Chapter 16

Compiling Programs in Functional Form

Linking with start-up code

```
java S5 m1
java S5 m2
a m1.a m2.a sup
e m1 /c
```

Separately-compiled modules

a)

m1.s

```
extern int x;
void main()
{
  println(x);
}
```

m2.s

```
int x = 5;
```

b)

m1.a

```
        extern    x
        public    main
main:   .
        .
        .
        p         x
        .
        .
        .
```

m2.a

```
        public    x
x:      dw        5
        .
        .
        .
```

Calling a funciton

```
f(2, y, y+3);
```

```
; push values of args onto stack to create
; the corresponding parameters
pwc       2
p         y
p         y
pwc       3
add

; call the function f
call      f

; add 3 to sp which effectively pops three values previously pushed
asp       3
```

Example

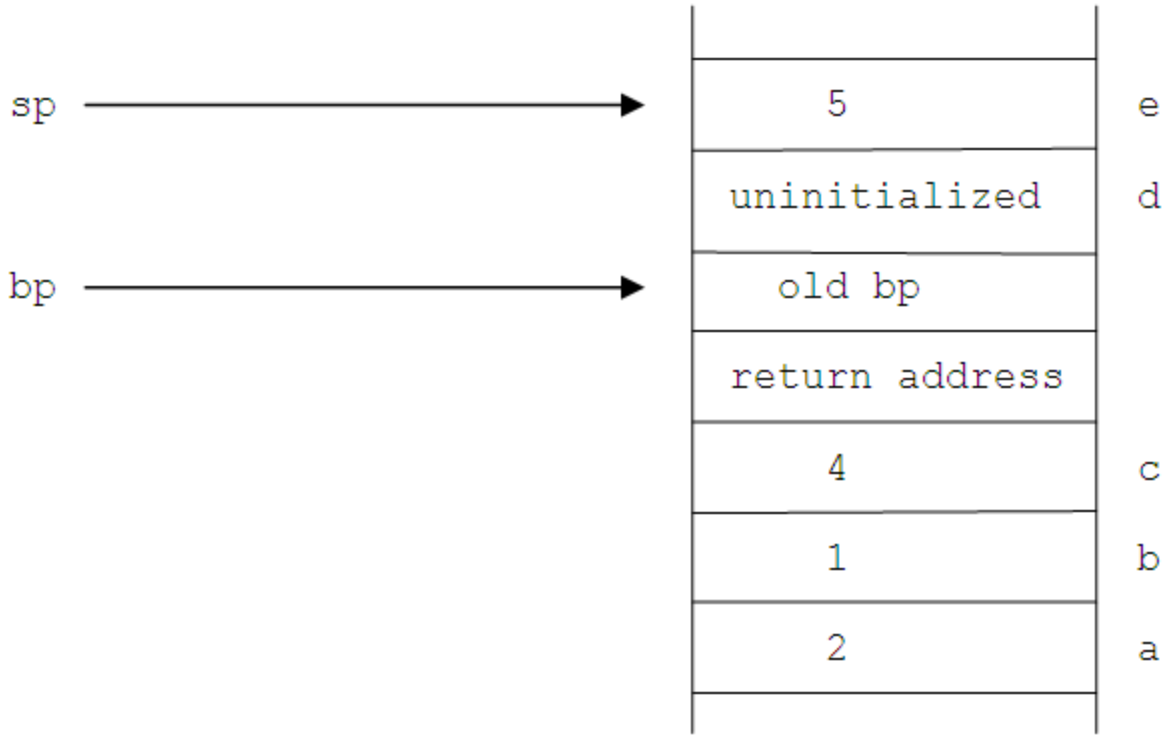
```
 1 int x, y = 1;                    public    x       ; x global
 2                                x:  dw    0
 3                                    public    y       ; y global
 4                                y:  dw    1
 5
 6 void main()                       public    main    ; main global
 7                                main:
 8
 9
10                                      esba          ; set up bp
11 {
12
13    f(2, y, y + 3);                   ; create parms a, b, and c
14                                      pwc   2       ; create a
15                                      p     y       ; create b
16                                      p     y       ; push y value
17                                      pwc   3       ; push 3
18                                      add           ; create c
19
20                                      ; transfer control to f
21                                      call  f
22
23                                      ; remove parms a, b, and c
24                                      asp   3
25
26 }                                    reba          ; prepare for ret
27                                      ret           ; to sup code
```

