

Compiler Technology of Programming Languages

Project 4 (Code Generation & Optimization)

系級：電機所碩一 姓名：楊冠彥 學號：R11921091

tags: `Compiler`

Part 1. Code generation

1. Assignment statements

這邊的想法就是寫個genAssign的funciton，負責執行assign的功能，不論是一般宣告，還是陣列宣告，只要ID和leftOp的ID_Kind相等，就可進一步根據leftOp的datatype對rightOp做針對性的暫存器分配，再插入對應assembling language，把相關該指定的值帶入，讓leftOp和rightOp最後指向同個Reg_No，即可完成此小題要求，程式碼如下：

src/codeGen.c

```
1 void genAssign(AST_NODE *assignmentStmtNode){
2     AST_NODE *leftOp = assignmentStmtNode->child;
3     AST_NODE *rightOp = leftOp->rightSibling;
4     genExprRelated(rightOp);
5
6     if (ID_Kind(leftOp) == NORMAL_ID){
7         if (leftOp->dataType == INT_TYPE){
8             char *rightOpRegName = NULL;
9             allocate_reg(INT_REG, rightOp->Reg_No, 1, 0, &rightOpRegName);
10            if (!isGlobalVar(ID_SymTab(leftOp))){
11                emit("str %s, [x29, #0]", rightOpRegName,
12                    ID_SymTab(leftOp)->attribute->offset);
13            }
14            else{
15                int tmp_reg_index = get_reg(INT_REG);
16                char *tmp_reg_name = int64_reg[tmp_reg_index];
17                emit("ldr %s, =g_%s", tmp_reg_name, ID_Name(leftOp));
18                emit("str %s, [%s, #0]", rightOpRegName, tmp_reg_name);
19                free_reg(INT_REG, tmp_reg_index);
20            }
21            leftOp->Reg_No = rightOp->Reg_No;
22        }
23        else if (leftOp->dataType == FLOAT_TYPE){
24            char *rightOpRegName = NULL;
25            allocate_reg(FLOAT_REG, rightOp->Reg_No, 1, 0, &rightOpRegName);
26            if (!isGlobalVar(ID_SymTab(leftOp))){
27                emit("str %s, [x29, #0]", rightOpRegName,
28                    ID_SymTab(leftOp)->attribute->offset);
29            }
30            else{
31                int tmp_reg_index = get_reg(INT_REG);
32                char *tmp_reg_name = int64_reg[tmp_reg_index];
33                emit("ldr %s, =g_%s", tmp_reg_name, ID_Name(leftOp));
```

```

32         emit("str %s, [%s, #0]", rightOpRegName, tmp_reg_name);
33         free_reg(INT_REG, tmp_reg_index);
34     }
35     leftOp->Reg_No = rightOp->Reg_No;
36 }
37 }
38 else if (ID_Kind(leftOp) == ARRAY_ID){
39     int elementAddressRegIndex = genArrayAddr(leftOp);
40
41     char *elementAddressRegName = NULL;
42     allocate_reg(INT_REG, elementAddressRegIndex, 1, 0,
&elementAddressRegName);
43     if (leftOp->dataType == INT_TYPE){
44         char *rightOpRegName = NULL;
45         allocate_reg(INT_REG, rightOp->Reg_No, 1, 1, &rightOpRegName);
46         emit("str %s, [%s, #0]", rightOpRegName, elementAddressRegName);
47
48         leftOp->Reg_No = rightOp->Reg_No;
49     }
50     else if (leftOp->dataType == FLOAT_TYPE){
51         char *rightOpRegName = NULL;
52         allocate_reg(FLOAT_REG, rightOp->Reg_No, 1, 0, &rightOpRegName);
53
54         emit("str %s, [%s, #0]", rightOpRegName, elementAddressRegName);
55
56         leftOp->Reg_No = rightOp->Reg_No;
57     }
58     free_reg(INT_REG, elementAddressRegIndex);
59 }
60 }

```

下面是switch到ASSIGN_STMT的程式碼，基本上就是做完genAssign後把register清掉，才能繼續後續的動作。

src/codeGen.c

```

1  switch (Stmt_Kind(stmtNode)){
2      case ASSIGN_STMT: // 1) Assignment statements
3          genAssign(stmtNode);
4          if (stmtNode->child->dataType == INT_TYPE){
5              free_reg(INT_REG, stmtNode->child->Reg_No);
6          }
7          else if (stmtNode->child->dataType == FLOAT_TYPE){
8              free_reg(FLOAT_REG, stmtNode->child->Reg_No);
9          }
10         break;
11         ...

