به نام خدا

ساختمان داده ها

جلسه ششم

دانشگاه صنعتی همدان

گروه مهندسی کامپیوتر

نیم سال دوم 98-1397

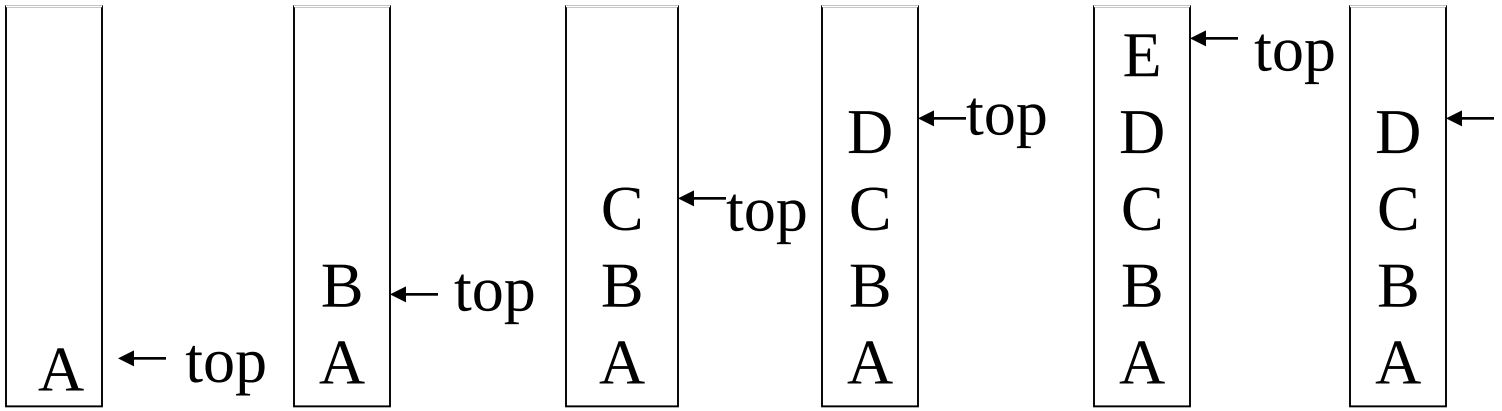
فصل سوم

پشته و صف

Stack & Queue

پشته

- پشته ساختمان داده ای است که داده ها را به ترتیب خاصی ذخیره می کند.
- در پشته اخرین عضوی که وارد می شود اولین عضوی است که خارج می شود.LAST IN FIRST OUT (LIFO)
- عنصر بالای پشته را top می گویند.


A
top
B
A
top
C
B
A
top
D
C
B
A
top
E
D
C
B
A
top
D
C
B
A

پشته ADT

objects: a finite ordered list with zero or more elements.
methods:
for all *stack* ∈ *Stack*, *item* ∈ *element*, *max_stack_size* ∈ positive integer

Stack createS(*max_stack_size*) ::=
create an empty stack whose maximum size is *max_stack_size*

Boolean isFull(*stack, max_stack_size*) ::=
if (number of elements in *stack* == *max_stack_size*)
return TRUE
else return FALSE

Stack push(*stack, item*) ::=
if (IsFull(*stack*)) *stack_full*
else insert *item* into top of *stack* and **return**

Boolean isEmpty(*stack*) ::=
if(*stack* == CreateS(*max_stack_size*))
return TRUE
else return FALSE

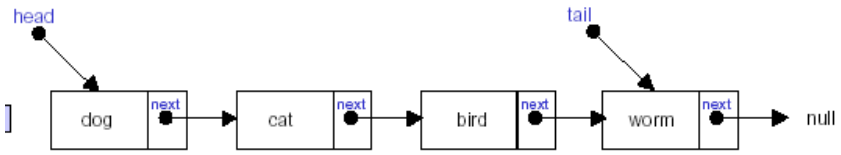
Element pop(*stack*) ::=
if(IsEmpty(*stack*)) **return**
else remove and return the *item* on the top of the stack.

