系統程式 期末專題

B0928005 湯嘉為

B0928010 陳柏曄

B0928011 陳廷恩

目錄

系統發展

1. 發展目標

   預計新增單行 FOR、IF ELSE 功能、GOSUB、SAVE/LOAD 指令，<<, >>, ! 運算子、() 優先運算功能以及數學函式

   其中 FOR, ELSE, GOSUB, LOAD/SAVE 以新增對應 handler 方式實作
   <<, >>, ! 加入在 solveExpression 中
   數學函式以 unary operator 的方式實作
   而 () 優先運算則需要稍加更改 solveExpression 的流程

   另額外增加 math constant, global register, command history 的功能

2. 發展人員

   B0928005 湯嘉為：數學函式, LOAD/SAVE, math constant
   B0928010 陳柏曄：FOR, ELSE, <<, >>, !, (), command history
   B0928011 陳廷恩：GOSUB/RETURN, global register

軟硬體執行平台及發展環境

1. 發展環境

   主要開發環境為 macOS Monterey 12.4
   另有在 Windows 7, 10 及 Ubuntu 20.04 進行過測試
   Python 版本為 3.9.7

2. 執行環境

   需要使用 Python3, Python2 並不支援
   除此之外，並無其他額外安裝套件的需求
   所有用到的套件皆為 Python 內建
   而 getch 則為自定模組，同樣附在專題資料夾內

```python
import os
import sys
import math
import platform
import traceback
from getch import getch
```

擴充功能

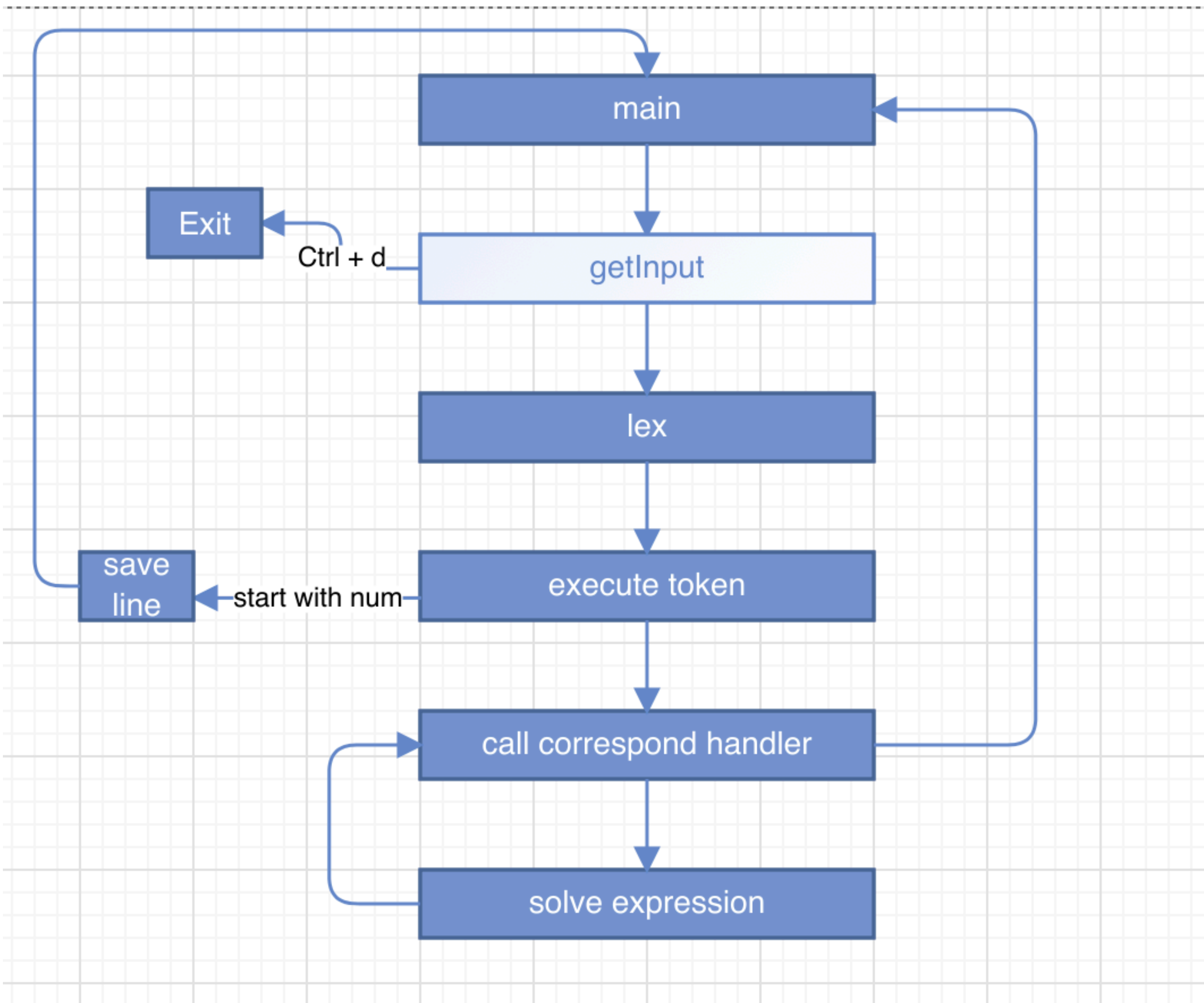
一、 簡介

1. GOSUB/RETURN (理論上可無限巢狀呼叫) (30 point)
2. IF…THEN…ELSE statement (5 points)
3. FOR…TO…DO statement (20 points)
4. SAVA/LOAD (20 points)
5. Extend operators: <<, >>, !, () (5 points * 4)
6. MATH functions:

   "COS", "SIN", "TAN", "ACOS", "ASIN", "ATAN",
   "COSH", "SINH", "TANH", "ACOSH", "ASINH", "ATANH",
   "DEG", "RAD", "ABS", "SQRT", "LOG", "LOG2", "LOG10",
   "EXP", "ROUND", "CEIL", " FLOOR" (5 points * 23)

7. 額外擴充功能
   - global register A, S, L. And commands LDA, STA, LDS, STS, LDT, STT to load from and store into registers.
     (so subroutine can pass value in and pass value back)
   - builtin math constant: PI, E, TAU
   - command history:
     repeat commands by press up and down arrow.

二、 完整流程圖



整體架構上除了以 getInput 這個自訂函式取代了原先使用的內建 input，以實現用上下方向鍵切換先前使用過的指令的功能，其餘流程基本保留，並沒有去做大幅度的更動。

三、 對原始程式潛在 BUG 的修復

1. 修復變數名稱檢查只會檢查第一個字元的錯誤

```
    for c in token[1:]:
-        if not(token[0].lower() in "abcdefghijklmnopqrstuvwxyz0123456789"):
+        # bug fixed: token[0].lower() -> c.lower()
+        if not (c.lower() in "abcdefghijklmnopqrstuvwxyz0123456789_"):
            return False
```

修正前：

```
└─❯ python original.py
Tiny BASIC version 1
by Chung-Yuan Huang
OK.
LET A%+* = 10
OK.
PRINT A%+*
10
OK.
```

修正後：

```
❯ python tb.py
Tiny BASIC version 2
by Jeffrey Chen

<Based on Tiny BASIC version 1 by Chung-Yuan Huang>

\> LET A%+* = 10
Error: Unknown operand A%+*
\>
```

2. 修復遇到未知指令時，不會終止執行的錯誤

```
     if tokens[0][1] != "RESVD":
-        print(f"Error: Unknown command {tokens[0][0]}.")
+        # bug fixed: stopExecution when run into a non-reserved word
+        stopExecution = True
+        print(f"Error: Unknown command {tokens[0]}.")
     else:
```

修正前：

```
OK.
10 PRINT 10
20 FOO
30 PRINT 20

RUN
10
Error: Unknown command FOO.
20
OK.
```

修正後：

```
\> 10 PRINT 10
\> 20 FOO
\> 30 PRINT 20
\>
\> RUN
10
Error: Unknown command ['FOO', 'ID'].
\>
```

3. 修復執行結束後，不會清除 identifiers 的錯誤

```
        elif command == "RUN":
            linePointer = 0
+           # bug fixed: clear identifiers before execution
+           resetExcution()
            while linePointer <= maxLine:
                if linePointer in lines:
                    executeTokens(lines[linePointer])
                    if stopExecution:
                        stopExecution = False
-                       return
+                       break
                linePointer = linePointer + 1
+           # bug fixed: clear identifiers after execution
+           resetExcution()
```