```

Result

測試 `assign.c` 的結果截圖

```

u16@ubuntu:~/project4/src$ ./run.sh ../test/assign.c
Parsing completed. No errors found.
1
2.000000
3
4.000000
5
55
6.000000
66.000000
7
77
8.000000
88.000000
9
99
10.000000
100.000000
1
2.000000
3
4.000000
5
55
6.000000
66.000000
7
77
8.000000
88.000000
9
99
10.000000
100.000000

```

2. Arithmetic expressions

如下方程式碼，針對 leftOp或rightOp有資料型別是FLOAT_TYPE或 exprNode 資料型別是 INT_TYPE的狀況，根據exprNode指到的semantic_value.exprSemanticValue.op.binaryOp，即value和value之間的Op，switch到ADD、SUB、MUL等等狀況。

src/codeGen.c

```

1 void genExpr(AST_NODE *exprNode)
2 {
3     if (Expr_kind(exprNode) == BINARY_OPERATION)
4     {
5         ...
6         if (leftOp->dataType == FLOAT_TYPE || rightOp->dataType ==
FLOAT_TYPE){
7
8             switch (exprNode->semantic_value.exprSemanticValue.op.binaryOp){
9                 case BINARY_OP_ADD:
10                     exprNode->Reg_No = leftOp->Reg_No;
11                     genTriple(FLOAT_REG, "fadd", exprNode->Reg_No, leftOp-
>Reg_No, rightOp->Reg_No);
12                     break;
13                 case BINARY_OP_SUB:

```

```

14         exprNode->Reg_No = leftOp->Reg_No;
15         genTriple(FLOAT_REG, "fsub", exprNode->Reg_No, leftOp-
>Reg_No, rightOp->Reg_No);
16         break;
17         case BINARY_OP_MUL:
18         exprNode->Reg_No = leftOp->Reg_No;
19         genTriple(FLOAT_REG, "fmul", exprNode->Reg_No, leftOp-
>Reg_No, rightOp->Reg_No);
20         break;
21         case BINARY_OP_DIV:
22         exprNode->Reg_No = leftOp->Reg_No;
23         genTriple(FLOAT_REG, "fdiv", exprNode->Reg_No, leftOp-
>Reg_No, rightOp->Reg_No);
24         break;
25         ...
26     }
27     else if (exprNode->dataType == INT_TYPE){
28         exprNode->Reg_No = leftOp->Reg_No;
29
30         switch (exprNode-
>semantic_value.exprSemanticValue.op.binaryOp)
31         {
32         case BINARY_OP_ADD:
33         genTriple(INT_REG, "add", exprNode->Reg_No, leftOp-
>Reg_No, rightOp->Reg_No);
34         break;
35         case BINARY_OP_SUB:
36         genTriple(INT_REG, "sub", exprNode->Reg_No, leftOp-
>Reg_No, rightOp->Reg_No);
37         break;
38         case BINARY_OP_MUL:
39         genTriple(INT_REG, "mul", exprNode->Reg_No, leftOp-
>Reg_No, rightOp->Reg_No);
40         break;
41         case BINARY_OP_DIV:
42         genTriple(INT_REG, "sdiv", exprNode->Reg_No, leftOp-
>Reg_No, rightOp->Reg_No);
43         break;
44         ...
45     }
46 } //endif BINARY_OPERATION
47 ...

```

這裡要說明一下上面每個case中呼叫的genTriple function，此function主要就是在分配暫存器，並將這些部分插入目前位置，並進行儲存到記憶體的动作。這裡的目前位置會是output.s中的程式碼位置，即寫到了第幾行。後面提到目前位置，也是此意思。

```

1 void genTriple(reg_type RegType, char *instruction, int reg1Index, int
reg2Index, int reg3Index)
2 {
3     char *reg1Name = NULL;
4     allocate_reg(RegType, reg1Index, 0, 0, &reg1Name);
5
6     char *reg2Name = NULL;
7     allocate_reg(RegType, reg2Index, 1, 0, &reg2Name);
8
9     char *reg3Name = NULL;

```

```

10     allocate_reg(RegType, reg3Index, 1, 1, &reg3Name);
11
12     emit("%s %s, %s, %s", instruction, reg1Name, reg2Name, reg3Name);
13
14     memory_store(RegType, reg1Index, reg1Name);
15 }

