Compiler final exam answer

1.

(a)

charA; Rule 5.
charA; Rule 4.
charA; Rule 4.
Rule 3: 1.
Rule 2: $$ = 0; $2 = 1.
Rule 2: $$ = 0; $2 = 1.
Rule 2: $$ = 0; $2 = 1.
Rule 2: $$ = 0; $2 = 1.
Rule 2: $$ = 0; $2 = 1.
Rule 2: $$ = 0; $2 = 1.
Rule 1: 1.

(b)

The rule :

R : charB R {$2 = $$ + 1; printf("Rule 2: $$ = %d; $2 = %d.\n", $$, $2);};

$$為 return value，在初始化之前就被拿去使用。

2.

fa:

```
addi sp,sp,-48
sd   ra,40(sp)
sd   s0,32(sp)
sd   s1,24(sp)
addi s0,sp,48
```

Allocate stack space, and store some register

```
mv   a5,a0
sw   a5,-36(s0)
li   a4,1
bne  a4, a5, .L2   // if a5 != 1, go to .L2
li   a5,1
j    .L3
.L2:
lw   a5,-36(s0)
bnez      a5, .L4   // if a5 != 0, go to .L4
li   a5,0
```

```
        j       .L3
    .L4:
        lw      a5,-36(s0)
        addi    a5,a5,-1
        mv      a0,a5       // set parameter value of fa(). a0 is used to pass parameter.
        call    fa
        mv      s1,a0
        lw      a5,-36(s0)
        addi    a5,a5,-2
        mv      a0,a5     // set parameter value of fa(). a0 is used to pass parameter.
        call    fa
        mv      a5,a0
        add     a5,s1,a5    // fa(a) + fa(a-1)
    .L3:
        mv      a0,a5       // return value stored in a0.
        ld      ra,40(sp)
        ld      s0,32(sp)
        ld      s1,24(sp)
        addi    sp,sp,48
        jr      ra
```

free stack space, and
recover some register

```
    fibonacci:
        addi    sp,sp,-32
        sd      ra,24(sp)
        sd      s0,16(sp)
        addi    s0,sp,32
        li      a5,5            //store an immediate value to a register
        sw      a5,-20(s0)    // The address -20(s0) represented as variable a.
        mv      a0,a5       // set parameter value of fa(). a0 is used to pass parameter.
        call    fa
        sw      a0,-24(s0)    // return value stored in a0, and the address -24(s0)
represented as variable b.
        ld      ra,24(sp)
        ld      s0,16(sp)
        addi    sp,sp,32
        jr      ra
```

Allocate stack space, and
store some register

free stack space, and
recover some register

3.

```
int index = 0;
struct var_node {
    string name;
    struct var_node* next;
}
struct symbolTable_entry symbolTable[enough];

P := DclList
DclList := Dcl
DclList := Dcl DclList
Dcl := Type VarList semicolon
        { while($2 != NULL) {
                symbolTable[index]->idtype = $1;
                symbolTable[index]->name = $2->name;
                index++;
                $2 = $2->next;
          }
         }
Type := int {$$ = 1;}
Type := float {$$ = 2;}
VarList := Var { $$ = $1;}
VarList := Var comma VarList {$1->next = $3; $$ = $1}
Var := id { $$ = newNode(); $$->name = $1;}
```

4.

- Activation record is used to manage the information needed by a single execution of a procedure.
- An activation record is pushed into the stack when a procedure is called and it is popped when the control returns to the caller function.
- Contents in the Activation record
  - Return Value: It is used by calling procedure to return a value to calling procedure.
  - Actual Parameter: It is used by calling procedures to supply parameters to the called procedures.
  - Control Link: It points to activation record of the caller.
  - Access Link: It is used to refer to non-local data held in other activation records.

- Saved Machine Status: It holds the information about status of machine before the procedure is called.
- Local Data: It holds the data that is local to the execution of the procedure.
- Temporaries: It stores the value that arises in the evaluation of an expression.

5.

Static Chain (在 compiler time 被建立)
A chain of static links that connects certain activation record instance (ARI) in the static.

Static Links
Static scope pointers to an ARI.

ARI
It is used to manage information needed by a procedure. It is pushed to stack while procedure is called.

Dynamic Chain(在 Runtime 時才會建立)
The collection of dynamic links in the stack.

