IgniV - Reference Manual

David Adebisi Adenugba, Nathan Kiehl, Sarah Melton, Matt Vollmar, and Reece Wehmeyer

November 8, 2021

1 Language Overview

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.

2 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
```

3 Example Programs

3.1 Hello World

```
1  {
2      print("Hello, World!");
3  }
```

3.2 Implementation and Test of Bubble Sort

```
1  int bubbleSort(int array) {
2          int i = 0;
3          int j = 0;
4          int temp = 0;
5          while (i < 4) {
6                  while (j < 4-i-1) {
7                          if (array[j] > array[j+1]) {
8                                  temp = array[j];
9                                  array[j] = array[j+1];
10                                 array[j+1] = temp;
11                         }
12                         j = j + 1;
13                 }
14                 i = i+1;
15         }
16 }
17
18 {
19         int array[5] = {5, 4, 3, 2, 1};
20
21         int i = 0;
22         while (i > 4) {
23                 print(array[i], "\n");
24         }
25
26         bubbleSort(array);
27         print("sorted array: ");
28
29         while (i > 10) {
30                 print(array[i], "\n");
31         }
32 }
```

3.3 Tower of Hanoi Solver

```
1  int towerOfHanoi(int n, char from_rod, char to_rod, char aux_rod) {
2          if (n == 1) {
3                  print("\n", "move disk 1 from rod ", from_rod, "to rod ",
                       to_rod);
4          } else {
5                  towerOfHanoi(n - 1, from_rod, aux_rod, to_rod);
6                  print("Move disk ", n, " from rod ", from_rod, " to rod
                      ", to_rod );
7                  towerOfHanoi(n - 1, aux_rod, to_rod, from_rod);
8          }
9  }
10
11 {
12         int n = 4;
13         towerOfHanoi(n, 'A', 'C', 'B');
14 }
```

4 Language Reference Manual

4.1 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

4.2 Operator Explanations

- **Parentheses "()"** – Parentheses are used for encapsulating arguments or other required data when making calls, creating functions, or declaring conditions for loops, not unlike other modern languages.
- **Brackets "[]"** – Like most languages, IgniV uses brackets for declaring array variables and specifying their size.
- **Curly Braces "{}"** – Curly braces are used to encapsulate either blocks or lists. All statements and data for a loop or function should be encased by curly braces.
- **Multiplication "*"** – The asterisk is used to multiply numbers just like on a calculator.
- **Division "/"** – The forward slash is used to divide one number by another just like on a calculator.
- **Addition "+"** – The plus sign is used to add numbers just like on a calculator.
- **Subtraction "-"** – The hyphen is used to subtract numbers just like on a calculator.
- **Less Than "<"** – The less than operator is used when comparing two values. The value on the left should be less than the value on the right to return true.
- **Less Than or Equal To "<="** – The less than or equal to operator is used when comparing two values. The value on the left should be less than or the same as the value on the right to return true.
- **Greater Than ">"** – The greater than operator is used when comparing two values. The value on the left should be more than the value on the right to return true.

- **Greater Than or Equal to ">="** – The greater than or equal to operator is used when comparing two values. The value on the left should be more than or the same as the value on the right to return true.
- **Equal To "=="** – The equal to operator is used when comparing two values. The values being compared should be the exact same to return true.
- **Not Equal To "!="** – The not equal to operator is used when comparing two values. The values being compared should be different to return true.
- **Assignment "="** – The assignment operator is used when assigning an expression, reference, literal, call, or list to a variable. This should not be confused with the comparison equal to operator.

4.3 Semantic Information

- **Arrays and Indices** - Arrays exist in IgniV, can be multidimensional, and start at index 0.
- **Type Coercion Rules** - IgniV only supports two kinds of type coercion. Strings can be converted to either an integer type or a float type.
- **Parameter Passing Information** - In IgniV, arrays are passed by reference. This saves memory and time. Everything else is passed by value.
- **Comments** - As stated in the BNF description, comments are one line long and start with a "#" character.
- **Line Termination** - Like Java and C, statements should end with a semicolon (";") to clarify separation of commands.