Stack Data Structure

LECTURE 9

Objectives

- At the end of this lecture the learner is able to:
  - Explain the purpose the structure of Stack
  - Understand the main operations on stack
  - Build and implement stack operations

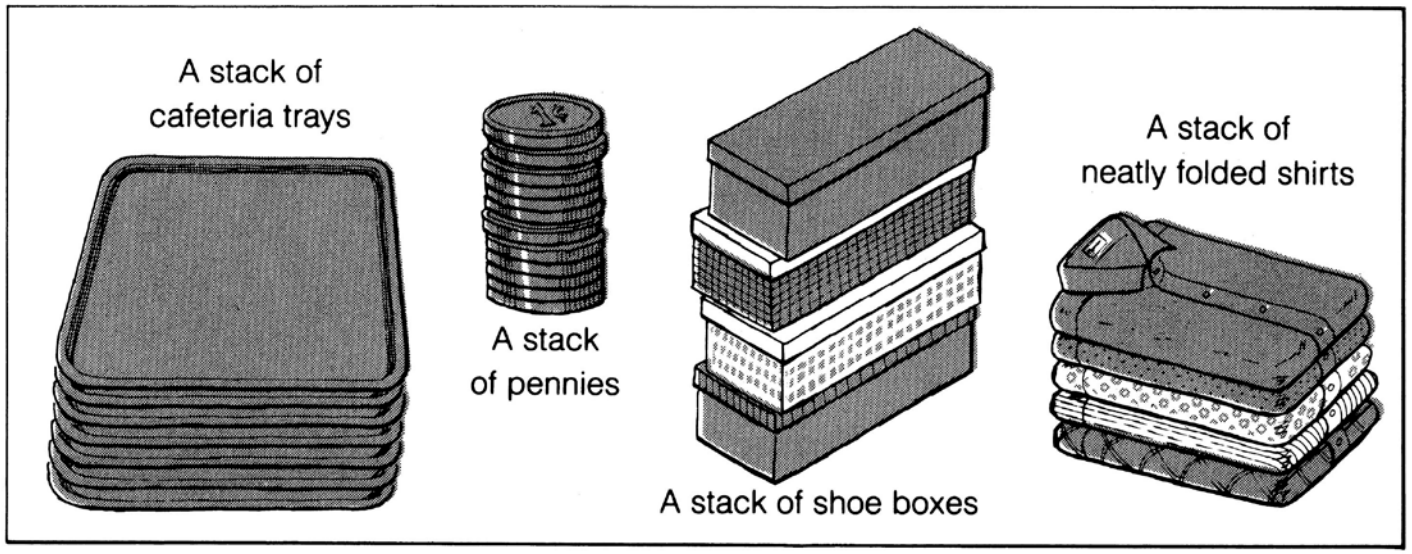
Outlines

- Definition and Basic operation
- Stack Analogy
- Stack Application
- Implementation of Stack
- Stack implementation linked list

Definition and Basic operation

- Stack is a linear data structure which follows a particular order in which the operations are performed.
- The order may be LIFO(Last In First Out) or FILO(First In Last Out).

