|  |
| --- |
| User Manual |
| T[] Programming Language |
| Version 0.1 |

|  |
| --- |
| Travis O’Donnell  5/15/2014 |

Table of Contents

[Introduction 2](#_Toc387909768)

[Chapters 2](#_Toc387909769)

[How to start 2](#_Toc387909770)

[Data types 3](#_Toc387909771)

[Operators 3](#_Toc387909772)

[Control Statements 4](#_Toc387909773)

[Loops 4](#_Toc387909774)

[Figure 1: Usage of Mini 3](#_Toc387909781)

[Figure 2: Usage of Mice 3](#_Toc387909782)

# Introduction

The T[] programming language is a LALR grammar based off of a PL/SQL scripting programming language. The name of the programming language, T[], represents the initials of the software developer Travis O’Donnell. The compiler was developed for the class “Compiler Designs and Implementation” at Frostburg State University, taught by Dr. Mohsen Chitsaz. Throughout the semester, the class was taught common techniques on building compilers, how compilers are structured, and how compilers operate. Over the semester, the class took the knowledge that had been taught to them and individually built their own compilers from a grammar that was given by Dr. Chitsaz.

# Chapters

## How to start

To run the compiler, the user needs to have installed

* Java SE Runtime Environment 1.7
* A simple text editor (Notepad, Vim, or your favorite IDE)

[1] Writing code for the T[] language can be done in your favorite text editor. For a list of example programs, see Appendix. Once code has been written, save input file within the folder that contains Compiler.jar.

[2] To compile the code, the user will need to open up a terminal and navigate to the directory that contains the Compiler.jar file and the input file you just created. To use the compiler, the following code must be executed

java –jar Compiler.jar “inputFile”

If the compiler executes with no errors, it will output

“Compiled with success! Run the output.t file into mini and mice for execution”

[3] The output file ‘output.t’ will be created automatically if it has not been done yet. It will need to be run through the external programs mini and mice. Mini will assemble the output into machine language, while Mice executes the machine language.

[4] Switch the terminal to the directory that contains the external applications.

Run the following command

java mini

Type in the input code file ‘output.t’ and press enter.

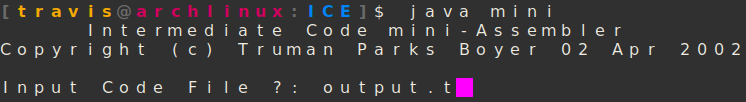


Figure : Usage of Mini

Mini will create the file ‘output.t.out’ which will need to ran through ‘Mice’.

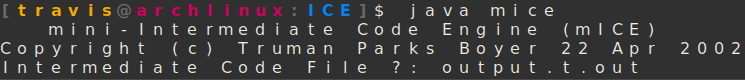


Figure : Usage of Mice

## Data types

* Boolean
  + **TRUE**
  + **FALSE**
  + **NULL**
* **CHAR**
  + All characters are to be wrapped in single quotation marks
* **INT**
  + A number within in the range of (-99999999,99999999)
* **SMALLINT**
  + A number within the range of (-9999,9999)
* **POSITIVE**
  + A number within the range of (0,99999999)

## Operators

**<** | **>** | **<=** | **<=** | **<>** | **+** | **-** | **\*** | **/** | **MOD**

## Control Statements

The simple format of a control statement is

IF expression THEN

Statement

END IF

|  |  |
| --- | --- |
| **T[] programming Language Code** | **Generated Intermediate Code** |
| IF 5 > 3 THEN  DBMS\_OUTPUT.NEW\_LINE  END IF | 0 JGT #5, #3, #3  1 STO #0, , 1  2 JMP , , #4  3 STO #1, , 1  4 JNE 1, #1, #6  5 SYS #0, ,  6 … |

6 Example: Control Statement

## Loops

The T[] programming language can run very basic loops. The grammar given to the class may have not been set up correctly to run more advanced loops. One huge problem is that the user cannot update a variable inside the while loop. So since a while loop runs based off of an expression, the expression is always going to be true or false. If the expression is true, the while loop runs indefinite. The format of a while loop is structured as

WHILE expression LOOP

Statement

END LOOP

|  |  |
| --- | --- |
| **T[] programming Language Code** | **Generated Intermediate Code** |
| WHILE myNum > 3 LOOP  DBMS\_OUTPUT.PUT ( myNum );  END LOOP | 0 JGT #5, #4, #4  1 STO #0, , 1  2 JMP , , #4  3 STO #1, , 1  4 JNE 1, #1, #7  5 sys #0, ,  6 jmp , , #0  7 … |

# Appendix

## Summary of Syntax

|  |  |
| --- | --- |
| block | declarations compound\_statement $ |
| declarations | DECLARE declare\_rest  λ |
| declare\_rest | id data\_type default ; declare\_rest |
| default | := righthandside  λ |
| data\_type | characters  numbers  BOOLEAN |
| characters | CHAR |
| size | num size\_option  λ |
| size\_option | , num  λ |
| numbers | NUMBER ( size )  INT ( size )  SMALLINT ( size )  POSITIVE ( size ) |
| compound\_statement | BEGIN optional\_statements END ; |
| optional\_statements | NULL ;  statement\_list |
| statement\_list | statement  statement\_list ; statement |
| statement | lefthandside  compound\_statement  DBMS\_OUTPUT.PUT\_LINE ( id )  DBMS\_OUTPUT.PUT ( id )  DBMS\_OUTPUT.NEW\_LINE  & id  IF expression THEN statement END IF  WHILE expression LOOP statement END LOOP |
| lefthandside | id := righthandside |
| righthandside | expression  ‘ c ‘ |
| expression | simple\_expression  simple\_expression relop simple\_expression |
| simple\_expression | term  simple\_expression addop term |
| term | factor  term mulop factor |
| factor | id  num  TRUE  FALSE  NULL  NOT factor |
| relop | >  >=  ==  <=  <  <> |
| addop | +  - |
| mulop | \*  /  MOD |

## Summary of Errors

|  |  |
| --- | --- |
| Could not read file | Compiler could not open specified input file |
| Unexpected end of file | Compiler was not finished parsing before it hit the end of the file |
| Unexpected token | While parser is reducing, an incorrect token was found. Error will print out the token id of the token found and the expected token id. |
| Type Error | User is trying to perform arithmetic operations to a non-number data type |
| Token name already found in symbol table | Variable name was already used |
| Mismatch data types | User is trying to assign a wrong data type |
| Unknown token found | Variable was never initialized in DECLARE |
| Internal Compiler Error | Bad syntax |