

۱- داده گونه های مجرد و اولیه

۱-۱- ویژگی های یک برنامه خوب

۱. درست کار کند

۲. فهمیدن برنامه آسان باشد و به راحتی قابل تغییر باشد

۳. پس زمینه ی استدلالی درست داشته باشد

یکی از راه ها برای رسیدن به هدف دوم استفاده از گونه داده ای مجرد (۱) است. ایده ی اصلی گونه های مجرد (ADT) جدا کردن روش های تشخیص است. یعنی اینکه بدانیم با چه نوع داده ای سر و کار داریم و چه نوع عملگر هایی بر روی این داده تعریف شده اند.

فواید استفاده از گونه های مجرد را میتوان در سه مورد خلاصه کرد:

۱. فهم آسان کد

۲. برای هدف های مختلف پیاده سازی برای گونه های داده ای مجرد (۲) ساده است، یعنی بدون اینکه نیاز باشد کل برنامه را تغییر دهیم با تغییرات اندک میتوان برنامه را برای گونه داده های مختلف استفاده کرد.

۳. ویژگی دوم سبب میشود تا بتوان کدی که بدین شکل نوشته شده را در آینده و برای پروژه های دیگر نیز استفاده کرد.

۱-۲- انواع گونه های مجرد

در کل دو نوع گونه ی مجرد داریم:

۱. public یا external که شامل قسمت های زیر است:

- دید ادراکی از تصویر داده (دید کاربر از اینکه داده ی مورد نظر چه شکلی است)

- دید ادراکی از عملکرد داده (دید کاربر نسبت به اینکه داده چه کارهایی میتواند بکند)

۲. private یا internal که شامل قسمت های زیر است:

- نمایش ساختاری داده (اینکه داده در حقیقت چگونه ذخیره میشود)

- اجرای عملگرهای داده (قسمت واقعی کد)

قسمت آخر یک گونه داده ی مجرد (اجرای عملگرها) در کل میتواند شامل دسته های زیر باشد:

۱. تعریف داده

۲. افزودن داده ی جدید به داده های درون داده ی اصلی

۳. دسترسی به داده های درون داده ی اصلی

۴. پاک کردن داده های درون داده ی اصلی

از جمله ی این داده گونه ها می توان به List، Graph، Queue، Container، Stack، Set و Pile اشاره کرد.

۱-۳- مراحل تبدیل یک کد به برنامه

۱. تجزیه (۳): تجزیه ی کد یعنی شکستن کد به قسمت های ریزتر. قبل از اینکه کد به یک اطلاعات معنی دار تبدیل شود باید تجزیه شود. بعد از تجزیه کد آماده ی مرحله ی بعدی است.

۲. کامپایل: کامپایل یک کد تجزیه شده در حقیقت تبدیل آن به یک برنامه است. البته این مرحله شامل ۲ گام است:

- تبدیل سورس کد تجزیه شده به یک object code

- object code توسط یک لینکر لینک میشود

۱-۴- مدیریت حافظه در حال اجرا در جاوا

جاوا یک پردازنده سطح سیستم عامل (۴) است. سیستم عامل و طراحی کامپیوتر یک سری محدودیت ها را برای اجرای برنامه به همراه می آورد. فرض میکنیم کامپیوتر دارای یک حافظه ی ۳۲ بیتی است، یعنی پهنای هر یک از خانه ها ی حافظه ۳۲ بیت است. اگر شماره ی خانه ها از ۰ تا 0xFFFFFFFF باشد یعنی حجم حافظه ۴ GB خواهد بود.

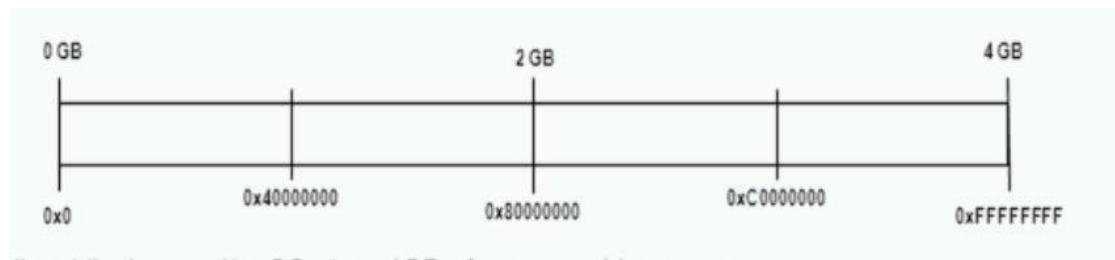


Figure ۱\نمایی از خانه های حافظه در یک کامپیوتر ۳۲ بیتی

مقداری از حافظه به OS (سیستم عامل) و C-Language Runtime اختصاص میابد.

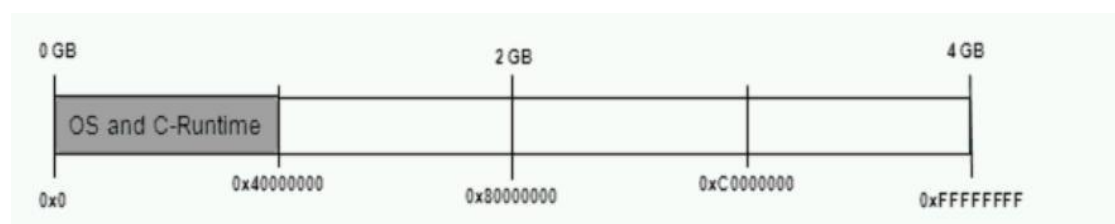


Figure ۲\نحوه ی اختصاص خانه های ابتدای حافظه

در سیستم عامل ویندوز این حجم ۲ گیگابایت است اما در لینوکس به ۱ گیگابایت کاهش میابد. به حجم باقیمانده فضای کاربری (user space) گفته میشود.