```

Result

測試 `expr.c` 的結果截圖

```

u16@ubuntu:~/project4/src$ ./run.sh ../test/expr.c
Parsing completed. No errors found.
8
-15
7.200000
-15.000000
0
1
1
1
1
1
0
0
1
1

```

3. Control statements: while, if-then-else

首先，while部分，我寫了一個 `genwhile` 的function，其中 `emit("_whileTestLabel_%d:", while_label);` 是插入 `_whileTestLabel_%d` 這個function call 到目前位置。後面看到類似格式也都是這樣。

`src/codeGen.c`

```

1  void genwhile(AST_NODE *whileNode)
2  {
3      AST_NODE *cond = whileNode->child;
4
5      int constantZeroLabelNumber = -1;
6      if (cond->dataType == FLOAT_TYPE)
7      {
8          float zero = 0.0f;
9          constantZeroLabelNumber = genConstant(FLOATC, &zero);
10     }
11
12     int while_label = while_label_no++;
13     emit("_whileTestLabel_%d:", while_label);
14
15     genTest(cond);
16
17     if (cond->dataType == INT_TYPE)
18     {
19         char *boolRegName = NULL;
20         allocate_reg(INT_REG, cond->Reg_No, 1, 0, &boolRegName);

```

```

21     emit("cmp %s, #0", boolRegName);
22     emit("beq _whileExitLabel_%d", while_label);
23     free_reg(INT_REG, cond->Reg_No);
24 }
25 else if (cond->dataType == FLOAT_TYPE)
26 {
27     emit("ldr %s, _CONSTANT_%d", float_reg_rest[0],
constantZeroLabelNumber);
28     char *boolRegName = NULL;
29     allocate_reg(FLOAT_REG, cond->Reg_No, 1, 1, &boolRegName);
30     emit("fcmp %s, %s\n", boolRegName, float_reg_rest[0]);
31     emit("beq _whileExitLabel_%d", while_label);
32     free_reg(FLOAT_REG, cond->Reg_No);
33 }
34
35 AST_NODE *bodyNode = cond->rightsibling;
36 gen_stmt(bodyNode);
37
38 emit("b _whileTestLabel_%d", while_label);
39 emit("_whileExitLabel_%d:", while_label);
40 }

```

if部分，我寫了一個 `genIf` 的function，同樣進行插入assembling language和插入function call到 `output.s` 目前位置的動作，值得注意的是我原本忘記寫14、15行，所以如果判斷式內的左右兩邊 `datatype` 不同就會導致超出預期的狀況，補上判斷，並在 `condNode->dataType == FLOAT_TYPE` 時，進行將float轉int的動作，即 `fcvtzs w%s, s%s`。

src/codeGen.c

```

1 void genIf(AST_NODE *ifNode)
2 {
3     AST_NODE *condNode = ifNode->child;
4     AST_NODE *thenNode = condNode->rightsibling;
5     AST_NODE *elseNode = thenNode->rightsibling;
6     int if_label = if_label_no++;
7
8     emit("_IF_%d:", if_label);
9
10    genTest(condNode);
11
12    char *condRegName = NULL;
13    allocate_reg(INT_REG, condNode->Reg_No, 1, 0, &condRegName);
14    if (condNode->dataType == FLOAT_TYPE)
15        emit("fcvtzs w%s, s%s", condRegName + 1, condRegName + 1);
16
17    emit("cmp %s, #0", condRegName);
18    emit("beq _elseLabel_%d", if_label);
19    free_reg(INT_REG, condNode->Reg_No);
20    emit("beq _elseLabel_%d", if_label);
21
22    gen_stmt(thenNode);
23
24    emit("b _ifExitLabel_%d", if_label);
25    emit("_elseLabel_%d:", if_label);
26
27    gen_stmt(elseNode);
28

