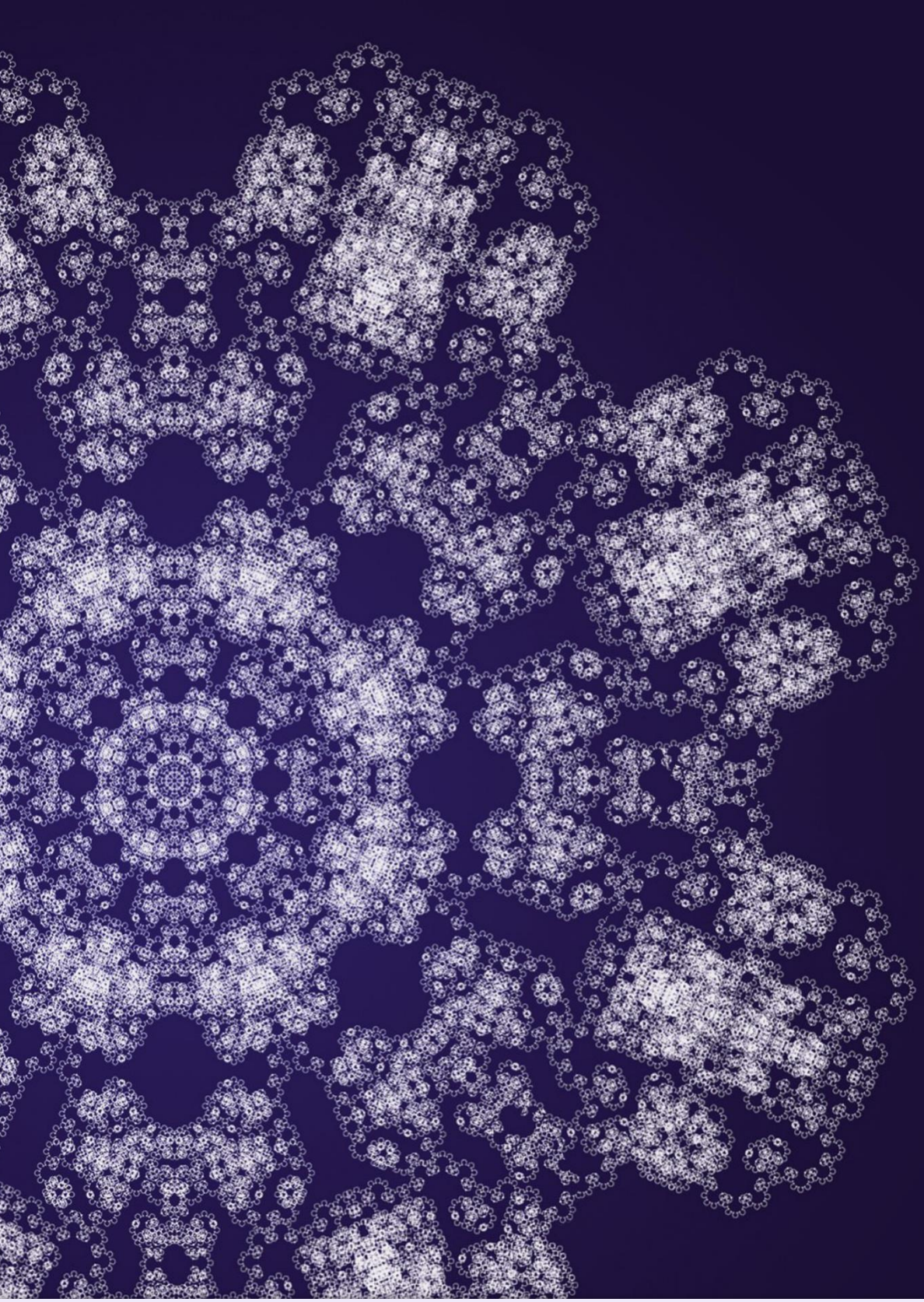

سند زبان برنامه‌نویسی

Atalk

نسخه‌ی ۲٫۲

فهرست مطالب

۱. مقدمه

زبان Atalk[۱] یک زبان برنامه‌نویسی مبتنی بر مدل محاسباتی اکتور[۲] است. در این زبان اکتورها عوامل محاسباتی همروند[۳] هستند که هیچ حالت مشترکی ندارند و ارتباط بین آن‌ها تنها توسط تبادل ناهمگام[۴] پیام امکان‌پذیر است. هر اکتور شامل یک ریسمان[۵] اجرا، صندوق پیام[۶] و نام منحصر به فرد و یکتاست. اکتورها به صورت دائمی پیامی را از صندوق پیام‌های خود بر می‌دارند و متعاقباً نسبت به آن واکنش می‌دهند. این واکنش‌ها می‌تواند به صورت یکی از فعالیت‌ها زیر باشد:

- ارسال پیام به بقیه‌ی اکتورها
- تغییر وضعیت کنونی[۷]

در این زبان کد برنامه درون یک فایل با پسوند atk. قرار دارد. این فایل شامل یک یا چند اکتور است و هر اکتور شامل تعدادی متغیر[۸] (که مشخص‌کننده‌ی وضعیت آن اکتور است) و گیرنده[۹] (که وظیفه‌ی رسیدگی به یک پیام خاص را دارد) است.

[1] Asynchronous Talk
[2] Actor Oriented
[3] Concurrent
[4] Asynchronous
[5] Thread

البته امکان پیاده‌سازی تک‌ریسمانه ولی همروند این مدل زبان نیز وجود دارد.

[6] Mail Box

[۷] ساخت اکتور جدید نیز از مهم‌ترین واکنش‌های هر اکتور است که در این زبان به جهت ساده‌سازی حذف شده‌است.

[8] Variable
[9] Receiver

۲. ساختار کلی

یک برنامه به زبان Atalk از قسمت‌های زیر تشکیل شده‌است:

- اکتور
  - متغیر
  - گیرنده
- کامنت

قطعه کد ۱

```
# Actors
actor Adder<10>
    # Variables
    int addsCount

    # Receivers
    receiver init()
        addsCount = 0
    end

    receiver add(int x, int y)
        addsCount = addsCount + 1
        sender << addCompleted(x + y)
    end
end

actor Runner<1>
    receiver init()
        self << run()
    end

    receiver run()
        Adder << add(2, 3)
    end

    receiver addCompleted(int result)
        write(result)
    end
end
```

در ابتدای اجرا در صندوق پیام همه‌ی اکتورها پیام `init` وجود دارد. اکتور `Runner` هنگام بررسی پیام `init` به خودش پیام `run` می‌فرستد. هنگام رسیدگی به پیام `run` در اکتور `Runner`، به اکتور `Adder` پیام `add` با پارامترهای ۲ و ۳ فرستاده می‌شود. اکتور `Adder` نیز پس از بررسی پیام `init`، هنگام رسیدگی به این پیام وضعیت (متغیر) `addsCount` را یکی افزایش می‌دهد و به فرستنده‌ی پیام (`sender`)، پیام `addCompleted` با محتوای جمع دو عدد را می‌فرستد. اکتور `Runner` نیز پس از دریافت پیام `addCompleted`، نتیجه را در کنسول چاپ می‌کند.

۲-۱. قواعد کلی نحو

زبان Atalk به بزرگ و کوچک بودن حروف حساس است[10]. در این زبان وجود کاراکترهای Tab[11] و Space[12] تاثیری در خروجی برنامه ندارند. خطوط برنامه نیز با استفاده از کاراکتر New Line[13] از یکدیگر جدا می‌شوند. جزئیات مربوط به Scopeها و خطوط برنامه در ادامه به طور مفصل توضیح داده می‌شوند.

قطعه کد ۲

```
# Actors
actor Adder<5>
    # Variables
    int addsCount

    # Receivers
    receiver init()
        addsCount = 0
    end

    receiver add(int x, int y)
        write(x + y)
        addsCount = addsCount + 1
    end
end
```

خطوط خالی (شامل Whitespace یا کامنت) تاثیری در خروجی برنامه ندارد.

۲-۲. کامنت‌ها

در این زبان کامنت‌ها تنها تک‌خطی هستند و تمامی کاراکترهای بعد از #[14] تا انتهای خط کامنت حساب می‌شوند و هیچ تاثیری در خروجی ندارند.

