pokelang

Pokelang is a general purpose, pokemon-based programming language. It was developed as a six-month project for the elective course of Programming Languages at Universidad Simón Bolívar.

This language supports most of common types, arithmetic and logical operators, control structures (if, while, for, and case), procedures and functions, recursion, block nesting, line and block comments, compound data structures (record) and unions. Likewise, this is an imperative and strongly typed language, it is designed to be compiled. In this lenguage you might find some similarities with C, python, ruby and haskell. Learn it if you want to be the very best, like no one ever was, to compile it's your real test and to program is your cause.

This document is designed to give the specification for the language syntax and semantics, required to know in order to implement any program using it.

Lexical considerations

The following are keywords. They are all reserved, which means they cannot be used as identifiers or redefined.

```
pINTachu, BOOLbasaur, squirtrue, squirfalse, CHARmander,
VOIDtorb, butterFloat, STRUCTtabuzz, arcticUNION,
ENUManyte, GLOBAt, nullikarp, procball, funcball,
vamo_a_para, vamo_a_segui, vamos_a_retorna, vamo_a_sali,
vamo_a_lee, vamo_a_escribi, vamo_a_imprimi, vamo_a_itera,
vamo_mientra, vamo_a_para, vamo_a_segui, vamos_a_retorna,
vamo_a_sali, si, y_si, si_no, vamo_a_empeza, vamo_a_calmano,
atrapar, liberar
```

An identifier is a sequence of letters, digits, underscores and the character "?", starting always with a letter. This language is case-sensitive and has no size limit for identifiers.

In the case of having an identifier that starts with `poke`, we"re speaking of a data type identifier.

Whitespace (i.e. spaces, tabs, and newlines) serves to separate tokens, but is otherwise ignored. Keywords and identifiers must be separated by whitespace or a token that is neither a keyword nor an identifier. `vamo_a_sufri` is a single identifier, not three keywords. `vamo_a sufri` and `(pINTachu)vamo_a` scans both as two tokens.

A boolean (BOOLbasaur) constant is either `squirtrue` or `squirfalse`. Like keywords, these words are reserved.

An integer constant can only be specified in decimal (base 10). A decimal integer is a sequence of decimal digits (0-9). Examples of valid integers: `8`, `12`, `-50`, `9999999`.

A float constant is a sequence of digits, a period, followed by any sequence of digits, maybe none. Thus, `.12` is not a valid double but both `0.12` and `12.` are valid. A double can also have an optional exponent, e.g., 12.2E+2 For a float in this sort of scientific notation, the decimal point is required, likewise the sign of the exponent, and the E can be lower or upper case. As above, `.12E+2` and `0.12E2` are invalid, but 12.E+2 is valid. Leading zeroes on the mantissa and exponent are allowed.

A string constant is a sequence of characters enclosed in double or simple quotes (`"|"`). Strings can contain any character except a newline (must be especified by `\n`) or quote. A string must start and end on a single line, it cannot be split over multiple lines:

```
>"this string is missing its close quote
this is not a part of the string above

>"even though,\n this one is a valid example split in two lines."

>"this is," this is not"

>"some declarations:
 CHARmander  charma;
 pINTachu    pika  = 2;
 butterFLOAT bfree = 3.4e-10;"
```

Operators

Operators and punctuation characters used by the language includes:

`*`, `+`, `-`, `^`, `/`, `//`, `%`, `<`, `<=`, `>`, `>=`, `=`, `==`, `!=`, `&&`, `and`, `||`, `or`, `!`, `!!`, `;`, `.`, `[`, `]`, `(`, `)`, `{`, `}`, `+=`, `*=`

A single-line comment is started by `#` and extends to the end of the line. Multi-line comments start with `--` and end with the first subsequent `--` . Any symbol is allowed in a comment except the sequence `--` which ends the current comment. Multi-line comments do not nest.

Grammar examples

Nabil

Scopes

This language supports nested program blocks and it“s statically scoped.

Global variables must be defined outside program blocks using the reserved word GLOBAt. These global variables can be seen everywhere in the program, and are hidden only on program blocks that redeclare them.