```

```

29     emit("_ifExitLabel_%d:", if_label);
30 }
31

```

下面是switch到WHILE_STMT及IF_STMT的程式碼，基本上就是直接call function。

src/codeGen.c

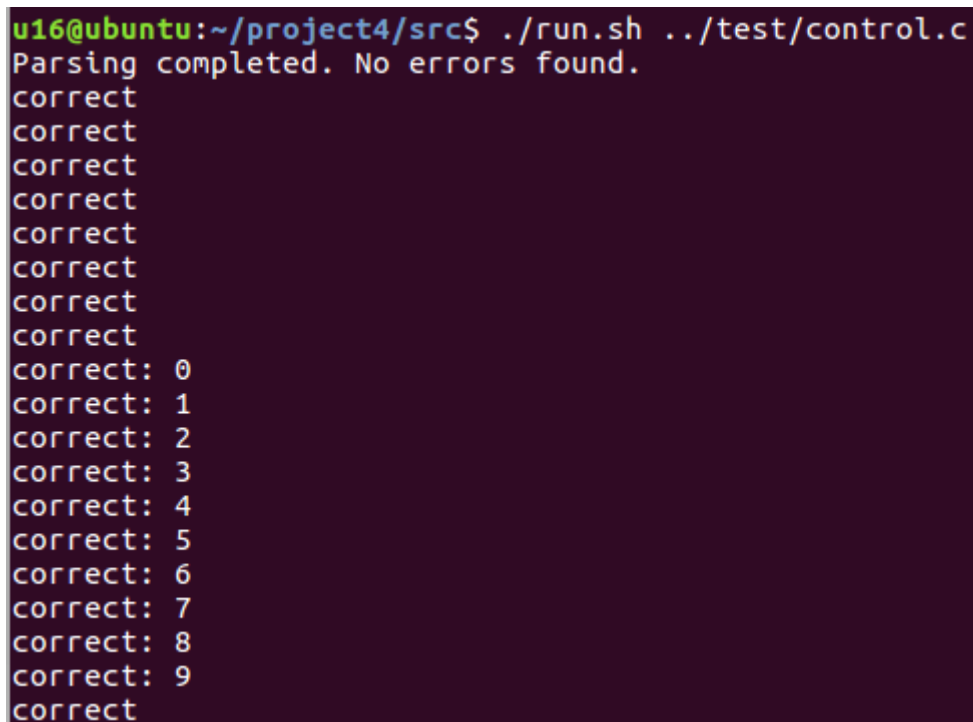
```

1  switch (Stmt_Kind(stmtNode)){
2      ...
3      case WHILE_STMT: // 3) Control statements: while, if-then-else
4          genWhile(stmtNode);
5          break;
6      case IF_STMT: // 3) Control statements: while, if-then-else
7          genIf(stmtNode);
8          break;
9      ...

```

Result

測試 `control.c` 的結果截圖



```

u16@ubuntu:~/project4/src$ ./run.sh ../test/control.c
Parsing completed. No errors found.
correct
correct
correct
correct
correct
correct
correct
correct
correct
correct
correct: 0
correct: 1
correct: 2
correct: 3
correct: 4
correct: 5
correct: 6
correct: 7
correct: 8
correct: 9
correct

```

4. Parameterless procedure calls

同樣針對不同的datatype，我分配暫存器，並插入對應不同的assembling language及不同的function call到output.s目前位置，細部 `genFuncCall` 的程式碼如下：

src/codeGen.c

```

1  void genFuncCall(AST_NODE *functionCallNode)
2  {
3      AST_NODE *functionIdNode = functionCallNode->child;
4      AST_NODE *parameterList = functionIdNode->rightsibling;

```

```

5     if (strcmp(ID_Name(functionIdNode), "write") == 0)
6     {
7         AST_NODE *firstParameter = parameterList->child;
8         genExprRelated(firstParameter);
9         char *parameterRegName = NULL;
10        switch (firstParameter->dataType)
11        {
12            case INT_TYPE:
13                allocate_reg(INT_REG, firstParameter->Reg_No, 1, 0,
14                &parameterRegName);
15                emit("mov w0, %s", parameterRegName);
16                emit("bl _write_int");
17                free_reg(INT_REG, firstParameter->Reg_No);
18                break;
19            case FLOAT_TYPE:
20                allocate_reg(FLOAT_REG, firstParameter->Reg_No, 1, 0,
21                &parameterRegName);
22                emit("fmov s0, %s", parameterRegName);
23                emit("bl _write_float");
24                free_reg(FLOAT_REG, firstParameter->Reg_No);
25                break;
26            case CONST_STRING_TYPE:
27                allocate_reg(INT_REG, firstParameter->Reg_No, 1, 0,
28                &parameterRegName);
29                emit("mov x0, %s", parameterRegName);
30                emit("bl _write_str");
31                free_reg(INT_REG, firstParameter->Reg_No);
32                break;
33            default:
34                printf("error in gen function call\n");
35                printf("firstParameter->Reg_No not free\n");
36                break;
37        }
38        return;
39    }
40
41    if (strcmp(ID_Name(functionIdNode), "read") == 0)
42    {
43        emit("bl _read_int");
44    }
45    else if (strcmp(ID_Name(functionIdNode), "fread") == 0)
46    {
47        emit("bl _read_float");
48    }
49    else
50    {
51        emit("bl _start_%s", ID_Name(functionIdNode));
52    }
53
54    if (ID_SymTab(functionIdNode))
55    {
56        if (ID_SymTab(functionIdNode)->attribute->attr.functionSignature-
57        >returnType == INT_TYPE)
58        {
59            functionCallNode->Reg_No = get_reg(INT_REG);
60            char *returnIntRegName = NULL;
61            allocate_reg(INT_REG, functionCallNode->Reg_No, 0, 0,
62            &returnIntRegName);