قطعه کد ۳

```
# Comment
actor Adder<4>
    # Comment
    int addsCount # Number of times add message received
    # int commentedVariable
end
```

[10] Case Sensitive
[11] '\t' – ASCII #9
[12] ' ' – ASCII #32
[13] '\n' – ASCII #10
[14] Hash Sign – ASCII #35

۲-۳. قواعد نام‌گذاری اکتورها، گیرنده‌ها و متغیرها

اسامی انتخابی برای نام‌گذاری اکتورها، گیرنده‌ها و متغیرها باید از قواعد زیر پیروی کنند:

- تنها از کاراکترهای a..z، A..Z، _ و ارقام تشکیل شده باشند.
- با رقم شروع نشوند.
- معادل کلیدواژه‌ها نباشند. در جدول زیر تمام کلیدواژه‌های زبان Atalk آمده‌است:

actor	receiver	int
char	quit	foreach
break	if	elseif
else	sender	self
in	begin	end
read	write	

- نام هر اکتور یکتاست.
- هر گیرنده‌ی داخل اکتور با استفاده از نام و نوع ورودی‌هایش یکتاست. به این معنی که می‌توان گیرنده‌ها را سربارگذاری[15] کرد.

قطعه کد ۴

```
actor Adder<4>
    receiver add(int x, int y)
        write(x + y)
    end
    receiver add(int x, int y, int z)
        write(x + y + z)
    end
end
```

مثال از سربارگذاری گیرنده‌ها

- نام متغیرهای داخل یک Scope یکتاست امّا می‌توان در Scopeهای درونی‌تر از نام‌های متغیرهای بیرونی استفاده کرد که درنتیجه در طول آن Scope متغیر درونی هنگام استفاده ارجعیت دارد.

[15] Overload

۳. اکتور

در زبان Atalk هر اکتور یک عامل محاسباتی است که شامل تعدادی گیرنده است که وظیفه‌ی رسیدگی یک نوع پیام خاص را دارد. هیچ اکتوری امکان دسترسی به متغیرهای یک اکتور دیگر را به طور مستقیم ندارد و ارتباط آن‌ها تنها از طریق ارسال پیام به یکدیگر امکان‌پذیر است.

۳-۱. تعریف اکتور

یک اکتور به شکل زیر تعریف می‌شود:

قطعه کد ۵

```
actor ActorName<4> # Actor definition
    # Actor variables(states) and receivers
end
```

توجه نمایید که شروع تعریف اکتور (خط ۱) و انتهای آن (خط ۳) هر کدام باید در یک خط جداگانه باشند و هیچ کد اجرایی (غیر کامنت و Whitespace) در این خطوط نباید قرار بگیرد. در واقع مواردی که با پس‌زمینه‌ی طوسی در کد بالا مشخص شده‌اند باید هر یک در خطی جداگانه باشند و محتوای آن‌ها نباید در دو یا چند خط قرار گیرد.

کدهای زیر هر یک از لحاظ نحوی اشتباه هستند:

قطعه کد ۶

```
actor
ActorName<4>
end
```

قطعه کد ۷

```
actor ActorName<4> int var
end
```

قطعه کد ۸

```
actor ActorName<4> end
```

قطعه کد ۹

```
actor FirstActor<4>
    int var
end actor SecondActor<5>
end
```

۳-۲. محدودیت تعداد پیام‌های پردازش نشده

تعداد پیام‌های پردازش نشده‌ای که در صندوق پیام هر اکتور است محدود می‌باشد و مقدار آن پس از نام آن اکتور درون <> معلوم می‌شود. اگر تعداد پیام‌های پردازش نشده‌ی یک اکتور از این تعداد بیشتر شود پیام‌های بعدی که به آن فرستاده می‌شود drop می‌شوند. این مقدار باید بیشتر از صفر باشد در غیر اینصورت خطای کامپایل گرفته می‌شود.

۳-۳. پایان برنامه

یک برنامه در زبان Atalk زمانی به پایان می‌رسد که در صندوق پیام هیچ اکتوری پیامی نباشد.

۴. گیرنده

هر اکتور شامل تعدادی گیرنده است که وظیفه‌ی رسیدگی به پیام مربوطه را دارد. هنگام رسیدگی به یک پیام، آن پیام از صندوق خارج می‌شود و کد مربوط به آن گیرنده اجرا می‌شود.

