



SOACT Programming Language Documentation

طراحی کامپایلر و زبان‌های برنامه‌نویسی

پاییز ۱۴۰۳

فهرست مطالب

2.....	۱) مقدمه
4.....	۲) ساختار کلی
4.....	۱-۲) قواعد کلی نحو
4.....	۲-۲) Comment ها
4.....	۳-۲) قواعد نامگذاری actor ها، پردازش کننده های پیام و متغیرها
5.....	۴-۲) ایجاد نمونه از اکتورها
6.....	۳) Actor ها
7.....	۴) نحوه ارسال پیام بین actor ها
7.....	۵) تعریف پردازشگرهای پیام
8.....	۱-۵) Service Message Handler ها
9.....	۲-۵) Observe Message Handler ها
9.....	۳-۵) مجوز مشاهده و ارسال پیام
11.....	۶) روش ارسال پیام
12.....	۷) انواع داده
13.....	۸) عملگرها
13.....	۱-۸) عملگرهای حسابی
14.....	۲-۸) عملگرهای مقایسه‌ای
15.....	۳-۸) عملگرهای منطقی
16.....	۵-۸) عملگر تخصیص
16.....	۶-۸) اولویت عملگرها
17.....	۹) گزاره‌های شرطی
18.....	۱۰) توابع پیش‌فرض
19.....	۱۱) حلقه‌ها
20.....	۱-۱۱) For Loop
20.....	۲-۱۱) While Loop
20.....	۱۲) Join-Block
21.....	۱۳) عملگر (<) Pipe
21.....	۱۴) public و private
21.....	۱-۱۴) منطق private
22.....	۲-۱۴) منطق public
22.....	۱۵) تعریف CustomPrimitive
23.....	۱۶) تعریف Record
24.....	۱۷) کلیدواژه self

۱) مقدمه

SOACT یک زبان برنامه‌نویسی است که بر پایه‌ی مدل actor طراحی شده است. این نوع زبان‌ها برای مدل‌سازی، درک و استدلال در مورد سیستم‌های موازی استفاده می‌شوند.

مدل actor-based، یک روش برای طراحی سیستم‌های محاسباتی است که بر اساس actor به عنوان واحد اصلی محاسبات بنا شده است. در این مدل، سیستم‌ها به جای استفاده از threads یا سایر روش‌های سنتی مدیریت concurrency، بر اساس پیام‌رسانی و همکاری بین actorها کار می‌کنند.

یکی از مزیت‌های مدل actor-based این است که همزمانی اجرا به شکلی طبیعی و بدون نیاز به مدیریت مستقیم رخ می‌دهد. هر actor به صورت مستقل عمل می‌کند و فقط به پیام‌هایی که دریافت می‌کند پاسخ می‌دهد. این مدل باعث می‌شود که سیستم‌های پیچیده موازی به شکلی ساده‌تر و قابل‌مدیریت‌تر طراحی شوند.

actorها به عنوان واحدهای محاسباتی موازی شناخته می‌شوند که می‌توانند پیام بفرستند و دریافت کنند و به پیام‌های دریافتی واکنش نشان دهند. تعامل بین actorها به صورت ارسال پیام انجام می‌شود. هر actor یک صندوق پیام دارد که پیام‌های دریافتی را در آن نگه‌داری می‌کند.

یک actor می‌تواند در پاسخ به پیام‌هایی که دریافت می‌کند:

- به actorهای دیگر پیام بفرستد.

- actorهای جدید بسازد.

- متغیرهایش را آپدیت کند

هر actor فقط با actorهایی در ارتباط است که آدرسشان را دارد یا به عبارتی آن‌ها را می‌شناسد.

هدف اصلی طراحی زبان SOACT پیاده‌سازی شبکه‌های اجتماعی و مدیریت محرمانگی در این شبکه‌ها است. کاربران شبکه اجتماعی نمونه‌ای از اکتورهای تعریف شده می‌باشند که می‌توانند با هم با ارسال پیام تعامل داشته باشند. همچنین در تعریف اکتور می‌توان کنترل کرد که چه مجموعه‌ای از پیام‌های

