ساختمان داده ها

دانشگاه صنعتی نوشیروانی بابل

دکتر حسام عمران پور

طراحان اسلاید:
زهرا ریحانیان و دانیال علیزاده

حل تمرین :
علی باقری

لینک کانال تلگرام اطلاع رسانی و حل تمرین:
t.me/ds_nit_4011

فصل ششم

پشته

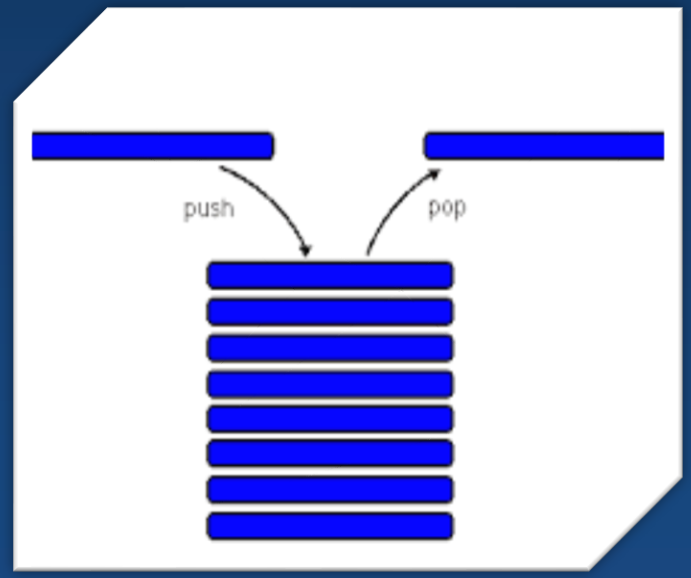
فهرست

نوع داده ی انتزاعی پشته

نوع داده ی انتزاعی پشته

پشته، مجموعه ای پویا از داده ها است که هنگام حذف عنصر از این مجموعه، آخرین عنصر اضافه شده به مجموعه از آن حذف می شود.

Last In First Out (LIFO)

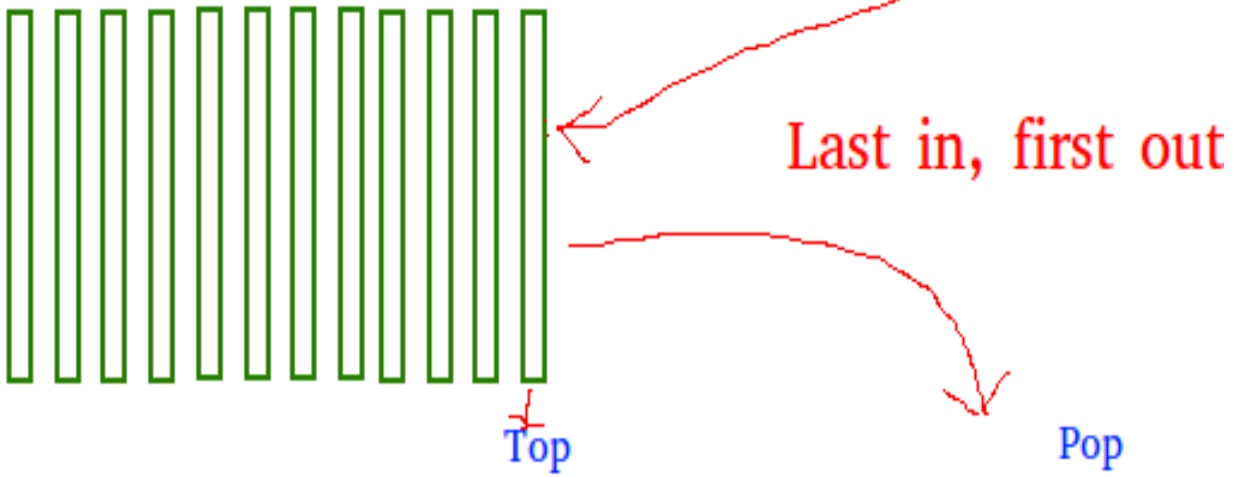


دسترسی به عناصر در پشته

- برای دسترسی به یک عنصر، ابتدا باید عناصری را که پس از آن به پشته وارد شده اند، از پشته خارج کرد.
- عنصر وارد شده به پشته در بالا داده های دیگر قرار گرفته و عنصر حذف شده از بالاترین عنصر است.
- به بالاترین عنصر پشته سر پشته(Top) گفته می شود.

Stack

Insertion and Deletion happen on same end



توابع ADT پشته

Create (ref S : Stack)	ایجاد پشته جدید :
Empty (S : Stack) : Boolean	بررسی خالی بودن پشته :
MakeNull (ref S : Stack)	خالی کردن پشته :
Push (ref S : Stack; x : ElementType)	افزودن عنصر به پشته :
Pop (ref S : Stack) : ElementType	خارج کردن یک عنصر از پشته:

? در یک صف به نام Q به ترتیب از راست، اعداد ۱ و ۲ و ۳ وارد شده اند. پشته ای به نام S نیز وجود دارد که با عمل PUSH یک عنصر را از صف Q خارج کرده و وارد پشته میکند و هنگام POP بالاترین عنصر خود را حذف و مقدار آن را در خروجی چاپ میکند. اگر حتما هر عنصر دقیقا یکبار PUSH شده و یکبار POP شود، تمام حالت های ممکن برای خروجی به چه صورت خواهد بود؟

✔ هر دنباله از دستورات، رشته ی مقابلش را به صورت زیر تولید میکند (از چپ به راست) :

PUSH, POP, PUSH, POP, PUSH, POP ------> 1, 2, 3
PUSH, POP, PUSH, PUSH, POP, POP ------> 1, 3, 2
PUSH, PUSH, POP, PUSH, POP, POP ------> 2, 3, 1
PUSH, PUSH, POP, POP, PUSH, POP ------> 2, 1, 3
PUSH, PUSH, PUSH, POP, POP, POP ------> 3, 2, 1

پس تعداد حالت های ممکن برای اعداد خروجی در این مثال برابر ۵ است.

پیاده سازی پشته

- به طور کلی پشته را می توان لیستی در نظر گرفت که عمل درج و حذف تنها از یـک سمت آن انجام می شود .
- این که درج و حذف عناصر از ابتدا انجام شود یا از انتها بستگی بـه پیاده سازی لیست استفاده شده داشته و از سمتی انجام می شود که مرتبه ی زمانی کوچک تری دارد.