توجه نمایید که گیرنده‌ها تابع نیستند و به صورت ناهمگام اجرا می‌شوند. به این معنی که وقتی کد A << message(arg1, arg2)[۱۶] اجرا می‌شود صرفاً یک پیام به اکتور A فرستاده می‌شود و اکتور کنونی برای جواب آن نمی‌ایستد (بلاک نمی‌شود).

۴-۱. تعریف گیرنده

گیرنده برای هر اکتور به شکل زیر تعریف می‌شود:

قطعه کد ۱۰

```
1  actor ActorName<4> # Actor definition
2      receiver receiverName(int arg1, char arg2) # Receiver definition
3          # Receiver body
4      end
5  end
```

همانند تعریف اکتور، شروع تعریف گیرنده (خط ۲) و انتهای آن (خط ۴) که با پس‌زمینه‌ی طوسی مشخص شده‌اند باید در یک خط جداگانه باشند و کد اجرایی در این خطوط نباید قرار گیرد و محتوای آن‌ها نباید در چند خط قرار بگیرد.

۴-۲. پارامترهای گیرنده

پارامترهای دریافتی هر گیرنده از سمت فرستنده کپی می‌شوند و تغییر آن‌ها تاثیری بر روی متغیرهای اصلی ندارد[۱۷].

۴-۲. گیرنده‌ی init

در ابتدای اجرای برنامه، داخل صندوق پیام هر اکتور پیام init وجود دارد، اما الزامی در وجود این گیرنده در اکتورها نیست. همچنین امکان ارسال پیام init به اکتوری که گیرنده‌ی init دارد امکان‌پذیر است. توجه نمایید که در گیرنده‌ی init امکان استفاده از کلمه‌ی کلیدی sender[۱۸] وجود ندارد.

---

[۱۶] عملگر << در ادامه به طور مفصل توضیح داده می‌شود.

[17] Copy by value

[۱۸] در ادامه راجع به کلمه‌ی کلیدی sender توضیح داده می‌شود

۳-۴. خروج از گیرنده (کلمه‌ی کلیدی quit)

امکان خروج از یک گیرنده با کلمه‌ی کلیدی quit وجود دارد:

قطعه کد ۱۱

```
actor Math<10>
    receiver divide(int x, int y)
        if y == 0
            quit
        else
            sender << divisionCompleted(x / y)
        end
    end
end
```

۵. انواع داده

۵-۱. تایپ‌های پایه

عدد: تنها نوع اعداد در زبان Atalk، اعداد صحیح می‌باشند که آن‌ها را با int نمایش می‌دهیم.
کاراکتر: یک کاراکتر ASCII را در این زبان با char نشان می‌دهیم. کاراکترها را در [19] ' ' قرار می‌دهیم. مانند 'c'.

۵-۲. آرایه‌ها

در زبان Atalk امکان تعریف آرایه‌هایی با عناصر **عدد** و **کاراکتر** و **آرایه** (جهت پیاده‌سازی آرایه‌های چندبعدی) وجود دارد. طول آرایه‌ها در این زبان ثابت است و در زمان کامپایل مشخص می‌شود. برای دسترسی به اعضای آرایه می‌توان از عملگر [20][ ] استفاده کرد که داخل آن یک عدد قرار می‌گیرد. اندیس شروع آرایه‌ها نیز صفر می‌باشد[21]. توجه نمایید که تعریف آرایه با طول صفر یا منفی غیرمجاز است. دسترسی به عناصر با اندیس‌های منفی یا بزرگترمساوی طول آرایه در زمان اجرا خطا می‌دهد.

[19] Single Quotation Mark – ASCII #39
[20] Bracket
[21] Zero-Based Index

قطعه کد ۱۲

```
actor Math<4>
    receiver calcDeterminant(int[2][2] mat)
        int determinant = mat[0][0] * mat[1][1] - mat[0][1] * mat[1][0]
        write(determinant)
    end
    receiver calcSum(int[4] vec)
        int sum = vec[0] + vec[1] + vec[2] + vec[3]
        write(sum)
    end
end
```

مثال از آرایه‌ها

نکته: امکان پیمایش آرایه با استفاده از foreach نیز در این زبان وجود دارد که در ادامه توضیح داده می‌شود.

