COMPILER FINAL PROJECT

學號：107502018
姓名：王昱承
系級：資工3B

Syntax Validation

當遇到語句錯誤、輸入錯誤時輸出，並給予特定警告

```
user@ubuntu:~/Desktop/HW$ ./final < input.txt
syntax error , no number
```

<=輸入運算子但沒給數值

```
user@ubuntu:~/Desktop/HW$ ./final < input.txt
syntax error
```

<=輸入運算但不符合規則

```
user@ubuntu:~/Desktop/HW$ ./final < input.txt
haven't difine fun yet
```

<=呼叫函式但尚未創建

⋮ ⋮

HOW TO COUNT

使用node來作樹連接，建完樹在
進行運算

```
struct Node *initial;//the first node
struct Node *Gpar = NULL;//used to point the node
struct Node *newnode;
```

主要運用這三者來進行指標與建樹

```
void createnode(char* type,int num){//create the node and give its type and number
  if(strcmp(type,"var"))newnode=malloc(sizeof(struct Node));//if is "var" ,dont malloc,else create
  /* printf("Gpar 的位址：%p\n", Gpar);
  printf("Gpar numeber %d\n", Gpar->number);
  printf("the newnode(create) 的位址：%p\n", newnode); */
  newnode->work = type;
  /* printf("the newnode(create) type is %s \n",newnode->work); */
  newnode->number = num;
  /* printf("the newnode(create) num is %d \n",newnode->number); */
  newnode->Childs = -1;
  Puttostack();

  //give the node its type
  if(!strcmp(type,"num")||!strcmp(type,"boolean"))newnode->type = type;
}
```

用來創建新節點

```
void setnode(char* work){//give the node its work
 if(strcmp(newnode->work,"("){
   printf("syntax error\n");
   yyerror("");
 }
 /* printf("newnode 的位址：%p\n", newnode); */

 newnode->work = work;//replace "(" to operator
 /* printf("the newnode work is %s\n",newnode->work);  */
 /* printf("the parent of Gpar %p\n",Gpar->Par); */
 Gpar=newnode;//child to parent
 newnode= NULL;//child release
 /* printf("Gpar 的位址：%p\n", Gpar); */
 int i;
 /* printf("the childs of Gpar is %d\n",Gpar->Childs); */
 //set the node's type
 if(!strcmp(work,"and")||!strcmp(work,"or")||!strcmp(work,"not")||!strcmp(work,">")||!strcmp(work,"<")||!
   Gpar->type="bool";
 }
 else if(!strcmp(work,"+")||!strcmp(work,"-")||!strcmp(work,"*")||!strcmp(work,"/")){
   Gpar->type="num";
 }
 //set the child max size
 if(!strcmp(work,"not"))Gpar->MChilds=1;
 else if (!strcmp(work,"-")||!strcmp(work,"/")||!strcmp(work,"mod")||!strcmp(work,">")||!strcmp(work,"<")
 else if (!strcmp(work,"+")||!strcmp(work,"*")||!strcmp(work,"=")||!strcmp(work,"and")||!strcmp(work,"or"
```

