OurC Project 234 Overview

This file gives a brief overview of the OurC project. This project consists of 3 parts, namely Project2, Project3, and Project4. You will have to use PAL to write these three projects. However, you can always use other software development system to write your code and then paste it into PAL (but you will not be able to copy PAL content to the outside world).

Basically, you are to write an OurC interpreter. OurC is a simplified but slightly extended C language. It has five primitive data types (int, float, char, bool, and string), one-dimensional array, and no struct or enum. There are function calls, compound statements, conditional constructs (if-then-else) and loop constructs (while-loop and do-while loop). A separate file contains the spec. (i.e., a grammar written in EBNF) of OurC. Please consult that file.

Since OurC is an interpreted language and not a compiled language, there are several aspects of OurC that make it different from "the usual C."

Here is how a simple session with OurC looks like :

```
// Suppose we ran an OurC executable on the command level ...

Our-C running ...              // greeting message

> int x;                       // user input
Definition of x entered ...      // system response

> x=10;
Statement executed ...

> cout << x;
10Statement executed ...            // note that there is no line-enter when printing

> cout <<
x
<< "\n"
;              // the system gets input until end of definition or statement
10
Statement executed ...

> void AddThree(int)
Line 1 : unexpected token : ')'  // the system stops parsing after reading in just one error ;
                                 // when an error occurs, the current user input is discarded ;
                                 // the system resumes parsing (of a brand new user input) from the start of the next line

> {y = y + 3; }
Line 1 : undefined identifier : 'y'

> void AddThree(int& y)
{y = y + 3;} // AddThree()            // comments should be skipped during parsing
Definition of AddThree entered ...
```

```
  > AddThree(x);
  Statement executed ...

  > cout << x;
  13Statement executed ...

  > ListAllVariables();
  x                              // one variable name per line, the variable names should be sorted
  Statement executed ...

  > ListAllFunctions();
  AddThree()                     // one function name per line, the function names should be sorted
  Statement executed ...

  > ListVariable("x");
  int x ;                        // all comments are "stripped off"
  Statement executed ...

  > ListFunction("AddThree");
  void AddThree( int & y )         // all comments are "stripped off" ; this is "pretty print"
  {
    y = y + 3 ;
  }
  Statement executed ...

  > Done();
  Our-C exited ...
```

The OurC interpreter maintains an environment. This environment is composed of zero or more "definitions". A definition can be either the definition of a variable or the definition of a function. The user enters the definitions directly.

Apart from definitions, the user can also enter one C-statement at a time on the prompt level. There are several "extra" system-supported functions, as listed below.

```
  ListAllVariables();         // just the names of the (global) variables,  sorted (from smallest to greatest)
  ListAllFunctions();         // just the names of the (user-defined)  functions, sorted
  ListVariable(char name[]);   // the definition of a particular variable
  ListFunction(char name[]);   // the definition of a particular function
  Done();                     // exit the interpreter
  cout << ...                 // output from program
  cin >> ...                  // input into program
```

A global variable may have a value, and the system can simply store this value "together with" the definition of the variable. With a skillful arrangement, there can be no need for a separate call stack. However, you will need a data structure that is capable of storing arbitrary types of data.

OurC Project 234 Overview

The project is composed of three parts:

Project 2 : implement OurC as a syntax checker and pretty printer, with the above-mentioned system supported functions (except cin and cout)
Project 3 : implement OurC (the interpreter) without function calls, but with conditionals, loops and arrays.
Project 4 : add function calls to OurC.

```
// ===========================================================================

// Please ignore the following (these are features that may be enforced in the future) :

  // > Dump("file1");  // Ask the system to 'dump' all definitions to a file
  // Statement executed ...
  //
  // > Load("file1");       // Ask the system to load from a file
  // Definition of x entered ...
  // Definition of AddThree entered ...     // Many output messages in practice

  // Dump(char name[]);          // print all definitions to a file
  // Load(char name[]);          // load from a file
```