امکان مقداری دهی سریع آرایه به شکل زیر وجود دارد:

```
char[3] oneDimCharArray = {‘a’, ‘b’, ‘c’}
char[2][3] twoDimCharArray = {{‘a’, ‘b’, ‘c’}, {‘d’, ‘e’, ‘f’}}
char[4] string = “abcd”
int[2] oneDimIntArray = {1, 2}
```

توجه نمایید که “abcd” معادل {‘a’, ‘b’, ‘c’, ‘d’} است.

۶. متغیرها

نحوه‌ی تعریف متغیرها به شکل زیر می‌باشد:

```
type variableName
type variableName  = expression
```

امکان تعریف چند متغیر پشت سر هم و مقداردهی اولیه آن‌ها نیز وجود دارد:

```
type var1 = expr1, var2 = expr2, var3, var4, var5 = expr3
```

توجه نمایید که امکان مقداردهی اولیه‌ی متغیرهای سراسری در تعریف آن‌ها وجود ندارد اما امکان تعریف چند متغیر سراسری پشت سر هم (به شکل بالا) وجود دارد.

در صورتی که یک متغیر مقداردهی نشده باشد مقدار آن برابر مقدار پیش‌فرض می‌شود:

مقدار پیش‌فرض	تایپ
0	int
‘\0’	char
هر عنصر مقدار پیش فرض مربوط به تایپ پایه را دارد	آرایه

۷. عملگرها

عملگرها در زبان Atalk به پنج دسته‌ی عملگرهای حسابی[۲۲]، مقایسه‌ای[۲۳]، منطقی[۲۴]، عملگر تخصیص[۲۵] و عملگر ارسال (یا گفتن)[۲۶] تقسیم می‌شوند.

۷-۱. عملگرهای حسابی

این دسته از عملگرها تنها روی اعداد عمل می‌کنند، لیست این عملگرها در جدول زیر آمده است. در مثال‌های استفاده شده A برابر 20 و B را برابر 10 در نظر بگیرید:

عملگر	شرکت‌پذیری	توضیح	مثال
+	چپ	جمع	A+B=30
-	چپ	تفریق	B-A=-10
*	چپ	ضرب	A*B=200
/	چپ	تقسیم	A/B=2 B/A=0
-	راست	منفی تک‌عملوندی	-A=-20

۷-۲. عملگرهای مقایسه‌ای

این عملگرها وظیفه‌ی مقایسه را دارند، پس نتیجه‌ی آن‌ها باید مقدار صحیح یا غلط باشد. در زبان Atalk به صورت قراردادی اعداد غیر صفر معادل مقدار صحیح و صفر را معادل غلط در نظر می‌گیریم. با این حساب خروجی این عملگرها یک عدد صحیح است. توجه داشته باشید که عملوند عملگرهای > و < تنها از جنس عدد صحیح هستند. همچنین برای عملگرهای == و <> نیز باید تایپ عملوندها یکسان باشند و در صورت آرایه بودن، اندازه‌ی آن‌ها نیز برابر باشد؛ در غیر اینصورت باید خطای کامپایل گرفته شود. لیست عملگرهای مقایسه‌ای در جدول زیر آمده است. در مثال‌های استفاده شده مقدار A را برابر 20 و مقدار B را برابر 10 بگیرید:

عملگر	شرکت‌پذیری	توضیح	مثال
==	چپ	تساوی	(A==B)=0 (false)
<>	چپ	عدم تساوی	(A<>B)=1 (true)
<	چپ	کوچکتر	(A<B)=0 (false)
>	چپ	بزرگتر	(A>B)=1 (true)

[22] Arithmetic
[23] Relational
[24] Logical
[25] Assignment
[26] Tell

۷-۳. عملگرهای منطقی

در زبان Atalk عملیات منطقی تنها روی اعداد صحیح قابل اعمال است.این عملگرها در جدول زیر لیست شده‌اند. در مثال‌های استفاده شده A را برابر 5، B را برابر 0 و C را برابر 10- بگیرید:

عملگر	شرکت‌پذیری	توضیح	مثال
and	چپ	عطف منطقی	(A and B) =0 (false)
or	چپ	فصل منطقی	(B or C) =0 (false) (A or B or C) =1 (true)
not	راست	نقیض منطقی	(not A) =0 (false)

