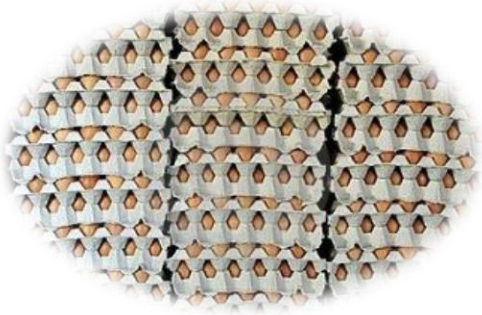
CS214-Data Structure

Lecturer: Dr. Salwa Osama

Stack

Stack – What?



Let’s Refresh our Mind

Solve

1. A stack is(Nonlinear- Linear)
2. Stack is (ordered - unordered) List of Elements of (Same- different Type).
3. In Stack, all Operations are permitted at(only one variable - Multiple variables).
4. When an element is removed from a stack using the standard removal operation, which item is accessed and removed?

   A. The most recently added element
   B. The least recently added element
   C. The element at the middle of the stack
   D. The element at a random position in the stack

5. Stack is (first in first out – first in last out – last in last out – last in first out)

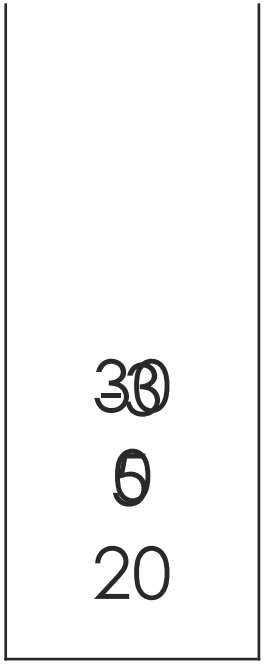
Stack – What?

- A stack is **Linear - non-primitive** data structure.
- Stack is Ordered List of Elements of **Same Type.**
- In Stack, all Operations are permitted at only one end called **Top**.

So, the stack is called Last-in-First-out (LIFO)

Stack – What?

- Add (20)
- Add(5)
- Add(30)
- Delete
- Delete
- Add(0)
- Add(-3)

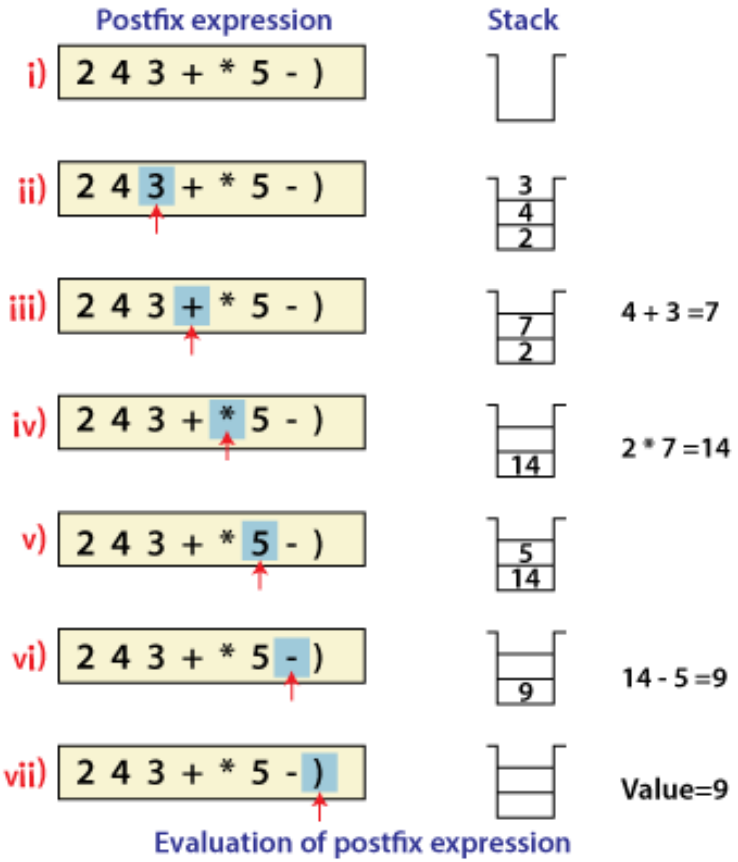


Stack Animation by Y. Daniel Liang (pearsoncmg.com)

