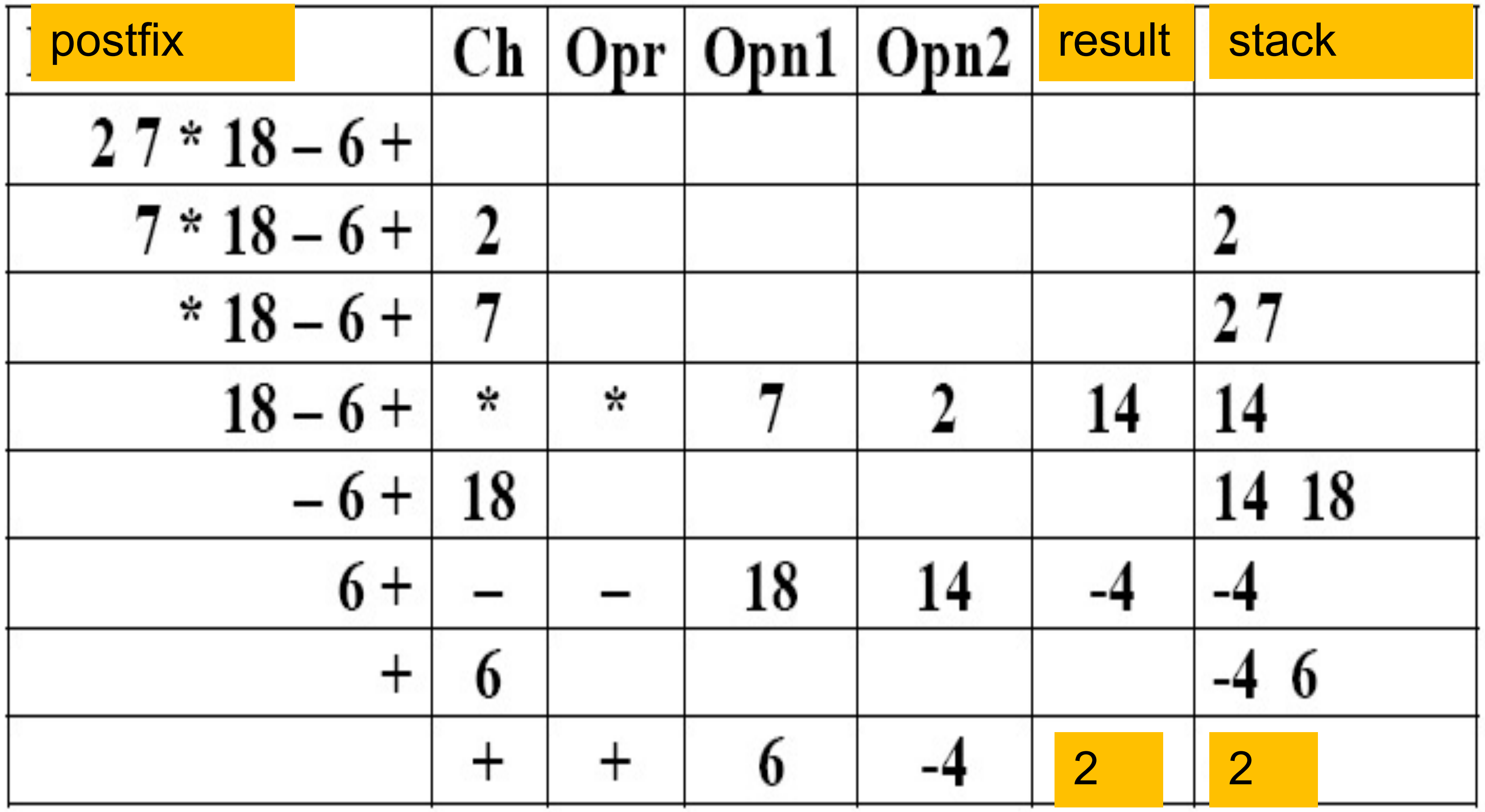


```
create ( s )
while ( not end of postfix notation )
{    ch = getch;
     if ( ch is operand )
          push ( ch );   // insert ch to stack
     else // if ch = operator
    {       operan1 = pop(   );
             operan2 = pop( );
             result = operan2 ch operan1;
             push ( result);
     } // end else
}// end while
pop ( result );
```

Evaluating Postfix Expression : 2 4 6 + *

Postfix	Ch	Opr	Opn1	Opn2	Result	Stack
2 4 6 + *						
4 6 + *	2					2
6 + *	4					2 4
+ *	6					2 4 6
*	+	+	6	4	10	2 10
	*	*	10	2	20	20

Evaluating Postfix Expression : 2 7 * 18 - 6 +



postfix	Ch	Opr	Opn1	Opn2	result	stack
2 7 * 18 – 6 +						
7 * 18 – 6 +	2					2
* 18 – 6 +	7					2 7
18 – 6 +	*	*	7	2	14	14
– 6 +	18					14 18
6 +	–	–	18	14	-4	-4
+	6					-4 6
	+	+	6	-4	2	2

Thank You