Principle Of Programming Language (P.O.P.L.)

FINAL REPORT

Project:- Designing a programming language.

TEAM AKATSUKI

Project Manager - Manav Garg

Language Guru - Koushik puppala

Example Creators - Yugal Garg

Sikander Kathat

Abhyuday Choumal

Umaid Shaan

Special Thanks to “Helping Hands”

Goals :-

- Cover basic concepts underlying programming languages (syntax, semantics, families, ...)
- Study Scheme as an example of a functional programming language.
- Study Python as an example of a modern scripting language.
- Study an interpreter for Scheme in both Scheme and Python.
- Touch on other languages and concepts along the way.

Introduction of Programming language (Bego)

We are designing a programming language which is inspired by C and its descendants, but a lot simpler to read and write. As we know the compilable language is easier to debug, So we are going to design a compilable language **BEGO**. The programming paradigm of this language is imperative as the program written in this language simply gives the solution to the specific problem. This language is going to specify how to control flow of computation.

After a lot of research and analysis we all got to know that the bracket system which we have to write in the C language is not that much needed. Also they lead to minor syntax errors, alternatives are like if we see in other languages like python, there they are having indentation. So after every function we don't have to put separate brackets for it, we just have to write all the inside things of that functions in a properly indented way, e.g, if we are

writing a for loop all the conditions or statement that we are gonna write in that loop should be in a proper indented way and hence the compiler will automatically understand those things that match that indentation are in that loop. They are easier and fully functional for loops which improve the looping concept of the language. We can use grammatical representation to represent logical operators instead of using symbolic representation and improve the visibility of code.

This language is far simpler than the C language because it provides a lot of new features and the syntax is far more simpler. Writing and readability of our language is simpler than C language by implementing the indentation scheme. There is no header file concept in our language, we just use a single import library to use the required functions in our program.

We implemented that if there is no pointer pointing to a memory allocated in heap, it will automatically be deleted to avoid memory leaks. We don't have to declare the type of variable; it automatically assigns the type of data during the time of execution.

Language Reference

This language manual is written and described by the **team members of Akatsuki**.

- **Indentation :**

We are following a proper indentation rule. Like if we are writing any function so the compiler will only take those statements inside it which are properly indented and once the indentation is broken the compiler will see it as the end of that function. Like if we are following a indentation rule of 4 spaces so in that whole function everything should be at least 4 space indented or the compiler will see it as another function or a function inside a function.

- **Variable declarations :**

No need to specifically declare a variable. We just have to write the variable name and it's value; the compiler will automatically see it's type and declare it itself

only. In C we had to specifically declare a variable and it's type. Here the language is having fluidity so it's easy here.

- **Input and output function :**

  Again we can see here too that in C it was very difficult for a new programmer to remember all the symbols perfectly and execute it. Here the input function is "in" and the output function is "out" and whatever we are gonna input or output should be in brackets with quotes.

- **English Keywords :**

  We are using english keywords like "And" , "or" , "not" like keywords instead of using symbols "&&" , "||" , "!=" respectively as these symbols are hard to use, difficult to remember and also takes more time for any programmer to write. So we brought the concept of writing what we are saying in our brain. So every time we are writing any multi conditional statement we are just typing the words and not some freaky symbols.

- **“For” loop :**

  As we were seeing in the C language the syntax of the for loop is a bit tricky with a lot of semicolons and brackets. In this language we took the syntax to the very peak of simplicity. In the C language the syntax of for loop will be like

  For (int i = 0; i<n ; i++), as you can see here that syntax is a bit tricky but in our language the syntax is like,

  For i in (n) , here you can see that the syntax is completely simple also for specially declaring the start of the variable and also the increment we can just write it like this,

  For i in (1,n,2) here i will start from 1 and will be incremented by 2 every time.

- **Array :**

  We have changed the syntax of the array too. Now we don’t have to declare the type of value it’s going to store. We will just have to declare the array and the type of data will be seen by the compiler automatically. The syntax now is “array dec[5]”

- **Comments :**

  We are going to use the conventional method to write the comments. For single line comments we are going to use “//” and for multi lines it will be “$.....This is our new language….$”

- **Keywords :**

  The keywords that we used till now are if, else, for, return, for, var, import, while etc.. in our examples.

