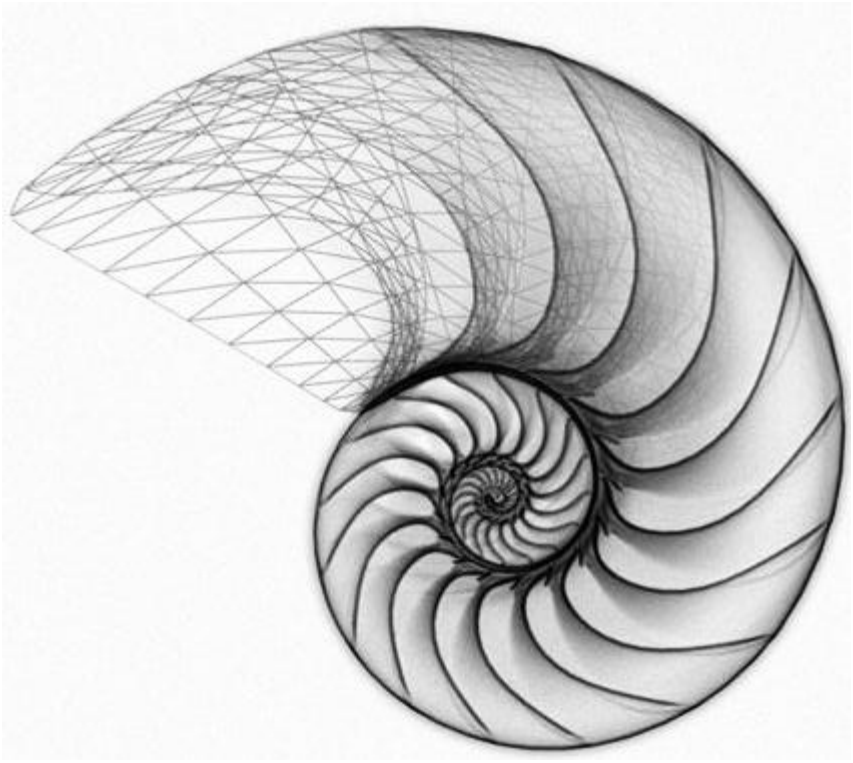
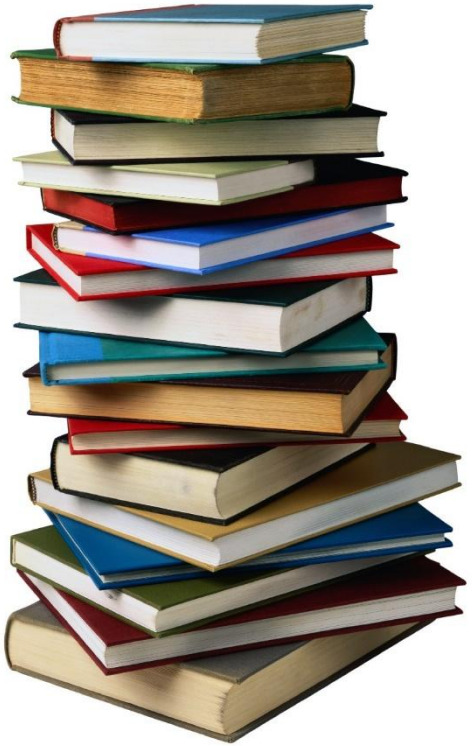


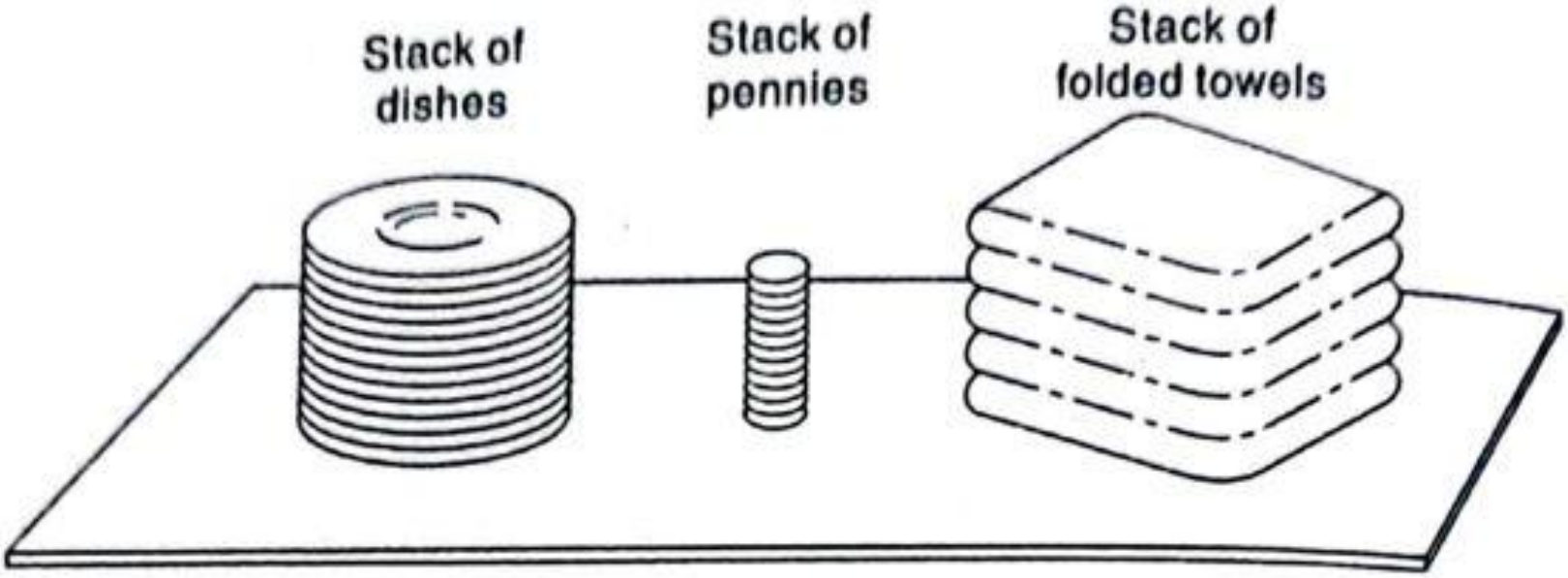
Lecture 15

Abstract Data Type Stack
(Array-based Implementation)

Stack

- A list
- Data items can be added and deleted
- Maintains **Last In First Out (LIFO)** order





A *stack* is a list of elements in which an element may be inserted or deleted only at one end, called the *top* of the stack. This means, in particular, that elements are removed from a stack in the reverse order of that in which they were inserted into the stack.

Special terminology is used for two basic operations associated with stacks:

(a) "Push" is the term used to insert an element into a stack.
(b) "Pop" is the term used to delete an element from a stack.

We emphasize that these terms are used only with stacks, not with other data structures.

Specification of `StackType`

Structure: Elements are added to and removed from the top of the stack.

Definitions (provided by user):

MAX_ITEMS	Maximum number of items that might be on the stack.
ItemType	Data type of the items on the stack.

Operations (provided by the ADT):

MakeEmpty

Function	Sets stack to an empty state.
Postcondition	Stack is empty.

Boolean IsEmpty

Function	Determines whether the stack is empty.
Precondition	Stack has been initialized.
Postcondition	Returns true if stack is empty and false otherwise.

Boolean IsFull

Function	Determines whether the stack is full.
Precondition	Stack has been initialized.
Postcondition	Returns true if stack is full and false otherwise.

Specification of `StackType`

Push(ItemType newItem)

Function	Adds newItem to the top of the stack.
Precondition	Stack has been initialized.
Postcondition	If (stack is full), exception FullStack is thrown, else newItem is at the top of the stack.

Pop()

Function	Removes top item from the stack.
Precondition	Stack has been initialized.
Postcondition	If (stack is empty), exception EmptyStack is thrown, else top element has been removed from stack.

ItemType Top()

Function	Returns a copy of the top item on the stack.
Precondition	Stack has been initialized.
Postcondition	If (stack is empty), exception EmptyStack is thrown, else a copy of the top element is returned.

stacktype.h

```
#ifndef STACKTYPE_H_INCLUDED
#define STACKTYPE_H_INCLUDED

const int MAX_ITEMS = 5;

class FullStack
{};  // Exception class thrown by Push when stack is full.
class EmptyStack
{};  // Exception class thrown by Pop and Top when stack is empty.

template <class ItemType>
class StackType
{
    public:
        StackType();
        bool IsFull();
        bool IsEmpty();
        void MakeEmpty();
        void Push(ItemType);
        void Pop();
        ItemType Top();
    private:
        int top;
        ItemType  items[MAX_ITEMS];
};
#endif // STACKTYPE_H_INCLUDED
```

stacktype.cpp

```
#include "StackType.h"
template <class ItemType>

StackType<ItemType>::StackType()
{
    top = -1;
}
template <class ItemType>
bool StackType<ItemType>::IsEmpty()
{
    return (top == -1);
}
void StackType<ItemType>::MakeEmpty()
{
    top = -1;
}
template <class ItemType>
bool StackType<ItemType>::IsFull()
{
    return (top ==  MAX_ITEMS-1);
}
```

```
template <class ItemType>
void
StackType<ItemType>::Push(ItemType
newItem)
{
    if( IsFull() )
        throw FullStack();
    top++;
    items[top] = newItem;
}
template <class ItemType>
void StackType<ItemType>::Pop()
{
    if( IsEmpty() )
        throw EmptyStack();
    top--;
}
template <class ItemType>
ItemType StackType<ItemType>::Top()
{
    if (IsEmpty())
        throw EmptyStack();
    return items[top];
}
```

stacktype.cpp

```
#include "StackType.h"
template <class ItemType>

StackType<ItemType>::StackType()
{
    top = -1;
}
```

O(1)

```
template <class ItemType>
bool StackType<ItemType>::IsEmpty()
{
    return (top == -1);
}
```

O(1)

```
void StackType<ItemType>::MakeEmpty()
{
    top = -1;
}
```

O(1)

```
template <class ItemType>
bool StackType<ItemType>::IsFull()
{
    return (top ==  MAX_ITEMS-1);
}
```

O(1)

```
template <class ItemType>
void
StackType<ItemType>::Push(ItemType
newItem)
{
    if( IsFull() )
        throw FullStack();
    top++;
    items[top] = newItem;
}
```

O(1)

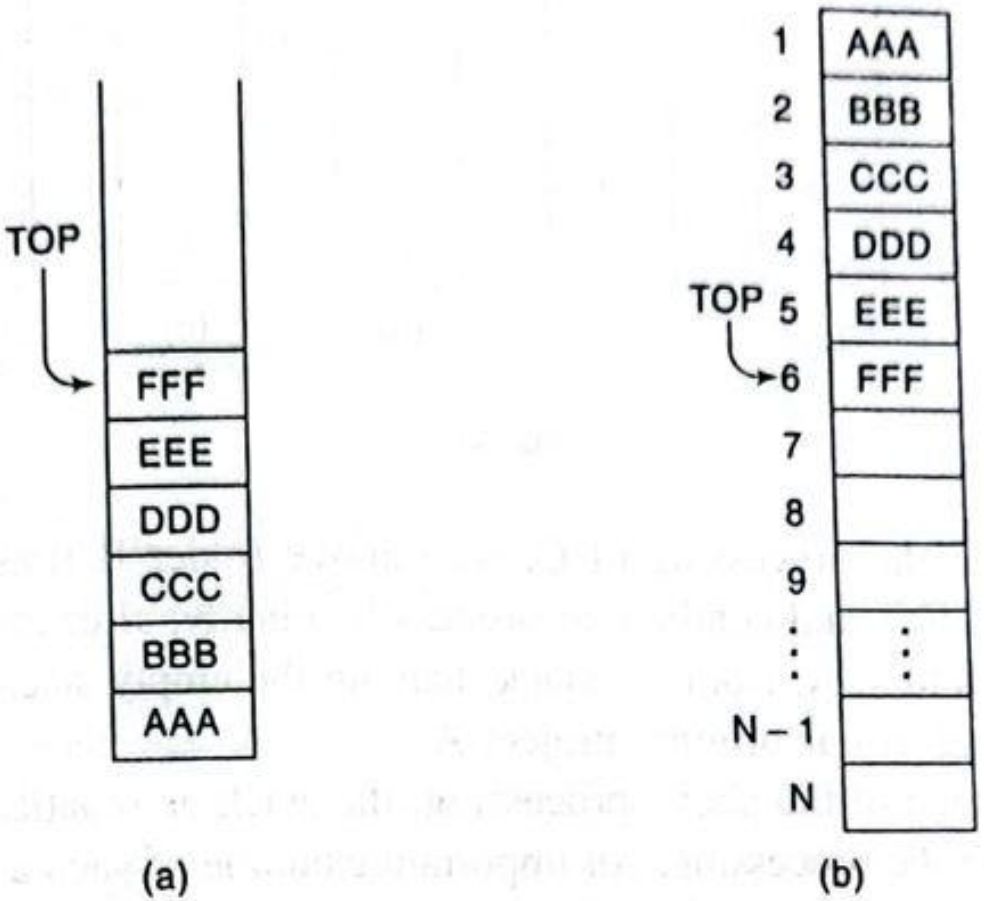
```
template <class ItemType>
void StackType<ItemType>::Pop()
{
    if( IsEmpty() )
        throw EmptyStack();
    top--;
}
```

