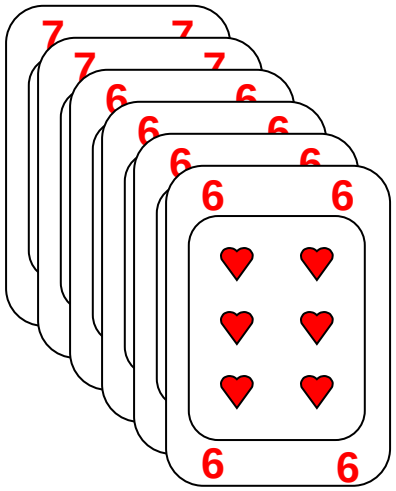
Objectives

- In this session, you will learn to:
  - Identify the features of a stack
  - Implement stacks
  - Apply stacks to solve programming problems

Stacks

- Let us play the game of Rummy.

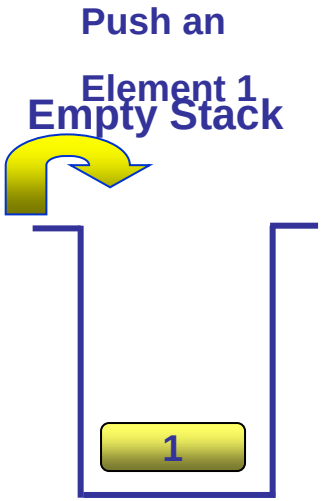


Defining a Stack

- What is a Stack ?
- A stack is a collection of data items that can be accessed at only one end, called top.
- Items can be inserted and deleted in a stack only at the top.
- The last item inserted in a stack is the first one to be deleted.
- Therefore, a stack is called a Last-In-First-Out (LIFO) data structure.

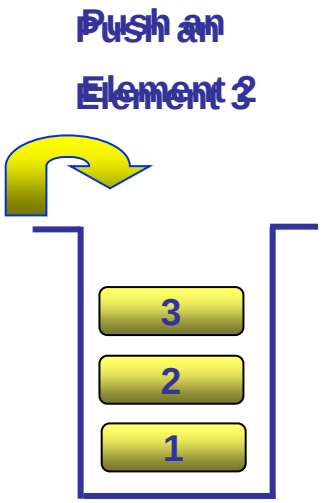
Identifying the Operations on Stacks

- PUSH and POP are two basic operations that are performed on the top of a stack.
  - PUSH
  - POP



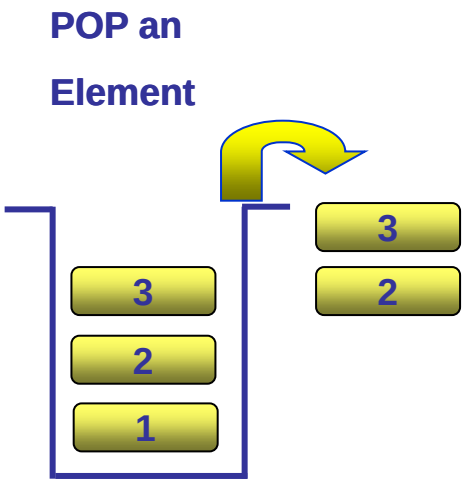
Identifying the Operations on Stacks (Contd.)

- **PUSH**: It is the process of inserting a new element on the top of a stack.



Identifying the Operations on Stacks (Contd.)

- **POP**: It is the process of deleting an element from the top of a stack.



Just a minute

- Elements in stacks are inserted and deleted on a _____ basis.

- Answer:
  - LIFO

Just a minute

- List down some real life examples that work on the LIFO principle.
- Answer:
  - **Pile of books**: Suppose a set of books are placed one over the other in a pile. When you remove books from the pile, the topmost book will be removed first. Similarly, when you have to add a book to the pile, the book will be placed at the top of the pile.
  - **Pile of plates**: The first plate begins the pile. The second plate is placed on the top of the first plate and the third plate is placed on the top of the second plate, and so on. In general, if you want to add a plate to the pile, you can keep it on the top of the pile. Similarly, if you want to remove a plate, you can remove the plate from the top of the pile.
  - **Bangles in a hand**: When a person wears bangles, the last bangle worn is the first one to be removed.

Implementing Stacks

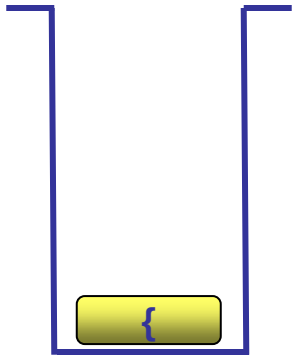
- You need to develop a method to check if the parentheses in an arithmetic expression are correctly nested.
- How will you solve this problem?
- You can solve this problem easily by using a stack.

Implementing Stacks (Contd.)

{(a + b) × (c + d) + (c × d)]}

- Consider an example.
- Suppose the expression is:

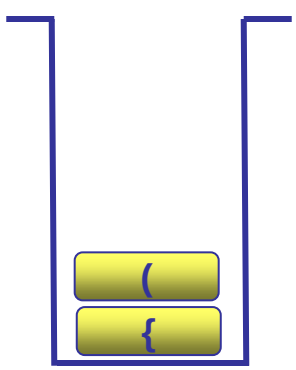
  {(a + b) × (c + d) + (c × d)]}
- Scan the expression from left to right.
- The first entry to be scanned is '{', which is a left parenthesis.
- Push it into the stack.



