ساختمان داده ها

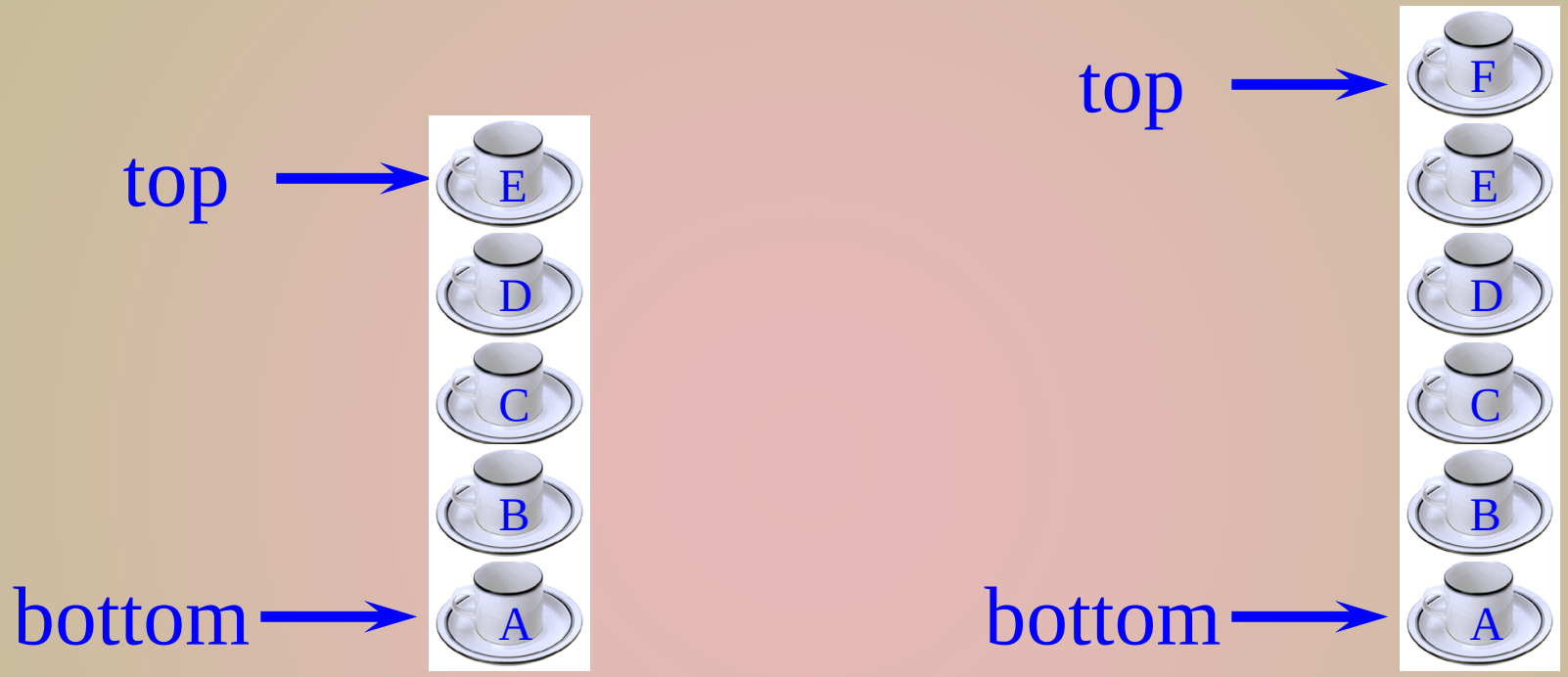
پشته (Stack)

مدرس: غیاثی‌شیرازی
دانشگاه فردوسی مشهد

پشته

- پشته نوعی لیست خطی است که عملیات دریافت، درج و حذف از آن به انتهای لیست محدود شده است.
- معمولا داده های پشته را به صورت عمودی تصور می کنیم و می گوییم که عملیات تنها بر بالای پشته قابل انجام است.
- پشته یک لیست با خاصیت LIFO (/laifo/) است.
- Last In First Out

پشته ای از فنجان ها



- عملیات تنها از بالای پشته قابل انجام است:
  - اضافه کردن یک فنجان
  - برداشتن (حذف) یک فنجان

عملیات بر روی پشته

ADT

push	درج عنصری در بالای پشته
pop	حذف عنصر بالای پشته
top	گرفتن عنصر بالای پشته
size	تعداد عناصر پشته را بر می گرداند
isEmpty	خالی بودن پشته را می آزماید.

طراحی نامناسب کلاس Stack در java.util

```
public class Stack<E> extends Vector<E> {
    public E push(E item)
    public synchronized E pop()
    public synchronized E peek()
    public boolean empty()
    public synchronized int search(Object o)
}
```

top

دلیل نامناسب بودن طراحی جاوا

- با ارث بری از کلاس Vector این امکان به استفاده کننده از پشته داده می شود که با آن همانند یک Vector کار کند و بر روی داده های میانی پشته نیز عملیاتی را انجام دهد.
- بنابراین قطعه کد زیر در جاوا کار می کند:

```
Stack<Integer> si = new Stack<Integer>();
        si.add(0,12);
        si.add(1,14);
        si.add(1,13);
```

Vector

نامناسب بودن طراحی ساختمان های داده جاوا از دیدگاهی دیگر

- ساختمان های داده بسته java.util خیلی عمومی طراحی شده اند. این باعث شده است که برخی از عملیات بسیار ناکارا پیاده سازی شوند.
- مثال: در جاوا رابط List عملیات درج و حذف از عناصر میانی را حمایت می کند. وقتی این رابط توسط کلاس ArrayList پیاده سازی می شود، منجر به انجام عمل درج به صورت بسیار ناکارا با زمان O(n) می شود.
- برخی ساختمان های داده مانند Queue اصولا به صورت interface طراحی شده اند و کارایی آنها مشخص نیست.

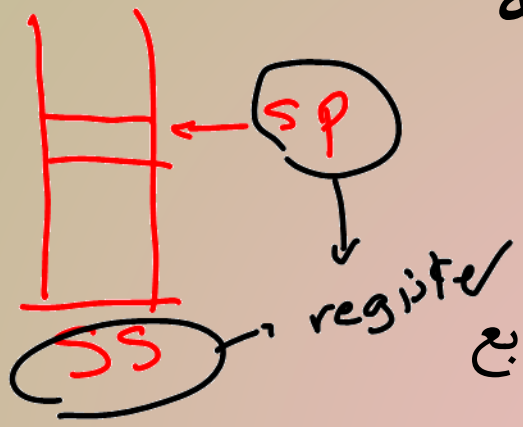
طراحی کلاس stack در C++

- در STL کلاس stack از کلاس دیگری ارث نمی برد.
- اما می توان کلاس خاصی را (به طور پیش فرض، deque) برای ذخیره داده ها استفاده کرد.

```
template< class T,    class Container = std::deque<T> >
class stack;
```

- در C++ کلاس stack تنها عملیات مجاز پشته را حمایت می کند و درج و حذف تنها از بالای پشته مجاز است.
- مطالعه بیشتر: http://en.cppreference.com/w/cpp/container/stack

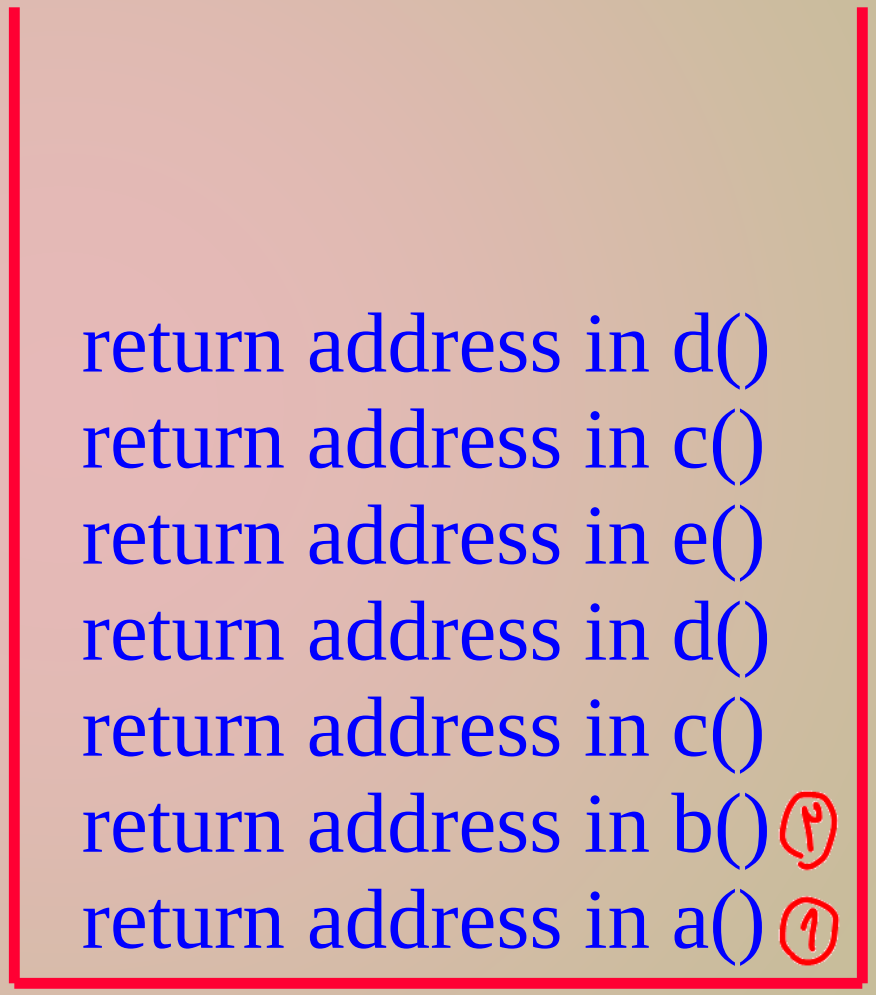
کاربردهای پشته



- کاربردهای کلاسیک
  - نگهداری آدرس بازگشت در فراخوانی توابع
  - پیاده سازی try-throw-catch
  - جستجوی عمق اول (توضیح در بخش پیمایش های گراف)
  - کامپیوترهای پشته (مانند برخی انواع ماشین حساب)
- کاربردهای دیگر:
  - تشخیص تطبیق پرانتزها در یک عبارت ریاضی
  - محاسبه عبارات ریاضی (الگوریتم حیاط راه آهن Dijkstra)

