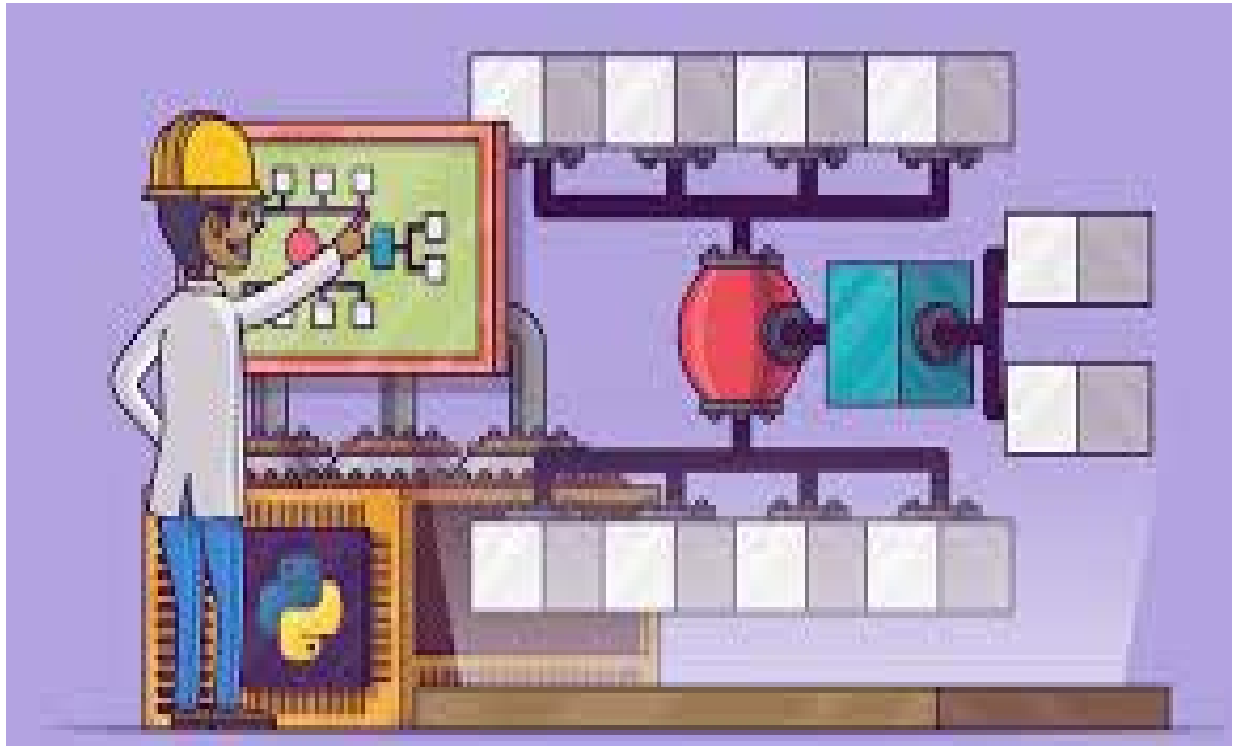
# ساختمان داده ها



مدرس:
سمانه حسینی سمنانی

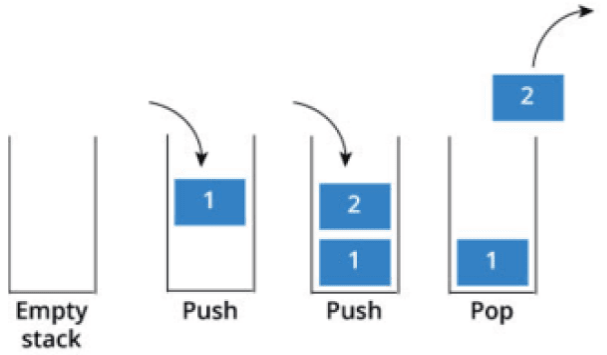
دانشگاه صنعتی اصفهان- دانشکده برق و کامپیوتر

