Odd Semester (2022)

A close up of a sign

Description automatically generated

**Assignment Cover Letter**

**(Individual Work)**

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | | | | |  | |  | |  |
| **Student Information**: **Surname** | | | | | **Given Names**    **Ravel** | | **Student ID Number**  **2301890320** | |  |
| 1. | | **Tanjaya** |  | |  |
|  |  |  |
| **Course Code** | **: COMP6502** |  |  | | **Course Name** | | **: Introduction to Programming** | |  |
| **Class** | **: L1AC** |  |  | | **Name of Lecturer(s)** | | **: Ida Bagus Kerthyayana Manuaba** | |  |
|  |  |  |  | |  | |  | |  |
| **Major** | **: CS** |  |  | |  | |  | |  |
| **Title of Assignment**  (if any) | : HunG programming language | |  |  | |  | |  | |
| **Type of Assignment**    **Submission Pattern** | **: Final Project** |  |  | |  | |  | |  |
| **Due Date** | **: 14-01-2019** |  |  | | **Submission Date** | | **: 14-01-2019** | |  |

# Plagiarism/Cheating

BiNus International seriously regards all forms of plagiarism, cheating and collusion as academic offenses which may result in severe penalties, including loss/drop of marks, course/class discontinuity and other possible penalties executed by the university. Please refer to the related course syllabus for further information.

# Declaration of Originality

By signing this assignment, I understand, accept and consent to BiNus International terms and policy on plagiarism. Herewith I declare that the work contained in this assignment is my own work and has not been submitted for the use of assessment in another course or class, except where this has been notified and accepted in advance.

**“HunG programming language”**

**Name : Ravel Tanjaya**

**ID : 2301890320**

# Project Specification & Solution Design

## Definition

Many beginner programmer finds it hard to start programming, this is mainly caused by the fact that many popular programming language are purpose oriented or in the other point of view they are not beginner friendly, this causes a lot of confusion. That is why I think it is a very good idea to create a “general purpose” programming language that is both easy and applicable to real life application for beginner programmer.

The HunG programming language is an interpreted programing language that is designed to be as simple as possible for 1 purpose which is, to be able to implement a programming language that is both simple and extensible that is easy for beginner to learn implement and extensible for experienced programmer to use in real life project. The current state of the project is in early beta, which means that we can expect rough edges and unimplemented features as the scale of the project is huge, But generally the interpreter itself is quite stable for normal usage.

There are mainly 3 steps in the implementation of this program which is the lexer(tokenizer), the parser and the interpreter, these 3 steps are required in order to convert the whole code in HunG to a working program, they also handle the whole error handling.

## Program Design

The way HunG is implemented is by letting it run on top of python and take certain command from the CLI windows which is then tokenize,parse and interpreted to create the final result, all of this processes are done fully in python without the use of other library, the only library used here is termcolor, which is used in order to colorize the output of the software.

Writing an interpreted programming language in python may not be an option for some people as they may think that writing an interpreter on top of an interpreter may result in bad performance, but this is not enterely true, as python actually doing a JIT compilation to python bytecode before the interpretation process we kind of get a semi compiled version of our interpreter, and of course this will result In better performance(even though we would not expect the result to be even close to java or even C). The other things that is taken into consideration while writing this software is the fact that python especially version 3 is pre included on many UNIX based distribution such as MacOs or Linux, this means that installation of HunG in MacOs and Linux is quite easy and seamless for most of the user, we should also take note that most of professional programmer prefer to use UNIX machine or UNIX emulation layer on Windows machine (such as WSL/Windows subsystem for linux).

The other consideration why I wrote HunG using python is that python is one of the most popular programming language in the world, which also means a lot of library is written for python, this also means if we can find a way to interface python library to HunG, the possibilities of using HunG is theoretically infinite, the implementation may be quite similar to typescript that may use javascript libraries except HunG is directly interpreted on top of python rather than being transcompiled to python.