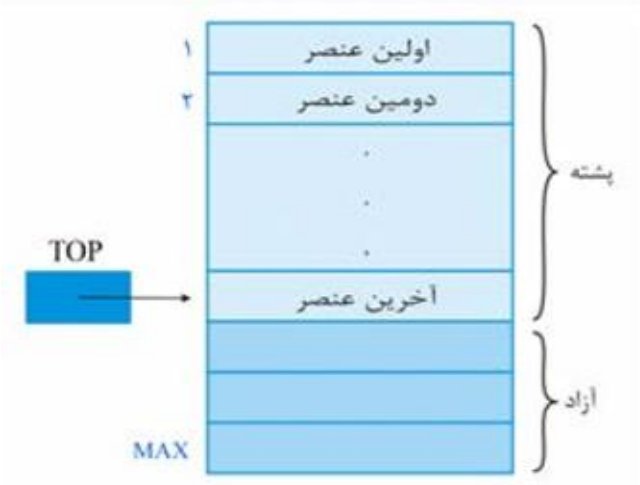
پیاده سازی آرایه ای پشته

پیاده سازی پشته با استفاده از لیست پیوندی

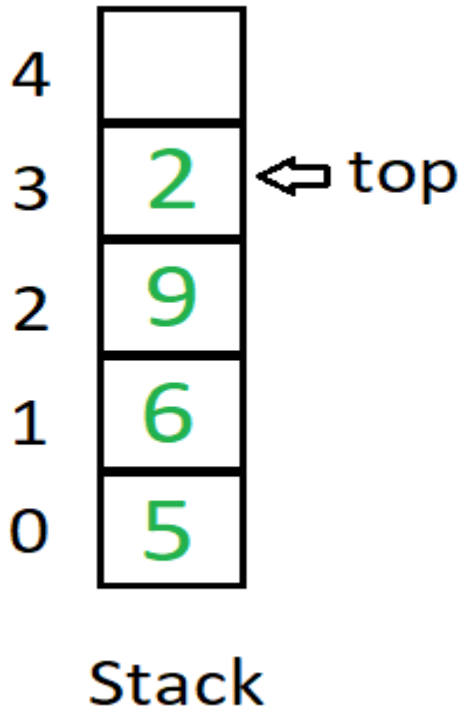
پیاده سازی آرایه ای پشته

ساختمان داده پشته با استفاده از آرایه

```
1 Type STACK = Record
2 {
3                Elements : ElementType [MAX]
4                Top : integer;
5 }
```



- برای پیاده سازی پشته بـا اسـتفاده از آرایـه، یـک آرایـه از نـوع ElementType در نظـر گرفتـه می شود .
- عناصر پشته، به ترتیب، در خانه های پشت سر هم آرایه قرار می گیرند.
- عنصر i ام در خانه ی i ام آرایه می نشیند.
- یک اشاره گر اندیسی به نـام Top نیـز وجود دارد که شماره ی خانه ی آخرین عنصر اضافه شده به پشته را نگاه می دارد.



پیاده سازی توابع پشته با استفاده از آرایه

ایجاد پشته جدید :

```
Create ( ref S : Stack )
{
        S.TOP = 0
}
```

بررسی خالی بودن پشته :

```
Empty ( S : Stack ) : Boolean
{
    if S.TOP == 0 then
            return TRUE

        return FALSE
}
```

بررسی پر بودن پشته : برای جلوگیری از خطای سرریزی

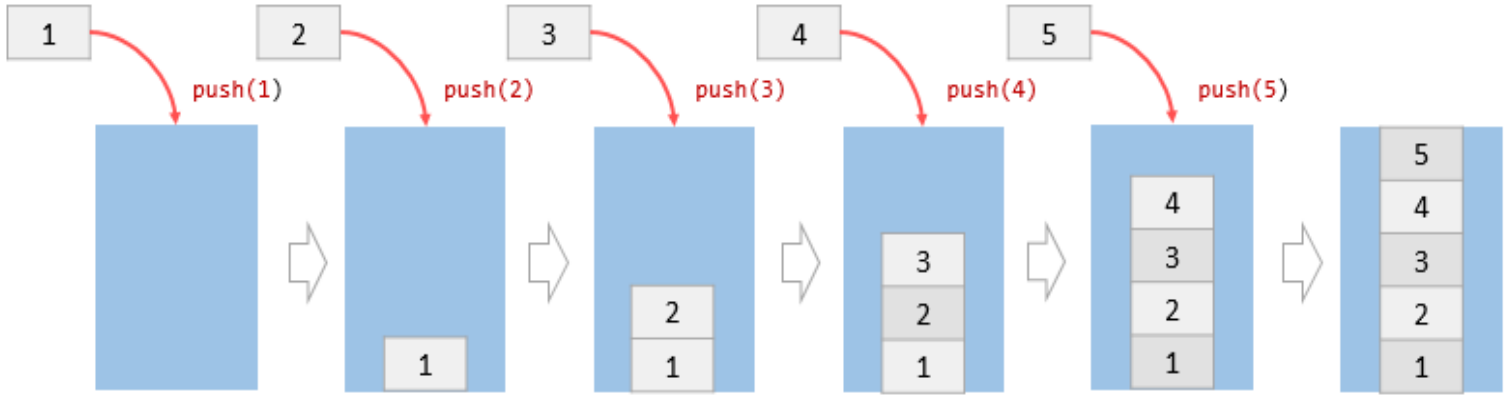
```
Full (S : Stack ) : Boolean
{
        if S.TOP == MAX then
                return TRUE
        return FALSE
}
```

خالی کردن پشته :

```
MakeNull (ref S : Stack) {
                S.TOP = 0
}
```

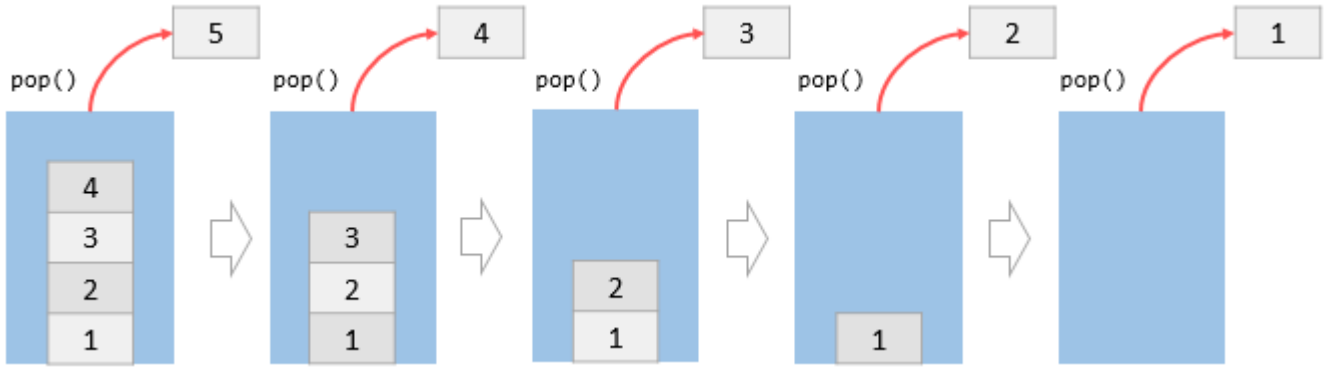
درج یک عنصر جدید:

```
Push ( ref S : Stack; x : ElementType )
{
        if S.TOP == Max then
                Error ( “Overflow” );
        else {
                S.TOP ++;
                S.Elements[S.TOP] = x;
        }
}
```



حذف و بازیابی آخرین عنصر:

```
Pop ( ref S : Stack ) : ElementType
{
        if S.TOP == 0 then
                Error ( “Underflow” );
        else {
                S.TOP -- ;
                return S.Elements[S.TOP + 1];
        }
}
```



پیاده سازی پشته با استفاده از لیست پیوندی

پیاده سازی پشته با استفاده از لیست پیوندی

- یکی از مشکلات اصلی پشته های آرایه ای، محدودیت تعداد عناصرآن ها است که این محـدودیت باعث اشکال در بسیاری از کاربردها می شود.

- همچنین در نظر گرفتن یـک مقـدار بـزرگ بـرای اندازه ی آرایه در زمان هایی که تعداد عناصر پشته کم است، موجـب مصـرف بـیش از حـد حافظه می شود.

- در این موارد می توان از ایده ی لیست های پیوندی بـرای وابسـته کـردن فضـای اشـغال شده به تعداد عناصر پشته استفاده کرد. همچنین با این کار، مشکل محـدودیت حـداکثر تعـداد عناصر برای پشته نیز رفع می گردد.