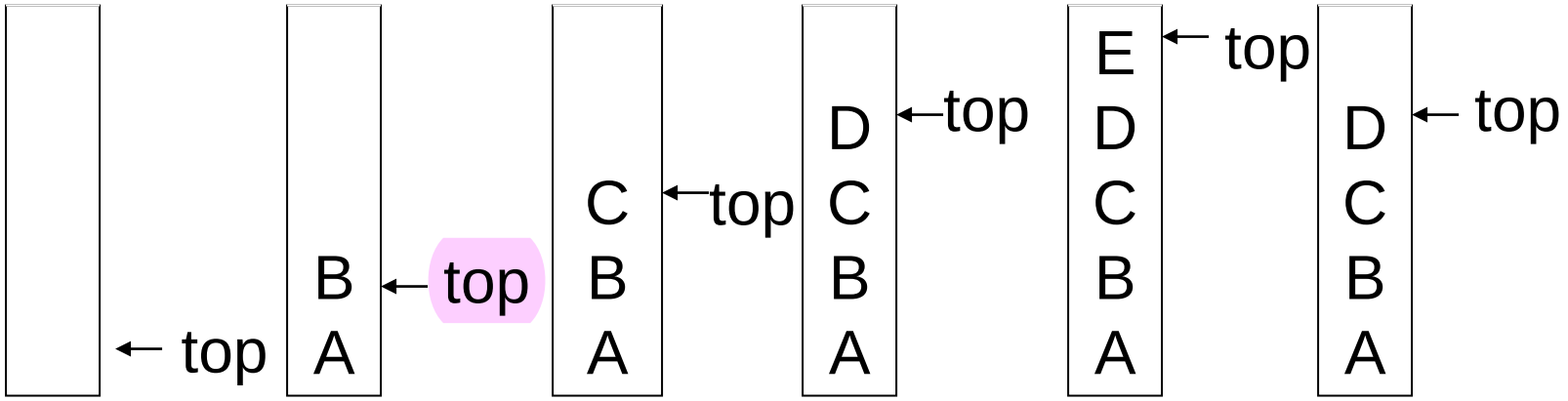
# پشته و صف

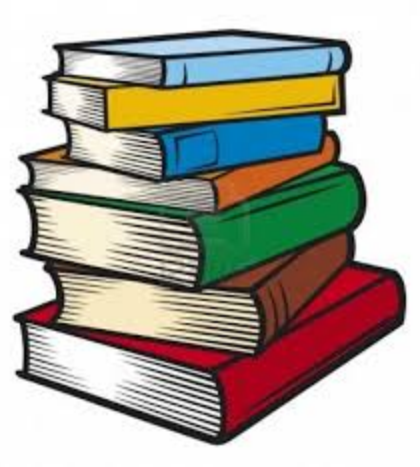
- ADT پشته
- کاربردهای پشته
  - پشته سیستم
  - پارکینگ قطارها
  - ارزیابی عبارات
- ADT صف
- کاربرد های صف

# Stacks

- **Stack**: an ordered list in which insertions and deletions are made at one end called the *top*.
- last-in, first-out: **LIFO**



# Stack ADT

```
private:
    T*stack;          // array for stack elements
    int top;          // array position of top element
    int capacity;     // capacity of stack array
template <class T>
class Stack
{ // A finite ordered list with zero or more elements.
public:
    Stack (int stackCapacity = 10);
    // Create an empty stack whose initial capacity is stackCapacity.

    bool IsEmpty() const;
    // If number of elements in the stack is 0, return true else return false.

    T& Top() const;
    // Return top element of stack.

    void Push (const T& item);
    // Insert item into the top of the stack.

    void Pop();
    // Delete the top element of the stack.
};
```

