Indian Institute of Technology, Gandhinagar

H** Programming Language

Authors:
Pulkit Gautam 21110169
Manav Parmar 21110120
Shubh Agarwal 21110205
Aman Singh 21110020

Assignment 1

Compilers

CS 327

Under the guidance of
Prof. Abhishek Bichawat

February 4, 2024

0.1 Introduction

H{*} is a minimalistic imperative programming language designed for simplicity and ease of use. It incorporates basic types, compound types, conditionals, loops, functions, closures, mutable variables, exceptions, and more.

0.2 Getting Started

0.2.1 Hello, World!

```
print("Hello, World!");
```

0.3 Basic Syntax

0.3.1 Keywords

H** has a set of keywords defined that can't be used as identifiers.

Keyword	Description
arr	mutable collection of fixed length
break	terminates a loop
catch	catches an exception during execution
const	identifier for constant value
continue	skips the iteration
do	do-while loops
elif	conditional statement
else	conditional statement
false	boolean
finally	runs the code block irrespective of the exceptions
for	iterate for some fixed number
func	define a function
if	conditional statement
list	mutable collections with variable length
null	denotes null value
print	returns the ouput to stdout
return	returns output of a function
this	scope variable
throw	throws an exception
true	boolean
try	error handling
tuple	immutable collection of objects
typeof	returns the data type
var	identifier for mutable object
while	iterate till a certain condition is met

Table 1: Keywords in H**

0.3.2 Variables and Assignment

H{*} has two types of variable declaration:

var : Declares a variable and optionally initializes its value.

const : Declares a read-only named constant.

H{*} is case-sensitive. An identifier can start with a letter (a-z, A-Z) or underscore (_) followed by letters, numbers (0-9), or underscores.

```
const x = 5;
var y = 10;
y = 15;

var a, b = 2, 3;
```

0.3.3 Basic Types

H{*} is **dynamically** typed - variables do not need declared types. Types are determined automatically at runtime.

```
var num = 42;        \\Integer

var isTrue = true;   \\Boolean

var text = "Hello";   \\String

var num = null;        \\null
```

0.3.4 Compound Types

There are three compound data types in H{*}:

tuple : Immutable collection of objects with possibly different types.

list : Mutable collection of objects of the same type with variable length.

arr : Mutable collection of objects of the same type.

```
tuple t = [1, "two", true];
list l = [1, 2, 3, 4];

arr a = [1, 2, 3, 4];
```

0.3.5 Conditionals

Conditionals use if, elif, and else statements similar to Java. If the condition in the if clause is true the if-block is run otherwise code jumps to the next elif condition. If any if or elif are true at some point then all the following conditionals are ignored.

```
if (condition) {
// code

} elif (condition) {
// code

} else {
// code

}
```

0.3.6 Loops

H** has three kind of loops namely 'while', 'for' and 'do while'.

while Here loop checks the condition first, if it would pass then run the code inside the ' //code ' brackets. again come to the top ,check the codition and so on till condition fails.

for Here we have three terms in initial bracket separated by two semi colon. first term - we declare and initialise a variable(let i) second term - we check the condition based on declared variable, if pass then go to loop third term - after running code inside the loop, we first return to the third term and do the mentioned operation on the variable and again check the condition mentioned on the second term and do the same as before till condition satisfied of the second term

do while Here we first run the code inside the 'do' loop and then check the condition inside the 'while' if passed then again do the same till condition fails.

```
while (condition) {
// code

}

for (var i = 0; i < 5; i++) {
// code

}

do {
// code

} while (condition);
```

0.4 Functions and Closures

0.4.1 Function Definition

func takes 0 or more argument and return values based on the code written in it. We first write keyword **func**, then name of our function we want to assign, arguments inside '()', and run the code inside ' '.

```
func add(var x, var y) {
return x + y;

}
```

0.4.2 Function Invocation

Functions are invoked by name:

```
var result = add(3, 4);
```

0.4.3 Closures

H{*} has global, module, and function variable scopes. Inner functions can access variables from outer scopes. This is called a *closure*:

```
func outerFunction() {
var outerVar = "I'm from outer!";
```

```
function innerFunction() {
    print(outerVar);
}

return innerFunction;

}

var closure = outerFunction();

closure();
```

0.5 Exception Handling

The *try...catch* statement in H** allows you to define a block of code to execute (the try block), and to define custom behavior in case there are exceptions while running that code (the catch block). When code within the try block throws an exception, execution is immediately shifted to the catch block, bypassing any remaining statements in the try block. It is important to note that *finally* block will execute whether or not an exception is thrown. You may use *finally* to release the resources tied up to your script.

```
try {
// code

} catch (ExceptionType e) {
// handle error

} finally {
// optional cleanup

}

throw ExceptionType("An error occurred");
```

0.6 Additional Features

0.6.1 Unary and Binary Operators

In (a = a + b) we first calculate right side of equivalent '=' and then assign the resultant value to the left side. a++ is equivalent to a = a+1 , but if we pass 'a++' as an argument then a will be passed not a+1 a+=b is equivalent to a = a+b.

```
var a = 5;

a++;

var b = 5

a += b;
```

0.6.2 Booleans

Boolean value is also included in var variable. There will be two values for boolean variable - *true* and *false*.

In condition we'll check if the value is *true* or *false* and proceed accordingly.

```
var a = false;

if (!a) {
// a is false

}
```

0.6.3 Strings

We can declare string using *var* or *const* keywords. String can be added using '+' sign. We can give the string itself or variable which contained the string for addition. Addition of string 'a + b' signifies that, after last character of string a we will have first character of string b and total string will end on last character of string b.

We use slice function if we want a substring of a string. a.slice(i, l) signfies that we want a substring which start from the index i of string a and have a length of l.

```
var a = "Hello";
var b = "World";

var c = a + " " + b;

var a = "Hello World";

var c = a.slice(0, 5);
```

0.6.4 Comparators and if-else

The comparators return a boolean value. If the relation is valid then it returns true otherwise false. like in (a ¿= c) if a is greater or equal to c then return true otherwise false. In (a ¡= c) if a is lesser or equal to c then return true otherwise false. In (a == c)if a is equal to c then return true otherwise false if (a != c)if a is not equal to c then return true otherwise false.

```
if (a >= c) {
  print("Hello");
} elif (a == c) {
  print("World");
} else (a != c) {
  print("Worlds");
}
```

0.6.5 Lists and Arrays

Lists are variable-length, arrays are fixed:

```
list l = [2, 3, 4];

l.append(5);

arr a = [2, 3, 4];
a.append(5); // Error
```

0.6.6 Variable Assignment

const : Assigns an identifier to a constant value.

var : Declares a variable and optionally initializes its value.

```
const a = 5;

var b = 5;
```