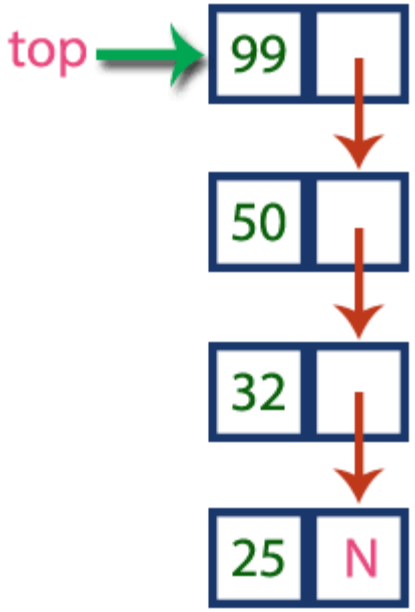
عمل Push در پشته همان عمل درج در ابتدای لیسـت و عمـل Pop در پشـته همـان عمـل حذف عنصر از ابتدای لیست خواهد بود.

ساختمان داده پشته با لیست پیوندی

```
Type cellType = Record
{
        Data : ElementType
        Next : Position
}

Type Stack = ^cellType

Type Position = ^cellType
```



پیاده سازی توابع پشته با استفاده از لیست پیوندی

ایجاد پشته جدید :

```
Create (ref S : Stack)
{
        S = Null
}
```

خالی کردن پشته :

```
MakeNull (ref S : Stack)
{
        P : Position
        while (S != Null)
        {
                P = S
                S = S^.Next
                dispose(p)
        }
}
```

مرتبه زمانی

O(n)

بررسی خالی بودن پشته :

```
Empty (S : Stack) : Boolean
{
        if S == Null
                return True
        return False
}
```

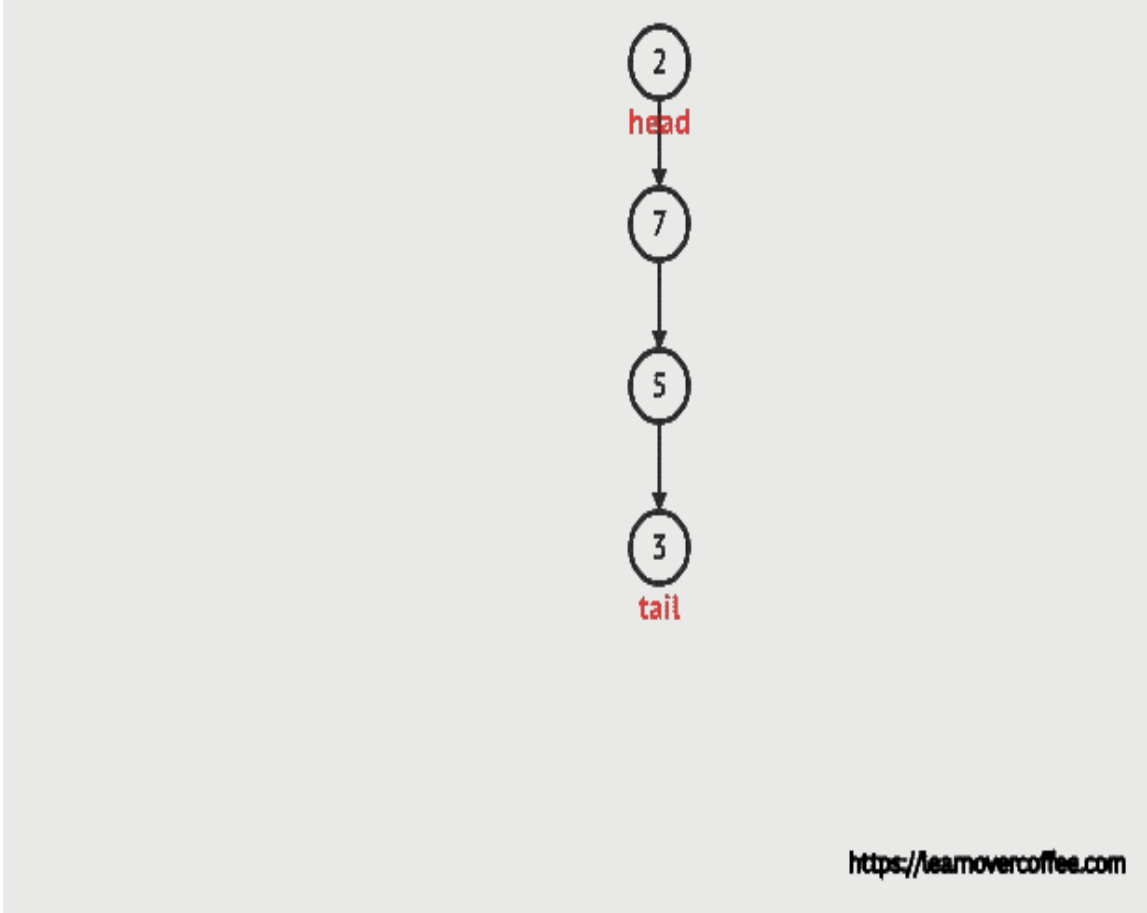
درج یک عنصر جدید:

```
Push (ref S : Stack ; x : ElementType)
{
        P : cellType
        new(P)
        P^.Data = x
        P^.Next = S
        S = P
}
```

مرتبه زمانی

O(1)

Push



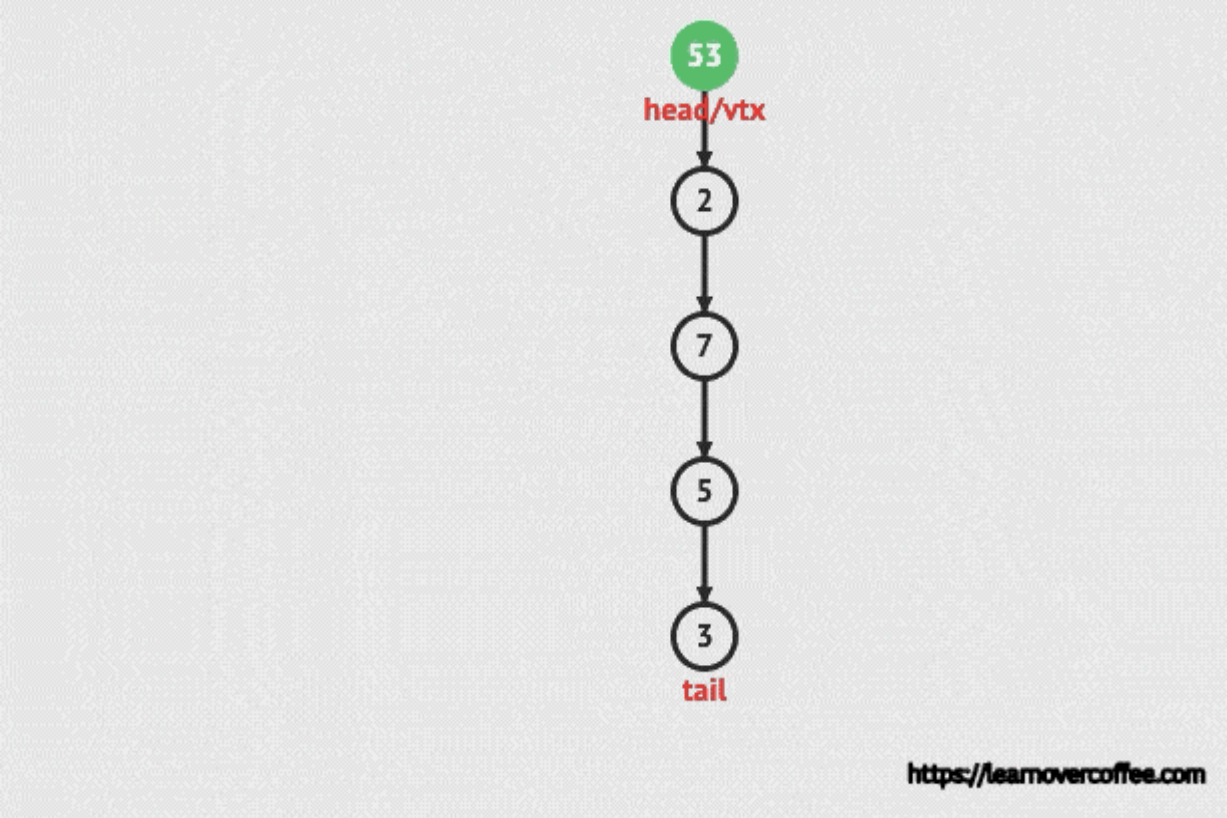
حذف یک عنصر :

```
Pop (ref S : Stack) : ElementType
{
        if S == Null
                return Error
        else {
                P : Position
                P = S
                S = S^.Next
                x : ElementType
                x = P^.Data
                dispose(P)
                return x
        }
}
```

