Project Phase 3 Report

SID	Name	Rate
11911202	袁恒宸	1/3
12011543	林洁芳	1/3
12011906	汤奕飞	1/3

Basic

In the intermediate representation generation section, we use tac class to generate IR code, including v, t variable and label. Based on semantic analysis, we use tac to connect them in series.

As for tac.h

```
#ifndef _TAC
#define _TAC

typedef struct tac
{
    enum { FUNC = 0, READ, WRITE, LABEL, GOTO, IF, ASS, OPER, RETURN, PARAM,
ARG, DEC} title;
    char* target;
    char* op;
    char* arg1;
    char* arg2;
    struct tac* next;
} Tac;

Tac * newTac(char* target, char* op, char* arg1, char* arg2);

void printTacs(Tac* head);

char* generateV(int v);

char* generateT(int t);

char* generateLabel(int lbl);

#endif
```

For translation.h

```
#ifndef _SEMANTIC_ANALYSIS
#define _SEMANTIC_ANALYSIS

#include "node.h"
#include "type.h"

typedef struct FuncParamLinkNode {
    char* tag;
```

```
    struct FuncParamLinkNode* prev;
    struct FuncParamLinkNode* next;
} FuncParamLinkNode;

typedef struct node *Node;

...

#endif
```

Bonus

multi-dimensional array

in translation.c

```
FieldList *parseVarDec(int isStructDef, int isFuncParam, Node varDec, Type
*type) {
    //VarDec: VarDec LB INT RB (Array)
    //       |ID
    ...
}
```

in test

```
struct Apple
{
    int a;
    int number[10][5][7][10];
};

int add(int a, int b){
    return a + b;
}

struct Apple test(int c, int d)
{
    struct Apple aa;
    aa.a = 15;
    aa.number[0][2][3][4] = add(c, d);
    return aa;
}
```

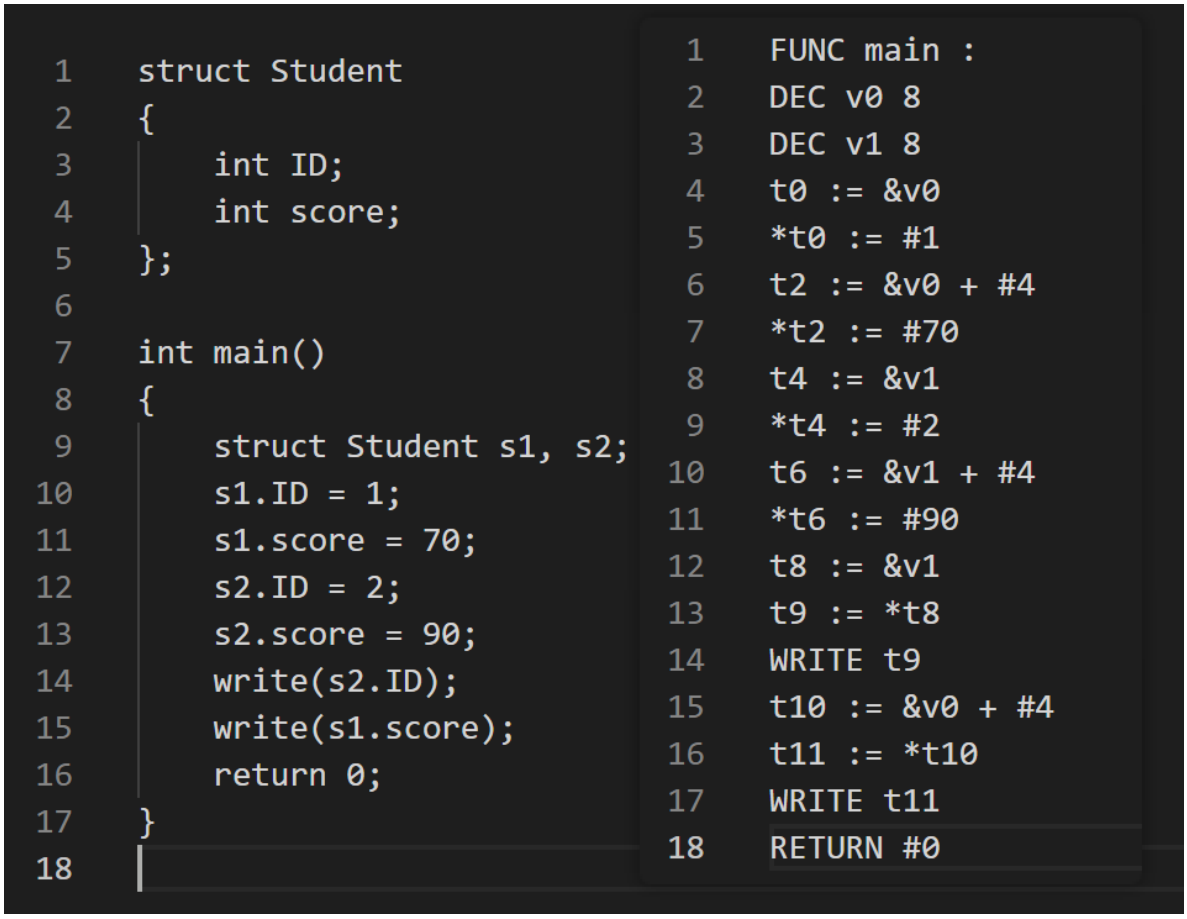
```
FUNC add :
PARAM v0
PARAM v1
t0 := v0 + v1
RETURN t0

FUNC test :
PARAM v2
PARAM v3
DEC v4 14004
t1 := &v4
*t1 := #15
t3 := &v4 + #4
t4 := #2 * #280
t5 := t3 + t4
t6 := #3 * #40
t7 := t5 + t6
t8 := #4 * #4
t9 := t7 + t8
ARG v3
ARG v2
*t9 := CALL add
RETURN &v4
```