قسمتی از فضای حافظه هنگام اجرای جاوا به JVM اختصاص میابد. این حافظه شامل موتور اجرا و JIT Compiler و غیره میباشد.

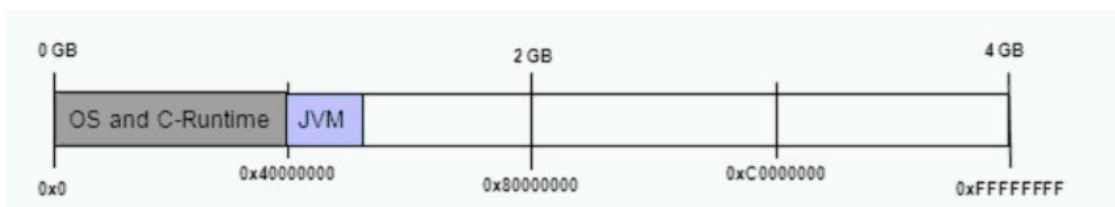


Figure ۳\حافظه ی اختصاص داده شده به JVM

بقیه ی حافظه به هیپ ها اختصاص میابند:

۱. java heap: هیپ های مربوط به اجرای برنامه که مقادیر ماکزیمم و مینیمم برایش تعیین میشود

۲. native(System) heap: این هیپ برای اجرای ریسمان های (threads) مختلف است

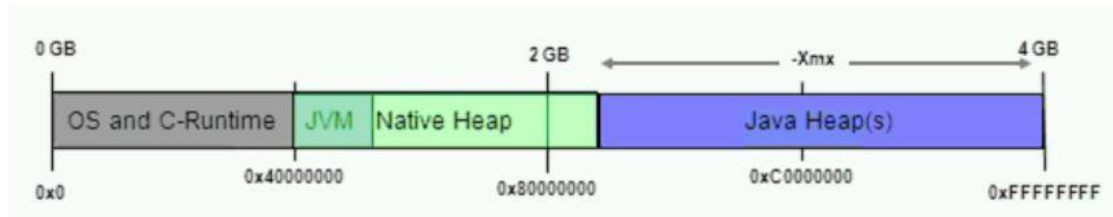


Figure ۴ حافظه ی مربوط به هیپ ها

جاوا یک زبان statically-typed است . یعنی همه ی متغیر ها باید قبل از استفاده شدن تعریف شوند.

حال که داده گونه های مجرد را بررسی کردیم به یک نوع داده گونه ی دیگر میپردازیم یعنی داده گونه های اولیه (Primitive Datatype). این داده گونه ها ویژگی مشترکی با هم دارند که آنها را از non Primitive Data Type جدا میکند، نداشتن متد (method) و اینکه حافظه های ثابتی اشغال میکنند. داده گونه های اولیه به چند دسته تقسیم میشوند:

Type	Contains	Range	Storage Requirment(bit)	Defualt
boolean	true or false	NA	1	false
char	Unicode character unsigned	\u0000 to \uFFFF	16	\u0000
byte	Signed integer	-128 to 127	8	0
short	Signed integer	-32768 to 32767	16	0

int	Signed integer	-2147483648 to 2147483647	32	0
long	Signed integer	-9223372036854775808 to 9223372036854775807	64	0
float	IEEE 754 floating point single-precision	$\pm 1.4E-45$ to $\pm 3.4028235E+38$	32	0.0
double	IEEE 754 floating point double-precision	$\pm 4.9E-324$ to $\pm 1.7976931348623157E+308$	64	0.0

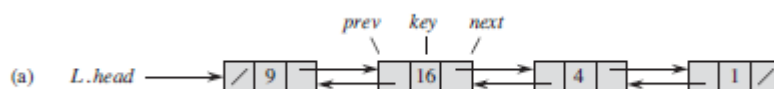
Table ۱ انواع داده گونه های اولیه

۲- آرایه

چگونه می توان اشاره گر و اشیاء را در زبان هایی، مثل فرترن، که آن ها را پشتیبانی نمی کند پیاده سازی کرد؟ برای این کار می توان از جایگزینی آنها با آرایه و اندیس آرایه استفاده کرد. این تنها یک نمونه از کاربرد های فراوان آرایه میباشد که در این بخش به آن پرداخته میشود.

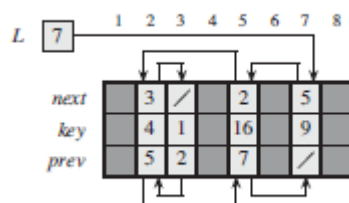
۲-۱- پیاده سازی یک لیست پیوندی با آرایه ی چند بعدی

شکل زیر پیاده سازی یک لیست پیوندی با آرایه را نشان می دهد.



پیاده سازی لیست پیوندی با آرایه

برای این پیاده سازی ما میتوان از یک آرایه چند بعدی استفاده کرد که طول ردیف های آن سه است.



پیاده سازی لیست پیوندی با آرایه

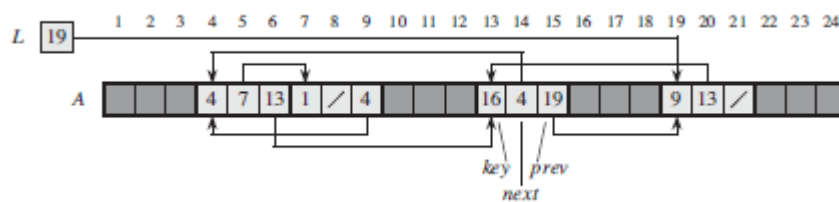
همانطور که در شکل بالادیده می شود، یک آرایه چند بعدی با سه ردیف داریم. ردیف اول اندیس خانه بعدی (next)، ردیف دوم کلید (key)، و ردیف سوم اندیس خانه قبلی (prev) هر جزء از لیست پیوندی را نگه میدارد. "L" نیز سر لیست پیوندی را مشخص میکند. هر قسمت عمودی از آرایه نمایش یک شی تنها است. خانه های روشن حاوی عناصر لیست هستند و خانه های تیره نیز بدون استفاده میباشند. برای مقداردهی خانه ((/)) از یک مقدار که در اندیس های حقیقی ظاهر نمیشود (مانند -۱) استفاده میکنیم.