Method Invocation And Return

```
public void a()
{ …; b(); …}
public void b()
{ …; c(); …}
public void c()
{ …; d(); …}
public void d()
{ …; e(); …}
public void e()
{ …; c(); …}
```



Try-Throw-Catch

try {

} catch ( )

- When you enter a try block, push the address of this block on a stack.
- When an exception is thrown, pop the try block that is at the top of the stack (if the stack is empty, terminate).
- If the popped try block has no matching catch block, go back to the preceding step.
- If the popped try block has a matching catch block, execute the matching catch block.

$X = Y + Z$

~~ADD(X, Y, Z)~~

PC

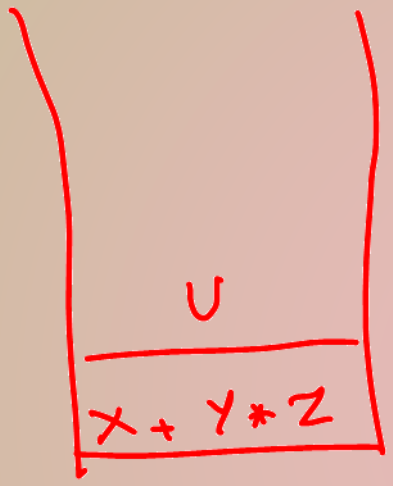
$X = Z$

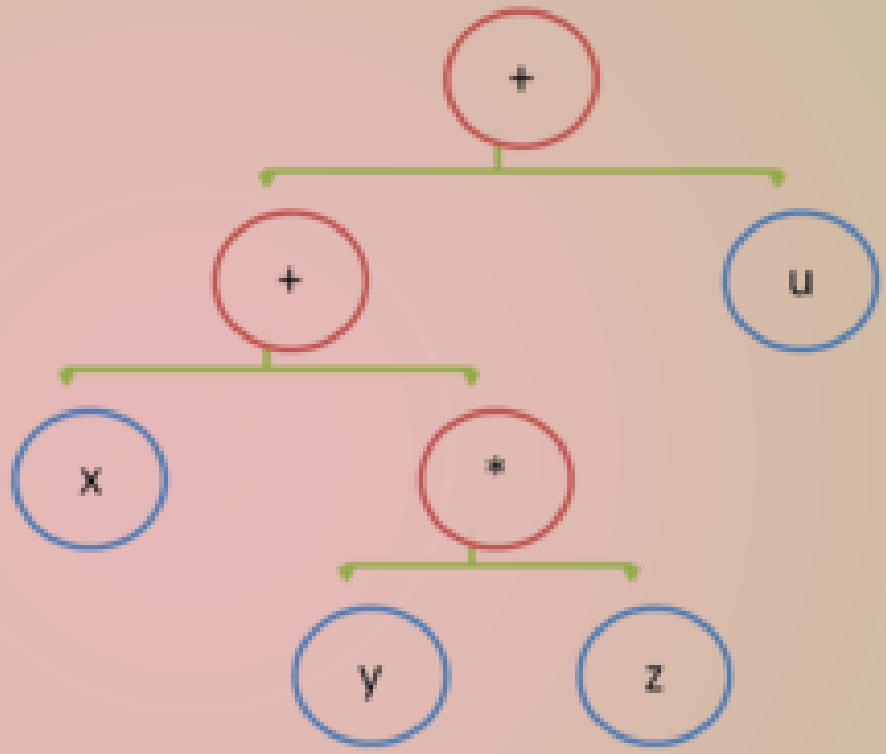
ADD X, Y

$X = X + Y$

ماشین های پشته (Stack Machines)

push x
push y
push z
multiply
add
push u
add





- http://en.wikipedia.org/wiki/Stack_machine

Parentheses Matching

- (((a+b)*c+d-e)/(f+g)-(h+j)*(k-l))/(m-n)
  - Output pairs (u,v) such that the left parenthesis at position u is matched with the right parenthesis at v.
    - (2,6) (1,13) (15,19) (21,25) (27,31) (0,32) (34,38)
- (a+b))*((c+d)
  - (0,4)
  - right parenthesis at 5 has no matching left parenthesis
  - (8,12)
  - left parenthesis at 7 has no matching right parenthesis

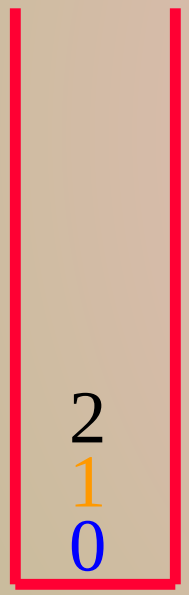
Parentheses Matching

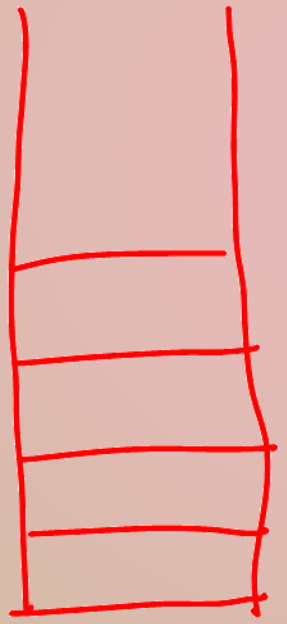
- scan expression from left to right
- when a left parenthesis is encountered, add its position to the stack
- when a right parenthesis is encountered, remove matching position from stack

Example

- (((a+b)*c+d-e)/(f+g)-(h+j)*(k-l))/(m-n)

0 12 6 13 15 19 21 25 27 31 32 34 38





(2, 6) (1, 13)

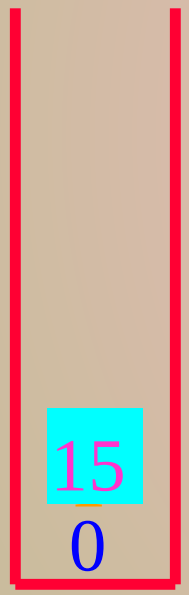
(15, 19) (21, 25)

(27, 31) (0, 32)

(34, 38)

Example

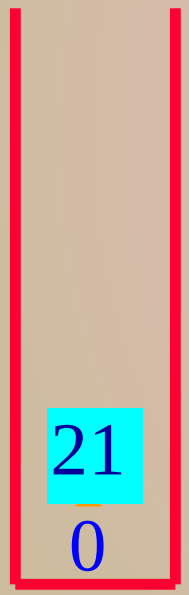
- (((a+b)*c+d-e)/(f+g)-(h+j)*(k-l))/(m-n)



(2,6) (1,13)

Example

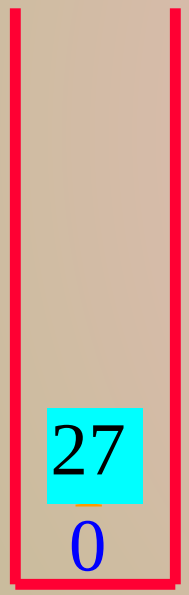
- (((a+b)*c+d-e)/(f+g)-(h+j)*(k-l))/(m-n)



(2,6) (1,13) (15,19)

Example

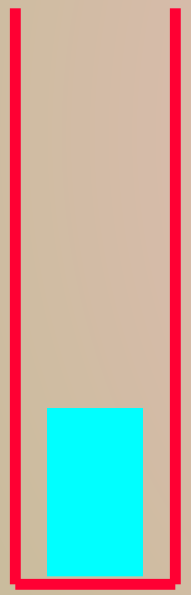
- (((a+b)*c+d-e)/(f+g)-(h+j)*(k-l))/(m-n)



(2,6) (1,13) (15,19) (21,25)

Example

- (((a+b)*c+d-e)/(f+g)-(h+j)*(k-l))/(m-n)



(2,6) (1,13) (15,19) (21,25)(27,31) (0,32)

- and so on

عبارات ریاضی به فرم پسوندی (Postfix) یا نمایش لهستانی معکوس (RPN)

- عبارات ریاضی: infix

- a+b
- (a+b)^3*2/7
- 3/sin(3.14*x/y^(y-z))/16

- در فرم RPN برابرند با:

- a b +
- a b + 3 ^ 2 * 7 /
- 3 3.14 x * y y z - ^ / sin / 16 /

نام گذاری

- RPN مخفف Reverse Polish Notation است و دلیل این نامگذاری این است که نمایش prefix با توجه به ملیت لهستانی مبدع آن (Jan Łukasiewicz) نمایش لهستانی (Polish Notation) نامیده می شود.