修正前：(第一次執行時定義的變數 a，在第二次執行仍然存在)

```
~/Desktop/repository
❯ python original.py
Tiny BASIC version 1
by Chung-Yuan Huang
OK.
10 LET a = 10
RUN
OK.
10 PRINT a
RUN
10
OK.
```

修正後：

```
\> 10 LET a = 10
\> RUN
\> 10 PRINT a
\> RUN
Error: Variable a not initialized.
\>
```

4. 修復 input 進來的數字，會以字串形式儲存的錯誤

```
            if is_number(varValue):
-               identifiers[varName] = [varValue, "NUM"]
+               # bug fixed: varValue -> float(varValue)
+               identifiers[0][varName] = [float(varValue), "NUM"]
                break
            else:
```

修正前：

```
→ python original.py
Tiny BASIC version 1
by Chung-Yuan Huang
OK.
10 LET a = 10
20 INPUT b
30 PRINT a < b
RUN
?5

Execution halted:
File "/Users/jeffrey/Desktop/repository/tinybasic/original.py", line 388, in sol
veExpression: [TypeError] '<' not supported between instances of 'float' and 'st
r'
OK.
```

試著將變數印出發現，input 進來的 b 是以 ‘5’ 這樣的字串形式儲存

因此 < 運算才會出錯

```
OK.
LET a = 10
{'a': [10.0, 'NUM']}
OK.
INPUT b
?5
{'a': [10.0, 'NUM'], 'b': ['5', 'NUM']}
OK.
```

修正後：

```
\> 10 LET a = 10
\> 20 INPUT b
\> 30 PRINT a < b
\> RUN
?5
0
\>
```

5. 修復 PRINT 指令會改變變數型態的錯誤

```
        return
+    # bug fixed: print out a number will cause it convert to int
+    value = exprRes[0]
     if exprRes[1] == "NUM":
-        exprRes[0] = getNumberPrintFormat(exprRes[0])
-    print(exprRes[0])
+        value = getNumberPrintFormat(value)
+    print(value)
+    return True
+
```

```
OK.
LET a = 10
{'a': [10.0, 'NUM']}
OK.
PRINT a
10
{'a': [10, 'NUM']}
OK.
```

可以看到，在 PRINT 之後，a 的儲存形式更改為整數了

雖目前不至於影響正常執行，但卻可能成為未來潛在 bug 的引子

6. 修復在程式執行後沒有重置 stop execution 的錯誤

```
                    executeTokens(lex(nextLine))
+                   # bug fixed: reset stopExecution when a command is done
+                   stopExecution = False
            except KeyboardInterrupt:
```

在執行指令後，若是沒有重置 stop execution ，會導致下次執行時只有第一行程式被成功執行。

修正前：

```
> python original.py
Tiny BASIC version 1
by Chung-Yuan Huang
OK.
PRINT b
Error: Variable b not initialized.
OK.
10 PRINT 10
20 PRINT 20
30 PRINT 30
RUN
10
OK.
RUN
10
20
30
OK.
```

如前面所述，在 PRINT b 時 stop execution 被設為了 True

但之後卻沒有被重置

以至於執行 RUN 指令時，只執行了第一行，便直接結束執行

執行第二次時，因為已經被重置

才順利的全部執行完全

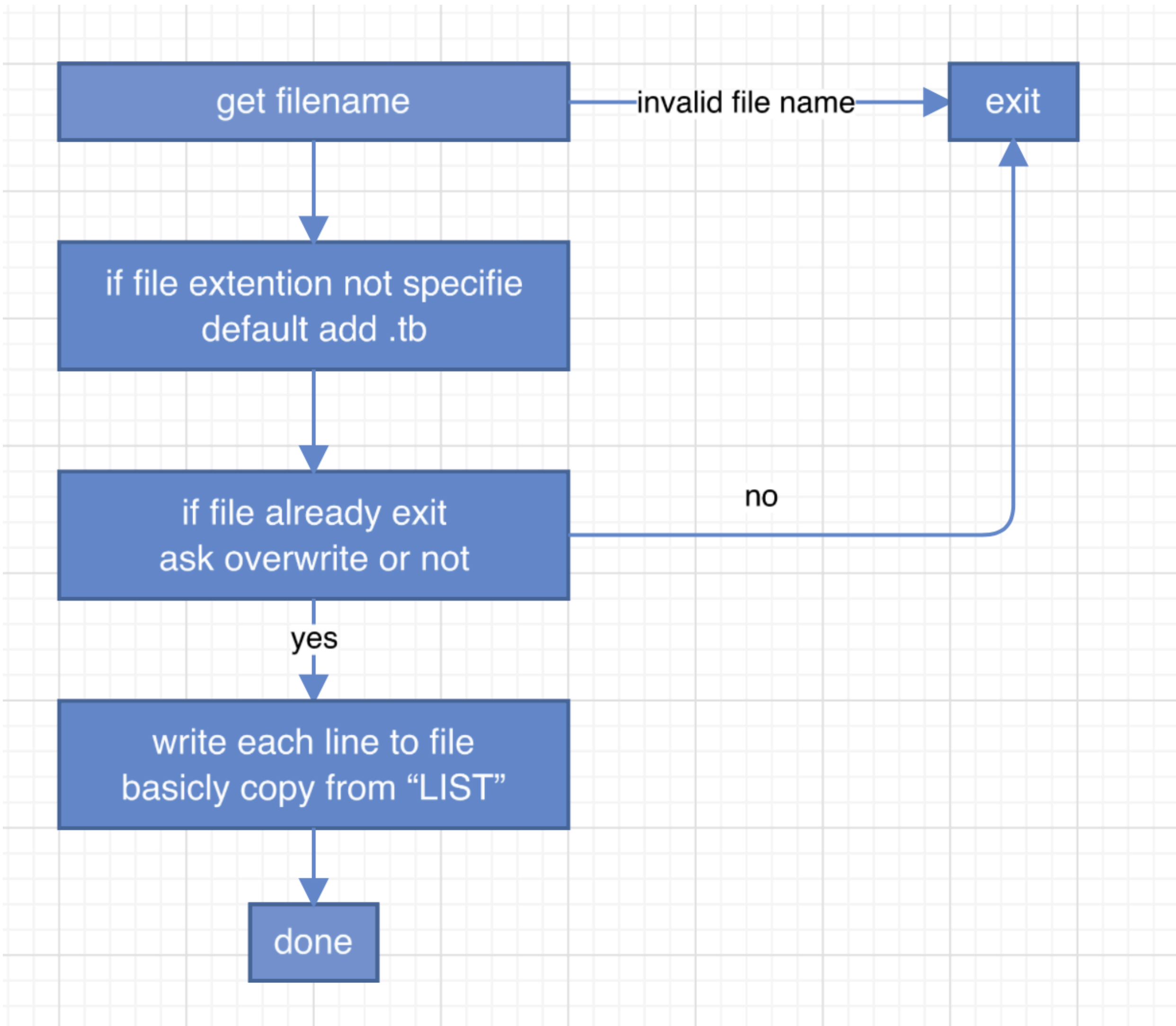
四、 新增功能

1. Load/save

SAVE 原始程式碼及註解：

```python
def saveHandler(tokens):
    global lines, maxLine, printReady
    printReady = True
    if len(tokens) != 1:
        print("Error: Invalid arguments.")
        return False
    if tokens[0][1] != "STRING":
        print("Error: Invalid filename.")
        return False
    filename = tokens[0][0]
    # if file extension not specified, add .tb
    if '.' not in filename:
        filename = filename + '.tb'
    # if the file already exists, ask the user if he wants to overwrite it
    if os.path.isfile(filename):
        overwrite = input(f"File {filename} already exists. Overwrite? (y/n)")
        if overwrite.lower() != "y":
            return False
    with open(filename, 'w') as f:
        # basicly copy from "LIST" command
        for i in range(maxLine + 1):
            if i in lines:
                line = str(i)
                for token in lines[i]:
                    tokenVal = ""
                    if token[1] == "NUM":
                        tokenVal = getNumberPrintFormat(token[0])
                    elif token[1] == "STRING":
                        tokenVal = f"\"{token[0]}\""
                    else:
                        tokenVal = token[0]
                    line += " " + str(tokenVal)
                f.write(line + "\n")
    return True
```