برای نمونه عدد ۴ در ستون دوم از ردیف key قرار دارد و مقدار next آن ۳ میباشد. بنابراین عنصر بعدی آن در لیست پیوندی در ستون سوم از آرایه ما قرار دارد که key آن عنصر ۱ است، پس عنصر بعد از عدد ۴ در لیست پیوندی، ۱ است.

از طرفی مقدار prev کلید ۴، ۵ بوده و بنابراین عنصر قبلی آن در لیست پیوندی در ستون پنجم از آرایه ما قرار دارد که key آن ۱۶ است، پس عنصر قبلی عدد ۴ در لیست پیوندی، ۱۶ است.

۲-۲ پیاده سازی لیست پیوندی و آرایه ی چندبعدی با آرایه یک بعدی

حال میخواهیم لیست پیوندی شکل ----- و آرایه چند بعدی شکل ----- را به وسیله یک آرایه یک بعدی نشان دهیم.



پیاده سازی لیست پیوندی با آرایه یک بعدی

شکل بالا نحوه پیاده سازی لیست پیوندی شکل ---- و آرایه چند بعدی شکل ----- در یک آرایه یک بعدی را نشان میدهد.

به این ترتیب که *key* هر عنصر لیست پیوندی ما در یک خانه از آرایه با اندیس j (که باقیمانده آن بر ۳، ۱ است) قرار میگیرد. سپس *prev* و *next* آن عنصر در خانه های $j+1$ (که باقیمانده آن بر ۳، ۲ است) و $j+2$ (که مضرب ۳ است) گذاشته میشود. همانند قبل *L* سر لیست پیوندی را مشخص میکند.

برای نمونه در خانه ۱۹ ام عنصر با "*key*" ۹ قرار دارد. در خانه ۲۰ ام $(1+19)$ اندیس عنصر بعدی عدد ۱۹ قرار دارد که ۱۳ است و چون در خانه ۱۳ ام عنصر ۱۶ قرار دارد پس عنصر بعدی عدد ۹ در لیست پیوندی ۱۶ است. از طرفی در خانه ۲۱ ام $(2+20)$ اندیس عنصر قبلی عدد ۱۹ قرار دارد که "/" است. پس عنصر قبلی برای عدد ۹ وجود ندارد و ۹ در ابتدای لیست پیوندی قرار دارد.

۲-۳- پیدا کردن آدرس خانه i ام در یک آرایه

ما میخواهیم با داشتن آدرس اولین خانه از یک آرایه، آدرس هر کدام از خانه های آرایه را که میخواهیم بدست آوریم. بدست آوردن این آدرس با توجه به جنس دز نظر گرفته شده برای آرایه متفاوت است. در اینجا مبنای جنس آرایه را *int* در نظر گرفته می شود که هر خانه ۴ بایت فضا را در حافظه اشغال میکند.

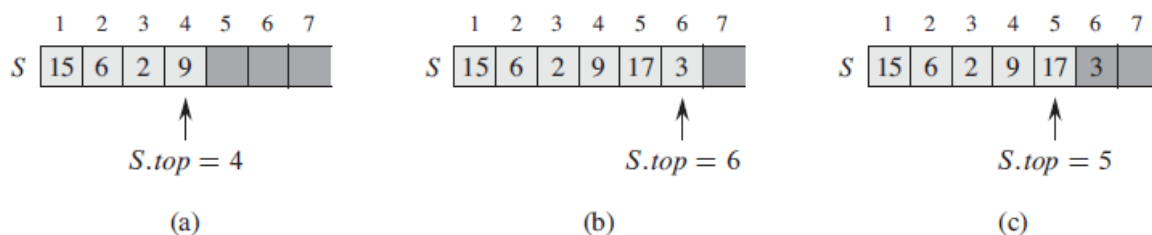
پس برای پیدا کردن خانه مورد نظر کافی است تا ۴ برابر "تعداد خانه های قبل آن خانه" را به آدرس خانه اول اضافه کنیم.

به عنوان مثال در یک آرایه دو بعدی مانند $\text{int } A[5][3]$ و با فرض اینکه آدرس خانه ی اول آرایه ۱۰۰۰ است، آدرس خانه ی $A[2]$ به این صورت بدست می آید.

$$\text{Address of } A[2] = 1000 + (2 \times 3 \times 4) = 1024$$

۳- پشته :

پشته یا "استک" ساختمان داده ای " LIFO " یا " Last In First Out " است، یعنی اولین خروجی از پشته، آخرین ورودی پشته است. همچنین پشته دارای دو عمل PUSH و POP است، push یک متغیر را داخل پشته و روی متغیرهایی که قبلاً داخل پشته بوده قرار میدهد و pop یک متغیر را از سر پشته برداشته و به عنوان خروجی بیرون میدهد.



مثالی از پشته

در قسمت a در شکل ----- یک ساختمان داده پشته با ۴ عنصر دیده میشود. در این لحظه عددی که تابع $top()$ میدهد ۴ است که یعنی بالاترین عنصر پشته در خانه چهارم قرار دارد.

در قسمت b در این شکل، دو عدد ۱۷ و ۳ به ترتیب در پشته push شده اند. اینبار عدد برگردانده شده توسط تابع $top()$ ۶ است، چون دو عنصر به پشته ما اضافه شده است.

