Crazy Language Specification
Version 1.3.1

In this course, you are required to implement some assignments on a programming language, called Crazy. The purposes of these assignments are for you:

- to practice specifying formally lexicon and grammar of a language,
- to embed the semantic actions into the grammar,
- to study how to use ANTLR, a compiler generator,
- to improve your programming skill in Python.

To complete these assignments, you firstly need to read the Crazy language specification, represented in the next section, and rewrite it formally using regular expression and Extended Backus–Naur form (EBNF). You also need to use ANTLR (www.antlr.org) to implement the translator.

1. Lexicon:

The character set ∑ of Crazy is the characters in ASCII. A letter is a character from **a** to **z** or from **A** to **Z**. A digit is a character from 0 to 9.

A lexicon in Crazy is an identifier, a constant, a keyword, a separator or an operator. An identifier starts with a lowercase letter optionally followed by uppercase or lowercase letters or digits. For example, **a1bc, bSaS, fOO,…** All kinds of name in Crazy are written as an identifier.

There are many kinds of constants: integer, real, boolean , string and array. An integer is a sequence of digits that always starts with a nonzero digit unless it is 0. For example **123, 21, 1, 0,**…

A real constant contains an integer portion, a fractional portion, and an exponent. The integer portion is similar to an integer constant. The fractional portion starts with a dot (.), followed by a non-empty sequence of digits that never ends with 0 unless it is 0. The exponent part starts with the letter 'E' or 'e', followed optionally by the negative sign (i.e. - or none) and a non-empty sequence of digit. The integer portion, the fractional portion and the exponent part are all optional but at least one of them must be present. In addition, the integer portion and exponent part never appears alone. Particularly, if there is no the integer portion, the fractional portion must be present. In the other case, at least one of the others (fractional portion and exponent part) are required. For example, `1.03e-2, 1.03e2, 1.03, ~~1.e-2~~, ~~1.e2~~, 1e-2, 1e2, ~~1.~~, .1e-2, .1e2, .12, 1.0e-2, 1.0e2`…

A boolean constant is keyword **true** or **false**.

A string constant starts with a single-quote ' followed by many characters and ends with another single-quote. An empty string constant consists of only two single quotes. Any character in ∑ except a newline, single quote, a backslash or a tab character can be appeared in a string constant. If a single-quote belongs to a string constants, it must be duplicated. The new line, backslash and tab character is replaced by escape sequences: \n, \\ and \t, respectively. For example, **'abc de', 'asnd " " ',", '\n \\ \t',…**

An array constant is a comma-separated list of elements enclosed in left and right square brackets. The list may NOT be empty. The elements may be integer, real, boolan, string or array constant. Note that the elements of an array must be in the same type. For example **[1,3,2], [3.2,.2E-2], [[1,2],[3,4]], {}, …**

The keywords in Crazy are **and, continue, of, then, array, div, function, or, begin, do, if, loop, procedure, boolean, integer, program, var, break, else, mod, real, while, const, end, not, string.** Ơ-=

The separators are comma (,), colon (:), semicolon (;) and period (.).

The operators are ( ) [ ] + - * / > < <= >= <> = :=

A whitespace is a blank, a tab or a newline character. There are two kinds of comments: block and line. A line comment starts with // and ends with a newline character. A block comment starts with /* and ends with */. A block comments may span on many lines and may not nest. Crazy ignores whitespaces and comments.

2. Program structure:

A program in Crazy always starts with a program declaration, followed by a block statement and terminated by a period. An optional declaration part may appear between the program declaration and the block statement if the programmer would like to declare some variables or their own procedures.

2.1. Program declaration

The program declaration starts with the **program** keyword, followed by the name of that program and end up by a semi-colon (;).

For example:

```
program myMPProgram;
```

In Crazy, the program name is provided just for programmers convenience. It does not have any effect in that program.

2.2. Declaration part

The declaration part contains a variable (and constant) declaration part and a procedure (and function) declaration part. Note that the procedure declaration part **must follow** the variable declaration part.

2.2.1. Variable declaration part

The variable declaration part consists of several (or no) constant and variable declarations.