عملگر یگانی (Unary) منفی کردن

- در RPN عملگر یگانی منفی کردن (-) باید با علامت دیگری (مثلا ؟) نشان داده شود تا با عملگر تفریق اشتباه نشود.
- برای مثال عبارت 1-(-2+3) در فرم RPN چنین است:
- 1 2 ? 3 + -

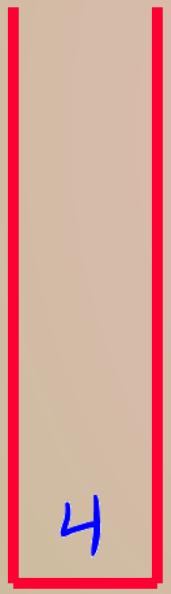
1 2 ? 3 + -
1

نحوه محاسبه عبارتی به فرم RPN

- الگوریتم محاسبه عبارات ریاضی به فرم RPN چنین است:
  - نشان (token) بعدی را بخوان
  - اگر token یک عدد یا یک متغیر است، آن را در پشته قرار بده.
  - اگر token یک عملگر است، آن را به عناصر بالای پشته اعمال کن و حاصل را در بالای پشته ذخیره کن.
  - عملیات بالا را تکرار کن تا به انتهای عبارت ریاضی برسی.
  - در پایان حاصل عبارت در پشته قرار دارد.

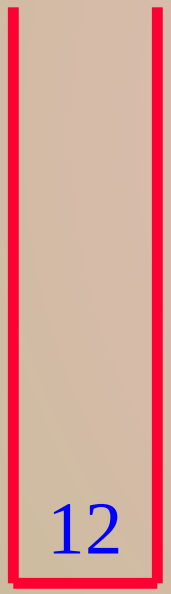
مثال: محاسبه عبارتی به فرم RPN

- 12 3 30 * 1 1 2 - ^ / sin / 3 /



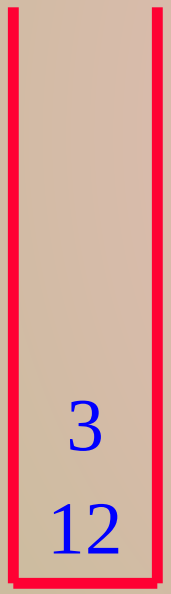
مثال: محاسبه عبارتی به فرم RPN

- 12 3 30 * 1 1 2 - ^ / sin / 3 /



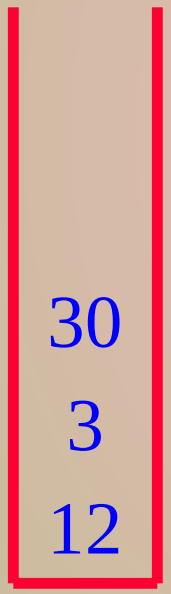
مثال: محاسبه عبارتی به فرم RPN

- 12 3 30 * 1 1 2 - ^ / sin / 3 /



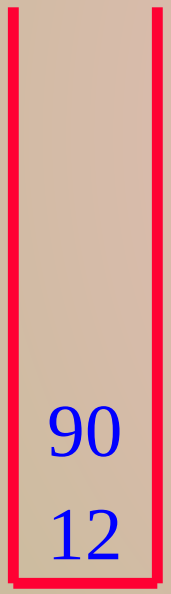
مثال: محاسبه عبارتی به فرم RPN

- 12 3 30 * 1 1 2 - ^ / sin / 3 /



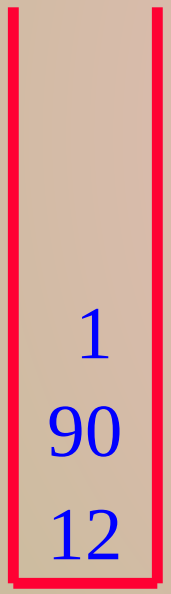
مثال: محاسبه عبارتی به فرم RPN

- 12 3 30 * 1 1 2 - ^ / sin / 3 /



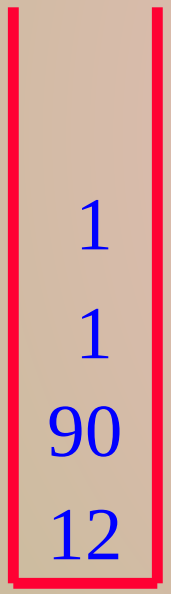
مثال: محاسبه عبارتی به فرم RPN

- 12 3 30 * 1 1 2 - ^ / sin / 3 /



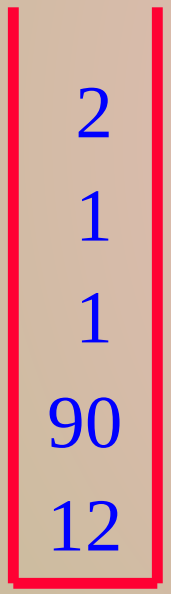
مثال: محاسبه عبارتی به فرم RPN

- 12 3 30 * 1 1 2 - ^ / sin / 3 /



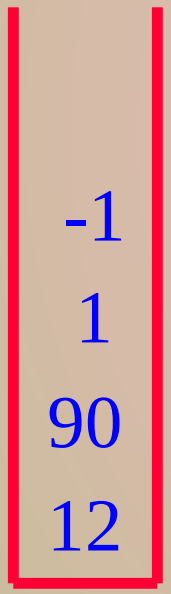
مثال: محاسبه عبارتی به فرم RPN

- 12 3 30 * 1 1 2 - ^ / sin / 3 /



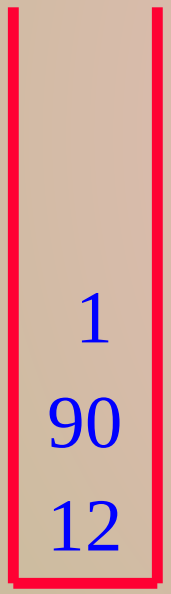
مثال: محاسبه عبارتی به فرم RPN

- 12 3 30 * 1 1 2 - ^ / sin / 3 /



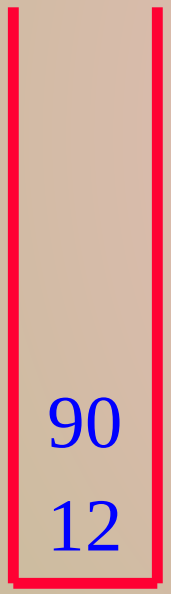
مثال: محاسبه عبارتی به فرم RPN

- 12 3 30 * 1 1 2 - ^ / sin / 3 /



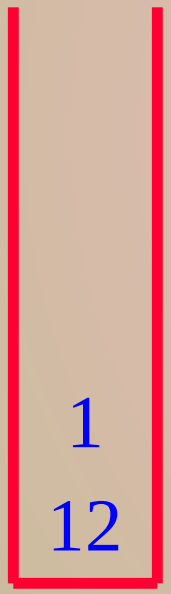
مثال: محاسبه عبارتی به فرم RPN

- 12 3 30 * 1 1 2 - ^ / sin / 3 /



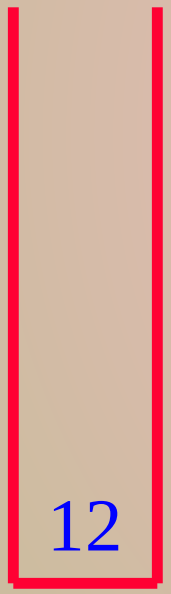
مثال: محاسبه عبارتی به فرم RPN

- 12 3 30 * 1 1 2 - ^ / sin / 3 /



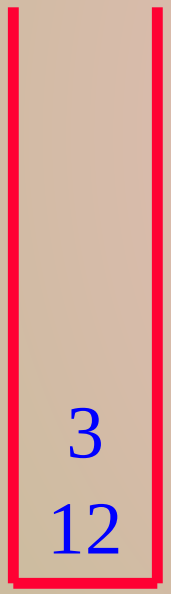
مثال: محاسبه عبارتی به فرم RPN

- 12 3 30 * 1 1 2 - ^ / sin / 3 /



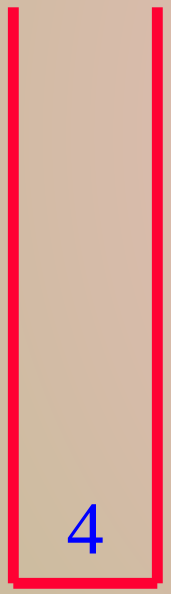
مثال: محاسبه عبارتی به فرم RPN

- 12 3 30 * 1 1 2 - ^ / sin / 3 /

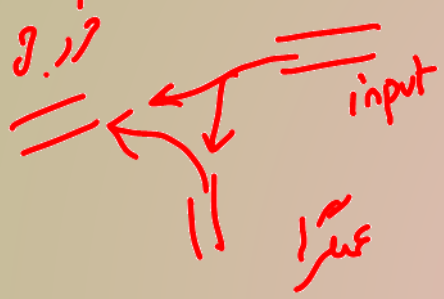


مثال: محاسبه عبارتی به فرم RPN

- 12 3 30 * 1 1 2 - ^ / sin / 3 /



Dijkstra's Shunting-yard algorithm
الگوریتم حیاط راه آهن دایجکسترا



http://en.wikipedia.org/wiki/Shunting-yard_algorithm

کاربرد

- کاربردهای الگوریتم حیاط راه آهن دایجکسترا
  - به دست آوردن فرم RPN عبارت ریاضی
  - محاسبه یک عبارت ریاضی داده شده به فرم میانوند (معمولی)

خاصیت انجمنی Associativity

- در عبارت a ~ b ~ c می گوییم عملگر ~ دارای خاصیت انجمنی از چپ (Left Associativity) است هرگاه مقدار عبارت فوق به صورت (a ~ b) ~ c تفسیر شود.
- در عبارت a ~ b ~ c می گوییم عملگر ~ دارای خاصیت انجمنی از راست (Right Associativity) است هرگاه مقدار عبارت فوق به صورت a ~ (b ~ c) تفسیر شود.