در قسمت c، ابتدا تابع POP فراخوانی شده و بنابراین عنصر سر پشته که ۳ بود از آن خارج شده است. حالا اینبار عدد برگردانده شده توسط تابع $top()$ ، میشود.

شبه کد تابع PUSH در پشته : (مقدار اولیه $top=-1$)

```
Int push(Datatype x) {  
    If ( is.full() )  
        Return -1;  
    S[++top] = x;  
    Return top;  
}
```

شبه کد تابع POP در پشته : (مقدار اولیه $top=-1$)

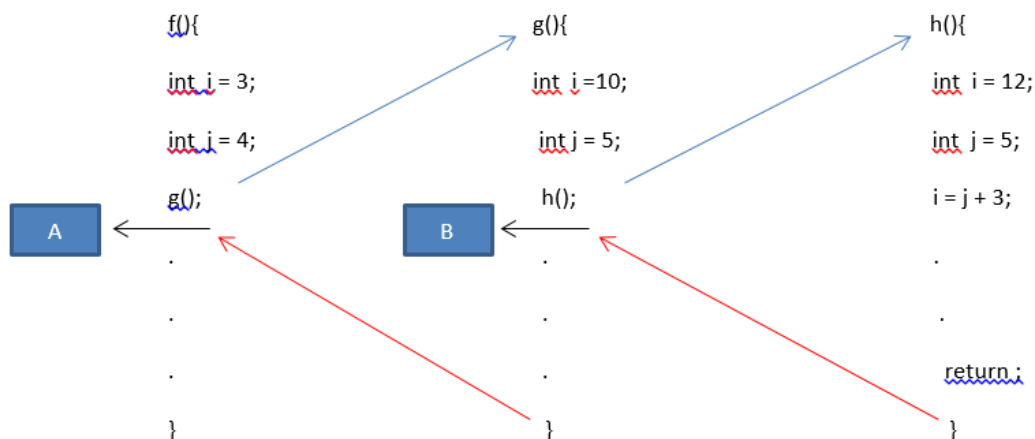
```
Datatype pop() {  
    If ( is.empty() )  
        Return -1;  
    Return S[top--];  
}
```

۳-۱- کاربرد های پشته

از کاربردهای متعدد پشته میتوان به مدیریت توابع و عبارت های محاسباتی اشاره کرد.

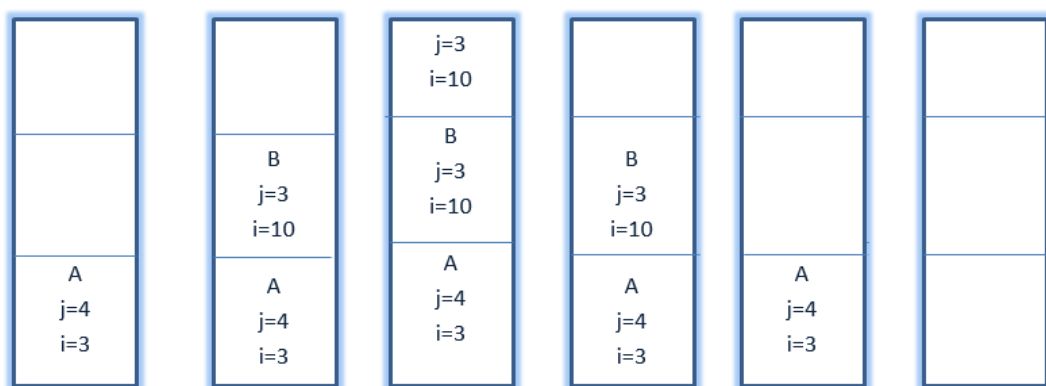
۳-۱-۱- مدیریت توابع

در فراخوانی توابع و پیاده سازی آنها ما از پشته ها و توابع آنها استفاده میکنیم. مثال زیر طرز پیاده سازی و استفاده از این توابع را برای شما روشن میکند :



مدیریت و فراخوانی توابع

برای انجام تابع $f()$ ما از یک پشته استفاده می کنیم. در ابتدا متغیرهای خود تابع $f()$ در قسمتی از پشته PUSH میشوند. سپس آدرس مکان A (آدرس بازگشت) و سپس متغیرهای تابع $g()$ و آدرس مکان B و در آخر متغیرهای تابع $h()$ ، PUSH میشوند. پس از پایان کار تابع $h()$ و خارج شدن از آن، متغیرهای آن نیز از پشته POP میشوند و فرآیند کار به آدرس B بازمیگردد و ادامه تابع $g()$ تا پایان انجام شده و پس از اتمام آن متغیرهای این تابع نیز از پشته POP شده و فرآیند کار به آدرس A میرود و ادامه تابع $f()$ انجام شده و کار به پایان میرسد؛ و بدین صورت فرآیند تابع $f()$ با استفاده از یک پشته و دو تابع PUSH و POP آن پیاده سازی می شود.



مراحل فراخوانی و بازگشت از تابع

شکل بالا ۶ مرحله انجام این تابع را به ما نشان می دهد.

۳-۱-۲- عبارت های محاسباتی

اولویت عملگرهای محاسباتی: ۱- (داخل پرانتز) ۲- (یکتایی ها) ۳- (^) ۴- (* / %) ۵- (+ -)

انواع عبارت های محاسباتی:

الف) پیشوند (prefix): عملگر قبل از عملوند می آید.

ب) میانوند (infix): عملگر وسط از عملوندها می آید.

پ) پسوند (postfix): عملگر بعد از عملوند می آید.