۷-۴. عملگر تخصیص

این عملگر که به صورت = نمایش داده می‌شود وظیفه‌ی تخصیص را بر عهده دارد. عملگر تخصیص مقدار عملوند سمت راست را به عملوند سمت چپ اختصاص می‌دهد (برای آرایه‌ها مقدار تک‌تک عناصر عملوند سمت راست به عناصر متناظر سمت چپ تخصیص می‌یابد). مقدار خروجی این عملگر برابر با مقدار تخصیص داده شده به عملوند سمت چپ آن است. به عنوان مثال در برنامه‌ی زیر مقدار نهایی c برابر 30، مقدار نهایی x برابر 35 و مقدار نهایی x و y برابر 40 خواهد بود.

```
int a = 10, b = 20
int c, x, y
y = x = (c = a + b) + 5
```

دقت داشته باشید که عملوند سمت چپ باید حتماً از نوع lvalue باشد. مفهوم lvalue و rvalue در زبان Atalk مشابه زبان C است. عبارات lvalue عباراتی هستند که به یک مکان در حافظه اشاره می‌کنند، در مقابل عبارات rvalue به مکان خاصی در حافظه اشاره نمی‌کنند و صرفاً یک عبارت دارای مقدار هستند. به عنوان مثال یک متغیر یک عبارت lvalue است اما عبارت 10+30 یک عبارت rvalue محسوب می‌شود. در زبان Atalk عبارات rvalue تنها می‌توانند سمت چپ عملگر تخصیص قرار بگیرند.

```
int g = 20, f = 5 # Valid Assignment
g + 10 = 30 # Invalid Assignment
(g = f + 5) = 50 # Invalid Assignment
```

۷-۵. عملگر ارسال (گفتن)

عملگر ارسال که با علامت >> مشخص می‌شود برای ارسال یک پیام به یک اکتور به کار می‌رود. این اکتور می‌تواند فرستنده‌ی پیام (sender)، اکتور کنونی (self) یا یک اکتور دیگر (که با نام آن مشخصی‌شود) باشد. نحوه‌ی استفاده از آن به شکل زیر است:

قطعه کد ۱۳

```
actor A<1>
    receiver ping()
        write(“ping received”)
        sender << pong()
    end
end
actor B<1>
    receiver init()
        A << ping()
    end
    receiver pong()
        write(“pong received”)
        sender << ping()
    end
end
```

مثال از عملگر ارسال

توجه نمایید در صورتی که اکتوری که به آن پیامی ارسال می‌شود گیرنده‌ی آن را نداشته باشد در صورت ذکر نام اکتور مقصد یا استفاده از کلمه‌ی کلیدی self باید خطای زمان کامپایل گرفته شود. اما در صورت استفاده از کلمه‌ی کلیدی sender برای اکتور مقصد خطا می‌تواند در زمان کامپایل یا زمان اجرا گرفته شود. به مثال‌های زیر توجه کنید:

قطعه کد ۱۴

```
actor A<1>
    receiver ping()
        write(“ping received”)
        sender << pong()
    end
end
actor B<1>
    receiver init()
        A << invalidReceiver() # Compile Error
    end
    receiver pong()
        write(“pong received”)
        sender << ping()
    end
end
```

اکتور A دارای گیرنده‌ی invalidReceiver نیست. بنابراین خط شماره‌ی ۹ دارای خطای کامپایل است.

قطعه کد ۱۵

```
actor A<1>
    receiver ping()
        write(“ping received”)
        sender << invalidReceiver()
    end
end
actor B<1>
    receiver init()
        A << ping()
    end
    receiver pong()
        write(“pong received”)
        sender << ping()
    end
end
```

در گیرنده init از اکتور B، گیرنده‌ی ping از اکتور A استفاده شده‌است. در گیرنده‌ی ping از اکتور A نیاز به گیرنده‌ی invalidReceiver از فرستنده (sender) وجود دارد در حالی که اکتور B دارای این گیرنده نیست. پس یا در خط شماره‌ی ۹ هنگام کامپایل باید خطا گرفته شود و یا هنگام بررسی پیام‌ها توسط اکتور B و با مشاهده‌ی پیام invalidReceiver در صندوق پیام آن باید خطای زمان اجرا گرفته شود.