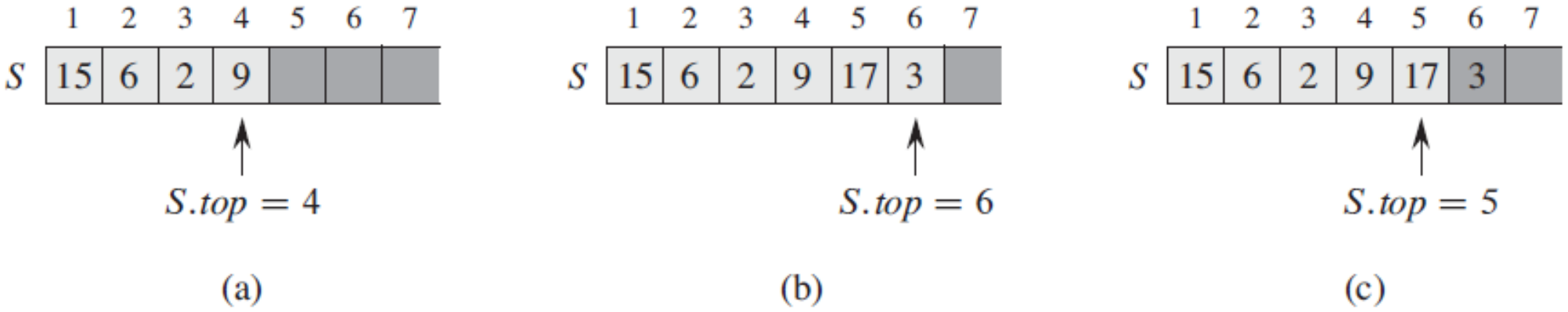
فرق Top با Pop چیست؟



سمانه حسینی سمنانی
هیات علمی دانشکده برق و کامپیوتر- دانشگاه صنعتی اصفهان

# Stack ADT

- we can implement a stack of at most n elements with an array $S[1..n]$



# Stack ADT Constructure

```
private:
      T*stack;          // array for stack elements
      int top;          // array position of top element
      int capacity;     // capacity of stack array
template <class T>
Stack<T>::Stack (int stackCapacity) : capacity (stackCapacity)
{
      if (capacity < 1) throw "stack capacity must be > 0";
      stack = new T[capacity];
      top = -1;
}
```

Capacity=stackCapacity

# Stack ADT, Top, Push functions

```
template <class T>
inline T& Stack <T> :: Top ( ) const
{
        if (IsEmpty ( )) throw "Stack is empty";
        return stack [top];
}
```

```
template <class T>
void Stack<T>::Push (const T& x)
{ // Add x to the stack.
        if (top = = capacity – 1)
        {
                ChangeSize1D(stack, capacity, 2*capacity );
                capacity *= 2;
        }
        stack[++top] = x;
}
```

# Stack ADT, Pop functions

```
template <class T>
void Stack<T>::Pop()
{ // Delete top element from the stack.
        if (IsEmpty()) throw "Stack is empty. Cannot delete.";
        stack [top--].~T();       //destructor for T
}
```

# Stack applications

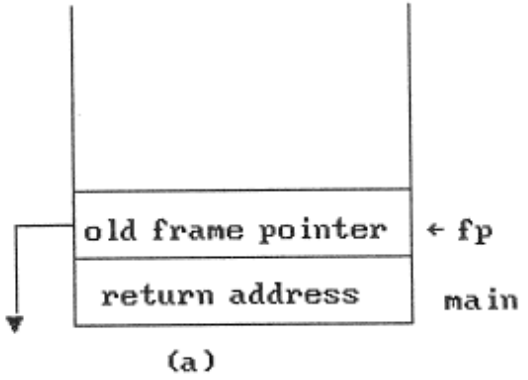


سمانه حسینی سمنانی
هیات علمی دانشکده برق و کامپیوتر- دانشگاه صنعتی اصفهان

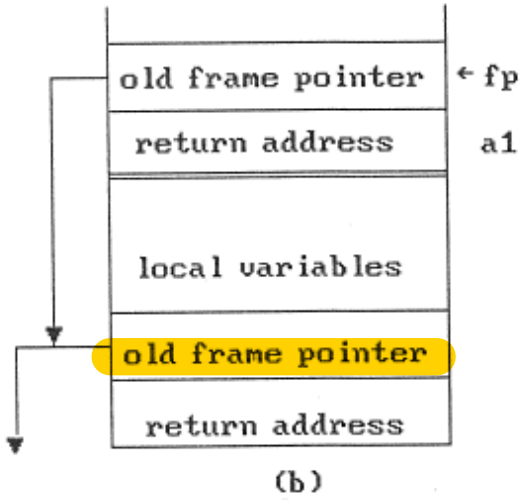
# Application 1: Stack frame of function call

- A program places an activation record or a stack frame on top of the system stack when it invokes a function.