انواع عملگرهای محاسباتی:

الف) یکتایی (unary): مثل \sin , \cos , $++$, $--$, \sim

الف) دوتایی (binary): مثل $+$, $-$, $*$, $/$, $\%$

الف) سه تایی (tinary): مثل $a?b:c$

۳-۱-۳- تبدیل عبارت میانوندی به پسوندی با استفاده از پشته

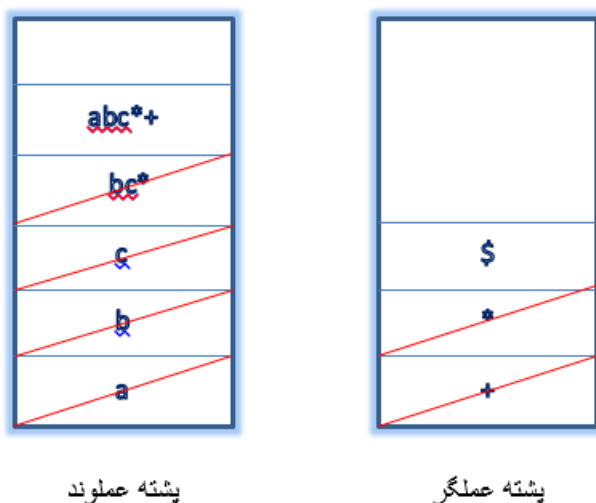
ما میتوانیم این کار را با استفاده از دو پشته انجام دهیم. یک پشته برای "عملوند ها" و یک پشته برای "عملگرها". از ابتدای عبارت میانوندی شروع میکنیم و عملوند ها را در پشته خود و عملگرها را نیز در پشته خود PUSH میکنم. در PUSH کردن عملگرها در پشته باید به این نکته توجه کنیم که یک عملگر فقط روی عملگری با اولویت کمتر از خود PUSH میشود، یعنی ما نمیتوانیم یک عملگر را روی عملگری با اولویت بیشتر از خود PUSH کنیم. اگر این اتفاق در حال افتادن بود ما باید عملگر قبلی را POP کرده و با توجه به نوع عملگر (یکتایی، دوتایی یا سه تایی) یک یا دو یا ۳ عنصر از پشته عملوند ها را POP کرده و عملگر را روی آنها اعمال و عبارت پسوندی حاصل را در پشته عملوندها PUSH میکنیم و این کار را تا جایی ادامه میدهیم که در پشته عملگرها فقط عملگر \$ که نشان دهنده پایان کار است باقی بماند.

نکات:

۱- قبل از انجام عملیات در پایان هر عبارت میانوندی عملگر \$ را میگذاریم. این عملگر دارای کمترین اولویت بوده و نشان دهنده پایان عملیات میباشد.

۲- یک عملگر نمیتواند روی همان عملگر در پشته PUSH شود به جز عملگر ^ (توان).

برای مثال عبارت $a+b*c\$$ را به شکل پسوندی تبدیل میکنیم.

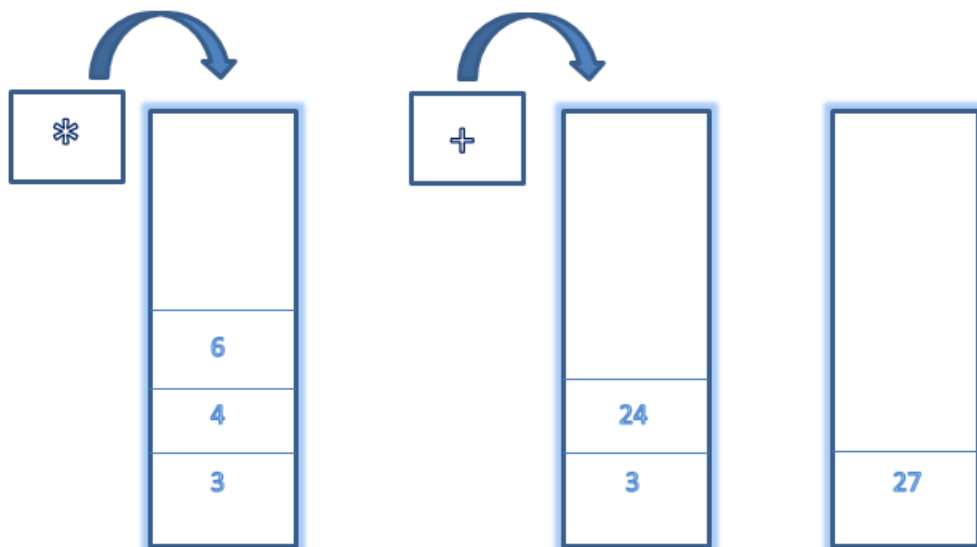


مراحل انجام عملیات :

- ۱- ابتدا به ترتیب a و b و c در پشته عملوند ها، و + و * در پشته عملگر ها PUSH میشوند.
- ۲- بعد نوبت به PUSH شدن \$ میرسد ولی چون اولیتش از * کمتر است نمیتواند وارد شود پس * باید POP شود.
- ۳- عملگر * و دو عملوند سر پشته عملوندها یعنی b و cPOP شده و حاصل پسوندی آنها یعنی bc* در پشته عملوندها PUSH میشود.
- ۴- حالا دوباره \$ میخواهد وارد شود ولی چون اولویتش کمتر از + است نمیتواند و برای همین + POP شده و مانند مرحله قبل اینبار c و bc* از پشته عملوندها POP شده و +abc* به جای آنها PUSH میشوند.
- ۵- حالا \$ در پشته عملگر ها PUSH میشود بنابراین عملیات تمام است و عبارت پسوندی ما بدست آمده است.