Local variables must be declared at the begining of a program block, and will be visible inside that block, hidden only inside nested blocks that redeclares that identifier.

Types

Pokelang includes the primitives data types.

Type	Label	Size (bytes)
Int	pINTachu	4
bool	BOOLbasaur	4
float	butterFloat	4
char	CHARmander	4
void	VOIDtorb	NO
char	CHARmander	4
enum	ENUManyte	4
pointer	*	4

- Integers are stored in 2“s complement.
- Bools are represented with the words 0x1 and 0x0.
- Floating porint numbers are 32 bits precision with the IEEE 754 standar.
- Void type variables cannot be stored.

This lenguage has uses value model like c, pointers can be a reference to any primitive type they are 32 bits long and can refer to any data type.

Arrays variables are pointers that might refer to any of the primitive or composite type. Pointers are 32bits long.

Pokelang requires explicit casting, no type conversion will be done implicitly.

Dynamic structures as dynamic arrays and structs can be allocated in heap with the built-in procedure *atrapar* and can be freed with *liberar*

Variables

- Variables can be declared of non-VOIDtorb base type, array type, or named type.
- Variables declared outside any function have global scope likewise those declarde using GLOBAt.
- Variables declared within a subrutine declaration have subrutine scope.
- Variables declared in the formal parameter list or function body have local scope. A variable is visible from scope entry to exit.
- Variables identifiers must only start by `poke` if it refers to an articUNION, ENUManyte or STRUCTabuzz (or a pointer to these).

Arrays

Arrays are a reference that points to the first element of it.

Arrays can be be stored in the static area if it“s declared as a global var and has a fixed size given explicitly, otherwise the content will be stored in the heap and it”s memory must be allocated by the user.

This data type is indexed starting at 0, static array declarations must include the number of elements that can hold.

Pokelang array“s are homogenous,

Structs

Structs is implemented as a reference to the first field of it. Every struct field has a name and type.

Structs can only be allocated in the heap.

Struct definitions must start with the prefix poke.

i.e

```
articunion pokeTrainer {
    name                 :: CHARmander[],
    socialSecurityNumber :: pINTachu
}
pokeTrainer ash;
```

Unions

Like the already existent in C, a union in pokelang is a special data type that allows to store different data types in the same memory location. You can define a union with many members, but only one member can contain a value at any given time. Unions provide an efficient way of using the same memory location for multiple-purpose.

The keyword `articUNION` is reserver to declare an Union. Also, the identifier of the data type that will referer to the declaration, must staring with the sub-string `poke`.

For example:

```
articUNION poketrainer {
    name                 :: CHARmander[],
    socialSecurityNumber :: pINTachu
}
```

As may be seen, to declare an articUNION, must first appear the keyword `articUNION` followed by an identifier starting with `poke` and then the elements definition, specifying `name :: type` for each member, separated by commas (except for the last one) inside of braces. All identifiers on the definition of an `articUNION`, must be different.

Enums

Pokelang provides enums and the built-in functions *evolucion* and *preevolucion* to get the successor and predecessor of an element from an enum element.

The elements that are part from an enum enum are unique (cannot be in another enum, or twice in the same declaration), and must start with a capital letter.

Every element from an enum has an integer asociated starting form 0, this number can be obtained using the function *obtener_numero*

Functions and Procedures

Functions

A function declaration includes the name of the function and its associated typesignature, which includes the return type as well as number and types of formal parameters. In pokelang, functions are pure, meaning that any change inside of the function will only affect that scope.

- Formal parameters can be of any base type, array type, pointers or named types.
- Parameters identifiers are unique within a function definition.
- Function may have zero or more formal parameters.
- If a function has a `VOIDtorb` return type, it may only use the empty `vamos_a_retorna` statement.
- If a function has a non-VOIDtorb return type, any `vamos_a_retorna` statement must return a value compatible with that
- The function return type can be any base, array, or named type. `VOIDtorb` type is used to indicate the function returns no value.
- Functions do not nest.
- Functions identifiers have global scope.
- Mutual recursion can be done with forward declations.
- Function overloading is not allowed.