ارسالی توسط چه کسانی قابل مشاهده است (مانند تنظیمات WhatsApp). نمونه‌ای کلی از کد در این زبان آمده است:

```
1 Actor X {
2     actorVars {
3         Set<ID> Followers;
4         Set<ID> Followings;
5         String name;
6     }
7
8     X(String name_, ID firstFollower, ID firstFollowing){
9         name = name_;
10        Followers.add(firstFollower);
11        Followings.add(firstFollowing);
12    }
13
14    msgRcv senderOfX(String msgContent) {
15        for (ac in Followers) {
16            ac.receiveMessage(msgContent) @observers(private(Followers, null));
17        }
18    }
19 }
20
21 Actor Y {
22     ...
23
24    msgRcv receiveMessage(String msgContent) { ... }
25 }
26
27 Actor Z {
28     ...
29
30    msgObs receiveMessage(String msgContent) { ... }
31 }
32
33 main() {
34     ID receiverActor = Y("Bob");
35     ID observerActor = Z("Charlie");
36     ID senderActor = X("Alice", observerActor, receiverActor);
37
38     senderActor.senderOfX("Hello, Charlie!");
39 }
```

برای مثال ممکن است اکتور X به اکتور Y پیام m را ارسال کند درحالی که اکتور Z اجازه مشاهده این تعامل را دارد. بنابراین در صف اکتور Y پیام ارسالی m (که Y خودش دریافت کننده است) و در صف Z پیام m (که اکتور Z مشاهده گر است) قرار می گیرد.

۲) ساختار کلی

در این زبان، کد برنامه درون یک فایل با پسوند so. قرار دارد. یک برنامه به زبان SOACT از قسمت های زیر تشکیل شده است:

- یک یا چند actor که تعریف هر actor شامل موارد زیر است:

- ❖ متغیرهای actor
- ❖ یک initializer در صورت وجود (معادل constructor در زبان های شیءگرا)
- ❖ مدیریت کننده های پیام (معادل متدهای کلاس در زبان های شیءگرا)

- main

- ❖ نمونه هایی از actorهای برنامه ساخته می شوند و شروع به فعالیت می کنند.

۲-۱) قواعد کلی نحو

زبان SOACT به بزرگ و کوچک بودن حروف حساس است. در این زبان، وجود کاراکترهای tab و space تاثیری در خروجی برنامه ندارند. جزئیات مربوط به scope و خطوط برنامه در ادامه توضیح داده خواهد شد.

۲-۲) Comment ها

در این زبان کامنت ها تک خطی هستند و تمامی کاراکترهای بعد از % تا انتهای خط کامنت به حساب می آیند.

۳-۲) قواعد نامگذاری actor ها، پردازش کننده های پیام و متغیرها

اسامی انتخابی برای نامگذاری باید از قواعد زیر پیروی کنند:

- تنها از کاراکترهای [a..z] ، [A..Z] ، _ و ارقام تشکیل شده باشند (محدودیتی روی تعداد کاراکترهای یک اسم در زبان SOACT وجود ندارد).

- با رقم شروع نشوند.

- معادل کلیدواژه‌ها نباشند. در جدول زیر تمامی کلیدواژه‌های زبان actor آمده است:

msgRcv	observers	string	if	int	actor
msgObs	for	break	else	true	self
authorized	while	continue	boolean	false	public
private	join	null	in	range	Record
actorVars	Actor	Set	length	toLower	toUpper
reverse	print	main	add	include	remove
new	List	ID	primitive		

- نام هر actor یکتاست.

- نام هر پردازش کننده‌های پیام در اسکوپ هر actor یکتاست.

- نام متغیرهای actor در هر اسکوپ actor یکتاست ولی در اسکوپ‌های درونی تر از نام‌های متغیر بیرونی می‌توان استفاده کرد. در نتیجه در طول آن اسکوپ متغیر درونی هنگام استفاده ارجحیت دارد.

- نام نمونه اکتورهای ساخته شده در main یکتاست.

۴-۲) ایجاد نمونه از اکتورها

در زبان SOACT، برای ایجاد یک نمونه از یک actor، از سازنده (constructor) و دستور new استفاده می‌کنیم. دستور new سازنده‌ی actor را فراخوانی کرده و نمونه‌ی جدیدی از آن ایجاد می‌کند. به هر

actor که با new ساخته می‌شود، یک شناسه (ID) اختصاص می‌یابد. دستور new هم نمونه‌ی actor را می‌سازد و هم یک ID منحصر به فرد به آن اختصاص می‌دهد. با استفاده از این ID، سایر actorها می‌توانند به آن actor پیام بفرستند یا پیام‌های او را دریافت کنند. مثال از نمونه کلی: اکتورهای X، Y و Z به همین ترتیب ساخته می‌شوند.