قطعه کد ۱۶

```
actor A<1>
    receiver ping()
        write(“ping received”)
        sender << pong()
    end
end
actor B<1>
    receiver init()
        A << ping()
    end
    receiver pong()
        write(“pong received”)
        sender << invalidReceiver()
    end
end
```

در گیرنده‌ی init از اکتور B، گیرنده‌ی ping از اکتور A استفاده شده‌است. در گیرنده‌ی ping از اکتور A نیاز به گیرنده‌ی pong از فرستنده (sender) که در این جا B است وجود دارد. در گیرنده‌ی pong از اکتور B نیاز به invalidReceiver از اکتور فرستنده (sender) که در این جا A است وجود دارد در حالی که در اکتور A گیرنده‌ی invalidReceiver وجود ندارد. پس یا باید در خط ۹ خطای کامپایل گرفته شود و یا اکتور A هنگامی که پیام invalidReceiver را در صندوق پیام خود می‌بیند باید خطای زمان اجرا گرفته شود.

۶-۷. اولویت عملگرها

اولویت عملگرها طبق جدول زیر است:

اولویت	دسته	عملگرها	شرکت‌پذیری
۱	پرانتز	()	چپ
۲	دسترسی به عناصر آرایه	[]	چپ
۳	تک عملوندی	- not	راست
۴	ضرب و تقسیم	*/	چپ
۵	جمع و تفریق	+-	چپ
۶	رابطه‌ای	< >	چپ
۷	مقایسه‌ی تساوی	== <>	چپ
۸	عطف منطقی	and	چپ
۹	فصل منطقی	or	چپ
۱۰	تخصیص	=	راست
۱۱	کاما	,	چپ به راست
۱۲	ارسال	<<	-

۸. ساختار تصمیم‌گیری

در زبان Atalk تنها ساختار تصمیم‌گیری if..elseif...else می‌باشد:

```
if exp1
    # statement(s) will execute if the exp1 is true
elseif exp2
    # statement(s) will execute if the exp1 is false and exp2 is true
elseif exp3
    # statement(s) will execute if the exp1 & exp2 are false and ...
...
else
    # statement(s) will execute if all the expressions are false
end
```

همانطور که پیش‌تر توضیح داده‌شد به دلیل عدم وجود تایپ‌های صحیح/غلط در این زبان، عبارات با مقدار عددی غیرصفر را معادل صحیح و عبارات با مقدار عددی صفر را غلط در نظر می‌گیریم. لازم به ذکر است که ساختار if می‌تواند بدون elseif و else نیز استفاده گردد.

توجه نمایید که موارد با پس‌زمینه‌ی طوسی رانگ باید هر یک در خطی جداگانه باشند و محتوای آن‌ها نمی‌تواند در دو (یا چند) خط قرار بگیرد.

۹. ساختار تکرار

تنها ساختار تکرار در این زبان foreach می‌باشد که امکان پیمایش بر روی یک آرایه را امکان‌پذیر می‌کند. امکان خروج از تکرار با استفاده از کلمه‌ی کلیدی break وجود دارد. اگر یک خط داخل چندین foreach تودرتو باشد break باعث خروج از درونی‌ترین آن می‌شود.

مثال زیر نحوه‌ی استفاده از این ساختار را نشان می‌دهد:

قطعه کد ۱۷

```
actor Math<10>
    receiver calcSumUpToZero(int[10] array)
        int sum = 0
        foreach element in array
            if element <> 0
                sum = sum + element
            else
                break
            end
        end
        sender << calcSumUpToZeroCompleted(sum)
    end
end
```

به موارد زیر در مورد کد بالا توجه نمایید:

۱. متغیر element، rvalue می‌باشد و قابل تخصیص نمی‌باشد.
۲. نوع element نیز متناسب با عناصر آرایه‌ای است که در حال پیمایش است.
۳. همانند موارد قبلی قسمت‌هایی که با پس‌زمینه‌ی طوسی مشخص شده‌اند هر یک باید در خطی جدا باشند و محتوای آن‌ها نمی‌توانند در دو خط قرار بگیرند.

۱۰. قوانین Scopeها و خطوط

۱۰-۱. کلمات کلیدی begin و end

در زبان Atalk می‌توان با استفاده از کلمات کلیدی begin و end یک Scope جدید تعریف کرد. توجه نمایید که تنها می‌توان از این کلمات داخل بدنه‌ی گیرنده استفاده نمود.

