Logic PL
Documentation

1 .مقدمه

زبان LogicPL یک زبان برنامه‌نویسی منطقی است. برنامه‌هایی که به این زبان نوشته می‌شوند شامل تعدادی fact و تعدادی قواعد اگر آنگاه (implication) هستند که با توجه به این موارد می‌تواند به کوئری‌ها پاسخ دهد. در صورتی که در برنامه function تعریف شده باشد، تعریف آن قبل از fact ها و قواعد implication نوشته می‌شود.

2. ساختار کلی

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

- در صورت وجود function، تعریف آن در ابتدای فایل نوشته می شود.
- در ادامه کلیدواژه ی main نوشته می شود و سایر قسمت های برنامه درون brace نوشته می‌شوند.

در این زبان برنامه‌نویسی گزاره‌ها می‌توانند از نوع assignment، تعریف متغیر، تعریف predicate، از نوع implication، از نوع return (در بدنه‌ی function) و یا از نوع چاپ (مثلا چاپ کوئری یا چاپ مقدار متغیر) باشند. در پایان هر گزاره، یک semicolon وجود دارد.

در صورتی که در یک scope بیش از یک گزاره وجود داشته باشد، گزاره‌ها داخل {} نوشته می‌شوند.

2-1. قواعد کلی نحو

این زبان به بزرگ و کوچک بودن حروف حساس است. در این زبان برنامه‌نویسی وجود کاراکترهایی مثل space و tab و enter تأثیری در اجرای برنامه ندارد.

2-2. کامنت ها

در این زبان برنامه‌نویسی comment ها فقط به صورت یک خطی هستند و با # شروع می‌شوند. مثال:

```
# this is a comment
```

2-3. قواعد نامگذاری متغیرها و function ها

- نام متغیرها و تابع‌ها فقط می‌تواند از حروف بزرگ و کوچک و ارقام 0 تا 9 و underscore تشکیل شده باشد و باید با حرف کوچک شروع شوند.
- در هر برنامه که به این زبان نوشته می‌شود، نام تابع‌ها unique هستند.
- معادل کلید واژه ها نباشد. تمام کلیدواژه های این زبان در جدول زیر آمده است:

function	main	int	float	boolean	true
false	print	for	return		

2-4. قواعد نامگذاری predicate ها

هر predicate یک نام دارد که فقط از حروف و اعداد تشکیل شده است و باید با حرف بزرگ شروع شود. همچنین نام predicate ها نباید معادل با نام کلیدواژه ها (که در جدول بالا آمده است) باشد.

3. انواع داده:

در این زبان برنامه‌نویسی سه نوع داده‌ی اولیه‌ی int، float و boolean وجود دارد. در صورتی که یک متغیر مقداردهی اولیه نشده باشد، برای آن مقادیر پیش‌فرض در نظر گرفته می‌شود. مقادیر پیش‌فرض برای انواع int و

float صفر است. برای متغیرهای از نوع boolean، مقدار پیش‌فرض true است. در این زبان برنامه‌نویسی برخی از متغیرها نیز به عنوان آرگومان predicate ها به کار می‌روند. این متغیرها لازم نیست از قبل تعریف شده باشند.

4. تعریف متغیرها:

ممکن است متغیرها در بدنه‌ی برنامه‌ی اصلی (main) یا در function ها تعریف شده باشند. scope متغیرهای تعریف شده در function ها در همان function (از محلی که متغیر تعریف شده است به بعد) می‌باشد. مثال‌هایی از تعریف متغیرها در این زبان برنامه‌نویسی به صورت زیر می‌باشد:

```
int a = 8;
float b; # the default value for the float variable is zero
boolean p; # the default value for the boolean variables is true
```

5. آرایه:

در این زبان امکان تعریف آرایه وجود دارد. همه‌ی عناصر یک آرایه باید از یک نوع باشند. طول آرایه حداقل یک است و باید در زمان تعریف آرایه مشخص شود (به صورت یک عدد صحیح). index آرایه از 1 شروع می‌شود. در صورتی که آرایه مقداردهی اولیه نشده باشد، مقدار پیش‌فرض متناسب با نوع عناصر آن در نظر گرفته می‌شود. برای دسترسی به عناصر آرایه، index آن در [ ] نوشته می‌شود. Index آرایه ممکن است یک عدد صحیح باشد یا یک عبارت ریاضی از نوع int باشد. مثال:

```
int[10] a1; # the initial values are zero
a1[1] = a1[1] + 10;
int[4] a2 = [2, 4, 6, 8];
int num = a1[1] +a2[4]; # num is equal to 10 + 8 = 18
```

6. عملگرها:

عملگرهای تعریف شده در این زبان و چگونگی شرکت‌پذیری آن‌ها در جدول زیر خلاصه شده‌اند. در این جدول، عملگرهایی که اولویت بیشتری دارند، بالاتر نوشته شده‌اند.

Operator	Type	Associativity
( )	parentheses	L-to-R
[ ]	Access to array elements	
+, -, !	Unary operator	R-to-L
*, /, %	Binary arithmetic operator	L-to-R
+, -	Binary arithmetic operator	L-to_R
<, <=, >, >=	Relational operator	L-to-R
==, !=	Relational operator	L-to-R
&&	Logical AND operator	L-to-R
||	Logical OR operator	L-to-R
=	Assignment operator	R-to-L

در این جدول عملگر !، عملگر NOT منطقی می‌باشد. عملگرهای تک‌عملوندی !، + و – از نوع پیشوندی (prefix) هستند.

