This is *ginseng 1.1.0*, our very first compiler for the *Ginseng* programming language, developed by us.

**Ginseng programming language**

*Ginseng* is a small, compact and strongly typed programming language. It has the following characteristics:

* **Set-oriented paradigm**: *Ginseng* provides a native *set* type, that reflects the properties of its mathematical counterpart (more on this in the type system section).
* **Embedded operators**: provides the developers with a collection of built-in operators to efficiently manipulate data.
* **Simple and intuitive notation**: *Ginseng* operations follow the most common notational conventions, making it an intuitive and easy to use programming language for those who have even just basic knowledge of math.

**Ginseng type system**

Ginseng is a strongly typed language: *type inference* is used during compilation to infer variable types and enforcing operations on such. The reason behind this is to keep the syntax as light as possible, making the task of writing a ginseng source file as fast and less verbose as possible, while maintaining a good level of expressiveness in the meantime. Variables are all global-scoped.

Ginseng offers the following 3 native data types:

* **number**: this is the basic data type. It is used to represent numbers of any kind, i.e. integers and floating-points.
* **array (sequences)**: this data type is the equivalent to the array data structure native to most programming languages. It fits good in abstracting mathematical sequences, as they have the same ordering and the same indexing. It is possible to declare an array variable, populate it and retrieve elements from it, either via expression or via indexing. Arrays are have unbounded length and can grow dynamically as new elements are added, or as an element is inserted in a non-existent index.
* **set**: data type which abstracts the mathematical concept of a set. Basically, it represents an unordered collection of unique numbers. Sets support the common set operations, such as membership tests, intersection with another set, union and difference. It is also possible to add single number variables to a set, and even to use them as operands in one of the previously mentioned set operations! A number used as operand along with a set is interpreted as a *singleton*, which is a special type of set containing just a single element.  
  Following the most widespread mathematical convention, sets are declared with identifiers starting with an uppercase letter.

**Syntax and keywords**

The following are language reserved keywords. They cannot be used to define variables, and, in case of operators, they cannot be part of a variable’s name.

|  |  |
| --- | --- |
| **KEYWORD** | **DESCRIPTION** |
| print | Prints the value of a variable, string, expression or a concatenation of any of the previous. Called as print varname; |
| ginseng | If supplied as argument to print, triggers a funny easter egg |
| length | Retrieve the length of a set or array |
|  |  |
| if | Keyword for conditional construct |
| then | Keyword for conditional construct |
| else | Keyword for conditional construct |
| and | Logical and for conditions |
| or | Logical or for conditions |
| == | Boolean comparison operator for equality |
| != | Boolean comparison operator for difference |
| < > <= >= | Logical comparison operators |
|  |  |
| union | Operator for set union |
| intersection | Operator for set intersection |
| difference | Operator for set difference |
| is\_subset | Operator for subset check between sets |
| is\_equal | Operator for equality check between sets |
| contains | Operator for membership of an element against a set |
|  |  |
| abs | Operator for absolute value |
| % | Operator for modulo operation |
| ! | Operator for factorial |
| ^ | Operator for exponentiation |
| \_ | Operator for concatenation |
| + - \* / | Standard arithmetic operators |

Note: all binary operators are defined as **infix** operators, allowing for a more natural definition. This means, intersection of two set A and B is called by doing C = A intersection B; similarly to the other operators listed above.

**Syntactic elements**

A ginseng program is made of source code saved in .g format.  
The line separator is the character “;” and the program terminator is the dot “.” : a valid ginseng program must have its last instruction terminated by it.  
Comments are marked by “//” and span an entire line. They are ignored by the compiler.

**Variables declaration and operations**

A valid **numeric variable** identifier must start with a lowercase letter and is made of alphanumeric characters only.

a = 1;  
b = 3.1415;  
c = -12;

A valid **array** identifier is made up of any combination of alphanumeric characters enclosed between the so-called diamond: “< >”. Elements are enclosed in square brackets and separated by a comma character “,” .  
Array initialization is bounded to the declaration: when the variable is first declared, one must supply at least one member to the array. There are no empty arrays. However, it is possible to initialize the array to contain as many members as desired.

<arr1> = [1]; //valid  
<arr2> = [1, 2.12, -3, 4, -3.14, 1, 3]; //valid  
<arr3> = []; //not valid

Elements array can be referenced with the classic zero-indexed bracket indexing and assigned to a numeric variable if needed. Note that this does not remove the element from the array, but it returns a copy of it. If the index is out of bounds, the compiler will raise an error.

<arr> = [1, -12.1, 3]  
num = <arr>[0] //num is now 1  
num2 = <arr>[5] //syntax error

Same notation can also be used to insert an element into the array at a given position, overwriting the element already present. If the index is out of bounds, array is automatically expanded to fit the index specified for the new element. Any element corresponding to indexes between the last element’s index and the newly inserted element is initialized to 0 automatically.

<arr> = [1, 2, 3];  
<arr>[5] = 10 //<arr> is now [1,2,3,0,0,10]

A valid **set** identifier starts with an uppercase letter (as the usual mathematical notation requires) and can contain any combination of alphanumeric characters. Similarly to arrays, they must be declared and initialize to contain at least one element: set elements are listed within curly brackets and are separated by a comma.

S = {12, 0.6, 0.7, 1, 0.001}; //valid  
S = {}; //syntax error when used as initializer

Differently from arrays, elements can be removed from a set and the result can be the empty set. Note that any operation involving a set will return a result of type set, which should be reassigned to a set variable in order to be stored or reused.

S = {1};  
S difference 1; //syntax error  
S = S difference 1 //valid: 1 is converted to {1} and now S={}

Set operations can always have as argument either two sets, or a set and a number (except is\_subset). To add elements to a set, the union operations is used. Set does not allow duplicates by definition: any number which is added to a set, and is already present, is ignored.  
Set membership operations, namely contains and is\_subset, return a number variable that can be interpreted as a boolean (1.0 true, 0.0 false).