۱۰-۲. Scopeهای موجود در زبان

به طور کلی در زبان Atalk، موارد زیر در Scope جدیدی قرار دارند:

۱. خطوط کد داخل یک اکتور
۲. خطوط کد بین دو کلمه‌ی کلیدی begin و end پشت سر هم.
۳. پارامترها و خطوط کد داخل گیرنده
۴. متغیر foreach و خطوط کد داخل آن
۵. Expression مورد بررسی ساختار if (یا elseif) و خطوط کد داخل آن – خطوط کد داخل بدنه‌ی else

۱۰-۲. قوانین Scopeها

نکات زیر نیز در مورد Scopeها وجود دارد:

- تعریف اکتورها در بیرونی‌ترین Scope است.
- خطوط خالی از کد اجرایی هیچ تاثیری در خروجی و اجرای برنامه ندارد.
- کدهای داخل هر گیرنده در Scope آن گیرنده هستند.
- متغیرهایی که داخل یک Scope تعریف می‌شوند در Scopeهای بیرون آن دسترس‌پذیر نیستند و صرفاً در Scopeهای درون آن قابل دسترسی هستند.
- امکان تعریف متغیر با نام یکسان در یک Scope وجود ندارد امّا در Scopeهای درونی آن امکان تعریف مجدد وجود دارد و تا زمان خروج از Scope درونی، نزدیکترین تعریف به آن استفاده می‌شود.

قطعه کد ۱۸

```
actor Program<10>
    char var
    receiver init()
        char var = ‘1’
        begin
            char var = ‘2’
            begin
                char var = ‘3’
            end
            write(var) # Writes 2
        end
        if 1
            char var = ‘4’
            write(var) # Writes 4
        end
        write(var) # Writes 1
    end
end
```

همانند قسمت‌های قبل موارد با پس‌زمینه‌ی طوسی هر یک باید در یک خط جداگانه باشند و هیچ کد اجرایی در خط آن‌ها نباید باشد.

۳-۱۰. قوانین خطوط برنامه

نکات زیر نیز در مورد خطوط برنامه وجود دارد:

- تمامی دستورهای ارسال، تعریف متغیر، خروج از گیرنده (quit) و خروج از حلقه (break) باید در یک خط قرار بگیرند و کد معنی‌دار دیگری در آن خطوط نباشد.
- دستور تخصیص اگر به تنهایی مورد استفاده قرار گرفته باشد و اصطلاحاً یک Statement باشد باید در یک خط جداگانه باشد.
- تعریف اکتور و انتهای آن هر یک باید در یک خط باشند و کد معنی‌دار دیگری در آن خطوط نباشد. در مورد گیرنده، ساختار تکرار و شرطی نیز این قانون وجود دارد.

۱۱. توابع پیش‌فرض

در زبان Atalk دو تابع پیش‌فرض وجود دارد:

۱۱-۱. تابع write

این تابع به صورت ضمنی تعریف شده است و می‌تواند آرایه‌ای از کاراکترها (با هر طولی) و یا یک مقدار int یا char دریافت کند و آن را در کنسول چاپ کند. توجه نمایید که کد معنی‌دار دیگری داخل خط این تابع نمی‌تواند قرار بگیرد (یعنی استفاده از آن یک Statement است نه Expression).

۱۱-۲. تابع read

این تابع نیز به صورت ضمنی تعریف شده است و یک عدد ثابت را به عنوان ورودی دریافت می‌کند. سپس به تعداد آن عدد کاراکتر از کنسول خوانده و به صورت یک آرایه از کاراکتر برمی‌گرداند.

قطعه کد ۱۹

```
actor Program<10>
    receiver init()
        char[2] data
        write(“Do you want to continue?”)
        data = read(2)
        if data == “no”
            quit
        else
            write(“OK :)”)
            self << init()
        end
    end
end
```

توجه نمایید که توابع بالا استثنائاً به طور همگام[۲۷] اجرا می‌شوند و ریسمان مربوطه تا زمان اتمام آن‌ها بلاک می‌شود. البته شایان ذکر است که امکان پیاده‌سازی موارد بالا به شکل ناهمگام توسط I/O Interruptها نیز امکان‌پذیر است.

[27] Synchronous