編譯器 Bossattack 3 - ＢＣ小題

此大題要實作 **python list** 的子集語言，能跟 **python** 中的 **list** 做相似的操作

ex:
下面是實際 python 在做 list 相關操作的範例，註解是預期的結果

```
print([1, 3, 4] + [5, 6] + [7]) // expected [1, 3, 4, 5, 6, 7]
print([1, 3, 4] * 2) // expected [1, 3, 4, 1, 3, 4]
print([0] * 5) // expected [0, 0, 0, 0, 0]
print([1, [3, 4]] * 4) // expected [1, [3, 4], 1, [3, 4], 1, [3, 4], 1,
[3, 4]]
print([0] * 5 + [1] * 3) // expected [0, 0, 0, 0, 0, 1, 1, 1]
print([1,2,3,4,5,6,7][2:6]) // has slice, expected [3, 4, 5, 6]
print(([1,2,3,4,5,6,7] + [6] * 5)[2:]) // has slice, expected [3, 4, 5, 6,
7, 6, 6, 6, 6, 6]
print([1, 2, 3, 4, 5, 6][0:2:3]) // has slice, expected [1]
```

python list說明

List 是 Python 中最基本的數據結構。List 中的每個 Item 都分配一個數字 - 它的位置，或索引，第一個 Index 是0，第二個 Index 是1，依此類推。
List 都可以進行的操作包括切片，加，乘。

此題使用的名詞定義

1. **item**: 指 list 裡面的東西，像是 [1,2,3] 有三個 **item**，分別是數字 1,2,3
2. **slice**: 是 python 獨有的概念，可以優雅地根據原本的list 建立新的 **list**
    1. **startIndex**: 描述開始的位置
    2. **endIndex**: 描述結束的位置(不包含)
    3. **step**: 描述間隔

注意事項:

- 此題需實作的 List 的每一個 **Item** 都是 **integer** ，並且有可能有負號
- 測資只會有符合 Grammar 的輸入，不會有不符合的情況出現
- 參考資料在最下面，請不要忘記去看

加號(binary operator)跟python list的運算

只能 **list** 跟 **list** 做此運算
list1 + **list2**，會建立一個新的 **list**，內容為 **list1** 和 **list2** 相接的結果 (copy by value)

範例一

```
# 輸入：
[1, 2, 3, 4] + [5, 6, 7]

# 輸出：
[1, 2, 3, 4, 5, 6, 7]
```

範例二

```
# 輸入：
[1] + []

# 輸出：
[1]
```

範例三

```
# 輸入：
[] + []

# 輸出：
[]
```

乘號(binary operator)跟python list的運算

只能 **list** 跟 **number** 做此運算
list1 * **number**，會建立一個新的 **list**，內容為 **list1** 所有 **item** 重複出現 **number** 次(copy by value)

範例一

```
# 輸入：
[1, 2, 3] * 3

# 輸出：
[1, 2, 3, 1, 2, 3, 1, 2, 3]
```

範例二

```
2 * [4, 5, 6]

# 輸出：
[4, 5, 6, 4, 5, 6]
```

範例三

```
# 輸入：
[1] * 6

# 輸出：
[1, 1, 1, 1, 1, 1]
```

範例四

```
# 輸入：
6 * []

# 輸出：
[]
```

測資輸出規範

1. 印出 list 時，list 中間的逗號後面要有一個空格，ex: [1, 2, 3]、[2, 3, 4, 5, 6]
2. output 印出 list 時要在結尾印一個換行字元(\n)

題目需要實作的部分

有很多部分已經有現成的程式可以用，只需完成bison/yacc在grammar's action的部分(在reduce時需要做甚麼事)

1. flex/lex的檔案已經Token的定義在此文件最後的區塊 - 「有可以完全參考的程式碼」的list.l檔，可以直接用
2. 文法的定義和list相關的操作函數定義在此文件最後的區塊 - 「有可以完全參考的程式碼」的list.y檔，可以直接用

B 小題 - 處理只有+和*符號的操作(35分)

注意事項:

- 只有 ***** , **+**, 沒有 **slice**

- 可能由有多個運算符號 (*、+) 組成，詳細請看以下測資

下面描述此題的文法，使用的格式可以直接在`bison/yacc`使用

```
S : Sum
Sum : Term ADDITION Sum
     | Term
Term : List MULTIPLY MulDigit
     | MulDigit MULTIPLY List
     | MulDigit MULTIPLY List MULTIPLY MulDigit
     | List
MulDigit : MulDigit MULTIPLY  DIGITS
         |  DIGITS
List : LBRACKET ListItem RBRACKET
ListItem : DIGITS COMMA ListItem
         | DIGITS
```

下面會列出所有 open test case，並且描述 test case 的特性，hidden test case 一定會符合 open test case 的特性!

