Stack ADT

Stack

- Stack is a linear data structure in which the operations are performed based on LIFO (Last In First Out) principle.

- The following operations are performed on the stack…
  - Push (To insert an element on to the stack)
  - Pop (To delete an element from the stack)
  - Display (To display elements of the stack)

- Stack data structure can be implemented in two ways:

  - Using Array
  - Using Linked List

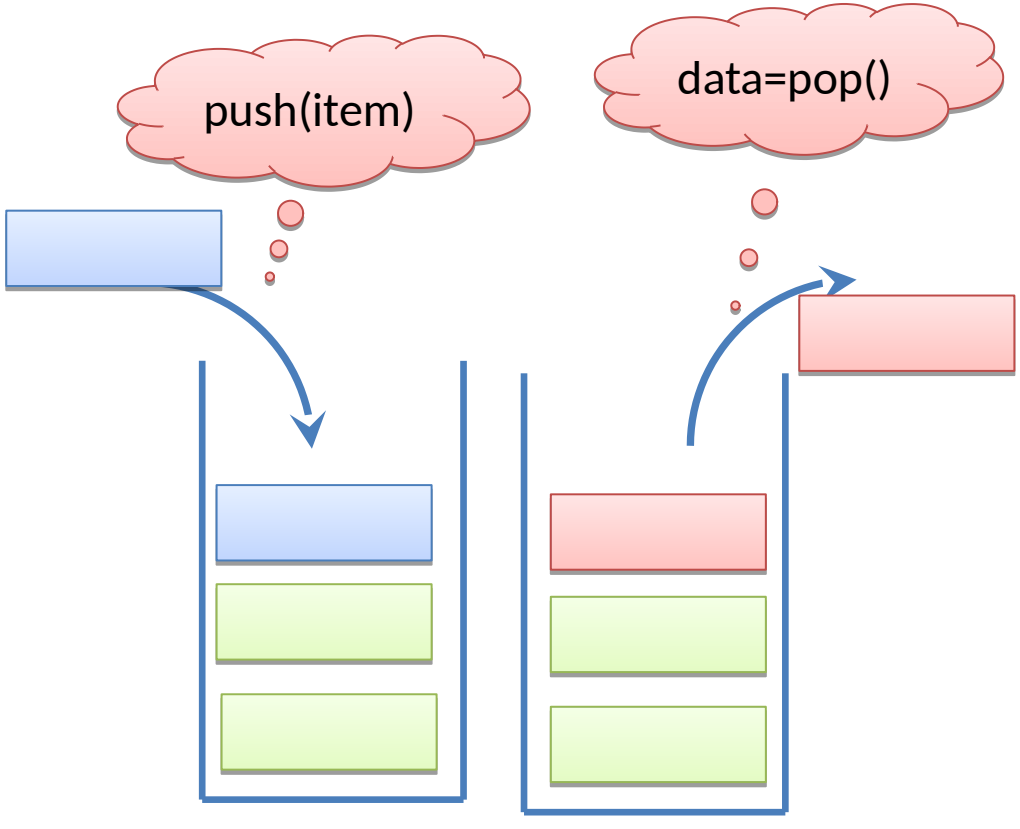
- When a stack is implemented using an array, that stack can organize an only limited number of elements. When a stack is implemented using a linked list, that stack can organize an unlimited number of elements.

Source: http://www.btechsmartclass.com/data_structures/stack-adt.html

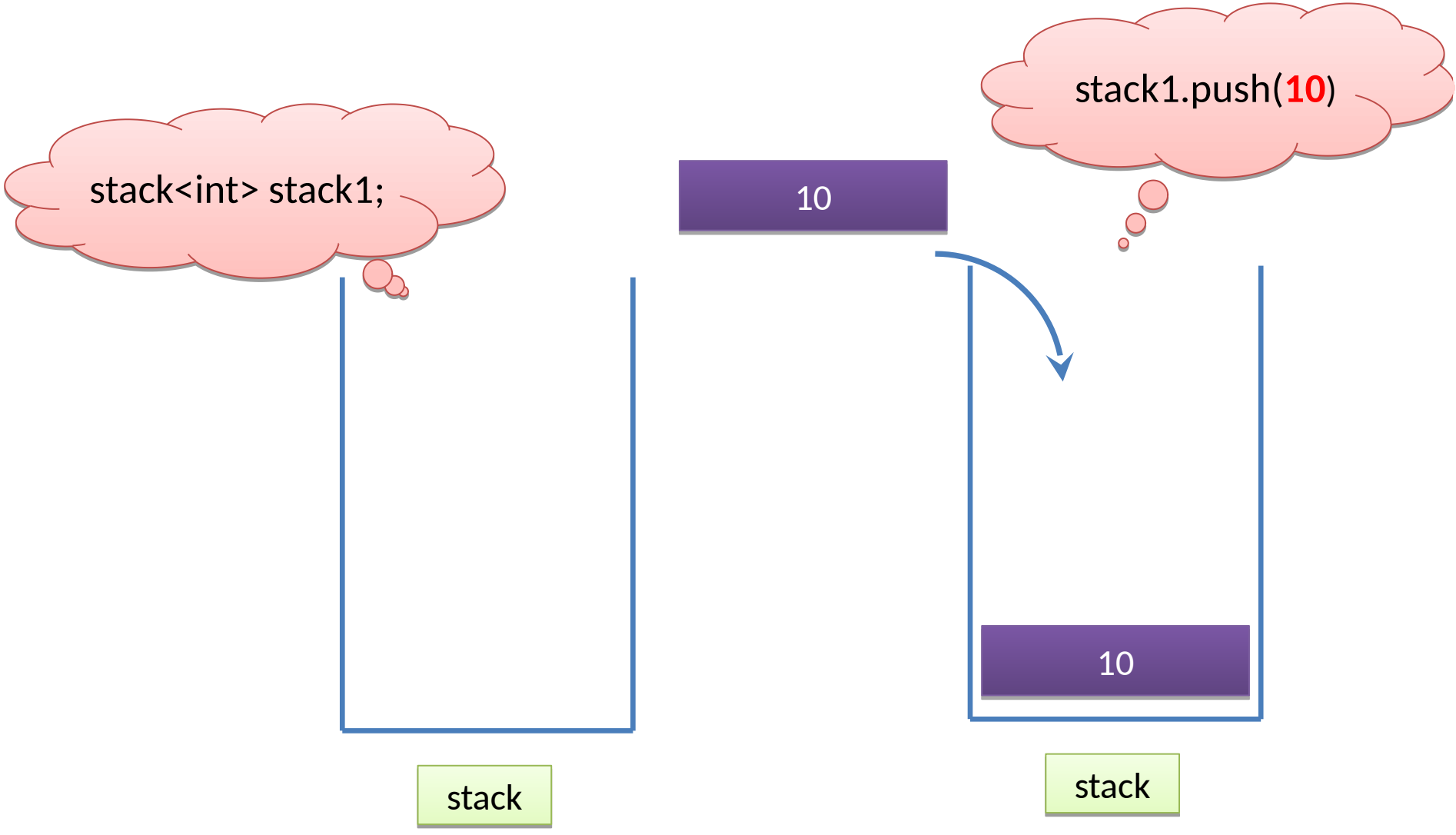
Stack operations

- Stack ADT basic operations:
- **void push(data)**
- **void pop()**
- **item top() or peek()**
- **void clear()**
- **bool is_empty()**