7. تعریف predicate ها:

نوعی از گزاره‌های این زبان، تعریف predicate ها هستند. هر predicate یک نام دارد که فقط از حروف و اعداد تشکیل شده است و باید با حرف بزرگ شروع شود. در این زبان برنامه‌نویسی هر predicate دقیقا یک آرگومان دارد که داخل پرانتز و در جلوی نام آن نوشته می‌شود. مثلا $EvenNumber(a)$ یک predicate است که نام آن EvenNumber و آرگومان آن a است و به این معنی است که در مورد متغیر a، ویژگی EvenNumber برقرار است. متغیرهایی که به عنوان آرگومان predicate ها استفاده می‌شوند ممکن است از قبل تعریف شده باشند (اما

ضروری نیست که از قبل تعریف شده باشند). یک متغیر ممکن است به صورت همزمان آرگومان چندین predicate باشد. مثلا در مورد متغیر a ممکن است $EvenNumber(a)$ و $DivisibleByFive(a)$ به صورت همزمان برقرار باشند. مثال:

```
PrimeNumber(p);
PrimeNumber(q);
CompositeNumber(num1);
```

8. گزاره‌های implication:

این نوع از گزاره‌ها به صورت «اگر آنگاه» تعریف می‌شوند و به صورت (p) => (q) نوشته می‌شوند به طوری که p یک عبارت boolean می‌باشد و q شامل یک یا تعدادی گزاره است. مثال:

```
(a % 2 == 0 && a % 5 == 0) => (DivisibleBy10(a);)
```

در این مثال اگر مقدار متغیر a بر اعداد 2 و 5 بخش پذیر باشد، ویژگی $DivisibleBy10$ در مورد a برقرار است (یعنی مقدار متغیر a بر 10 بخشپذیر است) که این ویژگی با استفاده از تعریف یک predicate به صورت $DivisibleBy10(a)$ مشخص شده است.

9. ساختار loop:

در این زبان برنامه‌نویسی، برای ساختار loop از کلیدواژه‌ی for استفاده می‌شود و در آن iteration روی عناصر یک آرایه انجام می‌شود. مثال:

```
int num = 100;
int[4] divisor_array = [ 2, 3, 5, 7];
int[4] remainder_array;
int sum = 0;
int i = 1;
for (a: divisor_array){
    remainder_array[i] = num % a;
    i = i + 1;
}
```

10. تعریف function:

برای تعریف تابع از کلیدواژه‌ی function استفاده می‌شود. سپس نام تابع نوشته می‌شود. آرگومان‌های تابع و نوع آن‌ها داخل پرانتز نوشته می‌شوند. سپس یک ":" و بعد نوع بازگشتی تابع نوشته می‌شود.

- آرگومان‌های تابع می‌توانند از نوع int، float یا boolean باشند.
- در این زبان برنامه‌نویسی، نوع بازگشتی توابع می‌تواند به صورت int، float یا boolean باشد. بدنه‌ی تابع داخل brace نوشته می‌شود.
- در هر function باید حداقل یک گزاره از نوع return وجود داشته باشد.مثال تعریف function:

```
function isEvenNumber(int a): boolean {
    (a % 2 == 0) => (return true;)
    (a % 2 != 0) => (return false;)
}
```

11. تابع print:

این تابع مقدار یک متغیر یا پاسخ یک کوئری که آرگومان آن هستند را در خروجی چاپ می‌کند. تابع print حتما باید ورودی داشته باشد.

12. کوئری‌ها:

در این زبان برنامه‌نویسی برای تعریف کوئری‌ها از [ ] استفاده می‌شود. در این زبان دو نوع کوئری تعریف می‌شود که با توجه به predicate ها و قواعد implication تعریف شده در برنامه به آن‌ها پاسخ داده می‌شود:

1. کوئری نوع 1: پاسخ کوئری‌هایی که به صورت $[?\,EvenNumber(a)]$ هستند به صورت true یا false است. یعنی باید نشان دهند که آیا در مورد a، ویژگی EvenNumber برقرار است یا نه. خروجی این نوع از کوئری‌ها از نوع boolean است. بنابراین از آن‌ها می‌توان در گزاره‌های منطقی (مثلا در گزاره‌های implication) نیز استفاده کرد.
2. کوئری نوع 2: پاسخ کوئری‌هایی که به صورت $[EvenNumber(?)]$ هستند، شامل لیستی از نام همه‌ی متغیرهایی است که در مورد آن‌ها EvenNumber برقرار است.

برای چاپ نتایج هر یک از این کوئری‌ها از تابع print استفاده می‌شود. مثالی از کاربرد کوئری‌ها:

```
# predicate definitions
PositiveInteger(a);
Integer(b);
GreaterThanZero(b);

# Example of using type 1 queries in implication statements:
([?Integer(b)] && [?GreaterThanZero(b)]) => (PositiveInteger(b);)

# Example of type 2 queries:
print([PositiveInteger(?)]); # prints "{a,b}"
```

13. مثال

در ادامه یک مثال از برنامه به این زبان برنامه‌نویسی آمده است:

```
# definition of functions
function isDivisible(int a, int b): boolean {
    (b == 0) => (return false;)
    (a % b == 0) => (return true;)
    (a % b != 0) => (return false;)
}

main{
    int num1 = 50;
    EvenNumber(num1);

    ([?EvenNumber(num1)] && isDivisible(num1, 5)) => (DivisibleBy10(num1);)

    int[2] int_array = [4, 5];
    int num2 = 200;

    boolean divisible = true;
    for (a: int_array){
     (!isDivisible(num2, a)) => (divisible = false; return;)
    }
    (divisible == true) => (DivisibleBy20(num2);)

    ([?DivisibleBy20(num2)]) => (EvenNumber(num2);DivisibleBy10(num2);)

    print([?EvenNumber(num2)]); # prints "true"
    print([DivisibleBy10(?)]); # prints "{num1, num2}"
}
```