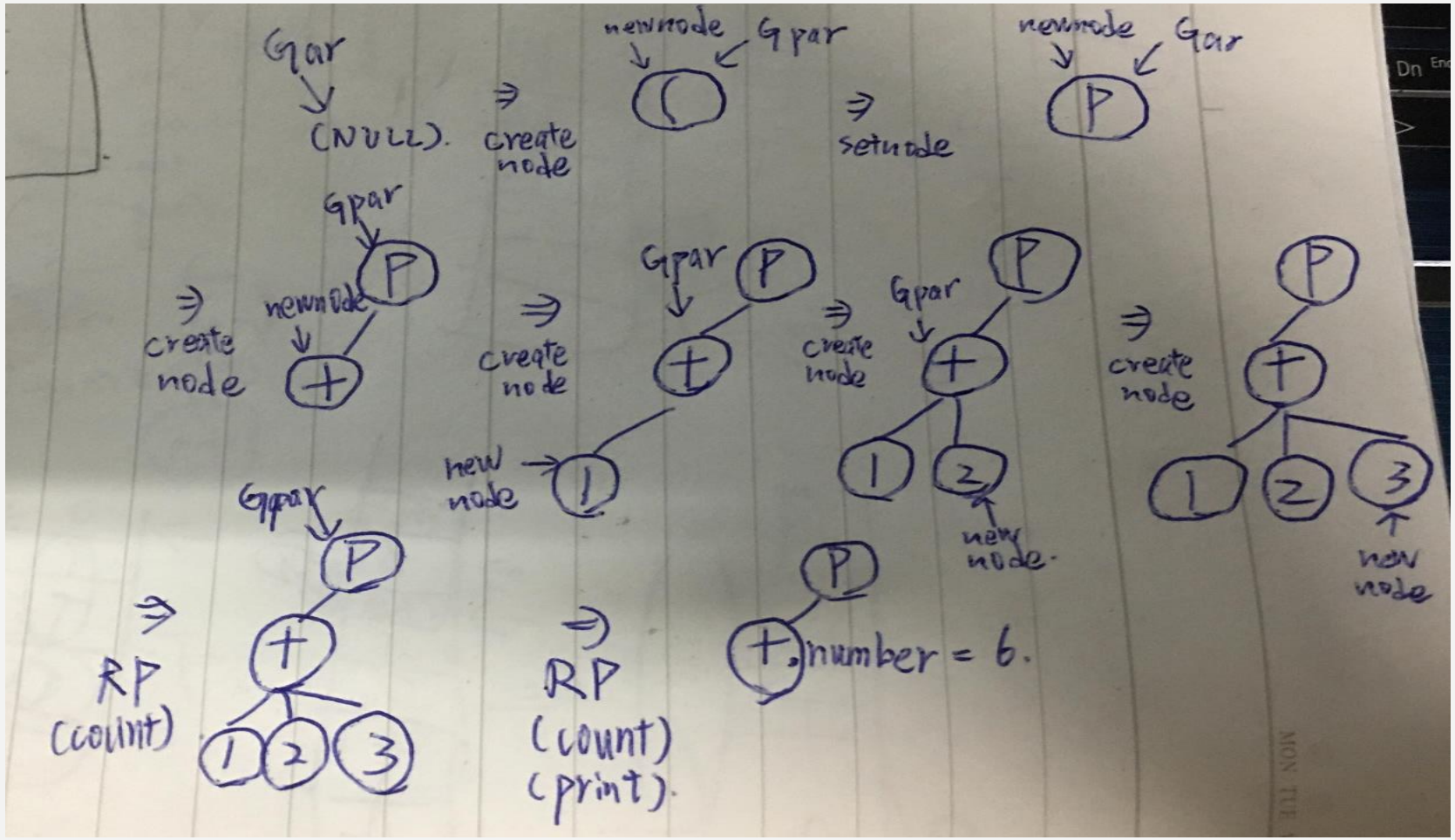
設置節點(因為遇左括弧會生成新節點，所以直接換掉即可)以及將Gpar下移至newnode。

```
void Puttostack(){//put in the parent's childs stack then connect
    Gpar->Childs=Gpar->Childs+1;
    /* printf("the childsize of Gpar is %d \n",Gpar->Childs); */
    Gpar->Child[Gpar->Childs] = newnode;
    /* printf("the Child[%d] have new node\n",Gpar->Childs); */
    newnode->Par=Gpar;
}
```

新增子節點，並新節點與Gpar所在節點連接。

HOW TO COUNT

EX : (print-num (+ 1 2 3))



PRINT

當遇到“print-num”或是 “print-bool”時，根據要輸出數字或布林值做輸出，否則省略

```
user@ubuntu:~/Desktop/HW$ ./final < input.txt
-> 0
-> -123
-> 456
```

<=

```
(+ 1 2 3)
(* 4 5 6)
(print-num 0)
(print-num -123)
(print-num 456)
```

```
user@ubuntu:~/Desktop/HW$ ./final < input.txt
-> #t
-> #f
```