Implementing Stacks (Contd.)

{(a + b) × (c + d) + (c × d)]}

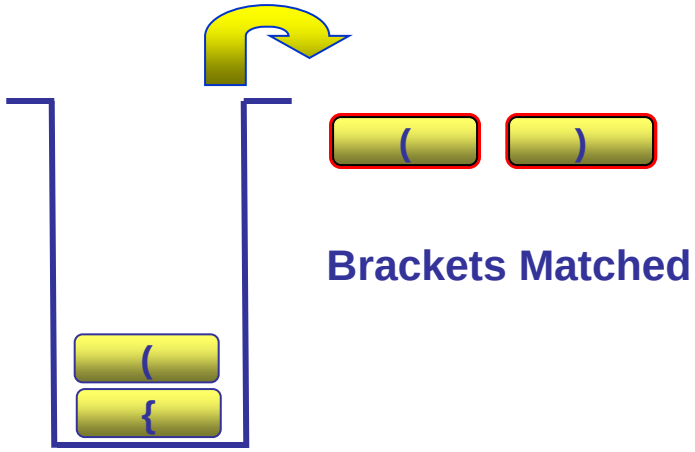
- The next entry to be scanned is '(', which is a left parenthesis.
- Push it into the stack.
- The next entry is 'a', which is an operand. Therefore, it is discarded.
- The next entry is '+', which is an operator. Therefore, it is discarded.
- The next entry is 'b', which is an operand. Therefore, it is discarded.



Implementing Stacks (Contd.)

- The next entry to be scanned is ‘)’, which is a right parenthesis
- POP the topmost entry from the stack.
- Match the two brackets.

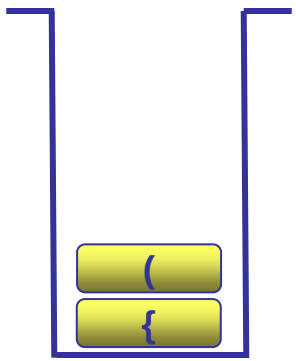
{(a + b) × (c + d) + (c × d)]}



Implementing Stacks (Contd.)

- The next entry to be scanned is '×', which is an operator. Therefore, it is discarded.
- The next entry to be scanned is '(', which is a left parenthesis
- Push it into the stack
- The next entry to be scanned is 'c', which is an operand. Therefore it is discarded
- The next entry to be scanned is '+', which is an operator. Therefore it is discarded
- The next entry to be scanned is 'd', which is an operand. Therefore it is discarded

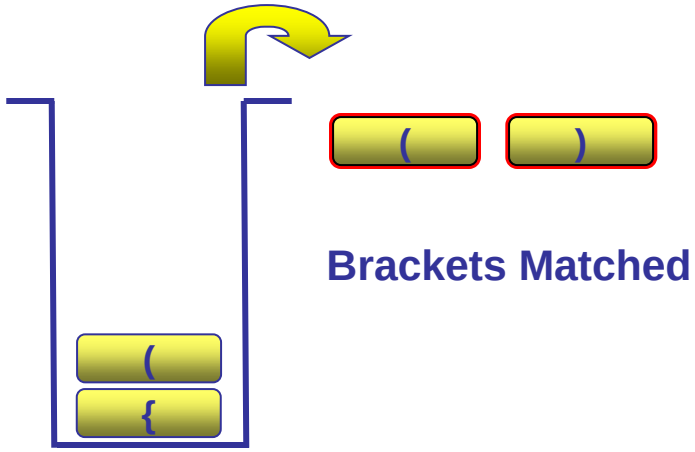
{(a + b) × (c + d) + (c × d)]}



Implementing Stacks (Contd.)

- The next entry to be scanned is ‘)’, which is a right parenthesis.
- POP the topmost element from the stack.
- Match the two brackets.

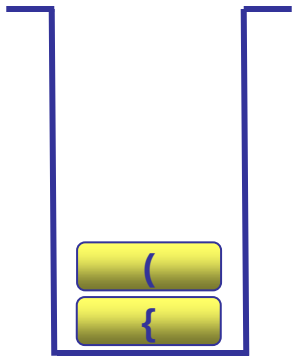
{(a + b) × (c + d) + (c × d)]}



Implementing Stacks (Contd.)

- The next entry to be scanned is '+', which is an operator. Therefore, it is discarded.
- The next entry to be scanned is '(', which is a left parenthesis.
- Push it into the stack.
- The next entry to be scanned is 'c', which is an operand. Therefore, it is discarded.
- The next entry to be scanned is '×', which is an operator. Therefore, it is discarded.
- The next entry to be scanned is 'd', which is an operand. Therefore, it is discarded.

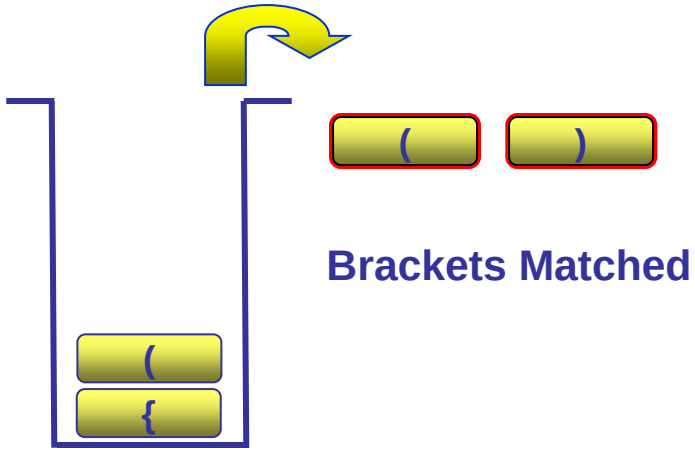
{(a + b) × (c + d) + (c × d)]}



Implementing Stacks (Contd.)