The first time I’m writing HunG, I am planning to create a compiled programming language using the LLVM compiler architecture, the LLVM are capable to create a set of object code in a way that is far more easy that writing a compiler from scratch, the object code than can be directly translated into machine code with programs such as g++ or gcc, but at the end I didn’t choose to implement this feature as the resulting code may be too complex and too hard to debug.

# Implementation

A picture containing text, map

Description automatically generated

The first step of the process is to get the input to the shell, the shell is the part of the program that make the decision weather certain input is a command or syntax, then if the input is not a command the process will continue to the interpreter.

The implementation of the interpreter is done in 3 main steps:

1. Lexer (Tokenizer)

Lexing or Tokenizing is the first process of the interpreter. In this process the input of the user will be decoded to a single character and then the meaning of that character is determined before being converted to a token, the Lexer class also holds all of the constants that act as the keyword of the tokens. The tokenizer also do check for errors that is available inside the code, for example for Unknown syntax.

Sample input:

var a = 10-5

will result in the following token output stream:

[Token(KEYWORD,var), Token(IDENTIFIER,a), Token(EQUAL,OPERATOR), Token(NUM,10.0), Token(MIN,OPERATOR), Token(NUM,5.0), Token(EOF,EOF)]

1. Parser

Parsing the input is the second process in the HunG interpreter, the process of parsing is used to create a stream of AST(Abstract Syntax Tree) will is understandable by the interpreter, the AST it self is an representation of the Tokens which is provided by the Lexer that had been converted to a form that lets the interpreter understand the order of execution. The parser also take care of error caused by syntax will illegal pattern.

For example if we input the output of the previous sample to the parser, I will convert the Tokens stream to an AST.

In this case the AST is:

(C|Token(IDENTIFIER,a),(B|Token(NUM,10.0), Token(MIN,OPERATOR), Token(NUM,5.0)))

If we take a look at the AST we can see that actually certain tokens are removed entirely from the Tree such as the EOF (End Of File) tokens as of course knowing the end of certain file may not be necessary in the interpreter. Also we can notice that the parser had added certain parentheses in order to notify the interpreter the order of how the whole computational process should occur.

1. Interpreter

Interpreting is the last process that occur in the interpreter, it convert the AST stream that is generated by the Parser to a result that is human understandable, in this steps all variable are being managed (such as being accessed or being initialized) as the symbol table\* are managed from the interpreter. \*(Symbol Table is the part of the interpreter that is responsible to store or retrieve certain data from the variable system).This part of the system also take care of specific set of error call runtime error which means that the whole syntax are understandable by the system but the system is unable to complete the request due to the request causing the result to be impossible to be displayed/stored (such as division by zero).

For example the previous AST will return 5.0

The interpreter will first access the inner command which is to substract 5 from 10 which will return 5.0 then it will assign the value to the variable a (annotated by C in the AST which means variable create node), the variable create node in HunG will return the same value that is assigned to the variable, so the command will return 5.0 (HunG programming language uses a single data type “num” in order to represent all integer and float based number, so that 5 is completely equal to 5.0).

## Running and installing HunG

The installation of HunG is quite simple, you can clone the software with

Git clone <https://github.com/raveltan/hung-lang.git>

And if you have python3 installed on your machine you can diretly run the software by running hung.bat on a windows based machine or hung.sh on UNIX based machines.

One of the objectives of HunG is of course to be able to be operated and installed easily regardless of the operating system. This is achieved with the help of the following features:

1. Launch Script

Most of the python programs are created to be launched from the command line, or by clicking on the file itself. From certain people this may not be the best way of running the application as this kind of launch system may not have error handling and may be quite confusing for people who are not usually running python.

In order to make HunG easier to run and to add the error handling capabilities in all Windows, Linux, and MacOs we have created a set of script that allows the user to run the app easily this also adds error handling capabilities(for example in windows machines where user have not yet installed python3), the script are consisted of:

* 1. Hung.sh

The launch script for UNIX based system, users are able to run the program from cli or directly by clicking on it, it also checks if python is installed on the system (modern UNIX based distribution usually have python3 installed by default).

\*UNIX system includes MacOs and Linux.