a ~ b ~ c ⟸ انجمنی از چپ ⟹ (a ~ b) ~ c

a ~ b ~ c ⟸ انجمنی از راست ⟹ a ~ (b ~ c)

$2 / 3 / 4$

$\frac{(2/3)}{4}$

$2 - 2 - 3$

$(2 - 2) - 3$

x^{y^z}

$x^{(y^z)}$

$x \uparrow y \uparrow 2$

$x \uparrow (y \uparrow z)$

تقدم و خاصیت انجمنی عملگرها

operator	precedence	associativity
^	4	Right
*	3	Left
/	3	Left
+	2	Left
-	2	Left

$2 + 3 \wedge 4$

$2 + (3^4)$

$1 + 2 \times 3 \wedge 4$

$1 + (2 \times (3^4))$

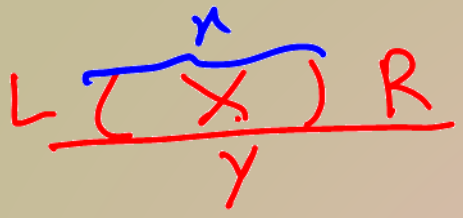
ترتیب اعداد و متغیرها

- ترتیب اعداد و متغیرها در نمایش میانوند و پسوندی تغییر نمی کند. بنابراین مساله یافتن فرم پسوندی معادل قرار دادن عملگرها به نحو صحیح در میان دنباله اعداد و متغیرها است.
- فرم میانوند:

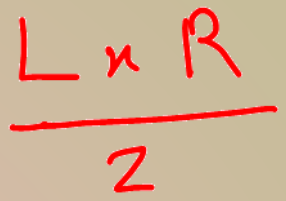
- (a+b)^3*2/7
- 3/sin(3.14*x/y^(y-z))/16

- فرم پسوندی:

- a b + 3 ^ 2 * 7 /
- 3 3.14 x * y y z - ^ / sin / 16 /



عبارات داخل پرانتز



- فرض کنیم X عبارتی داخل پرانتز باشد که درون عبارت Y ظاهر شده است.
- فرض کنیم با جایگزینی عبارت X با متغیر x در عبارت Y عبارت Z به دست می آید.
- نمایش پسوندی Y همان نمایش پسوندی Z است که x در آن با نمایش پسوندی X جایگزین شده است.

$Y = 2 * (3 + 4) \wedge 5$

$Z = 2 * x \wedge 5$

RPN(Z) = 2 x 5 ^ *

RPN(X) = 3 4 +

RPN(Y) = 2 3 4 + 5 ^ *

دسته بندی علایم یک عبارت ریاضی

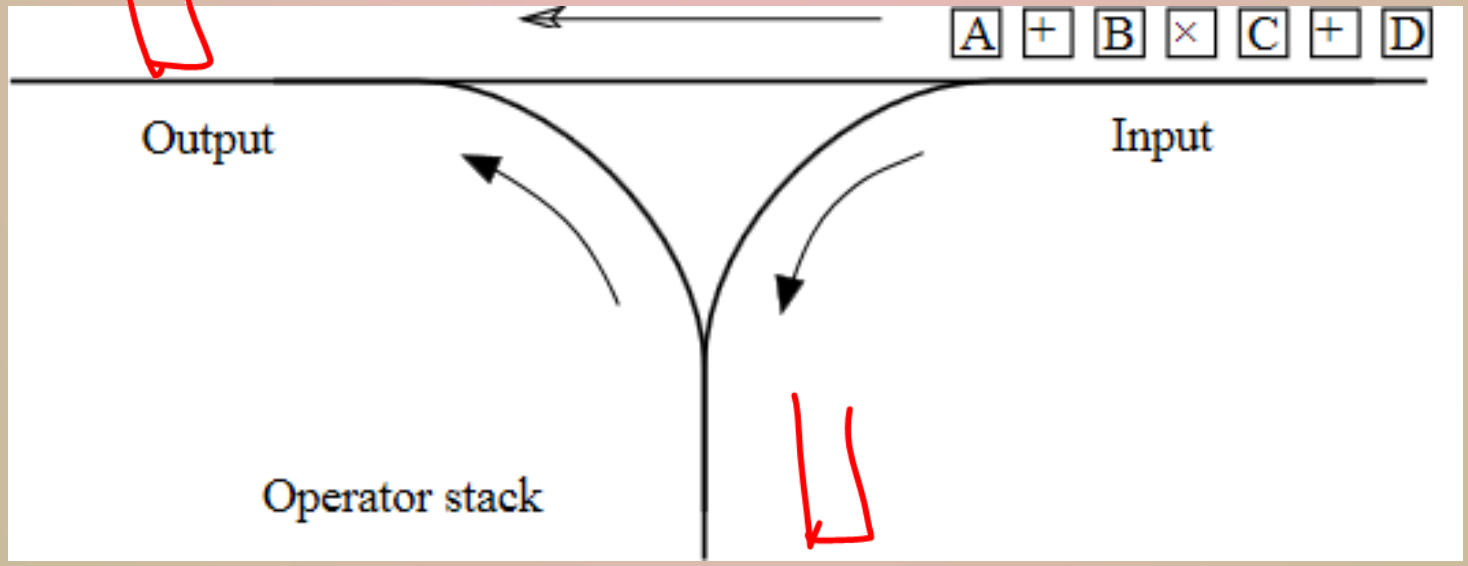
- اعداد
- متغیرها
- عملگرهای دودویی میانوند (+ * - / ^)
- پرانتزهای باز و بسته
- توابع (sin cos tan atan)
- عملگر یگانی منفی (-)

ایده اصلی

- از آنجا که در نمایش پسوندی عملگرها پس از عملوندها می‌آیند، عملگرها را در پشته ذخیره کنیم و در زمان مناسب در نمایش پسوندی درج کنیم.

ساختار کلی الگوریتم دایجکسترا

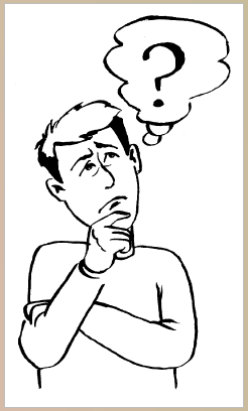
- دو پشته داریم:
  - پشته عملگرها
  - پشته خروجی (که در آن نمایش پسوندی را تولید می کنیم)



حالت هایی که در الگوریتم حفظ می کنیم

- اگر در حال مشاهده عنصر k ام نشانه های ورودی هستیم، آنگاه عبارت پسوندی موجود در پشته خروجی در کامل ترین شکل خود پیش از دیدن نشانه k ام است.

عمل صحیح هنگام مشاهده عدد یا متغیر

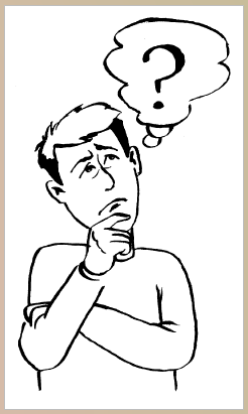


عمل صحیح هنگام مشاهده عدد یا متغیر

- عدد یا متغیر می تواند بعد از یکی از علایم زیر بیاید
  - در ابتدای عبارت
  - پس از پرانتز باز
  - پس از عملگر های دوگانی
  - پس از نام تابع
- در همه موارد فوق ابتدا باید عدد/متغیر در پشته خروجی قرار گیرد.

عمل صحیح هنگام مشاهده نام تابع

RPN (sin 2π) = 2π sin

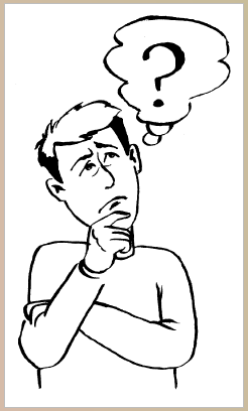


عمل صحیح هنگام مشاهده نام تابع

- وقتی نام یک تابع را می بینیم، عملوند آن هنوز مشاهده نشده است. بنابراین باید نام تابع را در پشته ذخیره کنیم تا در مراحل بعدی پس از قرار گرفتن عملوندها در پشته خروجی، نام تابع نیز در پشته خروجی درج شود.

عمل صحیح هنگام مشاهده عملگر یکانی منفی

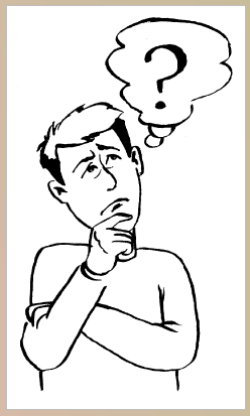
– 2



عمل صحیح هنگام مشاهده عملگر یکانی منفی

- اولا از آنجا که هم علامت تفریق و هم علامت منفی کردن با "-" نشان داده می شوند، برای تمیز این دو علامت از یکدیگر باید به علایم اطراف توجه کنیم.
  - اگر "-" اولین علامت، بلافاصله پس از پرانتز و یا اولین علامت داخل تابع باشد به معنای عملگر منفی کننده است.
  - در غیر این صورت عملگر دودویی تفریق است.
- در هر حال از آنجا که در نمایش RPN عملگر پس از عملوند می آید، باید در پشته ذخیره شود. عملگر منفی را با علامت دیگری (مثلا ?) در پشته عملگرها ذخیره می کنیم.