O(1)

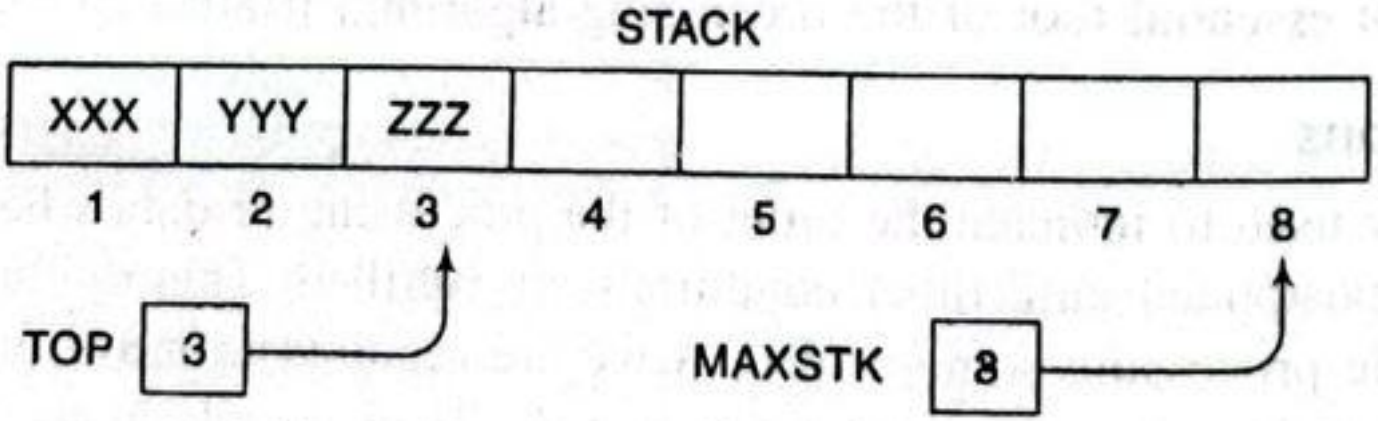
```
template <class ItemType>
ItemType StackType<ItemType>::Top()
{
    if (IsEmpty())
        throw EmptyStack();
    return items[top];
}
```

O(1)

ARRAY REPRESENTATION OF STACKS



ARRAY REPRESENTATION OF STACKS



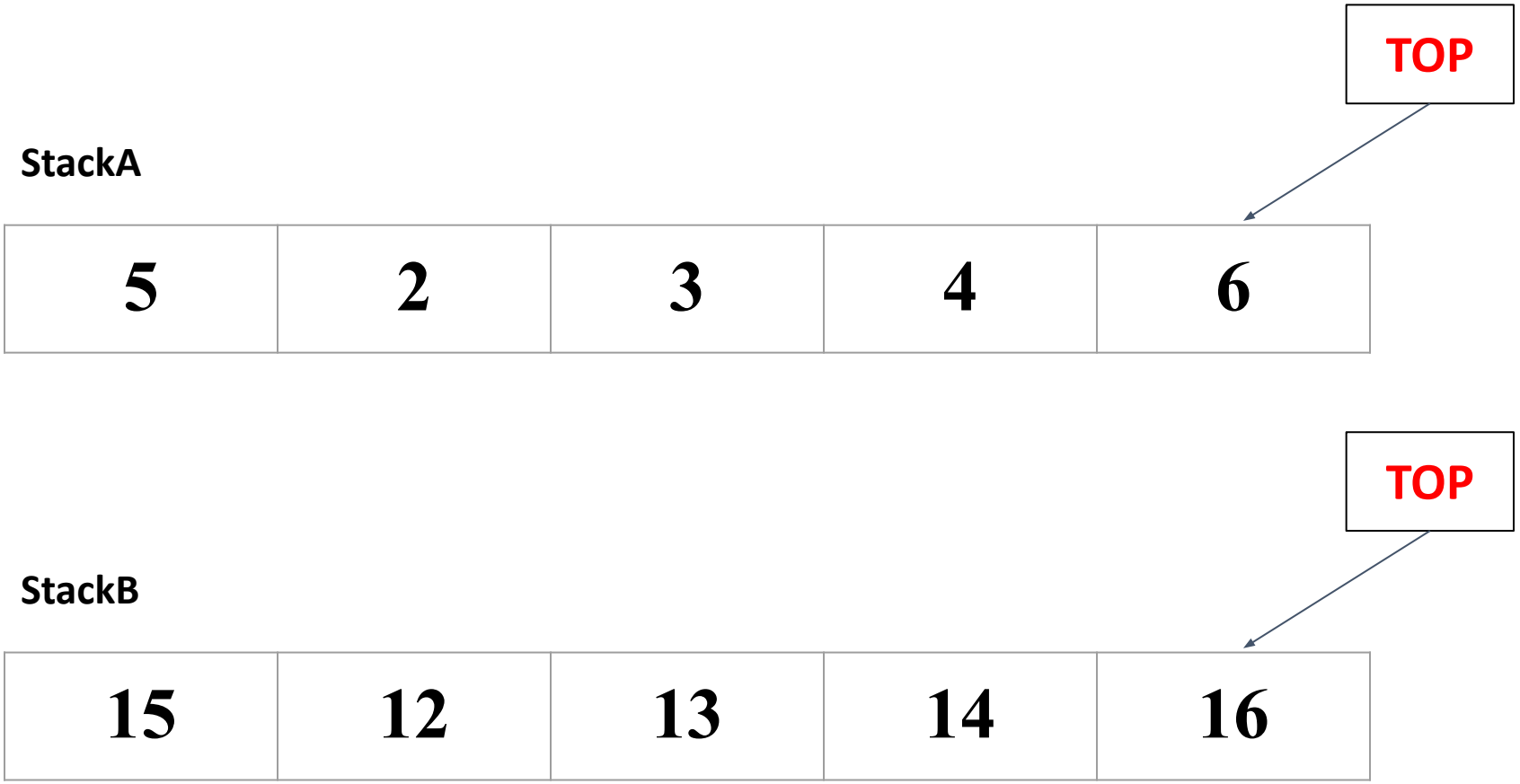
PUSH(STACK, TOP, MAXSTK, ITEM)
This procedure pushes an ITEM onto a stack.

1. [Stack already filled?]
   If TOP = MAXSTK, then: Print: OVERFLOW, and Return.
2. Set TOP := TOP + 1. [Increases TOP by 1.]
3. Set STACK[TOP] := ITEM. [Inserts ITEM in new TOP position.]
4. Return.

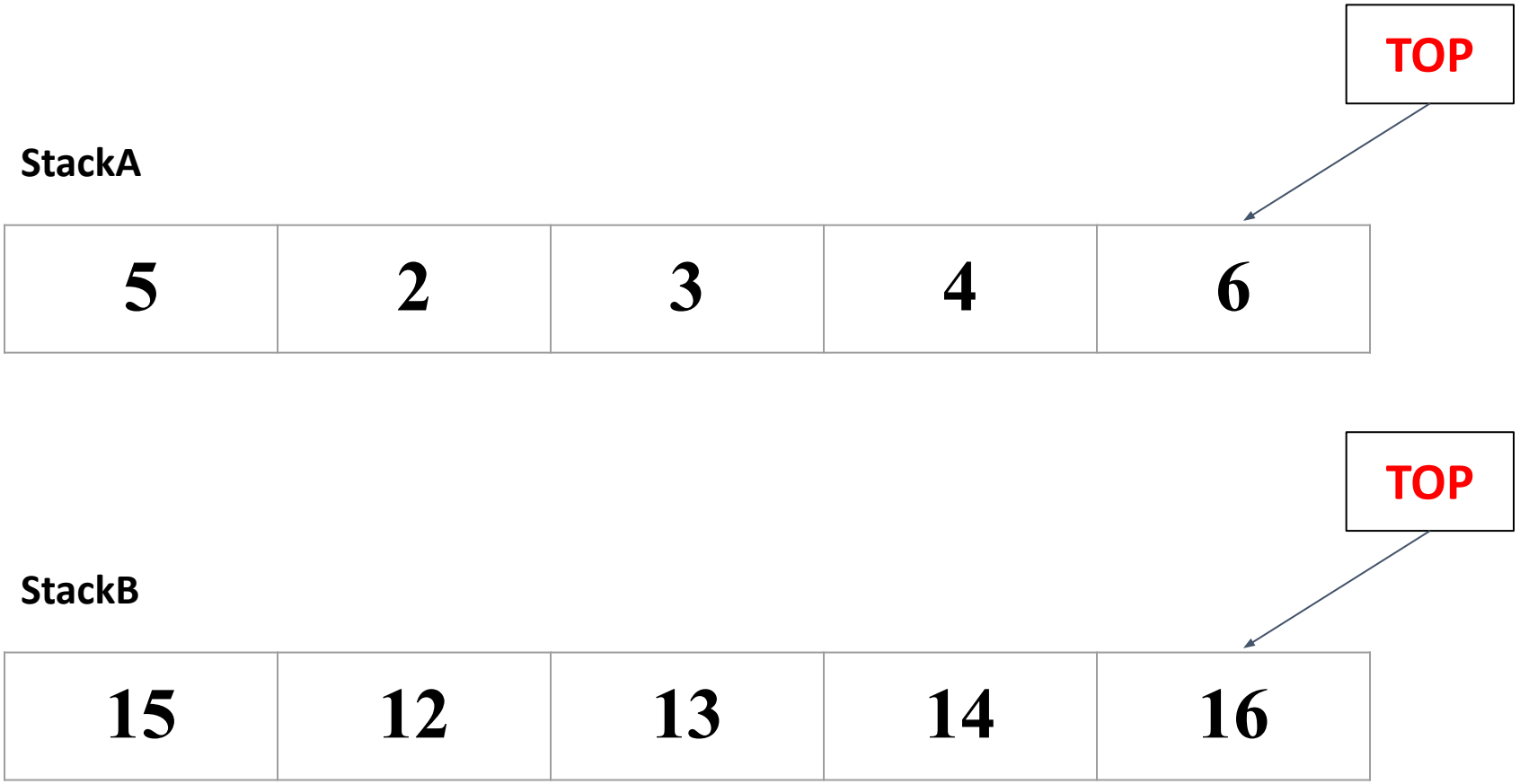
POP(STACK, TOP, ITEM)
This procedure deletes the top element of STACK and assigns it to the variable ITEM.

1. [Stack has an item to be removed?]
   If TOP = 0, then: Print: UNDERFLOW, and Return.
2. Set ITEM := STACK[TOP]. [Assigns TOP element to ITEM.]
3. Set TOP := TOP – 1. [Decreases TOP by 1.]
4. Return.