- The next entry to be scanned is ‘)’, which is a right parenthesis.
- POP the topmost element from the stack.
- Match the two brackets.

{(a + b) × (c + d) + (c × d)]}

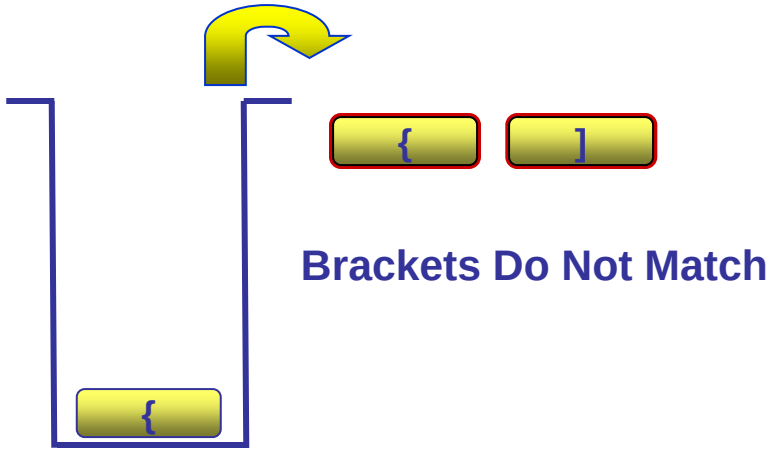


Implementing Stacks (Contd.)

- The next entry to be scanned is ‘]’, which is a right parenthesis.
- POP the topmost element from the stack.
- Match the two brackets.

The Expression is INVALID

{(a + b) × (c + d) + (c × d)]}



Brackets Do Not Match

Implementing a Stack Using an Array

- A stack is similar to a list in which insertion and deletion is allowed only at one end.
- Therefore, similar to a list, stack can be implemented using both arrays and linked lists.
  - Declare an array:

```
int stack[5];  //Maximum stack size needs to be specified in
               //advance
```

  - Declare a variable, top to hold the index of the topmost element in the stacks:

```
int top;
```

  - Initially, when the stack is empty, set:

    top = –1

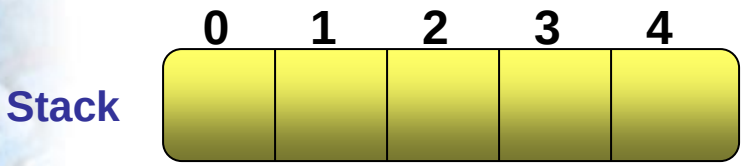
Implementing a Stack Using an Array (Contd.)

- Let us now write an algorithm for the PUSH operation.

Initially:

top = – 1

PUSH an element 3



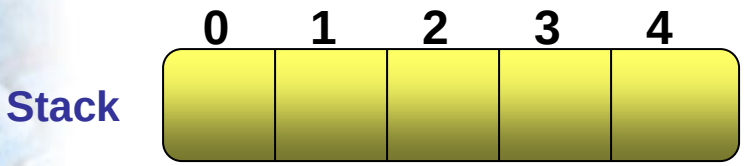
1. Increment top by 1.
2. Store the value to be pushed at index top in the array. Top now contains the index of the topmost element.

Implementing a Stack Using an Array (Contd.)

1. Increment top by 1.
2. Store the value to be pushed at index top in the array. Top now contains the index of the topmost element.

top = – 1

PUSH an element 3

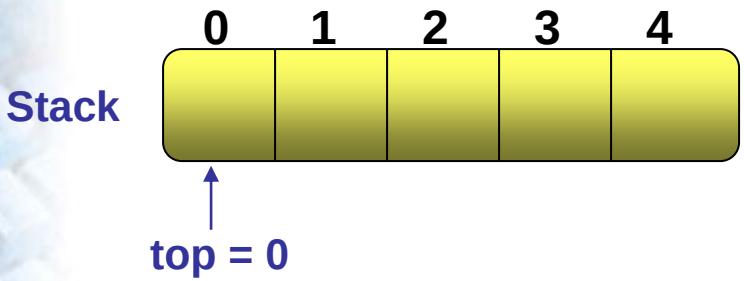


Implementing a Stack Using an Array (Contd.)

1. Increment top by 1.
2. Store the value to be pushed at index top in the array. Top now contains the index of the topmost element.

top = 0

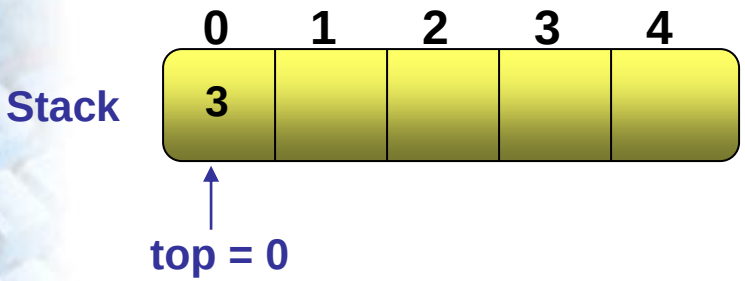
PUSH an element 3



Implementing a Stack Using an Array (Contd.)