STACK APPLICATIONS

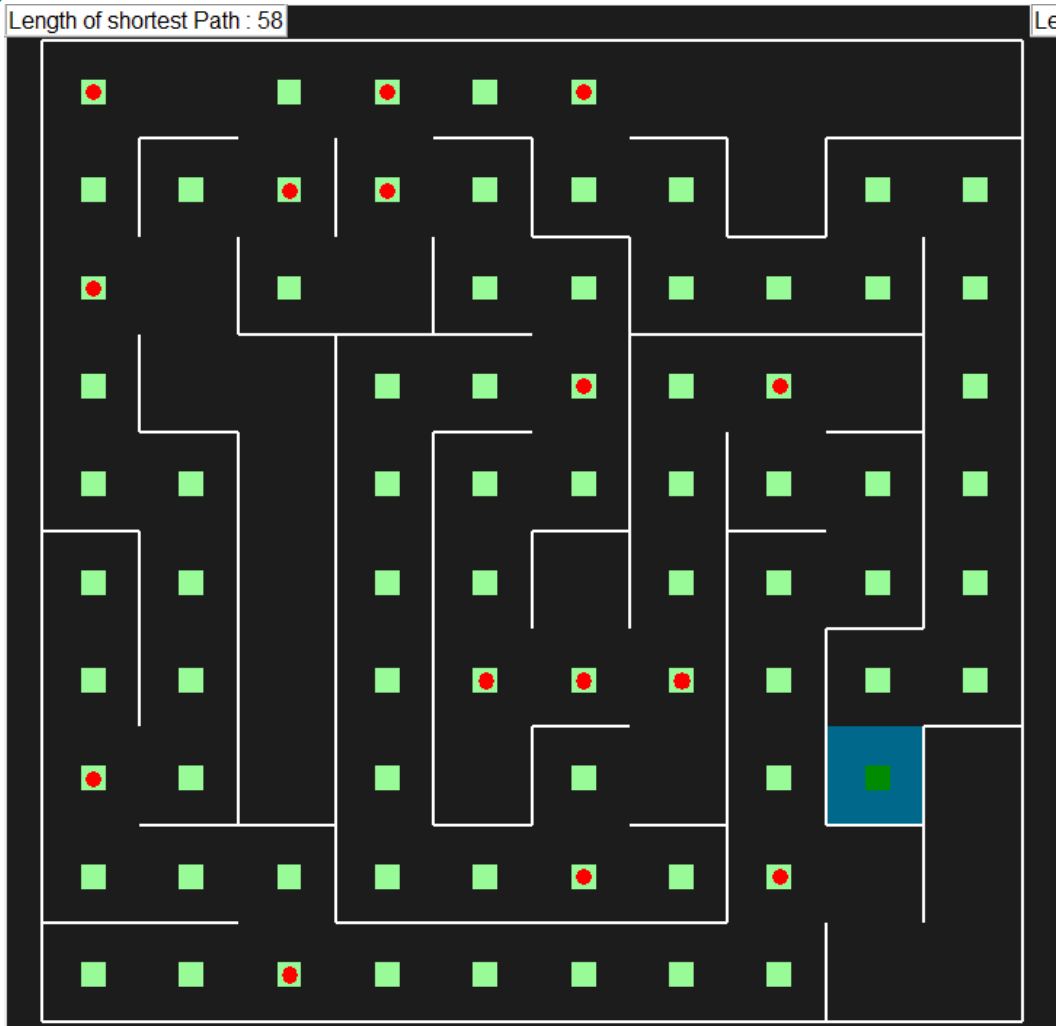
Stack Application
Evaluation of Arithmetic Expression

2 * (4+3) - 5	Stack	Postfix Expression
2 * (4+3) – 5		2
2 * (4+3) – 5	*	2
2 * (4+3) – 5	(*	2
2 * (4+3) – 5	(*	24
2 * (4+3) – 5	+ (*	24
2 * (4+3) – 5	+ (*	243
2 * (4+3) - 5		243+*
2 * (4+3) - 5	-	243+*
2 * (4+3) - 5	-	243+*5
2 * (4+3) - 5		243+*5-



Stack Application
Backtracking

Demo
Maze.py



Stack Application Processing Function Calls

Q(1)
Q(2)
~~N()~~
M()

```
M(){
...
N();
...
O();
...
...
...
...
}
N(){
...
}
```

```
O(){
...
i=2
Q(i);
...
}
Q(int i){
...
If (i>1)
   Q(i-1);
   ...
}
```

Demo CallStack

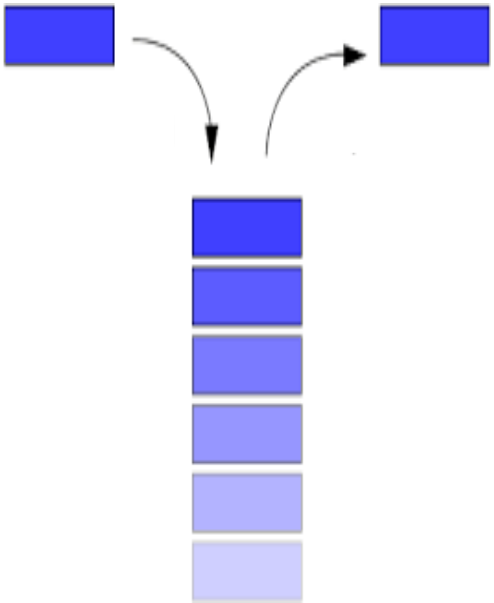
Stack Application Delimiter Checking

Valid Delimiter	Invalid Delimiter
While (i > 0)	While (i >
/* Data Structure */	/* Data Structure
{ (a + b) - c }	{ (a + b) - c