※fp: a pointer to current stack frame




system stack before a1 is invoked




system stack after a1 is invoked

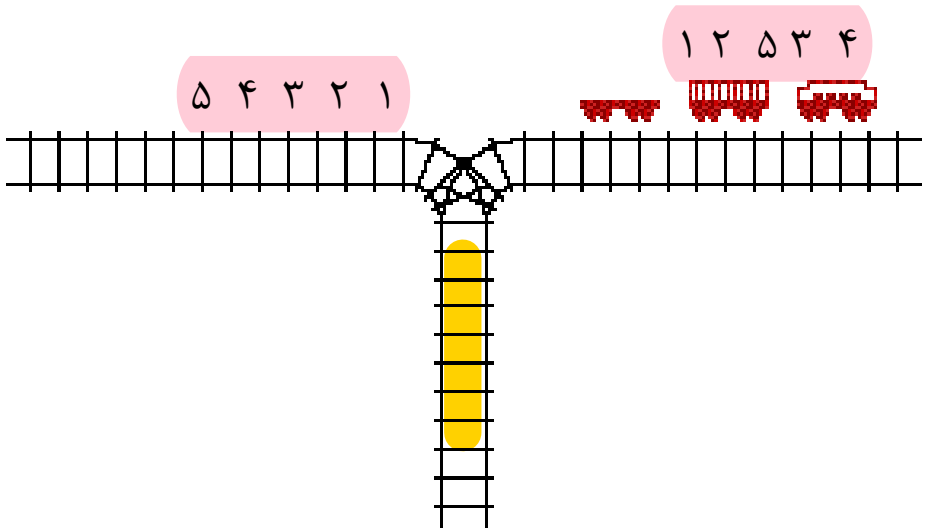
# Stacks-Example of function call

```
#include <stdio.h>
main()
{
   int x;
   x = fact(5);
}
int fact(int n)
{
   if (n>1)
     return n*fact(n-1);
   else
     return 1;
}
```

**X = ?**

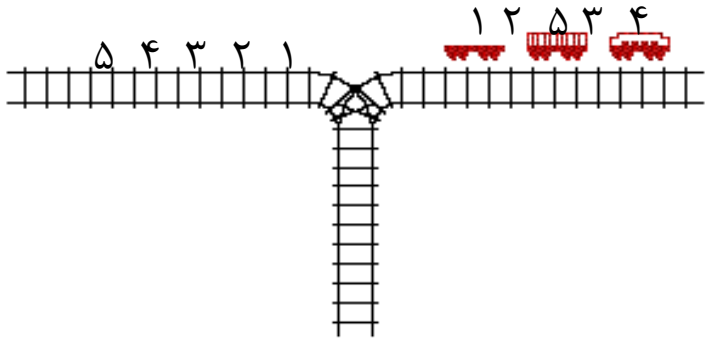
invoke fact(5)
invoke fact(4)
invoke fact(3)
invoke fact(2)
invoke fact(1)
return from fact(1) = 1
return from fact(2) = 2
return from fact(3) = 6
return from fact(4) = 24
return from fact(5) = 120

# Application 2: پارکینگ قطارها –railroad switching




- آیا می توانیم شرایطی را تعریف کنیم که اگر خروجی آن شرایط را داشت حتما می توان آن را تولید کرد؟
  - راه حل تئوری
  - پیاده سازی با پشته

# Application 2: پارکینگ قطارها –railroad switching




- پیاده سازی با پشته
  - اولین عنصر خروجی را مشخص کن
  - تمام عناصر قبل از آن را در استک push کن
- اگر تمام عناصر در خروجی قرار گرفته اند ترتیب خواسته شده قابل تولید است و برنامه تمام می شود.
- در غیر این صورت عنصر بعدی خروجی را بررسی کن
  - بالای استک قرار دارد
  - در لیست عناصریست که هنوز وارد استک نشده اند
- اگر هیچ یک از دو حالت بالا نبود تولید این خروجی غیر ممکن است و برنامه تمام می شود.

# Application 3: Evaluation of expression

## ارزیابی عبارت ها

X = a / b - c + d * e - a * c

a = 4, b = c = 2, d = e = 3

Interpretation 1:
((4/2)-2)+(3*3)-(4*2)=0 + 8+9=1

Interpretation 2:
(4/(2-2+3))*(3-4)*2=(4/3)*(-1)*2=-2.66666…

- How to generate the machine instructions corresponding to a given expression?

precedence rule + associative rule

# Precedence hierarchy

- In any programming language, a precedence hierarchy determines the order in which we evaluate operators.
- Operators with highest precedence are evaluated first.
- With right associative operators of the same precedence, we evaluate the operator furthest to the right first.
- Expressions are always evaluated from the innermost parenthesized expression first.

