2023 年度
言語処理 課題

言語処理
実験レポート

提出日：2024 年 2 月 7 日（水）

学修番号：22140003
氏名：佐倉仙汰郎

1　課題 1

```
Hello sentooooooon! This is the Monkey programming language!
Feel free to type in commands
>> let x = 10
>> x
10
>>
```

図 1: 変数の宣言

2　課題 2

```
sentooooooon@Sentarodell:~/TMU/2/language_pro/waiig_code_1.4/04
Hello sentooooooon! This is the Monkey programming language!
Feel free to type in commands
>> let x = 10
>> x
ERROR: identifier not found: x
>>
```

図 2: Environment 型変数の定義を for ループの中に移動

```go
func Start(in io.Reader, out io.Writer) {
    scanner := bufio.NewScanner(in)
    //env := object.NewEnvironment()

    for {
        fmt.Printf(PROMPT)
        scanned := scanner.Scan()
        if !scanned {
            return
        }

        line := scanner.Text()
        env := object.NewEnvironment()
        l := lexer.New(line)
        p := parser.New(l)

        program := p.ParseProgram()
        if len(p.Errors()) != 0 {
            printParserErrors(out, p.Errors())
            continue
        }

        evaluated := evaluator.Eval(program, env)
        if evaluated != nil {
            io.WriteString(out, evaluated.Inspect())
            io.WriteString(out, "\n")
        }
    }
}
```

図 3: repl/repl.go で変更したコード

Environment 型変数の定義を for ループの中に移動することで、毎回新しい環境が生成される．新しい環境が作られるときに、保持されていた変数がリセットされてしまうため、identifier not found となる．

3 課題 3

```go
func (e *Environment) Get(name string) (Object, bool) {
    var obj Object
    ok := false
    if e.outer != nil {
        obj, ok = e.outer.Get(name)
    }
    if e.outer == nil || !ok {
        obj, ok = e.store[name]
    }
    return obj, ok
}
```

図 4: object/environment.go で変更したコード

```
Hello sentooooooon! This is the Monkey programming language!
Feel free to type in commands
>> let i = 10
>> let f = fn(i){i}
>> f(5)
5
>> i
10
>>
```

図 5: 変更前の結果

```
Hello sentooooooon! This is the Monkey programming language!
Feel free to type in commands
>> let i = 10
>> let f = fn(i){i}
>> f(5)
10
>> i
10
>>
```

図 6: 変更後の結果

let x = 10 では x という変数に 10 が代入されている．また、let f = fn(i)i では引数をそのまま返す関数 f が定義された．変更後のプログラムでは、関数 f の中で変数 i を参照しているときに、i を現在の環境ではなく外部環境から取得するようになる．そのため、f(5) を実行すると、関数 f 内の i が外部環境から取得され、その値である 10 が返されることになり図 6 ような結果が得られる．

4 課題 4

```
>> let newAdder = fn(x){fn(y){x+y}};
>> let addTwo = newAdder(2);
>> let addThree = newAdder(3);
>> addThree
fn(y) {
(x + y)
}
>> addTwo
fn(y) {
```

```
(x + y)
}
```

まず newAdder 関数は x と y を加算したものを返す関数として定義されている．let addTwo = newAdder(2); では newAdder の x に 2 が代入されたものとして定義されており,let addThree = newAdder(3); も同様に x に 3 が代入されている．したがって、addTwo と addThree は、newAdder が返す同一の関数リテラルを参照しており、そのために同じ関数リテラルが表示される．

```
>> addTwo(0)
2
>> addThree(0)
3
```

ここでは二回目に関数を呼び出すことで、addTwo(0) では y に 0 が addThree(0) でも y に 0 が代入されている．addTwo では x = 2,y = 0 として newAdder が呼び出されるので、2 + 0 = 2 として結果が出力され、これは addThree でも同様である．以上の理由からこのような結果をえる．