

# Compiler Project Phase3

---

## Workload Division:

**贺晗 12012013:**

Function Invocation, Array Initialization and accessing

**郑鑫颖 11912039:**

Translation of the conditional Expression.

**张海涵 12012222:**

Translation of the basic Expressions.

## Design:

```
we implement several translation functions.
- char* new_place();
- char* new_label();
- char* translate_cond_Exp(node* Exp, char* lb1, char* lb2);
- char* translate_stmt(node* stmt);
- char* translate_Exp(node* Exp, char* place);
- char* translate_Args(node* Args, int* arglist);
- char* translate_def(node* def);
By invoking this function recursively, we can translate the code into TAC.
```

## Optimization:

We adopt 2 strategies to decrease the number of instructions.

### #1 cut one register when using ID

```
int main()
{
    int a, b, c;
    int final = 0;
    a = 5;
    b = a * a * (a + 2);
    write(b);
    c = b / a + 1;
    write(c);
    final = a + b - c * 3 + (b / a - 4);
    write(final);
    return 0;
}
```

phase3 > out.txt

```
1 FUNCTION main :  
2 t5 := #0  
3 t4 := t5  
4 t7 := #5  
5 t1 := t7  
6 t12 := t1  
7 t13 := t1  
8 t10 := t12 * t13  
9 t14 := t1  
10 t15 := #2  
11 t11 := t14 + t15  
12 t9 := t10 * t11
```

phase3 > out1.txt

```
1 FUNCTION main :  
2 t4 := #0  
3 t1 := #5  
4 t10 := t1 * t1  
5 t11 := t1 + #2  
6 t9 := t10 * t11  
7 t2 := t9  
8 WRITE t2  
9 t20 := t2 / t1  
10 t19 := t20 + #1  
11 t3 := t19  
12 WRITE t3
```

## #2 cut one register when using INT

FUNCTION main :

t5 := #0

t4 := t5

t7 := #5

t1 := t7

1 FUNCTION main :

t4 := #0

3 t1 := #5

4 t10 := t1 \* t1

5 t11 := t1 + #2

## Total # of Instructions before and after optimization

	Before	After
test_3_r01.spl	41	21
test_3_r02.spl	147	98
test_3_r03.spl	61	37
test_3_r04.spl	33	22
test_3_r05.spl	70	44
test_3_r06.spl	63	41
test_3_r07.spl	115	75
test_3_r08.spl	53	38
test_3_r09.spl	188	117
test_3_r10.spl	28	20

## For executed # instruction

test\_3\_r09.spl

before:

```
[program output] 370  
[program output] 371  
[program output] 407  
[program output] 3  
[INFO] Total instructions = 114229
```

after:

```
[program output] 370
[program output] 371
[program output] 407
[program output] 3
[INFO] Total instructions = 61214
```

## Bonus:

### 1. We implement initializing and accessing 1-dim and 2-dim arrays.

The screenshot displays the SUSTech-CS323 IR-Simulator interface. At the top, two code editors are visible. The left editor, titled 'out.txt', contains the following assembly-like code:

```
1 FUNCTION main :
2 DEC t1 16
3 t5 := #1 * #4
4 t3 := &t1 + t5
5 *t3 := #6
6 t10 := #1 * #4
7 t11 := &t1 + t10
8 t8 := *t11
9 WRITE t8
10 RETURN #0
11
```

The right editor, titled 'test.spl', contains the following C-like code:

```
1 int main()
2 {
3     int b[4];
4     b[1]= 6;
5     write(b[1]);
6     return 0;
7 }
```

Below the code editors, the 'TERMINAL' tab is active, showing the message: 'File "urwid/wimp.py", line 540, in keypress'.

The main interface is divided into two panels. The left panel, titled 'CODE', shows the assembly code from the 'out.txt' file. The right panel, titled 'SYMBOLS', displays a table of variables and their values:

Variable	Value
t1	[0, 6, 0, 0]
t10	4
t11	4
t3	4
t5	4
t8	6

Below the symbols table, the output is shown:

```
[program output] 6
[INFO] Total instructions = 10
```

At the bottom of the interface, there are two rows of dashed lines for additional output or instructions.

phase3 > out.txt

```
1 FUNCTION main :
2 DEC t1 80
3
4 t5 := #2 * #4
5 t7 := #5
6 t7 := t7 * #4
7 t9 := #1 * t7
8 t5 := t9 + t5
9 t3 := &t1 + t5
10 *t3 := #6
11 t14 := #2 * #4
12 t16 := #5
13 t16 := t16 * #4
14 t18 := #1 * t16
15 t14 := t18 + t14
16 t19 := &t1 + t14
17 t12 := *t19
18 WRITE t12
19 RETURN #0
```

phase3 > sample > test.spl

```
1 int main()
2 {
3     int b[4][5];
4     b[1][2] = 6;
5     write(b[1][2]);
6     return 0;
7 }
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL

SUSTech-CS323 IR-Simulator [out.txt]

CODE

< step > < hex > < stop >

```
t16 := t16 * #4
t18 := #1 * t16
t14 := t18 + t14
t19 := &t1 + t14
t12 := *t19
WRITE t12
@ RETURN #0
```

SYMBOLS

t1		[0, 0, 0, 0, 0, 0, 0, 6, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]
t12		6
t14		28

```
[program output] 6
[INFO] Total instructions = 18
```