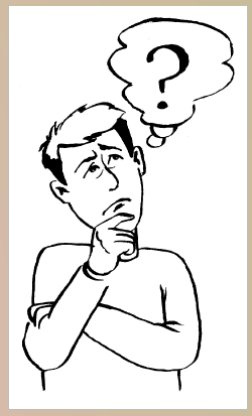
عمل صحیح هنگام مشاهده پرانتز چپ



عمل صحیح هنگام مشاهده پرانتز چپ

- پرانتز چپ را در پشته ذخیره می کنیم تا هنگام مشاهده پرانتز راست بدانیم که محدوده پرانتز تا کجا بوده است.

عمل صحیح هنگام مشاهده عملگرهای دودویی



+

x

−

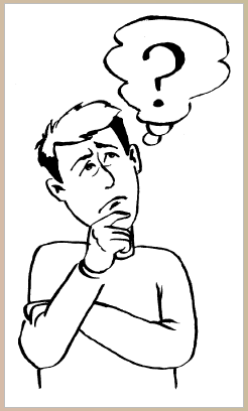
/

^

عمل صحیح هنگام مشاهده عملگرهای دودویی

- فرض کنیم عملگر دودویی o مشاهده شده است.
  - فرض کنیم عملگر p در بالای پشته است
  - اگر تقدم p از o بالاتر است یا (تقدم p و o مساوی است و o انجمنی از چپ است) p را از پشته عملگرها حذف کن و به پشته خروجی اضافه کن.
  - عمل فوق را تا زمانی که عملگر بالای پشته شرایط فوق را دارد تکرار کن
  - در نهایت عملگر o را درون پشته عملگرها قرار بده
- **توجه**: فرض می کنیم که توابع نیز عملگر هستند و اولویت آنها از عملگرهای دوگانی بالاتر است.

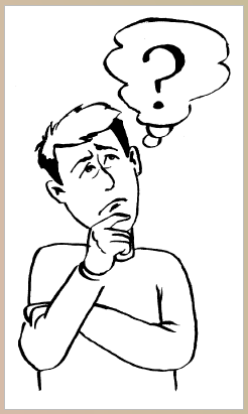
عمل صحیح هنگام مشاهده پرانتز بسته



عمل صحیح هنگام مشاهده پرانتز بسته

- تا زمانی که به پرانتز باز نرسیده ایم، عملگرهای بالای پشته عملگرها را برمی داریم و در پشته خروجی می ریزیم.
- پرانتز باز را هم از پشته عملگرها حذف می کنیم و دور می اندازیم.
- اگر عنصر بالای پشته عملگرها یک تابع است، آن را حذف کرده و در پشته خروجی می ریزیم. (البته به نظر می رسد که اگر اولویت توابع را نسبت به دیگر عملگرها تعیین کنیم, نیازی به این کار نیست)

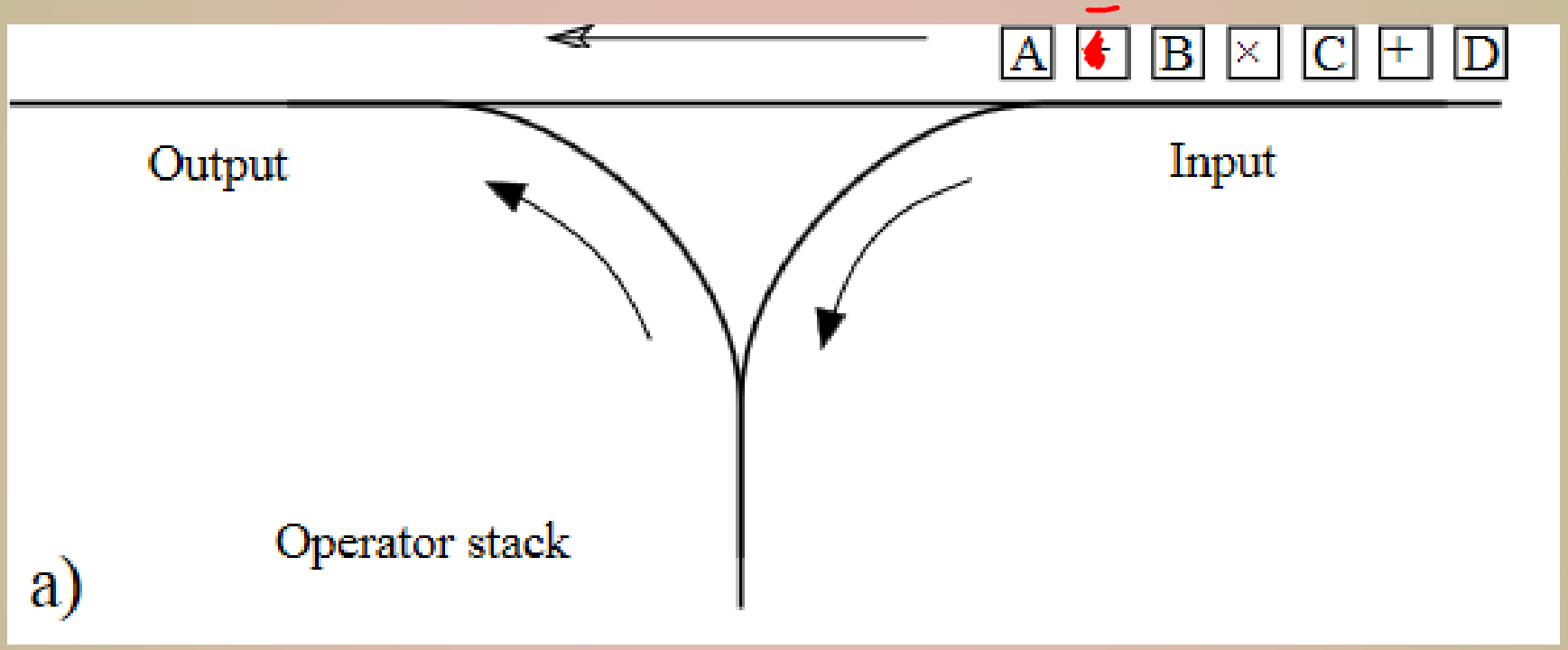
عمل صحیح پس از خواندن عبارت ورودی



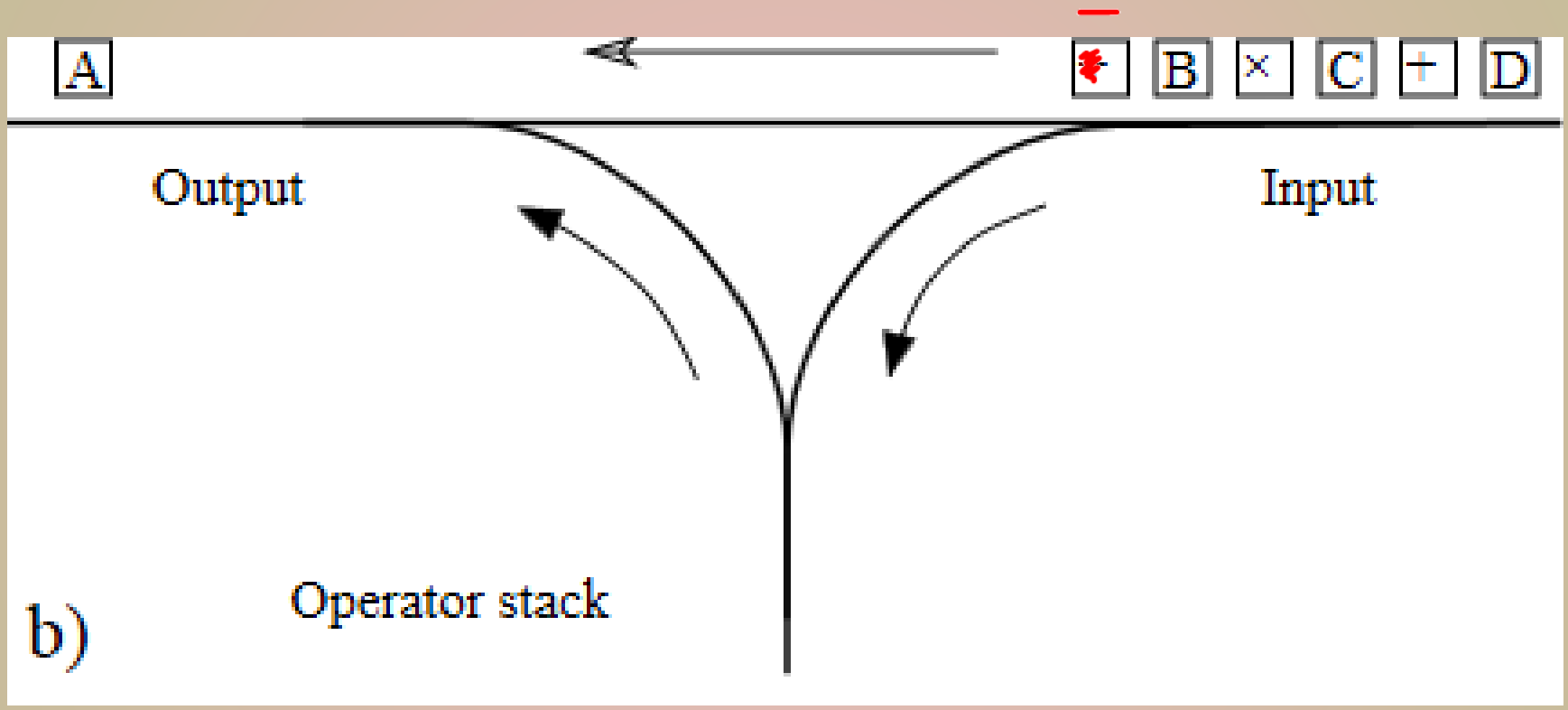
عمل صحیح پس از خواندن عبارت ورودی

- تمام علایم موجود در پشته عملگرها را در پشته خروجی قرار می دهیم.