Each constant declaration must begin with the **const** keyword, followed by an identifier, a '=', a literal (could be integer, real, string or boolean one) and a semicolon.

Each variable declaration has the form:

var identifier-list **:** *type* **;**

The identifier-list is a non-empty comma-separated list of identifiers. *type* could be a primitive type or an array type which will be discussed later in the Type section.

For example:

```
const myIntConst = 5;
var my1stVar: integer;
var myArrayVar: array[5] of real;
var my2ndVar, my3rdVar: boolean;
var my2ndArray, my3rdArray: array[6][8] of real;
const myRealConst = 5.3e4;
```

2.2.2. Procedure declaration part

The procedure declaration part contains several (or no) function and procedure declarations.

The function declaration begins with a **function** keyword, then the function name, an opening parenthesis ('('), a semicolon-separated parameter list, a closing parenthesis (')'), a colon (:), a return type, a semi-colon and the body of the function. A function declaration is terminated by a semi-colon (;).

The parameter list of a function declaration may contain zero or more parameters. A parameter consists of an identifier, a colon and a parameter type. If two or more consecutive parameters have the same type, they could be reduced to a shorter form: a comma delimited list of these parameter names, followed by a colon (:) and the shared parameter type.

For example

`function area(a:real;b:real;c:real):real;` could be rewritten as follows

```
function area(a,b,c:real):real;
```

A return type must be a primitive type. A parameter type could be a primitive type or an array type.

The body of a function is also simply a block statement.

A procedure declaration is similar to a function one except that the procedure declaration uses keyword **procedure** instead of **function** and does not contain the return type. We will use the term procedure to refer both procedure and function unless we specify.

For example:

```
program example;
      var childShare:integer;
      function child1():real;
      begin
            childShare := 1;
            child1 := childShare;
      end;  // The ; here is required because "A function
            //declaration is terminated by a semi-colon"
      procedure child2();
      begin
            childShare := 2;
      end;
begin
      child1();
      writeIntLn(childShare);
end.
```

2.3. Block statement

A block statement begins by the keyword "**begin**" and ends up with the keyword "**end**". Between the two keywords, there may be a list of statements optionally preceded by a local

variable (and constant) declaration part (described in section 2.2.1). The list of statements may be empty.

For example:

```
begin
   //start of declaration part
   var r,s: real;
   const myPI=3.14;
   //list of statements
   r:=2.0;
   s:=r*r*myPI;

end
```

3. Types

There are 4 primitive types: **integer**, **real, boolean** and string in Crazy. In addition, Crazy supports multi-dimension array type.

The keyword **boolean** denotes a boolean type. The following operators accept their operands in boolean type: **= <> not and or**

The keyword **integer** is used to represent an integer type. Integer values can be operands of the following operators: **div mod + - * / < <= > >= = <>**

The keyword **real** represents a real type The operands of the following operators can be in real type: + - * / < <= > >=.

In general, the operands of an operator must be in the same type. However, for + - * /, their operands can be in mixed types: integer and real. In this case, the integer operand will automatically be conversed into real type.

The keyword **string** can be used for a string type. A string constant can be assigned to a variable or a parameter in an assignment statement or initialized to a string constant. There is no other operation on this type.

A variable or a parameter can be declared as an array. An array type declaration starts with the keyword array, list of dimensions, keyword **of**, and ends with an integer, real, string or boolean type. An array has at least one dimension which is described by the sequence: '[', an integer constant, and ']'. The integer constant represents the size of the corresponding dimension and it must be a positive number, i.e greater than 0. The first index of a dimension is 0.

4. Expressions

An expression is a combination of operands and operators. An operand is a variable, a constant, a function call, or an expression. A function call starts with an identifier,i.e. the function name, followed by an opening parenthesis, an optional comma-separated list of expressions, and ends with a closing parenthesis. The following table lists all Crazy operators in order of their precedence (highest to lowest).