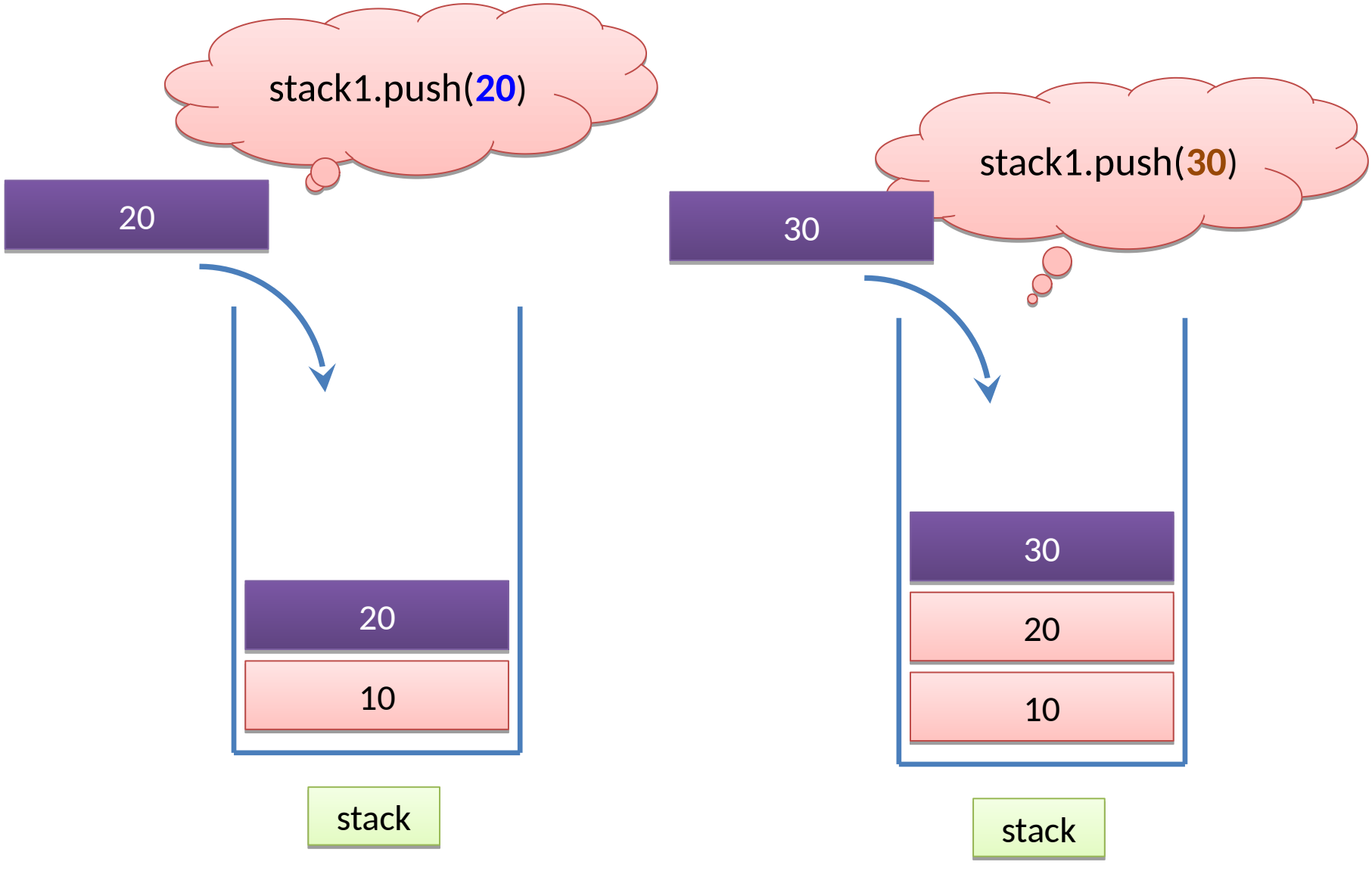
…



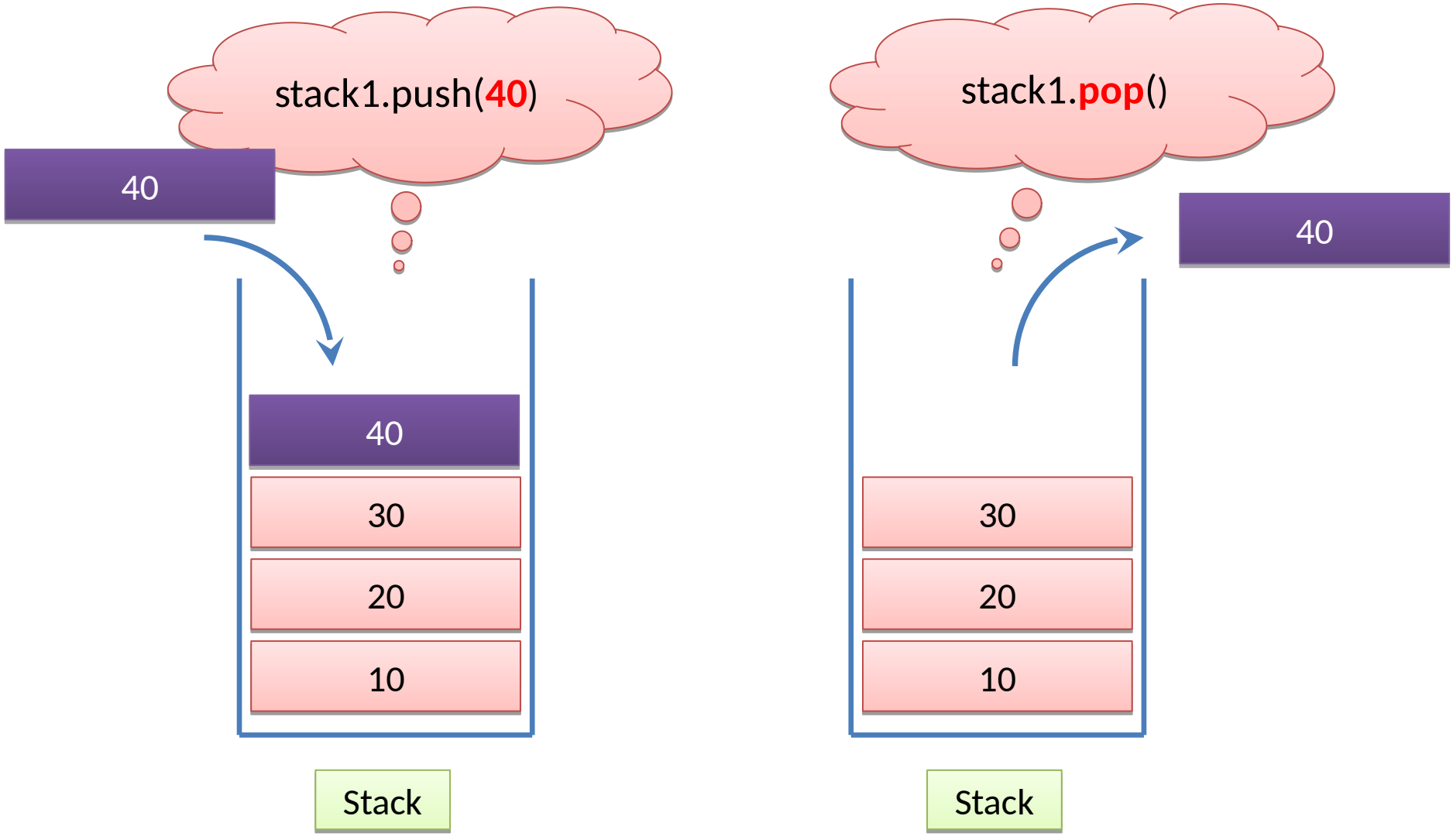
Stack operations



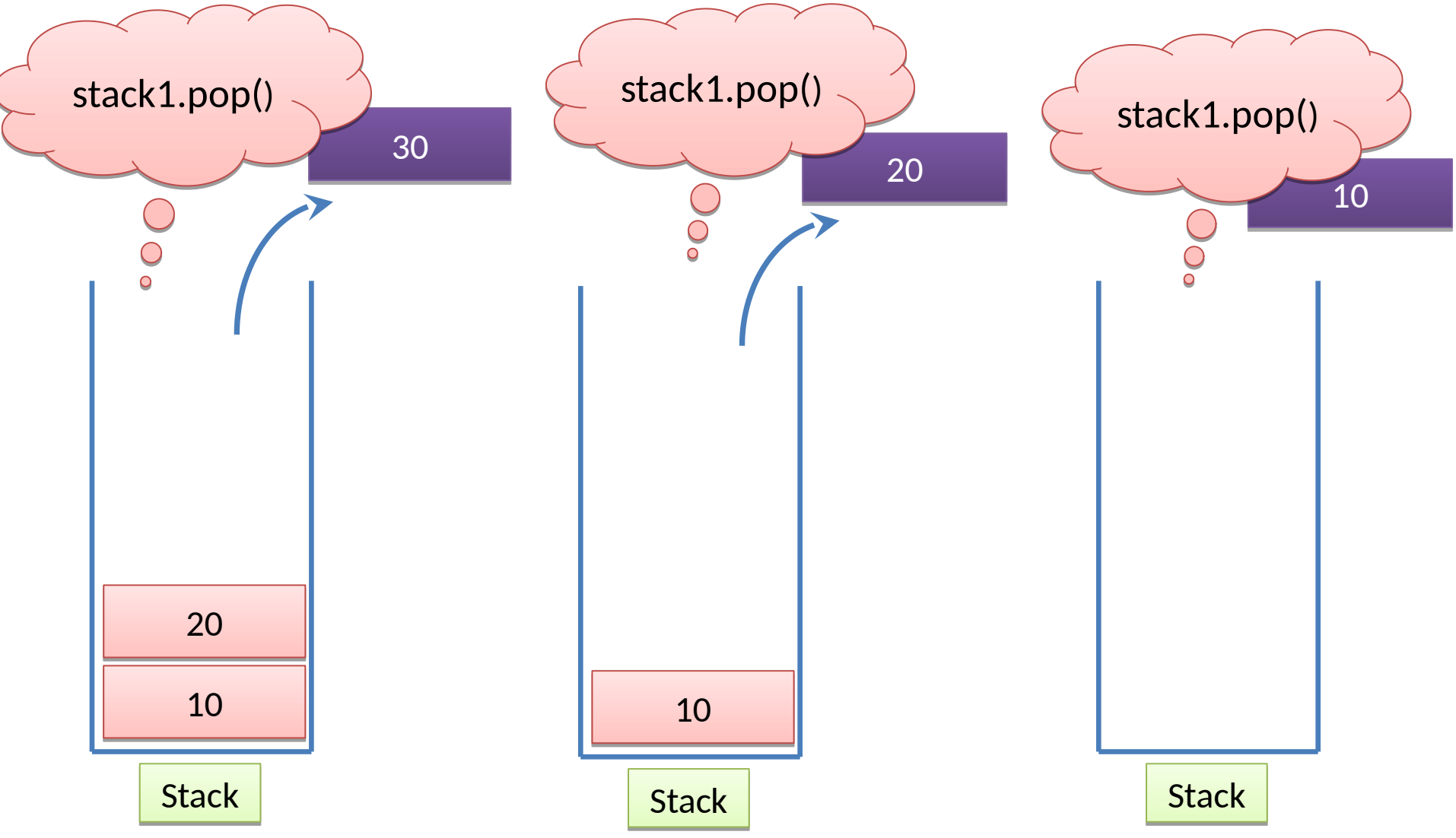
Stack operations



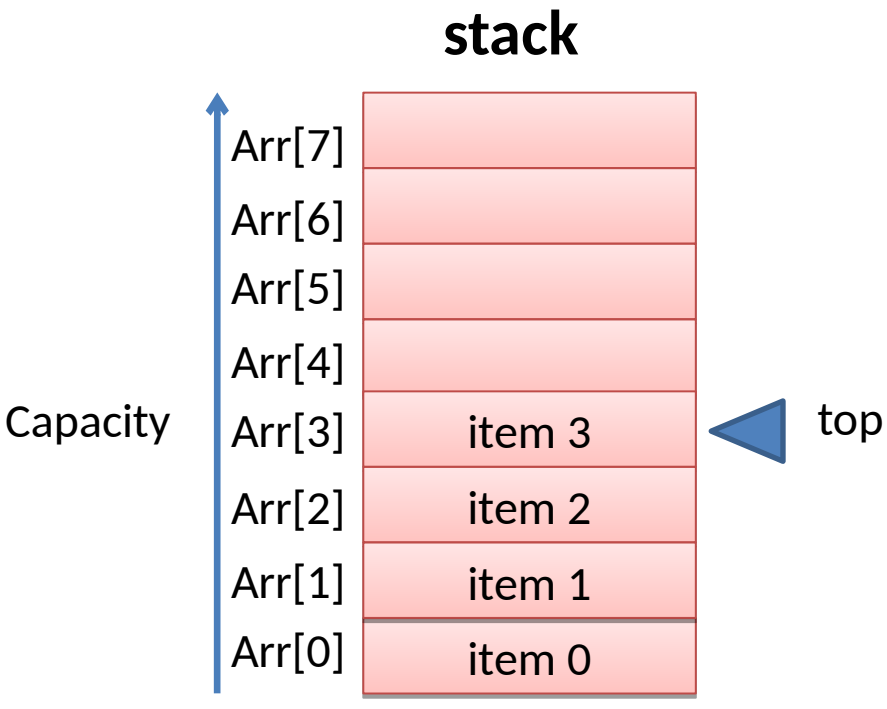
Stack operations



Stack operations

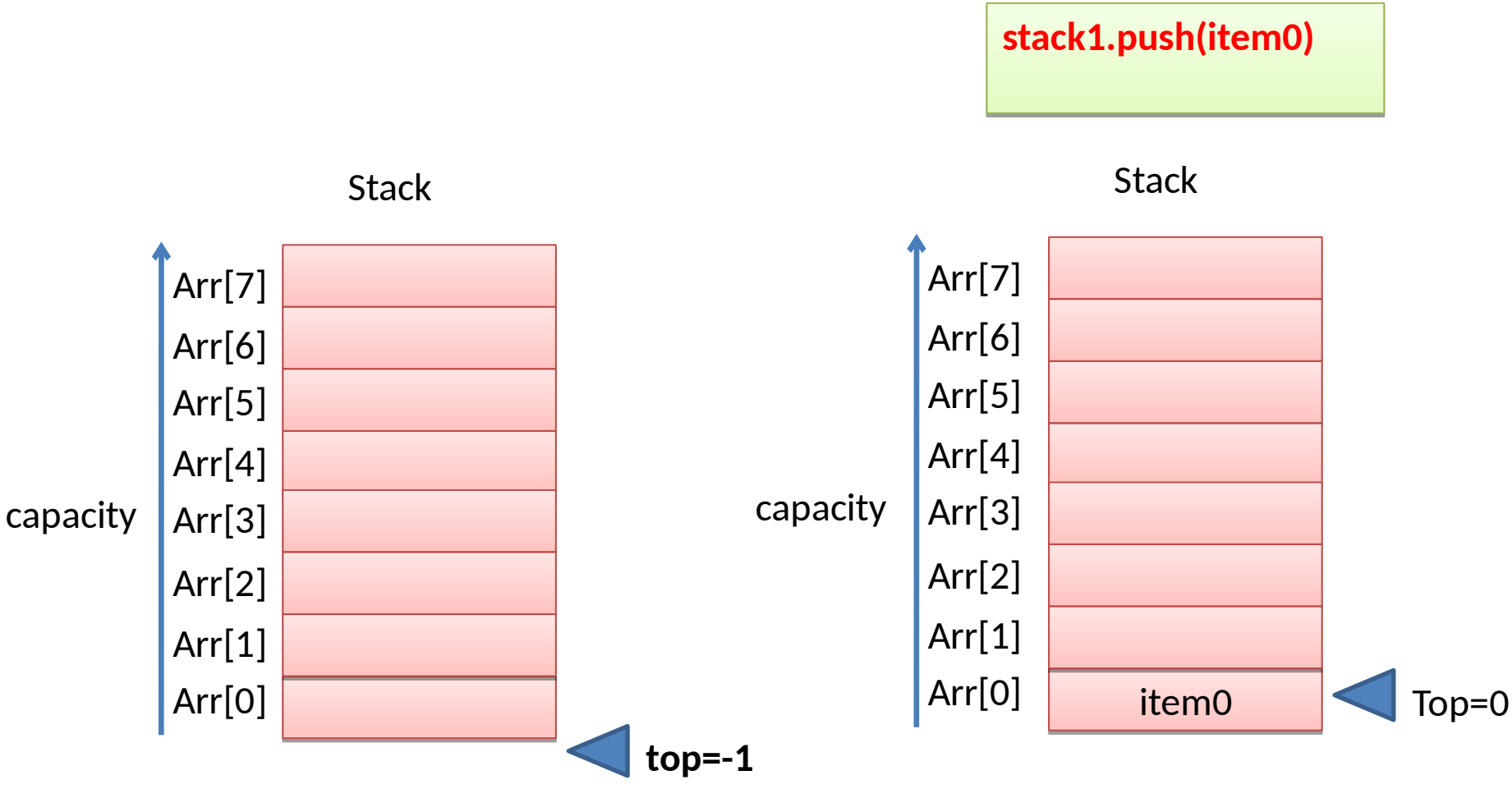


Array implementation of Stack

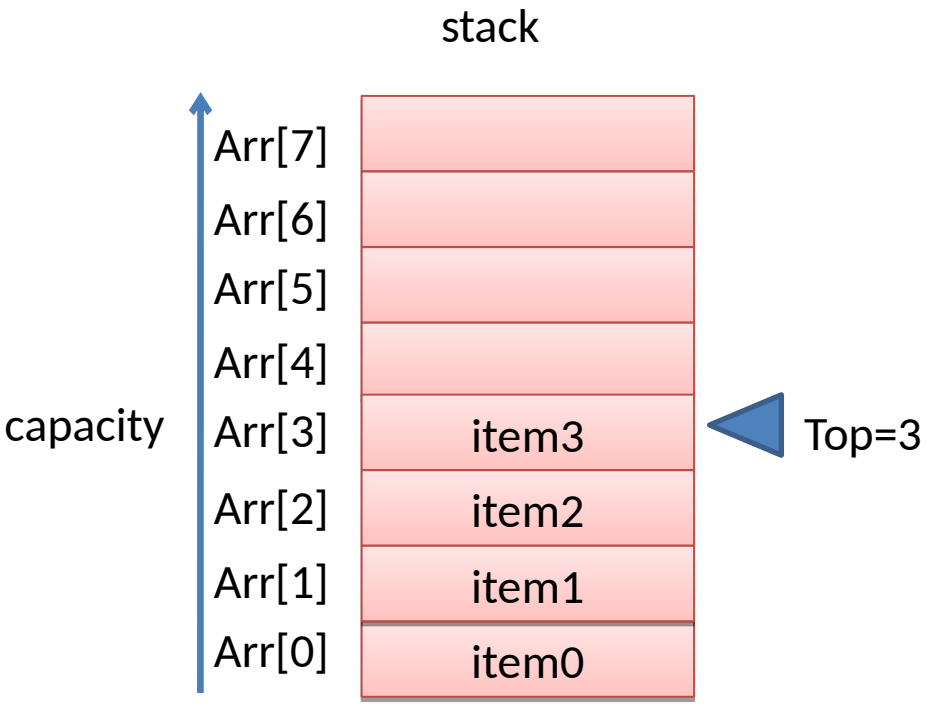


- In the implementation with an array, the stack elements are kept on the array.
- An integer variable is used to keep track of the address of the element at the top of the stack.