--X--

Language evolution

We were programming our programs in C language and we came across multiple problems like the complexity of syntax. Everytime when we are writing any code we have to deal with a lot of stress to make the syntax proper. Sometimes it is even happening that our code is giving a critical error just because of one small syntax error of semi colon or some brackets. So, we took these things out completely . Now our language is not having any semicolon and lots of brackets because it is indentation based as lots of modern languages are right now e.g. Python. Now sometimes there are some difficult syntax that we have to memorize and pay a lot of effort like in for loops and remembering the syntax in multi conditional statements. So we tried to write the same as we are speaking, like replacing && with and directly. Also in the for loops we simplified the syntax a lot and now it's just a child's play to write it. Now we just have to write it like "for i in (1,20,1)" which is less time consuming.

Now there is no need to declare the type of any variable, which is again a confusion, now we just have to declare the variable and we are good to go.

Defined Examples:-

Example 1 (Age eligibility program)

```
import lib
var age
out('enter your age\n')
in (age)
if (age >=12 and <= 50)
    out ('young\n')
else
    out("not young\n')
if(age<12 or age>50)
    out('Eligible for the offer\n')
else
    out('Not Eligible for the offer\n')
```

Example 2 (finding element in an array)

```
import lib
array a[10] = { 1, 2, 3, 4, 5, 6, 7, 8, 9, 10}
var key
out ('enter the key element\n')
in (key)
for i in (10)
    if (key == a[i])
        out ('the key element is found at '  +i )
        exit
out ('element is not found')
```

Example 3(sum of 2D array)

```
import lib
array A[2][3] = {3, 3, 3, 3, 3, 3}
array B[2][3] = {1, 1, 1, 1, 1, 1}
array C[2][3]

for i in (2)
    for j in (3)
        C[i][j] = A[i][j] + B[i][j]

for i in (2)
    for j in (3)
        out(C[i][j] + ' ')
    out('\n')
out('\n')
```

Example 4(leap year)

```
import lib
var year
out('Enter a year: ')
in(year)
if(year%4 == 0)
    if(year%100 == 0)
```

```
        if(year%400 == 0)
            out(year + ' is a leap year.')
        else
            out(year + ' is not a leap year.')
    else
        out(year + ' is a leap year.')
else
    out(year + ' is not a leap year.')
```

Example 5(Factorial calculation)

```
import lib
var factorial(n)
    if n== 0
        return 1
    else
        return n * factorial(n-1)

var n, m
n = 10
m = factorial(n)
out('the factorial of n: ' +m)
```

Example 6(Find greater number)

```
import lib
var a , b

out ('enter value of a and b')
in (a , b )
if (a=b)
    out ( 'numbers are equal' )
else if (a>b)
    out (' a is greater than b')
else
    out (' b is greater than a')
```

Example 7 (add two numbers)

```
import lib
var a , b , c      // Type of variable is
                   // taken by  compiler
out ('enter two numbers')
in (a , b )
c = a + b
out (' the sum is : ' + c)
```

Example 8 (Average of given numbers)

```
import lib
```

```
var n, i, sum=0.0, average
array num[100]
out('Enter the  numbers of elements: ')
in(n)

for i in (n)
    out(i+1 + '.Enter number')
    in(num[i])
    sum = sum + num[i]

average = sum/n
out('Average ' + average)
```

Example 9(reverse a number)

```
import lib
var n, num, digit, rev = 0
out('Enter a positive number: ')
in(num)

n = num

do
    digit = num % 10
    rev = (rev * 10) + digit
    num = num / 10
while(num != 0)
```

```
out('The reverse of the number is: ' + rev + '\n')

if(n ==rev)
    out('The number is palindrome')
else
    out('The number is not a palindrome')
```

Example 10 (find denominator)

```
import lib
var a, b, c
out('enter two numbers\n')
in( a , b)
if (b == 0)
    out('invalid denominator\n')
else
    c = a/b
    out  (c)
```

Criticism about the language:

- The language is limited to small threadings i.e only the smallest sequences of programs can be executed.

- Multiprocessing the running of two or more programs simultaneously cannot be implemented to the most of the programs.

- The compatibility of doing large user defined programs with the language makes the running of the program a complex task due to its limitations.

- The program output depends on the design of the program, mainly the indentation. The outcome is unpredictable if the indentation is not done properly.

- The program language does not support dynamic typing unlike the advanced languages.

- It is hard to justify switching to a new version of the language if none of the libraries work. Libraries could only port once their own dependencies had ported,

Conclusions containing lessons learned:

- The knowledge of the programming language was very well used to the extent we learnt.
- Exploring the different languages to extract the best and important, efficient features.
- We got very well revised with the basic programming language C, C++ and advanced language like python as we used the specifications of the mentioned languages to the most
- We also noticed the limitations of few features in the most used languages and our main errors in the usage of those languages to make our programming languages more useable for the users