مرتبه زمانی

O(1)

Pop



پیاده سازی یک صف با دو پشته

```
Type Queue = Record
{
        s1, s2 : Stack;
}
```

```
Create (ref Q; Queue)
{
        s1 = Null;
        s2 = Null;
}
```

```
Enqueue ( ref Q: Queue, x : ElementType )
{
        if Empty ( s1 )
                push ( s1 , x );
        else
        {
                while s1 != Null
                        push ( s2 , pop ( s1 ) ) ;

                push ( s1 , x ) ;

                while s2 != Null
                        push ( s1 , pop ( s2 ) ) ;

        }
}
```

مرتبه زمانی

O(n)

```
Dequeue ( ref Q: Queue ) : ElementType
{
        if Empty ( s1 )
                Error ;
        return pop ( s1);
}
```

مرتبه زمانی

O(1)

تبدیل زیر برنامه های بازگشتی به غیر بازگشتی به کمک پشته

تبدیل توابـع بازگشتی به غیربازگشتی

- ✓ یکی از کاربردهای مهـم پشـته در پیاده سازی توابـع بازگشتی در زبان برنامه های زبان نویسی است .
- ✓ اغلب زبان های برنامه نویسی که اجازه ی نوشتن توابع بازگشتی را به برنامه نویس می دهند، از ساختمان داده ی پشته برای نگهداری مقدار متغیرهای هر یک از توابع فعال برنامه استفاده می کنند .
- ✓ استفاده از بازگشت در توابع باعث ساده تر شدن ساختار بسیاری از برنامه ها می شود .
- ✓ با این حال در برخی از زبان های برنامه نویسی ممکن است امکان نوشتن توابع بازگشتی وجـود نداشته باشد و یا هزینه ی زمانی فراخوانی تابع به قدری بالا باشد که سرعت مورد نیاز برنامه تأمین نشود .
- ✓ بنابراین ممکن است نیاز به تبدیل توابع بازگشـتی نوشـته شـده بـه توابـع غیربازگشـتی وجود داشته باشد.

مراحل فراخوانی زیربرنامه ها

مراحل این فراخوانی در کامپیوتر:

1. ذخیره ی مقدار کنونی متغیرهای تابع فراخوانی کننده (F1).
2. ذخیره ی آدرس بازگشت.
3. تخصیص مقدار به متغیرهای ورودی (پارامتر) تابع فراخوانی شده (F2).
4. پرش به خط ابتدای تابع فراخوانی شده (F2).

پس از به پایان رسیدن اجرای تابع فراخوانی شده (F2) نیز مراحل زیر انجام می شود :

1. بازیابی آدرس بازگشت.
2. بازیابی مقادیر ذخیره شده ی تابع فراخوانی کننده (F1).
3. پرش به آدرس بازگشت برای ادامه ی فراخوانی اجرای تابع فراخوانی کننده (F1).

حذف انتهایی های بازگشت

در حالت خاصی از فراخوانی بازگشتی، اگر دستور مربوط به فراخوانی بازگشتی آخرین دستور تابع باشد، به آن بازگشت انتهایی یا دم بازگشتی گفته می‌شود.

ویژگی این نوع فراخوانی بازگشتی این است که نیازی به ذخیره ی وضعیت کنونی تابع فراخوانی کننده نیست، زیرا هنگام بازگشت به این تابع، هیچ دستوری پس از فراخوانی بازگشتی وجود نداشته و اجرای این تابع به پایان می رسد.

بنابراین، برای حذف یک فراخوانی با دم بازگشتی تنها کافی است که متغیرهای ورودی تابع فراخوانی شونده مقداردهی شده و به ابتدای دستورات آن پرش شود.

حذف فراخوانی های بازگشتی در حالت کلی

1. تعریف یک پشته برای ذخیره سازی مقدار کنونی هنگام فراخوانی بازگشتی.
2. مشخص کردن خط شروع تابع برای پرش هنگام فراخوانی جدید.
3. ذخیره سازی مقادیر کنونی متغیرهای محلی در پشته، تخصیص مقدار جدید پارامترهـای تابع و پرش به خط اول تابع به جای فراخوانی بازگشتی.
4. مشخص کردن خط دستور بعد از فراخوانی بازگشتی، برای ادامـه ی اجـرای تـابع هنگـام بازگشت از فراخوانی. سپس قرار دادن پرش به این خط در تمام نقاط پایانی تابع(یعنـی در خط آخر تابع و به جای دستورات بازگشت).
5. بیرون کشیدن مقدار متغیرهای محلی از پشـته و تخصـیص مجـدد مقـدار بـه آن‌هـا . در صورتی که فراخوانی پشته خالی باشد، به این معنی است که فراخوانی‌های بازگشـتی انجـام شـده توسط تابع به پایان رسیده است .در این صورت تابع اصلی باید خاتمه یافتـه و بازگشت کند، بنابراین پیش از بیرون کشیدن مقدار متغیرهای محلی از پشته بایـد خـالی بـودن آن نیز بررسی شود.

? تابعی بازگشتی به صورت زیر برای چاپ هر رقم یک عدد صحیح مثبت در یک خط وجود دارد. این تابع بازگشتی را به یک تابع غیربازگشتی تبدیل کنید.

```
PrintDigits (n : integer)
{
        if n > 0 then
        {
                WriteLine( n % 10)
                PrintDigits( n / 10)
        }
}
```

- برای تبدیل این تابع به غیر بازگشتی، کافی است آدرس خط ابتدای تابع مشخص شده و به جای فراخوانی بازگشتی، متغیر n (ورودی تابع) مقداردهی شده و پرش به خط اول تابع انجام شود :

```
NPrintDigits (n : integer)
{
        1 :     if n > 0 then
                {
                        WriteLine( n % 10)
                        n = n / 10
                        GoTo 1
                }
}
```

نکته: در بسیاری از موارد میتوان پس از تبدیل توابع بازگشتی به غیر بازگشتی، دستورات GoTo موجود را حذف کرده و به برنامه شکل ساخت یافته داد. به عنوان نمونه در مثال قبلی در تابع NRPrintDigit میتوان دستور GoTo و برجسب را حذف کرده و به جای آن، شرط if را تبدیل به while کرد.