Operators (op)	Description	Associativity	Syntax
()	Parentheses (grouping)		(<expr>)
[]	Brackets (array subscript to access array element)		<id>[<expr>]…[<expr>]

- not	Unary minus/logical negation		op <expr>
and	Logical AND	left	<expr> op <expr>
or	Logical OR	left	<expr> op <expr>
> >= < <=	Relational greater than/ less than	left	<expr> op <expr>
<> =	Relational is equal to	left	<expr> op <expr>
* / div mod	Multiplication/division/modulus	left	<expr> op<expr>
+ -	Addition/Subtraction	left	<expr> op <expr>

The type of an expression is as follows

Operands:
- id => type of id
- constant => type of constant
- function call => return type of the function

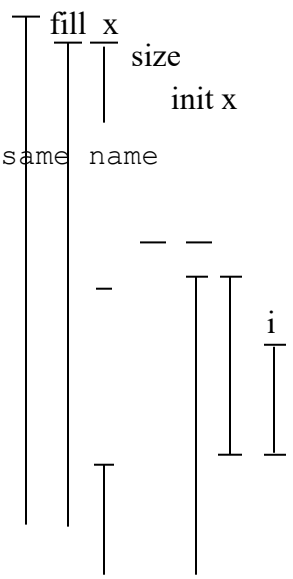
Operators:
- array access => element type
- not, and, or, >, <, >=, <=, <>,= => boolean
- div, mod => integer
- +, - ,* => integer if both operands are integers, otherwise real.
- / => real

5. Variables

There are two kinds of variables in Crazy: global and block. A global variable (or constant) is declared in the variable declaration part of a program and is visible in the whole-prgram. A block variable (or constant) is declared in a block and accessed inside the block only. A parameter of a procedure is belong to the block statement of the procedure. A block variable may hide an outside-block or global variable if they have the same name. For example,

```
program declDemo;
  var a: array[5] of integer; // global variable
  procedure fill(x:array[5] of integer);
  begin
   var a:real; // hiding global var. a
   var x:real; // WRONG because parameter x has the same name
   a:=5.9;
   init(x);
  end;
  procedure init(x:array[5] of integer);
  begin
   var i:integer; //block variable
   i:=0;
   x[i]:=a[i]; // a is global var.
  end;

begin
   fill(a);
```



```
end.                                                    — —
```

6. Statements

There are nine kinds of statements: **assignment, block, if, call, while, do while, loop do, break, continue.**

An assignment statement takes the following form:

<left handside> := <expression> ;

where <left handside> is a scalar variable or an element of an array.

The block statement was described in section 2.3.

The form of an **if** statement is:

if <expression> **then** <statement> **else** <statement>

or

if <expression> **then** <statement>

A call statement is a call to a procedure. Its form is like a function call except it always ends with a semicolon (;).

The form of a **loop** statement is as follows

loop <expression> **do** statement

The value of the expression is calculated first. If it is a positive number n then do statement will be executed n times. Otherwise, the do statement will be skipped.

A **while** statement can be written as follows

while <expression> **do** statement

while a **do while** statement is

do <list of statements> **while** <expression> ;

Note that the list of statements in a do statement may be empty.

The break and continue are only used inside a loop, while or do while statement. They are always terminated by a semi-colon.

break ;

continue ;

Their semantics are similar to those in C.

7. Built-in procedures and functions

There are 12 built-in procedures and functions specified in the following table:

Built-in function name	Parameter type	Semantic
`readInt()`	no	Read an integer number from keyboard
`writeInt(anArg)`	Integer	Write an integer number to the screen
`writeIntLn(anArg)`	Integer	Write an integer number to the screen and a new line
`readReal()`	no	Read an real number from keyboard
`writeReal(anArg)`	Real	Write a real value to the screen
`writeRealLn(anArg)`	Real	Write a real value to the screen and a new line
`readBool()`	no	Read a boolean value from keyboard

`writeBool(anArg)`	Boolean	Write a boolean value to the screen
`writeBoolLn(anArg)`	Boolean	Write a boolean value to the screen and a new line
`writeStr(anArg)`	String	Write a string to the screen
`writeStrLn(anArg)`	String	Write a string to the screen and a new line
`writeLn()`	no	Write a new line to the screen