structure

in translation.c

```
else if (!strcmp(NDtypes[leftmost->type], "StructSpecifier")) {
    //STRUCT ID
    ...
    //STRUCT ID LC DefList RC
    ...
}
```

in test



```
struct Student
{
    int ID;
    int score;
};

int main()
{
    struct Student s1, s2;
    s1.ID = 1;
    s1.score = 70;
    s2.ID = 2;
    s2.score = 90;
    write(s2.ID);
    write(s1.score);
    return 0;
}
```

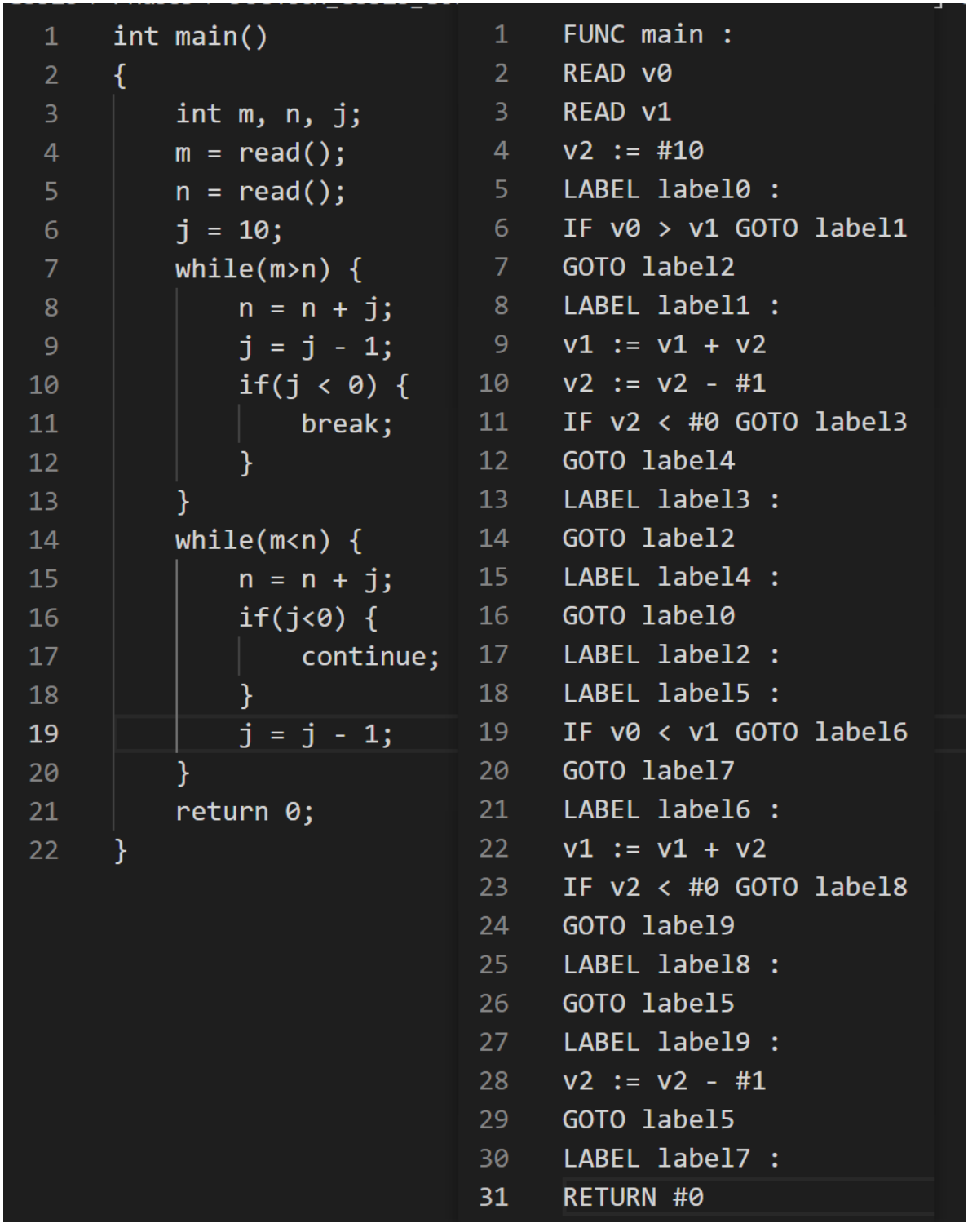
```
FUNC main :
DEC v0 8
DEC v1 8
t0 := &v0
*t0 := #1
t2 := &v0 + #4
*t2 := #70
t4 := &v1
*t4 := #2
t6 := &v1 + #4
*t6 := #90
t8 := &v1
t9 := *t8
WRITE t9
t10 := &v0 + #4
t11 := *t10
WRITE t11
RETURN #0
```

