TRAVEL ASSIST SCRIPTING LANGUAGE (TASL)

Language Reference Manual

Yogi Saxena (ys2332@columbia.edu)

1 Introduction

There are several websites that offer a web based interface to get quotes for air fare, accommodation and other services such as rental cars. The user is required to enter several pieces information related to his/her travel plans such as dates, destination and quantity. This information is not stored and the user is required to re enter when he starts a new search. Most of the users are looking for the cheapest fare for their travel. This may require multiple searches with a combination of different dates and/or destinations.

The TASL language attempts to provide a simple scripting language that would make it easier for the travelers to do their search. The user input will be stored and used for periodic searches carried out at a pre-defined interval. If the results of the search match the user defined criteria then the results could be stored in a file and/or the user could be alerted via email.

2 Lexemes

2.1 Identifiers

Identifiers refer to or identify function names, data objects, class names etc. They consist of a string of characters such as one or more uppercase or lower case alphabets and numbers. The first character of an identifier must start with an alphabet. Upper case and lower case letters are treated differently. Keywords are reserved identifiers and cannot be re defined.

2.2 Keywords

The following reserved identifiers are used as keywords:

boolean	int	void	if
else	function	while	break
return	search	repeat	email
for	float	modify	delete
reserve			

2.3 Numbers

A number is a string of digits and a decimal point. The types of Numbers supported in the language are Integer and float. An Integer consists of a sequence of digits. A floating point number consists of an integer part, a decimal point and a fraction part.

Example: 10, 100.00, 250.00

2.4 String

String is a sequence of zero or more characters. String literals are character strings surrounded by double quotes (“EWR”). String literals can include any valid character, including white-space characters and character escape sequences.

2.5 Operators

An operator is a token that specifies an operation on at least one operand, and yields some result (a value, designator, side effect or some combination). Operands are expressions or constants.

The following are the operators used in TASL.

+	-	/	*	=
= =	>	<	&	|

3 Data Types

The language defines the following data types:

Numeric: Integer represents date and quantity, float represents price.

True or False: These represent Boolean values.

4 Declarations

4.1 General declaration

All objects must be declared before use. The general syntax of a declaration is as follows:

type-specifier init-declarator-list(opt);

init-declarator-list:

declarator
init_declarator-list, declarator

The type specifiers can be int, float or boolean.

4.2 Arrays

Arrays can be declared with square brackets [ ]. The following is the syntax for declaring an array.

type-specifier declarator [constant-expression-list(opt)];

the type-specifier can be int or float.

5 Functions

5.1 Function Types

A function has the derived type "function returning type". The type can be a data type except array or a function type. A function that does not return any value is referred to as a void function.

Functions can be defined in the following ways:

1) A function definition can create a function designator, define its parameters and their types, defined the type of its return value and construct the body of the function.
2) A function declaration specifies the properties of a function defined elsewhere.

5.2 Function Definitions

A function definition includes the body of the function. Function definitions can appear in any order, and can be referenced in one or more source files. Function definitions cannot be nested.

A function definition has the following syntax:

function-definition:
declaration-specifiers (opt) declarator declaration-list(opt) compound-statement

declaration specifiers
The declaration-specifiers (type qualifier and type-specifier) can be listed in any order.
Type specifiers are: int, float and boolean.

Example:

```
main ( ) {
        search "EWR" "SFO" 10182007 10242007 2
        if ( Results( ) ) {
                email(xyz@columbia.edu);
        }
}

boolean function Results( ) {

        if ( FARE[0] < desiredValue) {
                return true;
        }
        return false;
}
```

5.3 Function Declarations

For all functions if the function definition is located after the calling function, the function must be declared before calling it.

5.4 Function Parameters and Arguments

The functions exchange information by means of parameters and arguments. The term parameter refers to any declaration within the parentheses following the function name in a function declaration or definition. The term argument refers to any expression within the parenthesis of a function call.

The following rules apply to the parameters and arguments:

1) The number of arguments in a function call must be the same as the number of parameters declared by the function definition.
2) The maximum number of arguments (and corresponding parameters) is 50 for a single function.
3) Arguments are separated by commas. The comma is not an operator in this context and the arguments can be evaluated by the compiler in any order.
4) Arguments are passed by value.
5) Modifying a parameter does not modify the corresponding argument passed by the function call.

5.5 Function invocation and return

Function Call:
A function call can be a single statement followed by a ";".

Return Statement:
The return statement is used to return from the function at the point the return statement is specified. Return statements can return a value. The return statement is terminated by a ";".

5.6 Built-In Functions

5.6.1 print()

Sends the output to the screen

5.6.2 load()

File I/O functions.

5.6.3 email (xxx@yyy.edu)

Emails the results to the supplied email id.

6 Expressions and Operators

6.1 Primary Expressions

Simple expressions are called primary expressions; they denote values. Primary expressions include previously declared identifiers, constants, string literals and parenthesized expressions.

Primary expressions have the following syntax:

primary expression:
identifier
constant
strings
(expression)

6.1.1 Identifier

An identifier is a primary expression provided it is declared as designating an object or a function

6.1.2 Constant

A constant is a primary expression. Its type depends on its form (ie either boolean or int or float).

6.1.3 Expression

A parenthesized expression is a primary expression whose type and value are identical to those of the unadorned expression. The presence of parentheses does not affect whether the expression is an lvalue.

