|  |  |  |
| --- | --- | --- |
|  | **TABLE OF CONTENT** |  |
| **1. Name of the Language**   1. HAT | |  |
| **2. Programming Paradigm/Power of Language** | |  |
| 2.1. Functional | |  |
| **3. Target Language** | |  |
| 3.1. | C++ |  |
| **4. Translated Type**  4.1. Compiled | |  |

**5. Flex**

**6. Bison**

**7. Keywords**

**8. Syntax**

**Programming Paradigm**

* **Functional language**

It is also known as declarative language. We want to make **HAT** a functional language. The reason is that functional languages have simple syntax as well as simple semantics. In functional type, programs can be made concurrent (Parallelism) automatically. But it has one issue of inefficient execution which needs to be addressed.

**Target Language**

Our language is Trans-piled to C++. The reason for choosing C++ is its performance and also large feature set. C++ is compiled type language. In C++, source code runs through a compiler. C++ is faster, high performance and mostly used language. It is simple, portable, powerful, rich library and most efficient language due to high-level functionalities. So, it is convenient to trans-pile some language to C++.

**Translated Type**

**HAT** will be trans-piled to C++ which is a compiled type language. To convert compiled code to another compiled code is easier. Compiler type language tends to have high performance. We figure out that there is a lack of languages that are simply-oriented and have good performance. Therefore we decided to go with compiled type. **HAT** will be compiled type language.

**Flex**

Flex is used as lexical analyzer. It generates lexers. We give a file of special syntax to it and it generates a C program. This program lexes a string and produced the desired result. It provides multiple built-in functionalities of a lexical analyzer which prevent us to write it from scratch.

**GNU Bison**

It is a predominant parsing library. It uses a file of grammar information to generate C program. This program will do the parsing. Bison provides multiple built-in functionalities, which prevent us to write our own parser.

**Keywords**

They are also known as reserved words. These words are used for special purposes. They can’t be used as variable or function name. Keywords for **HAT** are given below.

|  |  |
| --- | --- |
| func | && |
| elif | variable |
| if | constant |
| else | get (For import) |
| || | ret (For return) |
| show (For print) | true |
| break | false |
| continue | none (For null) |
| class | for |
| while | let (For Datatype) |

**Syntax:**

|  |  |
| --- | --- |
| **Defining variable**  **variable** y  **Assigning value**  **variable** z=21  **Defining a Function:**  **func** first\_function( ){  variable x = 100  **ret** x  }  **func** second\_function(num){  **ret** num+20  }  **Function Calling**  **show**(first\_function( ))  **show**(second\_function(2))  **variable** greet = "Hello"  **constant** name= "Hamza"  **variable** age = 20  **Control Statements**  **variable** num\_1= 0  **if** (num\_1 == 0){  variable num\_2 = first\_function( )  }  **else**{  variable num\_2 = second \_function(5)  } | **Recursion**  **func** third\_function(num){  **if** (num == 1) {  **ret** 1  }  **ret** third\_function(num-1)  }  third\_function(15)  **High Order Function**  **func** f1(my\_func){  **ret** **func** () {  my\_func()  }  }  **constant** f = def () {  **show**("HAT")  }  **variable** var = f1(f)  var() |