1. Increment top by 1.
2. Store the value to be pushed at index top in the array. Top now contains the index of the topmost element.

PUSH an element 3

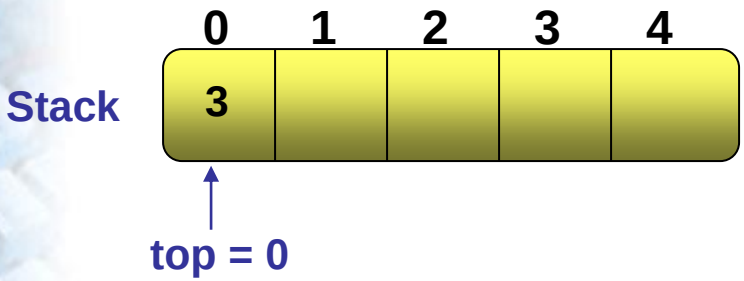


Item pushed

Implementing a Stack Using an Array (Contd.)

1. Increment top by 1.
2. Store the value to be pushed at index top in the array. Top now contains the index of the topmost element.

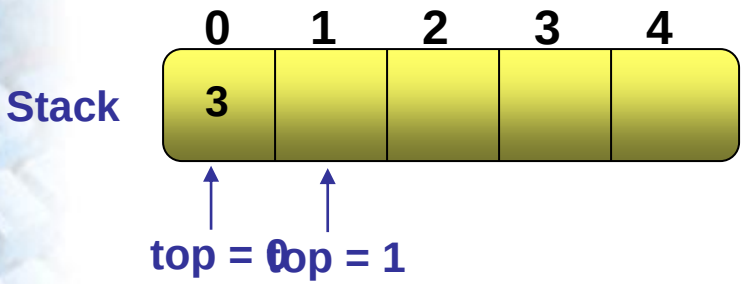
PUSH an element 8



Implementing a Stack Using an Array (Contd.)

1. Increment top by 1.
2. Store the value to be pushed at index top in the array. Top now contains the index of the topmost element.

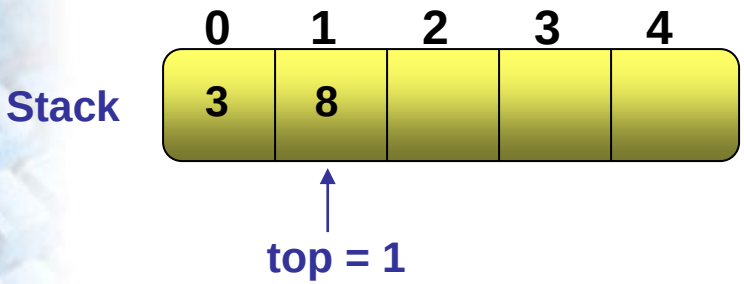
PUSH an element 8



Implementing a Stack Using an Array (Contd.)

1. Increment top by 1.
2. Store the value to be pushed at index top in the array. Top now contains the index of the topmost element.

PUSH an element 8

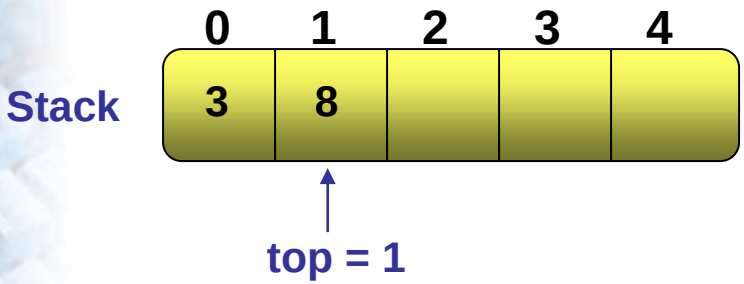


Item pushed

Implementing a Stack Using an Array (Contd.)

1. Increment top by 1.
2. Store the value to be pushed at index top in the array. Top now contains the index of the topmost element.

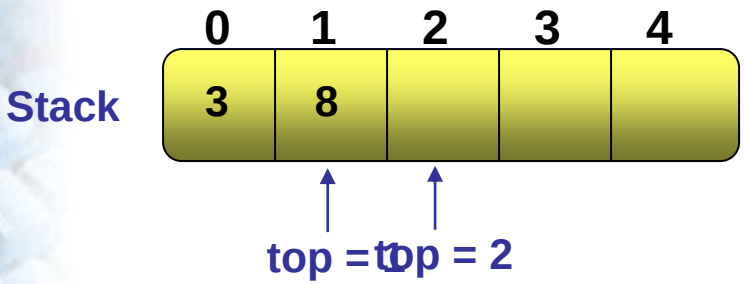
PUSH an element 5



Implementing a Stack Using an Array (Contd.)

1. Increment top by 1.
2. Store the value to be pushed at index top in the array. Top now contains the index of the topmost element.