۳ Actor ها

همانطور که بالاتر به آن اشاره شد، در تعریف هر actor، بخش‌های ثابتی وجود دارند که باید در تعریف هر اکتور رعایت شوند. این بخش‌ها شامل:

1. **نام actor:** هر actor باید یک نام منحصر به فرد داشته باشد که برای شناسایی آن در کد استفاده می‌شود.
2. **سایز صف پیام (queue یا mailbox):** هر actor یک صف پیام دارد که پیام‌هایی که توسط سایر actorها ارسال می‌شوند، در آن ذخیره می‌شوند تا توسط پردازشگرها پردازش شوند. اندازه این صف باید به عنوان یک عدد ثابت بزرگ‌تر از صفر در تعریف actor مشخص شود. این صف به طور ضمنی ایجاد می‌شود و نیازی به تعریف مستقیم آن در کد نیست.

اجزای داخلی هر actor:

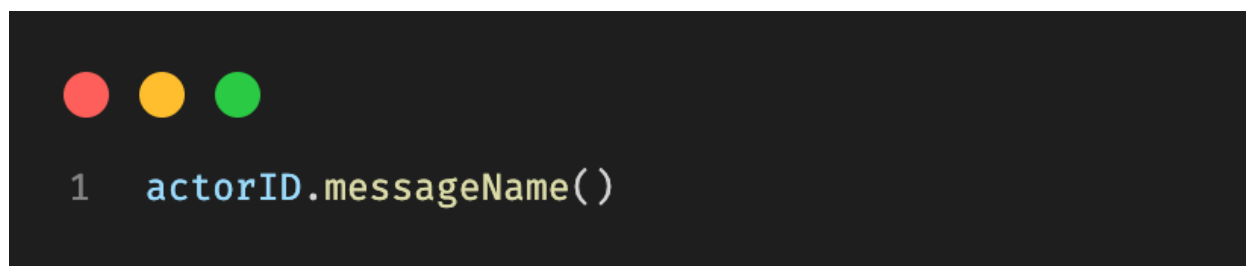
1. **متغیرهای actor (actorVars):** هر actor می‌تواند متغیرهای داخلی خود را داشته باشد که وضعیت یا داده‌هایی را که نیاز به حفظ و استفاده مکرر دارند ذخیره می‌کنند. این متغیرها به طور خاص برای هر actor تعریف می‌شوند و تنها درون آن actor قابل دسترسی هستند.
2. **constructor:** هر actor می‌تواند یک سازنده داشته باشد که برای مقداردهی اولیه متغیرهای آن استفاده می‌شود. سازنده‌ها می‌توانند پارامترهایی را بپذیرند و مقادیر اولیه متغیرهای actor را تنظیم کنند.
3. **پردازش کننده های پیام‌های دریافتی یا مشاهده شده:** هر اکتور می‌تواند پیام‌هایی را دریافت کند که این پیام‌ها از طریق پردازشگرها پردازش می‌شوند. آنها پردازشگرهایی هستند که برای مدیریت و پاسخ به پیام‌های دریافتی تعریف می‌شوند. هر پیام که از سایر actorها ارسال می‌شود، در صف پیام اکتور قرار می‌گیرد و توسط پردازشگرها، پردازش می‌گردد. در صورت

مشاهده پیامی که میان اکتورها رد و بدل شده است و از آن ها آگاه شده است و در صف او نیز قرار گرفته است که برای این پیام های مشاهده شده پردازش گر دارد.

۴) نحوه ارسال پیام بین actorها

برای ارسال پیام به یک actor، از آیدی actor مورد نظر و توسط پردازشگر مربوطه استفاده می شود. پیام ها به صف پیام های actor مقصد اضافه می شوند و وقتی که actor در صف خود به آن ها رسید، آن ها را پردازش می کند.

مثال از نحوه ارسال پیام بین actorها:



actorID: آیدی اکتوری که می خواهیم به آن پیام ارسال کنیم

messageName: پیامی که می خواهیم به actor مدنظر ارسال کنیم

مثال از نمونه کلی: ارسال پیام senderOfX به senderActor

۵) تعریف پردازشگرهای پیام

پردازشگرهای پیام (message handler ها)، بعد از تعریف در بدنه actor ها پیام ها را از صف actor دریافت می کنند این پردازنده ها مجموعه ای از دستورالعمل ها یا منطق را تعریف می کنند که هنگام دریافت یک پیام اجرا می شود. در واقع هر اکتور به طور ضمنی صف خود را بررسی می کند و در صورت وجود پیامی درون آن، آن را به کمک پردازشگر مربوط پردازش می کند. نام پیام با نام پردازشگرها یکسان است. به دو روش پیامی در صف actor ها قرار می گیرد:

- خود اکتور دریافت کننده پیام باشد

- خود اکتور دریافت کننده نیست ولی اجازه مشاهده پیام ارسالی به اکتور دیگر را دارد.