Function Declaration The reserved keyword `funcball` indicates that we"re dealing with a function. We must indicate its return type, followed by `funcball`, the name of the function and then the formal agurments with their respective types and the keychar `:`. Declaration finishes with the reserved keyword `vamo_a_calmano`.

For example: ~ butterFloat funcball hola2(butterFloat[] w,CHARmander z,BOOLbasaur b): #This is a valid declaration si b==squirTRUE: vamo_a_escribi(z); si_no pInktachu h = 3; w[4]=(butterFloat) h // 2; vamo_a_calmano;

```
vamo_a_devolve w !! 4;
```

vamo_a_calmano; ~

Function Invocation Function invocation involves passing argument values from the caller to the callee, executing the body of the callee, and returning to the caller, possibly with a result. When a function is invoked, the actual arguments are evaluated and bound to the formal parameters. All Pokelang functions parameters are passed by value (any change made on the argument doens"t affect the original variable where it comes from).

Invocations use strict evaluation: all arguments are evaluated before sending them to the callee.

Procedures

In pokelang, a procedures is used to declare a non-pure subrutine where all its argumentes are passed by reference and will always return VOIDtorb. Because of this, it shall not have any type on its declaration or any non-empty

`vamos_a_retorna`. Likewise function, the keyword `procball` indicates that a procedure it“s been declared.

i.e. ~ procball hola(butterFloat[] w,CHARmander z,BOOLbasaur b): #All is passed by reference. si *b==squirTRUE: vamo_a_escribi(*z); si_no pInktachu h = 3; *w[4]=(butterFloat) h // 2; vamo_a_calmano; vamo_a_calmano; ~

Types Equivalence

Pokelang uses name equivalence (two types are not equal if their types don“t have the same name).

Two variables of the same type are compared by value.

When two composite types (structures, arrays or unions) are compared, the result will be the comparison of their pointers.

Assignment

For the base types, pokelang uses value-copy semantics; the assignment LValue = Expr copies the value resulting from the evaluation of Expr into the location indicated by LValue. For arrays, pokelang uses reference-copy semantics: the assignment LValue = Expr causes LValue to contain a reference to the object resulting from the evaluation of Expr (i.e., the assignment copies a pointer to an object rather than the object). Said another way, assignment for arrays, objects, and strings makes a shallow, not deep, copy.

- An LValue must be an assignable variable location.
- The right side type of an assignment statement must be compatible with the left side type.
- nullikarp can only be assigned to a variable of named type.
- Tt is legal to assign to the formal parameters within a function, such assignments affect only the function scope.

An especial case of assignments are the cases that uses the operatos `+=`, `*=` and `-=`. In order to work, these requires theirs LVALUES and RVALUES to be of the same type, and these may only be pINTachu or butterFLOAT. These operators will first apply the operations that they refer to and then they assing its new value. `A+=1` is equivalent to `A=A+1`. These doesn“t work using pointers.

Control Structures

Conditionals

The conditional control structure keywords *if elseif* and *else* are represented by the keywords *si y_si* and *si_no*.

The construct "si (expression):" introduces a new non-empty code block that ends with the "y_si", "si_no" or "vamo_a_calmano" keywords.

The construct "y_si (expression):" works like "si (expression):".

The construct "si_no" works like "else" introduces a non-empty code block that will end with the reserved word *vamo_a_calmano*

Pokelang supports case statements and provives some low level optimizations over the if,elseif, else structures.

Bounded iterations

Bounded iterations work over integers ranges and enumerated types.

The structure of a bounded iteration must include explicitly the begining,values as constants (Integers or enum types) or as variables that cannot be changed during the execution of the block.

i.e

```
vamo_a_itera x = 1 |10|1: /* var = first | last | step */
    instruccion0;
vamo_a_calmano
```

or

```
vamo_a_itera dias desde Lunes hasta Viernes:
    instruction0;
vamo_a_calmano
```

Instructions between the tokens *:* and *vamo_a_calmano* introduce a new non-empty code block

Unbounded iterations

Unbounded iterations require the keyword *vamo_mientra* followed by an expression that must evaluate to a boolean (BOOLbasaur).