Array implementation of Stack

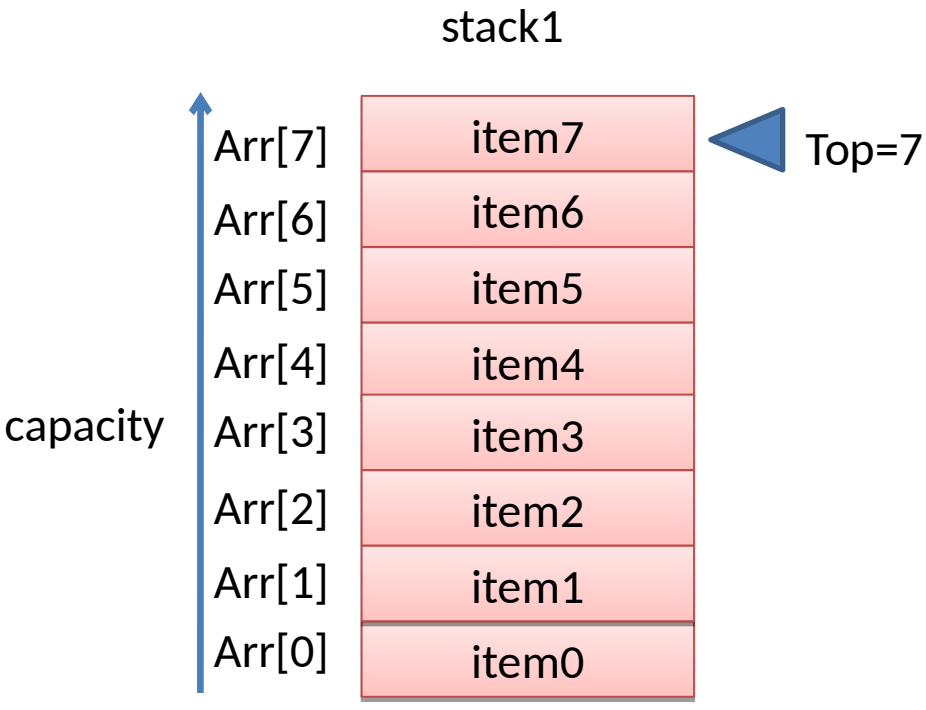


Array implementation of Stack



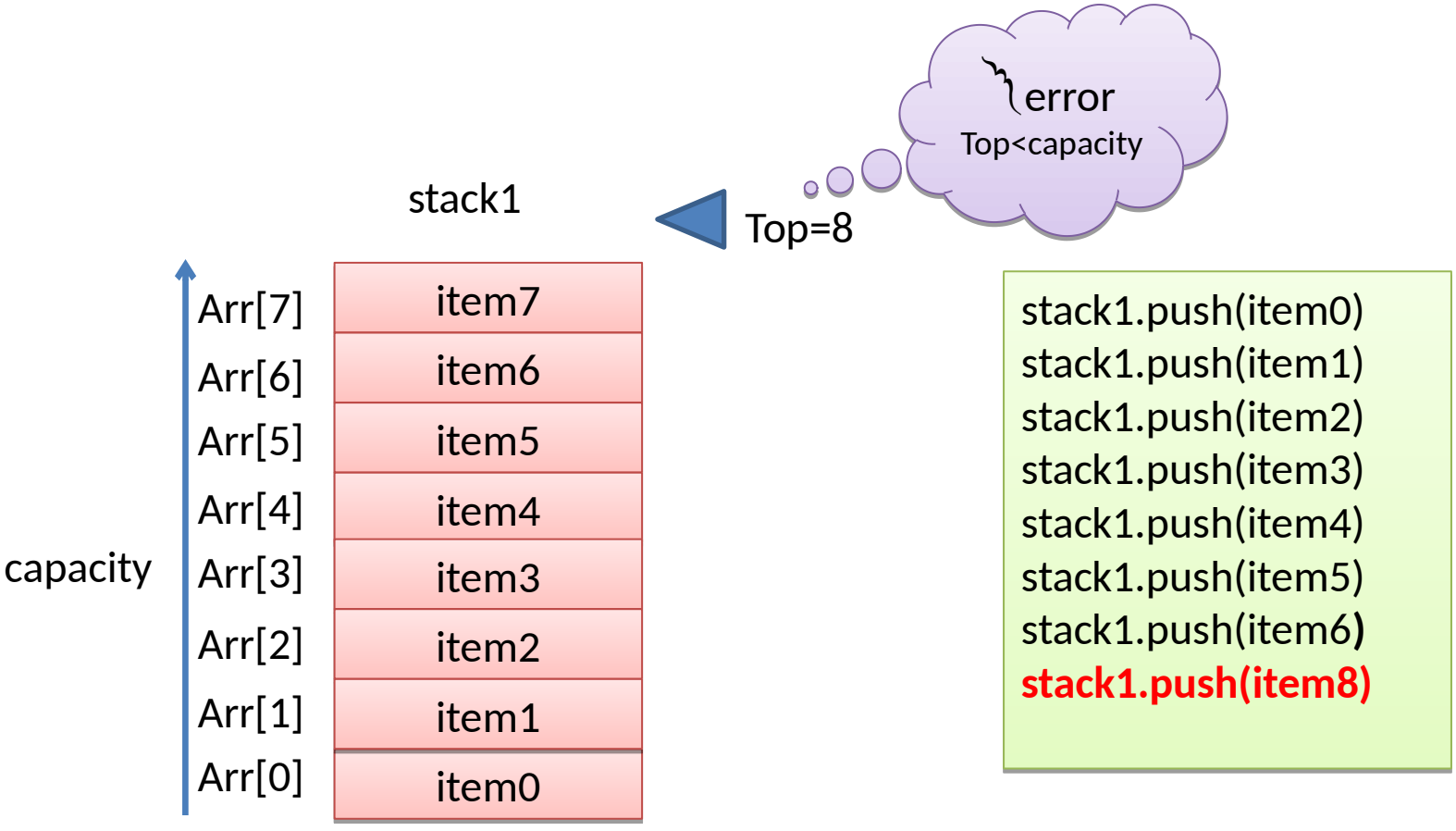

```
stack1.push(item0)
stack1.push(item1)
stack1.push(item2)
stack1.push(item3)
```

Array implementation of Stack

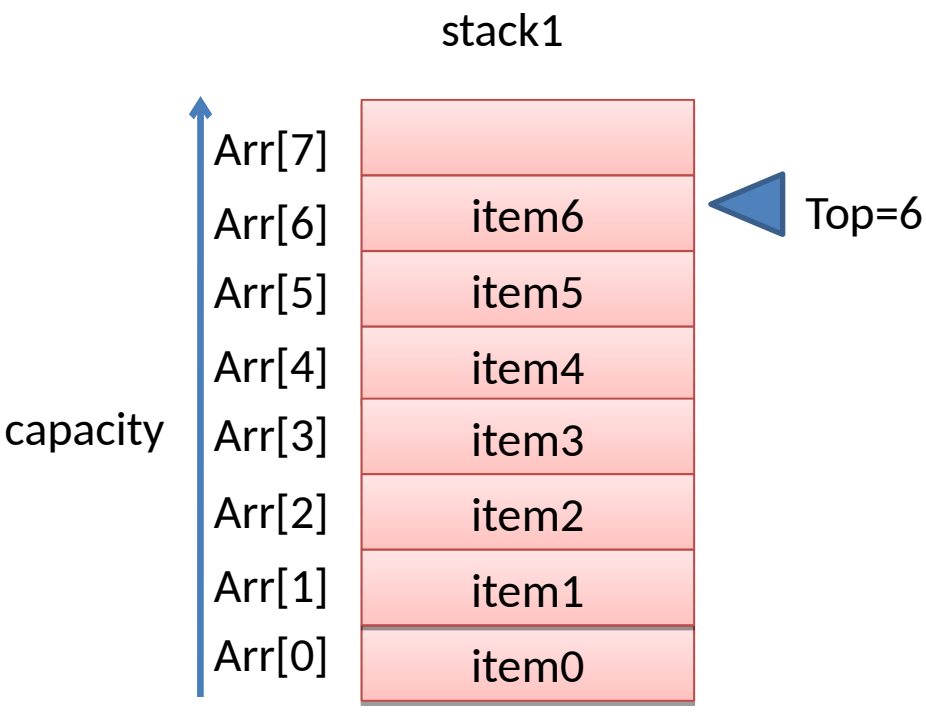



```
stack1.push(item0)
stack1.push(item1)
stack1.push(item2)
stack1.push(item3)
stack1.push(item4)
stack1.push(item5)
stack1.push(item6)
stack1.push(item7)
```

Array implementation of Stack

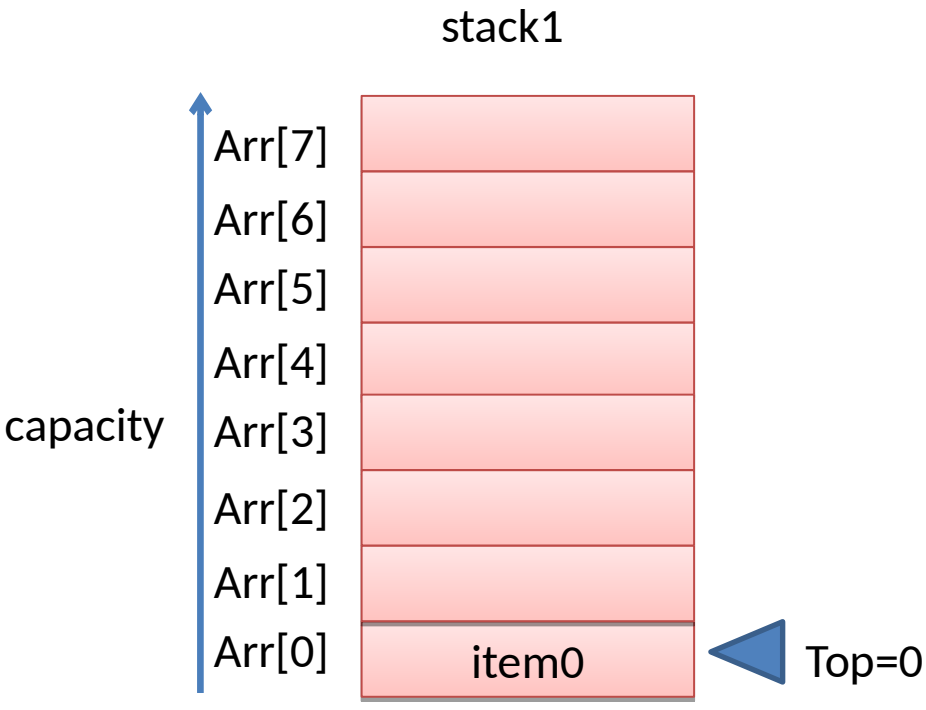


Array implementation of Stack



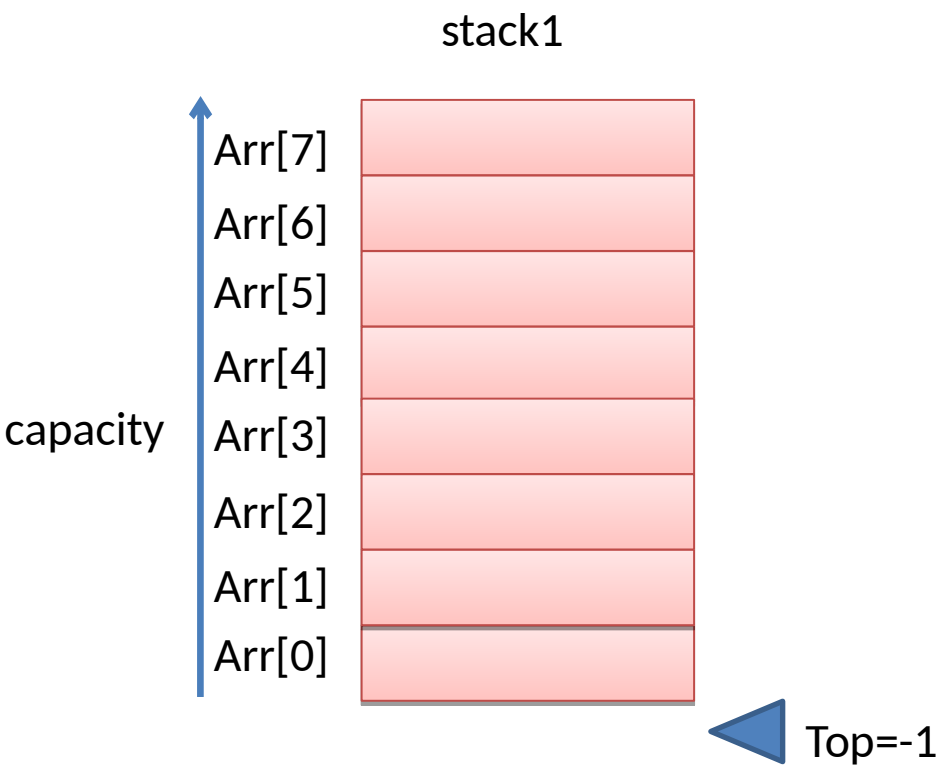

```
stack1.push(item0)
stack1. push(item1)
stack1. push(item2)
stack1. push(item3)
stack1.push(item4)
stack1.push(item5)
stack1.push(item6)
stack1.push(item7)
stack1.pop()
```

Array implementation of Stack



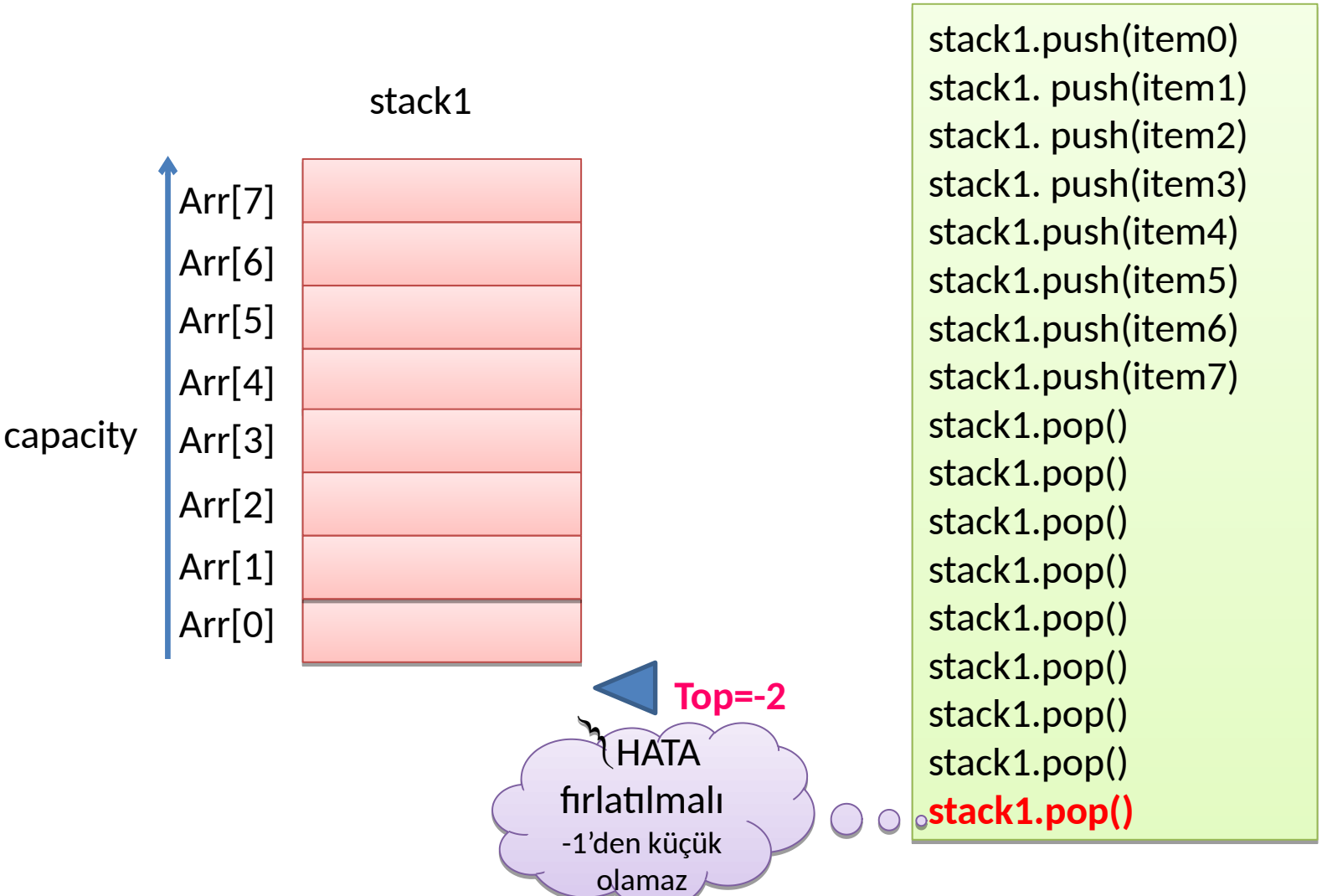

```
stack1.push(item0)
stack1. push(item1)
stack1. push(item2)
stack1. push(item3)
stack1.push(item4)
stack1.push(item5)
stack1.push(item6)
stack1.push(item7)
stack1.pop()
stack1.pop()
stack1.pop()
stack1.pop()
stack1.pop()
stack1.pop()
```