continue and break

in translation.c

```
else if (!strcmp(NDtypes[leftmost->type],"BREAK")){
    //BREAK SEMI
    ...
}
    }else if (!strcmp(NDtypes[leftmost->type],"CONTINUE")){
    //CONTINUE SEMI
    ...
}
```

in test



```
int main()
{
    int m, n, j;
    m = read();
    n = read();
    j = 10;
    while(m>n) {
        n = n + j;
        j = j - 1;
        if(j < 0) {
            break;
        }
    }
    while(m<n) {
        n = n + j;
        if(j<0) {
            continue;
        }
        j = j - 1;
    }
    return 0;
}
```

```
FUNC main :
READ v0
READ v1
v2 := #10
LABEL label0 :
IF v0 > v1 GOTO label1
GOTO label2
LABEL label1 :
v1 := v1 + v2
v2 := v2 - #1
IF v2 < #0 GOTO label3
GOTO label4
LABEL label3 :
GOTO label2
LABEL label4 :
GOTO label0
LABEL label2 :
LABEL label5 :
IF v0 < v1 GOTO label6
GOTO label7
LABEL label6 :
v1 := v1 + v2
IF v2 < #0 GOTO label8
GOTO label9
LABEL label8 :
GOTO label5
LABEL label9 :
v2 := v2 - #1
GOTO label5
LABEL label7 :
RETURN #0
```

--

in translation.c

```
else if (!strcmp(NDtypes[leftmost->type],"DMINUS")) {
    //DMINUS Exp
    ...
}
```

in test

```
int main()
{
    int m, n;
    m = read();
    n = read();
    while(m--) {
        n = n-1;
    }
    return 0;
}
```

```
FUNC main :
READ v0
READ v1
LABEL label0 :
v0 := v0 - #1
IF v0 > #0 GOTO label1
GOTO label2
LABEL label1 :
v1 := v1 - #1
GOTO label0
LABEL label2 :
RETURN #0
```

Ternary operator(...? ...: ...)

in translation.c

```
else if(!strcmp(NDtypes[operator->type],"QM")){
    //Exp QM Exp COLON Exp
    ...
}
```

in test

```
int main()
{
    int m, n, j;
    m = read();
    n = read();
    j = (m==n)? 1: 0;
    return 0;
}
```

```
FUNC main :
READ v0
READ v1
IF v0 == v1 GOTO label0
GOTO label1
LABEL label0 :
t2 := #1
LABEL label1 :
v2 := #0
RETURN #0
```