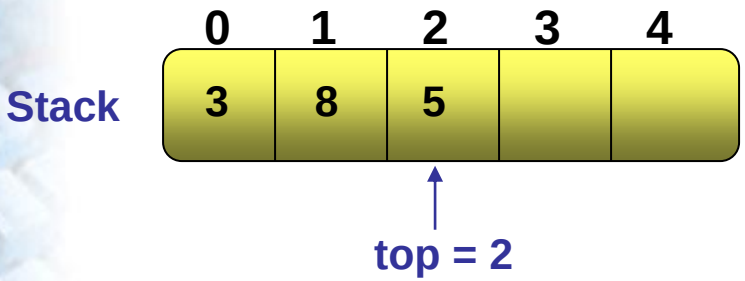
PUSH an element 5



Implementing a Stack Using an Array (Contd.)

1. Increment top by 1.
2. Store the value to be pushed at index top in the array. Top now contains the index of the topmost element.

PUSH an element 5

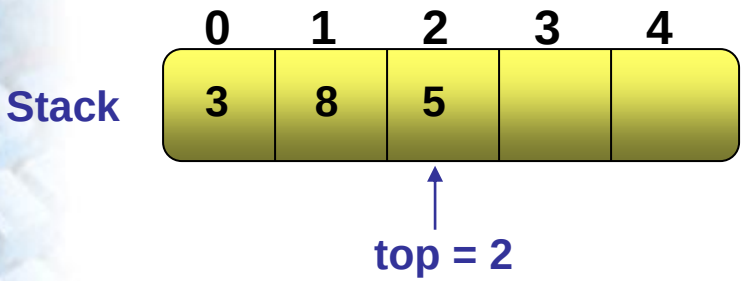


Item pushed

Implementing a Stack Using an Array (Contd.)

1. Increment top by 1.
2. Store the value to be pushed at index top in the array. Top now contains the index of the topmost element.

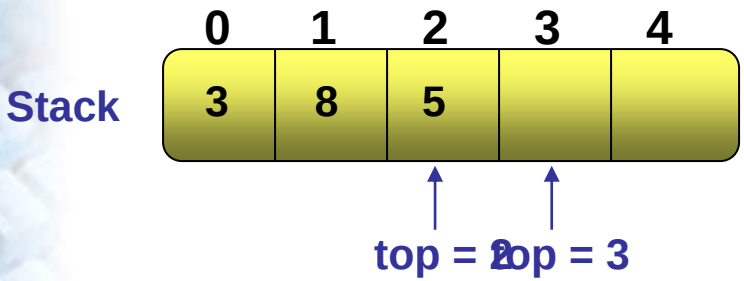
PUSH an element 1



Implementing a Stack Using an Array (Contd.)

1. Increment top by 1.
2. Store the value to be pushed at index top in the array. Top now contains the index of the topmost element.

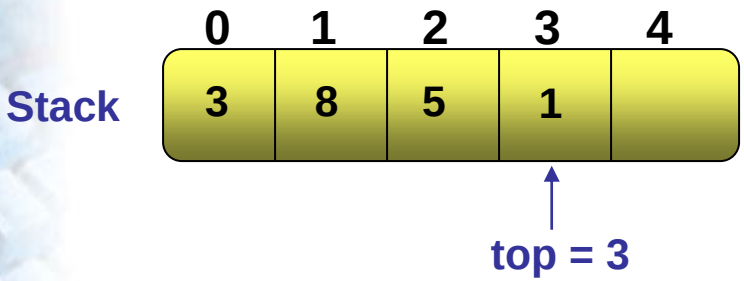
PUSH an element 1



Implementing a Stack Using an Array (Contd.)

1. Increment top by 1.
2. Store the value to be pushed at index top in the array. Top now contains the index of the topmost element.

PUSH an element 1

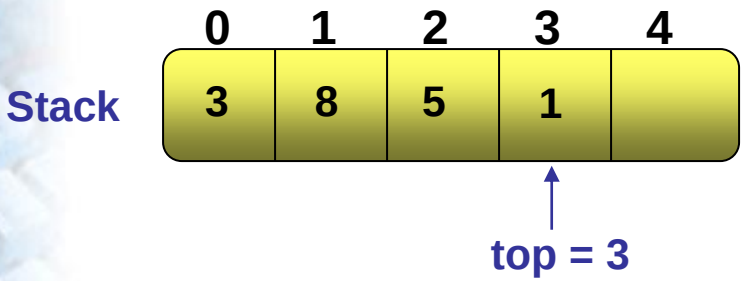


Item pushed

Implementing a Stack Using an Array (Contd.)

1. Increment top by 1.
2. Store the value to be pushed at index top in the array. Top now contains the index of the topmost element.

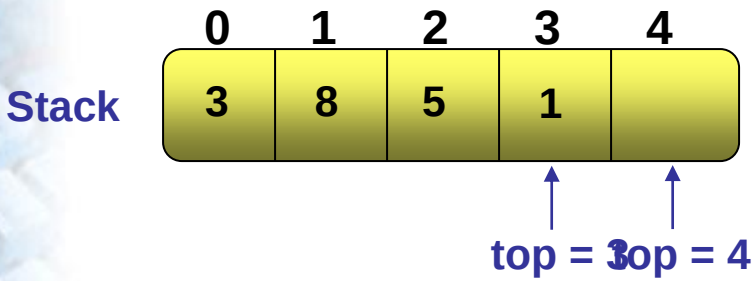
PUSH an element 9



Implementing a Stack Using an Array (Contd.)

1. Increment top by 1.
2. Store the value to be pushed at index top in the array. Top now contains the index of the topmost element.

