“Happy” Language Specification

Southeast Missouri State University

CS390: Programming languages

Professor: Dr. Lowe Bob

11/07/2022

Members: Tuyen Nguyen, Xiao Pu, Dongjun Li,

Shashik Mihiranga Rebera, Swapnil Moon

## I. Language Overview

### Introduction to the Language

This language is named “happy.” Giving the language such a name, the authors of the language wish coders to have “happy” coding moments with the language.

The language is built on top of the “fun” language. It has subtle changes in the syntax and additional functionalities/features compared to “fun”.

### Discussion of Influences

Several factors influence the creation of the language.

1. Firstly, the language supports basic features that are shared by most of the languages out there such as functions, calculations, branch statements, loop statements, input, and print.
2. The language is invented to successfully run the three programs “Hello World”, “Implementation and Test of Bubble Sort”, “Tower of Hanoi Solver”
3. Desired features and language syntax are suggested by our group members.
4. The language is inspired by other languages:

* The language starts with the "MAIN" function and can follow with one or many functions. Each function’s block starts and ends with curly braces (JAVA)
* The “swap” statement makes coding “Bubble Sort” much simpler, so we decide to keep it (FUN)
* For array, our language provides a static method “LEN” to retrieve an array’s current length (Python)
* The language provides “println” in addition to a "print" statement to print a list of arguments and then enters the next line (JAVA)
* “input” statement can go with a prompt (BASIC)

5. We, the language’s authors are also ambitious to provide the language with:

* A “for-loop”: is quite advantageous, we can write assignment statements starting the loop, stopping conditions to check at every iteration, and assignment statements happening at every iteration just in a single line of code.
* Conditions with “and” and “or” operators
* A stack with push and pop operations: help solve the “Tower of Hanoi”
* Recursive functions: help code a recursive solution for the problem “Tower of Hanoi”

### BNF Description of Syntax

<program> ::= <main-fun> <fun-declist>

| <main-fun>

<main-fun> ::= "MAIN" "(" ")" <block>

<fun-declist> ::= <fun-declist> <fun>

| <fun>

<fun> ::= <id> "(" ")" <block>

|<id> "(" <param-list> ")" <block>

<param-list> ::= <param-decl> "," <param-list>

| <param-decl>

<param-decl> ::= < type > <id>

| < type > "[]" <id>

| < type > "|\_|" <id>

< type > ::= "NUMBER"

| "STRING"

<block> ::= "{" <var-stmt-list>"}"

<var-stmt-list> ::= <var-decl>

| <stmt>

| <var-decl> <var-stmt-list>

| <stmt> <var- stmt-list>

<var-decl> ::= <type> <id>

|<type> "[" <expr> "]" <id>

|<type> “|\_|" <id>

<stmt> ::= <assign>

| <swap>

| <branch>

| <for-loop>

| <expr>

| <print>

| <println>

| <read>

| <stack-push>

| <stack-pop>

<assign> ::= <ref> ":=" <expr>

<swap> ::= <ref> ":=:" <ref>

<branch> ::= "IF" <condition-list> <block> “ELSE” <block>

<for-loop> ::= "FOR" "(" <assign-list> ";" <condition-list> “;" <assign-list>")" <block>

|"FOR" "(" ";" <condition-list> ";" <assign-list>")" <block>

<condition-list> ::= <condition>

| <condition> "&&" <condition-list>

| <condition> "||" <condition-list>

<condition> ::= <expr> "=" <expr>

|<expr> "<>" <expr>

|<expr> "<" <expr>

|<expr> "<=" <expr>

|<expr> ">" <expr>

|<expr> ">=" <expr>

|”(“ <condition-list “)”

<assign-list> ::= <assign>

|<assign> "," <assign-list>

<stack-push> ::= < id > “.” “PUSH” “(” <expr> “)”

<stack-pop> ::= < id > “.” “POP” “(” “)”

<expr> ::= <term>

|<expr> "+" <term>

|<expr> "-" <term>

<term> ::= <factor>

| <term> "\*" <factor>

| <term> "/" <factor>

<factor> ::= <exponent>

| <factor> "\*\*" <factor>

<exponent> ::= "(" <expr> ")"

| <ref>

| <literal>

| <call>

| "LEN" "(" <id> ")"

