دفترچه‌ی مرجع سریع نحو AZRA (AZRA Cipher Syntax)

نسخه‌ی فارسی مستند رسمی زبان برنامه‌نویسی AZRA

azralang.online | global.azralang.online

```
Imvoke(:"system/64/user….")
World-type(:Azra , [ip = 1234])
 name int, float
-- } فضای مربوط به کد اصلی
Extract : 123456
 int, float
Extract(:Hello World)
 *استفاده کنید "str" برای رشته‌های بیش از ۵۰ کاراکتر از
 str
Delog{int, float, str…}
 این دستور برای وارد کردن متغیرهایی که در کد نیاز داریم
استفاده می‌شود؛
 در غیر این صورت می‌توان این خط را حذف کرد.
Submit(:1.azr)
```

زبان برنامه‌نویسی AZRA — قواعد تعریف متغیر و نحو

نسخه ۱.۰ | نوع مستند: مشخصات رسمی زبان

۱. ساختار کلی تعریف متغیر

در AZRA، هر تعریف متغیر باید از یک الگوی دقیق و قابل تشخیص پیروی کند. این ساختار به سیستم اجازه می‌دهد تعریف متغیرها را به‌درستی شناسایی و تفسیر کند.

۱.۱ فرمت کلی

```
-<شماره متغیر><نام متغیر> = <مقدار>-
```

خط تیره‌های (-) ابتدایی و انتهایی به‌عنوان مرزهای ساختاری عمل می‌کنند. اگر هر یک از این دو خط تیره وجود نداشته باشد، سیستم قادر به تشخیص تعریف نخواهد بود و خطای نحوی (Syntax Error) رخ می‌دهد.

۲. قواعد پایه

قاعده‌ی ۱ — خط تیره‌ی ابتدایی الزامی است

هر تعریف متغیر باید با یک خط تیره (-) آغاز شود.

درست: `-0x = 2-`

نادرست: `0x = 2`

قاعده‌ی ۲ — شماره‌گذاری اجباری متغیرها

هر متغیر باید دارای یک شاخص عددی باشد که مستقیماً قبل از نام متغیر قرار می‌گیرد.

- شماره‌گذاری از ۰ آغاز می‌شود.
- هر متغیر جدید باید به‌صورت متوالی افزایش یابد.
- هیچ جهشی (شکاف عددی) مجاز نیست.

مثال (سومین متغیر): `-2msg = Hello-`

توجه: مقادیر اعشاری (Floating-point) بر قواعد شماره‌گذاری تأثیری ندارند.

قاعده‌ی ۳ — عدم وجود اعداد اضافی قبل از نام

تنها عددی که پیش از نام متغیر مجاز است، همان شاخص متغیر است.

نادرست: `-42A = Hello`

قاعده‌ی ۴ — اعداد درون نام متغیر

اعداد تنها زمانی می‌توانند در نام متغیر ظاهر شوند که پس از یک حرف بیایند.

درست: `-1Am1r = 10-` `-2x2 = 5-`

قاعده‌ی ۵ — خط تیره‌ی پایانی الزامی است

هر تعریف متغیر باید با یک خط تیره (-) پایان یابد.

درست: `-3b = Hello World3-`

نادرست: `-3b = Hello World3`

عدم وجود خط تیره‌ی پایانی، منجر به خطای «متغیر باز» (Open Variable Syntax Error) می‌شود.

۳. قواعد رشته‌ها

قاعده‌ی ۶ — رشته‌های کوتاه‌تر از ۵۰ کاراکتر

رشته‌هایی با کمتر از ۵۰ کاراکتر می‌توانند بدون علامت نقل‌قول نوشته شوند.

مثال‌ها: `-1y = Hello World-` `-2z = Hi-`

قاعده‌ی ۷ — رشته‌های ۵۰ کاراکتر یا بیشتر

رشته‌هایی با ۵۰ کاراکتر یا بیشتر باید داخل علامت نقل‌قول دوتایی (") قرار گیرند.

مثال:

```
-3longMsg = "This is a long text containing more than fifty characters…"-
```

قاعده‌ی ۸ — علامت نقل‌قول تکی مجاز نیست

علامت‌های نقل‌قول تکی (' ') در AZRA شناخته‌شده نیستند و باعث خطای نحوی می‌شوند.