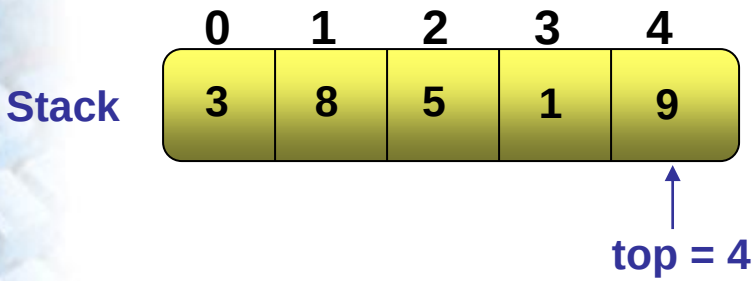
PUSH an element 9



Implementing a Stack Using an Array (Contd.)

1. Increment top by 1.
2. Store the value to be pushed at index top in the array. Top now contains the index of the topmost element.

PUSH an element 9

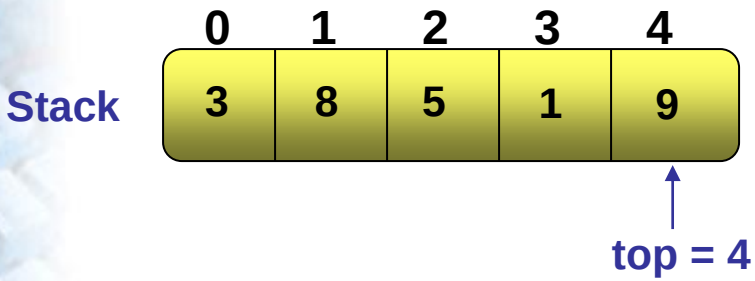


Item pushed

Implementing a Stack Using an Array (Contd.)

1. Increment top by 1.
2. Store the value to be pushed at index top in the array. Top now contains the index of the topmost element.

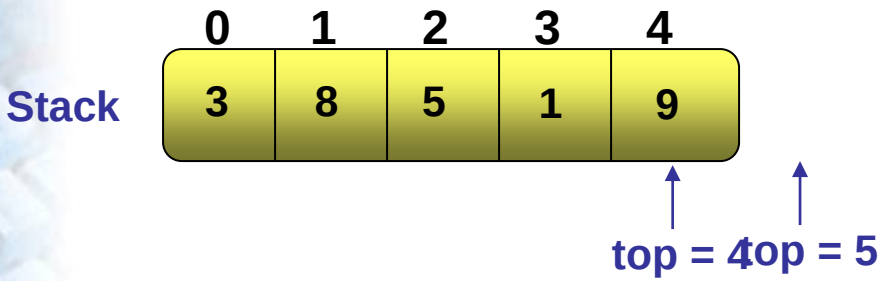
PUSH an element 2



Implementing a Stack Using an Array (Contd.)

1. Increment top by 1.
2. Store the value to be pushed at index top in the array. Top now contains the index of the topmost element.

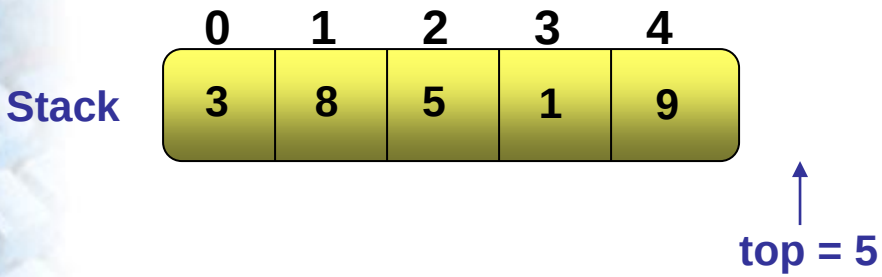
PUSH an element 2



Implementing a Stack Using an Array (Contd.)

1. Increment top by 1.
2. Store the value to be pushed at index top in the array. Top now contains the index of the topmost element.

PUSH an element 2

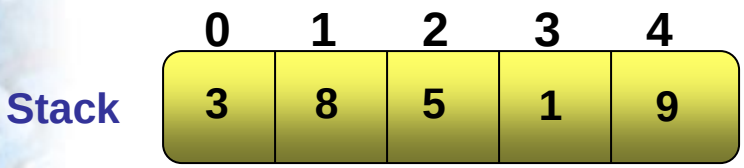


Stack overflow

Implementing a Stack Using an Array (Contd.)

- To avoid the stack overflow, you need to check for the stack full condition before pushing an element into the stack.
- Therefore, you cannot store more than 5 elements in the stack.
- Let us modify the algorithm to check for this condition.

1. If top = MAX – 1:
   a. Display "Stack Full"
   b. Exit
2. Increment top by 1.
3. Store the value to be pushed at index top in the array



Implementing a Stack Using an Array (Contd.)

- Write an algorithm to implement the POP operation on a stack.
- Algorithm for POP operation:
  1. If top = – 1:
     a. Display “Stack Empty”
     b. Exit
  2. Retrieve the value stored at index top
  3. Decrement top by 1

Just a minute

- In a stack, data can be stored and removed only from one end of the stack called the _____ of the stack.

- Answer:
  - top

Implementing a Stack Using a Linked List

- Write algorithm for PUSH operation to insert a node in a linked list.
- Write algorithm for POP operation to delete a node from a linked list.
- Write algorithm for PUSH and POP operations using a Linked List.