بنابراین در این زبان، دو نوع پردازشگر پیام داریم که با توجه به نوع پیام دریافتی، با یکدیگر تفاوت دارند. در ادامه هر یک را توضیح خواهیم داد.

۵-۱) Service Message Handler ها

خود اکتور دریافت کننده پیام باشد و در واقع آدرس دریافت کننده در پیام با آدرس اکتور یکی است ، نوع پردازشگری که باید این نوع پیام‌ها را پردازش کند، service message handler ها هستند. درواقع service message handler ها رفتار یک actor را زمانی که پیام خاصی دریافت می‌کند، تعریف می‌کنند. این نوع پردازشگرها در صورت مشاهده service message هم نام با خود، action خود را انجام می‌دهند و ممکن است وضعیت actor را تغییر دهند یا عملیات‌های دیگری را آغاز کنند.

مثال از تعریف Service Message Handler ها:

```
1 msgRcv MessageName() {
2     // Code for receiving a service message
3     // action
4 }
```

msgRcv: کلمه کلیدی که پردازش کننده پیام را به عنوان یک Service Handler اعلام می‌کند.

MessageName: نام پیامی که پردازش کننده پیام آن را پردازش می‌کند.


action: کد و عملیاتی که بعد از دریافت پیام (استخراج پیام از صف)، به ترتیب اجرا می‌شوند.

مثال در نمونه کلی: پردازشگر senderOfX داخل اکتور X و پردازشگر receiveMessage داخل اکتور Z

۲-۵ Observe Message Handler ها

زمانی که پیام را به یک actor مشخص ارسال می‌کنیم، می‌توانیم مشخص کنیم که گروهی از actorهای دیگر نیز آن پیام را مشاهده کنند. بنابراین زمانی که یک actor بتواند پیامی را مشاهده کند، آن پیام در صف او قرار می‌گیرد. نوع پردازشگری که باید این نوع پیام‌ها را پردازش کند، observe message handler ها هستند. این نوع پردازشگرها رفتار یک actor را زمانیکه یک پیام خاص، broadcast/multicast می‌شود، تعریف می‌کنند. درواقع این پردازشگرها نیز در صورت دریافت observe message همانم با خود، اقداماتی را انجام می‌دهند و ممکن است وضعیت actor را تغییر دهند یا عملیات‌های دیگری را آغاز کنند.

مثال از تعریف Observe Message Handler ها:



```
1 msgObs MessageName() {
2     // Code for broadcasting a message
3     // action
4 }
```

msgObs: کلمه کلیدی که پردازشگر پیام را به عنوان یک observe message handler اعلام می‌کند.

MessageName: نام پیامی که پردازشگر پیام آن را پردازش می‌کند

action: کد و عملیاتی که بعد از دریافت پیام (استخراج پیام از صف)، به ترتیب اجرا می‌شوند.

مثال در نمونه کلی: پردازشگر receiveMessage داخل اکتور Z

۳-۵ مجوز مشاهده و ارسال پیام

service message handler ها می‌توانند سیاست‌های کنترل ارسال داشته باشند که از طریق authorized تنظیم می‌شوند. سیاست ارسال برای پیام m مشخص می‌کنند که کدام actor ها

می‌توانند از طریق پیام `m` با اکتور تعامل داشته باشند. اطمینان حاصل می‌شود که فقط actor های مجاز می‌توانند پیام `m` را ارسال کنند.

لازم نیست همه‌ی actor ها برای همه‌ی `observe message` های داخل صف، پردازشگر (`message handler`) داشته باشند. این پیام‌ها فقط به گروه خاصی از actor ها فرستاده می‌شوند و بعضی از آن‌ها ممکن است پردازشگر خاصی برای این پیام‌ها نداشته باشند و بنابراین آن‌ها را نادیده می‌گیرند. اما برای پیام‌های `service` در صف هر actor، حتماً باید پردازشگر متناظر پیاده‌سازی شده باشد و نمی‌توان آن‌ها را نادیده گرفت.