SAVE 流程圖：



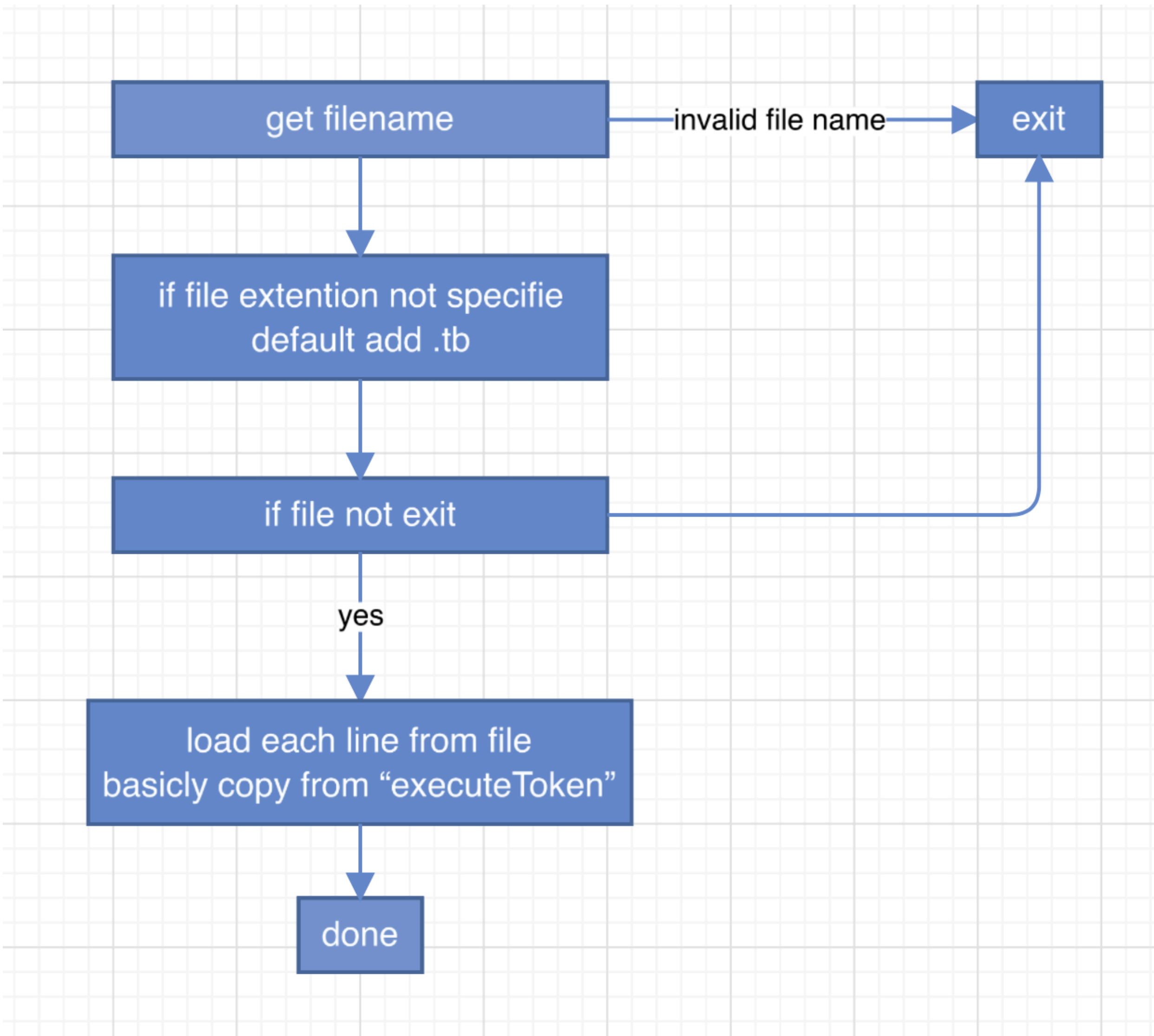

```
\>
\> SAVE "hello.tb"
File hello.tb already exists. Overwrite? (y/n)n
\>
```

若是檔案已存在，詢問是否覆寫

LOAD 原始程式碼及註解：

```python

def loadHandler(tokens):
    global lines, maxLine, printReady
    printReady = True
    if len(tokens) != 1:
        print("Error: Invalid arguments.")
        return False
    if tokens[0][1] != "STRING":
        print("Error: Invalid filename.")
        return False
    filename = tokens[0][0]
    # if file extension not specified, add .tb
    if '.' not in filename:
        filename = filename + '.tb'
    try:
        with open(filename, 'r') as f:
            # basicly copy from "if tokens[0][1] == "NUM":" in executeTokens()
            clearLines()
            for line in f:
                tokens = lex(line.strip())
                if len(tokens) == 0:
                    continue
                if tokens[0][1] == "NUM":
                    lineNumber = int(tokens.pop(0)[0])
                    if len(tokens) != 0:
                        lines[lineNumber] = tokens
                        if lineNumber > maxLine:
                            maxLine = lineNumber
                    else:
                        lines.pop(lineNumber, None)
                else:
                    print("Error: Invalid line number.")
                    return False
    except FileNotFoundError:
        print("Error: File not found.")
        return False
    return True