Array implementation of Stack

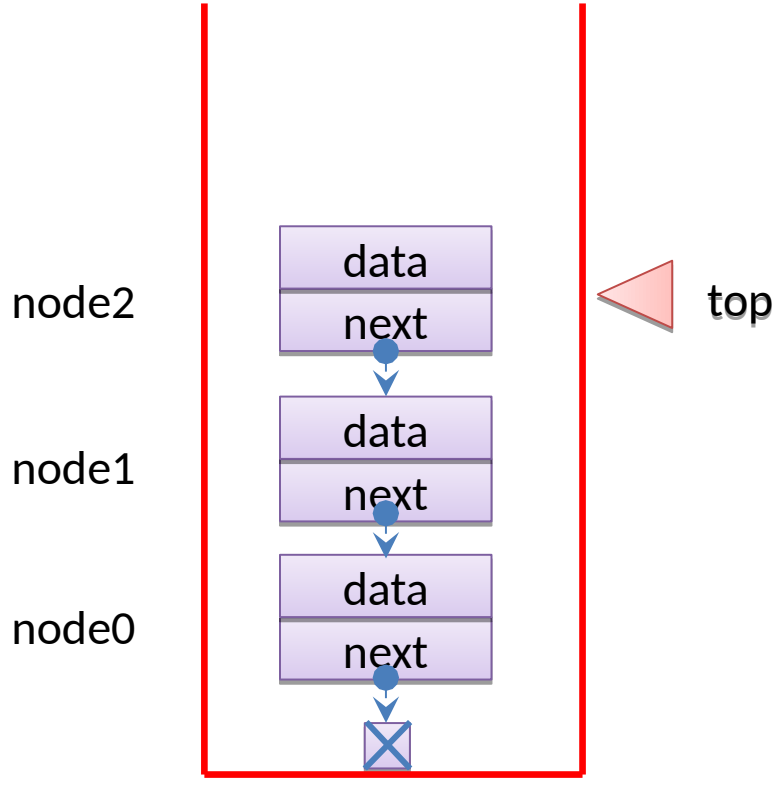



```
stack1.push(item0)
stack1. push(item1)
stack1. push(item2)
stack1. push(item3)
stack1.push(item4)
stack1.push(item5)
stack1.push(item6)
stack1.push(item7)
stack1.pop()
stack1.pop()
stack1.pop()
stack1.pop()
stack1.pop()
stack1.pop()
stack1.pop()
```

Array implementation of Stack

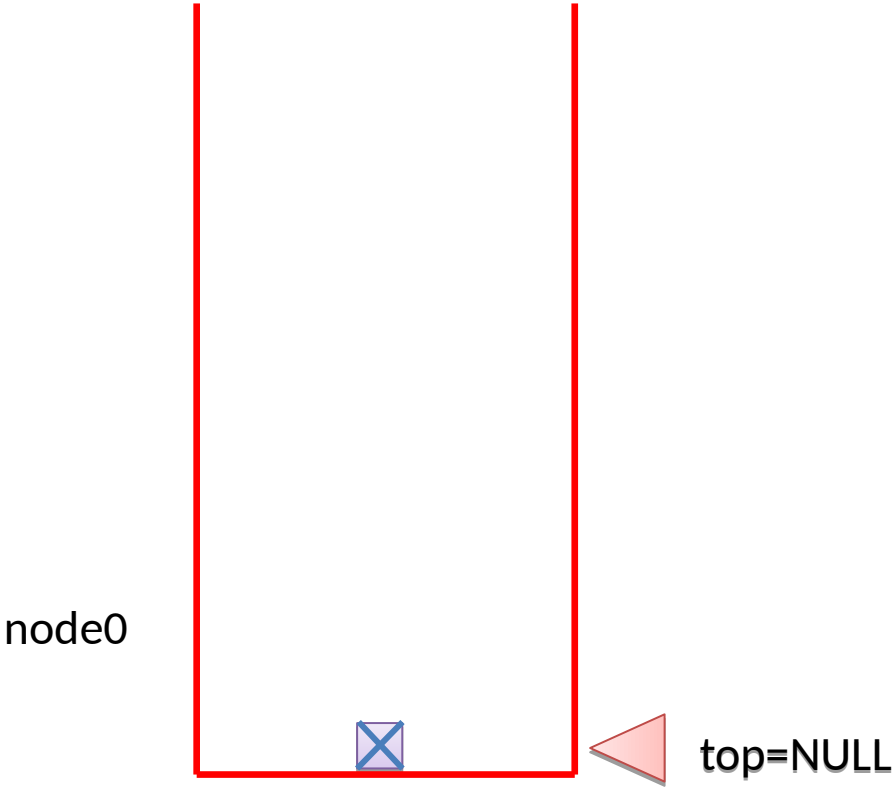


Linked implementation of Stack



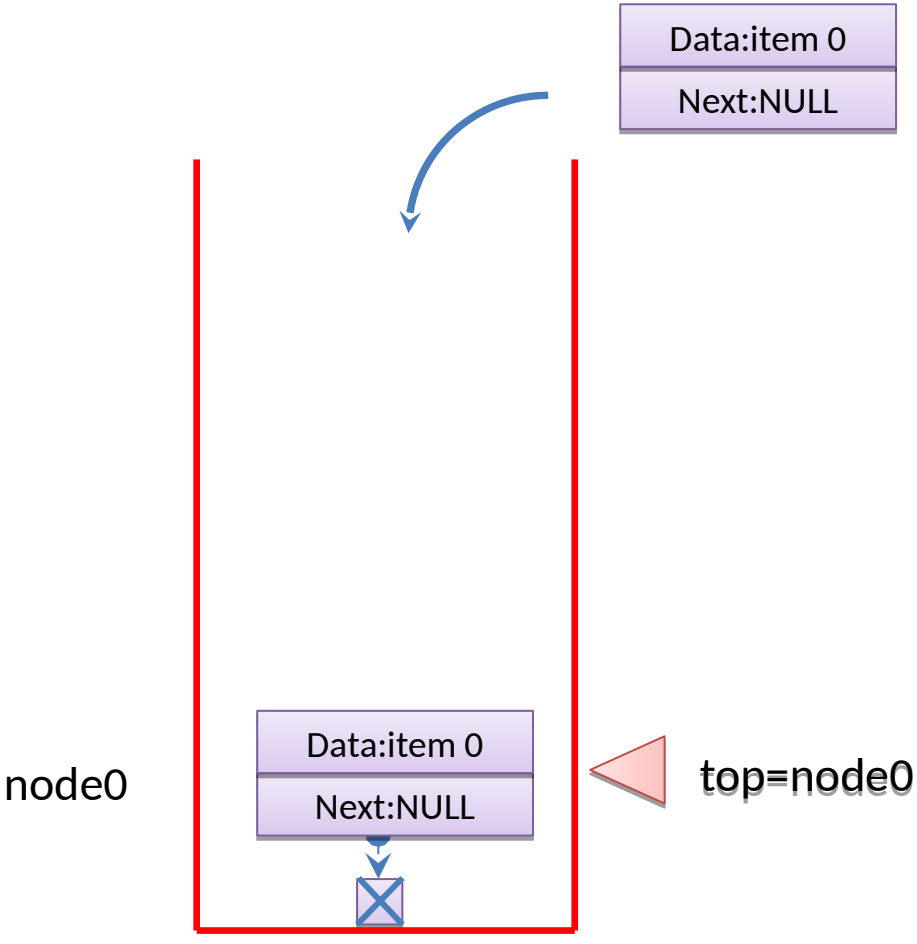
- The nodes that make up the linked list can be used to create the stack data structure.
- Pushing items to a stack can be thought of as adding items to the front of a linked list (push_front(item))
- Deleting an item from the stack is like deleting the first item of the linked list. (pop_front())
- You can obtain the same behavior using push_back() and pop_back()
- The top of the list indicates the top of the stack.