نحوه استفاده از `authorized`

سیاست دسترسی (`authorized`) می‌تواند عمومی (`public`) یا خصوصی (`private`) باشد.

`authorized`: مشخص می‌کند چه actor هایی می‌توانند این سرویس مسیج را برای این actor ارسال کنند. توجه شود که این `keyword` فقط برای `service message handler` ها استفاده می‌شود.

A terminal window with a dark background and three colored window control buttons (red, yellow, green) in the top left corner. It displays a single line of code: `1 @authorized(public(Block, null))`.

```
1 @authorized(public(Block, null))
```

`public(Group1, Group2)`: همه actor ها به جز آن‌هایی که در اجتماع `Group1` و `Group2` ذکر شده‌اند، اجازه دارند پیام را ارسال کنند.

`private(Group1, Group2)`: فقط actor هایی که در اجتماع `Group1` و `Group2` فهرست شده‌اند اجازه دارند پیام را ارسال کنند.

مثال از سیاست دسترسی `service message handler`:



```
1 msgRcv @authorized(private(Followers, Followings)) MessageName() {  
2     // action  
3 }
```

در این مثال، تنها اجتماع actor های Followers و Followings می‌توانند این service message را ارسال کنند.

دقت کنید که یک actor می‌تواند پردازشگرهای های متعددی برای انواع پیام‌ها داشته باشد که هر یک نحوه واکنش actor به پیام‌های خاص را تعریف می‌کنند.

۶ روش ارسال پیام (service message- observe message)

- ارسال service handler message ها با استفاده از پرانتز و ID اکتور انجام می‌شود (به صورت مستقیم، observe message ها با استفاده از ID اکتور و نام پردازش ارسال نمی‌شوند). در صورتی که بیش از یک آرگومان ورودی داشته باشند، بین آرگومان‌ها **comma** نوشته می‌شود.

- برای استفاده از لیست‌ها به عنوان آرگومان، از reference آنها استفاده می‌شود. اما برای سایر انواع داده از مقدار آنها استفاده می‌شود.

سیاست ارسال observe message ها



```
1 @observers(private(Followers, null))
```

observers: مشخص می‌کند برای چه actorهایی observe message ساخته و ارسال می‌شود.

مثال از ارسال observe message ها:

```
1 self.MessageName() @observers(public(Block, null));
```

در این مثال، برای actor مشخص شده با self (که actor ID است)، یک service message ارسال می‌شود و برای همه actorها بجز Block، یک observe message ارسال می‌شود.

۷) انواع داده

تعریف متغیر در این زبان مانند زیر انجام می‌شود:

```
1 int studentId;  
2 int grades[10];  
3 string name;  
4 boolean isFemale;
```

در این زبان امکان تعریف چند متغیر در یک خط وجود ندارد.


در صورتی که به متغیرهای از جنس string، int، boolean و []int مقدار اولیه نسبت داده نشود، مقدار اولیه آنها برابر با مقدار پیش‌فرض تایپ خود در نظر گرفته می‌شود. مقادیر پیش‌فرض آنها به صورت جدول زیر است:

int	0
boolean	false

string	'''
int[]	مقدار تمام خانه ها برابر با 0 است.

تایپ‌های موجود در این زبان مطابق جدول بالا هستند. سه تایپ اول از نوع primitive هستند (خود مقادیر در آنها ذخیره میشوند نه پوینتری به خانه ای از حافظه) و تایپ آخر از نوع non-primitive است و در آن پوینتری به خانه‌ی از حافظه وجود دارد.

برای تعریف متغیر از نوع آرایه، مقداردهی آن و همچنین دسترسی به آن به شکل زیر عمل می‌کنیم:



```

1  int arr[10];
2  arr[0] = 7;
3  int x = arr[2];

```

لازم به ذکر است که اندازه ی یک آرایه تنها می‌تواند یک عدد ثابت بزرگتر از صفر باشد (دقت کنید عبارتی مثل 2+3 نیز که مقدار ثابت دارد، به عنوان اندازه‌ی آرایه قابل قبول نیست).

۸) عملگرها

عملگرها در زبان SOACT به چهار دسته ی عملگرهای حسابی، مقایسه ای، منطقی و تخصیص تقسیم میشوند.

۸-۱) عملگرهای حسابی

این دسته از عملگرها تنها روی اعداد عمل میکنند. لیست این عملگر ها در جدول زیر آمده است. در مثال های استفاده شده A را برابر با 20 و B را برابر با 10 در نظر بگیرید.