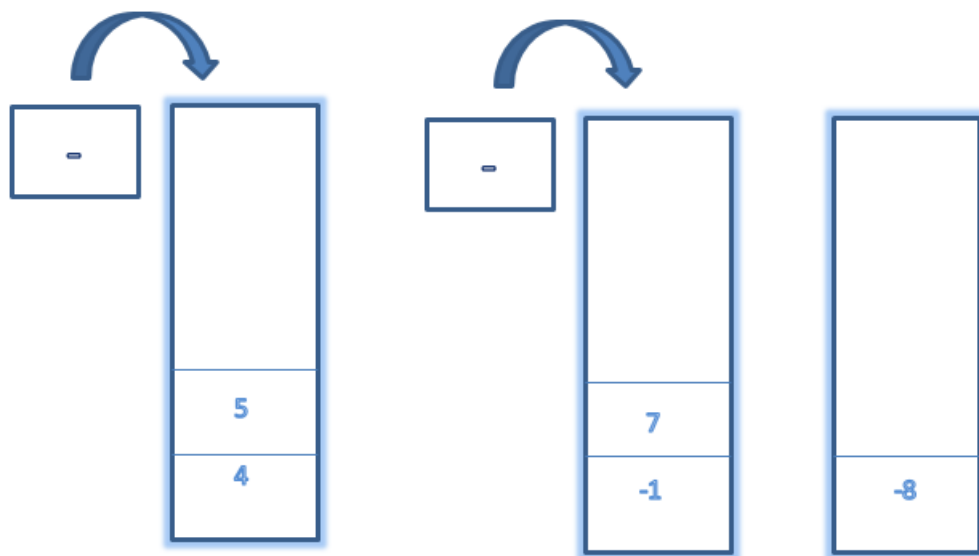
نحوه محاسبه مقدار عددی یک عبارت پسوندی با پشته ۳-۱-۴-

به عنوان مثال مقدار عددی عبارت +346* را محاسبه می کنیم.



از ابتدای عبارت پسوندی خود شروع کرده و عملوندها را در پشته PUSH میکنیم تا جایی که به یک عملگر برسیم. وقتی به عملگر رسیدیم بسته به نوع آن، اگر یکتایی بود عنصر سر پشته، اگر دوتایی بود دو عنصر سر پشته و اگر سه تایی بود ۳ عنصر سر پشته را POP کرده و عملگر را روی آنها اعمال میکنیم و حاصل را در پشته PUSH میکنیم، و این کار را ادامه میدهیم تا جایی که حاصل عبارت ما بدست آید.

شکل زیر محاسبه ی عبارت -7-45 را نشان می دهد.



۴- ساختمان داده‌های ساده

در این فصل نمایی از مجموعه‌های پویا که به وسیله ساختمان داده‌های ساده‌ای که از اشاره‌گرها استفاده می‌کنند را مورد مشاهده قرار می‌دهیم. با وجود آنکه بسیاری از ساختمان داده‌های پیچیده را می‌توان با اشاره‌گرها پیاده‌سازی کنیم، اما در اینجا فقط ساختمان داده‌های اصلی مانند پشته، صف، لیست پیوندی و درخت‌های ریشه‌دار را مورد بررسی قرار می‌دهیم. ما همچنین روش‌هایی را که به وسیله آن اشیاء و اشاره‌گرها به وسیله آرایه ترکیب می‌کنند را مورد بررسی قرار می‌دهیم.

۴-۱- پشته‌ها و صف‌ها

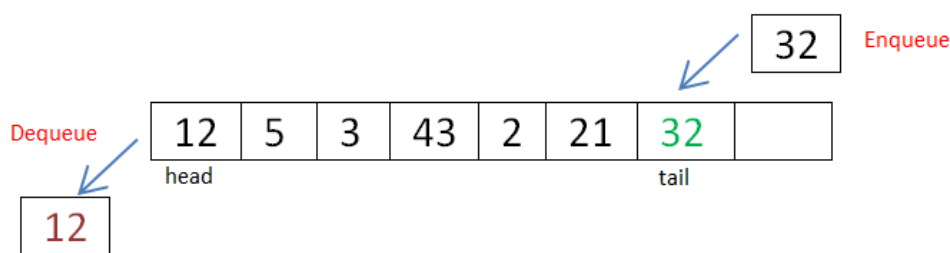
پشته (Stack) و صف (Queue) مجموعه‌های پویا هستند که در آن، عناصر به وسیله عمل پاک کردن (Delete) از مجموعه خارج می‌شوند. در پشته عنصر پاک شده از مجموعه همان آخرین عنصری که وارد آن شده است. به عبارت دیگر در پشته قانون «آخرین ورودی، اولین

خروجی «Last-in, First-out» برقرار است. به طور مشابه در صف همیشه عنصری پاک می شود که مدت طولانی تری در صف بوده است. بنابراین صف با روش «اولین ورودی، اولین خروجی» (First-in, First-out) پیاده سازی می شود. چندین راه برای پیاده سازی صف و پشته در کامپیوتر وجود دارد. در این قسمت نشان می دهیم که چگونه از آرایه برای پیاده سازی هر دو ساختمان داده استفاده کنیم.

۴-۲- صف ها

۴-۲-۱- اضافه کردن (Enqueue) و خارج کردن (Dequeue) از صف

عملیات وارد کردن یک عنصر به صف Enqueue و عملیات خارج کردن عنصر Dequeue نامیده می شود و مانند عمل Pop در پشته، Dequeue هیچ آرگومانی را به عنوان ورودی دریافت نمی کند. ویژگی "اولین ورودی، اولین خروجی" در صف سبب می شود که عملکردی مشابه صف دانشجویان در صف دریافت غذا داشته باشد. هر صف یک ابتدا (Head) و یک انتها (Tail) دارد. هر زمان که یک عنصر وارد صف شود، همیشه در انتهای صف قرار می گیرد. درست مانند صف سلف دانشجویان که دانشجوی تازه وارد در آخر صف قرار می گیرد. همچنین همیشه عنصری حذف می شود که در سر صف قرار دارد، مانند دانشجویی که بعد از مدتی طولانی به سر صف رسیده و غذا دریافت می کند. (شکل ۱)



شکل ۱. ورود و خروج از ساختمان داده ی صف

صف دو ویژگی و ویژگی دارد. Q.head که سر صف را مشخص می کند و Q.tail که موقعیت مکانی را که عنصر جدید قرار است وارد آن شود را نشان می دهد. عناصر در صف در مکان های Q.head، Q.head+1 و ... Q.tail-1 قرار دارند، به طوری که «دور می زند» یعنی بعد از مکان ۱ مکان n به صورت دایره وار قرار دارد. بنابراین هرگاه Q.head=Q.tail صف خالی است. در حالت اولیه داریم Q.head = Q.tail = 1. وقتی صف خالی است سعی برای حذف کردن یک عنصر از صف با

«زیرریز» همراه است. هرگاه $Q.head = Q.tail + 1$ صف پر است و اگر بخواهیم یک عنصر به صف اضافه کنیم آنگاه صف «سر ریز» می کند.