پیاده سازی پشته ها

با چندین روش می توان آنها را پیاده سازی کرد:

- آرایه ها

- لیستهای پیوندی (در قسمت لیستهای پیوندی بررسی می شود)



پیاده سازی پشته ها به روش ارایه – نمایش داده ها

- برای اینکه بتوانیم از پشته ها برای انواع داده های مختلف استفاده کنیم از قالبها (template) استفاده می کنیم.

```
private:
     int top;
     KeyType *stack;
     int MaxSize;
```

سازنده کلاس پشته

```
template <class KeyType>
Stack<KeyType>::Stack (int MaxStackSize):MaxSize (MaxStackSize)
{
    stack = new KeyType[MaxSize];
    top = -1;
}
```

پیاده سازی اعمال روی پشته ها

```
template <class KeyType>
inline Boolean  Stack<KeyType>::IsFull()
{
    if (top == MaxSize -1) return TRUE;
    else return FALSE;
}

template <class KeyType>
inline Boolean  Stack<KeyType>::IsEmpty()
{
    if (top == -1) return TRUE;
    else return FALSE;
}
```

پیاده سازی اعمال روی پشته ها

تابع افزودن یک عنصر به پشته

```
template <class KeyType>
void Stack<KeyType>::Add (const KeyType& x)
// add x to the stack
{
   if (IsFull()) StackFull();
   else stack[++top] = x;
}
```

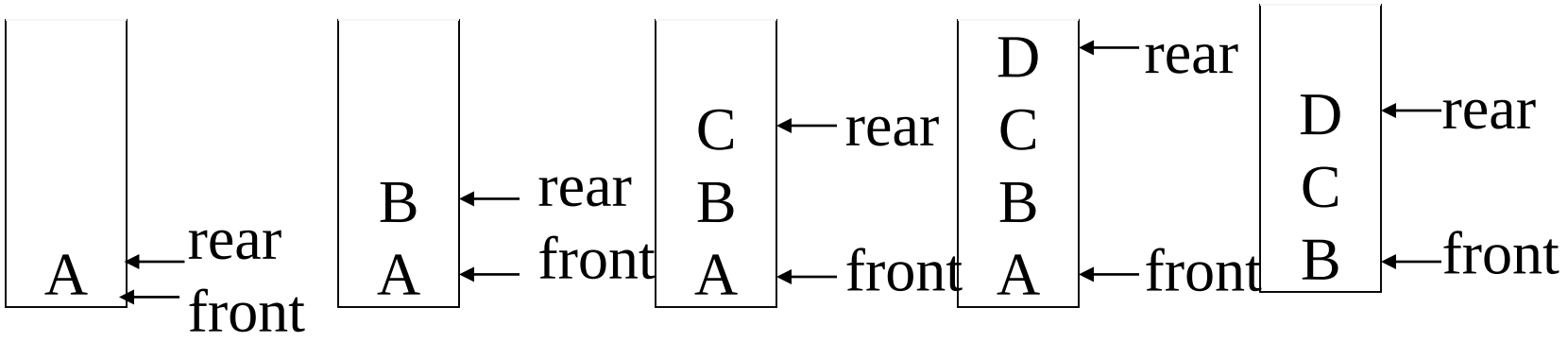
تابع حذف یک عنصر از پشته

```
template <class KeyType>
KeyType* Stack<KeyType>::Delete (KeyType& x)
// remove and return top element from stack
{
   if (IsEmpty()) {StackEmpty(); return 0;}
   x = stack[top--];
   return &x;
}
```

```
void main()
{
    Stack<int> s;
    int x;
    s.Add(5);
    s.Add(7);
    s.Delete(x);
    s.Add(9);
    s.Add(10);
    s.Delete(x);
    s.Delete(x);
    s.Delete(x);
}
```

صف (Queue)

- صف یک لیست ترتیبی است که تمام درج ها از یک سمت و حذف ها از سمت دیگر انجام می گیرد.
- بنابر این به صف FIFO (First In First Out) می گویند.
- اولین عضو ورودی اولین عضوی است که خارج می شود.
- عنصر ابتدای صف را Front و انتهای صف را Rear می گویند.