Stack Analogy



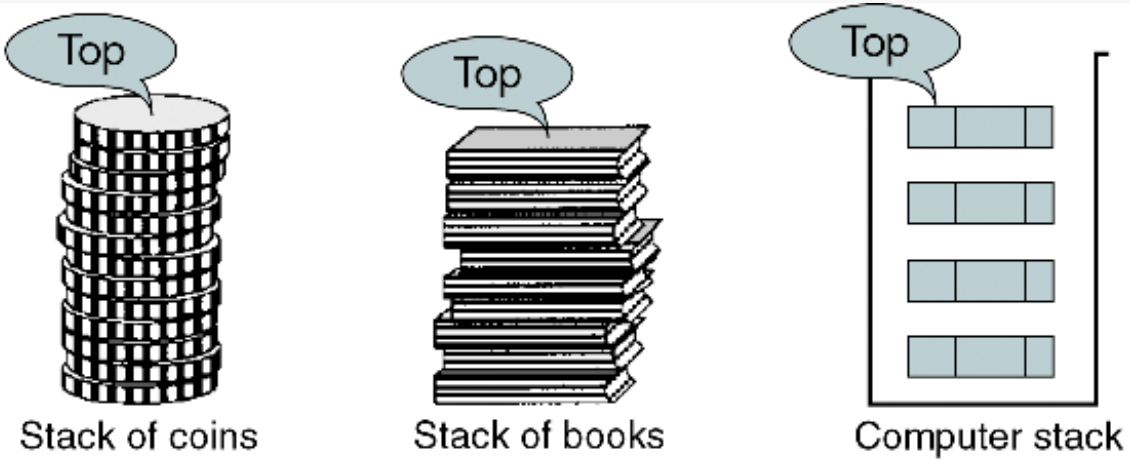


FIGURE 3-1 Stack

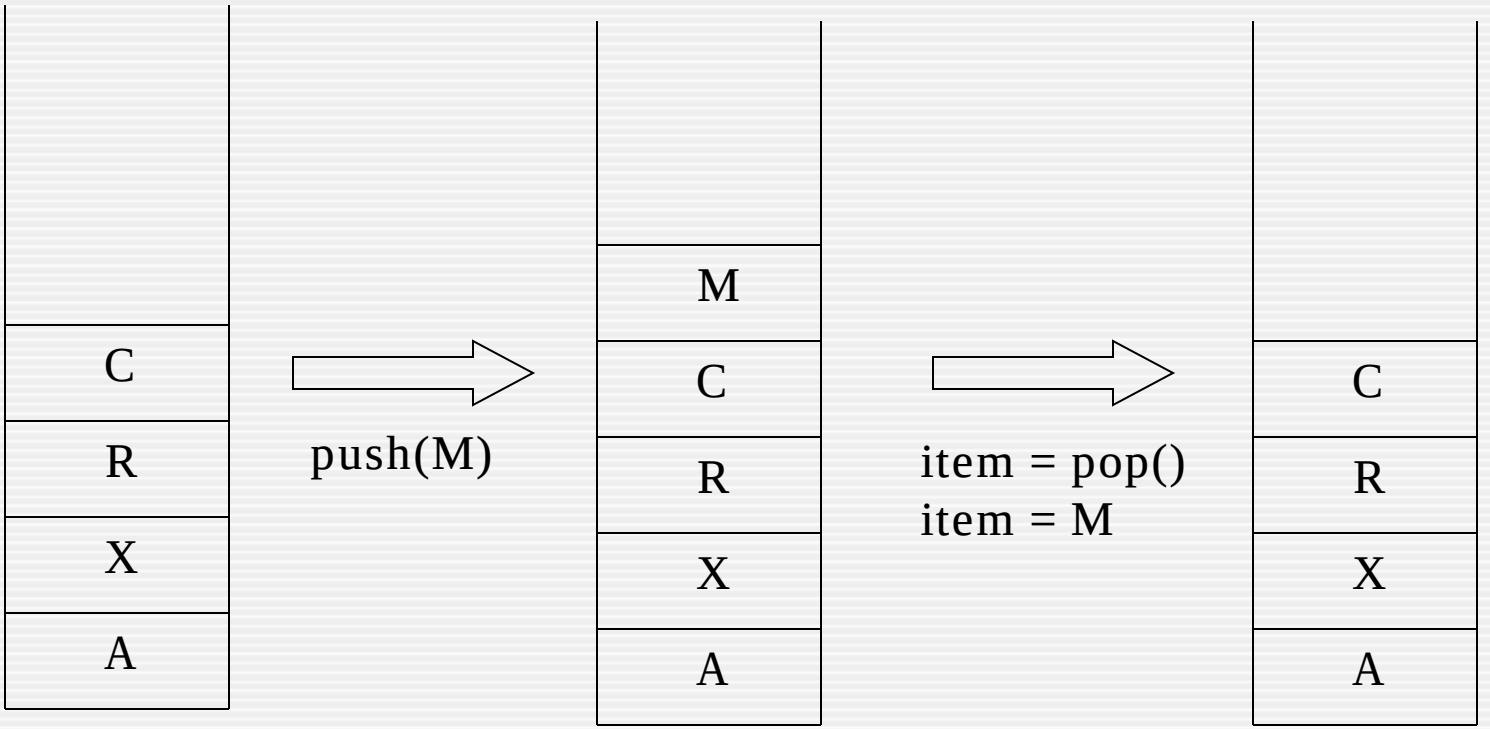
Stack Application

- Operating systems
- Control Programs
- Automation Programs

Definition and Basic operation

- **Push:** Adds an item in the stack.
  - If the stack is full, then it is said to be an Overflow condition.
- **Pop:** Removes an item from the stack. The items are popped in the reversed order in which they are pushed.
  - If the stack is empty, then it is said to be an Underflow condition.
- **isEmpty:** Returns true if stack is empty, else fals.