نادرست: `-1x = 'Hello'-`

قاعده‌ی ۹ — اعداد درون رشته‌ها

اگر رشته‌ای با ترکیب عدد و حرف آغاز یا پایان یابد، همچنان به‌عنوان رشته در نظر گرفته می‌شود.

مثال‌های معتبر: `-1y = 1Hello World-` `-2z = Hello 1 World-`

`-3b = Hello World3-`

۴. قواعد تخصیص مقدار

قاعده‌ی ۱۰ — تشخیص خودکار نوع داده

سیستم نوع مقدار را با منطق زیر تعیین می‌کند:

- مقدار بدون نقل‌قول و بدون حرف → عددی (Numeric)
- مقدار داخل نقل‌قول دوتایی → رشته (String)
- مقداری که هم شامل حرف و هم عدد است → رشته (String)
- مقداری که شامل @ یا ref: باشد → نماد/مرجع (Symbol / Reference)

قاعده‌ی ۱۱ — قواعد مرجع/نماد

مراجعی مانند @2 به‌تنهایی یک مقدار مستقل نیستند. این‌ها باید هنگام استفاده در دستورات پشتیبانی‌شده (مانند Extract) داخل پرانتز قرار گیرند.

درست: `Extract(:@2)`

نادرست: `Extract : @2`

قاعده‌ی ۱۲ — رشته‌های چند کلمه‌ای

اگر رشته‌ای شامل چند کلمه باشد، حتی اگر کمتر از ۵۰ کاراکتر باشد، توصیه می‌شود برای وضوح بیشتر از نقل‌قول دوتایی استفاده شود.

پیشنهادی: `-0name = "Amir Hosseini"-`

۵. قواعد تکمیلی

قاعده‌ی ۱۳ — فاصله‌گذاری اطراف علامت تساوی

فاصله‌گذاری اطراف علامت = آزاد و سلیقه‌ای است. همه‌ی موارد زیر معتبرند:

`-0x = 2-` `-0x = 2 -` `-0x=2-`

قاعده‌ی ۱۴ — یک تعریف در هر خط

هر تعریف متغیر باید در یک خط جداگانه نوشته شود. تقسیم تعریف بین چند خط یا قرار دادن چند تعریف در یک خط، ممنوع است.

قاعده‌ی ۱۵ — عدم امکان جهش در شماره‌گذاری

شماره‌گذاری متغیرها باید متوالی باشد.

نادرست:

```
-0x = 5-
-2y = 10-
```

(شاخص ۱ وجود ندارد.)

قاعده‌ی ۱۶ — عدم مجاز بودن شاخص منفی

شاخص متغیرها باید اعداد صحیح غیرمنفی باشند.

نادرست: `-1x = 5-`

قاعده‌ی ۱۷ — نام متغیر نمی‌تواند با عدد شروع شود

پس از شاخص متغیر، نام متغیر باید با یک حرف آغاز شود.

نادرست:

```
-0 1name = 2-
-3 12x = ok
```

قاعده‌ی ۱۸ — کاراکترهای مجاز در نام متغیرها

نام متغیرها می‌توانند شامل موارد زیر باشند:

- حروف بزرگ (A–Z)
- حروف کوچک (a–z)
- عدد (فقط پس از یک حرف)
- زیرخط (Underscore) _

هیچ کاراکتر دیگری مجاز نیست.

قاعده‌ی ۱۹ — مقادیر رشته‌ای که با عدد آغاز می‌شوند

اگر مقدار رشته‌ای با یک عدد آغاز شود، باید یکی از این دو حالت را داشته باشد:

- داخل نقل‌قول دوتایی قرار گیرد
- یا بلافاصله پس از عدد، یک حرف بیاید

درست: `-0a = "1Hello"-` `-1code = 1xTest-`

نادرست: `-0a = 1-`

قاعده‌ی ۲۰ — تخصیص مستقیم نمادها ممنوع است

نمادهایی مانند @2 نمی‌توانند مستقیماً به یک متغیر تخصیص داده شوند.