```

LOAD 流程圖：




```
\>
\> LOAD "hello"
\> |
```

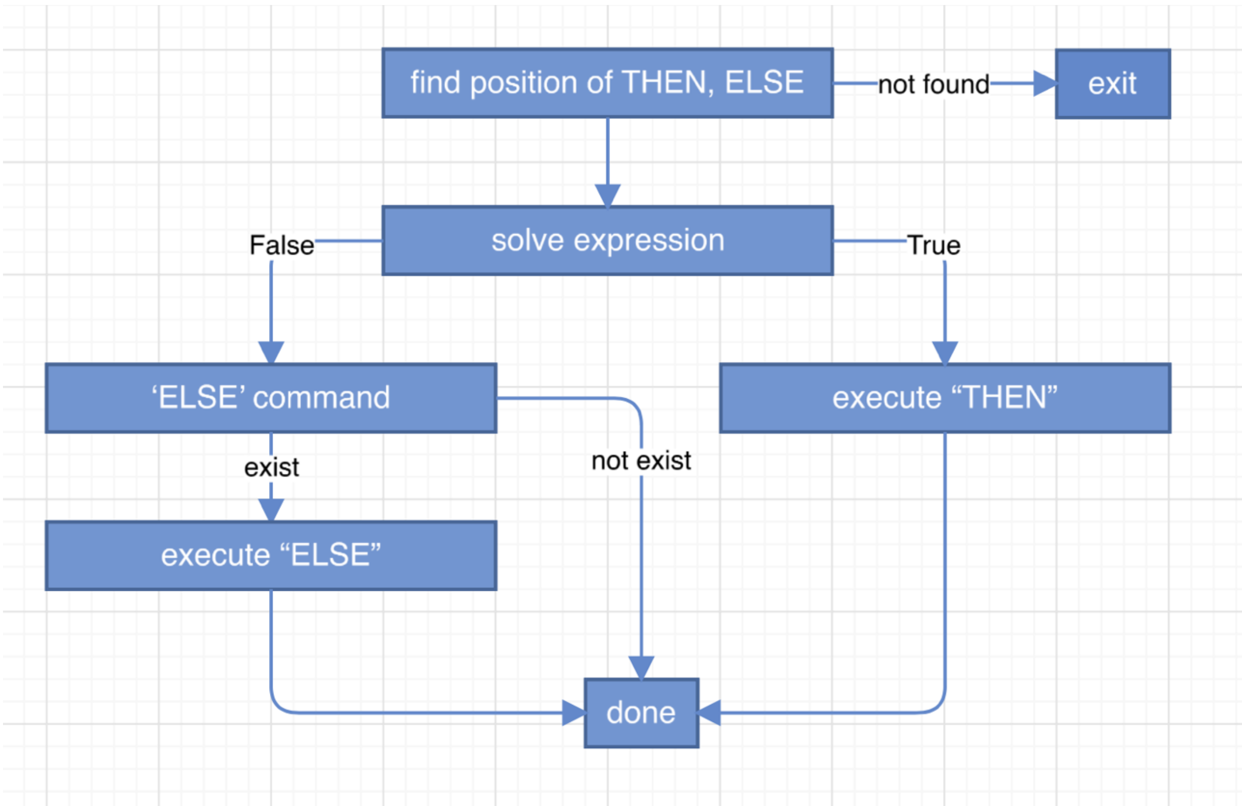
若沒有指定副檔名，預設為 .tb

2. if...then...else

原始程式碼及註解：

```python
def ifHandler(tokens):
    thenPos = elsePos = None
    for i in range(0, len(tokens)):
        if tokens[i] == ["THEN", "RESVD"]: # THEN is change to be a reserved word
            thenPos = i
            break
    for i in range(0, len(tokens)): # find the position of "ELSE"
        if tokens[i] == ["ELSE", "RESVD"]:
            elsePos = i
            break
    # if "THEN" is not found or "ELSE" is found before "THEN"
    if thenPos == None or (elsePos and thenPos > elsePos):
        print("Error: Malformed IF statement.")
        return
    exprValue = solveExpression(tokens[0:thenPos], 0)
    if exprValue == None:
        return
    elif exprValue[0] != 0:
        if len(tokens[thenPos+1:elsePos]) == 0:
            print("Error: Malformed IF statement.")
            return
        executeTokens(tokens[thenPos+1:elsePos])
    # if "ELSE" is found, and exprValue is False
    # execute the expression after "ELSE"
    elif elsePos:
        if len(tokens[elsePos+1:]) == 0:
            print("Error: Malformed IF statement.")
            return
        executeTokens(tokens[elsePos+1:])
    return True
```

If…then…else 流程圖：



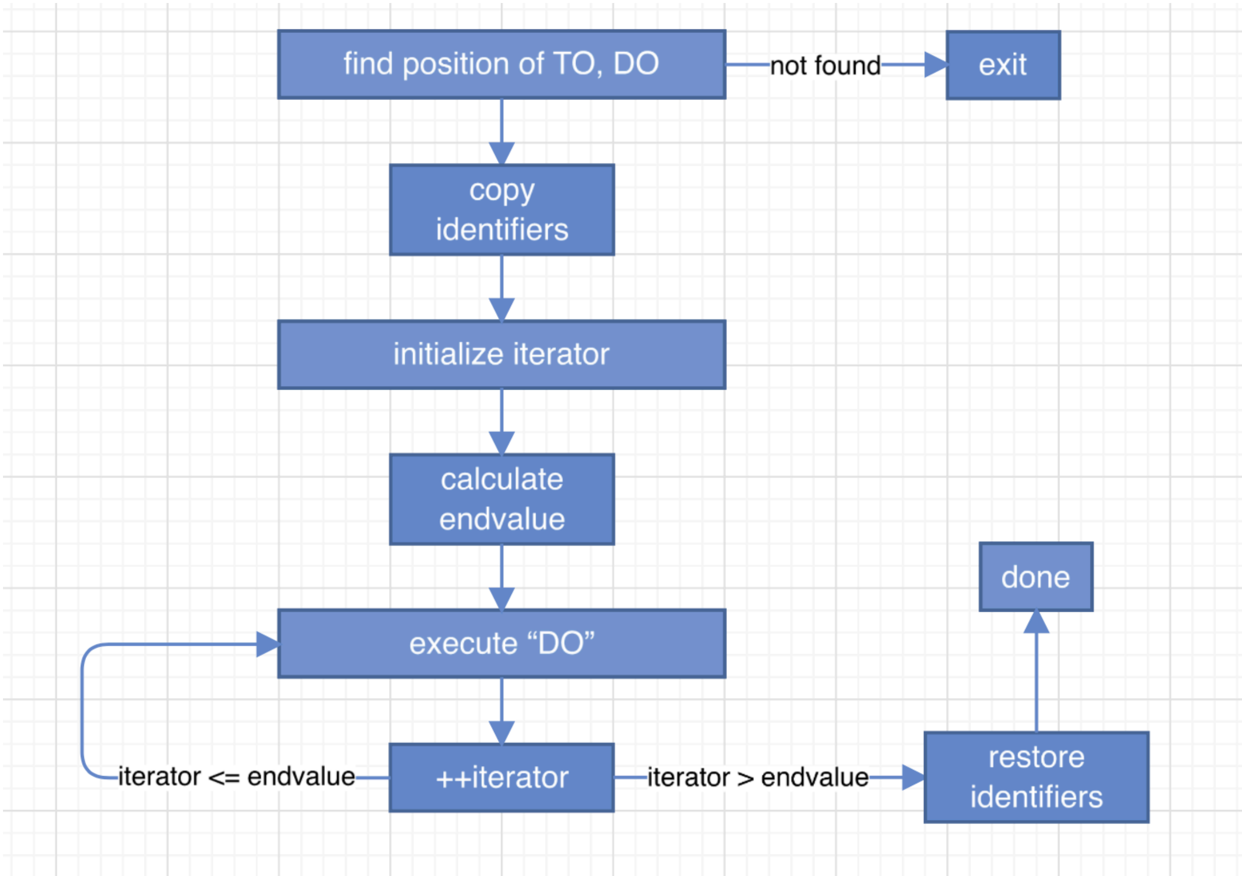
在原先程式的架構基礎上，新增在 expression 為 False 且 ELSE 存在的情況下，執行 ELSE statement

3. FOR…TO…DO

原始程式碼及註解：

```python
def forHandler(tokens):
    global identifiers
    toPos = doPos = None
    # find the position of "TO" and "DO"
    for i in range(0, len(tokens)):
        if tokens[i] == ["TO", "RESVD"]:
            toPos = i
            break
    for i in range(0, len(tokens)):
        if tokens[i] == ["DO", "RESVD"]:
            doPos = i
            break
    if toPos == None or doPos == None or toPos > doPos:
        print("Error: Malformed FOR statement.")
        return
    # get a copy of iterator variable
    iterVar = None
    if getIdentifierValue(tokens[0][0]) != None:
        iterVar = getIdentifierValue(tokens[0][0])
    # set the iterator to the first value
    executeTokens([["LET", "RESVD"]] + tokens[0:toPos])
    # calculate the end value
    endValue = solveExpression(tokens[toPos+1:doPos], 0)
    if endValue == None:
        return
    if endValue[1] != "NUM":
        print("Error: Expected number.")
        return
    endValue = endValue[0]
    # execute the FOR statement
    while getIdentifierValue(tokens[0][0])[0] <= endValue:
        executeTokens(tokens[doPos+1:])
        tokens[toPos - 1][0] += 1
        executeTokens([["LET", "RESVD"]] + tokens[0:toPos])
    # restore the iterator variable
    if iterVar != None:
        executeTokens([["LET", "RESVD"]] + tokens[0:toPos - 1] + [iterVar])
    return True
```

FOR…TO…DO 流程圖：



為了在 iterator 不會干擾到外界變數的同時，又可以正常使用外界變數。

若是有同名外界變數存在，會先將其值複製一份下來，結束後再恢復。

若是 iterator 和外界變數名稱一致時，將暫時作為區域變數將其覆蓋。

4. GOSUB/RETURN, global register

程式碼及註解：

```
 stopExecution = False
-identifiers = {}
+# change identifiers to be a list of set, in order to call subroutine
+identifiers = [{}]
+returnPos = []
 printReady = True
```

為了達成 subroutine 能夠擁有自己的變數 scope，這邊將 identifiers 更改為了 list of set，同時所有使用到 identifiers 的程式片段都改使用 identifiers[0]

GOSUB:

```python

def gosubHandler(tokens):
    global linePointer, identifiers, returnPos
    if len(tokens) == 0:
        print("Error: Expected expression.")
        return
    newNumber = solveExpression(tokens, 0)
    if newNumber[1] != "NUM":
        print("Error: Line number expected.")
    else:
        returnPos.insert(0, linePointer) # push current line number to stack
        identifiers.insert(0, {}) # variable scope for subroutine
        linePointer = newNumber[0] - 1 # jump to subroutine
    return True
```

RETURN:

```python

def returnHandler():
    global linePointer, identifiers, returnPos
    if len(returnPos) == 0:
        print("Error: Not in a subroutine.")
        return
    linePointer = returnPos.pop(0) # pop current line number from stack
    identifiers.pop(0)
    return True