? تابعی بازگشتی به صورت زیر برای چاپ رقم های یک عدد صحیح مثبت به صورت معکوس وجود دارد، تابع غیر بازگشتی آن را بنویسید.

```
Reverse (n : integer)
{
        if n >= 10 then
                Reverse( n / 10)
        WriteLine(n % 10)
}
```

```
NRReverse ( n : integer )
{
        S : Stack
        Create(S)
 1:      if n >= 10 then
        {
                push(S,n)
                n = n / 10
                GoTo 1

99:             if Empty(S) then
                        return
                else
                        n = pop(S)
        }

        WriteLine(n % 10)
        GoTo 99
}
```

نکته: در مواردی که تابع بازگشتی مورد نظر برای تبدیل به غیر بازگشتی دارای مقدار خروجی است، از یک متغیر محلی کمکی برای نگهداری مقدار خروجی تابع استفاده می شود.
بنابراین، این متغیر باید در نقاط پایانی تابع (یعنی در خط آخر تابع و به جای دستورات بازگشت)، پیش از عمل پرش مقداردهی شده و از مقدار آن در محل فراخوانی بازگشتی استفاده شود.

? تابعی بازگشتی برای محاسبه فاکتوریل یک عدد به صورت زیر وجود دارد. تابع غیر بازگشتی آن را بنویسید.

```
Factorial (n : integer) : integer
{
        if n == 0 then
                return 1

        return n * factorial(n-1)
}
```

```
NRFactorial (n : integer) : integer
{
        S : Stack
        Res : integer
        Create(S)
 1:     if n == 0 then
        {
                Res = 1
                GoTo 99
        }
        Push(S, n)
        n = n – 1
        GoTo 1
99:     if Empty(S) then
                return Res
        n = pop(S)
        Res = n * Res
        GoTo 99
}
```

در خط ۵ باید دقت کرد که در این خط سه دستور نهفته است. ابتدا فراخوانی بازگشتی انجام شده، سپس نتیجه ی آن در مقدار n ضرب میشود و در پایان نتیجه ی حاصل به عنوان خروجی تابع بازگردانده میشود. بنابراین پس از انجام مراحل تبدیل این تابع به غیر بازگشتی، تابع به دست آمده به این صورت خواهد بود :

? تابع بازگشتی حل مسأله برج های هانوی به صورت زیر است. تابع غیر بازگشتی آن را بنویسید.

```
Hanoi ( n : integer; A, B, C : pivot)
{
        if n == 1 then
                MOVE( A → C )
        else {
                Hanoi (n-1, A, C, B)
                MOVE( A → C )
                Hanoi(n-1, B, A, C)
        }
}
```

پاسخ

برای تبدیل این تابع به غیر بازگشتی، ایتدا باید دو فراخوانی بازگشتی از یکدیگر متمایز شوند. به عنوان مثال؛ میتوان فراخوانی اول را با حرف A و فراخوانی دوم را با حرف B مشخص کرد. همچنین با توجه به این که فراخوانی بازگشتی دوم یک فراخوانی باگشتی انتهایی است، در این فراخوانی نیازی به ذخیره ی مقدار متغیرهای محلی نیست. در نتیجه هنگام بازگشت از فراخوانی بازگشتی، در صورتی که فراخوانی بازگشتی دوم باشد نیازی به بازیابی مقدار متغیرهای محلی نیست. البته میتوان بدون در نظرگرفتن دُم بازگشتی بودم فراخوانی دوم، با آن نیز مانند یک فراخوانی بازگشتی عادی برخورد کرد تا شکل تابع به دست آمده یک دست تر شود. با توجه به این توضیحات، شکل غیر بازگشتی تابع هانوی فوق، در صفحه بعد دیده می شود :

```
NRHanoi ( n : integer; A, B, C : pivot )
{
        S : Stack
        ret-adr : char
        Create(S)
 1:     if n == 1 then
                MOVE( A → C )
        else {
                push(S, n, A, B, C)
                push('A')
                n = n – 1, A = A, B = C, C = B
                GoTo 1
        }
99:     if Empty(S) then
                return
        ret-adr = pop(S)
        Case (ret-adr) of
        'A' :
                [n, A, B, C] = pop(S)
                MOVE( A → C )
                push(S, n, A, B, C)
                push('B')
                n = n – 1, A = B, B = A, C = C
                GoTo 1
        'B' :
                GoTo 99
}
```

نمایش عبارت‌های ریاضی

نمایش عبارت‌های ریاضی

در کاربردهای مختلف، عبارت های ریاضی معمولاً به سه شکل میان وندی ، پیشوندی و پسوندی نمایش داده می شوند. تفاوت این نمایش ها در محل قرار گرفتن عملگر نسبت به محل عملوند است .در این قسمت این سه نمایش مورد بررسی قرار خواهند گرفت.

نمایش میان وندی

- شیوه ی نمایش میان وندی از معمول ترین شیوه های نمایش عبارت های ریاضی در نوشتار است .
- در این شیوه ی نمایش، عملگر دوتایی بین دو عملوند قرار گرفته و عملگر یکتایی نیز پیش از عملوند می‌آید .
- عبارت های میان وندی را می توان به صورت بازگشتی با استفاده از گرامر زیر تولید کرد:

$$E \rightarrow \alpha E' \mid E' \beta E' \mid E'$$
$$E' \rightarrow \alpha \mid E' \beta E' \mid (E)$$

در این گرامر، E یک عبارت میان وندی است و α نمایانگر یک عملوند ساده است. همچنین α یک عملگر یکتایی و β عملگر دوتایی است.
گرامر فوق ابهام دارد و ممکن است برای یک عبارت میان وندی بیش از یک تفسیر وجود داشته باشد که در نتیجه ی آن، نتایج مختلفی برای عبارت به دست آید . بنابراین، برای رفع ابهام عبارات میان وندی، از پرانتزگذاری و تعیین اولویت برای عملگرها استفاده می شود.

تنها مزیت شیوه ی نمایش عبارت های میان وندی نسبت به شیوه های دیگر نمایش، آشنایی عمومی با این شیوه ی نمایش است.
ارزیابی عبارت های میان وندی (محاسبه ی نتیجه ی عبارت) به صورت مستقیم، نسبتا کند و دشوار بوده و از مرتبه ی پیچیدگی زمانی $O(n^2)$ است.

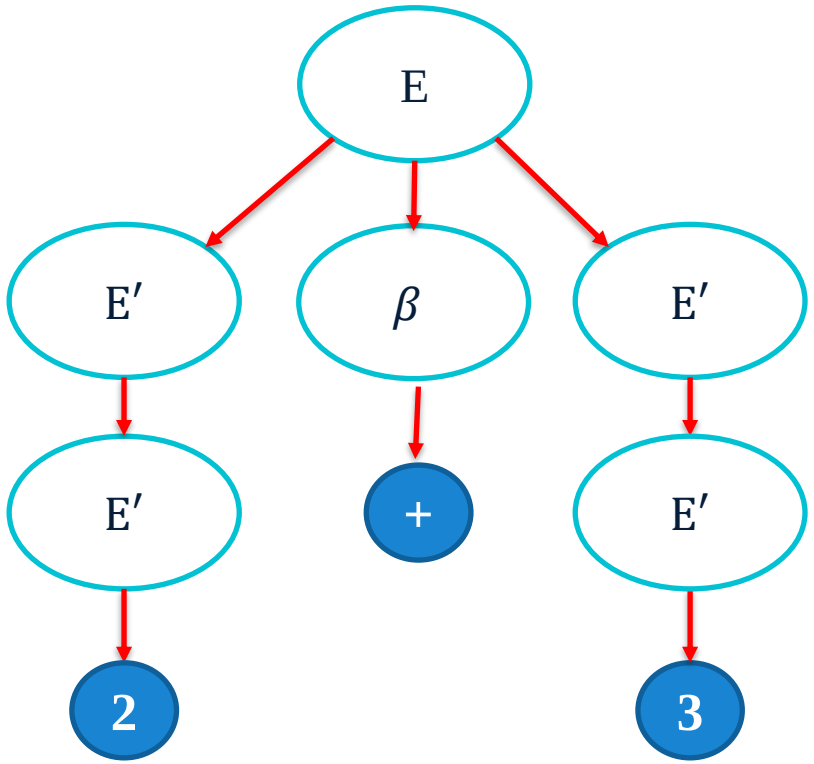
? عبارت a + b * c را تفسیر کنید.

