Identifiers

A valid identifier is a sequence containing at least a letter and possibly numbers and underscores.
Its is recommended :

- to make an identifier start by a lowercase letter
- that an identifier does not contain two successive underscores

An identifier cannot contain spaces, special characters or keywords.

[a-zA-Z_][a-zA-Z0-9_]*

Fundamental types

The available fundamental types are the following :
Integer : whole numbers that can be positive, negative or zero
Real : positive or negative decimal numbers
String : all existing characters
Boolean : true or false values

Variables

Syntax : `variable name is type`

EZ Language	C++	Result

program p
procedure p()
begin
//Déclaration des variables locales
variable a,b,c are integer
// Initialisation
a = 10
b = 20
c = a + b
print "C = "+ c
end procedure
	#include <iostream>
	int main ()
	{
	//Déclaration des variables locales
	int a,b,c;
	// Initialisation
	a = 10;
	b = 20;
	c = a + b;
	std::cout<<"C = "<<c;
	}
		C = 30

EZ Language	C++	Result

program p
// Declaration of a global variable
global g is integer = 0
procedure p()
begin
variable a,b are integer
// Initialisation
a = 10
b = 20
g = a + b
print "G = "+g
end procedure
	#include <iostream>
	// Declaration of a global variable
	int g = 0;
	int main ()
	{
	int a,b;
	// Initialisation
	a = 10;
	b = 20;
	g = a + b;
	std::cout<<"G = "<<g;
	}
		G = 30

Constants

A constant is an expression with a fixed value. The EZ Language compiler prevent developers from modifying a constant value.

Syntax : **`constant`** `name` **`is`** `type = value`

EZ Language	C++	Result

program p
procedure p()
begin
// constant declaration:
constant c is integer = 5
c = 10
end procedure
	int main()
	{
	// constant declaration:
	const int c = 5;
	c = 10;
	}
		error: assignment of
		read-only variable ‘c’

Regular Expressions

A regular expression (also called rational expression) is a sequence of characters that define a search pattern that a matching text must conform to.
A target sequence is the string that must match the regular expression.
A pattern is the sequence of characters representing what we search.
A match is a substring of the target sequence that match the pattern.

Regex operations

regex.match(target sequence, pattern) : return true if the sequence match the pattern, false otherwise.
regex.search(target sequence, pattern, match) return true if a subsequence in the target sequence match the pattern, false otherwise. If found, the subsequence is stored in match.
regex.replace(target sequence, pattern, replacement)

EZ Language	C++	Result

program p
procedure p()
begin
variable s is string =
"subject"
variable e is regex =
"(sub)(.*)"
if e.match(s) then
print " String object matched
\n";
endif
end procedure
	#include <iostream>
	#include <string>
	#include <regex>
	int main ()
	{
	std::string s ("subject") ;
	std::regex e ("(sub)(.*)");
	if(std::regex_match (s,e))
	std::cout << " String object
	matched \n";
	return 0;
	}
		String object matched

program p
procedure p()
begin
variable s is string = " this
subject has a submarine as
a subsequence"
variable e is regex =
"\\b(sub)([^]*"
variable m is smatch
while e.search (s,m) do
foreach()
print " \n";
end while
end procedure
	#include <iostream>
	#include <string>
	#include <regex>
	int main ()
	{
	std::string s ("this subject
	has a submarine as a
	subsequence") ;
	std::regex e ("\\b(sub)([^]*")
	); // matches words
	beginning by "sub"
	std::smatch m;
	while (std::regex_search
	(s,m,e)) {
	for (auto x:m) std::cout << x
	<< " ";
	}
	return 0;
	}
		subject submarine
		subsequence

program p procedure p() begin variable s is string = "there is a subsequence in the	#include <iostream> #include <string> #include <regex> int main () {	There is a sub-sequence in the string

string"
variable e is regex =
"\\b(sub)([^]*"
print e.replace(s, " sub-$2")
end procedure
	std::string s ("there is a subsequence in the string")
	;
	std::regex e ("\\b(sub)([^]*)"
	); // matches words beginning by "sub"
	std::cout <<
	std::regex_replace(s,e,"
	sub-$2");
	}	

Numeric Literals

In order to make the reading of numeric literals easier, underscores can be used.
Exemple: variable number is integer = 1_000_000

In EZ Language, hexadecimals (base 16) begin with 0x
Exemple : 4b#16 = 75 (decimal)

Octals (base 8) begin with 0
Exemple : 113#8 = 75 (décimal)

Binary numbers (base 2) begin with 0b
Exemple : 01001011#2 = 75 (décimal)

EZ Language	C++	Result

program p
procedure p()
begin
variable x is integer =
01001011#2;
variable byte1 is integer =
113#8;
variable byte2 is integer =
4b#16;
variable total is integer =
256 * byte1 + byte2;
print "x = ",x, "\n"
print "byte1 = ",byte1, "\n"
print "byte2 = ",byte2, "\n"
print "total = ",total, "\n"
end procedure
	#include <iostream>
	int main(){
	int x = 0b01001011 ;
	int byte1 = 0113 ;
	int byte2 = 0x4b;
	int total = 256 * byte1 +
	byte2;
	std::cout<<"x = "<<x <<
	std::endl;
	std::cout<<"byte1 = "<<
	byte1 << std::endl;
	std::cout<< "byte2 = "<<
	byte2 << std::endl;
	std::cout<< "total = "<< total
	<< std::endl;
	return 0;
	}
		X = 75
		Byte1 = 75
		Byte2 = 75
		Total = 19275