```



```

58
59         emit("mov %s, w0", returnIntRegName);
60
61         memory_store(INT_REG, functionCallNode->Reg_No,
returnIntRegName);
62     }
63     else if (ID_SymTab(functionIdNode)->attribute-
>attr.functionSignature->returnType == FLOAT_TYPE)
64     {
65         functionCallNode->Reg_No = get_reg(FLOAT_REG);
66         char *returnfloatRegName = NULL;
67         allocate_reg(FLOAT_REG, functionCallNode->Reg_No, 0, 0,
&returnfloatRegName);
68
69         emit("fmov %s, s0", returnfloatRegName);
70
71         memory_store(INT_REG, functionCallNode->Reg_No,
returnfloatRegName);
72     }
73 }
74 }

```

下面是switch到FUNCTION_CALL_STMT的程式碼，基本上就是做完 genFuncCall 後把register清掉，才能繼續後續的動作。

src/codeGen.c

```

1  switch (Stmt_kind(stmtNode)){
2      case FUNCTION_CALL_STMT:
3          genFuncCall(stmtNode);
4          if (stmtNode->Reg_No != -1){
5              if (stmtNode->dataType == INT_TYPE){
6                  free_reg(INT_REG, stmtNode->Reg_No);
7              }
8              else if (stmtNode->dataType == FLOAT_TYPE){
9                  free_reg(FLOAT_REG, stmtNode->Reg_No);
10             }
11         }
12         break;

```

Result

測試 func.c 的結果截圖

```

1
u16@ubuntu:~/project4/src$ ./run.sh ../test/func.c
Parsing completed. No errors found.
0
1
2
3
4
5
6
7
8
9

```

5. Read and Write I/O calls (See Appendix I)

Read 和 Write 將被translate成extenal function call。

Read部分，見下方 `codeGen.c` 程式碼第3~22行，根據datatype先分配暫存器，再將暫存器的某個值傳到暫存器%s，即parameterRegName，再於目前位置插入特定的function call，如case INT_TYPE插入 `b1 _write_int`。

Write部分，當functionIdNode等於"read"字串則在目前位置插入 `b1_read_int`；當functionIdNode等於"fread"字串則插入 `b1 _read_float`；否則插入 `b1 _start_%s`，%s會是test_file的function call。

src/codeGen.c

```
1 void genFuncCall(AST_NODE *functionCallNode){
2     ...
3     switch (firstParameter->dataType)
4     {
5         case INT_TYPE:
6             allocate_reg(INT_REG, firstParameter->Reg_No, 1, 0,
7 &parameterRegName);
8             emit("mov w0, %s", parameterRegName);
9             emit("b1 _write_int");
10            free_reg(INT_REG, firstParameter->Reg_No);
11            break;
12        case FLOAT_TYPE:
13            allocate_reg(FLOAT_REG, firstParameter->Reg_No, 1, 0,
14 &parameterRegName);
15            emit("fmov s0, %s", parameterRegName);
16            emit("b1 _write_float");
17            free_reg(FLOAT_REG, firstParameter->Reg_No);
18            break;
19        case CONST_STRING_TYPE:
20            allocate_reg(INT_REG, firstParameter->Reg_No, 1, 0,
21 &parameterRegName);
22            emit("mov x0, %s", parameterRegName);
23            emit("b1 _write_str");
24            free_reg(INT_REG, firstParameter->Reg_No);
25            break;
26        default:
27            printf("error in gen function call\n");
28            printf("firstParameter->Reg_No not free\n");
29            break;
30    }
31    return;
32 }
33 ...
34 if (strcmp(ID_Name(functionIdNode), "read") == 0){
35     emit("b1 _read_int");
36 }
37 else if (strcmp(ID_Name(functionIdNode), "fread") == 0){
38     emit("b1 _read_float");
39 }
40 else{
41     emit("b1 _start_%s", ID_Name(functionIdNode));
42 }
43 }
```