1. جمع دو عدد a و b و سپس ضرب نتیجه ی آن در c
2. ضرب دو عدد b و c و سپس جمع نتیجه ی آن با a

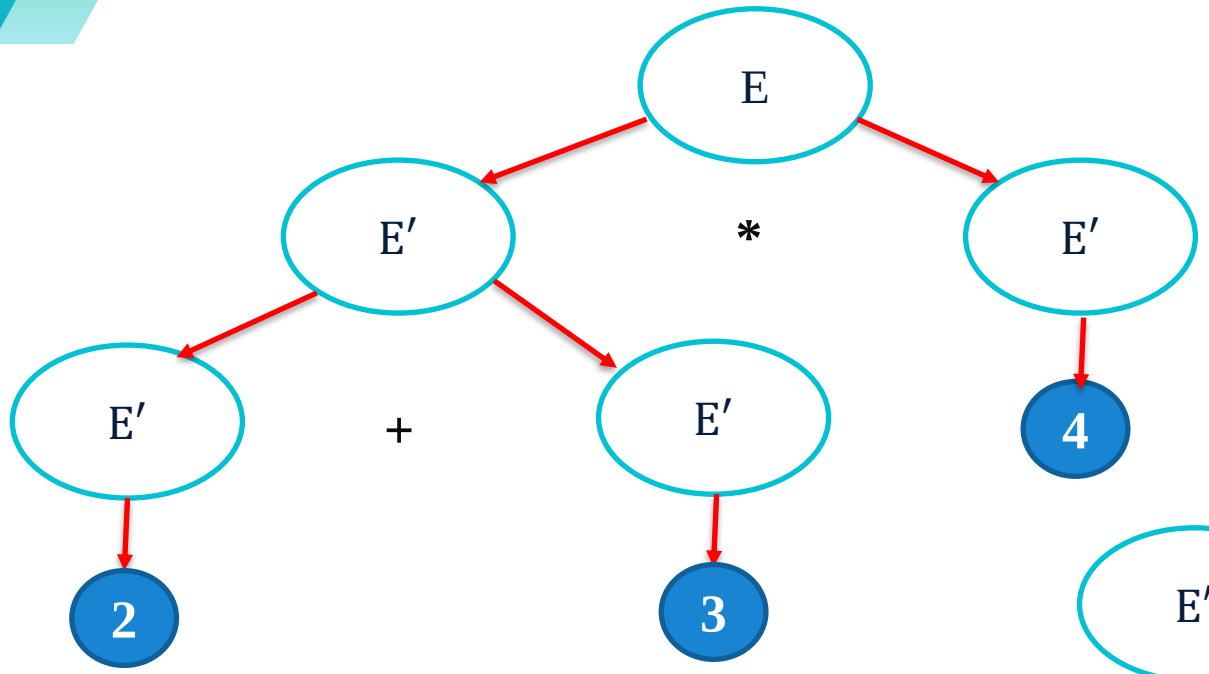
این دو تفسیر نتایج کاملا متفاوتی را بدست میدهند. برای رفع این ابهام این عبارت با استفاده از پرانتزگذاری میتوان آن را به صورت (a+b)*c برای حالت اول، یا a+(b*c) برای حالت دوم تبدیل کرد. همچنین با تعیین اولویت بالاتر برای عملگر ضرب نسبت به عملگر جمع، تفسیر عبارت فوق به صورت حالت دوم خواهد بود.

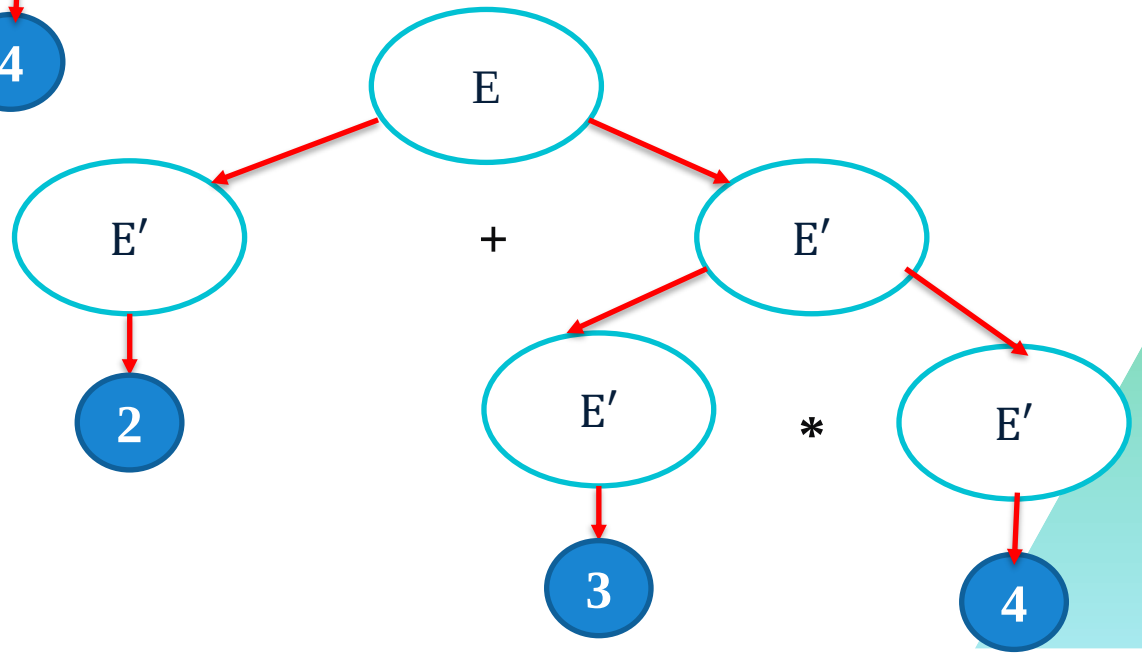
مثال : 2 + 3

$$\mathrm{E} \rightarrow \mathrm{E}'\ \beta\ \mathrm{E}' \rightarrow \mathrm{a}\ \beta\ E' \rightarrow 2\ \beta\ E' \rightarrow 2 + E' \rightarrow 2 + a \rightarrow 2 + 3$$



مثال : 2 + 4 * 3





نمایش پیشوندی

در شیوه ی نمایش پیشوندی عبارت های ریاضی، عملگرها دقیقاً پیش از عملوندهای خود قرار می گیرند.
به دلیل ابداع این شیوه ی نمایش عبارت ها توسط یک منطق دان لهستانی، این شکل به نمایش لهستانی نیز شهرت دارد.
عبارت های پیشوندی را به صورت بازگشتی می توان با استفاده از گرامر زیر تولید کرد :

$$E \rightarrow \alpha \mid \alpha\, E \mid \beta\, E\, E$$

در این گرامر، E یک عبارت پیشوندی و α نمایانگر یک عملوند ساده است .همچنین α یک عملگر یکتایی و β عملگر دوتایی است.
با دقت به گرامر فوق مشاهده می شود که این گرامر ابهام ندارد و مرتبه ی پیچیدگی زمانی O (n) است.