* 1. Hung.bat

This is the windows version of the launch script which allows user to launch the program way easier, this also will give the user the required error message if the user haven’t install python3.

* 1. PIP autoinstallation

HunG also take cares of dependency installation(Currently only termcolor), in order to reduce the hassle that is faced by the user during installation, HunG will attempt to do an automatic installation of the dependency.

## Implementation of HunG language features

As the HunG language is written entirely in vanilla python(pure python), the whole HunG language should be in the capabilities of the python language itself, these are the explaination of how certain language features are implemented in HunG.

1. DataTypes
2. On the current state of HunG, there is only one data type available which is NUM this datatype is also the implementation of the primate Boolean and float datatype in various programming language.

For example:

5 is stored as 5.0 (stored in a float form In python) as there is no difference between int and Boolean value in HunG.

Also 1.0 and 0.0 is the Boolean replacement in the HunG programming language.

1. Unary Operators

The are currently 3 implemented unary operator(operator that only takes 1 value as an input) which are +,-.! (ADD,MIN,NOT). The ADD operator are doing nothing to the value given it is just implemented to avoid unwanted confusion related to unimplemented unary operator. The following are the sample implementation of the Unary operator

+5 will return a 5.0 (as this will multiply float(5) with a +1) which will essentially do nothing.

-5 will return -5.0 (this will multiply float(5) with -1) which will negate the value of the input.

!5 will return 0.0\*

(The result of the calculation will result to 0.0 as with not, any num which is greater than or equals to 1.0 will be considered 1.0 or essentially [true](Boolean)).

1. Binary Operators

Binary Operators are all operation that takes 2 input and will return 1 value, The Binary operators are divided to 4 main category:

* 1. Calculational Operators

The example of this operators are (+,-,\*,/) which is essentially calling the same function inside of python

* 1. Comparator Operators

The example of this operators are >,<,:. The symbol : is exclusive to hung which represent the meaning of == in python(equality comparator operator).

* 1. Bitwise Operators

The are 2 bitwise operators which is classified in the Binary Operator which is AND(&) and OR(|).

* 1. Conditional Operators

The majority of programming language should have a kind of decision taking statement(s) such as if or swich statements, HunG also have it’s own conditional decision taking syntax but it is not implemented as a statement, but it is implemented as an Binary Operator, The implementation itself is quite unique compared to other languages.

The grammar of the syntax is CONDITION ~ RESULT, if the condition is false it will return 0.0 while if the condition is true it will return RESULT.

10 > 5 ~ 20 , this will return 20.0 as 10 is greater than 5

But the possibility doesn’t stop here as we can create an elif (else if) statements by combining multiple expression.

(10>20 ~20):0~30 , this expression will return 30 as 10 is not greater than 20 will return 0 and 0 equals 0 will return 30.

* 1. Helper Commands

HunG also provides user with helper commands that will help them to operate the interpreter or to get the reason of certain actions. For example:

* + 1. .vardump

This helper command will dump the whole variable (implemented in dictionary inside a class in python) to the screen allowing user to see the whole initialized variable.

* + 1. .debug

This helper command will toggle the verbose output and will allow the user to understand what happens behind the scene of the interpreter, this will also display the AST and Tokens that is generated in the parsing and lexing process.

# Evidence of working program

We will be running the program with the following order, the program is launched in a MacOs UNIX system with python3.

1.running the program with the hung.sh

A screenshot of a cell phone

Description automatically generated

A screenshot of a social media post

Description automatically generated

The HunG interpreter will run immediately and will display the interpreter screen.

In order to demonstrate the steps of the interpretation, we would need to toggle verbose output by using .debug.

A screenshot of a cell phone

Description automatically generated

Then we will do a simple variable initialization with an if statement.

Which is equivalent to the current python code:

if (10>5):

a = 10

else:

a = 0

or in the hung lang is:

var a = 10 > 5 ~ 10

A screenshot of a cell phone

Description automatically generated

The program returns the value of 10 which is the result of variable a.

It also displays the whole tokens and AST of the program.

In order to show the value of the variable a we can do a .vardump.

A screenshot of a cell phone

Description automatically generated