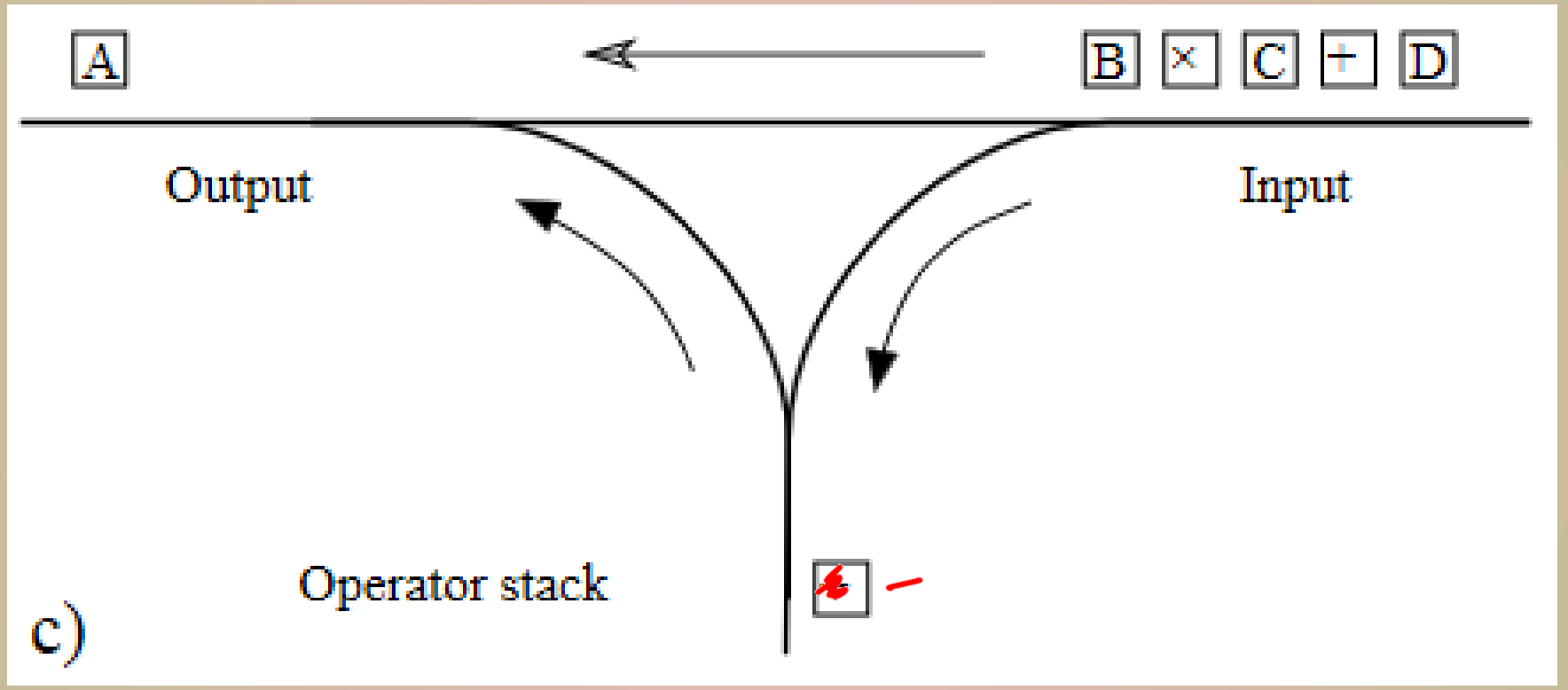
مثال



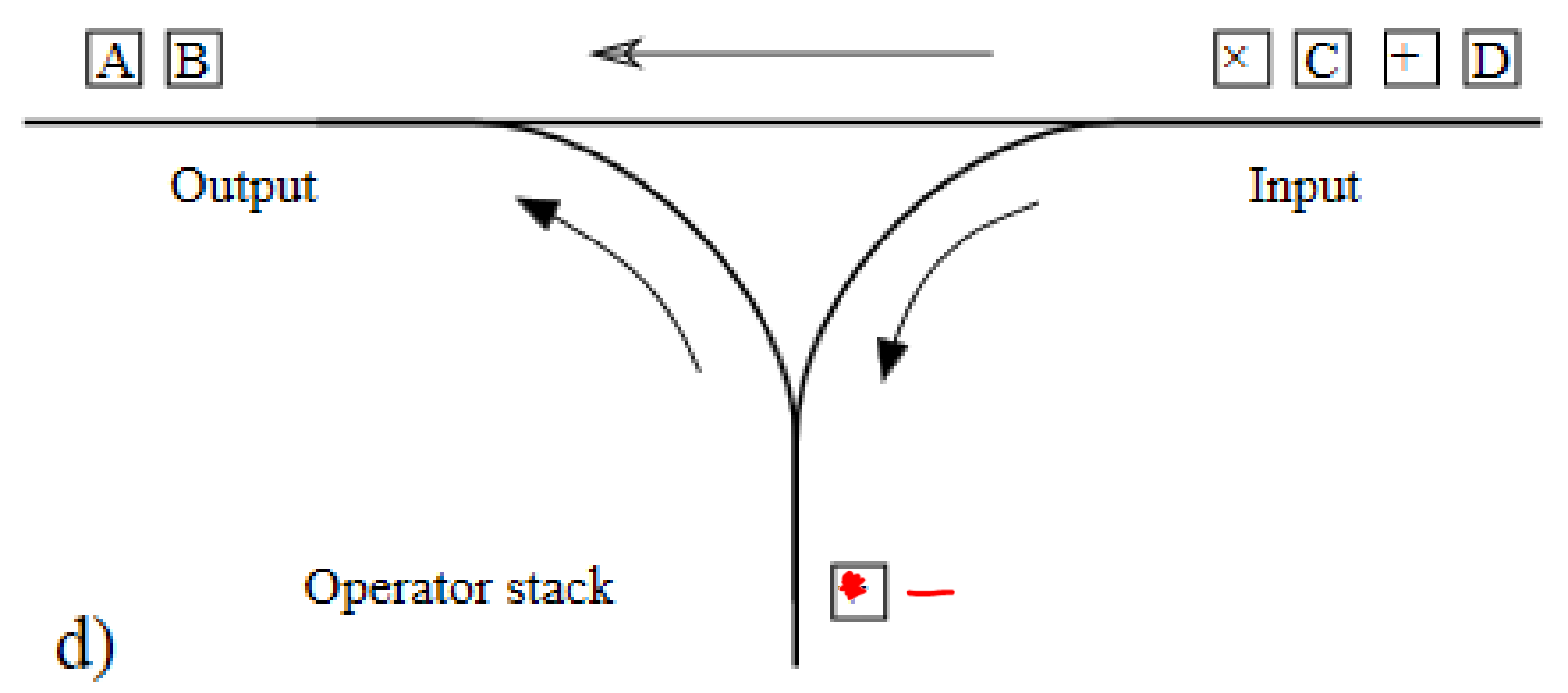
مثال



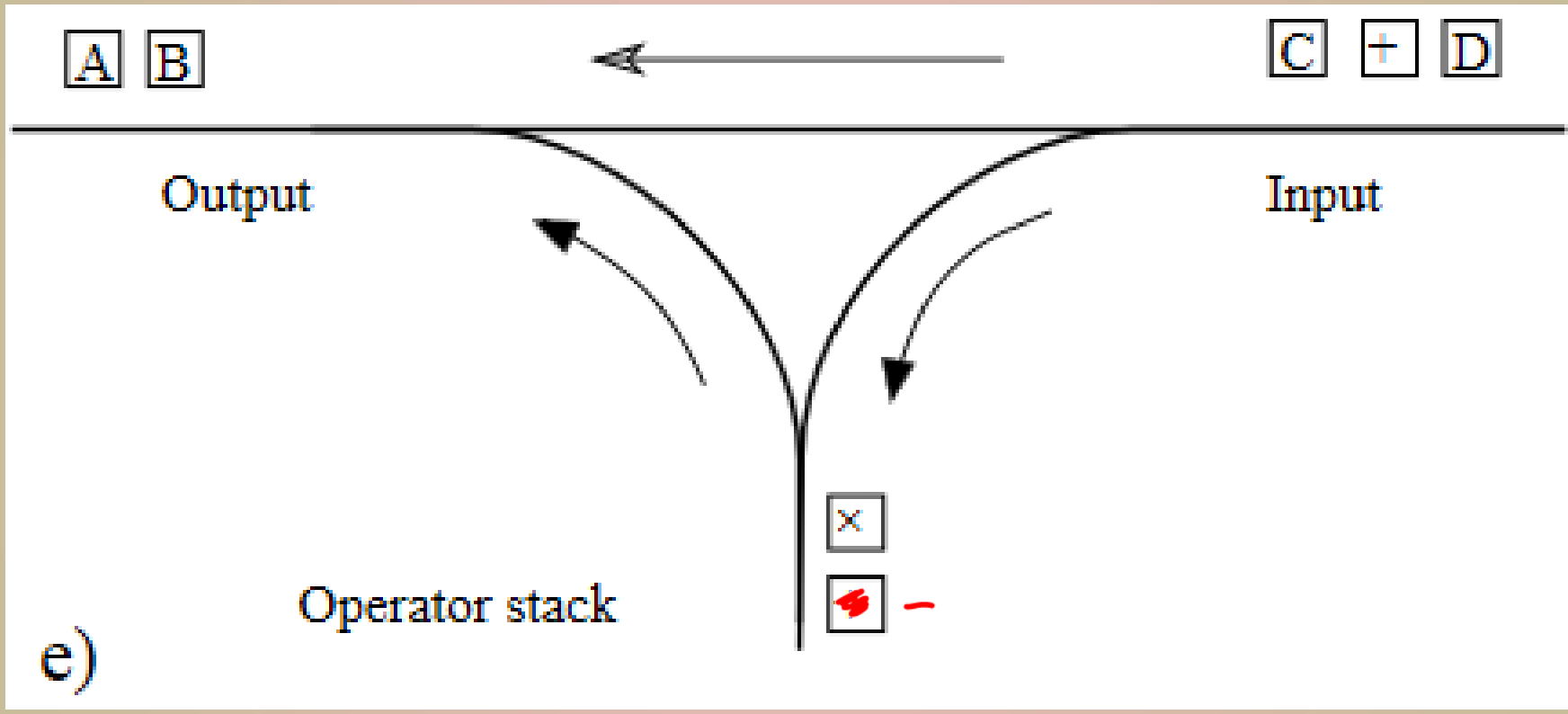
مثال



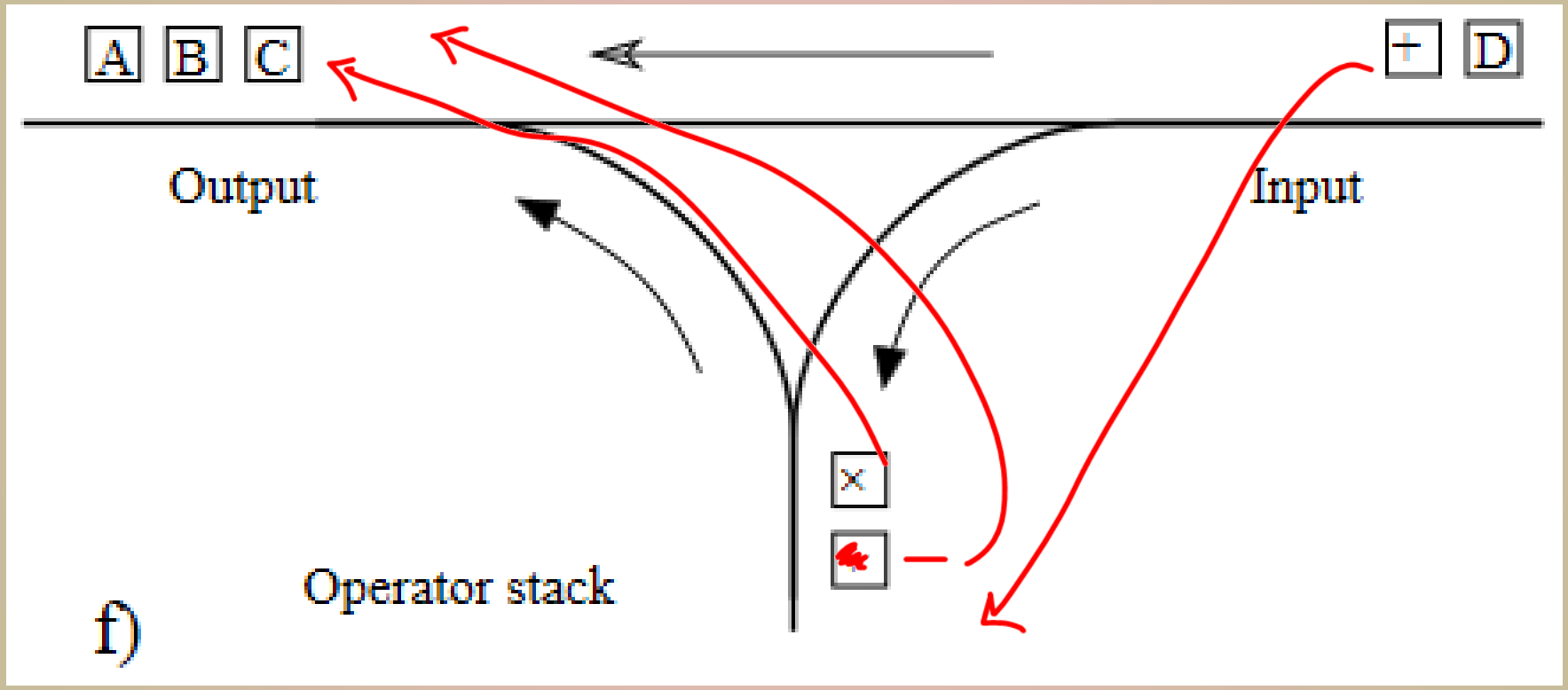
مثال



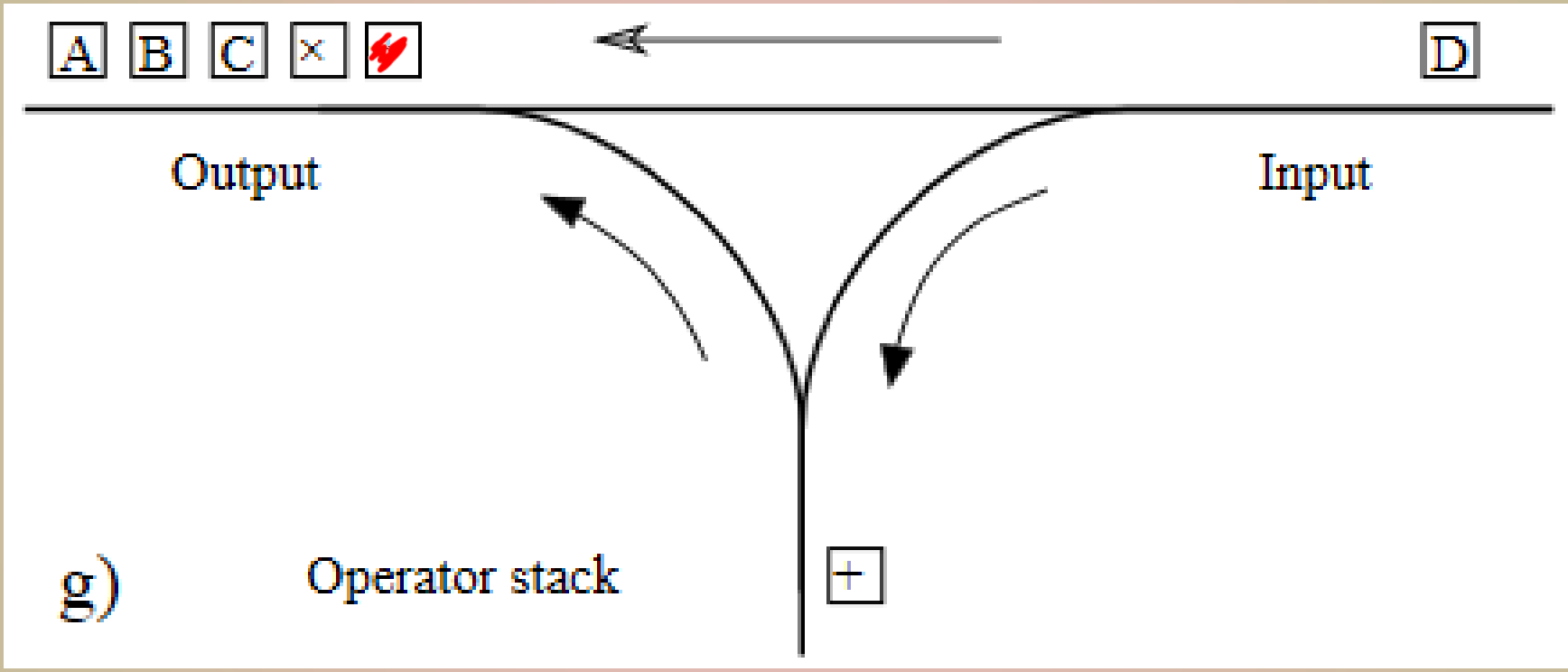
مثال



