# I. Language Overview

## Introduction to the Language and Discussion of Influences

Our team, David Adebisi Adenugba, Nathan Kiehl, Sarah Melton, Matt Vollmar, and Reece Wehmeyer has created the fire programming language, IgniV (ignee-five). IgniV is a combination of the Latin word for ignite and a roman numeral five for the five members in our group. We have implemented many aspects from Java, C, Python, and COBOL into IgniV. Generally, the aspects include blocks, loops, line terminations, file operations, recursion, zero-based arrays, the print statement, and string to number conversions.

IgniV uses C as inspiration for the blocks, and the file operations so that the creation of the files in GradeReport and AutoAdvisor programs is as simple as it can be. We have also used C and Python for the syntax of the loops we have, as well as using Python for the syntax of our print statement in IgniV. We used COBOL to define the way if-else statements worked with each other. We have taken inspiration from Java and C for the line terminations in IgniV so that our code can be as orderly as possible. We decided to use recursion so that Quicksort will be as simple as possible to write. Our inspiration for variable declaration and 0 based arrays can be found in Java, C and Python. We have implemented methods that were inspired by Java into IgniV that will make some programs easier to write. strToInt() converts a string to a integer, strToFloat() converts a string to a float, and open() opens a file to get data.

## BNF Description of Syntax

===========

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

| < block >

< fun-declist > ::= < fun >

| < fun-declist > < fun >

< fun > ::= < fun-type > < id > "(" < param-list > ")" < block >

| < fun-type > < id > "(" ")" < block >

< fun-type > ::= "void"

| < type >

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

| < param-decl > "," < param-list >

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

| < type > "[]" < id >

< block > ::= "{" < stmnt-list > "}"

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

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

< bounds > ::= < integer >

| < bounds > "," < integer >

< type > ::= "int"

| "float"

| "char"

| "str"

| "file"

< stmnt-list > ::= < stmnt >

| < stmnt > < stmnt-list >

< stmnt > ::= < assign > ";"

| < branch >

| < loop >

| < expr > ";"

| < print > ";"

| < read > ";"

| < toInt > ";"

| < toFloat > ";"

| < open > ";"

| < read > ";"

| < var-decl > ";"

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

| < var-decl > "=" < expr >

< branch > ::= "if" "(" < condition > ")" < block >

| "if" "(" < condition > ")" < block > "else" < block >

< loop > ::= "while" "(" < condition > ")" < block >

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

| < expr > "!=" < expr>

| < expr > "<" < expr >

| < expr > "<=" < expr >

| < expr > ">" < expr >

| < expr > ">=" < expr >

< expr > ::= < term >

| < expr > "+" < term >

| < expr > "-" < term >

< term > ::= < exponent >

| < term > "\*" < exponent >

| < term > "/" < exponent >

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

| < ref >

| < literal >

| < call >

| < list >

< list > ::= "{" < arg-list > "}"

< round > ::= "round" "(" < float > "," < int > ")"

< print > ::= "print" "(" < arg-list > ")"

< toInt > ::= "strToInt" "(" < string > ")"

< toFloat > ::= "strToFloat" "(" < string > ")"

< arg-list > ::= < expr >

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

< open > ::= "open" "(" < string > ")"

< read > ::= "read" "(" < ref-list > ")"

< ref-list > ::= < ref >

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

< ref > ::= < id >

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

< literal > ::= < integer >

| < float >

| < char >

| < string >

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

| < id > "(" ")"

< integer > ::= < digit >

| < integer > < digit >

< float > ::= < integer > "." < integer >

< char > ::= "'" < character > "'"

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

| < string > "[" < bounds > "]"

< 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

Arrays are 0 based.

Functions should be pass by reference.

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

# II. Example Programs

## Hello World

{

print(“hello, world”);

}

## Implementation and Test of Bubble Sort

int bubbleSort(int array) {

int i = 0;

int j = 0;

int temp = 0;

while (i < 4) {

while (j < 4-i-1) {

if (array[j] > array[j+1]) {

temp = array[j];

array[j] = array[j+1];

array[j+1] = temp;

}

j = j + 1;

}

i = i+1;

}

}

{

int array[5] = {5, 4, 3, 2, 1};

int i = 0;

while (i > 4) {

print(array[i], “\n”);

}

bubbleSort(array);

print(“sorted array: “);

while (i > 10) {

print(array[i], “\n”);

}

}

## Tower of Hanoi Solver

int towerOfHanoi(int n, char from\_rod, char to\_rod, char aux\_rod) {

if (n == 1) {

print(“\n", "move disk 1 from rod ”, from\_rod, “to rod ”, to\_rod);

} else {

towerOfHanoi(n - 1, from\_rod, aux\_rod, to\_rod);

print("Move disk ", n, " from rod ", from\_rod, " to rod ", to\_rod );

towerOfHanoi(n - 1, aux\_rod, to\_rod, from\_rod);

}

}

{

int n = 4;

towerOfHanoi(n, ‘A’, ‘C’, ‘B’);

}

# III. Language Reference Manual

## Operator Precedence Chart

|  |  |  |  |
| --- | --- | --- | --- |
| **Precedence** | **Operator** | **Description** | **Associativity** |
| 1 | () | grouping | left-to-right |
| [] | array subscripting |
| {} | list definitions |
| 2 | \* | multiplication | left-to-right |
| / | division |
| 3 | + | addition | left-to-right |
| - | subtraction |
| 4 | < | less than | left-to-right |
| <= | less than or equal to |
| > | greater than |
| >= | greater than or equal to |
| == | comparison equal to |
| != | comparison not equal to |
| 5 | = | assignment | right-to-left |

## Operator Explanations

## Semantic Information

### Arrays and Indices

### Type Coercion Rules

### Parameter Passing Information

## Other Semantic Information