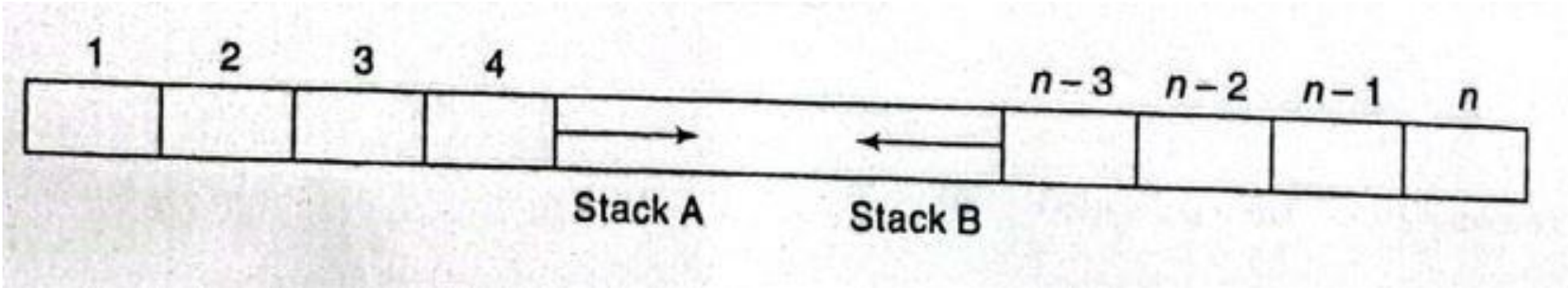
Stack Overflow



Can you reduce the number of times overflow occurs even though we have not increased the total amount of space reserved for the two stacks?



Can you reduce the number of times overflow occurs even though we have not increased the total amount of space reserved for the two stacks?



LINKEDLIST REPRESENTATION OF STACKS

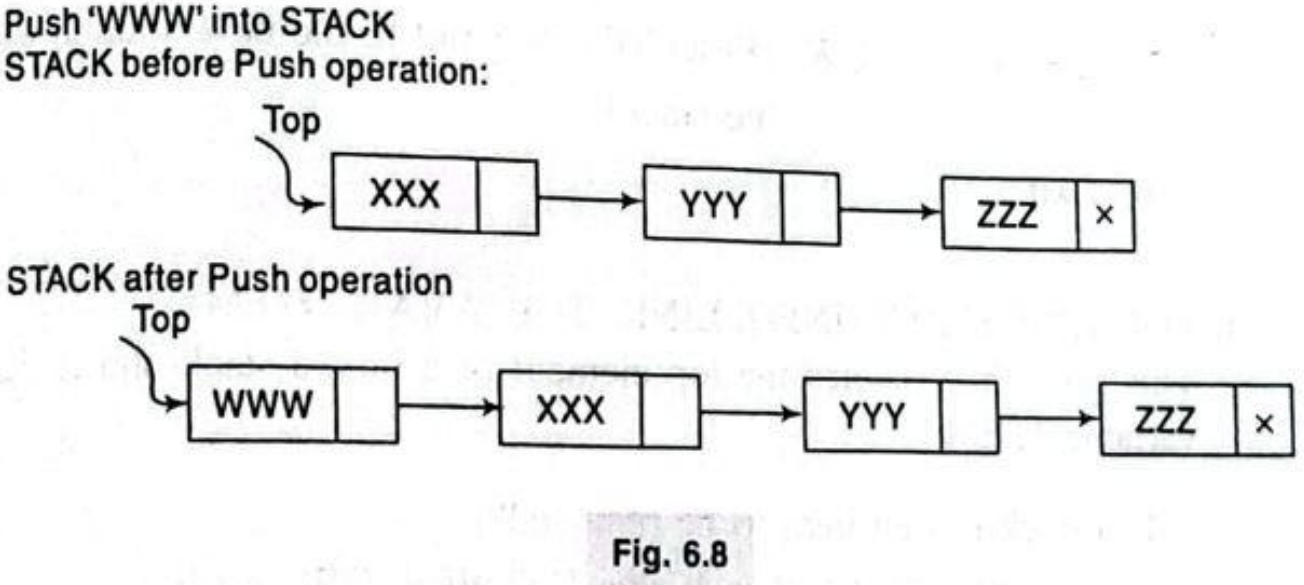
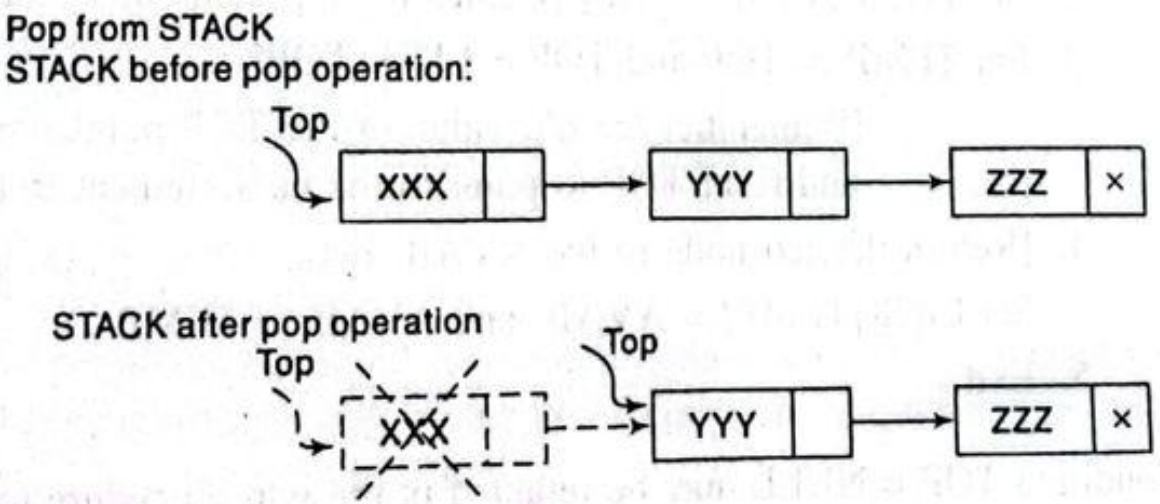


Fig. 6.8



LINKEDLIST REPRESENTATION OF STACKS

PUSH_LINKSTACK(INFO, LINK, TOP, AVAIL, ITEM)
This procedure pushes an ITEM into a linked stack

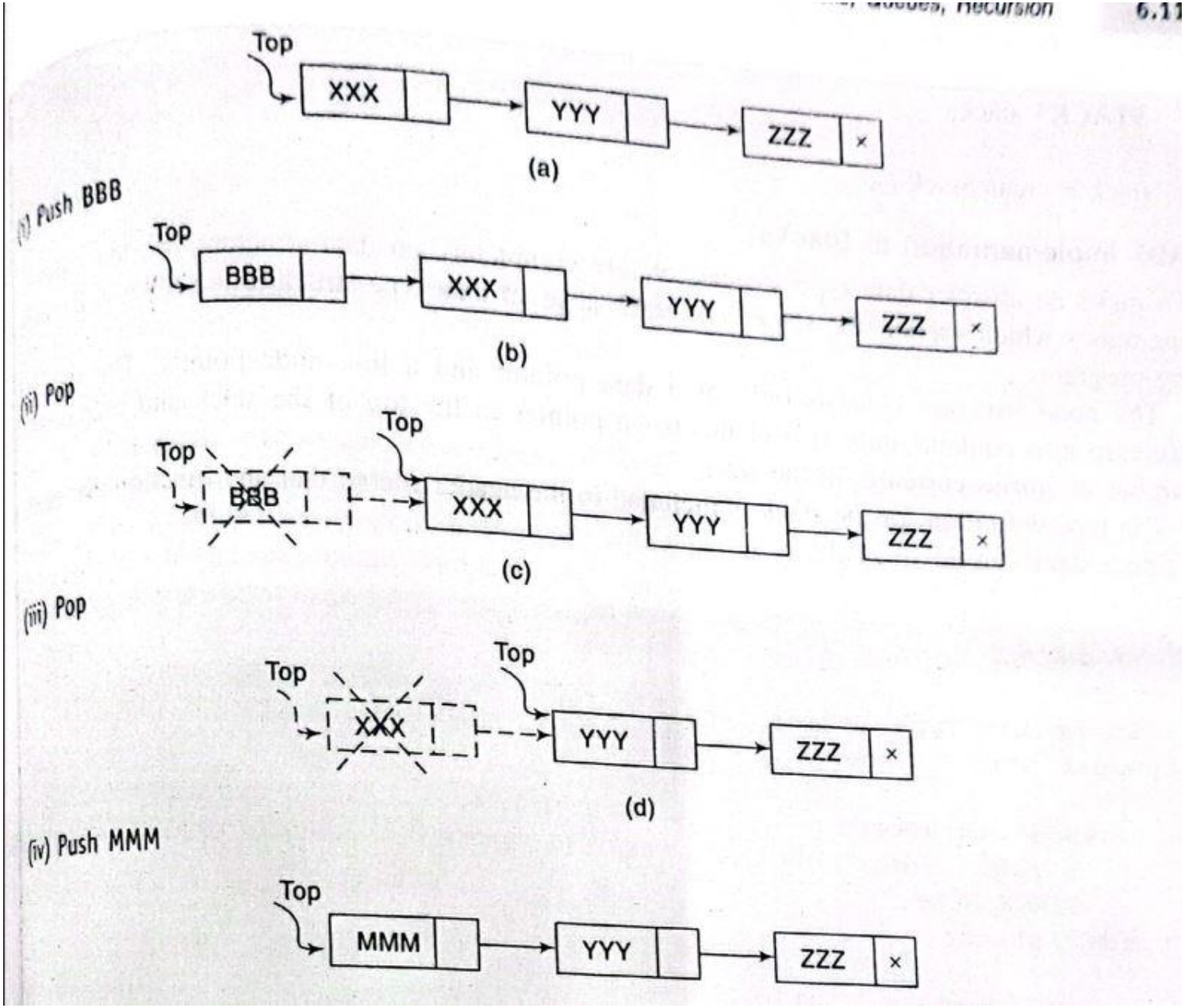
1. [Available space?] If AVAIL = NULL, then Write OVERFLOW and Exit
2. [Remove first node from AVAIL list]
   Set NEW := AVAIL and AVAIL := LINK[AVAIL].
3. Set INFO[NEW] := ITEM [Copies ITEM into new node]
4. Set LINK[NEW] := TOP [New node points to the original top node in the stack]
5. Set TOP = NEW [Reset TOP to point to the new node at the top of the stack]
6. Exit.

LINKEDLIST REPRESENTATION OF STACKS

POP_LINKSTACK(INFO, LINK, TOP, AVAIL, ITEM)
This procedure deletes the top element of a linked stack and assigns it to the variable ITEM

1. [Stack has an item to be removed?]
   IF TOP = NULL then Write: UNDERFLOW and Exit.
2. Set ITEM := INFO[TOP] [Copies the top element of stack into ITEM]
3. Set TEMP := TOP and TOP = LINK[TOP]
   [Remember the old value of the TOP pointer in TEMP
   and reset TOP to point to the next element in the stack]
4. [Return deleted node to the AVAIL list]
   Set LINK[TEMP] = AVAIL and AVAIL = TEMP.
5. Exit.

LINKEDLIST REPRESENTATION OF STACKS



Application of Stack

( )
(( )) ( ) (( ) ( )) ( )
(( )) ( ) ((( )
(( )))) ((( )

Which of the strings of parentheses are balanced?

Application of Stack

Algorithm for matching parentheses string

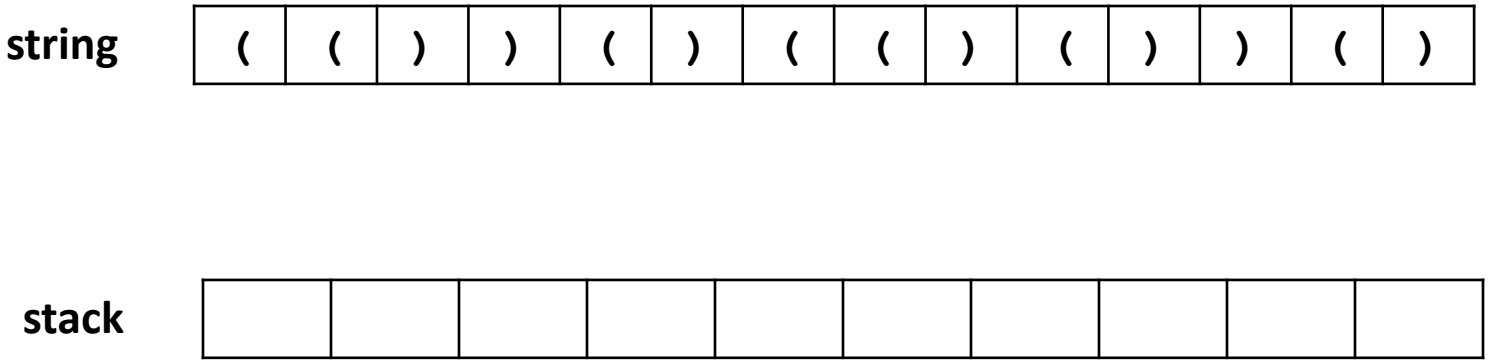
1. Initialise an empty stack
2. Read next item in the string
   a) If item is an opening parentheses, push it into the stack
   b) Else, if item is a closing parentheses, pop from stack
3. If there are more items to process, go to step 2
4. Pop the answer off the stack.