عملگر	شرکت‌پذیری	توضیح	مثال
+	چپ	جمع	$A+B=30$
-	چپ	تفریق	$A-B=10$
*	چپ	ضرب	$A*B=200$
/	چپ	تقسیم	$A/B=2$ $B/A=0$
mod	چپ	باقی‌مانده	$A\%B=10$
-	راست	منفی تک عمل‌وندی	$-A = -20$
-- و ++	راست	پیشوندی	--A
-- و ++	چپ	پسوندی	++A

۲-۸ عملگرهای مقایسه‌ای

این عملگرها وظیفه ی مقایسه را دارند؛ پس نتیجه ی آنها باید مقدار true یا false باشد. یعنی خروجی آنها یک boolean است.

توجه داشته باشید که عملوندهای عملگرهای < و > تنها از جنس اعداد صحیح هستند.

همچنین برای عملگر == و != نیز باید تایپ عملوندها یکسان باشند و در صورت آرایه بودن، اندازه ی آنها نیز برابر باشد. در غیر این صورت باید خطای کامپایل گرفته شود.

لیست عملگرهای مقایسه ای در جدول زیر آمده است. مقادیر A و B را همانند قبل در نظر بگیرید.

عملگر	شرکت‌پذیری	توضیح	مثال
==	چپ	تساوی	$(A == B) = \text{false}$
!=	چپ	عدم تساوی	$(A != B) = \text{true}$
<	چپ	کوچکتر	$(A < B) = \text{false}$
>	چپ	بزرگتر	$(A > B) = \text{true}$

۸-۳) عملگرهای منطقی

در این زبان، عملگرهای منطقی تنها روی تایپ Boolean قابل اعمال است. این عملگرها در جدول زیر آمده است. A را برابر true و B را برابر false در نظر بگیرید.

عملگر	شرکت‌پذیری	توضیح	مثال
&&	چپ	عطف منطقی	$(A \&\& B) = \text{false}$
	چپ	فصل منطقی	$(A B) = \text{true}$
!	راست	نقیض منطقی	$(!A) = \text{false}$

۵-۸) عملگر تخصیص

این عملگر که به صورت = نمایش داده میشود وظیفه ی تخصیص را برعهده دارد. یعنی مقدار عملوند سمت راست را به عملوند سمت چپ اختصاص میدهد. برای آرایه مقدار تک تک عناصر آرایه ی سمت راست را به عناصر آرایه ی سمت چپ تخصیص میدهد. دقت شود که برای استفاده از این عملگر برای دو آرایه، اندازه ی آرایه ها باید برابر باشد.

همچنین دقت داشته باشید که عملوند سمت چپ باید از نوع left-value باشد. عبارات left-value عباراتی هستند که به یک مکان در حافظه اشاره میکنند. در مقابل عبارات right-value به مکان خاصی در حافظه اشاره نمیکنند و صرفاً یک عبارت دارای مقدار هستند. به عنوان مثال یک متغیر یا یک دسترسی به یکی از عناصر آرایه یک عبارت left-value است اما عبارت 3 + 10 یک عبارت right-value محسوب میشود. عبارات right-value تنها در سمت راست عملگر تخصیص قرار میگیرند.

۶-۸) اولویت عملگرها


اولویت عملگرها طبق جدول زیر است:

اولویت	دسته	عملگرها	شرکت پذیری
1	عملگر pipe	<	چپ به راست
2	پرانتز	()	چپ به راست
3	دسترسی به عناصر آرایه	[]	چپ به راست
4	تک عملوندی پسوندی	++ --	چپ به راست
5	تک عملوندی پیشوندی	-- ++ ! -	راست به چپ
6	ضرب و تقسیم و باقی مانده	*/%	چپ به راست

چپ به راست	+ -	جمع و تفریق	7
چپ به راست	< >	رابطه ای	8
چپ به راست	== !=	مقایسه ی تساوی	9
چپ به راست	&&	عطف منطقی	10
چپ به راست		فصل منطقی	11
راست به چپ	=	تخصیص	12
چپ به راست	,	کاما(ورودی مسج هندلرها)	13

۹) گزاره‌های شرطی

در زبان SOACT تنها ساختار شرطی، if...else است. ساختار نحوی آن مشابه زیر است:



```

1  if (arr[i] > max) {
2      max = arr[i];
3  } else {
4      i++;
5  }

```

این ساختار بدون else نیز می‌تواند بکار رود.