# Precedence hierarchy for C

| Token | Operator | Precedence[1] | Associativity |
|---|---|---|---|
| ( )<br>[ ]<br>-> . | function call<br>array element<br>struct or union member | 17 | left-to-right |
| -- ++ | increment, decrement[2] | 16 | left-to-right |
| -- ++<br>!<br>-<br>- +<br>& *<br>sizeof | decrement, increment[3]<br>logical not<br>one's complement<br>unary minus or plus<br>address or indirection<br>size (in bytes) | 15 | right-to-left |
| (type) | type cast | 14 | right-to-left |
| * / % | mutiplicative | 13 | Left-to-right |

1.The precedence column is taken from Harbison and Steele.
2.Postfix form
3.prefix form

# Precedence hierarchy for C

| | | | |
|---|---|---|---|
| + - | binary add or subtract | 12 | left-to-right |
| << >> | shift | 11 | left-to-right |
| > >=<br>< <= | relational | 10 | left-to-right |
| == != | equality | 9 | left-to-right |
| & | bitwise and | 8 | left-to-right |
| ^ | bitwise exclusive or | 7 | left-to-right |
| \| | bitwise or | 6 | left-to-right |
| && | logical and | 5 | left-to-right |
| \|\| | logical or | 4 | left-to-right |

# Precedence hierarchy for C

| | | | |
|---|---|---|---|
| ?: | conditional | 3 | right-to-left |
| = += -= /= *= %= <<= >>= &= ^= \|= | assignment | 2 | right-to-left |
| , | comma | 1 | left-to-right |

X = a / b - c + d * e - a * c ⟶ X = (((a / b) - c) + (d * e) )- (a * c)

# Infix and Postfix notation

- چگونه کامپایلر یک عبارت را بررسی می کند؟
  - ابتدا با استفاده از پشته معادل پسوندی عبارت محاسبه می شود..
  - سپس با استفاده از پشته عبارت ارزیابی می شود.

- چرا کامپایلر از نمادگزاری پسوندی استفاده میکند؟
  - به دلیل سادگی:
    - در عبارت پسوندی پرانتزگزاری لازم نیست.
    - در عبارت پسوندی بررسی اولویت عملگرها لازم نیست.

- دو سوال مطرح:
  - چگونه عبارت میانوندی را به عبارت پسوندی تبدیل کنیم؟
  - چگونه عبارت پسوندی حاصل را ارزیابی کنیم؟

# Infix and Postfix Notation

| Infix | Postfix |
|---|---|
| 2+3*4 | 234*+ |
| a*b+5 | ab*5+ |
| (1+2)*7 | 12+7* |
| a*b/c | ab*c/ |
| (a/(b-c+d))*(e-a)*c | abc-d+/ea-*c* |
| a/b-c+d*e-a*c | ab/c-de*+ac*- |

# Evaluating Postfix Expressions

**AB/C – DE*+AC*–**

| عمل | پسوند |
|---|---|
| $T_1 = A/B$ | $T_1C - DE*+AC*-$ |
| $T_2 = T_1 - C$ | $T_2DE*+AC*-$ |
| $T_3 = D*E$ | $T_2T_3+AC*-$ |
| $T_4 = T_2+T_3$ | $T_4AC*-$ |
| $T_5 = A*C$ | $T_4T_5-$ |
| $T_6 = T_4- T_5$ | $T_6$ |

# Evaluating Postfix Expressions

Example: 6 2 / 3 – 4 2 * +

| Token | Stack [0] | [1] | [2] | Top |
|---|---|---|---|---|
| 6 | 6 | | | 0 |
| 2 | 6 | 2 | | 1 |
| / | 6/2 | | | 0 |
| 3 | 6/2 | 3 | | 1 |
| - | 6/2-3 | | | 0 |
| 4 | 6/2-3 | 4 | | 1 |
| 2 | 6/2-3 | 4 | 2 | 2 |
| * | 6/2-3 | 4*2 | | 1 |
| + | 6/2-3+4*2 | | | 0 |

# Evaluating Postfix Expressions

- Evaluation process
- Make a single left-to-right scan of the expression.
- Place the operands on a stack until an operator is found.
- Remove, from the stack, the correct numbers of operands for the operator, perform the operation, and place the result back on the stack.