نادرست: `-0ref = @2-`

نمادها باید از طریق ساختارهای پشتیبانی‌شده در دسترس قرار گیرند: `Extract(:@n)`

مدل مفهومی و معماری اصلی

فراداده‌ی فایل (File Metadata)

سیستم مقصد (Target System)

سیستم مقصد به سیستمی گفته می‌شود که کد روی آن اجرا خواهد شد. این سیستم می‌تواند دو شکل داشته باشد:

1. همان سیستمی که کد در آن نوشته می‌شود.
2. دستگاهی اختصاصی که مخصوص اجرای زبان AZRA ساخته شده است.

نام فایل (Filename)

فایل حاوی کد باید تنها با استفاده از اعداد نام‌گذاری شود.

مثال‌ها: `1.azr` `561.azr`

هیچ کاراکتر حرفی در نام فایل مجاز نیست.

وابستگی‌ها (Dependencies)

یک فایل AZRA فقط می‌تواند به فایل‌های دیگری که به همان زبان (AZR) نوشته شده‌اند متصل شود. اگر فایل نیاز به اتصال به کد خارجی یا یک فایل غیر AZR داشته باشد، آن فایل ابتدا باید به فرمت AZR تبدیل (کامپایل) شود. بنابراین، فایل‌های AZRA فقط با فایل‌های هم‌نوع خود ارتباط برقرار می‌کنند.

فراداده‌ی پیش از اجرا (Pre-Execution Metadata)

پیش از آغاز بدنه‌ی اصلی کد، برخی فراداده‌ها باید تعریف شوند. این موارد شامل:

- سیستم فراخوانی‌شده
- نوع فایل (پیش‌فرض: azr)
- شناسه‌ی سیستم برای مجوز اجرا
- نماد «–» برای مشخص کردن آغاز بخش کد اصلی

Imvoke

در ابتدای هر فایل، دستور Imvoke باید ظاهر شود. این دستور مسئول موارد زیر است:

- فراخوانی سیستم اجرایی
- تعیین نسخه‌ی سیستم
- تعیین سطح دسترسی
- مشخص‌کردن محیطی که مجاز به اجرای فایل است

مثال: `Imvoke(:"system/64/user.001.alpha")`

این خط به سیستم اعلام می‌کند:

- کدام محیط باید فایل را اجرا کند
- کدام نسخه لازم است
- چه سطح دسترسی مجاز است

اگر سیستمی غیر از سیستم مشخص‌شده بخواهد فایل را اجرا کند، اجرا رد خواهد شد. بدون دستور Imvoke، فایل مجاز به اجرا نیست و رد می‌شود.

World-type

دستور World-type محیط اجرا و پیکربندی جهان (world configuration) را تعریف می‌کند.

مثال: `World-type(:Azra | [ip = 1234])`

این خط مشخص می‌کند که کد در جهان Azra با پارامترهای محیطی تعریف‌شده (مانند آدرس IP یا شناسه‌ی شبکه) اجرا می‌شود.

> **نکته‌ی مهم نحوی:** در AZRA، نام متغیرها هرگز نباید شامل فاصله باشند.
>
> مثال نادرست: `-0amir reza = 2-` — این مورد به دلیل وجود فاصله بین amir و reza، خطای نحوی ایجاد می‌کند.

روش صحیح برای نام متغیرهای چند کلمه‌ای

برای جدا کردن کلمات درون نام متغیر، باید از کاراکتر زیرخط (Underscore) _ استفاده شود.

نسخه‌ی صحیح: `-0amir_reza = 2-`

این نسخه نیز معتبر است: `-0amir-reza = 2-`

توجه: در World-type، قاعده‌ی زیرخط اعمال نمی‌شود؛ زیرا World-type یک دستور اصلی سیستم است و تابع دستور زبان (Grammar) استاندارد متغیرهای AZRA نیست.

فراداده‌ی پس از اجرا (Post-Execution Metadata)

در انتهای فایل، دستورات خاصی باید اضافه شوند تا به سیستم اجرایی اطلاع دهند:

- چه داده‌ای باید پردازش شود
- چه داده‌ای باید نادیده گرفته شود

این بخش معمولاً شامل دو دستور نهایی است: `Delog{ … }` و `Submit`

Delog

فرمت کلی: `Delog{int, str, …}`

این دستور مشخص می‌کند کدام نوع‌های داده از متغیرهای فایل باید هنگام کامپایل شناسایی شوند.