Dynamic Link
永遠指向 caller ARI 的頂端

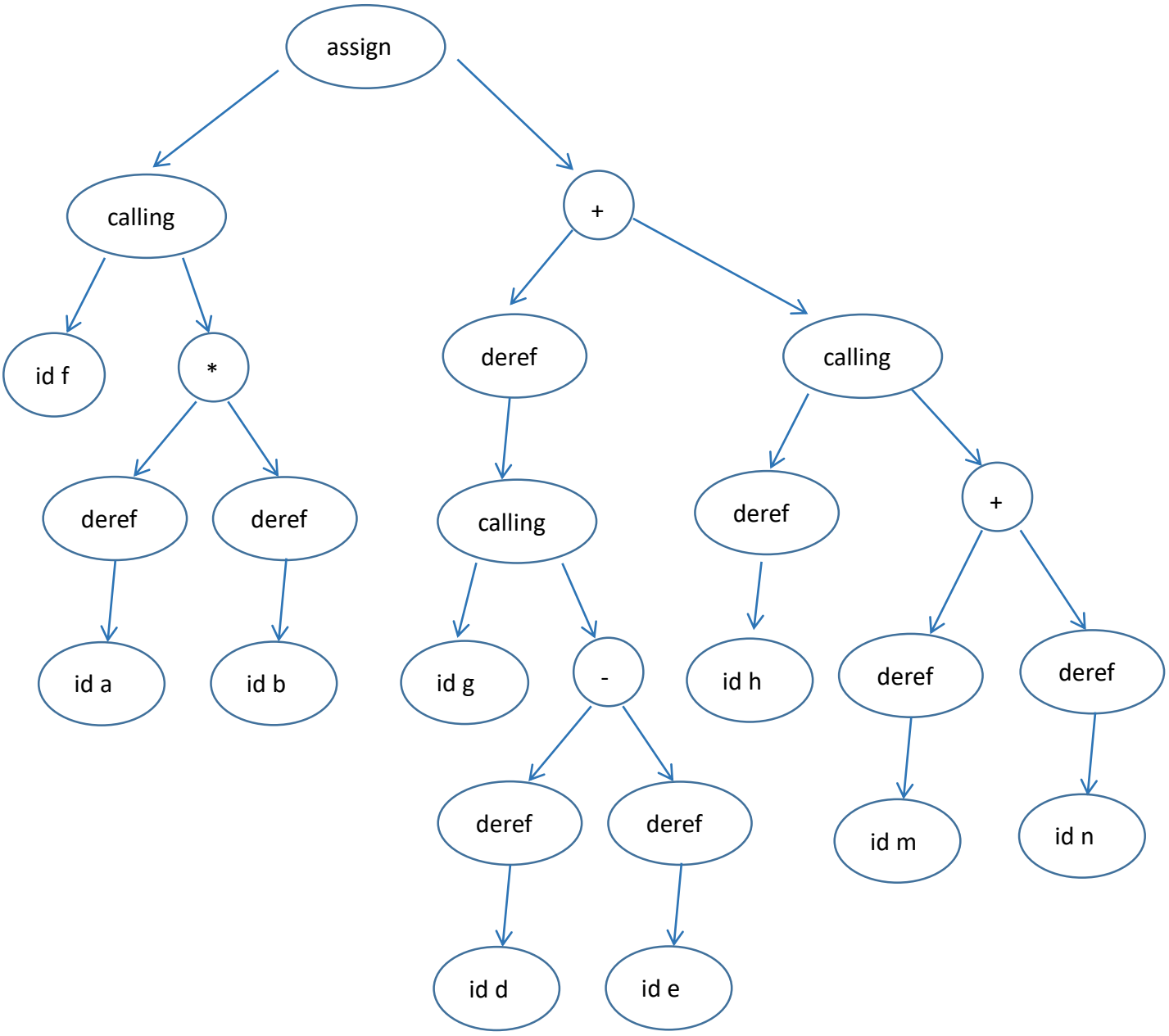
6.

expr -> expr PLUS term	{$$ = new add($1, $3)}
expr -> term	{$$ = $1}
term -> factor EXP term	**{$$ = new exp($1,$3)}**
term -> factor	{$$ = $1}
factor -> num	{$$ = new num($1.val)}
factor -> LPAR expr RPAR	{$$ = $2}

i.e. 1+1^1^1^1^1^1+1

State	Action						GOTO		
	PLUS	EXP	num	LPAR	RPAR	$	expr	term	factor
I0			S3	R6			1	4	2
I1	S6					Acc			
I2		S8				R4			
I3		R5							
I4						R2			
I5			S3	S5			9	4	2
I6				S5				7	2
I7						R1			
I8			S3	S5				7	2
I9					S10				
I10		R6							
I11			S3`	S5`				12	2`
I12					R1				
I2`		S8`							
I3`					R5				
I4`					R2				
I5`			S3`	S5`			9`	4`	2`
I7`					R1				
I8`			S3`	S5`				7`	2`
I9`					S10`				
I10`					R6				

7.


assign
calling
+
id f
*
deref
calling
id a
deref
deref
id b
id g
-
deref
deref
id d
id e
deref
calling
deref
+
id h
deref
deref
id m
id n