در شبه کدهای زیر کنترل خطای مربوط به «سرریزی» و «زیرریزی» را حذف کرده ایم. این شبه کد فرض می کند $n = Q.length$

ENQUEUE (Q, X)

1. $Q[Q.TAIL] = X$
2. IF $Q.TAIL == Q.LENGTH$
3. $Q.TAIL = 1$
4. ELSE
5. $Q.TAIL = Q.TAIL + 1$

DEQUEUE (Q)

1. $X = Q[Q.HEAD]$
2. IF $Q.TAIL == Q.LENGTH$
3. $Q.HEAD = 1$
4. ELSE
5. $Q.HEAD = Q.HEAD + 1$
6. RETURN X

تمرین‌ها - ۲-۲-۴

(تمرین ۱) نشان دهید که چگونه می‌توان دو پشته بر روی آرایه $A[1..n]$ پیاده‌سازی کرد به طوری که هیچ‌کدام از پشته‌ها overflow نکنند مگر آنکه تعداد کل اعضا در هر دو پشته n شود. هزینه عمل $push$ و pop باید $O(1)$ باشد.

(تمرین ۲) اگر شکل 10.2 را به عنوان مدل بگیریم نتیجه هر یک از عمل‌های منظم و پشت سر هم $Enqueue(Q, 4)$, $Enqueue(Q, 1)$, $Enqueue(Q, 3)$, $Dequeue(Q)$, $Enqueue(Q, 1)$, $Dequeue(Q)$, $Enqueue(Q, 8)$ ، مشخص کنید. صفی با استفاده از آرایه $S[1..6]$ مشخص کنید. صف در حالت اولیه خالی است.

(تمرین ۳) $Enqueue$ و $Dequeue$ را دوباره بنویسید تا $underflow$ و $overflow$ را در یک صف مشخص کنید.

(تمرین ۴) پشته اجازه اضافه کردن و خارج کردن عناصر را از یک طرف می‌دهد حال صف اجازه وارد کردن از یک طرف و خارج کردن از طرف دیگر را می‌دهد. یک صف دوسرطراحی کنید که اجازه اضافه کردن و خارج کردن عناصر را از دو طرف بدهد. 4 تابع با هزینه $O(1)$ بنویسید که عمل اضافه کردن و خارج کردن عناصر یک صف در دو سر صف که با استفاده از آرایه پیاده‌سازی می‌شود، را انجام دهد.

(تمرین ۵) نشان دهید که چگونه می‌توان یک صف را با استفاده از دو پشته پیاده‌سازی کرد. هزینه عملیات \neg های صف را به دست آورید.

(تمرین ۶) نشان دهید که چگونه یک پشته را می‌توان با استفاده از دو صف پیاده‌سازی کرد. هزینه عملیات‌های پشته را به دست آورید.

۳-۴- لیست‌های پیوندی

لیست‌های پیوندی نوعی از ساختمان داده است که در آن اشیاء در یک ترتیب خطی قرار گرفته‌اند. بر خلاف آرایه که در آن ترتیب‌ها به وسیله اندیس‌های آرایه مشخص می‌شود، ترتیب در لیست‌های پیوندی به وسیله اشاره‌گر داخل هر شیء مشخص می‌شود. لیست‌های پیوندی یک نمایش ساده و قابل انعطافی برای مجموعه‌های پویا فراهم می‌کند.

با توجه به شکل زیر هر یک از عناصر یک لیست پیوندی دو طرفه L ، یک شیء است که یک متغیر کلید key و دو متغیر اشاره‌گر: بعدی $next$ و قبلی $prev$ دارد. یک شیء ممکن است متغیرهای جانبی دیگری هم داشته باشد. با در نظر گرفتن عنصر x در لیست، $x.next$ به عنصر بعدی و $x.prev = NIL$ به عنصر قبلی اشاره می‌کند. اگر $x.prev = NIL$ باشد یعنی عنصر قبل از آن وجود ندارد و این بدان معنا است که x سر لیست است. اگر $x.next = NIL$ باشد، یعنی عنصری بعد از آن وجود ندارد این بدان معنا است که x عنصر آخر لیست است. صفت $L.head$ به اولین عنصر لیست اشاره می‌کند. اگر $L.head = NIL$ لیست خالی است.



شکل ۲ ساختمان داده‌ی لیست پیوندی

یک لیست می‌تواند به فرم‌های گوناگون باشد. ممکن است لیست دوطرفه یا یک‌طرفه باشد. ممکن است مرتب شده یا غیرمرتب باشد و همچنین ممکن است حلقوی یا غیرحلقوی باشد. اگر لیست پیوندی یک طرفه باشد، اشاره‌گر $prev$ را از عناصر حذف خواهیم کرد، اگر لیست مرتب باشد ترتیب عناصر کلیدها در لیست ترتیب خطی دارند و عنصر مینیمم در سر لیست و عنصر ماکزیمم در ته لیست قرار دارد. اگر لیست نامرتب باشد عناصر به هر صورتی قرار خواهند داشت. در لیست پیوندی حلقوی اشاره‌گر $prev$ سر لیست، به آخر لیست اشاره می‌کند و اشاره‌گر $next$ از ته لیست به سر لیست اشاره می‌کند. ممکن است که لیست به صورت حلقه نمایش داده شود. توجه داشته باشید در این قسمت فرض می‌کنیم لیست‌ها دو طرفه و نامنظم هستند.