1. 一個單純的 list，沒做任何操作
   input: [1,2,3]
   output: [1, 2, 3]
2. 建立一個 list，由兩個 list 串接起來
   input: [1, 2, 3] + [4, 5, 6]
   output: [1, 2, 3, 4, 5, 6]
3. 建立一個 list，由多個 list 串接起來
   input: [1, 2, 3] + [4,5,6] + [2]
   output: [1, 2, 3, 4, 5, 6, 2]
4. 建立一個 list，為某個 list 的 items 重複多次(乘號在後，可能有多個)
   input: [1,2,3] * 2
   output: [1, 2, 3, 1, 2, 3]
5. 建立一個 list，為某個 list 的 items 重複多次(乘號在前，可能有多個)
   input: 2 * [1,2,3]
   output: [1, 2, 3, 1, 2, 3]
6. 建立一個 list，為某個 list 的 items 重複多次(乘號有多個，且在 list 左右兩旁)
   input: 2 * [1,2,3] * 3
   output: [1, 2, 3, 1, 2, 3, 1, 2, 3, 1, 2, 3, 1, 2, 3, 1, 2, 3]
7. 會同時出現乘號和加號
   input: [1,2] * 3 + [3,4] * 2
   output: [1, 2, 1, 2, 1, 2, 3, 4, 3, 4]

---

slice

只能出現在 list 的右邊，用來根據原本的 list，指定 range 和 step 來建立新的 list。

> slice 結構: [**startIndex**:**endIndex**] 或 [**startIndex**:**endIndex**:**step**]

範例一

```
# 輸入：
[1, 2, 3, 4, 5, 6][2:5]

# 輸出：
[3, 4, 5]
```

範例二

```
# 輸入：（這裡 endIndex 超出範圍了，但是還是可以正常印到結束）
[1, 2, 3, 4, 5, 6, 7][2:100]

# 輸出：
[3, 4, 5, 6, 7]
```

範例三

```
# 輸入：（這裡 startIndex 沒有填入，所以會從頭開始到 index 3 ）
[1, 2, 3, 4, 5, 6, 7][:3]

# 輸出：
[1, 2, 3]
```

範例四

```
# 輸入：（這裡 endIndex 沒有填入，所以會從 index 5 開始到結束 ）
[1, 2, 3, 4, 5, 6, 7][5:]

# 輸出：
[6, 7]
```

範例五

```
# 輸入：（這裡 如果startIndex和endIndex為負數就代表倒數第幾個的意思 ）
# 會建立包含 `倒數第4個item` 到 `倒數第2個item`
[0,1,2,3,4][-4:-2]
```

```
# 輸出：
[1,2]
```

範例六

```
# 輸入：（如果有 Step 情況出現）
# 就是從index為 1 的item開始，每次加 2 拿取item建立新的list直到index為 4 (不包含)
[0,1,2,3,4,5][1:4:2]

# 輸出：
[1,2]
```

warning
注意： step 沒有描述的時候，預設值為1(也就是 [1,2,3][1:10] 這種情況)

C小題 - 需要處理slice(15分)

注意事項:

- 因為step為負數會很難處理，因此test case不會有step為負的情況(雖然文法允許)
- 有做 Slice 運算的 TestCase 就不會有其他的 **乘號** 或 **加號** 不過你 Grammar's action部分如果寫的好應該是不用擔心這種情況發生

下面描述此題的文法，使用的格式可以直接在`bison/yacc`使用

```
S : Sum
Sum : Term ADDITION Sum
     | Term
Term : List MULTIPLY MulDigit
     | MulDigit MULTIPLY List
     | MulDigit MULTIPLY List MULTIPLY MulDigit
     | List
MulDigit : MulDigit MULTIPLY  DIGITS
         |  DIGITS
List : LBRACKET ListItem RBRACKET Slice
Slice: LBRACKET StartIndex COLON EndIndex RBRACKET
     | LBRACKET StartIndex COLON EndIndex COLON Step RBRACKET
     |
StartIndex: DIGITS
          |
EndIndex: DIGITS
          |
Step:     DIGITS
          |
ListItem : DIGITS COMMA ListItem
         | DIGITS
```

下面會列出所有open test case，並且描述 test case 的特性，hidden test case 一定會符合 open test case 的特性!