```

global register 程式碼及註解：

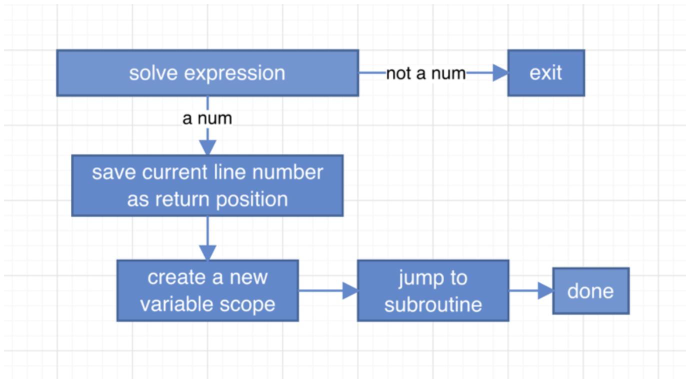
這邊只以 STA, LDA 為例，STS, STT, LDS, LDT 皆為相同架構

```python
registers = {
    "A": 0,
    "S": 0,
    "T": 0,
}
```

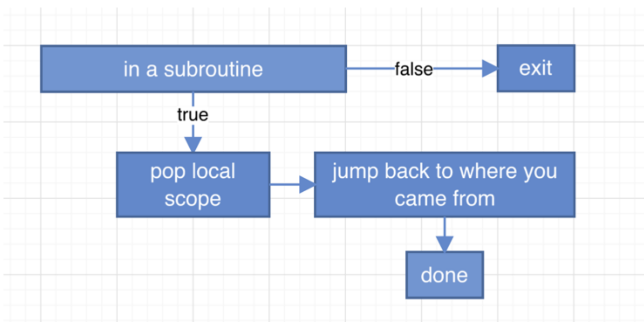
```python

# store number to rigister A
def staHandler(tokens):
    global registers
    if len(tokens) == 0:
        print("Error: Expected identifier.")
        return
    exprRes = solveExpression(tokens, 0)
    if exprRes == None:
        return
    if exprRes[1] != "NUM":
        print("Error: Rigister A expected number.")
        return
    registers["A"] = exprRes[0]
    return True
```

```python

# load number from rigister A
def ldaHandler(tokens):
    global registers
    varName = None
    if len(tokens) == 0:
        print("Error: Expected identifier.")
        return
    elif len(tokens) == 1 and tokens[0][1] == "ID":
        varName = tokens[0][0]
    else:
        varName = solveExpression(tokens, 0)[0]
        if not(isValidIdentifier(varName)):
            print(f"Error: {varName} is not a valid identifier.")
            return
    if getVarType(varName) != "NUM":
        print(f"Error: Variable {varName} is not a number.")
        return
    executeTokens([["LET", "RESVD"], [varName, "ID"], ["=", "ASGN"], [registers["A"], "NUM"]])
    return True

```

流程圖：

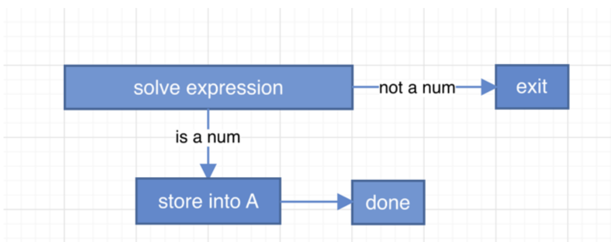


GOSUB

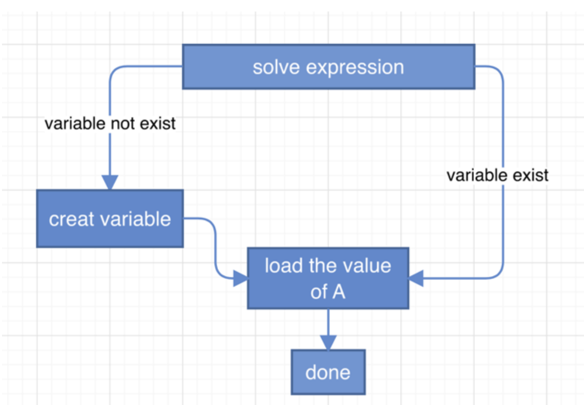


RETURN

STA, LDA 是為了使 subroutine 能夠做到類似傳入參數及回傳值的功能



STA



LDA

5. 新增 operators

- <<, >>

```
                    return [exprResL[0] >= exprResR[0], "NUM"]
                # operator <<
                elif tokens[i][0] == "<<":
                    if exprResL == None or exprResR == None:
                        print("Error: Operator expects value.")
                        return None
                    if exprResL[1] != "NUM" or exprResR[1] != "NUM":
                        print("Error: Operand type mismatch.")
                        return None
                    if int(exprResL[0]) != exprResL[0] or int(exprResR[0]) != exprResR[0]:
                        print("Error: Operand type mismatch.")
                        return None
                    return [float(int(exprResL[0]) << int(exprResR[0])), "NUM"]
                # operator >>
                elif tokens[i][0] == ">>":
                    if exprResL == None or exprResR == None:
                        print("Error: Operator expects value.")
                        return None
                    if exprResL[1] != "NUM" or exprResR[1] != "NUM":
                        print("Error: Operand type mismatch.")
                        return None
                    if int(exprResL[0]) != exprResL[0] or int(exprResR[0]) != exprResR[0]:
                        print("Error: Operand type mismatch.")
                        return None
                    return [float(int(exprResL[0]) >> int(exprResR[0])), "NUM"]
```

這部分程式碼架構和其他運算子幾乎一模一樣

除了因為左移右移運算子只接受整數，因此必須先轉為整數計算

再將結果轉回浮點數型態

- !

```
                        return [value1 + value2, "STRING"]
                # operator !
                elif tokens[i][0] == "!":
                    # as an unary operator, ! only takes one argument
                    if exprResR == None:
                        print("Error: Operator expects value.")
                        return None
                    if exprResL != None:
                        print("Error: ! is an unary operator.")
                        return None
                    if exprResR[1] == "NUM":
                        return [not exprResR[0], "NUM"]
                    else:
                        print("Error: Operand type mismatch.")
                        return None
```

由於 ! operator 是 unary operator

exprResL 必須有值，exprResR 則必須為 None

- ()

() operator 在實作上和其他運算子較為不同，是為了實現「括號內優先計算」這個四則運算規則，而非是單純的接收 operand 並回傳值。

```
    for c in line:
-       if not(inString) and c in " ()\"":
+       if not(inString) and c in " \"":
            if len(currentToken) != 0:
```

首先要先改寫 lex 使其不會將 () 忽略

```
            inString = False
        # to make expressions like "(1 + 2) * 3" work
        # parentesis don't have to separate with other tokens by spaces
        elif not(inString) and  c in ['(', ')']:
            if len(currentToken) != 0:
                tokens.append([currentToken, "TBD"])
                currentToken = ""
            tokens.append([c, "PAREN"]) # parentesis
        else:
            currentToken += c
```

接著，為了讓括號可以和數字貼在一起，不需要用空格分開。不能和其他類型 Token 一並處理，要拉出來做特殊的處置。

```
                    return None
            elif tokens[i][1] == "PAREN":
                # find the matching close parentheses
                close = findMatchingClose(tokens, i)
                if close == None:
                    print("Error: Unmatched parentheses.")
                    return None
                # solve the expression inside the parentheses
                subExpr = solveExpression(tokens[i+1:close], 0)
                if subExpr == None:
                    return None
                leftSideValues.append(subExpr)
                # continue to the next token
                i = close
```

在 solveExpression 中，當遇到上括弧時，將對應的下括弧找出，並優先計算出之間被包住的 subexpression，之後直接跳到下括弧的下一個 token。

為了達到任意跳過中間的 token，原先使用的 for 迴圈要改為使用 while 迴圈。

```
    rightSideValues = []
    if level < len(operators):
-       for i in range(0, len(tokens)):
-           if not(tokens[i][1] in ["OP", "NUM", "STRING", "ID"]):
+       i = 0
+       while i < len(tokens):
+           if not(tokens[i][1] in ["OP", "NUM", "STRING", "ID", 'PAREN']):
                print(f"Error: Unknown operand {tokens[i][0]}")
```

```python
def findMatchingClose(tokens, openIndex):
    openCount = 1
    for i in range(openIndex + 1, len(tokens)):
        if tokens[i][0] == "(":
            openCount += 1
        elif tokens[i][0] == ")":
            openCount -= 1
        if openCount == 0:
            return i
    return None
```