<=

```
(print-bool #t)
(print-bool #f)
```

NUMERICAL & LOGICAL OPERATIONS

一般的數字與邏輯計算，其中{ +,*,=,and ,or}可以輸入多筆資料，而{-,/,mod,>,<}則只能輸入兩筆,最後{not}只能輸入一筆，否則輸出錯誤訊息。

```
user@ubuntu:~/Desktop/HW$ ./final < input.txt
-> 133
-> #f
```

<=

```
(print-num (+ 1 (+ 2 3 4)
    (* 4 5 6) (/ 8 3) (mod 10 3)))

(print-bool (and #t (not #f)
    (or #f #t) (and #t (not #t))))
```

```
user@ubuntu:~/Desktop/HW$ ./final < input.txt
syntax error, too many for not
```

<=

```
(print-bool (not #t #t #f))
(print-num (-256 4 7))
```

IF EXPRESSION

當遇到關鍵字“if”時，先判斷，判斷後ture=>print前面、false=>print後面，這三項也可以式數學運算式

```
user@ubuntu:~/Desktop/HW$ ./final < input.txt
-> 1
-> 2
```

<=

```
(print-num (if #t 1 2))

(print-num (if #f 1 2))
```

```
-> 2
user@ubuntu:~/Desktop/HW$ ./final < input.txt
-> 6
```

<=

```
(print-num (if (< 1 2)
    (+ 1 2 3) (* 1 2 3 4 5)))
```

HOW TO COUNT(2)

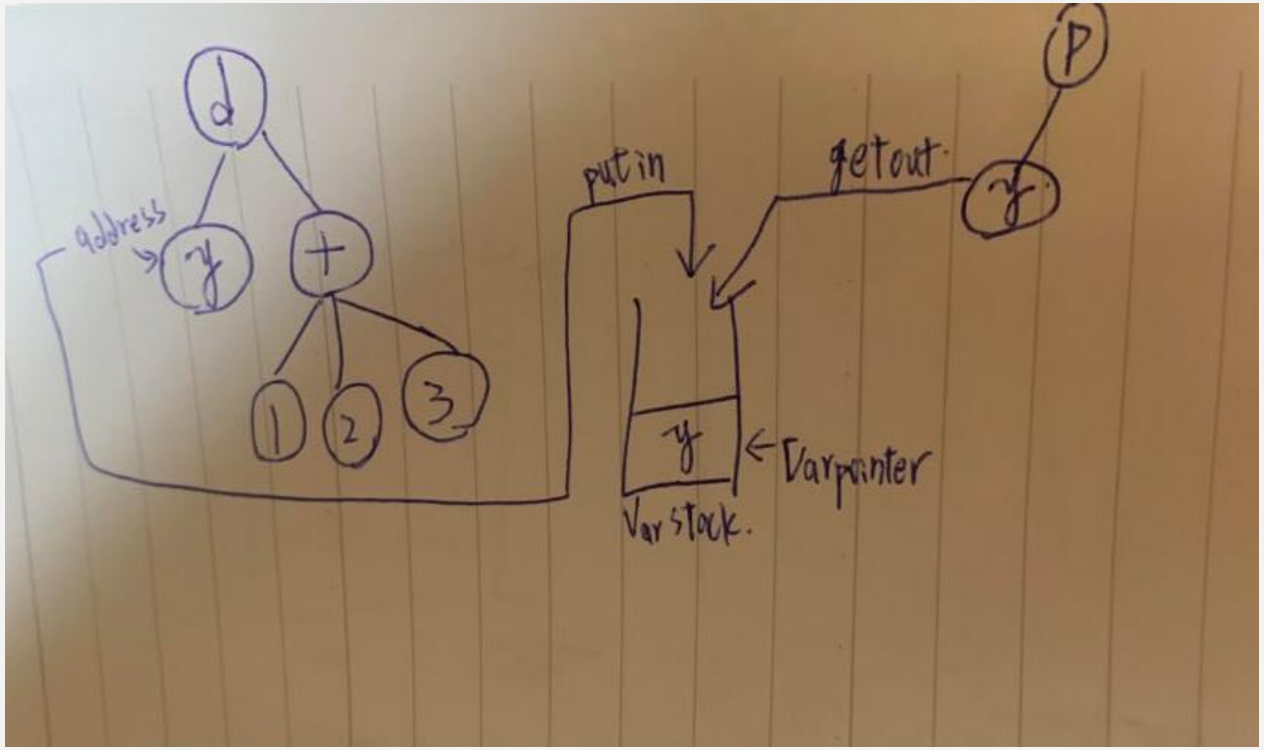
```
struct Node *Varstack[100];//to store the Var and Funname
int Varpointer=-1;
```

```
if(exist == 0){// if not exist,create new node and put address in Var
  newnode=malloc(sizeof(struct Node));
  newnode->var = $1;
  newnode->work = "var";
  // printf("the var is %s\n",newnode->var);
  // printf("the node address is %p\n",newnode);
  // printf("the newnode work is %s\n",newnode->work);
  newnode->Childs = -1;
  Varstack[++Varpointer]= newnode;
  Gpar->Childs=Gpar->Childs+1;
  // printf("the childsize of Gpar is %d \n",Gpar->Childs);
  Gpar->Child[Gpar->Childs] = newnode;
  // printf("the Child[%d] have new node\n",Gpar->Childs);
  newnode->Par=Gpar;
}else{//for fun call
 for(i=0;i<=Varpointer;i++){
  if(!strcmp(Varstack[i]->var,$1)){//if the var is in stack, connect
    // printf("hehehe %s\n",Varstack[i]->var);
    // printf("hehehe %p\n",Varstack[i]);
    // printf("ohohoh %s\n",$1);
    // printf("the Gpar %p\n",Gpar);
    // printf("the Gpar's child %p\n",Gpar->Child[0]);
    Gpar->Childs=Gpar->Childs+1;
    Gpar->Child[Gpar->Childs] = Varstack[i];
  }
}
```