۳-۴-۱- جستجو در لیست پیوندی

تابع $List_Search(L, k)$ اولین عنصری که کلید k را دارد در لیست L پیدا می‌کند که با یک جستجوی خطی ساده انجام می‌شود و اشاره‌گری به آن عنصر باز می‌گرداند. اگر هیچ شی‌ای با کلید k پیدا نشود NIL باز خواهد گرداند. برای لیست پیوندی در شکل بالا تابع $List_Search(L, 2)$

اشاره‌گری به عنصر سوم باز خواهد گرداند و تابع $List_Search(L, 7)$ مقدار بازگشتی NIL خواهد داشت.

$List_Search(L, k)$

1. $x = L.head$
2. While $x \neq NIL$ and $key[x] \neq k$
3. do $x = x.next$
4. return x

برای جستجو در یک لیست پیوندی با n عنصر در بدترین حالت با $\Theta(n)$ طول می‌کشد چون باید کل لیست را جستجو کند.

۴-۳-۲- اضافه کردن به لیست پیوندی

تابع $List_Insert$ عنصری را که کلید آن x است را به اول لیست اضافه می‌کند.

$List_Insert(L, x)$

1. $x.next = L.head$
2. If $L.head \neq NIL$
3. $L.head.prev = x$
4. $L.head = x$
5. $x.prev = NIL$

زمان اجرایی تابع $List_Insert$ برای اضافه کردن یک عنصر به لیست پیوندی n عنصری از $O(1)$ است.

۴-۳-۳- حذف کردن از لیست پیوندی

تابع $List_Delete$ عنصر x را از لیست L حذف خواهد کرد. باید اشاره‌گری به عنصر x داشته باشیم و بعد آن را از لیست حذف خواهیم کرد. اگر بخواهیم عنصری با یک کلید از قبل مشخص

شده را حذف کنیم، ما باید اول List_Search را صدا بزنیم تا اشاره گر به آن عنصر را داشته باشیم.

List_Delete(L, x)

1. If $x.\text{prev} \neq \text{NIL}$
2. $x.\text{prev}.\text{next} = x.\text{next}$
3. Else
4. $L.\text{head} = x.\text{next}$
5. If $x.\text{next} \neq \text{NIL}$
6. $x.\text{next}.\text{prev} = x.\text{prev}$

تابع List_Delete با هزینه $O(1)$ اجرا می شود. اما اگر بخواهیم عنصری را با کلید مشخص حذف کنیم هزینه اجرای آن در بدترین حالت از $\Theta(n)$ است. برای اینکه ما باید اول List_Search را صدا بزنیم.

تمرین ۴-۳-۴

(تمرین ۱) اضافه کردن یک عنصر بر روی یک لیست پیوندی یک طرفه با هزینه $O(1)$ اجرا می شود؟ در مورد حذف کردن چه طور؟

(تمرین ۲) پشته را با استفاده از یک لیست پیوندی L پیاده سازی کنید عملیات های push, pop باید باز هم با $O(1)$ اجرا شود.

(تمرین ۳) صف را با استفاده از یک لیست پیوندی L پیاده سازی کنید. عملیات های Dequeue, Enqueue باید باز هم با $O(1)$ اجرا شود.

(تمرین ۴) با توجه به تابع 'List_Search' که حلقه آن برای هر دفعه باید دو مورد را چک کند. یکی $x \neq \text{NIL}$ و دیگری $\text{key}[x] \neq k$ نشان دهید که چگونه چک کردن $x \neq \text{NIL}$ را حذف کنیم.

(تمرین ۵) عملیات اضافه و حذف و جستجو را در مورد فرهنگ لغت با یک لیست پیوندی یک طرفه دایروی پیاده سازی کنید. هزینه اجرای آن را محاسبه کنید.

(تمرین ۶) عملگر مجموعه پویا Union دو مجموعه مجزای S_1, S_2 از ورودی می‌گیرد و $S = S_1 \cup S_2$ را باز خواهد گرداند که محتویات تمام اعضای S_1, S_2 است. مجموعه‌های S_1, S_2 در این عمل از بین خواهند رفت. نشان دهید که چگونه عملیات Union را با هزینه $O(1)$ با توجه به ساختمان داده مناسب تامین کنیم.

(تمرین ۷) یک تابع معرفی کنید که غیر بازگشتی باشد و یک لیست پیوندی یک طرفه با n عنصر را معکوس سازد. حافظه‌ای بیشتر از حافظه که برای لیست پیوندی بکار می‌رود نگیرید.

(تمرین ۸) توضیح دهید که چگونه یک لیست پیوندی دو طرفه را با یک اشاره گر $x.\text{np}$ که جایگزین هر دو اشاره گر $x.\text{next}$ و $x.\text{prev}$ می‌شود پیاده سازی کنیم. فرض کنید هر اشاره گر را بتوان با یک عدد صحیح k بیتی نمایش دهیم و np به صورت زیر تعریف شده است: $x.\text{np} = x.\text{next} \text{ XOR } x.\text{prev}$ (یعنی np یای انحصاری next و prev است) مقدار NIL صفر است. اطلاعات لازم برای دسترسی به سر لیست را مشخص کنید. نشان دهید که چگونه عملیات Insert, Delete, Search را روی لیست پیاده سازی کنید. نشان دهید که چگونه لیست را معکوس کنیم که هزینه اجرایی آن $O(1)$ باشد.