Result

測試 `io.c` 的結果截圖

```
u16@ubuntu:~/project4/src$ ./run.sh ../test/io.c
Parsing completed. No errors found.
input:5
input:6
input:7
input:8
5
6
7.000000
8.000000
```

Bonus:

6. Short-circuit boolean expressions

Short-circuit boolean就是 and 和 or boolean判斷，所以就先寫個`genLogical`，這邊的動作就是先將`registerIndex`、`RegType`等參數餵進去，再判斷`RegType`是`INT_REG`還是`FLOAT_REG`，`FLOAT_REG`的話呼叫`genBoolfromFloat`將`Float` 轉成 `boolean`；`INT_REG`的話則分別呼叫`codeGen1Reg1ImmInstruction`和`codeGenSetReg_cond`，`codeGen1Reg1ImmInstruction`做的就是分配暫存器然後將`assembling language`寫到`output.s`再存到記憶體，其中`"cmp"`是算數處理指令，用於把一個暫存器的內容和另一個暫存器的內容或立即數進行減法比較，不存儲結果，會更改`Flag`；`codeGenSetReg_cond`做的也是分配暫存器然後將`assembling language`寫到`output.s`再存到記憶體，其中`"cset"`是如果條件為真，則條件設置將目標暫存器設置為 1，否則將其設置為 0；`"ne"`則是比較第一個和第二個`operands`，如果兩個`operands`相等則返回值 0（假），如果兩個操作數不相等則返回值 1（真）。。

```
1 void genLogical(reg_type RegType, char *instruction, int dstRegIndex, int
  srcReg1Index, int srcReg2Index)
2 {
3     int boolReg1Index = -1;
4     int boolReg2Index = -1;
5
6     if (RegType == FLOAT_REG)
7     {
8         boolReg1Index = get_reg(INT_REG);
9         boolReg2Index = get_reg(INT_REG);
10        genBoolfromFloat(boolReg1Index, srcReg1Index);
11        genBoolfromFloat(boolReg2Index, srcReg2Index);
12    }
13    else if (RegType == INT_REG)
14    {
15        int zero = 0;
16        boolReg1Index = srcReg1Index;
17        boolReg2Index = srcReg2Index;
18        codeGen1Reg1ImmInstruction(INT_REG, "cmp", srcReg1Index, &zero);
19        codeGenSetReg_cond(INT_REG, "cset", srcReg1Index, "ne");
20        codeGen1Reg1ImmInstruction(INT_REG, "cmp", srcReg2Index, &zero);
```

```

21     codeGenSetReg_cond(INT_REG, "cset", srcReg2Index, "ne");
22 }

```

這裡和前面都類似，根據dataType，執行不同程式碼，但同樣呼叫genLogical，將"and"和"orr"分別送進去，再視狀況把暫存器釋放掉(leftOp->dataType == FLOAT_TYPE || rightOp->dataType == FLOAT_TYPE 時先進行釋放)。

```

1  void genExpr(AST_NODE *exprNode)
2  {
3      if (Expr_kind(exprNode) == BINARY_OPERATION)
4      {
5          ...
6          if (leftOp->dataType == FLOAT_TYPE || rightOp->dataType ==
FLOAT_TYPE){
7              switch (exprNode->semantic_value.exprSemanticValue.op.binaryOp){
8                  ...
9                  case BINARY_OP_AND:
10                     exprNode->Reg_No = get_reg(INT_REG);
11                     genLogical(FLOAT_REG, "and", exprNode->Reg_No, leftOp-
>Reg_No, rightOp->Reg_No);
12                     free_reg(FLOAT_REG, leftOp->Reg_No);
13                     break;
14                     case BINARY_OP_OR:
15                     exprNode->Reg_No = get_reg(INT_REG);
16                     genLogical(FLOAT_REG, "orr", exprNode->Reg_No, leftOp-
>Reg_No, rightOp->Reg_No);
17                     free_reg(FLOAT_REG, leftOp->Reg_No);
18                     break;
19                     ...
20                     ...
21                     else if (exprNode->dataType == INT_TYPE){
22                         exprNode->Reg_No = leftOp->Reg_No;
23
24                         switch (exprNode-
>semantic_value.exprSemanticValue.op.binaryOp){
25                             ...
26                             case BINARY_OP_AND:
27                                 genLogical(INT_REG, "and", exprNode->Reg_No, leftOp-
>Reg_No, rightOp->Reg_No);
28                                 break;
29                             case BINARY_OP_OR:
30                                 genLogical(INT_REG, "orr", exprNode->Reg_No, leftOp-
>Reg_No, rightOp->Reg_No);
31                                 break;
32                             ...
33                     ...
34 }