کارکرد:

- مشخص می‌کند برنامه کدام نوع متغیرها (int، str، bool، ref و غیره) را باید پردازش کند
- به کامپایلر دستور می‌دهد متغیرهای انواع دیگر را نادیده بگیرد
- اطمینان می‌دهد که فقط انواع فهرست‌شده در {} در پردازش نهایی لحاظ می‌شوند

اگر Delog{...} وجود نداشته باشد: تمام متغیرهای فایل معتبر و ضروری در نظر گرفته می‌شوند و کامپایلر باید همه را پردازش کند.

مثال: `Delog{int, str}` – یعنی فقط متغیرهای عددی و رشته‌ای مرتبط هستند و انواع دیگر (مثل bool یا ref) نادیده گرفته می‌شوند.

Submit

فرمت کلی: `Submit(:<filename>)`

مثال: `Submit(:1.azr)`

این دستور باید در انتهای فایل ظاهر شود. کارکرد آن:

- پایان رسمی کد را مشخص می‌کند
- اجرای موفق را تأیید می‌کند
- نام فایل را برای گزارش‌دهی و ردیابی وابستگی‌ها ثبت می‌کند
- هیچ دستور اجرایی نمی‌تواند پس از Submit ظاهر شود

ساختار پایانی استاندارد

در یک فایل استاندارد AZRA، پایان فایل معمولاً به این شکل است:

```
Delog{int, str, ref}
Submit(:123.azr)
```

روند اجرا:

1. Delog مشخص می‌کند کدام نوع داده‌ها پردازش و کدام نادیده گرفته شوند.
2. Submit اجرای فایل را نهایی و رسماً می‌بندد.

Delog و Submit مکمل یکدیگرند و بلوک پایانی رسمی یک فایل AZR را تشکیل می‌دهند. اگر هریک از آن‌ها وجود نداشته باشد، سیستم فایل را ناقص یا نامعتبر در نظر می‌گیرد.

AZRA — قانون جداکننده‌ی بلوک خط‌تیره (Dash Block Separator / DBS)

مستند رسمی قوانین زبان

مقدمه

قانون DBS یکی از اصول ساختاری اصلی زبان برنامه‌نویسی AZRA است. هدف اصلی آن، اجرای سازمان‌دهی، افزایش خوانایی، و امکان بررسی ساختاری بلوک‌های کد در زمان کامپایل است. استفاده‌ی صحیح از DBS به کامپایلر امکان می‌دهد به‌طور دقیق مشخص کند هر بلوک کد از کجا شروع و کجا پایان می‌یابد.

قانون اصلی

پس از هر بلوک چندخطی، باید یک خط جداکننده وجود داشته باشد که تنها از کاراکترهای خط‌تیره (-) تشکیل شده است. تعداد خط‌تیره‌ها در آن خط باید دقیقاً برابر با تعداد خطوط موجود در بلوک پیشین باشد. این خط به‌عنوان «امضای بلوک» (Block Signature) عمل می‌کند و صحت ساختاری و کامل بودن آن بخش از کد را تأیید می‌کند.

تعریف بلوک

یک بلوک به‌عنوان یک دنباله از خطوط پیاپی تعریف می‌شود که در کنار هم یک واحد منطقی واحد را در برنامه تشکیل می‌دهند.

نمونه‌های رایج بلوک در AZRA عبارت‌اند از:

- چندین تعریف متغیر پیاپی
- چندین دستور Extract که در کنار هم گروه‌بندی شده‌اند
- مجموعه‌ای از عملیات مرتبط
- هر بخشی از کد که برنامه‌نویس آن را به‌عنوان یک واحد منسجم در نظر می‌گیرد

پس از هر بلوک، درج خط DBS الزامی است. حذف خط DBS منجر به خطای کامپایل می‌شود.

نمونه‌ها

نمونه‌ی ۱: بلوک دو خطی

```
-0x = Hello World-
Extract(:Hello World)
```

توضیح: چون بلوک شامل ۲ خط است، خط جداکننده شامل ۲ خط‌تیره است (--).