A
rear
front
B
A
rear
front
C
B
A
rear
front
D
C
B
A
rear
front
D
C
B
rear
front

صف ADT

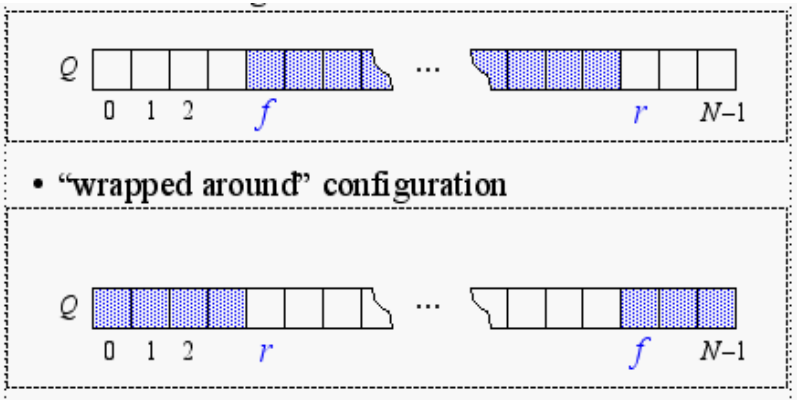
objects: a finite ordered list with zero or more elements.
methods:

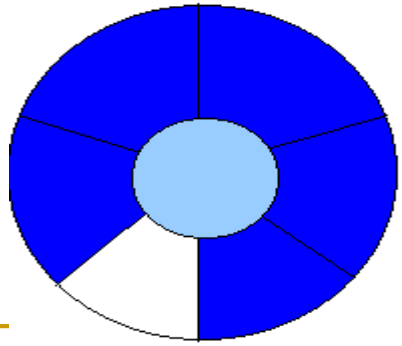
for all *queue* ∈ *Queue*, *item* ∈ *element*,
max_ queue_ size ∈ positive integer
Queue createQ(*max_queue_size*) ::=
create an empty queue whose maximum size is
max_queue_size
Boolean isFullQ(*queue, max_queue_size*) ::=
if(number of elements in *queue* == *max_queue_size*)
return *TRUE*
else return *FALSE*
Queue Enqueue(*queue, item*) ::=
if (IsFullQ(*queue*)) *queue_full*
else insert *item* at rear of *queue* and return *queue*
Boolean isEmptyQ(*queue*) ::=
if (*queue* ==CreateQ(*max_queue_size*))
return *TRUE*
else return *FALSE*
Element dequeue(*queue*) ::=
if (IsEmptyQ(*queue*)) **return**
else remove and return the *item* at front of queue.

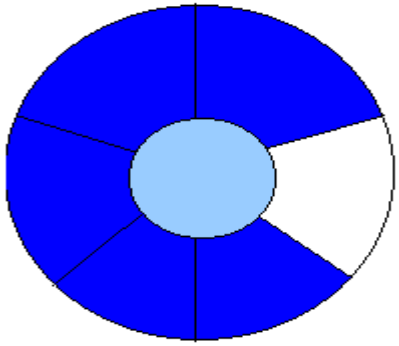
پیاده سازی صف

به چندین روش می توان صف را پیاده کرد:

- استفاده از ارایه ها
  - استفاده از ارایه با دو متغیر front و rear
  - استفاده از ارایه با یک متغیر rear (front همیشه صفر است)
  - استفده از ارایه به صورت حلقوی
- استفاده از لیستهای پیوندی







ساختار داده ای صف

```
private:
    int front;
    int rear;
    KeyType *queue;
    int MaxSize;
```

سازنده صف

```
template <class KeyType>
Queue<KeyType>::Queue (int MaxQueueSize) :
MaxSize(MaxQueueSize)
{
    queue = new KeyType[MaxSize];
    front = rear = -1;
}
```

پیاده سازی اعمال صف

تابع بررسی پر بودن صف

```
template <class KeyType>
inline Boolean  Queue<KeyType>::IsFull()
{
    if (rear == MaxSize -1) return TRUE;
    else return FALSE;
}
```

تابع بررسی خالی بودن صف

```
template <class KeyType>
inline Boolean  Queue<KeyType>::IsEmpty()
{
    if (front == rear) return TRUE;
    else return FALSE;
}
```

پیاده سازی اعمال صف

تابع اضافه کردن یک عنصر به صف

```
template <class KeyType>
void Queue<KeyType>::Add (const KeyType& x)
// add x to the queue
{
    if (IsFull()) QueueFull();
    else queue[++rear] = x;
}
```