6.2 Postfix Expressions

Postfix expressions include array references, function calls and postfix increment and decrement expressions. The operators in postfix expressions have left-to-right associativity. Postfix expressions have the following syntax:

Postfix-expression:
Primary-expression
postfix-expression ++
postfix-expression [expression]
postfix-expression (argument-expression-list)

6.2.1 Array References

A postfix expression followed by an expression in square brackets in a postfix expression denoting a subscripted array reference.

array-reference:
postfix-expression [expression]

6.2.2 Function Calls

A function call is a postfix expression consisting of a function designator followed by parentheses. The order of evaluation of any expressions in the function parameter list is undefined, but there is a sequence point before the actual call. The parentheses can contain a list of arguments (separated by commas) or can be empty.

Function calls have the following syntax:

Function-call:
Postfix-expression(argument-expression-list(opt))

Argument-expression-list(opt):
assignment-expression
argument-expression-list(opt), assignment-expression

6.3 Unary Operators

Unary expressions are formed by combining a unary operator with a single operand. All unary operators are of equal precedence and have right-to-left associativity.
Unary-operator: one of & * + -

6.4 Binary Operators

The binary operators are categorized as follows:

- Multiplicative operators: multiplication (*) and division (/)
- Additive operators: addition (+) and subtraction (-)
- Relational Operators: less than (<), greater then (>)
- Equality operators: equality (==) and inequality (!=)
- Logical Operators: AND (&) and OR (|)

6.4.1 Multiplicative Operators:

The multiplicative operators are *, /. Operands must have the arithmetic type. Operands are converted, if necessary, according to the usual arithmetic conversion rules.

The * operator performs multiplication and the / operator performs division.

6.4.2 Additive operators:

The additive operators + and – perform addition and subtraction. Operands are converted, if necessary, according to the usual arithmetic conversion rules.

6.4.3 Relational operators:

The relational operators compare two operands and produce a result of type int. The result is 0 if the relation is false and 1 if it is true. The operators are: less than (<), greater than (>). Both operands must have an arithmetic type.

The relational operators associate from left to right.

6.4.4 Equality Operators:

The equality operator, equal (==) and not equal (! =), produce a result of type int, so that the result is 1 if both operands have the same value and 0 if they do not.

6.4.5 Logical Operators:

The logical operators are AND (&) and OR (|). These operators guarantee left-to-right evaluation. The result of the expression (of type double) is either 0 (false) or 1 (true). The operands need not have the same type, but both types must be scalar. If the compiler can make an evaluation by examining only the left operand, the right operand is not evaluated.

6.5 Assignment Operators

Assignment results in the value of the target variable after the assignment. They can be used as sub-expressions in larger expressions.

Assignment expression has two operands: a modifiable value on the left and an expression on the right. A simple assignment consists of the equal sign (=) between the two operands:

Exp1=Exp2;

The value of expression Exp2 is assigned to Exp1. The type is the type of Exp1, and the result is the value of Exp1 after completion of the operation.

7 Statements

The following types of statements are supported.

- Expression statements
- Compound statements
- Selection statements
- Iteration statements
- Break statements
- search statement
- modify statement
- delete statement
- reserve statement

7.1 Expression statement

Most statements are expression statements, which have the form

expression-statement:
expression(opt);

7.2 Compound statements

Several statements can be combined to form a compound statement. The body of a function is a compound statement.

compound-statement:
{ declaration-list(opt) statement-list (opt)}

declaration-list:
declaration
declaration-list declaration
statement-list:
statement
statement-list statement

7.3 Selection statements

The if statement

If expression

then
statement

else (opt)
statement

The statement following the control expression is executed if the value of the control expression is true. An if statement can be written with an optional else clause that is executed if the control expression is false.

7.4 The iteration statement

Iteration statements are used for looping.

iteration-statement:
while (expression) statement
for (expression (opt) ; expression (opt); expression (opt)) statement

In the while statement the substatement is executed repeatedly so long as the value of the expression remains unequal to 0.

In the for statement, the first expression is evaluated once, and thus specifies initialization of the loop. The second expression must be a arithmetic type and is evaluated before each iteration, and if it becomes equal to 0 the for is terminated.

7.5 Break statement

The break statement causes the termination of the enclosing while and for statements. The control is passed to the statement following the terminated statement.

7.6 Return statement

The return statement causes the program control to return to the caller. An optional expression following the return keyword will cause the function to return the lvalue of the expression of the caller. If required, the expression is converted, as if by assignment, to the type of function in which it appears.

7.7 Search statement

The search statement is used to query the database for fares. The following is the syntax:

search origin_Identifier destination_Identifier departure_date return_date quantity

7.8 Modify Statement

The modify statement is used to modify the database for reservations. The following is the syntax:

modify origin_Identifier destination_Identifier departure_date return_date quantity

7.9 Delete Statement

The delete statement is used to delete the record of a reservation in the database. The following is the syntax:

delete origin_Identifier destination_Identifier departure_date return_date quantity

7.10 Reserve Statement

The reserve statement is used to insert a reservation in the database. The following is the syntax:

reserve origin_Identifier destination_Identifier departure_date return_date quantity

8 References

[1] C Reference Manual, Dennis and Ritchie