```

Result

Self-Testing Code

test_bonus/short_circuit.c

```

1  int T() {

```

```

2     write("true\n");
3     return 1;
4 }
5 int F() {
6     write("false\n");
7     return 0;
8 }
9 int main() {
10    write("test1:\n");
11    if (T() || F()) {
12        /* T() */
13    }
14    write("test2:\n");
15    if (F() && T()) {
16        /* F() */
17    }
18    write("test3:\n");
19    if ((T() && F()) || (F() || T())) {
20        /* T() -> F() -> F() -> T() */
21    }
22    write("test4:\n");
23    if ((T() || F()) || (F() || T())) {
24        /* T() */
25    }
26 }
27

```

Screenshot of Output

```

u16@ubuntu:~/project4/src$ ./run.sh ../test_bonus/short_circuit.c
Parsing completed. No errors found.
test1:
true
false
test2:
false
true
test3:
true
false
false
true
test4:
true
false
false
true

```

7. Multiple dimensional arrays

主要是下方程式碼在配置array相關的暫存器，並將這些結果存到記憶體。

```

1  int genArrayAddr(AST_NODE *idNode)
2  {
3      AST_NODE *traverseDim = idNode->child;
4      int *sizeInEachDimension = ID_SymTab(idNode)->attribute-
>attr.typeDescriptor->properties.arrayProperties.sizeInEachDimension;

```

```

5
6     genExprRelated(traverseDim);
7     int linearIdxRegisterIndex = traverseDim->Reg_No;
8     traverseDim = traverseDim->rightSibling;
9
10    int dimIndex = 1;
11
12    int shiftLeftTwoBits = 2;
13    genImmediate(INT_REG, "lsl", linearIdxRegisterIndex,
14    linearIdxRegisterIndex, &shiftLeftTwoBits);
15
16    char *linearOffsetRegName = NULL;
17    if (!isGlobalVar(ID_SymTab(idNode)))
18    {
19        int baseOffset = ID_SymTab(idNode)->attribute->offset;
20        genImmediate(INT_REG, "add", linearIdxRegisterIndex,
21        linearIdxRegisterIndex, &baseOffset);
22        allocate_reg_64(INT_REG, linearIdxRegisterIndex, 1, 0,
23        &linearOffsetRegName);
24        emit("add %s, %s, x29", linearOffsetRegName, linearOffsetRegName);
25    }
26    else
27    {
28        emit("ldr %s, =_g%s", int64_reg_rest[0], ID_Name(idNode));
29        allocate_reg_64(INT_REG, linearIdxRegisterIndex, 1, 1,
30        &linearOffsetRegName);
31        emit("add %s, %s, %s", linearOffsetRegName, linearOffsetRegName,
32        int64_reg_rest[0]);
33    }
34
35    memory_store(INT_REG, linearIdxRegisterIndex, linearOffsetRegName);
36
37    return linearIdxRegisterIndex;
38 }

```

Result

Self-Testing Code

test_bonus/multi_dim_array.c

```

1  int main(){
2      int b[3][4];
3      int b1[1][1][2];
4      b1[1][1][1] = 3;
5      b[3][4] = 7;
6      write(b1[1][1][1] + b[3][4]);
7      write("\n");
8      b1[1][1][2] = 4;
9      b[2][3] = 5;
10     write(b[2][3] - b1[1][1][2]);
11     write("\n");
12     return 0;
13 }

```

```

u16@ubuntu:~/project4/src$ ./run.sh ../test_bonus/multiple_dim_array.c
Parsing completed. No errors found.
10
1