<print> ::= "PRINT" <arg-list>

<println> ::= "PRINTLN" <arg-list>

<arg-list> ::= <expr>

|<expr> "," <arg-list>

<read> ::= "INPUT" <string> ";" <ref-list>

|"INPUT" <ref-list>

<ref-list> ::= <ref>

|<ref> "," <ref-list>

<ref> ::= <id>

| <id> "[" <arg-list> "]"

<literal> ::= <number>

|<string>

<call> ::= <id> "(" <arg-list> ")"

|<id> "(" ")"

<number> ::= <integer>

<integer> "." <integer>

< integer > ::= < digit >

| < integer > < digit >

<string> ::= "\"" <characters> "\""

<characters> ::= <character>

| <characters> <character>

<id> ::= <letter>

| "\_"

| <id> <digit>

| <id><letter>

Informally Stated (not in BNF)

==============================

< digit > Any digit 0-9

< character > Any character, with normal escapes \n, \t, \', \"

< letter> Any upper- or lower-case letter

Comments begin with "#" and extend to the end of a line

## II. Example Programs (Written in your programming language)

### Hello World

main ()

{

println "Hello, world!"

}

### Implementation and Test of Bubble Sort

main ()

{

NUMBER n

input "How many number? "; n

NUMBER [n] array

println "Enter n numbers?”

NUMBER i

for (i := 0; i < n; i := i+1){

input array[i]

}

bubbleSort(array)

println "Sorted: "

for(i := 0; i < n; i := i+1){

print array[i], " "

}

}

bubbleSort (NUMBER [] array)

{

NUMBER next

NUMBER i

NUMBER j

for (i :=0, next := 1; i < len(array) && next=1; i := i+1)

{

for (j:=0, next:=0; j<len(array)-1; j:=j+1)

{

if array[j]>array[j+1]

{

next := 1

array[j] :=: array[j+1]

}

}

}

}

### Tower of Hanoi Solver

main(){

NUMBER n

input "Enter number of disks: "; n

NUMBER |\_| source

NUMBER |\_| aux

NUMBER |\_| target

println "Disks in the source: "

NUMBER i

for (i:=n; i>=1; i:=i-1){

println i

source.push(i)

}

move(n, source, target, aux)

println "Disks in the target: "

for (i:=0; i<len(target); i:=i+1){

println target[i]

}

}

move(n, |\_| source, |\_| target, |\_| aux){

if n>0

{

move(n-1, source, aux, target)

NUMBER last

last := source.pop()

target.push(last)

move(n-1, aux, target, source)

}

}

## III. Language Reference Manual

### Operator Precedence Chart and Operator Explanations

|  |  |  |
| --- | --- | --- |
| Precedence | Operator | Type |
| 6 | ()  []  . | Parentheses  Array subscript  Member method selection |
| 5 | \*\* | Power |
| 4 | \*  / | Multiplication  Division |
| 3 | +  - | Addition  Subtraction |
| 2 | <  <=  >  >=  =  <> | Less than  Less than or equal  Greater than  Greater than or equal  Equal to  Not equal to |
| 1 | &&  || | Logical AND  Logical OR |
| 0 | :=  :=: | Assignment  Swap |

### Semantic Information

#### Arrays and Indices

* + An array declaration consists of declaring:
    - Type (number or string)
    - Size
  + Array declaration in BNF: <type> "[" <expr> "]" <id>
  + Array indices start at 0
  + Indexing an array by using square brackets “[]”
  + To get the array’s size, use LEN

#### Stacks

* + Stack declaration only require specifying the type of elements stored in the stack
  + Stack declaration in BNF: <type> “||” <id>
  + Stack allows indexing like with an array
  + Stack allows two operations: push and pop
    - <stack-push> ::= < id > “.” “PUSH” “(” <expr> “)”
    - <stack-pop> ::= < id > “.” “POP” “(” “)”
  + To get the stack’s size, use LEN

#### Type Coercion Rules

All numbers are coerced into floating point numbers and mathematical operations performed on the floating-point numbers.

#### Parameter Passing Information

A function parameter list contains multiple parameter declarations. Each parameter declaration consists of declaring type and identifier.

Happy Lang supports: two primitive types, and two reference types: NUMBER, STRING, NUMBER|STRING [], and NUMBER|STRING ||