用於找對應下括弧的函式

6. 數學函式

```diff
-        elif isValidIdentifier(token[0]):
+        elif isValidIdentifier(token[0]) and token[0] not in math_functions:
             token[1] = "ID" #Identifier
```

不接受和數學函式同名的變數

```python
        # math function also handle as a unary operator
        elif tokens[i][0] in math_functions:
            if exprResR == None:
                print("Error: Operator expects value.")
                return None
            if exprResL != None:
                print("Error: Function is unary operator.")
                return None
            if exprResR[1] == "NUM":
                return [math_functions[tokens[i][0]](exprResR[0]), "NUM"]
            else:
                print("Error: Operand type mismatch.")
                return None
```

實作上，以直接使用原有架構為優先，因此數學函式是採用 unary operator 的方式實作，直接加入到 solveExpression 的環節進行，並沒有額外延伸新的架構出來。

```python
math_functions = {
    "COS": math.cos,
    "SIN": math.sin,
    "TAN": math.tan,
    "ACOS": math.acos,
    "ASIN": math.asin,
    "ATAN": math.atan,
    "COSH": math.cosh,
    "SINH": math.sinh,
    "TANH": math.tanh,
    "ACOSH": math.acosh,
    "ASINH": math.asinh,
    "ATANH": math.atanh,
    "DEG": math.degrees,
    "RAD": math.radians,
    "ABS": math.fabs,
    "SQRT": math.sqrt,
    "LOG": math.log,
    "LOG2": math.log2,
    "LOG10": math.log10,
    "EXP": math.exp,
    "ROUND": lambda x: float(round(x)),
    "CEIL": lambda x: float(math.ceil(x)),
    "FLOOR": lambda x: float(math.floor(x))
}
```

以字典的形式將函式名稱和 Python 內建數學函式進行對應。

有些函式回傳值是以整數形式，因此以 lambda 的方式將其重新轉回浮點數再回傳。

```python
constants = {
    "PI": math.pi,
    "E": math.e,
    "TAU": math.tau,
}
```

```python
            elif value in constants:
                token[0] = constants[value.upper()]
                token[1] = "NUM" #built-in constant
```

數學常數在 lex 環節就會直接轉為對應浮點數，並以 NUM 形式儲存。

五、 其餘更動

```python
                    break
            except Exception as e: # show traceback when error occurs
                error_class = e.__class__.__name__ #取得錯誤類型
                detail = e.args[0] #取得詳細內容
                cl, exc, tb = sys.exc_info() #取得Call Stack
                lastCallStack = traceback.extract_tb(tb)[-1] #取得Call Stack的最後一筆資料
                fileName = lastCallStack[0] #取得發生的檔案名稱
                lineNum = lastCallStack[1] #取得發生的行號
                funcName = lastCallStack[2] #取得發生的函數名稱
                errMsg = "File \"{}\", line {}, in {}: [{}] {}".format(fileName, lineNum, funcName, error_class, detail)
                print("\nExecution halted:\n"+errMsg)
```

增加更詳盡的錯誤訊息描述，以更好的鎖定問題

```python
def clearLines(): # clear all codes
    global lines, maxLine
    lines = {}
    maxLine = 0

def resetExcution(): # reset all variables and registers
    global identifiers, returnPos, registers
    identifiers = [{}]
    returnPos = []
    registers = {
        "A": 0,
        "S": 0,
        "T": 0,
    }
```

將重置變數和清空程式碼獨立抽出來成函式，未來擴充更方便

```
-        elif value.upper() == "THEN":
-            token[0] = value.upper()
-            token[1] = "THEN"
```

將 THEN 改以 RESVD 儲存，不再單獨分出一個類別

```
                if len(rightSideValues) != 0:
                    exprResR = solveExpression(rightSideValues, level)
-               if exprResL == None or exprResR == None:
-                   return None
```

所有 operator 在執行前都有先確認過左右是否為空

重複確認似乎沒有必要性

```
                elif exprResL[1] == "NUM" and exprResR[1] == "NUM":
-                   return [float(exprResL[0]) + float(exprResR[0]), "NUM"]
+                   return [exprResL[0] + exprResR[0], "NUM"]
                else:
```

只要類別是 “NUM”，儲存方式一定是浮點數

特意轉為 float 似乎沒有必要

```
def getIdentifierValue(name):
    try:
        return identifiers[0][name].copy()
    except KeyError:
        return None
```

若是試圖取得不存在得變數，回傳 None

同時將回傳方式改為回傳 copy，避免產生「修改一個變數，其他變數也被修改」的錯誤

六、 Command History

通過上下方向鍵切換先前使用過的指令（如同一般電腦內建 terminal, Python shell, MATLAB 等等都有提供的功能）

目前無法支援左右方向鍵移動光標，從中間修改指令的功能

```diff
-                    print("OK.")
-               nextLine = input()
+                    # not a bug fixed, just prefer this way
+                    print("\>", end = " ", flush = True)
+               nextLine = getInput()
```

在 main 裡用自訂的 getInput() 取代掉 Python 內建的 input()

```python
def clearCommand():
    # clear current line in stdout
    if platform.system() == "Windows":
        print("\r" + ' ' * 64 + "\r", end = "", flush = True)
    else:
        print("\x1b[1K\r", end = "", flush = True)
    print("\>", end = " ", flush = True)

key = {
    "special": b'\xe0' if platform.system() == "Windows" else '\x1b',
    "enter": b'\x0d' if platform.system() == "Windows" else '\x0d',
    "backspace": b'\x08' if platform.system() == "Windows" else '\x7f',
    "ctrl+d": b'\x04' if platform.system() == "Windows" else '\x04',
    "up": b'\xe0H' if platform.system() == "Windows" else '\x1b[A',
    "down": b'\xe0P' if platform.system() == "Windows" else '\x1b[B'
}
```