Application of Stack

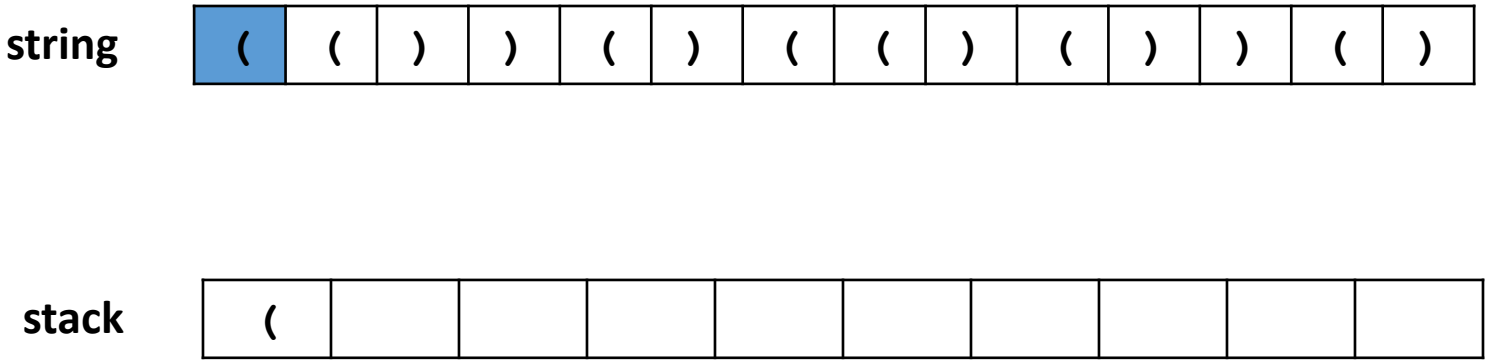
(( )) ( ) (( ) ( )) ( )

Application of Stack

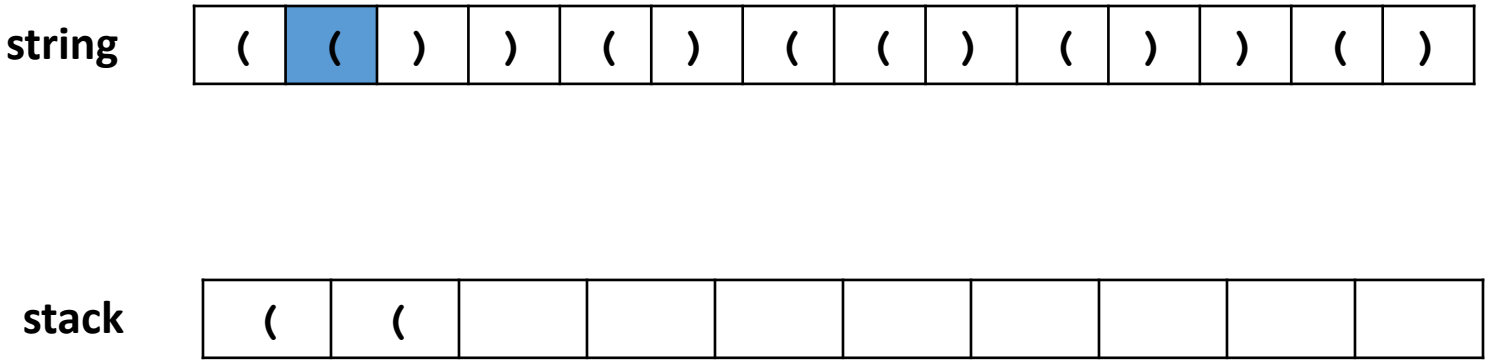
(( )) ( ) (( ) ( )) ( )