? نتیجه ی عبارت پیشوندی زیر را محاسبه کنید.

+ × 2 7 × 4 12

ابتدا نتیجه ی 2 × 7 و 4 × 12 بدست آمده و در عبارت اولیه جایگذاری میشود. عبارت بدست آمده به صورت 14 + 48 خواهد بود. بنابراین نتیجه ی نهایی عبارت برابر با 62 میشود.

نمایش پسوندی

در شیوه‌ی نمایش پسوندی عبارت های ریاضی، عملگرها دقیقاً پس از عملوندهای خود قرار می گیرند .
این شیوه ی نمایش عبارت ها به نمایش معکوس لهستانی نیز شهرت دارد .
عبارت های پسوندی را به صورت بازگشتی می توان با استفاده از گرامر زیر تولید کرد:

$$E \rightarrow \alpha \mid E\ \alpha \mid E\ E\ \beta$$

در این گرامر، E یک عبارت پسوندی و α نمایانگر یک عملوند ساده است . همچنین α یک عملگر یکتایی و β عملگر دوتایی است.
با دقت به گرامر فوق مشاهده می شود که این گرامر ابهام ندارد و مرتبه ی پیچیدگی زمانی O (n) است.

? نتیجه ی عبارت پسوندی زیر را محاسبه کنید.

2 7 × 4 12 × +

ابتدا نتیجه ی 2 7 × و 4 12 × به دست آمده و در عبارت اولیه جایگذاری می شود. عبارت به دست آمده به صورت 14 48 + خواهد بود. بنابراین نتیجه ی عبارت برابر 62 می شود.

? عبارت میان وندی (2+3)×5+A/B×2+(7/B) را به عبارت پسوندی تبدیل کنید.

با جایگذاری دو + 3 2→(2+3)=E1 تبدیل و / B 7→(7/B)=E2 در عبارت فوق، عبارت E1×5+A/B×2+E2 بدست می آید. حال با جایگذاری دو تبدیل × 5 E1→E1×5=E3 و / B A→A/B=E4 در این عبارت، عبارت E3+E4×2+E2بدست می آید. اکنون تبدیل × 2 E4→E4×2=E5 در عبارت قرار گرفته و عبارت E3+E5+E2 بدست می آید.

با انجام تبدیل + E5 E3→E3+E5=E6 عبارت به شکل E6+E2 در می آید. در نهایت نیز این عبارت به شکل + E2 E6→E6+E2=E7 تبدیل میشود. حال کافی است عبارت های تبدیل ها در عبارت قرار داده شوند تا عبارت پسوندی معادل عبارت اولیه بدست آید :

E7 → E6 E2 + → E3 E5 + 7 B / + → E1 5 × E4 2 × + 7 B - + → 2 3 + 5 × A B / 2 × + 7 B / +

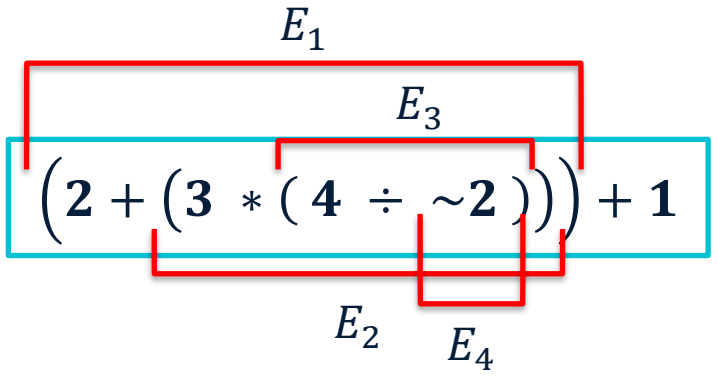
برای تبدیل عبارت میان وندی فوق به عبارت پیشوندی معادل نیز همین مراحل انجام میشود، تنها تفاوت در این است که در هر تبدیل باید عملگر پیش از دو عملوند قرار بگیرد. به عنوان مثال تبدیل اول به صورت E1=(2+3)→+ 2 3خواهد بود. بنابراین عبارت نهایی بدست آمده ی پیشوندی به صورت زیر خواهد بود :

E7 → + E6 E2 → + + E3 E5 / 7 B → + + × E1 5 × E4 2 / 7 B → + + × + 2 3 5 × / A B 2 / 7 B

توجه: با کمی دقت به مثال های فوق، دیده می شود که دو نمایش پیشوندی و پسوندی معکوس یکدیگر نیستند و از معکوس کردن مستقیم یک عبارت پیشوندی یا پسوندی نمیتوان به نمایش دیگر عبارت رسید. البته الگوریتم تبدیل این دو نمایش به یکدیگر از مرتبه ی پیچیدگی $\theta(n)$ است و این دو نمایش به راحتی به یکدیگر تبدیل می شوند، ولی مزیت نمایش پسوندی به نمایش پیشوندی این است که هنگام ورود عبارت به الگوریتم ارزیابی آن، به محض وارد شدن یک عملگر، نتیجه ی عمل محاسبه می شود و نیازی به ذخیره عملگرها در زمان ارزیابی نیست، که این مجر به ساده تر شدن پیاده سازی الگوریتم ارزیابی عبارت های پسوندی می شود.

? عبارت میان وندی زیر را به عبارت پسوندی تبدیل کنید.

$$\left(2 + \left(3 * \left(4 \div \sim 2\right)\right)\right) + 1$$



$E_{in} \rightarrow E_{1in} + 1$

$E_{pre} \rightarrow + E_{1pre} 1$

$E_{1in} \rightarrow 2 + E_{2in}$

$E_{1pre} \rightarrow +2\ E_{2pre}$

$E_{pre} \rightarrow + +2 E_{2pre}\ 1$

$E_{2in} \rightarrow 3 * E_{3in}$

$E_{2pre} \rightarrow * 3\ E_{3in}$

$E_{pre} \rightarrow + +2 * 3\ E_{3pre} 1$

$E_{3in} \rightarrow 4 \div E_{4in}$

$E_{3pre} \rightarrow \div 4 E_{4pre}$

$E_{pre} \rightarrow + +2 * 3 \div 4 E_{4pre} 1$

$E_{4in} \rightarrow \sim 2$

$E_{4pre} \rightarrow \sim 2$

$E_{pre} \rightarrow + +2 * 3 \div 4 \sim 2\ 1$

روش سریع تر

از داخلی ترین شروع می کنیم :

$$(2 + (3 * (4 \div \sim 2))) + 1$$

$$\div\ 4\ \sim 2$$

$$*\ 3\ \div\ 4\ \sim 2$$

$$+\ 2\ *\ 3\ \div\ 4\ \sim 2$$

$$+\ +\ 2\ *\ 3\ \div\ 4\ \sim 2\ 1$$

? عبارت پیشوندی زیر را به میان وندی تبدیل کنید.