1. Make a variable tmp point to the topmost node.
2. Assign the value in the data field of the topmost node.
3. Make top point to the next node in sequence.
4. Release memory allocated to the node marked by tmp.

Activity: Implementing a Stack Using an Array

- Problem Statement:
  - Write a program to implement a stack by using an array that can store five elements.

Activity: Implementing a Stack Using a Linked List

- Problem Statement:
  - Write a program to implement a stack by using a linked list.

Just a minute

- What will be the condition for stack full in a stack implemented as a linked list?

- Answer:
  - When a stack is implemented as a linked list, there is no upper bound limit on the size of the stack. Therefore, there will be no stack full condition in this case.

Just a minute

- If a stack is represented in memory by using a linked list, then insertion and deletion of data will be done _____.
  1. At the end of the list
  2. At the beginning of the list
  3. Anywhere in the list
  4. At the beginning and at the end of the list respectively

- Answer:
  2. At the beginning of the list

Applications of Stacks

- Some of the applications of stacks are:
  - Implementing function calls
  - Maintaining the UNDO list for an application
  - Checking the nesting of parentheses in an expression
  - Evaluating expressions

Implementing Function Calls

- Implementing function calls:
  - Consider an example. There are three functions, F1, F2, and F3. Function F1 invokes F2 and function F2 invokes F3, as shown.

Implementing Function Calls (Contd.)

```
     void F1()
     {
1100         int x;
1101         x = 5;
1102         F2();
1103         print(x);
     }
     void F2(int x)
     {
1120         x = x + 5;
1121         F3(x);
1122         print(x);
     }
     void F3(int x)
     {
1140         x = x × 2;
1141         print x;
     }
```

Assuming these instructions at the given locations in the memory.

Implementing Function Calls (Contd.)

```
     void F1()
     {
1100         int x;
1101         x = 5;
1102         F2();
1103         print(x);
     }
     void F2(int x)
     {
1120         x = x + 5;
1121         F3(x);
1122         print(x);
     }
     void F3(int x)
     {
1140         x = x × 2;
1141         print x;
     }
```

The execution starts from function F1

Implementing Function Calls (Contd.)

```
     void F1()
     {
1100         int x;
1101         x = 5;
1102         F2();
1103         print(x);
     }
     void F2(int x)
     {
1120         x = x + 5;
1121         F3(x);
1122         print(x);
     }
     void F3(int x)
     {
1140         x = x × 2;
1141         print x;
     }
```

Implementing Function Calls (Contd.)

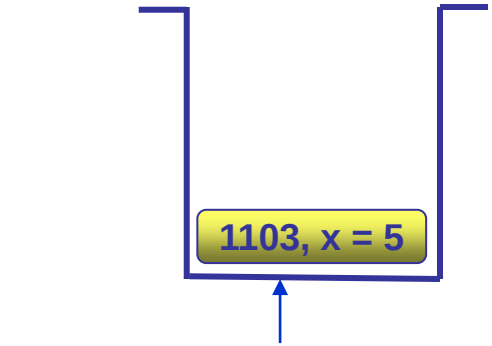
```
     void F1()
     {
1100         int x;
1101         x = 5;
1102         F2();
1103         print(x);
     }
     void F2(int x)
     {
1120         x = x + 5;
1121         F3(x);
1122         print(x);
     }
     void F3(int x)
     {
1140         x = x × 2;
1141         print x;
     }
```

x = 5

Implementing Function Calls (Contd.)

```
     void F1()
     {
1100         int x;
1101         x = 5;
1102         F2();
1103         print(x);
     }
     void F2(int x)
     {
1120         x = x + 5;
1121         F3(x);
1122         print(x);
     }
     void F3(int x)
     {
1140         x = x * 2;
1141         print x;
     }
```

x = 5

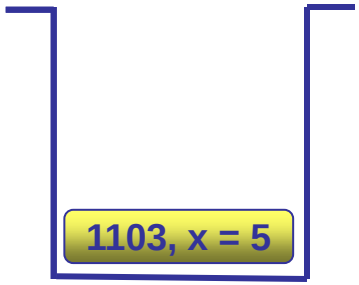


Address and the local variable of F1

Implementing Function Calls (Contd.)

```
     void F1()
     {
1100         int x;
1101         x = 5;
1102         F2();
1103         print(x);
     }
     void F2(int x)
     {
1120         x = x + 5;
1121         F3(x);
1122         print(x);
     }
     void F3(int x)
     {
1140         x = x * 2;
1141         print x;
     }
```

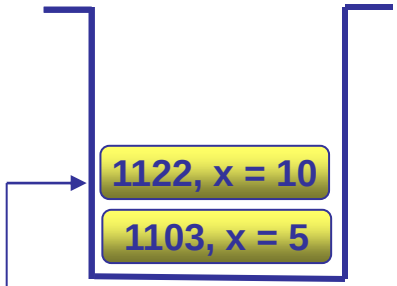
x = 5 10



Implementing Function Calls (Contd.)

```
     void F1()
     {
1100         int x;
1101         x = 5;
1102         F2();
1103         print(x);
     }
     void F2(int x)
     {
1120         x = x + 5;
1121         F3(x);
1122         print(x);
     }
     void F3(int x)
     {
1140         x = x * 2;
1141         print x;
     }
```