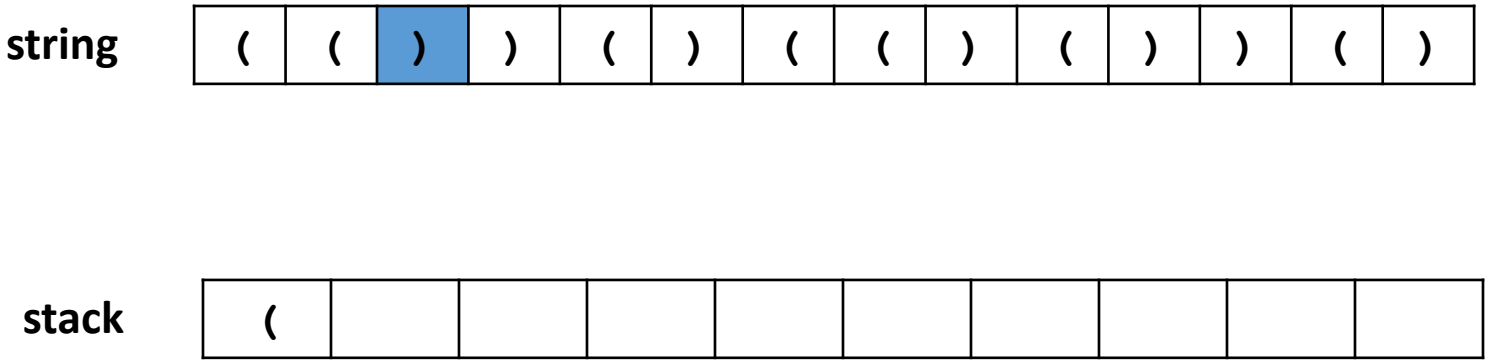
Application of Stack



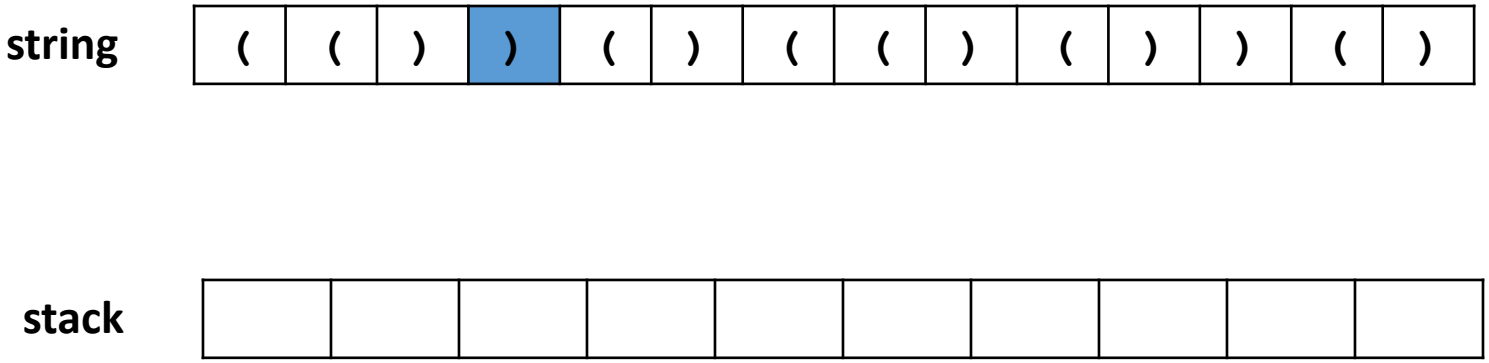
Application of Stack



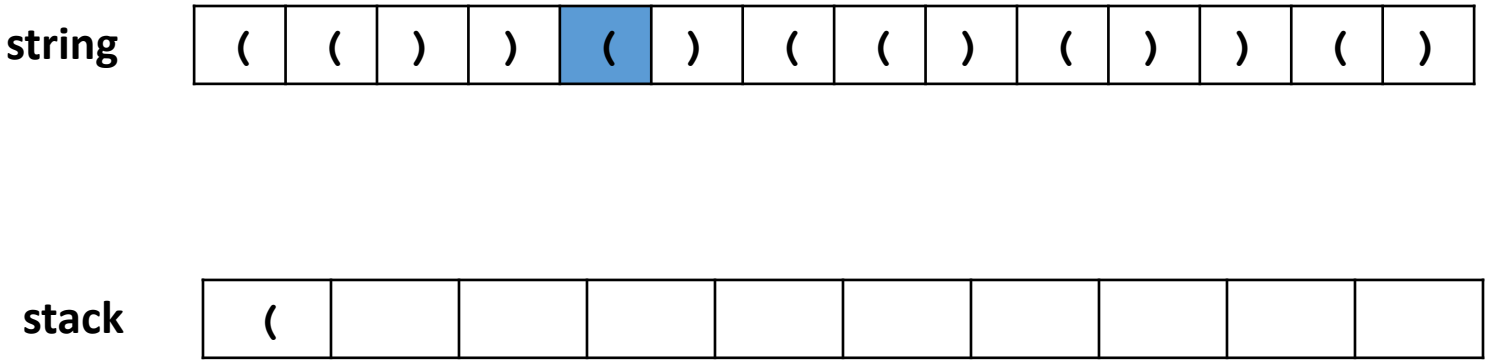
Application of Stack



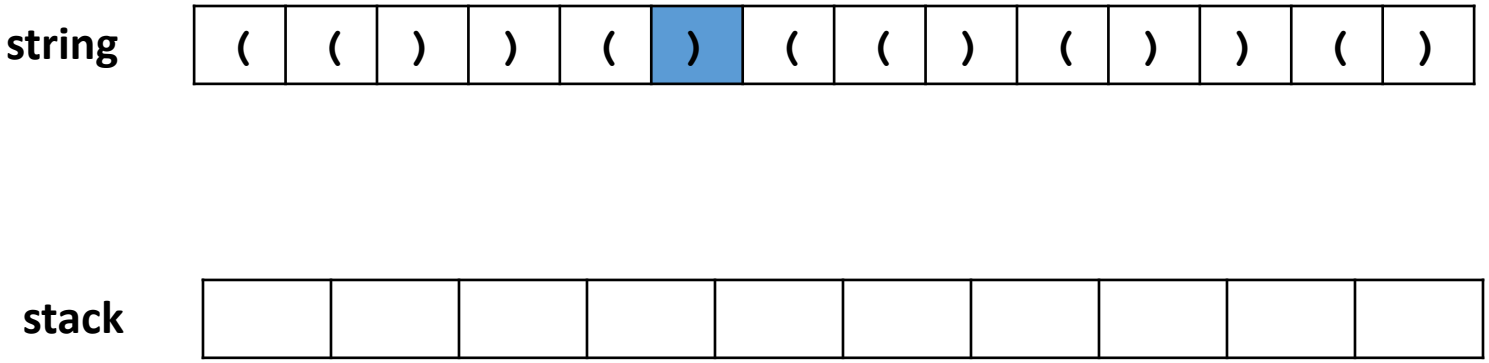
Application of Stack



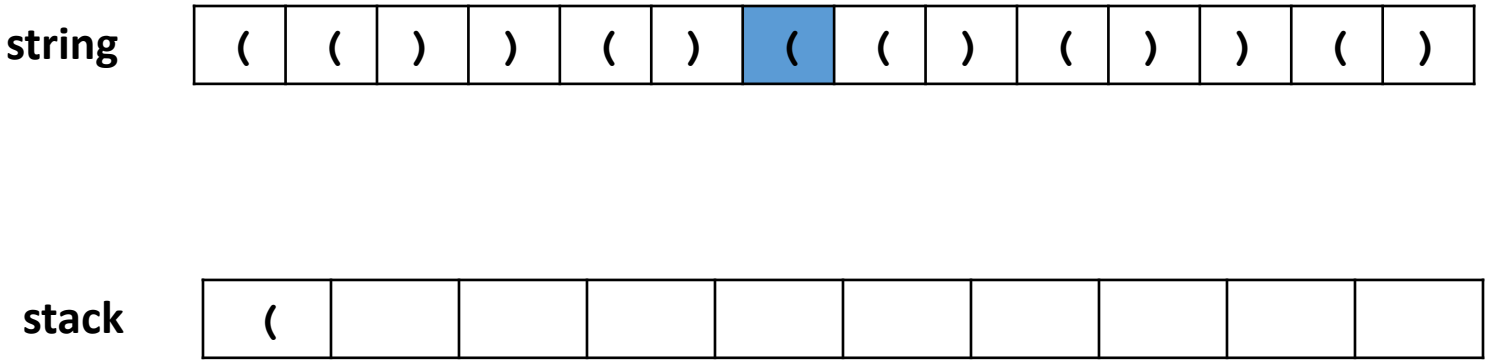
Application of Stack



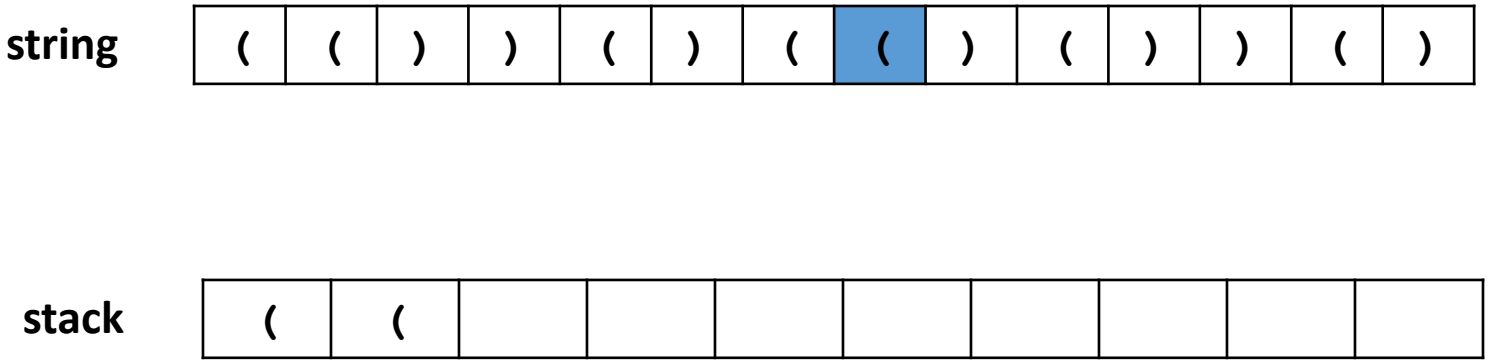
Application of Stack



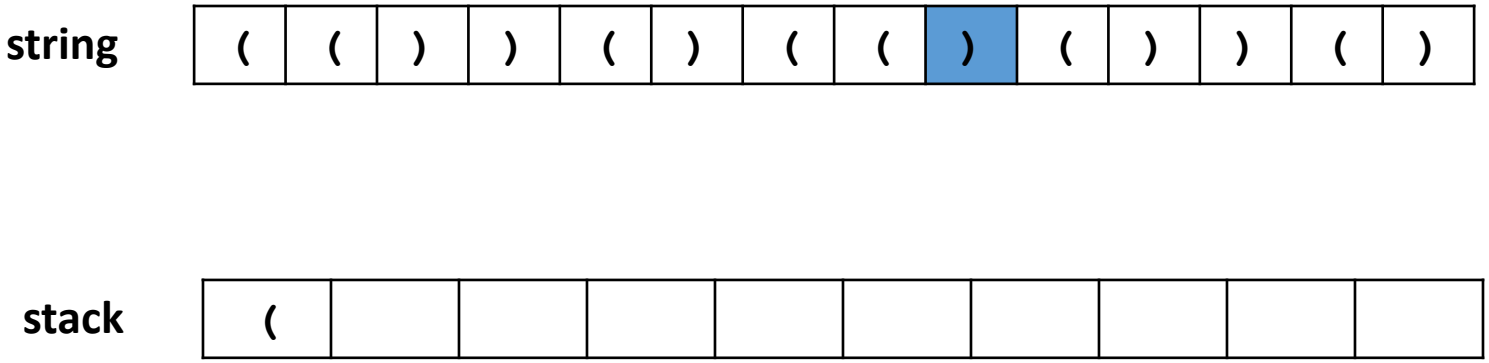
Application of Stack



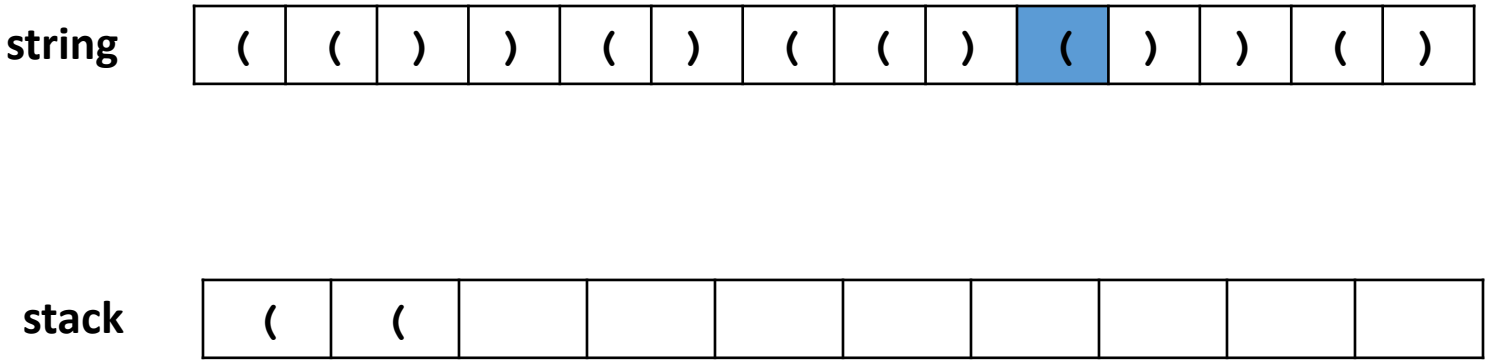
Application of Stack



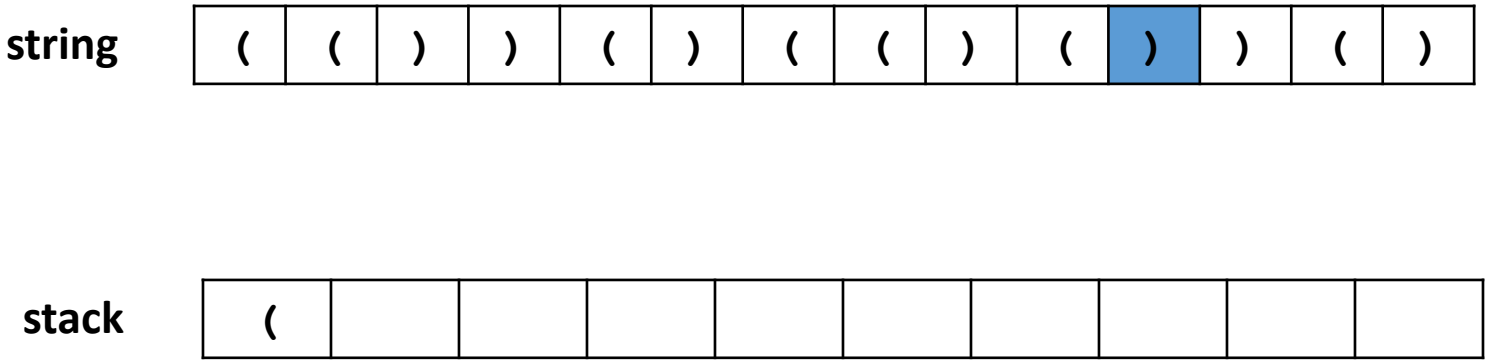
Application of Stack



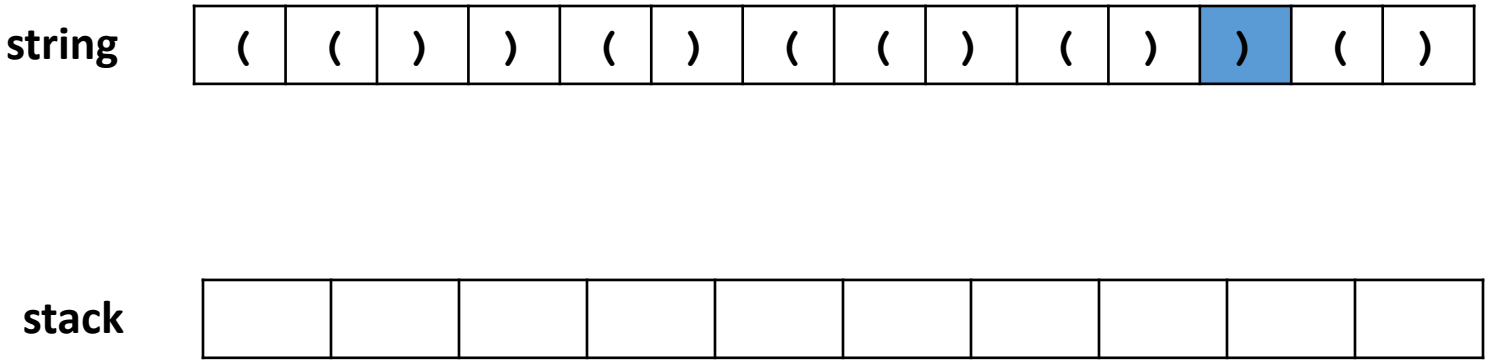
Application of Stack



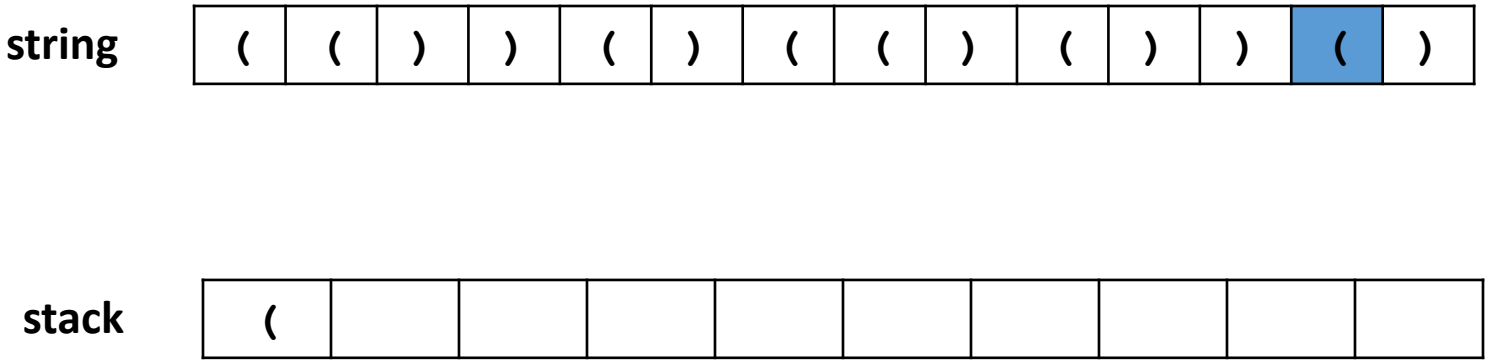
Application of Stack



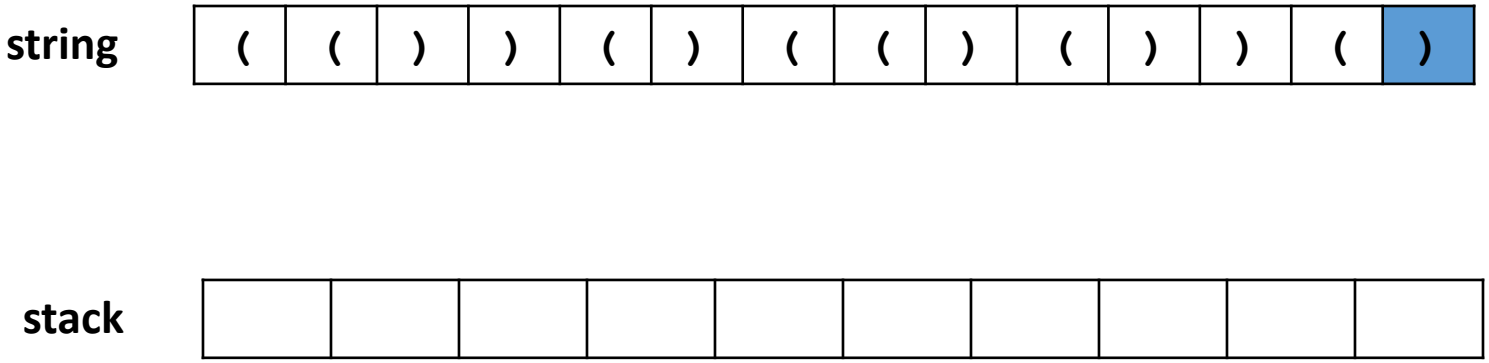
Application of Stack



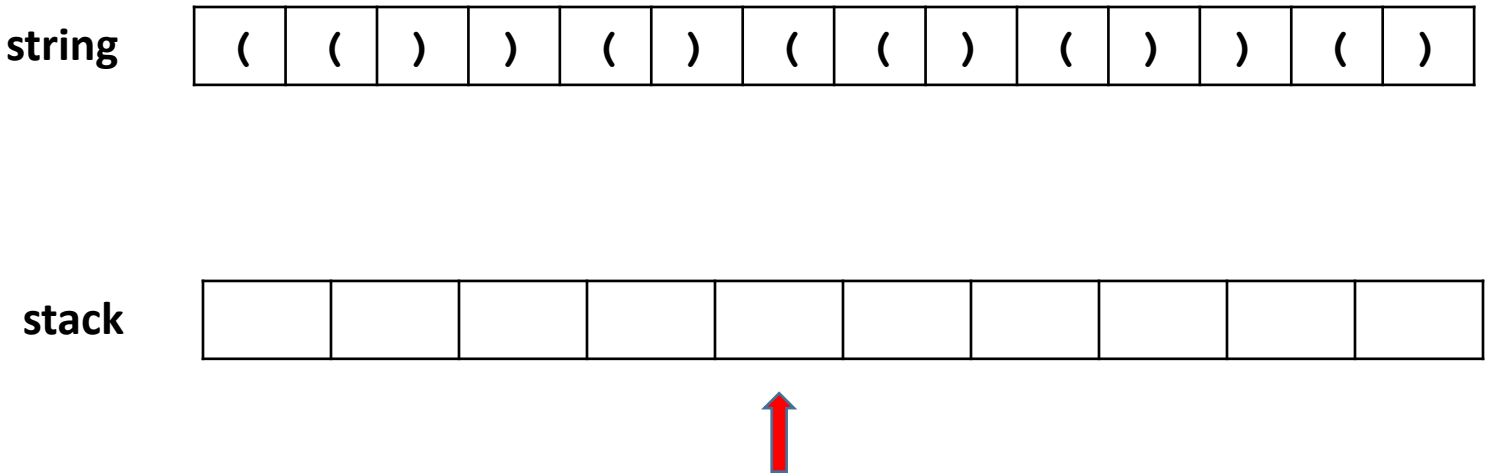
Application of Stack



Application of Stack



Application of Stack



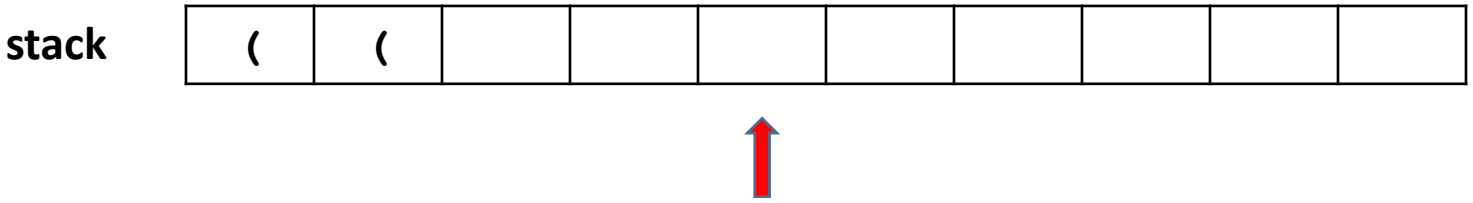
empty stack

Indicates balanced string of parentheses

Application of Stack

```
(())()(((()
```

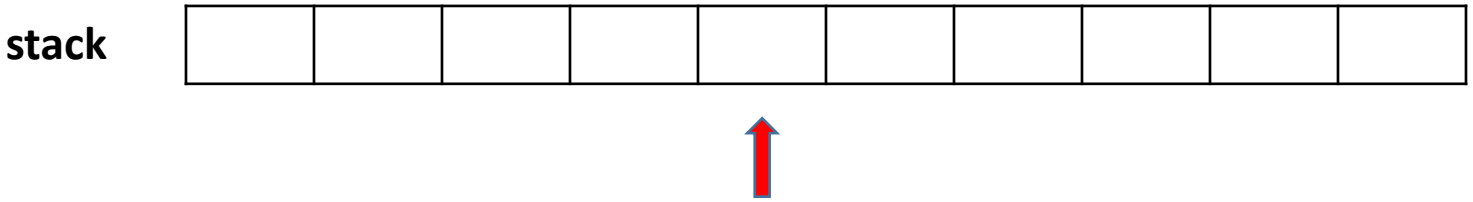
Consider this string. After processing each item, the stack is not empty.



Application of Stack

```
(()))))(((()
```

Consider this string. When processing indicated item, you are trying to pop from empty stack.



unsuccessful pop

Indicates unbalanced string of parentheses

Application of Stack

Evaluating arithmatic expressions

Infix	Postfix	Evaluation
2 - 3 * 4 + 5	2 3 4 * - 5 +	-5
(2 - 3) * (4 + 5)	2 3 - 4 5 + *	-9
