How to use the PSCode Interpreter

The PSCode programming language is a procedural programming language designed to make writing basic code extremely simple and user-friendly. The name PSCode was chosen because of its similarity to “pseudocode,” which I hand-wrote as a high school student while studying for my HSC. Below are some steps to use the interpreter.

Licensing

You are free to use my interpreter in any way, including modifications. Just make sure to appropriately reference my work when using code snippets in your own projects. Also, if modifying the interpreter, make sure to have a backup of the original, in case your modifications cause errors.

Steps to use

1. Download the PSCode Interpreter.cpp file and the MyOutput.txt file
2. Write your PSCode on a txt file, saved in the same folder as the interpreter and MyOutput.txt file
3. Open the interpreter in your C++ IDE
4. Run the interpreter file
5. When prompted by the command line, enter the name of txt file that contains your source code and press enter
6. Close the program and check the output file

How to write in PSCode

There are three types of statements you can write. They are **sequence**, **selection**, and **repetition**. All statements are represented by these three types. Below I have written the EBNF definition and corresponding examples for all valid statements.

Sequence

|  |  |  |
| --- | --- | --- |
| **Statement Type** | **EBNF Definition** | **PSCode Example** |
| Output | DISPLAY <Variable>|<Value> {+ <Variable>|<Value>}  Where a <Variable> is defined as:  Letter {Letter|Digit}  Letter = A...Z && a...z  Digit = 0...9  And a <value> can be any of the 4 data types:  string: "<character>" //includes any character combination inbetween double-quotes  char: '<some character>' //includes any ASCII character  int: Digit{Digit} // integer  float: Digit.{Digit} //floating-point  bool: TRUE|FALSE //boolean  Note that every variable must have a fixed data type and a value of that same type. | DISPLAY “age is ” + age |
| Variable Assignment | SET <variable> AS <value>  Note that the variable must have the same type as the value it is being assigned to. | SET age AS int |
| Variable Expression | <variable> = <variable> | <value> {<Arithmetic Operator> <variable>|<value>}  Where an <Arithmetic Operator> is:  Addition: +  Subtraction: -  Multiplication: \*  Division: /  Modulo operator: %  Note that brackets, (), are supported.  Also note that expressions with strings are dealt with completely differently.  In a string expression the EBNF is this:  <string variable> = <string variable> | <string value> { + <string variable | <string value>}  While brackets are supported, no arithmetic operators other than addition are allowed. | x = x \* (y - 1) + z |

Selection

An IF-statement has the following syntax

IF <conditional statement> THEN

    {<statement>}

{ELSE IF <conditional statement> THEN}

    {<statement}

[ELSE]

    {<statement>}

END IF

Where a <conditional statement> is defined as:

<condition> {AND|OR <condition>}

And a <condition> is defined as:

<variable>|<value> <Equality Operator> <variable|value>

Where an <Equality Operator> is:

IS EQUAL TO: =

IS NOT EQUAL TO: <>

IS GREATER THAN: >

IS GREATER THAN OR EQUAL TO: >=

IS LESS THAN: <

IS LESS THAN OR EQUAL TO: <=

Note that nesting is valid for both selection and repetition. Also, remember that <condition>s and <conditional statement>s are not <statement>s, but rather things to be evaluated in an IF statement or WHILE loop.

In PSCODE an example of a valid IF statement is:

IF age >= 18 THEN

DISPLAY “You can vote!”

IF age >= 21 THEN

DISPLAY “You can vote in America!”

END IF

ELSE IF age >= 16 THEN

DISPLAY “You can learn to drive!”

ELSE

DISPLAY “You poor small child!”

END IF

Repetition

|  |  |  |
| --- | --- | --- |
| **Statement Type** | **EBNF Definition** | **PSCode Example** |
| FOR loop | FOR <variable> = <value> TO <value>  {<statement>}  NEXT  Note that the <variable> after the FOR token in this statement must be previously declared to avoid a runtime error. | FOR x = a TO b  x = x + 1  NEXT |
| WHILE loop | WHILE <conditional statement>  {<statement>}  END WHILE  Note that the <conditional statement> definition for a WHILE loop is the same as the previously mentioned definition for <conditional statement> for an IF statement. | WHILE x <> 0  x = x – 1  WHILE y < 100  y = z \* x  END WHILE  END WHILE |