نمونه‌ی ۲: بلوک سه خطی

```
-1y = 1Hello World-
-2z = Hello 1 World-
-3b = Hello World3-
```

توضیح: بلوک شامل ۳ خط است، بنابراین جداکننده از ۳ خط‌تیره تشکیل شده است (---).

اهمیت DBS

DBS صرفاً یک جداکننده‌ی بصری نیست، بلکه نقش‌های ساختاری مهمی در زبان AZRA ایفا می‌کند:

۱. یکپارچگی ساختار فایل

اگر حتی یک خط درون یک بلوک جابه‌جا، اضافه یا حذف شود، کامپایلر بلافاصله با مقایسه‌ی تعداد خط‌تیره‌ها این ناسازگاری را تشخیص می‌دهد.

۲. اعتبارسنجی ترتیب بلوک‌ها

از آنجا که طول بلوک و طول جداکننده به‌هم مرتبط هستند، جعل، دستکاری یا تعریف ناقص یک بلوک بدون شناسایی، عملاً غیرممکن می‌شود.

انواع خطای DBS

کامپایلر موظف است خطاهای مرتبط با DBS زیر را شناسایی و گزارش کند:

1. **DBS-Mismatch** — تعداد خط‌تیره‌ها در جداکننده با تعداد خطوط بلوک پیشین مطابقت ندارد.

2. **DBS-Missing** — بلوک پایان یافته اما هیچ خط DBS قرار داده نشده است.
3. **DBS-Invalid** — خط جداکننده شامل کاراکترهایی غیر از خط‌تیره (-) است.

جمع‌بندی

قانون DBS موارد زیر را تضمین می‌کند:

- پایداری ساختاری کد AZRA
- تشخیص خودکار خطاهای ویرایش یا کپی‌برداری
- تأیید کامل بودن بلوک در زمان کامپایل

با اجرای الزامی DBS در پایان هر بلوک چندخطی، AZRA اعتبارسنجی ساختاری قطعی و یکپارچگی دقیق کامپایل را حفظ می‌کند.

AZRA – مشخصات: کلاس، متد، تابع

تعریف رسمی ساختاری و رفتاری

۱. تعریف کلاس در AZRA

یک کلاس (Class) در AZRA برای تعریف یک نوع داده‌ی سفارشی استفاده می‌شود. یک کلاس شامل موارد زیر است:

- نام کلاس
- فهرستی از ویژگی‌ها (Properties)
- بدنه‌ی متدها (اختیاری)

ساختار کلی

```
Class(:ClassName | [prop1:type , prop2:type , …])
```

- پرانتز اول نام کلاس را در بر دارد
- بلوک کروشه شامل فهرست ویژگی‌هاست

پس از تعریف یک کلاس، باید یک خط DBS با دقیقاً ۱ خط‌تیره قرار گیرد، زیرا تعریف کلاس یک خط ساختاری واحد محسوب می‌شود.

مثال: `Class(:Person | [name:str , age:int])`

این ساختار یک کلاس Person با دو ویژگی name و age می‌سازد.

۲. تعریف متد در AZRA

یک متد (Method) تابعی است که به یک کلاس وابسته است و رفتار یک شیء (Object) را توصیف می‌کند.

ساختار کلی

```
Method(:methodName) ->returnType
-[تعریف متغیرها و عملیات]-
turn(:value)
```

قواعد

- یک متد همیشه باید با کلیدواژه‌ی Method آغاز شود
- returnType نوع خروجی متد را تعیین می‌کند
- متغیرهای داخلی باید از قواعد تعریف متغیر AZRA پیروی کنند
- خروجی باید با استفاده از turn تولید شود
- متد باید با یک خط DBS پایان یابد، که تعداد خط‌تیره‌ها برابر با تعداد خطوط داخلی است

مثال:

```
Method(:greet) ->str
-0msg = "Hello" + name-
turn(:msg)
```

(۲ خط داخلی → --)

۳. تعریف تابع در AZRA

یک تابع (Func) قطعه کد مستقلی است که خارج از کلاس‌ها قرار دارد و اغلب برای منطق سطح بالاتر یا جریان اصلی برنامه استفاده می‌شود.