2- (3 * 4 +5)	2 3 4 * 5 + -	-15

Why ? No parentheses necessary !

Application of Stack

Use binary tree to convert the following expression

`(4+8)*(6-5)/((3-2)*(2+2))` into Postfix form

`4 8 + 6 5 - * 3 2 - 2 2 + * /`

Application of Stack

Use binary tree to convert the following expression

`(4+8)*(6-5)/((3-2)*(2+2))` into Postfix form

`48+ * 65- /(32- * 22+)`

Application of Stack

Use binary tree to convert the following expression

`(4+8)*(6-5)/((3-2)*(2+2))` into Postfix form

`48+ * 65- /(32- * 22+)`

`48+ * 65- /32-22+*`

Application of Stack

Use binary tree to convert the following expression `(4+8)*(6-5)/((3-2)*(2+2))` into Postfix form

`48+ * 65- /(32- * 22+)`

`48+ * 65- /32-22+*`

`48+65-* / 32-22+*`

Application of Stack

Use binary tree to convert the following expression `(4+8)*(6-5)/((3-2)*(2+2))` into Postfix form

`48+ * 65- /(32- * 22+)`

`48+ * 65- /32-22+*`

`48+65-* / 32-22+*`

`48+65-*32-22+*/`

Application of Stack

Now let’s evaluate this expression.

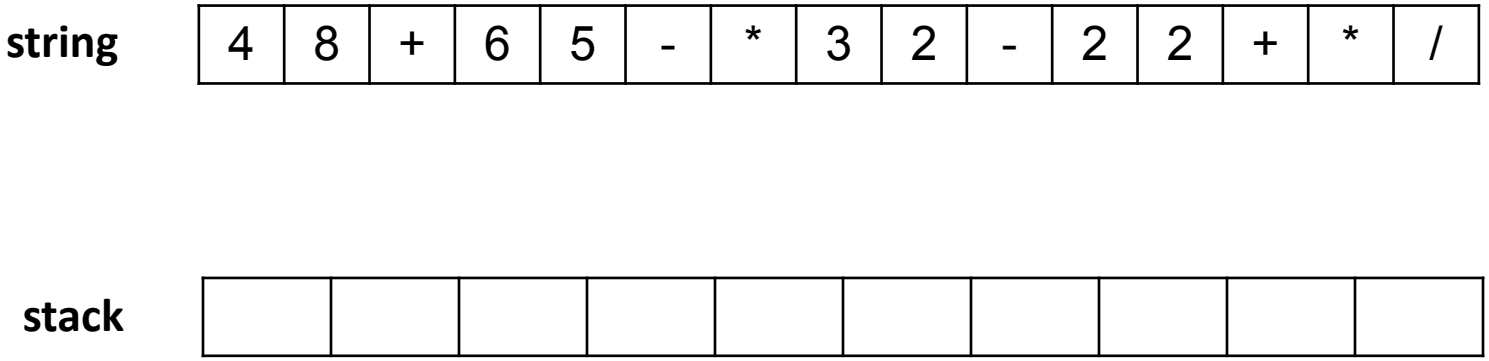
4 8 + 6 5 - * 3 2 – 2 2 + * /

Application of Stack

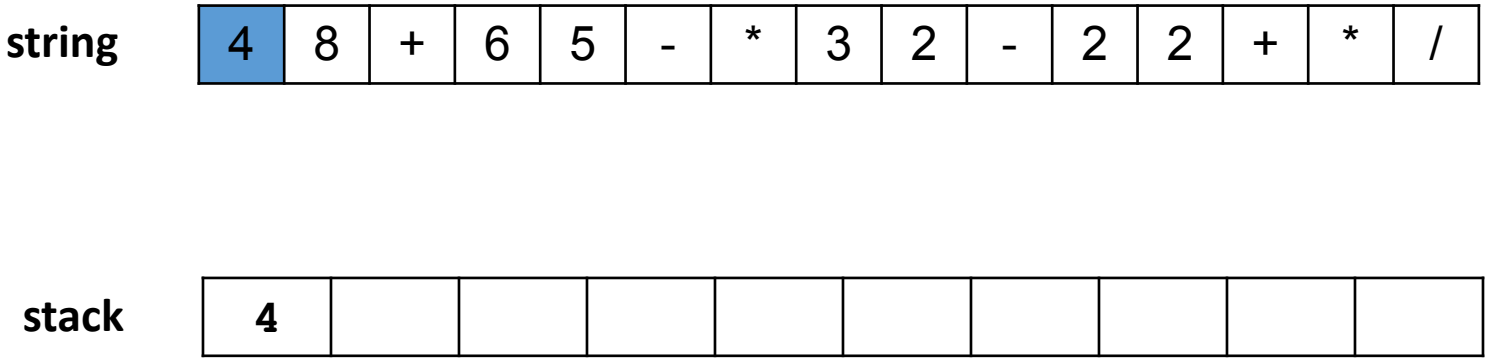
Algorithm for evaluating a postfix expression

1. Initialise an empty stack
2. Read next item in the expression
   a) If item is an operand, push it into the stack
   b) Else, if item is an operator, pop top two items off the stack, apply the operator, and push the answer back into the stack
3. If there are more items to process, go to step 2
4. Pop the answer off the stack.