Linked implementation of Stack



- Empty stack
- top=NULL

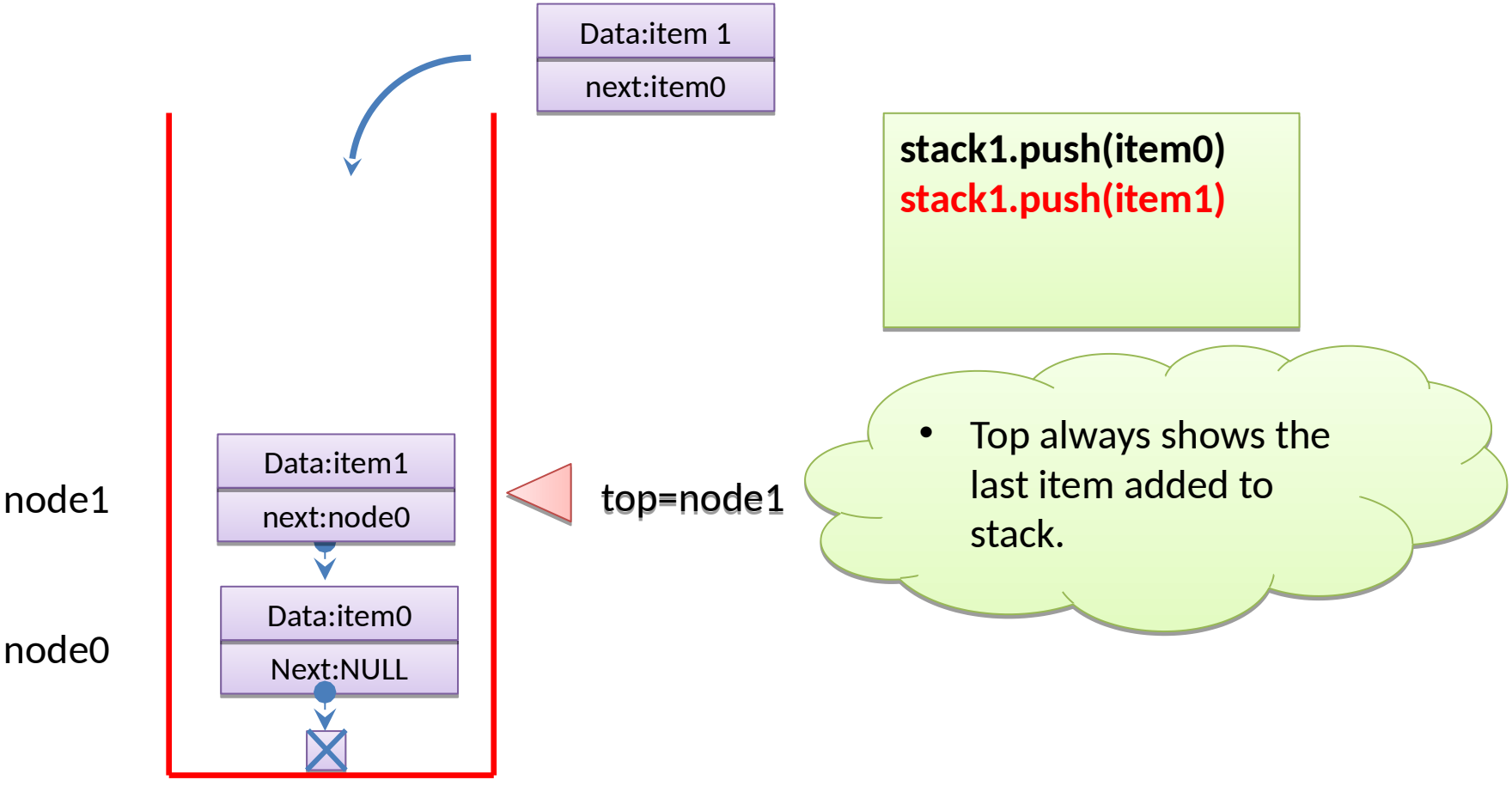
Linked implementation of Stack



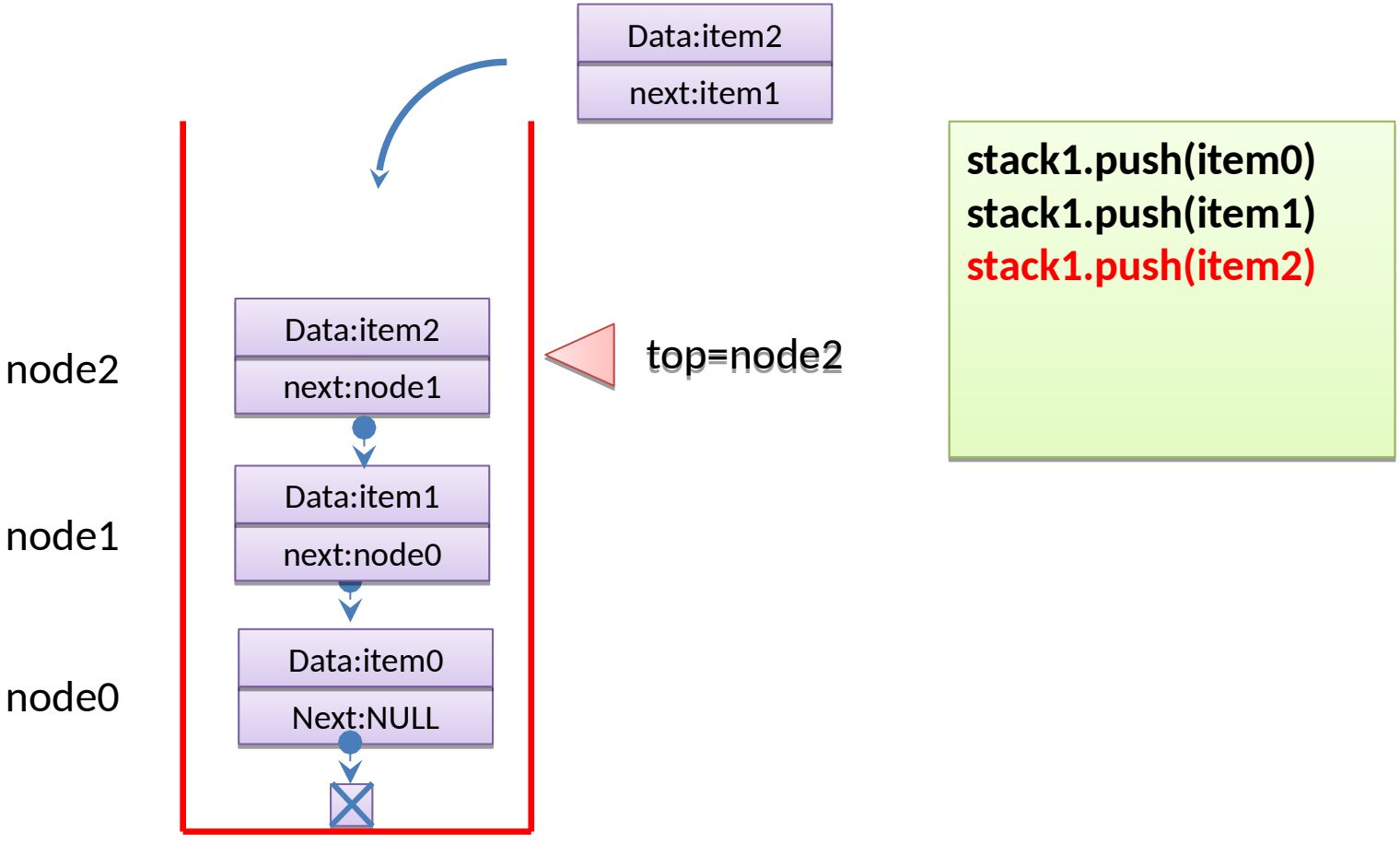
stack1.push(item0)

- Top always shows the last item added to stack.

Linked implementation of Stack

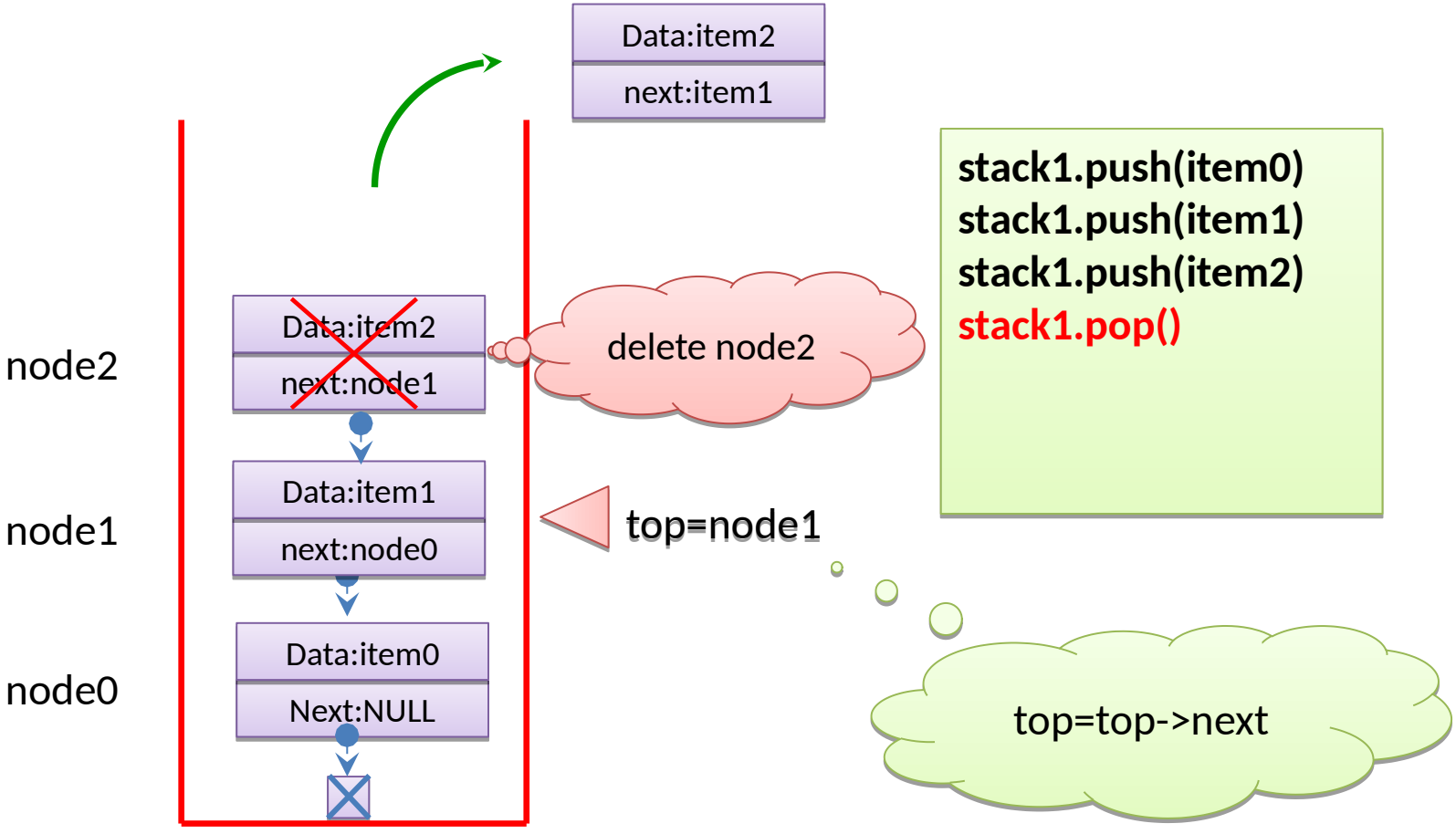


Linked implementation of Stack

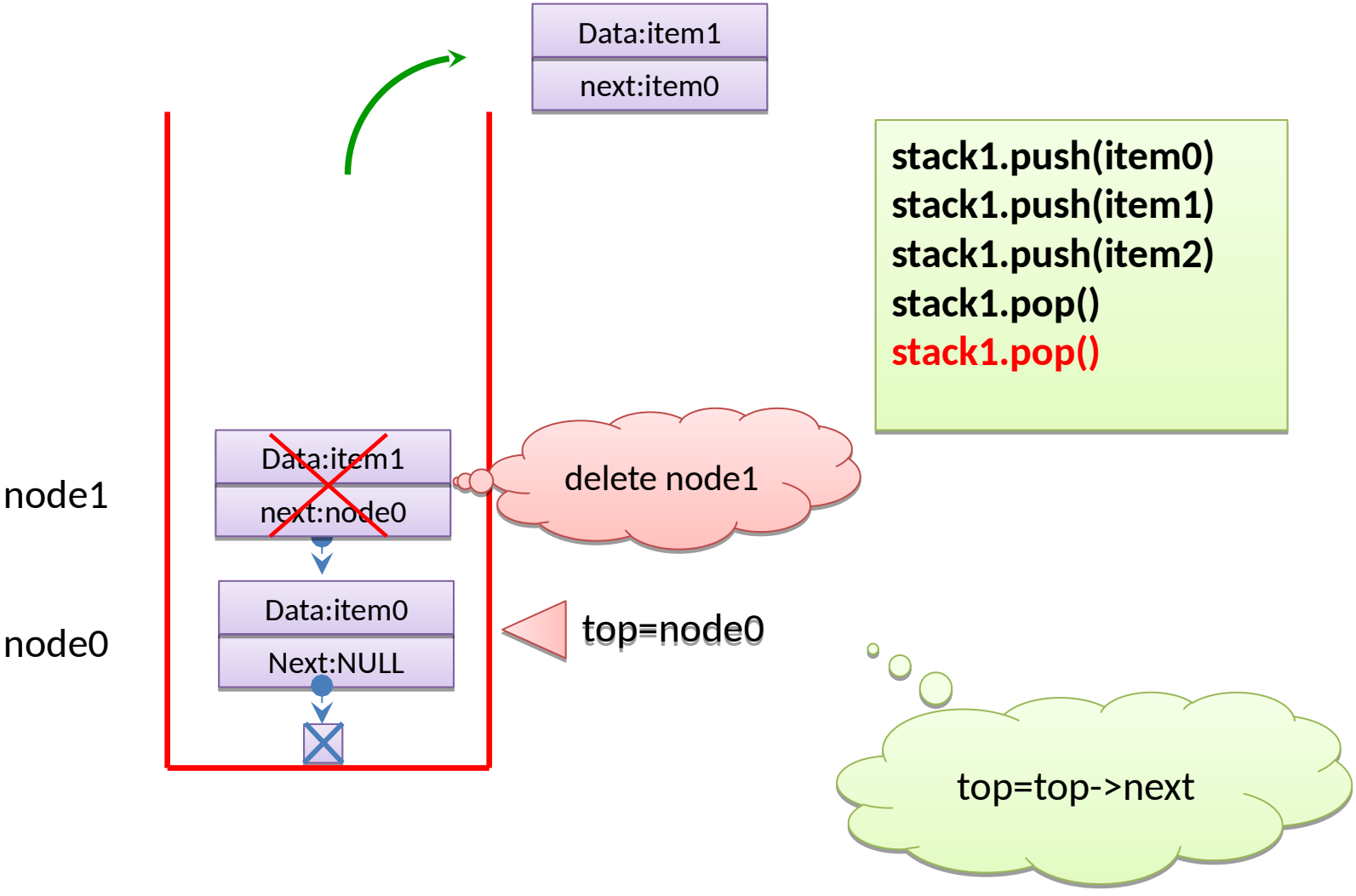


stack1.push(item0)
stack1.push(item1)
stack1.push(item2)

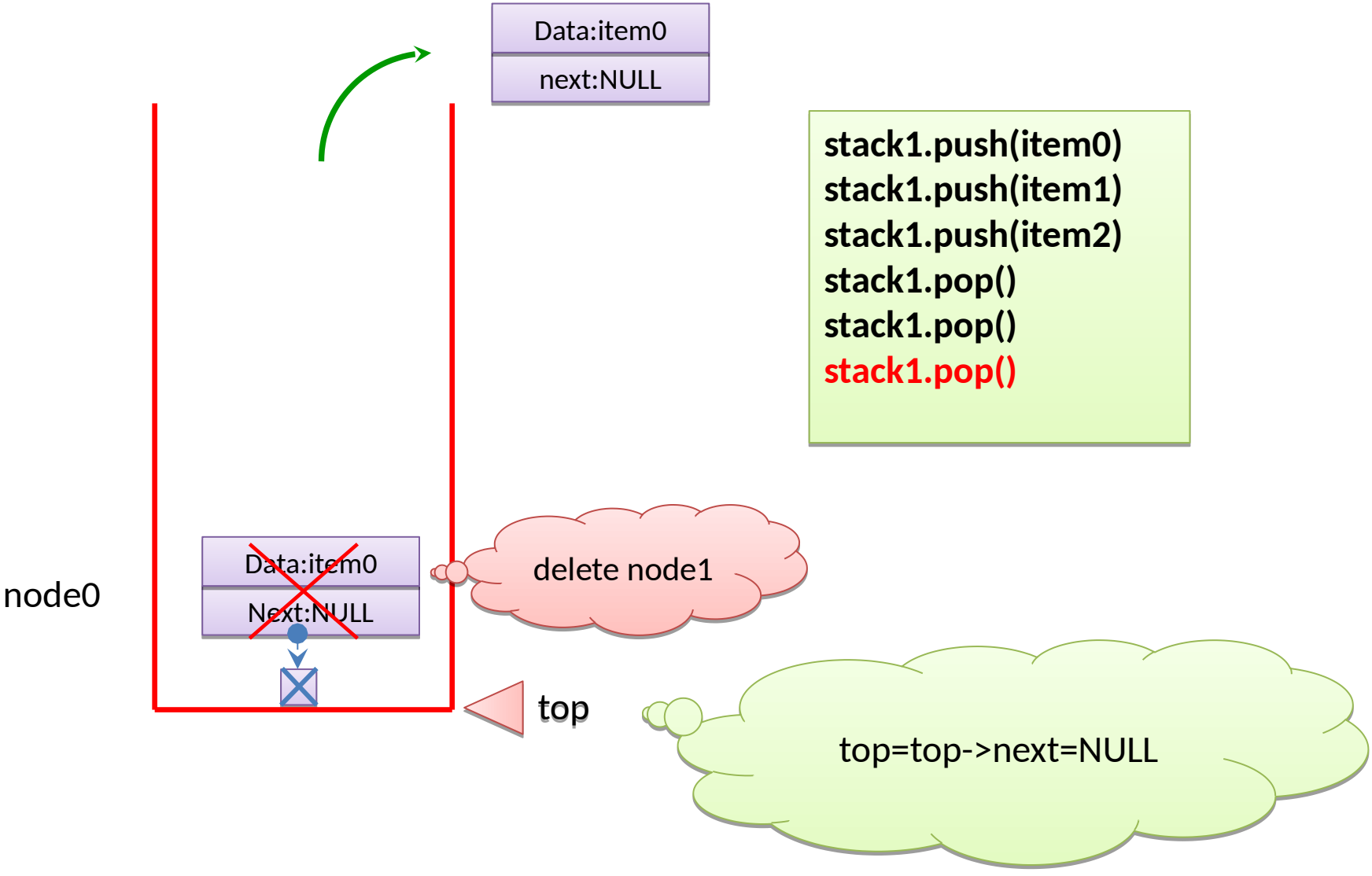
Linked implementation of Stack



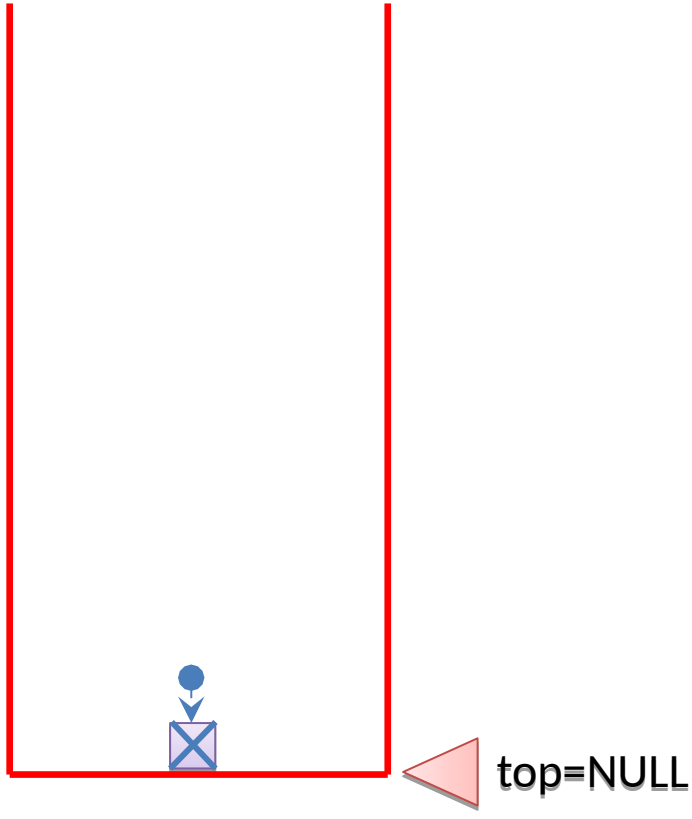
Linked implementation of Stack



Linked implementation of Stack



Linked implementation of Stack




```
stack1.push(item0)
stack1.push(item1)
stack1.push(item2)
stack1.pop()
stack1.pop()
stack1.pop()
```