使用Varstack 存取所有變數值“id”，之後遇到，使用Varpointer去找是否在stack之中

如果不存在就創一個到stack之中，否則降找到的var的位置丟給新的Gpar作為子節點

EX : (define y (+ 1 2 3))
(print-num y)



Variable define

先define一個變數後，可在另一個tree呼叫並給值，且可以define多個變數並將其相加

```
[[Auser@ubuntu:~/Desktop/./final < input.txt
> 1
> 6
```

<=

```
(define x 1)

(print-num x)

(define y (+ 1 2 3))

(print-num y)
```

```
user@ubuntu:~/Desktop/HW$ ./final < input.txt
-> 26
```

<=

```
(define a (* 1 2 3 4))

(define b (+ 10 -5 -2 -1))

(print-num (+ a b))
```

FUNCTION

類似define，只是是用將要用的變數名稱與要給的值放在一起，但是同一變數名稱在fun外與內，不可共用，另外如果遇到fun() =>直接回傳數字

```
user@ubuntu:~/Desktop/HW$ ./final < input.txt
-> 4
```

<=

```
(print-num
  ((fun (x) (+ x 1)) 3))
```

```
user@ubuntu:~/Desktop/HW$ ./final < input.txt
-> 610
-> 0
```

<=

```
(define x 0)

(print-num
  ((fun (x y z) (+ x (* y z))) 10 20 30))


(print-num x)
```

NAMED FUNCTION

將fun進行define命名，在”id”中判斷是fun變數或是單純define的變數，且可以呼叫多個fun，在“RP”中判斷是一般call fun 或是連續call fun

```
user@ubuntu:~/Desktop/HW$ ./final < input.txt
-> 91
```

<=

```
(define foo
  (fun (a b c) (+ a b (* b c))))

(print-num (foo 10 9 8))
```

```
user@ubuntu:~/Desktop/HW$ ./final < input.txt
-> 3
```

<=

```
(define bar (fun (x) (+ x 1)))

(define bar-z (fun () 2))

(print-num (bar (bar-z)))
```

Thanks for Your listening