Input left	Characters Read	Stack Contents
{{(a-b) * (c-d)}/f]	[	[
{(a-b) * (c-d)}/f]	{	[{
(a-b) * (c-d)}/f]	(	[{ (
a-b) * (c-d)}/f]	a	[{ (
-b) * (c-d)}/f]	-	[{ (
b) * (c-d)}/f]	b	[{ (
) * (c-d)}/f]	)	[{
* (c-d)}/f]	*	[{
(c-d)}/f]	(	[{ (
c-d)}/f]	g	[{ (
-d)}/f]	-	[{ (
d)}/f]	d	[{ (
)}/f]	)	[{
}/f]	}	[
/f]	/	[
f]	f	[
]		

Stack

Implementation



User View

Operations Performed On Stack

- **Create** the stack, leaving it empty.
- Determine whether the stack **is empty** or not.
- Determine whether the stack **is full** or not.
- **Push** a new entry onto the top of the stack
- **Pop** the entry off the top of the stack.

Stack Contiguous Implementation

- Each stack item is adjacent in memory to the next stack item, and so stack items are kept in an array.
- The top position is kept in an integer field.
- The top and the data array are grouped in a struct.

```
#define MAX 10
typedef char EntryType;
typedef   struct{
    int          top;
     char        entry[  10  ];
} StackType;
```

Stack Contiguous Implementation

- **Initialization:**

  Pre: None.

  Post: The stack is initialized to be empty.

```
CreateStack(s){
    s->top = -1;
}
```

Stack Contiguous Implementation

Stack empty operation:

Pre: The stack is initialized.

Post: If the stack is empty (1) is returned. Otherwise (0) is returned.

```
StackEmpty(s){
return (s.top==-1);
}
```

Stack Contiguous Implementation

Stack full operation:

Pre: The stack is initialized.

Post: If the stack is full (1) is returned. Otherwise (0) is returned.

```
int    StackFull(StackType      s){
       return (s.top==MAX-1);
}
```

Stack Contiguous Implementation

- **Push operation:**

  Pre: The stack is initialized and is not full.

  Post: Item is added to the top of the stack.

```
Push(                                    ){
    s->entry[++s->top] = item;
}
```

```
S->top++;
S->entry[s->top]=item;
```

Stack Contiguous Implementation

- **Push operation with another specification:**

  Pre: The stack is initialized.

  Post: If the stack is not full, item is added to the top of the stack. Otherwise, an error message is displayed and the stack is left unchanged

```
void  Push( EntryType  item, StackType *s  ){
    if (s->top == MAX-1)
            printf("Error: Stack Overflow")
    else
            s->entry[++s->top] = item;
}
```

NOW: Which specification is better ???

Stack Contiguous Implementation

- **Pop operation:**

  Pre: The stack is initialized and is not empty.

  Post: The top element of the stack is removed from it and is assigned to item.

```
Pop(){
*item = s->entry[s->top--];
}
```

Stack Contiguous Implementation

- **Pop operation with another specification:**

  Pre: The stack is initialized.

  Post: If the stack is not empty The top element of the stack is removed from it and is assigned to item. Otherwise, an error message is displayed and the stack is left unchanged

```
void Pop(Entry Type*item, Stack type*s){
    if (s->top == -1)
            printf("Error: Stack underflow")
    else
            *item = s->entry[s->top--];
}
```

NOW: Which specification is better ???

Using Of The Stack

Exercise

Assume that we need to read a line of text and write it back in a reverse order.

StackUser2

Answer

```
StackType stack;
//Initialize the stack to be empty
CreateStack(&stack);

item = getchar();
while (!StackFull(stack)&& item!= '\n'){
    //Push each item onto the stack
    Push(item, &stack);
    item = getchar();
}
while (!StackEmpty(stack)){
    //Pop an item from the stack
    Pop(&item, &stack);
    putchar(item);
  }
```

Exercise

- As a
  functio
  and le

```
EntryType  StackTop(StackType * s){
    EntryType  item;
    pop(&item, s);
    push(item, s);
    return (item);
}
```

ackTop

e stack

peekUserLEvel

Exercise

- **Rewrite the previous function as a part of stack ADT**

```
EntryType   StackTop(StackType *  s){
    EntryType   item;
    item = s->entry[s->top];
    return (item);
}
```

```
Return (s->entry[s->top])
```

peekImplLevel

Question1

1	Prints binary representation of n in reverse order
2	Prints binary representation of n
3	Prints the value of Logn
4	Prints the value of Logn in reverse order

```
void fun(int n)
{
   Stack S;  // Say it creates an empty stack S
   while (n > 0)
   {
     // This line pushes the value of n%2 to stack S
     push(&S, n%2);

     n = n/2;
   }

   // Run while Stack S is not empty
   while (!isEmpty(&S))
     printf("%d ", pop(&S)); // pop an element from S and print it
}
```

What does the above function do in general?

prob1

Question2

- Which one of the following is an application of Stack Data Structure?

1. Managing function calls
2. Arithmetic expression evaluation
3. All of the above

Question3

evaluatePostfix

- The following postfix expression with single digit operands is evaluated using a stack:

8 2 3 ^ / 2 3 * + 5 1 * -

- Note that ^ is the exponentiation operator.
- The output of that expression is …:


Any
Questions