ساختار کلی

```
Func(:functionName)
-[تعریف متغیرها و عملیات]-
turn(اختیاری)
```

قواعد

- یک تابع با Func آغاز می‌شود
- می‌تواند شامل متغیرها، عملیات، فراخوانی متدها و Extract باشد
- خروجی با turn تعریف می‌شود؛ می‌تواند خالی باشد
- خط DBS پایانی باید تعداد خط‌تیره‌ای برابر با تعداد خطوط داخلی داشته باشد

مثال:

```
Func(:main)
-0p = Person("Amir" , 20)-
-1result = p:greet()-
Extract(:@1)
turn()
```

(۴ خط داخلی → ----)

۴. فراخوانی متد

متدها با استفاده از عملگر : فراخوانی می‌شوند.

فرمت کلی

```
object:method()
```

مثال: `p:greet()` — این دستور متد greet را روی شیء p اجرا می‌کند.

۵. نمونه‌سازی شیء (Object Instantiation)

اشیاء با استفاده از نام کلاس به همراه آرگومان‌ها ساخته می‌شوند.

ساختار کلی

```
variable = ClassName(arg1 , arg2 , …)
```

مثال: `-0p = Person("Amir" , 20)-`

۶. دستور turn

دستور turn خروجی یک متد یا تابع را تعریف می‌کند.

فرم‌ها: `turn(:value)` `turn() // بدون خروجی`

۷. نقش DBS در کلاس‌ها، متدها و توابع

تمام تعریف‌های کلاس، متد و تابع باید با یک خط DBS پایان یابند. تعداد خط‌تیره‌ها باید دقیقاً با تعداد خطوط منطقی داخلی بلوک برابر باشد. این امر تضمین می‌کند:

- اعتبارسنجی ساختاری
- تشخیص خطوط گمشده، اضافی یا جابه‌جا شده
- یکپارچگی بلوک در زمان کامپایل

مشخصات ساختار شرطی (is / reply / shoot / wall)

مروری کلی

در AZRA، تصمیم‌گیری منطقی از طریق یک زنجیره‌ی شرطی چهار بخشی متشکل از کلیدواژه‌های رزروشده‌ی زیر انجام می‌شود:

1. is
2. reply
3. shoot
4. wall()

این ترتیب هرگز نباید تغییر کند. هرگونه تغییر در ترتیب، منجر به خطای نحوی می‌شود.

۱. is — شرط اصلی

ساختار شرطی همیشه با is آغاز می‌شود.

- شرط منطقی اصلی را تعریف می‌کند
- اگر شرط برابر با true ارزیابی شود، بلوک آن اجرا می‌شود
- پس از اجرا، کل زنجیره پایان می‌یابد
- اگر false باشد، اجرا به مرحله‌ی بعد (reply) منتقل می‌شود

۲. reply — شاخه(های) شرطی اضافی

- صفر یا چند بخش reply می‌تواند پس از is بیاید
- هر reply شرط مستقل خود را دارد
- یک reply فقط زمانی ارزیابی می‌شود که همه‌ی شرط‌های پیشین ناموفق بوده باشند
- اگر true باشد، بلوک آن اجرا شده و زنجیره پایان می‌یابد
- اگر false باشد، reply بعدی بررسی می‌شود

۳. shoot — مسیر پیش‌فرض

اگر هیچ‌یک از شرط‌های پیشین (is یا هر reply) true نشوند:

- اجرا وارد بلوک shoot می‌شود
- shoot به‌عنوان مسیر بازگشتی نهایی (Default Path) عمل می‌کند
- همیشه پس از تمام replyها می‌آید

۴. wall() — پایان‌دهنده‌ی زنجیره‌ی شرطی

زنجیره‌ی شرطی باید با wall() پایان یابد. این دستور:

- پایان رسمی بلوک شرطی را نشان می‌دهد
- به کامپایلر امکان می‌دهد ساختار را به‌درستی ببندد

اگر wall() وجود نداشته باشد، خطای نحوی رخ می‌دهد؛ زیرا کامپایلر نمی‌تواند مشخص کند ساختار تصمیم‌گیری کجا پایان می‌یابد.