Instructions between the tokens *:* and *vamo_a_calmano* introduce a new non-empty code block.

i.e

```
vamo_mientra  i < 10 :
    vamo_a_escribi(i);
    i += 1;
vamo_a_calmano
```

Program Stucture

Every pokelang program must have a *VOIDtorb* function with the identifier *hitMAINlee*, even though it isn"t a reserved word it is needed to define where does the program begins. This function won"t have arguments.

i.e

```
VOIDtorb funcball hitMAINlee():
    pInktachu p = 21 * 2;
vamo_a_calmano
```

Expressions

For simplicity, Pokelang does not allow co-mingling and explicit conversion of types within expressions (i.e. adding an integer to a double, using an integer as a boolean, etc.). In order to do such operation, convertion must be made explicty by casting the desired type (i.e. `(pINTachu) myFloat`).

Operators have the following procedence, from highest to lowest:

Operator	Description	Associativity
`*`	Pointer access	None
`(args)`	Function/Procedure call	Left to right
`[int]`	Array access to position	Left to right
`(expr)`	Change of precedence	Left to right
`.`	STRUCTabuzz and ArticUNION access	Left to right
`!!`	Array substraction	Left to right
`-`, `!`	Unary arithmetic and logical	Right to left
`^`	Power Operator	Left to right
`*`, `/`, `//`, `%`	Multiplicative, division, div and mod	Left to right
`-`, `+`	Additive	Left to right
`<`,`<=`,`>`,`>=`	Relational	None
`==`, `!=`	Equality	Left to right

Operator	Description	Associativity
`&&`	Logical AND	Left to right
`||`	Logical OR	Left to right
`and`	Short-circuit AND	Left to right
`or`,	Short-circuit OR	Left to right
`=`, `*=`, `+=`, `-=`,	Assignment and aritmetic ops before assignment	Right to left
`;`	Secuentiation	Right to left

The operators `<`, `<=`, `>`, and `>=` work on pINTachu, butterFLOAT and CHARmander, requiring two values of the same type and returning a BOOLbasaur: `A > B`, `A <= B`,`A > B`,`A >= B`

The operator `!!` recieves an array on its left and a integer on its right and returns the element on that position. `A !! 4` is equivalent to `A[4]`.

The operators `and` and `or` uses shor-circuit evaluation. That means that they will only evaluate the expresion if their first argument are `squirtrue` and `squirfalse`, respectively: `A and B`, `A or B`

The aritmetic operators `+`, `*`, `-`, `^` works using two pINTachu or butterFLOAT and returning the same type: `A + B`,`A - B`,`A * B`,`A ^ B`

The aritmetic operator `/` will only work using two butterFLOAT and returning a third: `A / B`

The aritmetic operator `//` will only work using one butterFLOAT or pINTachu, one pINTachu and returning a pINTachu: `A // B`

The aritmetic operator `%` will only work using one two pINTachu and returning a third: `A % B`

The operators `!`, `&&`, `||`, `and`, `or` will only work between BOOLbasaur and return BOOLbasaur.

The operators `==` and `!=` work on all types, taking two values of the same type and returning always a BOOLbasaur. `A == B`, `A != B`

The access to an STRUCTabuzz element is done by using the operator `.` and it return that element type: `A.e` and it“s type is the type of e.

The access to an STRUCTabuzz element is done by using the operator `.` and it return that element type: `A.e` and it"s type is the type of e.

In order to access the element to wich a pointer is pointing to, it must be used the operator *A, where A is of type* type with type as any of the basic types.

To secuence instructions it must be used the operator `;`: Instruction1 ; Instruction2

The operator `=` requires its LVALUE and its RVALUE to be of the same type.

The operators `+=`, `*=`, `-=` requires its LVALUE and its RVALUE to be of the same type, and these may only be pINTachu or butterFLOAT. These operators will first apply the operations that they refer to and the they assing its new value. `A+=1` is equivalent to `A=A+1`

Final notes

Javier & Nabil