۱۰) توابع پیش فرض

توابع پیش فرض به توابعی اطلاق می شود که message handler های آنها به طور پیش فرض موجود هستند و نیازی به تعریف یا پیاده سازی آنها توسط برنامه نویسنده نیست و از ابتدای اجرای برنامه در حال پردازش پیام ها هستند.

toLowerCase()	این تابع برای تبدیل حروف یک رشته به حروف کوچک استفاده می شود. این پردازشگر هیچ پارامتری نمی پذیرد و مستقیماً روی یک رشته اعمال می شود.
toUpperCase()	این تابع تمام حروف یک رشته را به حروف بزرگ تبدیل می کند.
reverse()	این تابع ترتیب حروف یک رشته را معکوس می کند و رشته معکوس شده را بازمی گرداند.
print()	این تابع برای چاپ مقادیر روی کنسول استفاده می شود. این پردازشگر ورودی های مختلفی مانند رشته، عدد و ... را می پذیرد و آنها را چاپ می کند.
main()	نقطه ورود (entry point) اصلی برنامه است. زمانی که برنامه اجرا می شود، تمامی کدهایی که درون آن قرار دارند، اجرا می شوند.
add()	این تابع برای افزودن یک عنصر به یک مجموعه (Set) یا لیست (List) استفاده می شود.

include()	این تابع بررسی می‌کند که آیا یک عنصر خاص در یک مجموعه (Set) یا لیست (List) وجود دارد یا خیر
remove()	این تابع برای حذف کردن یک عنصر از یک مجموعه (Set) یا لیست (List) استفاده می‌شود.
length()	این تابع طول یک لیست را به ما بر می‌گرداند.
private()	این تابع برای محدود کردن دسترسی به یک متغیر استفاده می‌شود. به این معنا که فقط actorهای خاصی، مانند actorی که خود متغیر را تعریف کرده است (مثل self)، می‌توانند به آن متغیر دسترسی داشته باشند.
public()	این تابع برای تعیین دسترسی عمومی به یک متغیر استفاده می‌شود. به این معنا که actorهای دیگر نیز می‌توانند به آن متغیر دسترسی داشته باشند.

۱۱) حلقه‌ها (Loops)

در زبان SOACT، دو نوع ساختار تکرار وجود دارد:

For Loop(۱-۱۱)

- ساختار نحوی آن به این شکل است که می‌توان از range یا مجموعه‌هایی مانند Set برای انجام تکرار استفاده کرد.

While Loop(۲-۱۱)

- این حلقه تا زمانی که شرط تعیین‌شده برقرار باشد، تکرار می‌شود.
 - شرط این حلقه باید از نوع boolean باشد و زمانی که شرط برقرار نباشد، حلقه متوقف می‌شود.
- در هر دو نوع حلقه می‌توان از دستورات break و continue برای کنترل جریان استفاده کرد.
- تمامی متغیرها باید در ابتدای برنامه تعریف شوند و امکان تعریف متغیر داخل for یا while وجود ندارد.

Join-Block (۱۲)

در زبان SOACT ، Join Block به شما این امکان را می‌دهد که مجموعه‌ای از عملیات‌ها یا دستورات را به شکل هم زمان و concurrency-safe اجرا کنید. یعنی، کدهایی که درون یک join block قرار می‌گیرند، یک سری محاسبات ریاضی هست که در پردازش جدا به صورت موازی انجام می‌شود (parallel computation) و در نهایت از خروجی آن استفاده می‌شود، به عبارتی در اسکوپ از برنامه قرار نیست این محاسبات انجام شوند.

```
1 join { % meow just computation and new GreetingActor("Hello", 3) can not be here
2
3     % Send a message to execute the sendGreeting handler
4     myGreeter.sendGreeting();
5
6     % Update the greeting and resend the message
7     myGreeter.updateGreeting("Hi");
8     myGreeter.sendGreeting();
9 }
```

۱۳) عملگر Pipe (>|)

```
1 msgRcv processString(string input, string suffix) {
2     result = input;
3     join {
4         result = result % the output of the previous goes into the input of the next
5         |> self.toLowerCase()
6         |> self.inverse()
7         |> self.addSuffix(suffix);
8     }
9     print("Processed string: " + result);
10 }
```

این عملگر برای اجرای زنجیره‌ای عملیات‌ها استفاده می‌شود. نکته قابل توجه در مورد این عملگر این است که فقط در join block استفاده می‌شود و خروجی هر مرحله به عنوان ورودی مرحله بعدی استفاده می‌شود. این عملگر به شما امکان می‌دهد که چندین عملیات را به صورت chained اجرا کنید، این تکنیک کد را خواناتر و ساده‌تر می‌کند، به خصوص زمانی که عملیات متوالی روی یک داده انجام می‌دهیم.