خلاصه‌ی روند اجرا

1. ارزیابی is — اگر true باشد → اجرا → پایان
2. در غیر این صورت، ارزیابی هر reply به ترتیب — اولین reply درست → اجرا → پایان
3. در غیر این صورت، اجرای shoot
4. اجرای wall() برای بستن رسمی ساختار
5. در نهایت، یک خط DBS باید دنبال شود

الزام DBS

پس از پایان زنجیره‌ی شرطی (بلافاصله پس از wall()):

- یک خط DBS باید ظاهر شود
- تعداد خط‌تیره‌ها باید با مجموع خطوط منطقی داخل بلوک شرطی برابر باشد
- عدم تطابق منجر به خطای DBS می‌شود

تعریف رسمی ساختار شرطی

مشخصات ساختار شرطی — زبان AZRA

۱. اجزای ساختاری زنجیره‌ی شرطی

۱.۱ عناصر اصلی

ساختار شرطی در AZRA از چهار جزء تشکیل شده است: «is()»، «reply»، «shoot»، «wall». این چهار عنصر در کنار هم زنجیره‌ی کامل تصمیم‌گیری منطقی را می‌سازند.

۱.۲ بخش «is»

۱.۲.۱ کلیدواژه‌ی «is» نخستین عنصر زنجیره‌ی شرطی است.

۱.۲.۲ شرط اصلی ساختار درون این بخش تعریف می‌شود.

۱.۲.۳ اگر شرط درون «is» به true ارزیابی شود، بلوک متناظر آن اجرا می‌شود.

۱.۲.۴ پس از اجرای بلوک «is»، زنجیره‌ی شرطی بلافاصله پایان می‌یابد.

۱.۳ بخش «reply»

۱.۳.۱ کلیدواژه‌ی «reply» برای تعریف شرط‌های اضافی پس از «is» استفاده می‌شود.

۱.۳.۲ هر «reply» تنها زمانی ارزیابی می‌شود که همه‌ی شرط‌های پیشین به false ارزیابی شده باشند.

۱.۳.۳ چند بخش «reply» در یک زنجیره‌ی شرطی واحد مجاز است.

۱.۳.۴ بخش‌های «reply» دقیقاً به همان ترتیبی که نوشته شده‌اند ارزیابی می‌شوند.

۱.۳.۵ اولین «reply» که شرطش true ارزیابی شود، بلوک آن اجرا شده و زنجیره در همان نقطه پایان می‌یابد.

۱.۴ بخش «shoot»

۱.۴.۱ کلیدواژه‌ی «shoot» مسیر بازگشتی (پیش‌فرض) زنجیره‌ی شرطی را تعریف می‌کند.

۱.۴.۲ «shoot» تنها زمانی اجرا می‌شود که هیچ‌یک از شرط‌های «is» یا «reply» درست نباشند.

۱.۴.۳ تعریف «shoot» اختیاری است.

۱.۴.۴ اگر «shoot» تعریف نشده باشد و هیچ شرطی true نشود، زنجیره بدون اجرای هیچ بلوکی پایان می‌یابد.

۱.۴.۵ «shoot» دو فرم نحوی دارد:

- فرم کامل: `shoot() ->`
- فرم کوتاه: `shoot ->`

۱.۵ بخش «wall()»

۱.۵.۱ کلیدواژه‌ی «wall()» پایان رسمی زنجیره‌ی شرطی را مشخص می‌کند.

۱.۵.۲ حضور «wall()» در پایان ساختار شرطی الزامی است.

۱.۵.۳ حذف «wall()» منجر به خطای نحوی می‌شود.

۲. ترتیب عناصر در ساختار شرطی

۲.۱ ترتیب معتبر و صحیح

```
"is" → "reply" (صفر یا چند بار) → "shoot" (اختیاری) → "wall()"
```

۲.۲ قواعد ترتیب

۲.۲.۱ ترتیب تعریف‌شده باید دقیقاً رعایت شود.

۲.۲.۲ هرگونه تغییر ترتیب، تکرار نامعتبر، یا جای‌گذاری نادرست عناصر، منجر به خطای نحوی می‌شود.

۲.۲.۳ فقط اولین شاخه‌ای که شرطش true ارزیابی شود اجرا می‌گردد. تمام شاخه‌های بعدی نادیده گرفته می‌شوند.