Stack application for Postfix operations

- The use of parentheses is important in infix notation:
- Example: 5+6*7
  - Summation first:

    (5+6)*7 =11*7=77
  - Multiplication first:

    5+(6*7)=5+42=47

Infix, Prefix, Postfix

- **Infix notation**
- Summing A and B:

  A+B
- Multiplying A and B:

  A*B
- Operators (*,+,-,/) are written between operands (A,B)

Infix, Prefix, Postfix

- **Prefix notation**
- Operator is written first.
- Summing A and B:

  + A B
- Multiplying A and B:

  *** A B**

Infix, Prefix, Postfix

- **Prefix notation**
- + 5 * 6 7 =

  = + 5 42

  = 47
- * + 5 6 7 =

  = * 11 7

  = 77

Operation priority can be defined without using parentheses

Infix, Prefix, Postfix

- **Postfix notation**
- operator is written after
- Summing A and B:

  A B +
- Multiplying A and B:

  A B *

Infix, Prefix, Postfix

- **Postfix notation**
- 5 6 7 * + =

  = 5 42 +

  = 47
- 5 6 + 7 * =

  = 11 7 *

  = 77

Evaluating postfix expression

- Typically, when a compiler computes an infix expression, it first converts it to its postfix form.
- Thus, any uncertainties that may arise are eliminated.

- 5*6+7*8 ⟶ 5 6 * 7 8 * +

Pseudo code for infix to postfix conversion

1. Create an empty stack
2. Convert the input infix string to a list
3. Scan the token list from left to right.
   - If the token is an operand, append it to the end of the output list.
   - If the token is a left parenthesis, push it on the stack
   - If the token is a right parenthesis, pop the stack
   - If the token is an operator, *, /, +, or -, push it on the stack. However, first remove any operators already on the stack that have ***higher precedence (or higher or equal precedence)*** and append them to the output list.
   - When the input expression has been completely processed, pop all the operators from the stack, if any.

http://interactivepython.org/runestone/static/pythonds/BasicDS/InfixPrefixandPostfixExpressions.html

Pseudo code for infix to postfix conversion

Algorithm InfixToPostFix (I)
Transform an infix expression I to a postfix expression P

```
create an empty stack S;
P ← empty expression;
index ← 1;
while we have not reached the end of I do
   ch ← I[index]; {store in ch the next character in I}
   if ch is an operand then
      append ch to the end of P;
   else if ch is a '(' then
      push ch onto S;
   else if ch is a ')' then
      repeat
         pop operators from S and append them to P;
      until a '(' is popped;
   else {ch is an operator}
      while S is not empty and top of S is not '(' and top of S is not a lower precedence operator do
         pop operators from S and append them to P;
      end while
      push ch onto S;
   end if
   index ← index + 1;
end while
```

Operator precedence

- operator precedence (and associativity) is

- -lowest: **+**, **-** (left to right,e.g., 1-2-3 = (1-2)-3)
- -middle: *, **/** (left to right, e.g., 1/2/3 = (1/2)/3)
- - highest: **^** (right to left, e.g., 1^2^3 = 1^(2^3))

Infix to postfix conversion

- **(10+5*3-16/2^3)*(5+7)**

- Output:
- Stack:

Infix to postfix conversion

- (10+5*3-16/2^3)*(5+7)

- Output:
- Stack: (

- Stack history:
- Stack: (

Infix to postfix conversion

- (10+5*3-16/2^3)*(5+7)

- Output: 10
- Stack: (

- Stack history:
- Stack: (

Infix to postfix conversion

- (10+5*3-16/2^3)*(5+7)

- Output: 10
- Stack: (+

- Stack history:
- Stack: (
- Stack: (+

Infix to postfix conversion

- (10+5*3-16/2^3)*(5+7)

- Output: 10 5
- Stack: (+

- Stack history:
- Stack: (
- Stack: (+

Infix to postfix conversion

- (10+5*3-16/2^3)*(5+7)

- Output: 10 5
- Stack: (+ *

- Stack history:
- Stack: (
- Stack: (+
- Stack: (+ *

Infix to postfix conversion

- (10+5*3-16/2^3)*(5+7)

- Output: 10 5 3
- Stack: (+ *

- Stack history:
- Stack: (
- Stack: (+
- Stack: (+ *

Infix to postfix conversion

- **(10+5*3-16/2^3)*(5+7)**

- Output: 10 5 3
- Stack: (+ *

- Stack history:
- Stack: (
- Stack: (+
- Stack: (+ *

Infix to postfix conversion

- **(10+5*3-16/2^3)*(5+7)**

- Output: 10 5 3 *
- Stack: (+

- Stack history:
- Stack: (
- Stack: (+
- Stack: (+ *
- Stack: (+

Infix to postfix conversion

- **(10+5*3-16/2^3)*(5+7)**

- Output: 10 5 3 *
- Stack: (+ -

- Stack history:
- Stack: (
- Stack: (+
- Stack: (+ *
- Stack: (+
- Stack: (+ -

Infix to postfix conversion

- (10+5*3-16/2^3)*(5+7)

- Output: 10 5 3 * 16
- Stack: (+ -

- Stack history:
- Stack: (
- Stack: (+
- Stack: (+ *
- Stack: (+
- Stack: (+ -

Infix to postfix conversion

- **(10+5*3-16/2^3)*(5+7)**

- Output: 10 5 3 * 16
- Stack: (+ - /

- Stack history:
- Stack: (
- Stack: (+
- Stack: (+ *
- Stack: (+
- Stack: (+ -
- Stack: (+ - /

Infix to postfix conversion

- **(10+5*3-16/2^3)*(5+7)**

- Output: 10 5 3 * 16 2
- Stack: (+ - /

- Stack history:
- Stack: (
- Stack: (+
- Stack: (+ *
- Stack: (+
- Stack: (+ -
- Stack: (+ - /

Infix to postfix conversion

- (10+5*3-16/2^3)*(5+7)

- Output: 10 5 3 * 16 2
- Stack: (+ - / ^

- Stack history:
- Stack: (
- Stack: (+
- Stack: (+ *
- Stack: (+
- Stack: (+ -
- Stack: (+ - /
- Stack: (+ - / ^

Infix to postfix conversion

- (10+5*3-16/2^3)*(5+7)

- Output: 10 5 3 * 16 2 3
- Stack: (+ - / ^

- Stack history:
- Stack: (
- Stack: (+
- Stack: (+ *
- Stack: (+
- Stack: (+ -
- Stack: (+ - /
- Stack: (+ - / ^

Infix to postfix conversion

- (10+5*3-16/2^3)*(5+7)

- Output: 10 5 3 * 16 2 3
- Stack: (+ - / ^

When the right parenthesis is read, the operators in between are printed to Output until the left parenthesis is deleted from the stack.

- Stack history:
- Stack: (
- Stack: (+
- Stack: (+ *
- Stack: (+
- Stack: (+ -
- Stack: (+ - /
- Stack: (+ - / ^

Infix to postfix conversion

- (10+5*3-16/2^3)*(5+7)

- Output: 10 5 3 * 16 2 3 ^
- Stack: (+ - /

When the right parenthesis is read, the operators in between are printed to the Output until the left parenthesis is deleted from the stack.

- Stack history:
- Stack: (
- Stack: (+
- Stack: (+ *
- Stack: (+
- Stack: (+ -
- Stack: (+ - /
- Stack: (+ - / ^
- Stack: (+ - /

Infix to postfix conversion

- (10+5*3-16/2^3)*(5+7)

- Output: 10 5 3 * 16 2 3 ^ /
- Stack: (+ -

When the right parenthesis is read, the operators in between are printed to Output until the left parenthesis is deleted from the stack.

- Stack history:
- Stack: (
- Stack: (+
- Stack: (+ *
- Stack: (+
- Stack: (+ -
- Stack: (+ - /
- Stack: (+ - / ^
- Stack: (+ - /
- Stack: (+ -

Infix to postfix conversion

- (10+5*3-16/2^3)*(5+7)

- Output: 10 5 3 * 16 2 3 ^ /
- Stack: (+

When the right parenthesis is read, the operators in between are printed to Output until the left parenthesis is deleted from the stack.

- Stack history:
- Stack: (
- Stack: (+
- Stack: (+ *
- Stack: (+
- Stack: (+ -
- Stack: (+ - /
- Stack: (+ - / ^
- Stack: (+ - /
- Stack: (+ -
- Stack: (+

Infix to postfix conversion

- (10+5*3-16/2^3)*(5+7)

- Output: 10 5 3 * 16 2 3 ^ / -
- Stack: (

When the right parenthesis is read, the operators in between are printed to Output until the left parenthesis is deleted from the stack.

- Stack history:
- Stack: (
- Stack: (+
- Stack: (+ *
- Stack: (+
- Stack: (+ -
- Stack: (+ - /
- Stack: (+ - / ^
- Stack: (+ - /
- Stack: (+ -
- Stack: (

Infix to postfix conversion

- (10+5*3-16/2^3)*(5+7)

- Output: 10 5 3 * 16 2 3 ^ / -
- Stack:

When the right parenthesis is read, the operators in between are printed to Output until the left parenthesis is deleted from the stack.

- Stack history:
- Stack: (
- Stack: (+
- Stack: (+ *
- Stack: (+
- Stack: (+ -
- Stack: (+ - /
- Stack: (+ - / ^
- Stack: (+ - /
- Stack: (+ -
- Stack:

Infix to postfix conversion

- **(10+5*3-16/2^3)*(5+7)**

- Output: 10 5 3 * 16 2 3 ^ / - +
- Stack: *

- Stack history:
- Stack: (
- Stack: (+
- Stack: (+ *
- Stack: (+
- Stack: (+ -
- Stack: (+ - /
- Stack: (+ - / ^
- Stack: (+ - /
- Stack: (+ -
- Stack:
- Stack: *

Infix to postfix conversion

- **(10+5*3-16/2^3)*(5+7)**

- Output: 10 5 3 * 16 2 3 ^ / - +
- Stack: * (

- Stack history:
- Stack: (
- Stack: (+
- Stack: (+ *
- Stack: (+
- Stack: (+ -
- Stack: (+ - /
- Stack: (+ - / ^
- Stack: (+ - /
- Stack: (+ -
- Stack:
- Stack: *
- Stack: *(

Infix to postfix conversion

- (10+5*3-16/2^3)*(5+7)

- Output: 10 5 3 * 16 2 3 ^ / - + 5
- Stack: * (

- Stack history:
- Stack: (
- Stack: (+
- Stack: (+ *
- Stack: (+
- Stack: (+ -
- Stack: (+ - /
- Stack: (+ - / ^
- Stack: (+ - /
- Stack: (+ -
- Stack:
- Stack: *
- Stack: *(

Infix to postfix conversion

- **(10+5*3-16/2^3)*(5+7)**

- Output: 10 5 3 * 16 2 3 ^ / - + 5
- Stack: * (+

- Stack history:
- Stack: (
- Stack: (+
- Stack: (+ *
- Stack: (+
- Stack: (+ -
- Stack: (+ - /
- Stack: (+ - / ^
- Stack: (+ - /
- Stack: (+ -
- Stack:
- Stack: *
- Stack: *(
- Stack: *(+

Infix to postfix conversion

- (10+5*3-16/2^3)*(5+7)

- Output: 10 5 3 * 16 2 3 ^ / - + 5 7
- Stack: * (+

- Stack history:
- Stack: (
- Stack: (+
- Stack: (+ *
- Stack: (+
- Stack: (+ -
- Stack: (+ - /
- Stack: (+ - / ^
- Stack: (+ - /
- Stack: (+ -
- Stack:
- Stack: *
- Stack: *(
- Stack: *(+

Infix to postfix conversion

- (10+5*3-16/2^3)*(5+7)

- Output: 10 5 3 * 16 2 3 ^ / - + 5 7
- Stack: * (+

- Stack history:
- Stack: (
- Stack: (+
- Stack: (+ *
- Stack: (+
- Stack: (+ -
- Stack: (+ - /
- Stack: (+ - / ^
- Stack: (+ - /
- Stack: (+ -
- Stack:
- Stack: *
- Stack: *(
- Stack: *(+

Infix to postfix conversion

- (10+5*3-16/2^3)*(5+7)