۱۴) public و private

این دو پردازشگر پیش فرض، دو مجموعه را به عنوان ورودی می‌گیرند و آن‌ها را با هم اجتماع می‌گیرند. پردازشگر private نتیجه را include می‌کند و پردازشگر public نتیجه را exclude می‌کند. در ادامه این روش را که به شما کمک می‌کند تا کنترل بیشتری بر دسترسی به داده‌ها داشته باشید، توضیح می‌دهیم.

۱-۱۴) منطق private

پردازشگر private برای محدود کردن دسترسی به داده‌ها طراحی شده است. این پردازشگر به این معناست که فقط self (کاربر یا شیء) و افرادی که اجازه دسترسی دارند (مثلاً دنبال‌کنندگان) می‌توانند به نتیجه دسترسی پیدا کنند. به عبارت دیگر، نتیجه این پردازشگر فقط برای خود کاربر و افرادی که به

طور خاص مجاز هستند، قابل مشاهده است. بنابراین در این پردازشگر ، فقط کاربر و افراد مجاز می‌توانند به نتیجه دسترسی داشته باشند (افراد مجاز را include می‌کند).

۲-۱۴ منطق public

پردازشگر public به همه افراد اجازه می‌دهد که به نتیجه دسترسی داشته باشند، اما می‌تواند محدودیت‌هایی را نیز ایجاد کند. این پردازشگر به معنای آن است که نتیجه برای تمامی افراد قابل مشاهده است، به جز افرادی که در لیست مسدود شده‌اند. بنابراین، پردازشگر public می‌تواند به صورت عمومی داده‌ها را به اشتراک بگذارد، اما با مشخص کردن اینکه چه کسانی نمی‌توانند به آن دسترسی پیدا کنند. بنابراین نتیجه برای همه قابل مشاهده است، به جز افرادی که مشخص شده‌اند نمی‌توانند به آن دسترسی پیدا کنند (افراد مشخص شده را exclude می‌کند).

دقت کنید public و private می‌توانند به صورت تو در تو نیز استفاده بشوند:



```
1 (private(friends, private(self,Followers)))
```

۱۵ تعریف CustomPrimitive

Custom Primitives در زبان برنامه‌نویسی SOACT، یک نوع داده‌ای است که به شما اجازه می‌دهد مجموعه‌ای از مقادیر ثابت و از پیش تعریف‌شده را تعریف کنید. این مقادیر معمولاً به صورت اسامی توصیفی هستند که به اعداد صحیح نگاشت می‌شوند. استفاده از Custom Primitives باعث می‌شود کد خواناتر و قابل فهم‌تر باشد، به‌ویژه زمانی که با مقادیر ثابت سروکار دارید.

برای مثال Custom Primitive زیر را در نظر بگیرید:



```
1 primitive DoorState {  
2     OpenedDoor,  
3     ClosedDoor  
4 }
```

که به صورت زیر از آن استفاده می‌کنیم:



```
1 doorState = DoorState::ClosedDoor;
```

۱۶) تعریف Record

Record یک ساختار داده‌ای است که به شما اجازه می‌دهد چندین نوع داده مختلف را تحت یک نام گرد هم آورید. با استفاده از Record می‌توانید متغیرهای مختلف (که ممکن است انواع داده‌ای متفاوتی داشته باشند) را در قالب یک واحد تعریف کنید.

برای مثال Record زیر را در نظر بگیرید:



```
1 Record Person {  
2     String name;  
3     int age;  
4     String address;  
5 }
```


که برای مقداردهی آن به صورت زیر عمل می‌کنیم:



```
1 Person person = Person{name: "Reza", age: 32, address: "address"};
```

۱۷) کلیدواژه self

این کلیدواژه، به actor ای که در آن قرار داریم اشاره می‌کند. در این زبان از self برای بدست آوردن ID اکتورها استفاده می‌کنیم تا بتوانیم service message یا observe message های خود را برای actor های مدنظر ارسال کنیم.