Nifty50

I would have written a shorter program but I did not have time. – A programmer

Aamir Sarwar Jahan (aj2599)
COMS W4115 – Fall 2014, CVN

Motivation:

Anyone who has ever read a computer program written by someone else has one complaint in common – the code is too long. With increasing code space available to programmers on target devices, the emphasis is no longer on writing succinct lines of code that does more. This has caused the source codes to become less modular, longer, and harder to debug.

Nifty50 is a high level programming language that allocates exactly fifty lines of instruction for each module in a program. The language will be object oriented where each module is considered as an object and every source code will have a single, unique, top module (think of it as the main function in C). The objects, or modules, will take arguments and return a single value or no value. The scope of the variables can be set to global or local to the module. But does this mean Nifty50 is just another version of a regular programming language that forces the program to be in sets of 50 or less lines?

Not really. The goal of this language is to increase readability, lessen debugging time and the time needed to convert pseudo-code to actual working source code. It is a language where engineering managers and directors (years away from their programming days) will look at the top module and understand just how the program works. The language will optimize algorithms like sorting, searching, array manipulation into easy to read (and implement) function-like structures.

Common Algorithms in Nifty50:

a. Loops: these are one of the most frequently used constructs in a source code. A for loop in C has the following syntax:

```
for ( variable initialization; condition; variable update ) {
  Code to execute while the condition is true
}
```

In the proposed language, the conditions of the for loop are more effectively expressed as:

```
Loop (Condition, {Code to execute if Condition is true});
```

The variable is also initialized in the condition section and the variable update is done in the statement to be executed rather than as part of the loop syntax. Also, the statement can execute a module.

b. Array Manipulation: the language can be used to perform common array manipulations like insert, reverse, or search very easily. An example of a reverse algorithm in C for an array named 'a' is:

```
for (c = n - 1, d = 0; c >= 0; c--, d++)
        b[d] = a[c];
for (c = 0; c < n; c++)
        a[c] = b[c];
```

In the proposed language, the list/array reverse function will be simply:

```
b = a.reverse;        //reverse all elements of a to new array b.
a = b;
```

c. Search: the algorithm for searching through an array or any data structure is also quite simple in the new language. An example of a search algorithm in an array containing characters is:

```
i = 0;
while (i < 'c'&& ch != a[i]) {
        i++;
  }
```

But for the new language, the search function will return a Boolean which will be 1 if search returns a result or a 0 if otherwise. The

```
bool search = a('c');//search is initialized to search for 'c' in a
search.query = 'c'; //search parameter has value 'c', the query
search.index = 4;   //the query is found in index 4.
```

Modules in Nifty50:

Modules are the building blocks of any program written in the language. A programmer can decide to divide a big program into modules that each performs a single function. Although each program can have multiple modules, every module should be declared in the top module and return a value to the top module. The program end statement must be in the top module. A module declaration in the language will be in the beginning of the program and can be grouped as per the return value.

```
//two modules that takes two integers as arguments and
returns a float
float module calcDivision(int, int), calcMean(int, float);
```

And calling a module would be as easy as:

```
Div_result = calcDivision(102, 43); //Div_result is a float
Mean_result = calcMean (35, 32.5); //Mean_result is a float
```

A module can also be invoked using the dot (.) operator similar to the array.reverse operation in previous examples. This will mean that a module can be written as an algorithm and can be invoked as part of different data structures.

Sample Program in Nifty50:

The programming language is designed to be used in particular for embedded systems application. For the sample program below, a 8 bit microcontroller is used for an occupancy sensor which detects movement in a certain peripheral area by using a PIR sensor and turns on a load (light bulb) if movement is detected. Also, the microcontroller has an ambient light sensor which it continually monitors and if there is enough ambient light, the load would not turn on for a movement. The sensor has some user inputs such as light sensitivity and light brightness that they can set externally using physical settings such as a knob.

```
//Sample program for an occupancy sensor
//Line
1.    start top:
2.    int module calcAmbient (int ambience), calcBright (int level);
3.    bool module calcMovement (int PortC, int sensitivity); //takes
      two arguments, the values read in from the PIR sensor is at
      Port C of microcontroller and sensitivity is set by the module
      calcAmbient)
4.    sensitivity = calcAmbient ( PortB); //PortB reads user set
      ambient light sensitivity
5.    turnOn: PortD.1= 1          //PortD pin 1 controls the load
6.    turnoff: PortD.1 = 0
7.    level = calcBright (PortA); //PortA reads user set brightness
8.    if (calcMovement, turnOn, turnoff); //the if statement
      evaluates the calcMovement which is the condition and turns
      on the load if it is true or turns off if the condition (movement)
      is false.
```

Conclusion:

It seems that the sample program is short enough to be taken as a pseudo-code implementation. The primary goal of the new language is to reduce the time a programmer takes to write the top module that will form the skeleton of the main program and then work on the little details in the individual modules. It is imperative that the new language is easy to read and debug. The compiler for it will evaluate the syntax and semantics of each module that will aid in debugging the software as each module can be debugged independently. The dot (.) operator can be used to invoke other modules and used to perform frequently used algorithms. Overall, Nifty50 will be a structural programming language with user-defined modules of 50 lines of instruction or less.