定義函示 clearCommand 用於清空目前所在這一行的 stdout

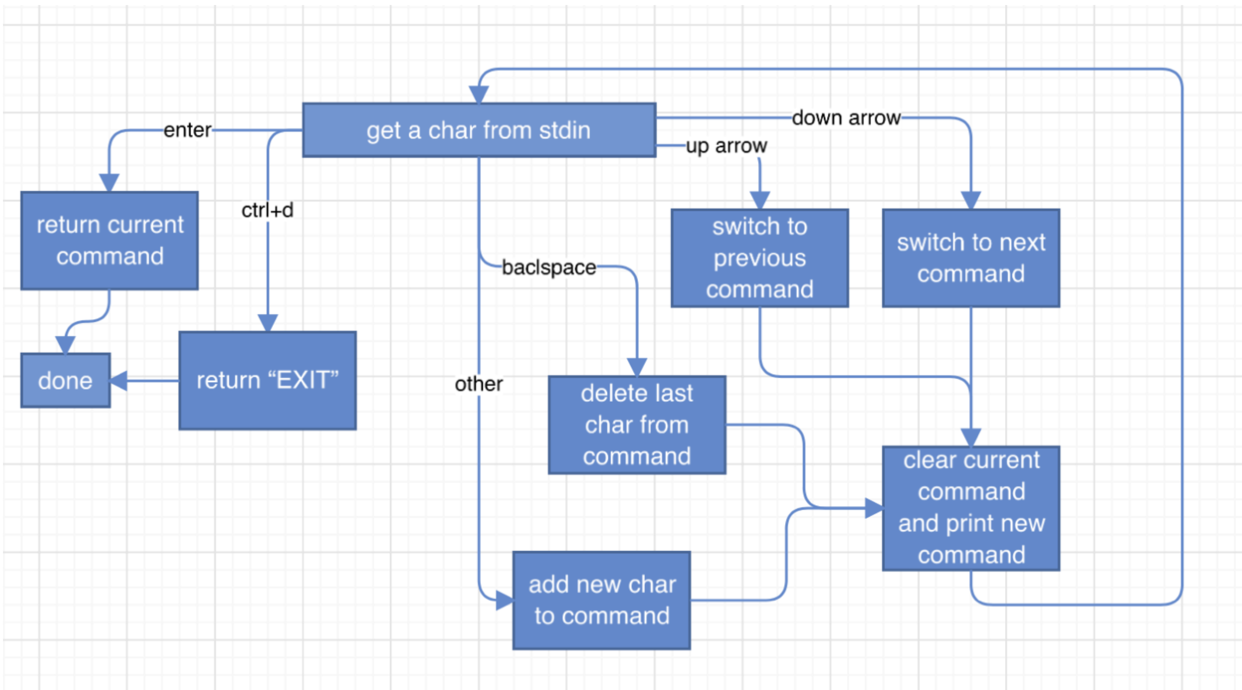
以及各按鍵對應的編碼

並分成 Windows 版本及 macOS, Linux 版本

程式碼及註解

```python

# function to get input command
# can repeat commands had been executed by up and down arrow
def getInput():
    global commands, currentCommand
    ss = ""
    while True:
        c = getch()
        if c == key["special"]:
            if platform.system() == "Windows": c += getch()
            else: c += getch() + getch()
            if c == key['up']:
                if currentCommand <= 0: continue
                currentCommand -= 1
            if c == key['down']:
                if currentCommand >= len(commands) - 1: continue
                currentCommand += 1
            clearCommand()
            print(commands[currentCommand], end = "", flush = True)
            ss = commands[currentCommand]
        elif c == key['enter']:
            print()
            if ss: # if there is a command in ss, return it
                commands.insert(-1, ss.strip())
                currentCommand = len(commands) - 1
                return ss
            else: # if there is no command in ss, return empty string
                return ""
        elif c == key['ctrl+d']:
            return "EXIT"
        # if c is a back space, delete the last character
        elif c == key['backspace']:
            if ss:
                ss = ss[:-1]
                clearCommand()
                print(ss, end = "", flush = True)
        else: # if c is a normal character, add it to ss
            if platform.system() == "Windows": c = c.decode('utf-8')
            print(c, end = "", flush = True)
            ss = ss + c

```

流程圖：

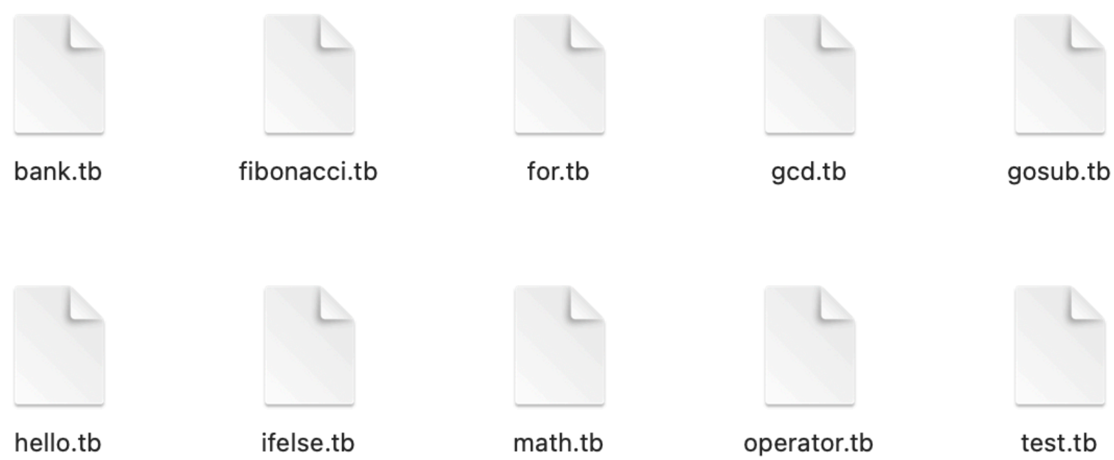


get a char from stdin
enter
ctrl+d
down arrow
up arrow
return current command
baclspace
switch to previous command
switch to next command
done
return “EXIT”
other
delete last char from command
clear current command and print new command
add new char to command

範例程式

程式碼內容在此只展示截圖和預期 output 說明

原程式檔都有附在專題資料夾內



共十個 .tb 檔，可直接用 LOAD 功能載入

1. hello.tb

展示功能：LOAD/SAVE

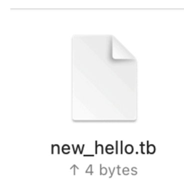
```
10 REM +--------------+
20 REM | hello world |
30 REM +--------------+
40 LET hello$ = "hello world"
50 PRINT hello$
60 SAVE "new_hello.tb"
70 END
```

```
~/Desktop/iCloud/CGU/系統程式/tinybasic/final  git  master
> python tb.py
Tiny BASIC version 2
by Jeffrey Chen

<Based on Tiny BASIC version 1 by Chung-Yuan Huang>

\> LOAD "hello.tb"
\> LIST
10 REM +--------------+
20 REM | hello world |
30 REM +--------------+
40 LET hello$ = "hello world"
50 PRINT hello$
60 SAVE "new_hello.tb"
70 END
\> RUN
hello world
\> |
```

Load in hello world program from ”hello.tb”

Print out the codes just load in

Execute and print out “hello world”.



執行完成後，資料夾內應產生出一個“new_hello.tb” file. 裡面存著一樣的程式碼。

2. ifelse.tb

```
10 REM +--------------------+
20 REM | if else statements |
30 REM +--------------------+
40 LET a = 10
50 PRINT "input a number"
60 INPUT b
70 IF a > b THEN GOTO 50 ELSE PRINT b . " is bigger than 10"
80 END
```

設立一個變數 a 並設值為 10

接著要求一個輸入

若輸入的數字小於 10 則重新輸入

反之則印出該數字

```
\> LOAD "ifelse.tb"
\> RUN
input a number
?2
input a number
?3
input a number
?9
input a number
?23
23 is bigger than 10
\>
```

3. for.tb

```
10 REM +----------------+
20 REM | for statements |
30 REM +----------------+
40 LET a = 10
50 PRINT "in global scope, a = " . a
60 INPUT b
70 FOR a = 1 TO b DO PRINT "a = " . a
80 PRINT "after leave the scope, a = " . a
90 END
```

宣告變數 a = 10 在全域

進入 for 迴圈之後全域變數 a 被區域變數 a 屏蔽

因此印出 “a = 1” ~ “a = {b}”

離開迴圈後，回到 global a，印出 “a = 10”

```
\> LOAD "for.tb"
\> RUN
in global scope, a = 10
?5
a = 1
a = 2
a = 3
a = 4
a = 5
after leave the scope, a = 10
\>
```

4. gosub.tb

```
10 REM +---------------+
20 REM | main program |
30 REM +---------------+
40 LET a = 10
50 PRINT "In global: a = " . a
60 GOSUB 120
70 PRINT "In global: a = " . a
80 END
90 REM +---------------+
100 REM | subroutine 1 |
110 REM +---------------+
120 LET a = 20
130 PRINT "In sub 1: a = " . a
140 GOSUB 200
150 PRINT "In sub 1: a = " . a
160 RETURN
170 REM +---------------+
180 REM | subroutine 2 |
190 REM +---------------+
200 LET a = 30
210 PRINT "In sub 2: a = " . a
220 GOSUB 280
230 PRINT "In sub 2: a = " . a
240 RETURN
250 REM +---------------+
260 REM | subroutine 3 |
270 REM +---------------+
280 LET a = 40
290 PRINT "In sub 3: a = " . a
300 RETURN
```

共三個 subroutine 一層一層向下呼叫

可看出各個 subroutine 擁有自己的變數 scope

並不互相影響

```
\> LOAD "gosub.tb"
\> RUN
In global: a = 10
In sub 1: a = 20
In sub 2: a = 30
In sub 3: a = 40
In sub 2: a = 30
In sub 1: a = 20
In global: a = 10
\>
```

5. operator.tb

```
operator.tb
10 REM +-----------+
20 REM | operators |
30 REM +-----------+
40 INPUT a
50 PRINT (a + 3) * 20
60 IF !(a < 10) THEN PRINT a >> 1 ELSE PRINT a << 1
70 END
```

以上內容展示出 (), !, <<, >> 四種運算子皆能正常運作

```
\> LOAD "operator"
\> RUN
?4
140
8
\> RUN
?32
700
16
\>
```