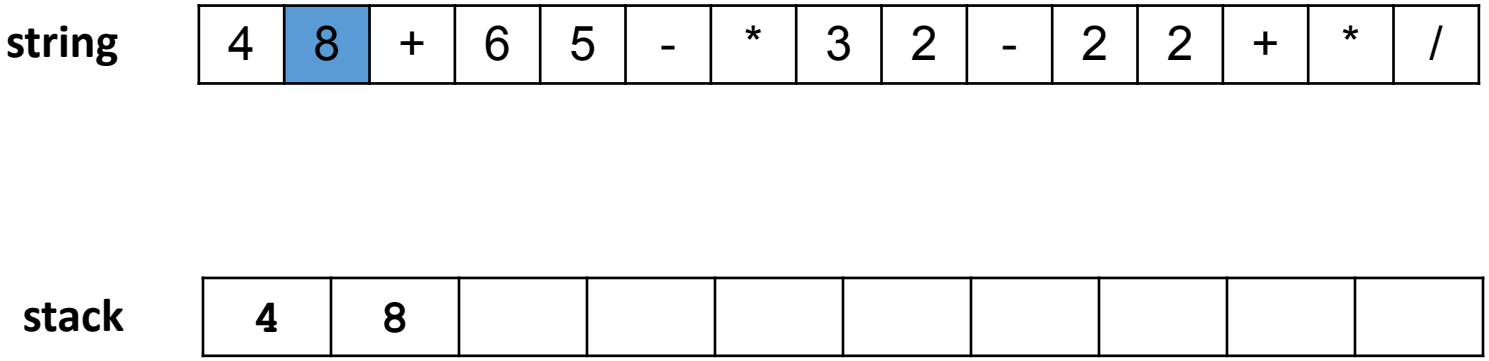
Application of Stack



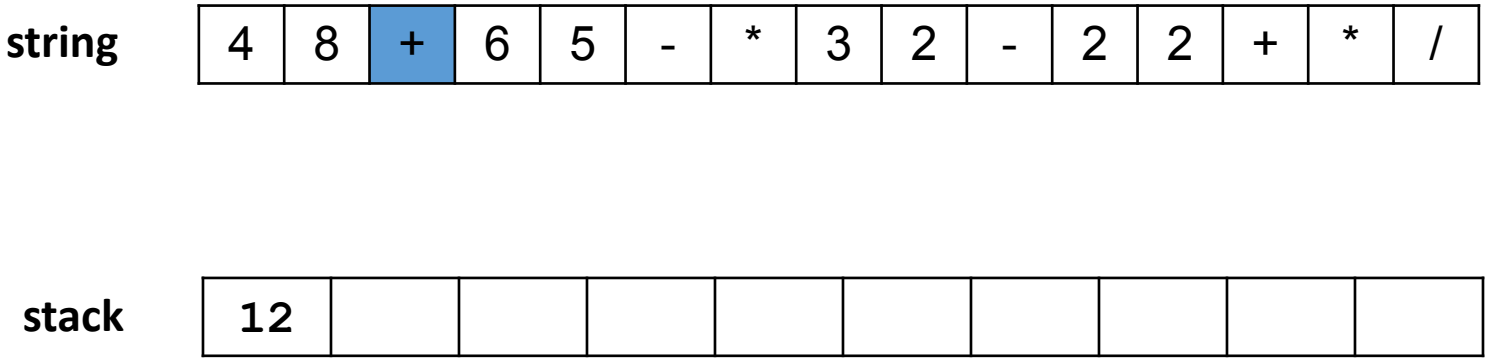
Application of Stack



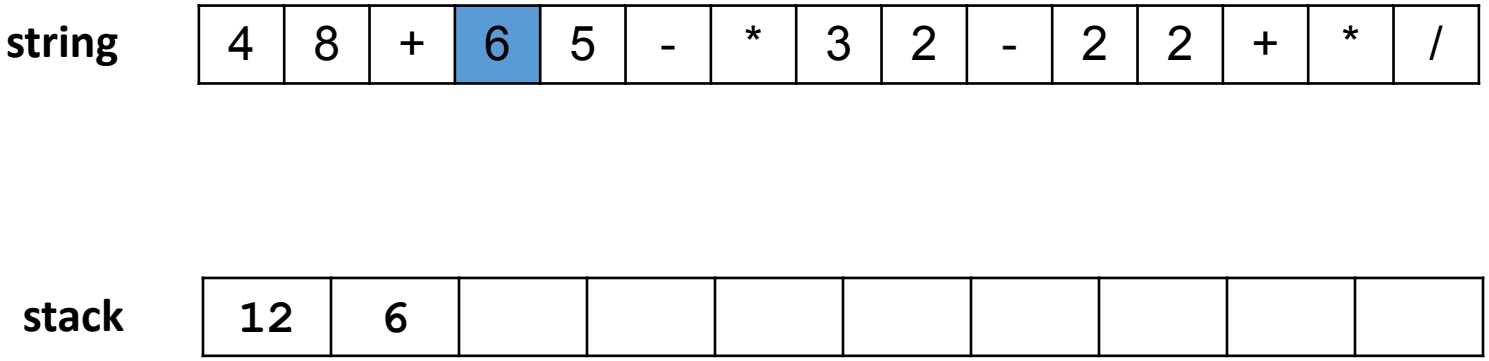
Application of Stack



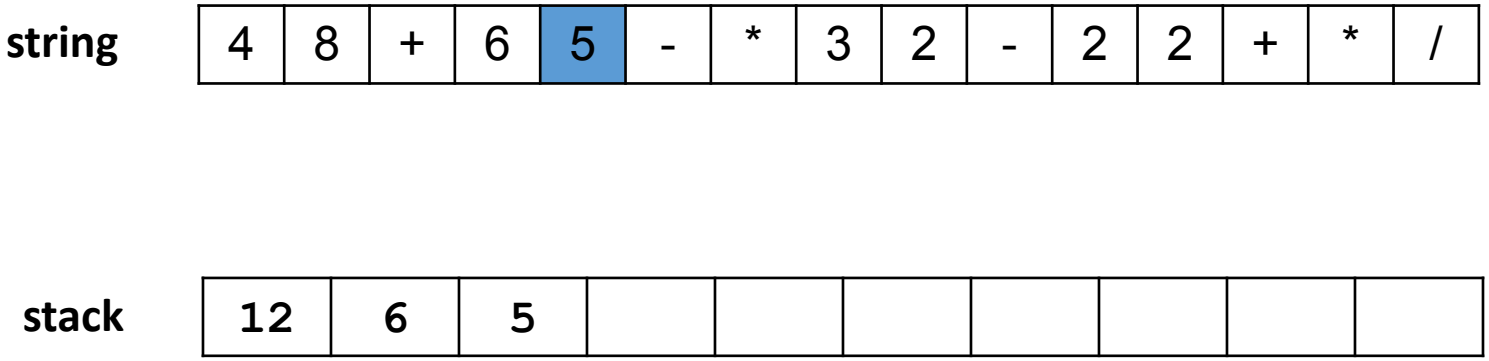
Application of Stack



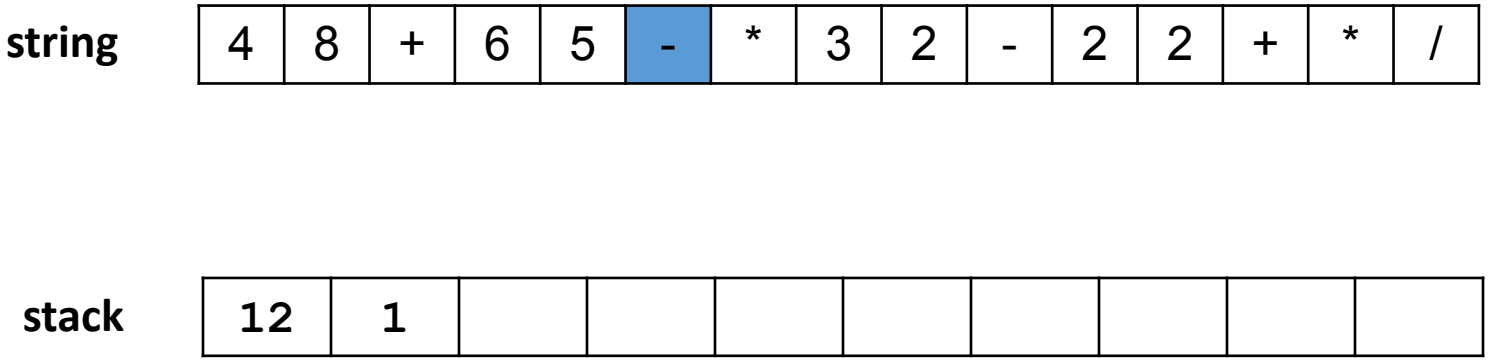
Application of Stack



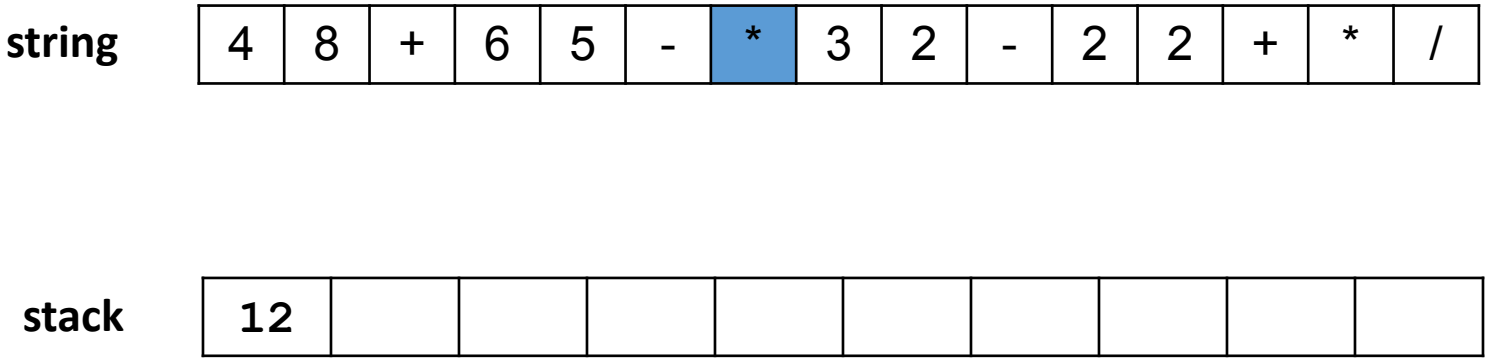
Application of Stack



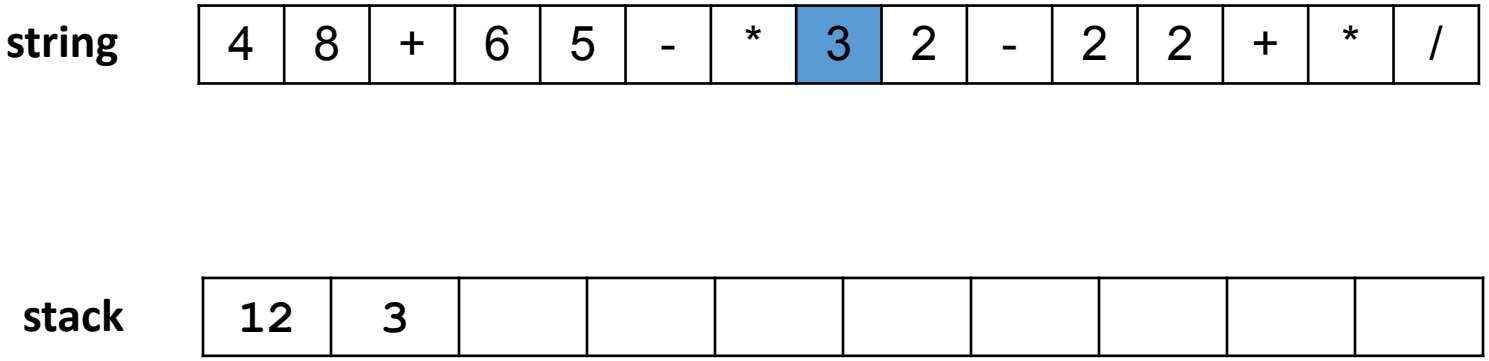
Application of Stack



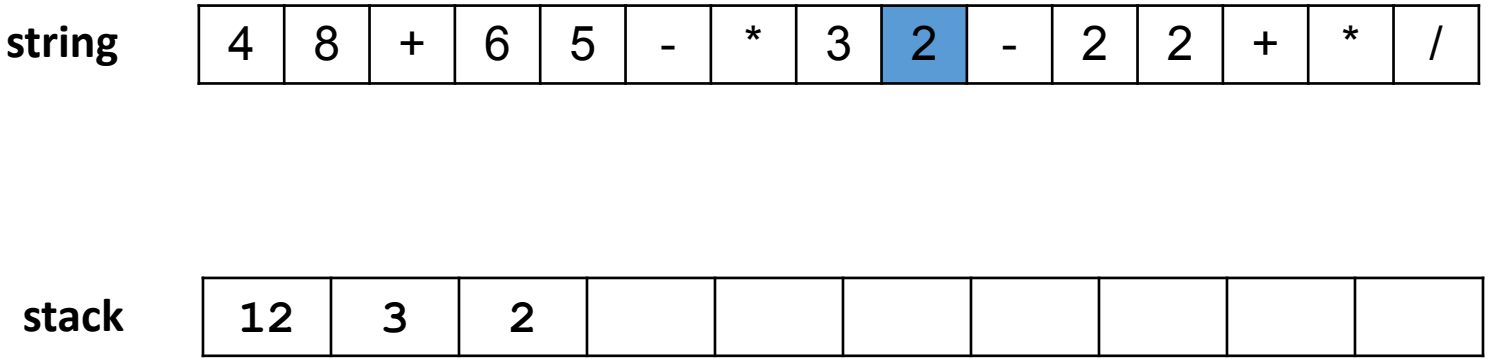
Application of Stack



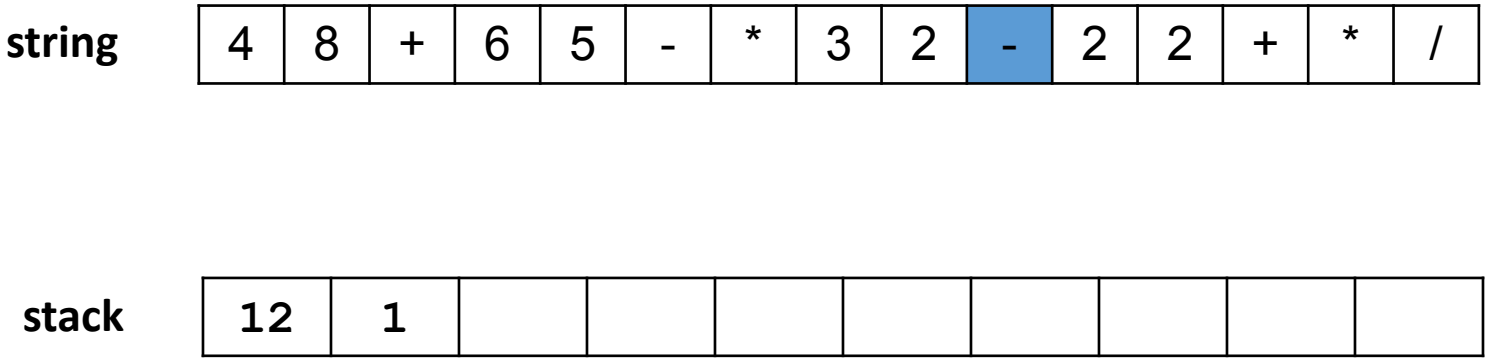