- Output: 10 5 3 * 16 2 3 ^ / - + 5 7 +
- Stack: * (

- Stack history:
- Stack: (
- Stack: (+
- Stack: (+ *
- Stack: (+
- Stack: (+ -
- Stack: (+ - /
- Stack: (+ - / ^
- Stack: (+ - /
- Stack: (+ -
- Stack:
- Stack: *
- Stack: *(
- Stack: *(+

Infix to postfix conversion

- **(10+5*3-16/2^3)*(5+7)**

- Output: 10 5 3 * 16 2 3 ^ / - + 5 7 +
- Stack: *

- Stack history:
- Stack: (
- Stack: (+
- Stack: (+ *
- Stack: (+
- Stack: (+ -
- Stack: (+ - /
- Stack: (+ - / ^
- Stack: (+ - /
- Stack: (+ -
- Stack:
- Stack: *
- Stack: *(
- Stack: *(+
- Stack: *(
- Stack: *

Infix to postfix conversion

- (10+5*3-16/2^3)*(5+7)

- Output: 10 5 3 * 16 2 3 ^ / - + 5 7 + *
- Stack:

Conversion complete

- Stack history:
- Stack: (
- Stack: (+
- Stack: (+ *
- Stack: (+
- Stack: (+ -
- Stack: (+ - /
- Stack: (+ - / ^
- Stack: (+ - /
- Stack: (+ -
- Stack:
- Stack: *
- Stack: *(
- Stack: *(+
- Stack: *(
- Stack: *
- Stack:

Example

Infix: A * (B + C * D) + E

Postfix: A B C D * + * E +

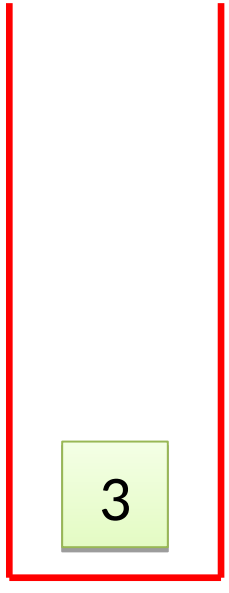
	current symbol	operator stack	postfix string
1	A		A
2	*	*	A
3	(	* (	A
4	B	* (	A B
5	+	* (+	A B
6	C	* (+	A B C
7	*	* (+ *	A B C
8	D	* (+ *	A B C D
9	)	*	A B C D * +
10	+	+	A B C D * + *
11	E	+	A B C D * + * E
12			A B C D * + * E +

Example

- Infix:(5+4-6)*(8+10)/((1+5)*(5-2))
- Postfix: 5 4 6 - + 8 10 + 1 5 + 5 2 - * / *
- Infix: 5-16/(4*2^2)
- Postfix: 5 16 4 2 2 ^ * / -
- Infix: 5-16/4*2^2
- Postfix: 5 16 4 / 2 2 ^ * -
- Infix: (5-16)/4*2^2
- Postfix: 5 16- 4 / 2 2 ^ *

Computing a postfix expression

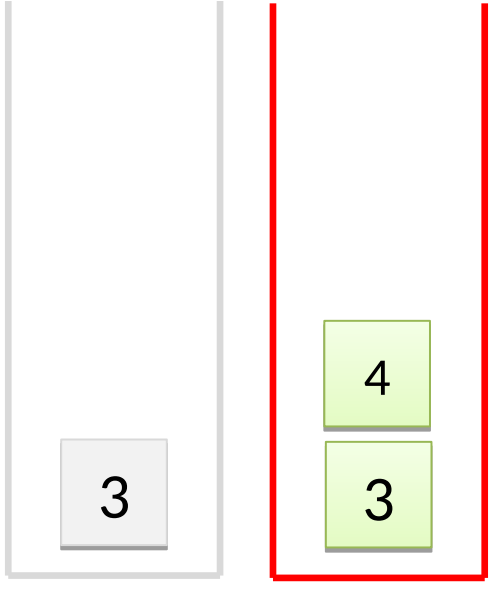
- Example: 3 4 + 5 6 * 9 2 - + *



- While the items are pushed to the stack, when it comes to an operator in the statement, the last two statements in the stack are processed.
- The last two items are removed and the result is written to the stack again.