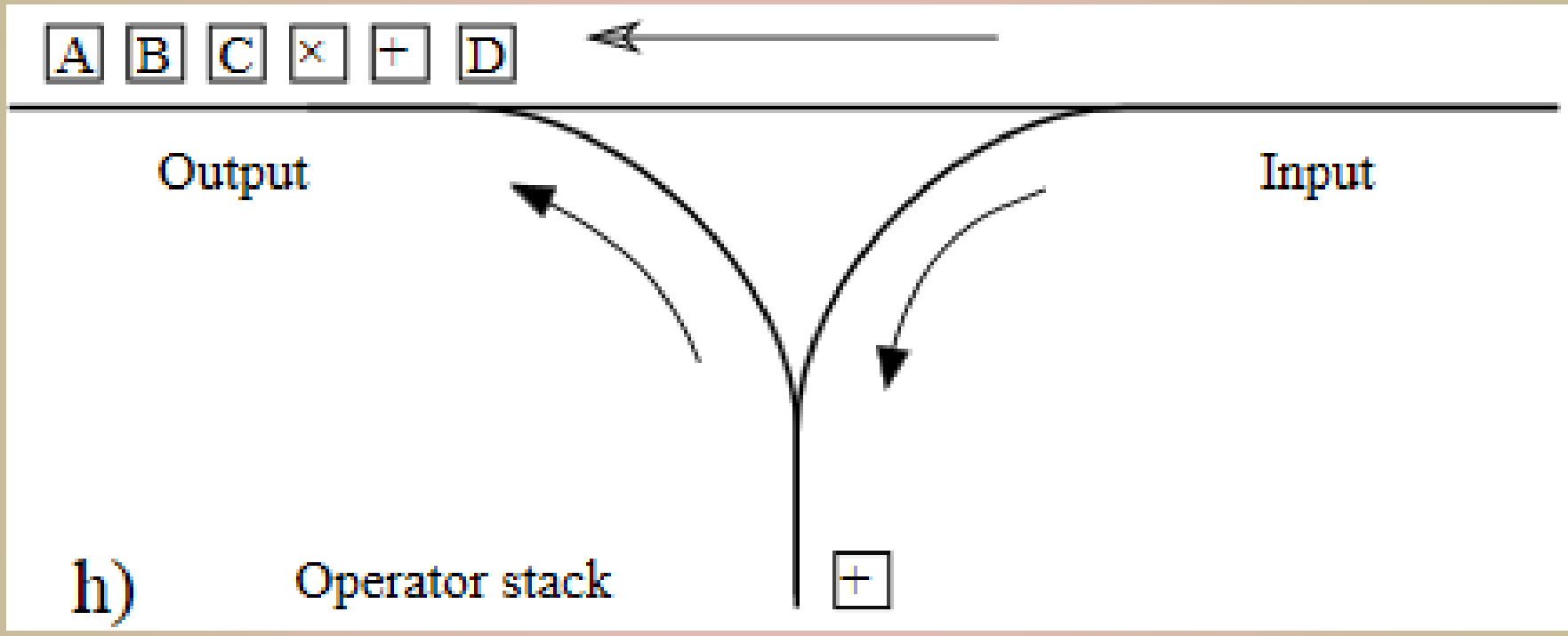
مثال



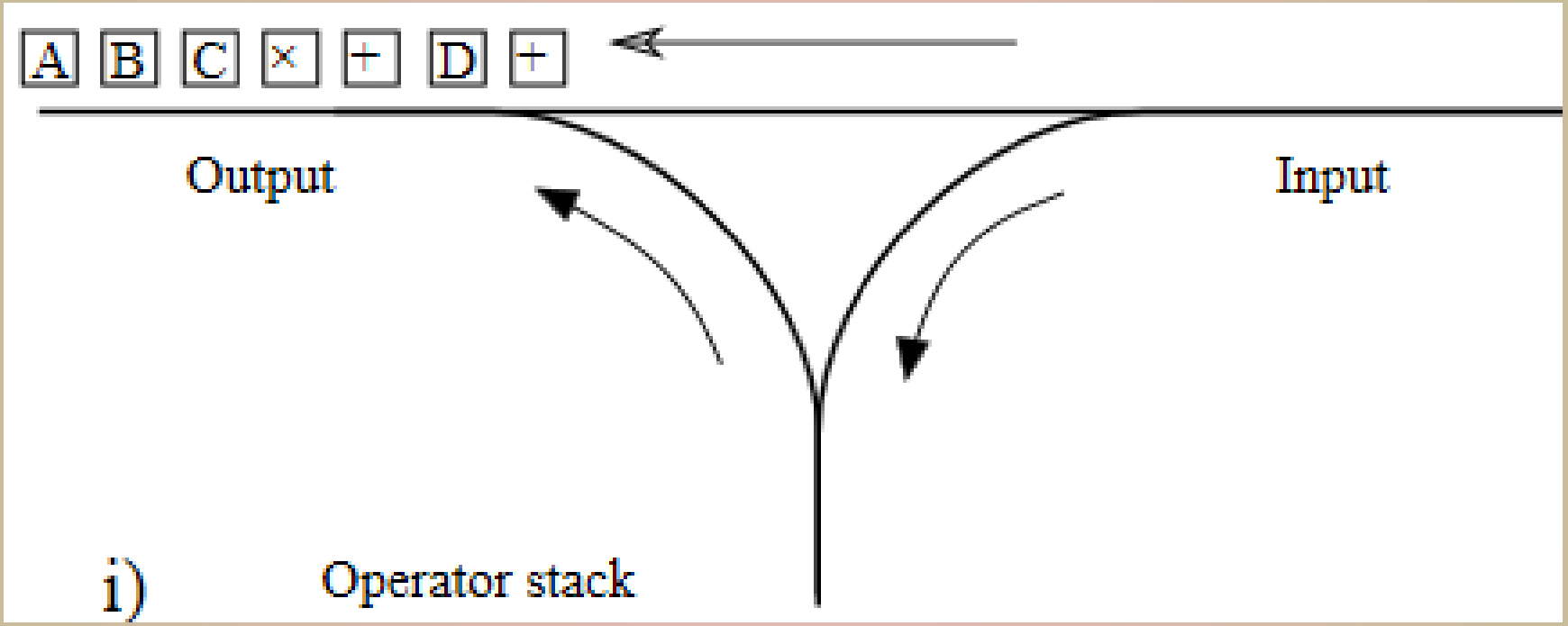
مثال



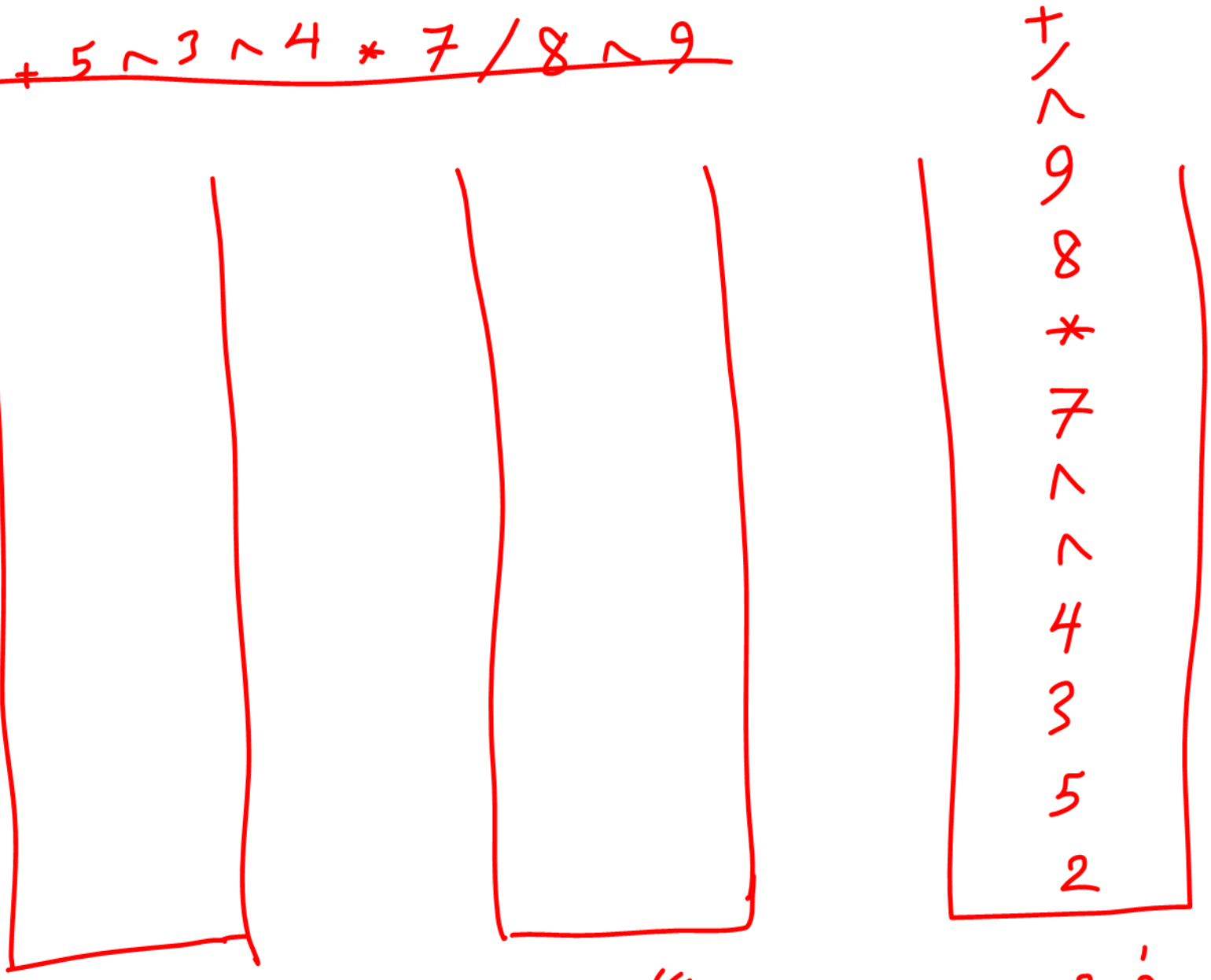
مثال



مثال



2 + 5 ^ 3 ^ 4 * 7 / 8 ^ 9



2 5 3 4 ^ ^ 7 * 8 9 ^ / +

عسرا

2 5 3 4 ^ ^ 7 * 8 9 ^ / +

~~2 + 5 ^ 3 ^ 4 * 7 / 8 ^ 9~~

2 + 5 ^ 3 ^ 4 * 7 / 8 ^ 9 ✓

2 + ([5 ^ (3 ^ 4)] * 7) / (8 ^ 9)

2 + 5 ^ 3 ^ 4 * 7 / 8 ^ 9

مراجع

- اسلایدهای درس ساختمان داده های آقای Sartaj Sahni
- http://en.wikipedia.org/wiki/Shunting-yard_algorithm
- کتاب و اسلایدهای درس الگوریتم آقای Robert Sedgewick