```

3

8. for-loop

主要就是做一些判斷看loopNode要對應到statement還是block(即general node) · 或是甚麼都不用做。

src/codeGen.c

```

1 void genFor(AST_NODE *forNode)
2 {
3     int for_label = for_label_no++;
4     AST_NODE *initNode, *condNode, *incrNode, *loopNode;
5     initNode = forNode->child;
6     condNode = initNode->rightsibling;
7     incrNode = condNode->rightsibling;
8     loopNode = incrNode->rightsibling;
9     emit("_forStart%d:", for_label);
10    if (initNode->nodeType == NONEMPTY_ASSIGN_EXPR_LIST_NODE)
11    {
12        AST_NODE *assignNode = initNode->child;
13        for (; assignNode != NULL; assignNode = assignNode->rightsibling)
14        {
15            genAssign(assignNode);
16        }
17    }
18    emit("_forLoop%d:", for_label);
19    if (condNode->nodeType == NONEMPTY_RELOP_EXPR_LIST_NODE)
20    {
21        AST_NODE *exprNode = condNode->child;
22        AST_NODE *last_cond;
23        int cond;
24        for (; exprNode != NULL; exprNode = exprNode->rightsibling)
25        {
26            last_cond = exprNode;
27            genExprRelated(exprNode);
28        }
29        if (last_cond->dataType == INT_TYPE)
30        {
31            emit("cmp w%d, #0", cond);
32        }
33        else
34        {
35            emit("fcmp s%d, #0", cond);
36        }
37        free_reg(INT_REG, cond);
38        emit("beq _forExit%d", for_label);
39    }
40    if (loopNode->nodeType == STMT_NODE)
41    {

```

```

42     gen_stmt(loopNode);
43 }
44 else if (loopNode->nodeType == BLOCK_NODE)
45 {
46     gen_block(loopNode);
47 }
48 else if (loopNode->nodeType == NUL_NODE)
49 {
50     // do nothing
51 }
52 if (incrNode->nodeType == NONEMPTY_ASSIGN_EXPR_LIST_NODE)
53 {
54     AST_NODE *assignNode = incrNode->child;
55     for (; assignNode != NULL; assignNode = assignNode->rightSibling)
56     {
57         if (assignNode->nodeType == STMT_NODE)
58         {
59             genAssign(assignNode);
60         }
61         else if (assignNode->nodeType == EXPR_NODE)
62         {
63             genExpr(assignNode);
64         }
65     }
66 }
67 emit("b _forLoop_%d", for_label);
68 emit("_forExit_%d:", for_label);
69 }

```

下面是switch到FOR_STMT的程式碼，基本上就是直接call function。

src/codeGen.c

```

1  switch (Stmt_Kind(stmtNode)){
2      ...
3      case FOR_STMT: // 9)    For loops
4          genFor(stmtNode);
5          break;
6      ...
7  }

```

Result

Self-Testing Code

test_bonus/for.c

```

1  int main(){
2      int a;
3      for (a = 1; a < 6; a=a+1){
4          write(a);
5          write("\n");
6      }
7  }

```



```
u16@ubuntu:~/project4/src$ ./run.sh ../test_bonus/for.c
Parsing completed. No errors found.
1
2
3
4
5
```

9. Implicit type conversions

Not done.

10. Variable initializations

Not done.

11. Procedure and function calls with parameters

Not done.

Part 2. Implement one optimization for your compiler

Part 2 我選擇實作 Constant folding 這個最佳化技術，Constant folding是在編譯時識別和evaluate constant expression，而不是在執行時才計算它們，主要實作程式碼如下：

src/semanticAnalysis.c

```
1 // Constant folding
2     if((exprNode->dataType != ERROR_TYPE) &&
3         (leftOp->nodeType == CONST_VALUE_NODE || (leftOp->nodeType ==
4     EXPR_NODE && leftOp->semantic_value.exprSemanticValue.isConstEval)) &&
5         (rightOp->nodeType == CONST_VALUE_NODE || (rightOp->nodeType ==
6     EXPR_NODE && rightOp->semantic_value.exprSemanticValue.isConstEval))
7     )
8     {
9         evaluateExprValue(exprNode);
10        exprNode->semantic_value.exprSemanticValue.isConstEval = 1;
11    }
```

主要就是當 leftnode 和 rightnode 都是constant，即const_value_node 或 expr_node 是constant 時，就簡化constant。