Application of Stack



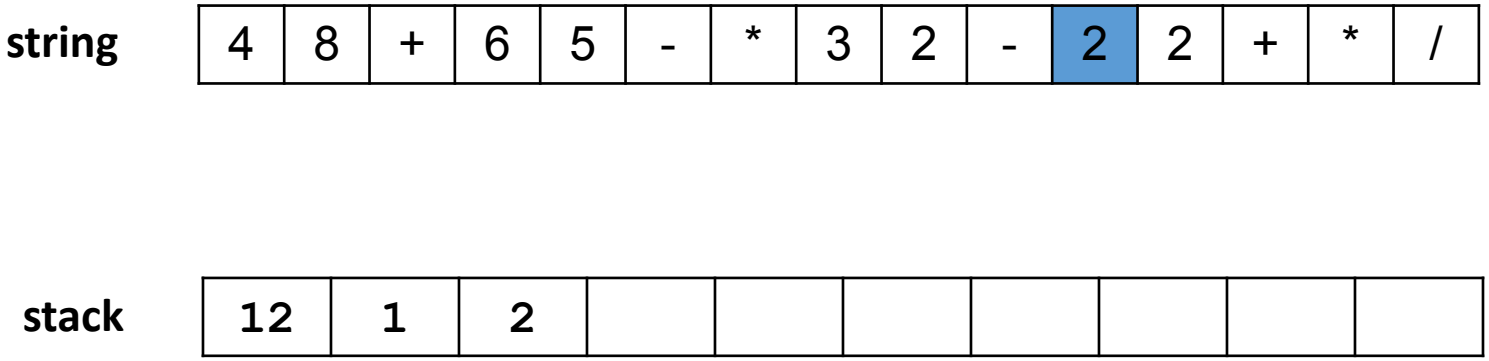
Application of Stack



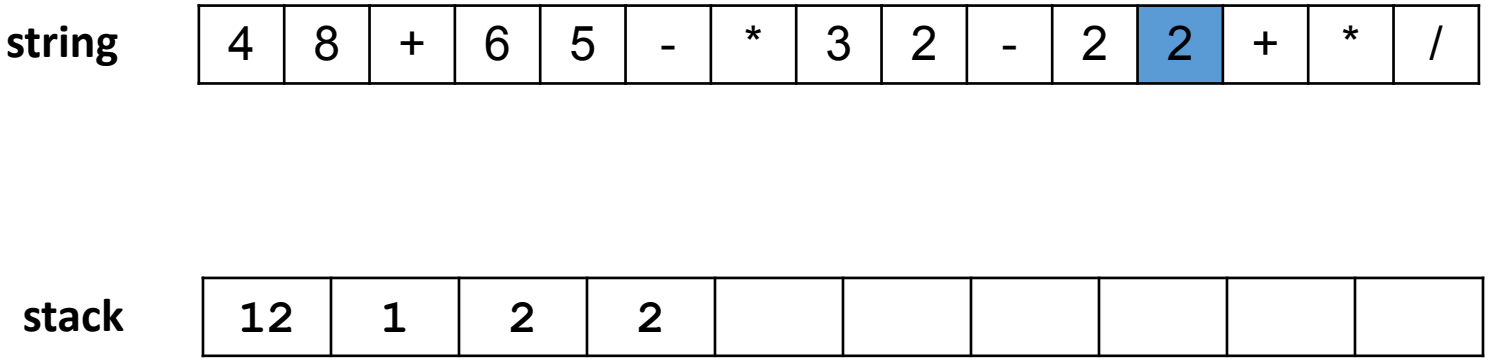
Application of Stack



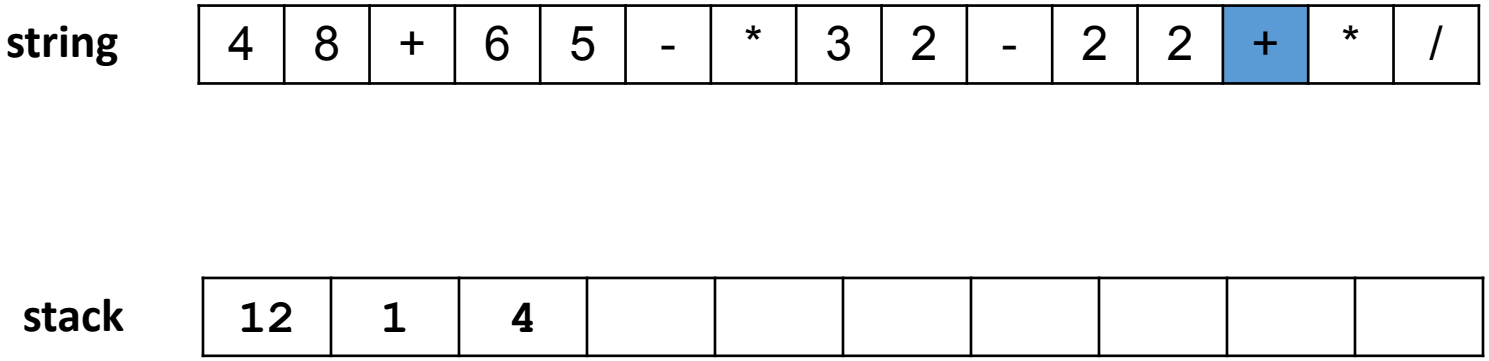
Application of Stack



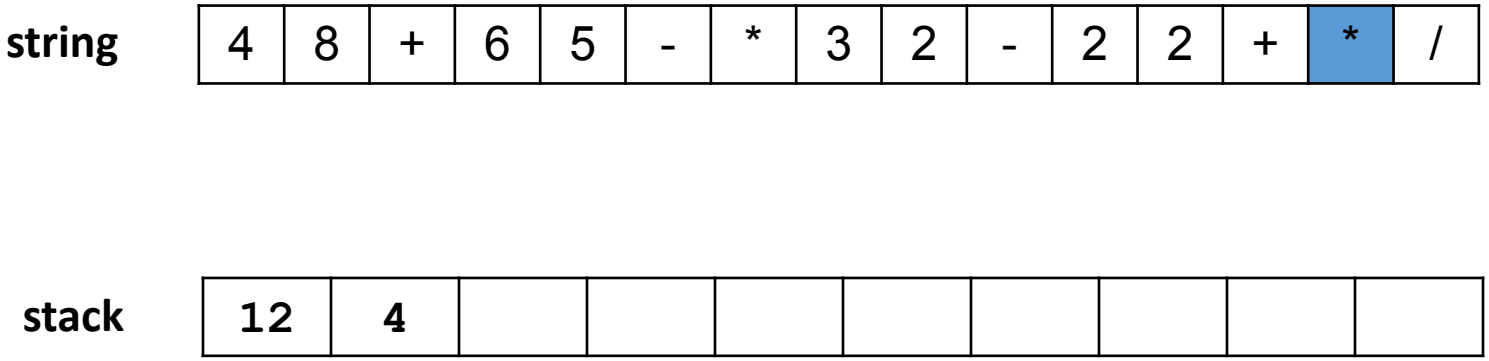
Application of Stack



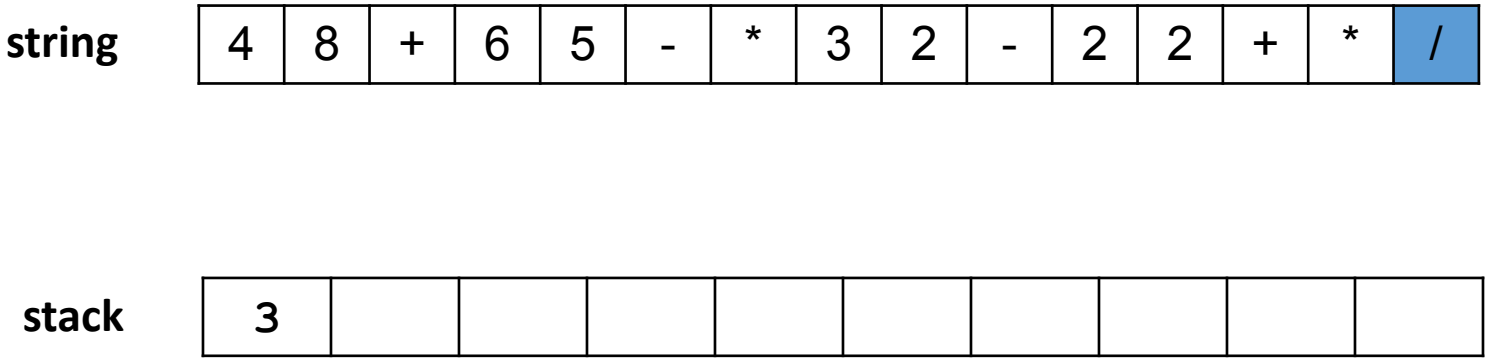
Application of Stack



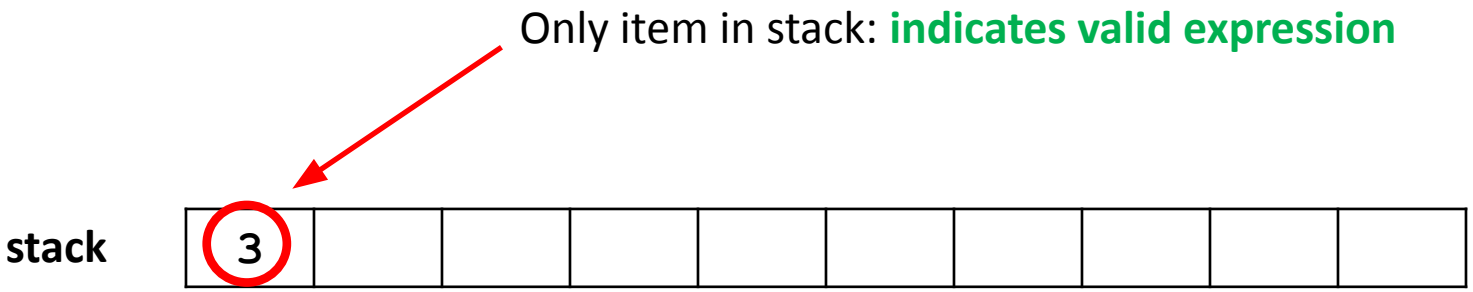
Application of Stack



Application of Stack



Application of Stack



More than one item in stack or unsuccessful pop:
indicates invalid expression