$$+ + \mathbf{2} \ * \mathbf{3} \ \div \mathbf{4} \sim \mathbf{2} \ \mathbf{1}$$

از راست به چپ به اولین عملگر که رسیدیم باید دو عملوند جدا کنیم :

$$+ + 2 \ * 3 \ \div 4 \sim 2 \ 1$$

$$+ + 2 \ * 3 \ (4 \div (\sim 2)) \ 1$$

$$+ + 2 \ \left(3 \ * \left(4 \div (\sim 2)\right)\right) 1$$

$$+ \left(2 + \left(3 \ * \left(4 \div (\sim 2)\right)\right)\right) 1$$

$$\left(2 + \left(3 \ * \left(4 \div (\sim 2)\right)\right)\right) + 1$$

الگوریتم تبدیل عبارت میان وندی به پسوندی با استفاده از پشته

۱. یک پشته ی خالی S ایجاد کنید.
۲. چپ ترین نشانه ی خوانده نشده را خوانده و درX قرار دهید.
۳. اگر X عملوند بود، آن را چاپ کرده و به گام ۷ بروید.
۴. اگر X پرانتز باز بود، آن را داخل پشته وارد کرده و به گام ۷ بروید.
۵. اگر X عملگر بود :
۵-۱: اگر پشته خالی بود و یا در بالای پشته پرانتز باز قرار داشت، X را داخل پشته وارد کرده و به گام ۷ بوید.
۵-۲: اگر در بالای پشته یک عملگرY قرار داشت:
۵-۲-۱: اگر اولویت X از Y بیش تر بود، X را در پشته وارد کرده و به گام ۷ بروید.
۵-۲-۲: اگر اولویت X از Yکمتر یا مساوی بود، Yرا از پشته خارج کرده،آن را چاپ کنید و سپس به گام ۵ بروید.
۶. اگر X پرانتز بسته بود، عناصر پشته را تا رسیدن به پرانتز باز از پشته خارج کرده و چاپ کنید.
۷. نشانه ی X بعدی را از ورودی خوانده و به گام ۳ بروید . اگر ورودی به پایان رسیده است، به گام ۸ بروید.
۸. تمام عناصر پشته را به ترتیب از پشته خارج کرده و چاپ کنید.

? عبارت میان وندی زیر را با استفاده از الگوریتم صفحه قبل به عبارت پسوندی تبدیل کنید.

$$A+(2+3\times5+B)\times(4+7)/8$$

نشانه ی x	پشته ی عملگر	خروجی پسوندی
A		A
+	+	A
(	+(	A
2	+(	A 2
+	+(+	A 2
3	+(+	A 2 3
×	+(+×	A 2 3
5	+(+×	A 2 3 5
+	+(+	A 2 3 5 ×
+	+(	A 2 3 5 × +
+	+(+	A 2 3 5 × +
B	+(+	A 2 3 5 × + B
)	+	A 2 3 5 × + B +
×	+×	A 2 3 5 × + B +
(	+×(	A 2 3 5 × + B +
4	+×(	A 2 3 5 × + B + 4
+	+×(+	A 2 3 5 × + B + 4
7	+×(+	A 2 3 5 × + B + 4 7
)	+×	A 2 3 5 × + B + 4 7 +
/	+	A 2 3 5 × + B + 4 7 + ×
/	+/	A 2 3 5 × + B + 4 7 + ×
8	+/	A 2 3 5 × + B + 4 7 + × 8
پایان ورودی		A 2 3 5 × + B + 4 7 + × 8 / +

۱. یک پشته ی خالی S ایجاد کنید.
۲. چپ ترین نشانه ی خوانده نشده را خوانده و درX قرار دهید.
۳. اگر X عملوند بود آن را داخل پشته وارد کرده و به گام ۵ بروید.
۴. اگر X عملگر بود :
۴-۱: اگر X عملگر یکتایی بود:
۴-۱-۱: بالاترین عملوند را از پشته خارج کرده و آن را Y نامگذاری کنید.
۴-۱-۲: عملگر X را بر روی عملوند Y اعمال کرده و نتیجه را در Z بریزید.
۴-۱-۳: Z را در پشته وارد کنید
۴-۲: اگر X عملگر دوتایی بود:
۴-۲-۱: دو عملوند بالای پشته را از پشته خارج کنید. اولین عملوند خارج شده را y۱ و دومی‌را y۲ نامگذاری کنید.
۵. نشانه‌ی X بعدی را از ورودی خوانده و به گام ۳ بروید. اگر ورودی به پایان رسیده است، به گام ۶ بروید.
۶. تنها یک عنصر در پشته وجود داردکه جواب ارزیابی عبارت است.

? عبارت پسوندی + × 2 7 / - 1 4 × 3 5 را با انجام مراحل ارزیابی کنید.

5 3 × 4 1 - / 7 2 × +

نشانه ی x	پشته ی عملوندها	عملیات انجام شده
5	5	PUSH(S, x)
3	5 3	PUSH(S, x)
×	15	Y2 = POP(S) = 3, Y1 = POP(S) = 5, Z = Y1×Y2, PUSH(S, Z)
4	15 4	PUSH(S, x)
1	15 4 1	PUSH(S, x)
-	15 3	Y2 = POP(S) = 1, Y1 = POP(S) = 4, Z = Y1-Y2, PUSH(S, Z)
/	5	Y2 = POP(S) = 3, Y1 = POP(S) = 15, Z = Y1/Y2, PUSH(S, Z)
7	5 7	PUSH(S, x)
2	5 7 2	PUSH(S, x)
×	5 14	Y2 = POP(S) = 2, Y1 = POP(S) = 7, Z = Y1×Y2, PUSH(S, Z)
+	19	Y2 = POP(S) = 14, Y1 = POP(S) = 5, Z = Y1+Y2, PUSH(S, Z)

با پایان الگوریتم ارزیابی، تنها عدد ۱۹ در پشته ی S باقی مانده است. که نتیجه ی ارزیابی عبارت فوق است. همچنین با توجه به این که دقیقا یک عدد در پشته باقی است، پس عبارت پسوندی ورودی صحیح است.

الگوریتم ارزیابی عبارت های پیشوندی با استفاده از پشته

الگوریتم ارزیابی و محاسبه ی نتیجه ی یک عبارت پیشوندی، بسیار شبیه بـه الگـوریتم ارزیـابی عبارت های پسوندی است، با دو تفاوت زیر:

۱. بنابر در این الگوریتم، رشته ی ورودی (عبـارت پیشـوندی ورودی) از راسـت بـه چـپ بررسـی می شود. این در گام ۲ باید راست ترین نشانه خوانده شده و درX قرار بگیرد.

۲. به دلیل این که عملونـدها بـه ترتیـب عکـس الگـوریم قبلی در پشـته وارد می‌شوند، در گام ۴-۲-۱ باید اولین عنصر خـارج شـده از پشـته در y۱ و دومـین عنصـرخارج شده از پشته در y۲ قرار بگیرد.

? عبارت پیشوندی 2 7 × 1 4 – 3 5 × / + را ارزیابی کنید.

+ / × 5 3 – 4 1 × 7 2

عملیات انجام شده	پشته ی عملوندها	نشانه ی x
PUSH(S, x)	2	2
PUSH(S, x)	2 7	7
Y2 = POP(S) = 7, Y1 = POP(S) = 2, Z = Y1×Y2, PUSH(S, Z)	14	×
PUSH(S, x)	14 1	1
PUSH(S, x)	14 1 4	4
Y2 = POP(S) = 4, Y1 = POP(S) = 1, Z = Y1-Y2, PUSH(S, Z)	14 3	-
PUSH(S, x)	14 3 3	3
PUSH(S, x)	14 3 3 5	5
Y2 = POP(S) = 5, Y1 = POP(S) = 3, Z = Y1×Y2, PUSH(S, Z)	14 3 15	×
Y2 = POP(S) = 15, Y1 = POP(S) = 3, Z = Y1/Y2, PUSH(S, Z)	14 5	/
Y2 = POP(S) = 5, Y1 = POP(S) = 14, Z = Y1+Y2, PUSH(S, Z)	19	+

با پایان الگوریتم ارزیابی، تنها عدد ۱۹ در پشته ی S باقی مانده است. که نتیجه ی ارزیابی عبارت فوق است. همچنین با توجه به این که دقیقا یک عدد در پشته باقی است، پس عبارت پیشوندی ورودی صحیح است.

پایان