Example continued

```
30 void f(int a, int b, int c)    public f
31                              f:
32 {                               esba          ; set up base reg
33
34    int d, e = 5;                asp   -1      ; allocate d
35                                 pwc   5       ; allocate/init e
36
37    x = y;                       pc    x       ; push x address
38                                 p     y       ; push y value
39                                 stav          ; do assignment
40
41    a = b + c;                   cora  4       ; push a address
42                                 pr    3       ; push b value
43                                 pr    2       ; push c value
44                                 add           ; compute b + c
45                                 stav          ; do assignment
46
47    d = e;                       cora  -1      ; push d address
48                                 pr    -2      ; push e value
49                                 stav          ; do assignment
50
51 }                               reba          ; prepare for ret
```

Activation record

b)



Symbol table

a)

```
 1  int q;            // entered as GLOBALVARIABLE
 2  extern r;         // entered as EXTERNVARIABLE
 3  void main()       // entered as FUNCTIONDEFINITION
 4  {
 5     int x = 70;
 6     g(x);          // entered as a FUNCTIONCALLl
 7  }
 8  void g(int a)     // change g entry to FUNCTIONDEFINITION
 9  {
10     println("hello");
11     h();           // entered as a FUNCTIONCALL
12  }
```

b)

	symbol (String)	relAdd (Integer)	category (Integer)
0	"q"	0	GLOBALVARIABLE
1	"r"	0	EXTERNVARIABLE
2	"main"	0	FUNCTIONDEFINITION
3	"x"	-1	LOCAL
	"g"	0	FUNCTIONDEFINITION
5	"a"	2	LOCAL
6	"h"	0	FUNCTIONCALL

Methods in symbol table

`public void enter(String sym, int ra, int cat)`

Enters `sym` into the symbol table

`public int find(String sym)`

Searches `symbol` for `sym`. If it finds it, it returns its index. Otherwise, it throws an exception. `find` searches in reverse order—that is , from the most recent entry to the least recent entry.

`public String getSymbol(int i)`

Returns `symbol`entry at index `i`.

`public Integer getRelAdd(int i)`

Returns `relAdd` entry at index `i`.

`public Integer getCategory(int i)`

Returns `category` entry at index `i`.

`public int getSize()`

Returns the size of the symbol table.

`public void localRemove()`

Removes all `LOCAL` entries in the symbol table.

Code generator

`public void emitString(String s)`

Calls `outFile.println(s)`.

`public void emitInstruction(String op)`

Outputs instruction that consists of the mnemonic `op` only.

`public void emitInstruction(String op, String opnd)`

Outputs instruction that consists of a mnemonic `op` and an operand `opnd`.

`public void emitdw(String label, String value)`

Outputs `label`, `":"`, and `value` by calling `printf()`.

`public void endCode()`

Outputs an `extern` statement for every `FUNCTIONCALL` entry in the symbol table.

`public String getLabel()`

Returns the strings in the sequence `"@L0"`, `"@L1"`, ... to serve as labels for string constants and for the jump instructions.

`public void emitLabel(String label)`

Outputs `label` followed by `":"`

Code generator

`public void push(int p)`

If the index `p` corresponds to a non-`LOCAL` entry, `push()` outputs the `p` mnemonic followed by the variable name obtained with `st.getSymbol(p)` (`st` is the reference to the symbol table). For example,

```
p       x           ; global
```

For a `LOCAL` entry, `push()` outputs the `pr` mnemonic followed by the variable's relative address (obtained by `st.getRelAdd(p)`). For example,

```
pr      -1          ; local
```

`public void pushAddress(int p)`

Similar to `push()`, except it outputs the mnemonics `pc` or `cora` if the variable is non-`LOCAL` or `LOCAL`, respectively. For example, for the global variable `x`, it outputs

```
pc      x
```

For the local variable with relative address -1, it outputs

```
cora    -1          ; local
```

Translation grammar for S5

S5.txt

Creating S5

```
javac S5.java
java S5 S5a
java S5 S5b
a S5a.a S5b.a sup
e S5a /c
```