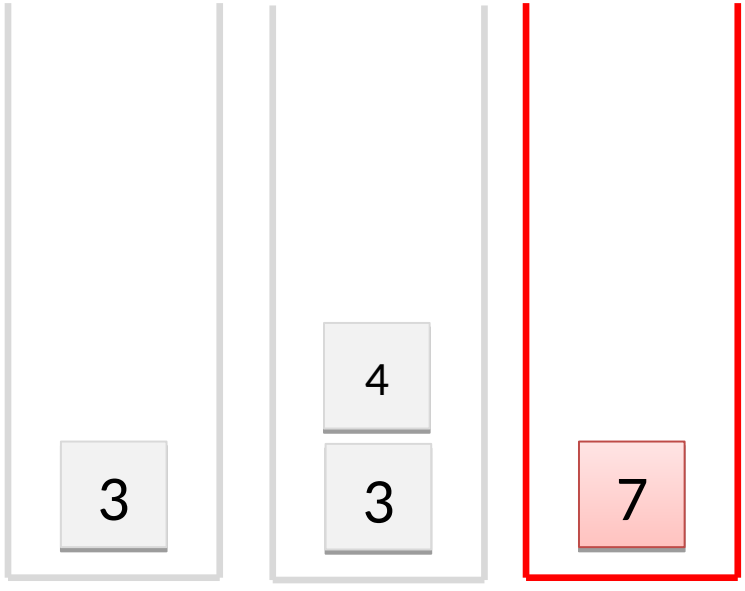
Computing a postfix expression

- Example: 3 4 + 5 6 * 9 2 - + *



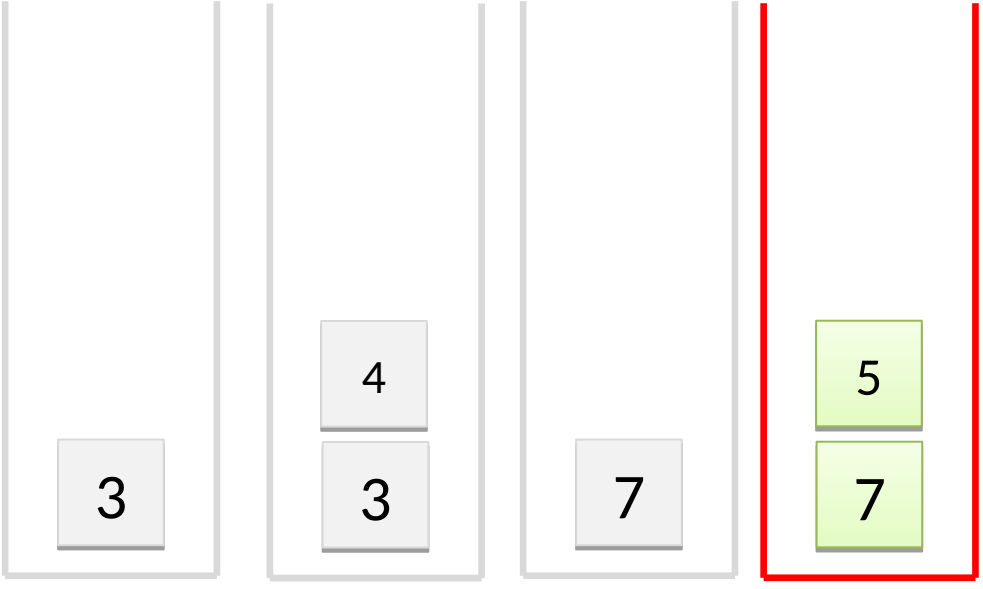
Computing a postfix expression

- Example: 3 4 + 5 6 * 9 2 - + *



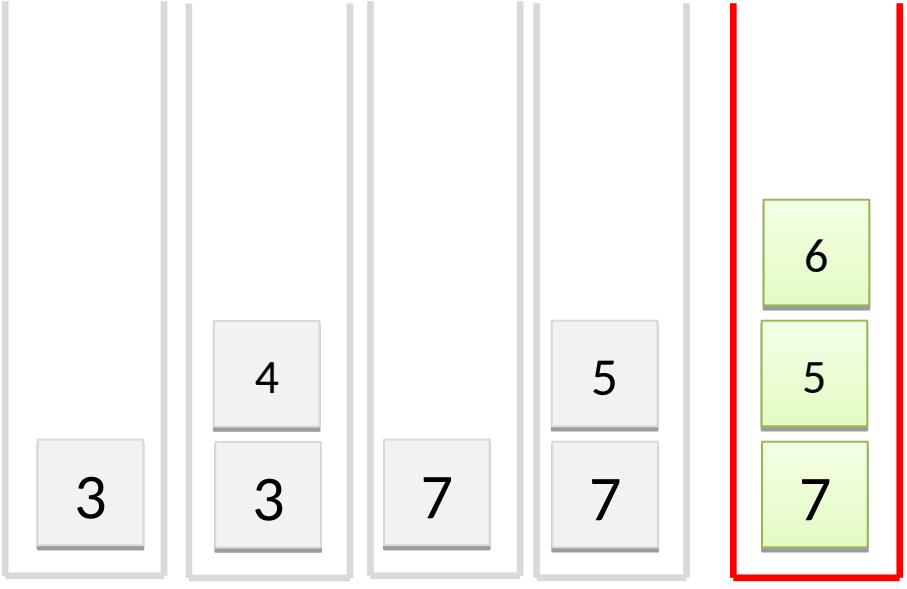
Computing a postfix expression

- Example: 3 4 + 5 6 * 9 2 - + *



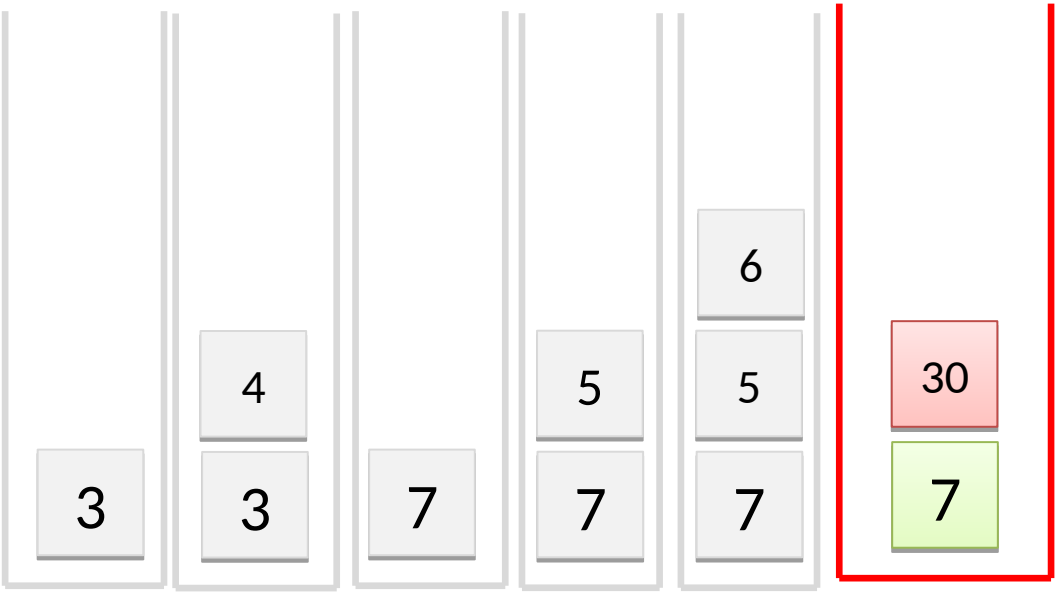
Computing a postfix expression

- Example: 3 4 + 5 6 * 9 2 - + *



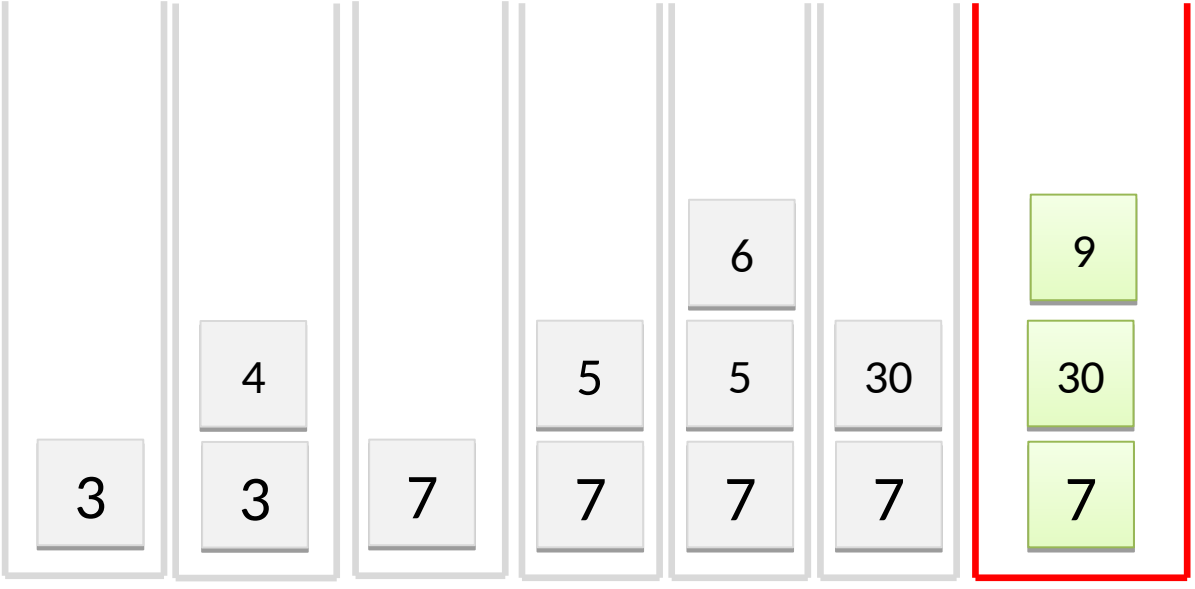
Computing a postfix expression

- Example: 3 4 + 5 6 * 9 2 - + *



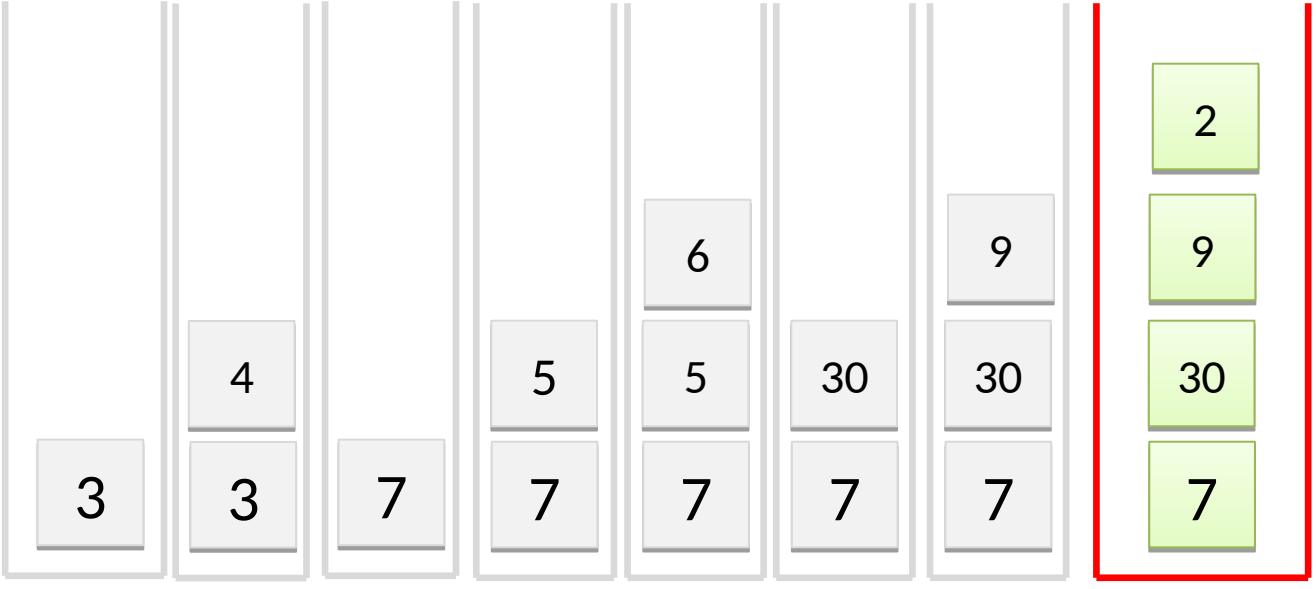
Computing a postfix expression

- Example: 3 4 + 5 6 * 9 2 - + *



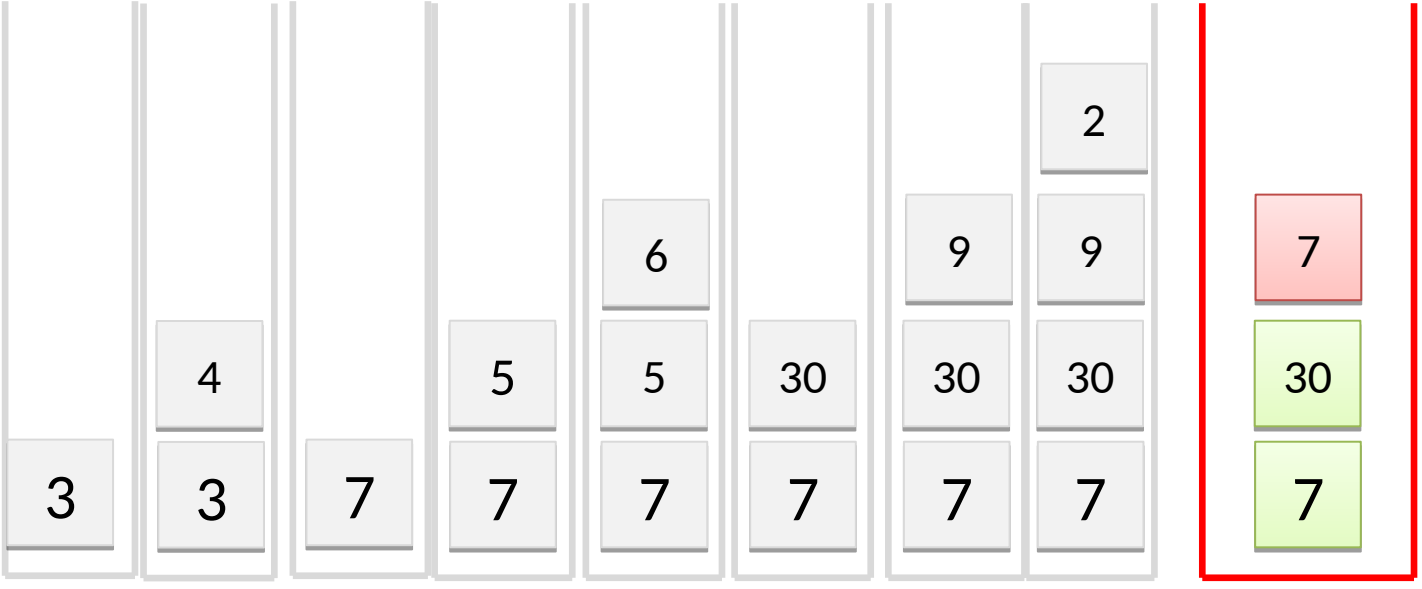
Computing a postfix expression

- Example: 3 4 + 5 6 * 9 2 - + *



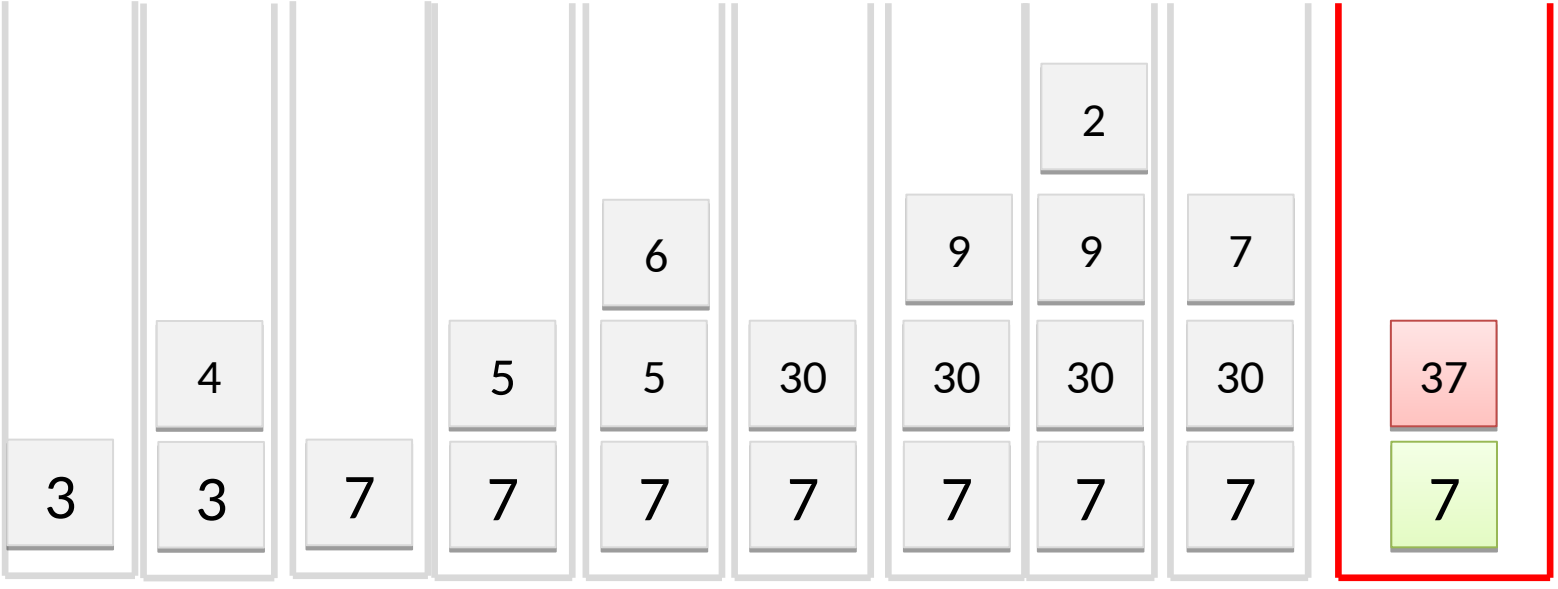
Computing a postfix expression

- Example: 3 4 + 5 6 * 9 2 - + *



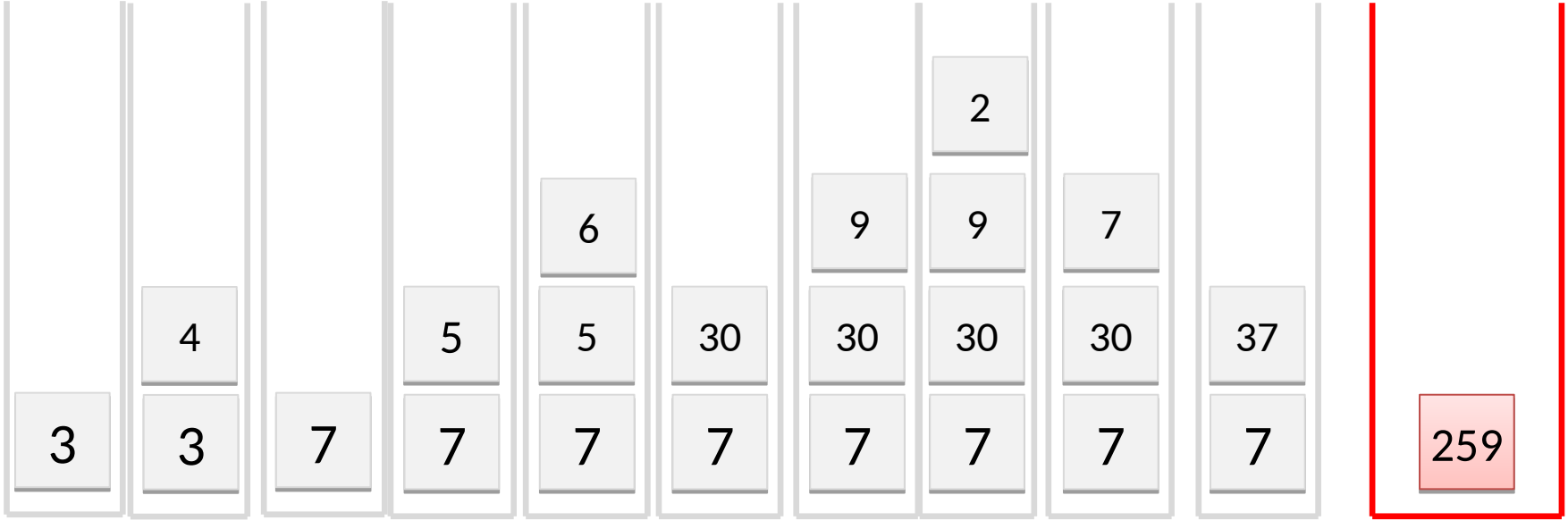
Computing a postfix expression

- Example: 3 4 + 5 6 * 9 2 - + *



Computing a postfix expression

- Example: 3 4 + 5 6 * 9 2 - + *



Algoritma- Computing a postfix expression

- Suppose P is an arithmetic expression in postfix notation. We will evaluate it using a stack to hold the operands.

```
Start with an empty stack. We scan P from left to right.
While (we have not reached the end of P)
  If an operand is found
    push it onto the stack
  End-If
  If an operator is found
    Pop the stack and call the value A
    Pop the stack and call the value B
    Evaluate B op A using the operator just found.
    Push the resulting value onto the stack
  End-If
End-While
Pop the stack (this is the final value)
```

- Notes:
- At the end, there should be only one element left on the stack.
- This assumes the postfix expression is valid.

- http://faculty.cs.niu.edu/~hutchins/csci241/eval.htm

C++ Stack ADT

Example

```
1  // stack::push/pop
2  #include <iostream>       // std::cout
3  #include <stack>          // std::stack
4
5  int main ()
6  {
7    std::stack<int> mystack;
8
9    for (int i=0; i<5; ++i) mystack.push(i);
10
11   std::cout << "Popping out elements...";
12   while (!mystack.empty())
13   {
14      std::cout << ' ' << mystack.top();
15      mystack.pop();
16   }
17   std::cout << '\n';
18
19   return 0;
20 }
```

http://www.cplusplus.com/reference/stack/stack/push/

Java and C#

java.util

Class Stack<E>

java.lang.Object
java.util.AbstractCollection<E>
java.util.AbstractList<E>
java.util.Vector<E>
java.util.Stack<E>

```
import java.util.Stack;

public class Program {
    public static void main(String[] args) {

        Stack<String> stack = new Stack<>();
        stack.push("fly");
        stack.push("worm");
        stack.push("butterfly");

        // Peek at the top of the stack.
        String peekResult = stack.peek();
        System.out.println(peekResult);

        // Pop the stack and display the result.
        String popResult = stack.pop();
        System.out.println(popResult);

        // Pop again.
        popResult = stack.pop();
        System.out.println(popResult);
    }
}
```

Output

```
butterfly
butterfly
```

http://www.dotnetperls.com/stack-java

Namespace: System.Collections
Assembly: mscorlib (in mscorlib.dll)

Inheritance Hierarchy

```
using System;
using System.Collections;
public class SamplesStack  {

   public static void Main()  {

      // Creates and initializes a new Stack.
      Stack myStack = new Stack();
      myStack.Push("Hello");
      myStack.Push("World");
      myStack.Push("!");

      // Displays the properties and values of the Stack.
      Console.WriteLine( "myStack" );
      Console.WriteLine( "\tCount:    {0}", myStack.Count );
      Console.Write( "\tValues:" );
      PrintValues( myStack );
   }

   public static void PrintValues( IEnumerable myCollection )  {
      foreach ( Object obj in myCollection )
         Console.Write( "    {0}", obj );
      Console.WriteLine();
   }

}
```

https://msdn.microsoft.com/en-us/library/system.collections.stack(v=vs.110).aspx

Example

```
int main(int argc, char** argv) {
    stack<int> stack1;
    for (int i=1;i<=5;i++){
        stack1.push(i*10);
    }


    stack<int> stack2;
    while (!stack1.empty()){
        stack2.push(stack1.top());
        stack1.pop();
    }


    while (!stack2.empty()){
        cout<<"stack1.top()="<<stack2.top()<<endl;
        stack2.pop();
    }
```

What will be the console output of the program given on the left?