1. 一個單純的 list，startIndex 和 endIndex 都>=0
   input: [1, 2, 3, 4, 5, 6][0:4]
   output: [1, 2, 3, 4]
2. startIndex 和 endIndex 都是負數
   input: [1, 2, 3, 4, 5, 6][-4:-2]
   output: [3, 4]
3. 沒有描述 startIndex
   input: [1, 2, 3, 4, 5, 6][:4]
   output: [1, 2, 3, 4]
4. 沒有描述 endIndex
   input: [1, 2, 3, 4, 5, 6][1:]
   output: [2, 3, 4, 5, 6]
5. 有定義 step，且 step 為正數
   input: [1, 2, 3, 4, 5, 6][1:5:2]
   output: [2, 4]

有可以完全參考的程式碼

檔案說明

下載網址

1. 公開測資
    1. all_open.txt 列出所有的 open test cases
    2. all_open_ans.txt 列出所有的 open test cases 的正確 output
2. lex 與 yacc 檔案
    1. lex 的檔案為 list.l，定義如何匹配 token 與將值存到 token 中(DIGITS token)
    2. yacc 的檔案為 list.y，定義 list 和 slice 的 struct，並且定義一些操作 list 的函數，以及描述 token 和 NonTerminal 儲存資料的型別，並且完整定義文法，只差 action 部份沒有實質的功能（只有印出訊息，讓人比較好知道 reduce 情況與順序)
3. parser 的 input
    1. list.y 在 main 函數那邊要特別注意，如果想要在自己電腦測試，有提供從檔案讀入 input，預設是讀入 t.txt，如果要上傳到 domJudge 要把那段設定 bison/yacc input 的 code 註解掉
4. list 相關的函數的使用範例有寫在 test.c，如果要用可以自己玩玩看

compile and run

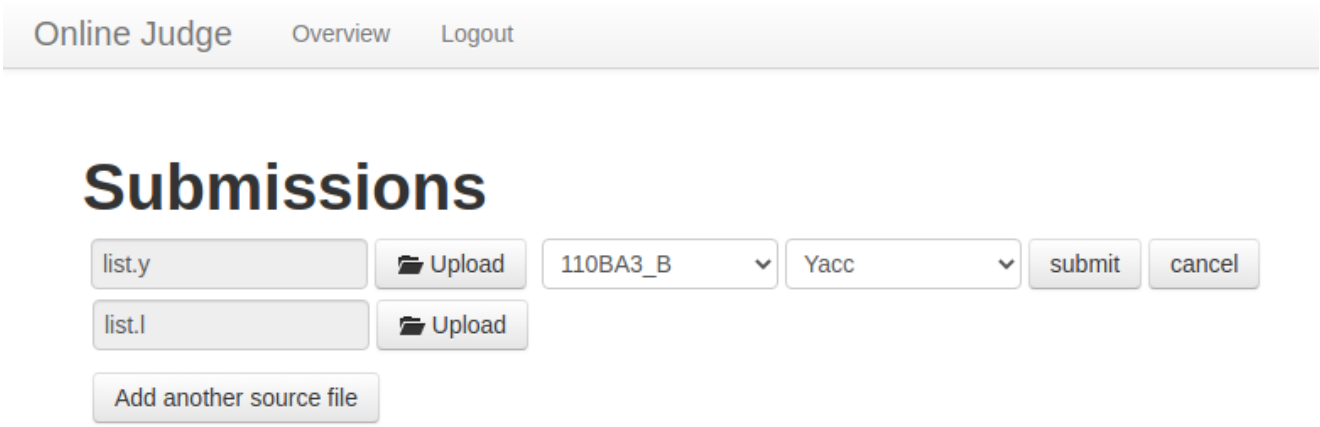
有提供 Makefile 方便編譯，指令如下
generate parser named a.out

```
make all
```

run parser

```
./a.out
```

繳交檔案的方式



先交list.y，再交list.l，題目是選自己想選B/C，最後選擇Yacc