6. math.tb

```
10 REM +-----------------+
20 REM | math functions |
30 REM +-----------------+
40 PRINT "COS(PI) = " . COS(PI)
50 PRINT "SIN(PI / 2) = " . SIN(PI / 2)
60 PRINT "TAN(PI / 4) = " . TAN(PI / 4)
70 PRINT "ACOS(-1) = " . ACOS(-1)
80 PRINT "ASIN(1) = " . ASIN(1)
90 PRINT "ATAN(0) = " . ATAN(0)
100 PRINT "COSH(1) = " . COSH(1)
110 PRINT "SINH(0) = " . SINH(0)
120 PRINT "TANH(0) = " . TANH(0)
130 PRINT "ACOSH(1) = " . ACOSH(1)
140 PRINT "ASINH(0) = " . ASINH(0)
150 PRINT "ATANH(0) = " . ATANH(0)
160 PRINT "DEG(PI / 2) = " . DEG(PI / 2)
170 PRINT "RAD(180) = " . RAD(180)
180 PRINT "ABS(-1) = " . ABS(-1)
190 PRINT "SQRT(4) = " . SQRT(4)
200 PRINT "LOG(E) = " . LOG(E)
210 PRINT "LOG2(8) = " . LOG2(8)
220 PRINT "LOG10(100) = " . LOG10(100)
230 PRINT "EXP(1) = " . EXP(1)
240 PRINT "ROUND(1.5) = " . ROUND(1.5)
250 PRINT "ROUND(1.4) = " . ROUND(1.4)
260 PRINT "CEIL(1.1) = " . CEIL(1.1)
270 PRINT "FLOOR(1.1) = " . FLOOR(1.1)
280 END
```

展示全部 23 種數學函式及數學常數的使用

```
\> LOAD "math"
\> RUN
COS(PI) = -1
SIN(PI / 2) = 1
TAN(PI / 4) = 0.9999999999999999
ACOS(-1) = 3.141592653589793
ASIN(1) = 1.5707963267948966
ATAN(0) = 0
COSH(1) = 1.5430806348152437
SINH(0) = 0
TANH(0) = 0
ACOSH(1) = 0
ASINH(0) = 0
ATANH(0) = 0
DEG(PI / 2) = 90
RAD(180) = 3.141592653589793
ABS(-1) = 1
SQRT(4) = 2
LOG(E) = 1
LOG2(8) = 3
LOG10(100) = 2
EXP(1) = 2.718281828459045
ROUND(1.5) = 2
ROUND(1.4) = 1
CEIL(1.1) = 2
FLOOR(1.1) = 1
\>
```

7. fibonacci.tb

```
10 REM +-------------------+
20 REM | Fibonacci program |
30 REM +-------------------+
40 PRINT "Enter a number: "
50 INPUT n
60 IF n < 1 THEN PRINT "Number must be greater than 0"
70 IF n < 1 THEN GOTO 50
80 STA n
90 GOSUB 160
100 LDA fibbo
110 PRINT "No." . n ." Fibonacci number is: " . fibbo
120 END
```

主程式部分，讀入一個整數，並呼叫 subroutine 計算出費氏數列

```
130 REM +--------------------+
140 REM | Fibonacci function |
150 REM +--------------------+
160 LDA n
170 IF n < 2 THEN RETURN
180 STA n - 1
190 GOSUB 160
200 LDA a
210 STA n - 2
220 GOSUB 160
230 LDA b
240 STA a + b
250 RETURN
```

Subroutine:

呼叫前將一整數存入暫存器 A 表示要取第幾位的費氏數列，RETURN 後 A 裡面的值就是結果。

若是 n < 2 則直接回傳，否則分別計算 n-1, n-2，並加起來存入 A，最後再回傳。

```
\> LOAD "fibonacci.tb"
\> RUN
Enter a number:
?2
No.2 Fibonacci number is: 1
\> RUN
Enter a number:
?10
No.10 Fibonacci number is: 55
\> RUN
Enter a number:
?20
No.20 Fibonacci number is: 6765
\>
```

8. gcd.tb

gcd.tb

```
10 REM +-------------+
20 REM | GCD program |
30 REM +-------------+
40 PRINT "Enter two numbers: "
50 INPUT A
60 INPUT B
70 STS A
80 STT B
90 GOSUB 150
100 LDS GCD
110 PRINT "The GCD is: " . GCD
120 END
130 REM +--------------+
140 REM | GCD function |
150 REM +--------------+
160 LDT B
170 IF B == 0 THEN RETURN
180 LDS A
190 STS B
200 STT A % B
210 GOSUB 150
220 RETURN
```

主程式部分，讀入兩個整數，並呼叫 subroutine 計算出 GCD

Subroutine:

呼叫前將兩整數存入暫存器 S, T，RETURN 後 S 裡面的值就是結果。

若是 T = 0 則直接回傳，否則將原先 T 的值存入 S，S % T 的值存入 T

接著遞迴呼叫 GCD

```
\> LOAD "GCD.tb"
\> RUN
Enter two numbers:
?363
?44
The GCD is: 11
```

9. bank.tb

```
10 REM +-------+
20 REM | bank |
30 REM +-------+
40 LET balance = 0
50 PRINT "Welcome to the bank"
60 PRINT "Please enter your name"
70 INPUT name$
75 PRINT ""
80 PRINT "What can I do for you, " . name$
90 PRINT "1. Check balance"
100 PRINT "2. Withdraw"
110 PRINT "3. Deposit"
120 PRINT "4. Quit"
130 PRINT "Please enter your choice"
140 INPUT choice
150 IF choice == 1 THEN GOTO 210
160 IF choice == 2 THEN GOTO 240
170 IF choice == 3 THEN GOTO 290
180 IF choice == 4 THEN GOTO 360
190 PRINT "Invalid choice, please try again"
200 GOTO 130
```

```
210 REM +-----------+
220 REM | choice 1 |
230 REM +-----------+
210 PRINT "Your balance is " . balance
220 GOTO 75
230 REM +-----------+
240 REM | choice 2 |
250 REM +-----------+
240 PRINT "Please enter the amount you wish to withdraw"
250 INPUT amount
260 IF amount > balance THEN PRINT "Insufficient funds" ELSE LET balance = balance - amount
270 GOTO 75
280 REM +-----------+
290 REM | choice 3 |
300 REM +-----------+
290 PRINT "Please enter the amount you wish to deposit"
300 INPUT amount
310 LET balance = balance + amount
320 GOTO 75
330 REM +-----------+
340 REM | choice 4 |
350 REM +-----------+
360 PRINT "Thank you for using the bank"
370 END
```

取得使用者輸入選項，並執行對應功能

```
\> LOAD "bank"
\> RUN
Welcome to the bank
Please enter your name
?Jeffrey

What can I do for you, Jeffrey
1. Check balance
2. Withdraw
3. Deposit
4. Quit
Please enter your choice
?3
Please enter the amount you wish to deposit
?1000

What can I do for you, Jeffrey
1. Check balance
2. Withdraw
3. Deposit
4. Quit
Please enter your choice
?2
Please enter the amount you wish to withdraw
?200
```

```
What can I do for you, Jeffrey
1. Check balance
2. Withdraw
3. Deposit
4. Quit
Please enter your choice
?1
Your balance is 800

What can I do for you, Jeffrey
1. Check balance
2. Withdraw
3. Deposit
4. Quit
Please enter your choice
?4
Thank you for using the bank
\>
```

10. test.tb

```
10  REM +--------------------+
20  REM |Tiny BASIC test code|
30  REM +--------------------+
40  LET a = 10
50  INPUT b
60  FOR a = 1 TO b DO PRINT a
70  INPUT c
80  IF !(a < c) THEN GOTO 70 ELSE PRINT c
90  INPUT d
100 STA d
110 GOSUB 170
120 PRINT "a = " . a
130 END
140 REM +--------------+
150 REM |SUBROUTINE 170|
160 REM +--------------+
170 LET a = 20
180 LDA d
190 PRINT ((1 << 1) + 2 * (1 + (8 >> 2))) * (a + d)
200 GOSUB 280
210 LDS e
220 PRINT e
230 IF !(10 > 20) THEN PRINT "hello world"
240 RETURN
250 REM +--------------+
260 REM |SUBROUTINE 280|
270 REM +--------------+
280 STS COS(PI) + LOG(E)
290 FOR i = 1 TO 3 DO PRINT "Test " . i
300 RETURN
```

```
\> LOAD "test.tb"
\> RUN
?3
1
2
3
?12
12
?3
184
Test 1
Test 2
Test 3
0
hello world
a = 10
```

參考文獻

在 getch.py 內所使用的 getch 程式碼來自於 StackOverflow

How to read a single character from the user?