Stack



Implementation of Stack

- There are two ways to implement a stack:
  - Using linked list
  - Using array

Stack implementation using linked list

```
1  // Fig. 12.8: fig12_08.c
2  // A simple stack program
3  #include <stdio.h>
4  #include <stdlib.h>
5
6  // self-referential structure
7  struct stackNode {
8     int data; // define data as an int
9     struct stackNode *nextPtr; // stackNode pointer
10 }; // end structure stackNode
11
12 typedef struct stackNode StackNode; // synonym for struct stackNode
13 typedef StackNode *StackNodePtr; // synonym for StackNode*
14
15 // prototypes
16 void push( StackNodePtr *topPtr, int info );
17 int pop( StackNodePtr *topPtr );
18 int isEmpty( StackNodePtr topPtr );
19 void printStack( StackNodePtr currentPtr );
20 void instructions( void );
```

Stack implementation using linked list

```
22  // function main begins program execution
23  int main( void )
24  {
25     StackNodePtr stackPtr = NULL; // points to stack top
26     unsigned int choice; // user's menu choice
27     int value; // int input by user
28
29     instructions(); // display the menu
30     printf( "%s", "? " );
31     scanf( "%u", &choice );
32
33     // while user does not enter 3
34     while ( choice != 3 ) {
35
36        switch ( choice ) {
37           // push value onto stack
38           case 1:
39              printf( "%s", "Enter an integer: " );
40              scanf( "%d", &value );
41              push( &stackPtr, value );
42              printStack( stackPtr );
43              break;
44           // pop value off stack
45           case 2:
46              // if stack is not empty
47              if ( !isEmpty( stackPtr ) ) {
```

Stack implementation using linked list

```
48                printf( "The popped value is %d.\n", pop( &stackPtr ) );
49             } // end if
50
51             printStack( stackPtr );
52             break;
53          default:
54             puts( "Invalid choice.\n" );
55             instructions();
56             break;
57       } // end switch
58
59       printf( "%s", "? " );
60       scanf( "%u", &choice );
61    } // end while
62
63    puts( "End of run." );
64 } // end main
```

Stack implementation using linked list

```
66  // display program instructions to user
67  void instructions( void )
68  {
69     puts( "Enter choice:\n"
70        "1 to push a value on the stack\n"
71        "2 to pop a value off the stack\n"
72        "3 to end program" );
73  } // end function instructions
74
75  // insert a node at the stack top
76  void push( StackNodePtr *topPtr, int info )
77  {
78     StackNodePtr newPtr; // pointer to new node
79
80     newPtr = malloc( sizeof( StackNode ) );
81
82     // insert the node at stack top
83     if ( newPtr != NULL ) {
84        newPtr->data = info;
85        newPtr->nextPtr = *topPtr;
86        *topPtr = newPtr;
87     } // end if
88     else { // no space available
89        printf( "%d not inserted. No memory available.\n", info );
90     } // end else
91  } // end function push
92
93  // remove a node from the stack top
94  int pop( StackNodePtr *topPtr )
95  {
96     StackNodePtr tempPtr; // temporary node pointer
97     int popValue; // node value
98
99     tempPtr = *topPtr;
100    popValue = ( *topPtr )->data;
```

Stack implementation using linked list

```
101     *topPtr = ( *topPtr )->nextPtr;
102     free( tempPtr );
103     return popValue;
104  } // end function pop
105
106  // print the stack
107  void printStack( StackNodePtr currentPtr )
108  {
109     // if stack is empty
110     if ( currentPtr == NULL ) {
111        puts( "The stack is empty.\n" );
112     } // end if
113     else {
114        puts( "The stack is:" );
115
116        // while not the end of the stack
117        while ( currentPtr != NULL ) {
118           printf( "%d --> ", currentPtr->data );
119           currentPtr = currentPtr->nextPtr;
120        } // end while
121
122        puts( "NULL\n" );
123     } // end else
124  } // end function printList
125
126  // return 1 if the stack is empty, 0 otherwise
127  int isEmpty( StackNodePtr topPtr )
128  {
129     return topPtr == NULL;
130  } // end function isEmpty
```

Stack implementation using linked list

```
Enter choice:
1 to push a value on the stack
2 to pop a value off the stack
3 to end program
? 1
Enter an integer: 5
The stack is:
5 --> NULL

? 1
Enter an integer: 6
The stack is:
6 --> 5 --> NULL

? 1
Enter an integer: 4
The stack is:
4 --> 6 --> 5 --> NULL
```

End