تابع حذف کردن یک عنصر از صف

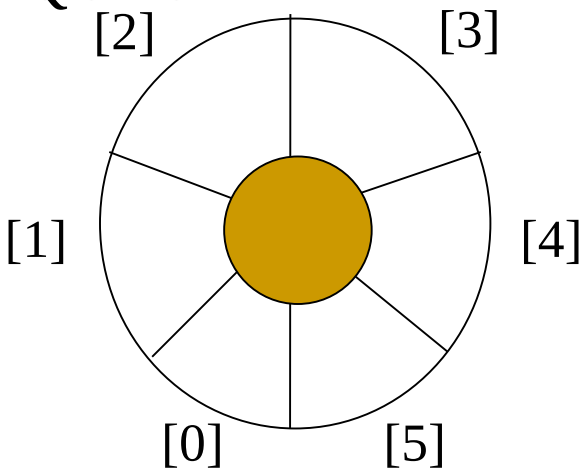
```
template <class KeyType>
KeyType* Queue<KeyType>::Delete (KeyType& x)
// remove and return front element from queue
{
    if (IsEmpty()) {QueueEmpty(); return 0;}
    x = queue[++front];
    return &x;
}
```

```
void main()
{
     Queue<int> s(2);
     int x;
     s.Add(5);
     s.Add(7);
     s.Delete(x);
     s.Add(9);
     s.Add(10);
     s.Delete(x);
     s.Delete(x);
     s.Delete(x);
}
```

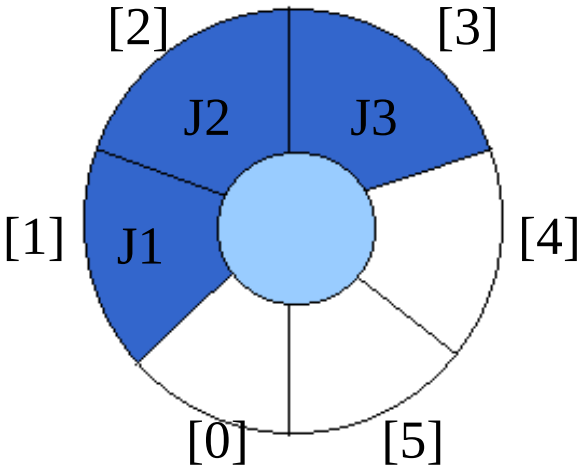
معایب این روش

- در این روش اعضا به ترتیب وارد صف می شوند و به ترتیب از انتهای صف حذف می شوند ولی فضای عناصری که حذف می شوند دوباره استفاده نمی شوند و صف بعد از مدتی دیگر فضای خالی نخواهد داشت:
- دو روش برای حل این معضل وجود دارد:
  - شیفت دادن عناصر
  - استفاده از صف دایره ای