x = 10

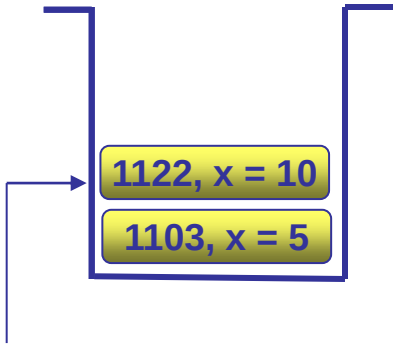


Address and the local variable of F2

Implementing Function Calls (Contd.)

```
     void F1()
     {
1100         int x;
1101         x = 5;
1102         F2();
1103         print(x);
     }
     void F2(int x)
     {
1120         x = x + 5;
1121         F3(x);
1122         print(x);
     }
     void F3(int x)
     {
1140         x = x * 2;
1141         print x;
     }
```

x = 20



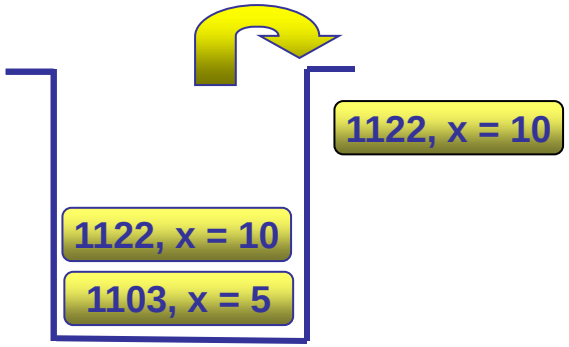
Address and the local variable of F2

Implementing Function Calls (Contd.)

```
     void F1()
     {
1100         int x;
1101         x = 5;
1102         F2();
1103         print(x);
     }
     void F2(int x)
     {
1120         x = x + 5;
1121         F3(x);
1122         print(x);
     }
     void F3(int x)
     {
1140         x = x * 2;
1141         print x;
     }
```

20

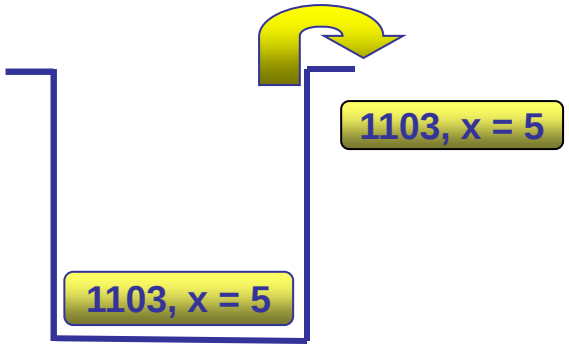
x = 20



Implementing Function Calls (Contd.)

```
     void F1()
     {
1100         int x;
1101         x = 5;
1102         F2();
1103         print(x);
     }
     void F2(int x)
     {
1120         x = x + 5;
1121         F3(x);
1122         print(x);
     }
     void F3(int x)
     {
1140         x = x * 2;
1141         print x;
     }
```

x = 5 10

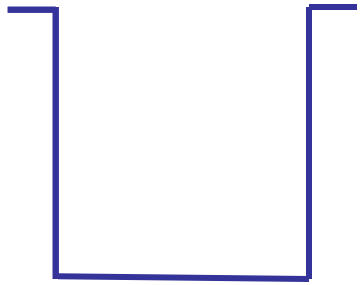


20 10

Implementing Function Calls (Contd.)

```
     void F1()
     {
1100         int x;
1101         x = 5;
1102         F2();
1103         print(x);
     }
     void F2(int x)
     {
1120         x = x + 5;
1121         F3(x);
1122         print(x);
     }
     void F3(int x)
     {
1140         x = x × 2;
1141         print x;
     }
```

x = 5



20 10 5

Maintaining the UNDO list for an Application

- Maintaining the UNDO list for an application:
  - Consider that you made some changes in a Word document. Now, you want to revert back those changes. You can revert those changes with the help of an UNDO feature.
  - The UNDO feature reverts the changes in a LIFO manner. This means that the change that was made last is the first one to be reverted.
  - You can implement the UNDO list by using a stack.

Checking the Nesting of Parentheses in an Expression

- Checking the nesting of parentheses in an expression:
  - You can do this by checking the following two conditions:
    - The number of left parenthesis should be equal to the number of right parenthesis.
    - Each right parenthesis is preceded by a matching left parenthesis.

Evaluating Expressions

- Evaluating an expression by using stacks:
  - Stacks can be used to solve complex arithmetic expressions.
  - The evaluation of an expression is done in two steps:
    - Conversion of the infix expression into a postfix expression.
    - Evaluation of the postfix expression.

Activity: Implementing a Stack using a Linked List

- Problem Statement:
  - Write a program that accepts an infix expression, and then converts it into a postfix expression. You can assume that the entered expression is a valid infix expression.

Summary

- In this session, you learned that:
  - A stack is a collection of data items that can be accessed at only one end, called top. The last item inserted in a stack is the first one to be deleted.
  - A stack is called a LIFO data structure.
  - There are two operations that can be performed on stacks. They are:
    - PUSH
    - POP
  - Stacks can be implemented by using both arrays and linked lists.

Summary (Contd.)

- Stacks are used in many applications. Some of the application domains of stacks are as follows:
  - Implementing function calls
  - Maintaining the UNDO list for an application
  - Checking the nesting of parentheses in an expression
  - Evaluating expressions