EMPTY QUEUE

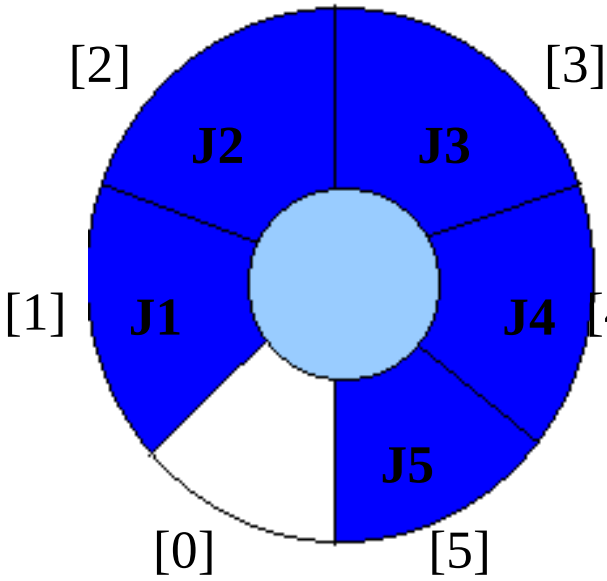


front = 0
rear = 0



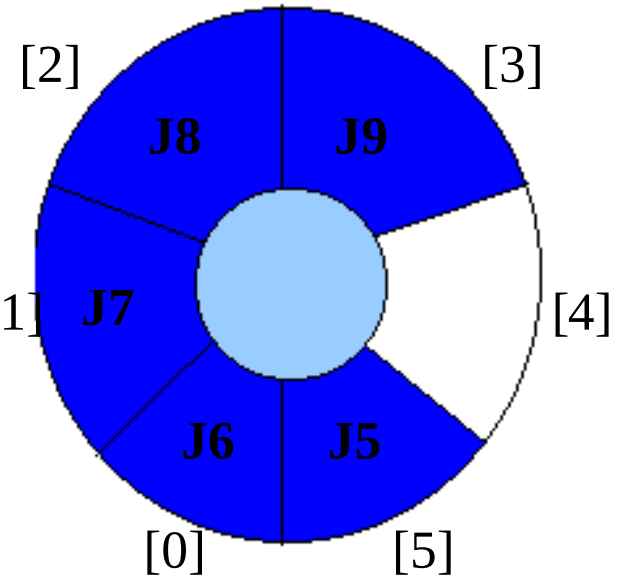
front = 0
rear = 3

FULL QUEUE



front =0
rear = 5

FULL QUEUE



front =4
rear =3

پیاده سازی صف حلقوی

```
private:
    int front;
    int rear;
    KeyType *queue;
    int MaxSize;
```

سازنده کلاس

```
template <class KeyType>
Queue<KeyType>::Queue (int MaxQueueSize) :
MaxSize(MaxQueueSize)
{
    queue = new KeyType[MaxSize];
    front = rear = 1;
}
```

توابع پر بودن و خالی بودن صف

```
template <class KeyType>
Boolean  Queue<KeyType>::IsFull()
{
    if (rear == MaxSize -1) return TRUE;
    else return FALSE;
}

template <class KeyType>
Boolean  Queue<KeyType>::IsEmpty()
{
    if (front == rear) return TRUE;
    else return FALSE;
}
```

حذف و اضافه کردن در صف حلقوی

```
template <class KeyType>
void Queue<KeyType>::Add (const KeyType& x)
// add x to the queue
{
    int k = (rear + 1) % MaxSize;
    if (front == k) QueueFull();
    else queue[rear = k] = x;
}

template <class KeyType>
KeyType* Queue<KeyType>::Delete (KeyType& x)
// remove and return top element from queue
{
  if (front == rear) {QueueEmpty(); return 0;}
    x = queue[++front %= MaxSize];
    return &x;
}
```

```
void main()
{
    Queue<int> s(5);
    int x;
    s.Add(5);
    s.Add(7);
    s.Delete(x);
    s.Add(9);
    s.Add(10);
    s.Delete(x);
    s.Delete(x);
    s.Delete(x);
}
```

مثالهایی از کاربردهای پشته و صف

ارزیابی عبارتهای ریاضی

- نحوه نمایش عبارتهای ریاضی
  - روش میانوندی nfix A+B
  - روش پسوندی Postfix AB+
  - روش پیشوندی Prefix +AB

مزیت روشهای پیشوندی و پسوندی این است که پرانتزها و کروشه ها دیده نمی شوند و اولویت اعمال هم وجود ندارد و اعمال به ترتیب قرار گرفتن اجرا می شوند.

بنابراین کامپایلرها برای ارزیابی عبارتهای ریاضی آنها را به یکی از فرمهای پسوندی یا پیشوندی تبدیل می کنند.

تبدیل عبارتهای میانوندی به پسوندی

Example: A / B – C + D * E – A * C

```
- ( ( ( ( A / B ) - C ) + ( D * E ) ) - ( A * C ) )
- ( ( ( ( A   B /   C -   ( D   E * +     A   C * -
- A B / C - D E * + A C * -
```

برای تبدیل دستی یک عبارت میانوندی به پسوندی به صورت زیر عمل می کنیم:

۱- عبارت را به صورت کامل پرانتز گزاری می کنیم

۲-عملگرها را به سمت راست حرکت می دهیم و جایگزین پرانتزهای سمت راست می کنیم

۳- تمام پرانتزها را حذف می کنیم

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*-

تبدیل عبارتهای میانوندی به پسوندی

- آیا این روش برای یک الگوریتم کامپیوتری هم موثر است؟

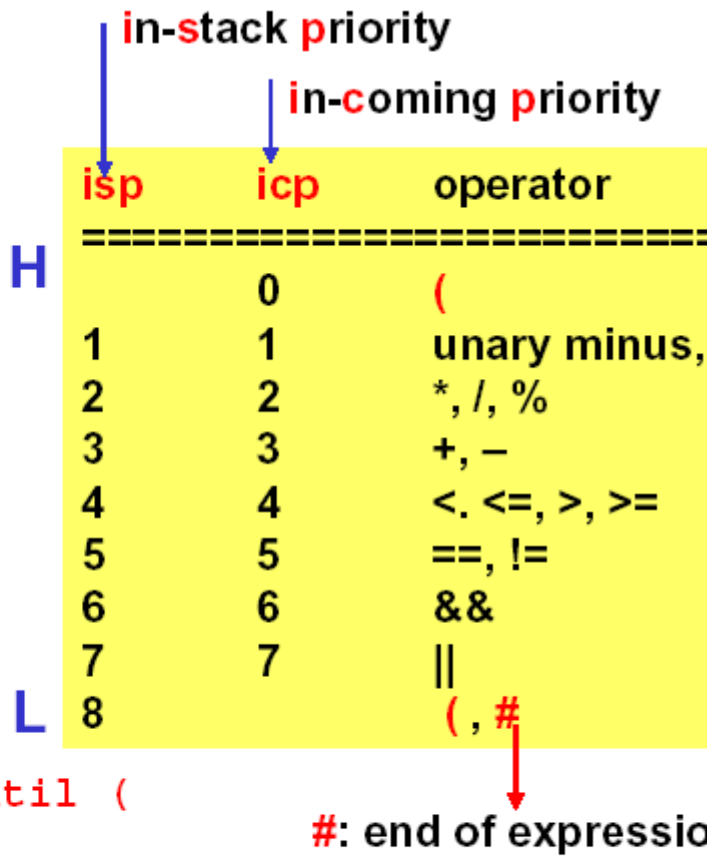
می توان روش بهتری ارایه کرد :

- عبارت پسوندی را از چپ به راست جمله به جمله (توکن به توکن) در نظر می گیریم
- اگر توکن مورد نظرعملوند بود آنرا در خروجی می نویسیم
- اگر توکن مورد نظر عملگر بود
  - اگر اولویت ان از اولویت عنصر بالای پشته بیشتر بود در پشته قرار می دهیم
  - اگر اولویت آن کمتر یا مساوی عنصر بالای پشته بود عنصر بالای پشته را خارج کرده و در خروجی می نویسیم. این عمل انقدر تکرار می شود تا اولویت توکن از اولویت عنصر بالای پشته بیشتر شود.

- (در مورد اولویتها یک استثنا در مورد پرانتز وجود دارد: پرانتزچپ داخل پشته کمترین اولویت دارد و پرانتز راست باعث بیرون آمدن آن می شود و پرانتز چپ بیرون پشته بالاترین اولویت را دارد)

Example: A*(B+C)/D

next token	stack	output
none	#	none
A	#	A
*	#*	A
(	#*(	A
B	#*(	AB
+	#*(+	AB
C	#*(+	ABC
)	#*	ABC+ // pop until (
/	#/	ABC+*
D	#/	ABC+*D
done		ABC+*D/# // empty the stack



تابع تبدیل عبارت میانوندی به پسوندی

```
void postfix (expression e)
// Output the postfix form of the infix expression e. NextToken
// and stack are as in function eval (Program 3.18). It is assumed that
// the last token in e is '#.' Also, '#' is used at the bottom of the stack
{
   Stack<token> stack; // initialize stack
   token y ;
   stack.Add('#');
   for (token x = NextToken(e) ; x != '#' ; x = NextToken(e))
   {
      if (x is an operand)  cout << x ;
      else if (x == ')')  // unstack until '('
         for (y = *stack.Delete (y); y != '('; y = *stack.Delete (y)) cout << y ;
      else { // x is an operator
         for (y = *stack.Delete (y); isp (y) <= icp (x); y = *stack.Delete (y)) cout << y ;
         stack.Add(y);   // restack the last y that was unstacked
         stack.Add(x);
      }
   }

   // end of expression; empty stack
   while (!stack.IsEmpty()) cout << *stack.Delete(y) ;
} // end of postfix
```

ارزیابی یک عبارت پسوندی

```
void eval(expression e)
// Evaluate the postfix expression e. It is assumed that the last token (a token
// is either an operator, operand, or '#') in e is '#.' A function NextToken is
// used to get the next token from e. The function uses the stack stack
{
   Stack<token> stack ; //initialize stack
   for (token x = NextToken (e) ; x != '#'; x = NextToken (e))
      if (x is an operand) stack.Add(x) // add to stack
      else { // operator
         remove the correct number of operands for operator x from stack; perform the
         operation x and store the result (if any) onto the stack;
      }
```