SERIES
ONE

CODE GENERATION (OF ASSEMBLY CODE)

Exercise 1

We consider the following program.

```
main(c) {
   int x, y, z;
   x = 5;
   y = 1;
   z = x + y;
}
```

1. Determine the symbol table (integers are coded on 4 bytes).
2. Generate assembly code.

Exercise 2

Let us consider the following program with its symbol table.

```
var x, y : int ;
while (not x = y) do
   if x > y then x := x - y else y := y - x fi
od
```

Environment

Procedure	Shifting
main	x $\rightarrow$ 4 y $\rightarrow$ 8

1. Generate code.

Exercise 3

1. Extend the code generation function to handle the following construct:
   ```
   for (var = lbound; var < ubound; step) {
       S(variable)
   }
   ```
   where:
   - `lbound` and `ubound` are respectively the lower and upper bounds, given by arithmetical expressions,
   - `S(var)` is a statement that depends on `variable`, and
   - step is an expression that is added to `variable` at the end of each iteration.
2. For the program below, generate assembly code.
   ```
   int t1[10];
   int i;
   for(i=0;i<10;i++) { t1[i] = 0 }
   ```
3. For the program below, generate assembly code.

```
int t1[10][10] ;
int i,j ;
for(i=0; i<10; i++) {
  for(j=0; j<10; j++)
    t1[i][j] = 0;
 }
```

Exercise 4

Extend the code generation function seen for While.

1. To consider statements of the form `repeat` S `until` b ,
2. To consider Boolean expressions of the form `b1 xor b2`,
3. To consider arithmetical expressions of the form `b ?  e1 :  e2`.

Exercise 5

Let us consider the two following programs with their symbol tables. For each of them, generate code.

```
begin
 var x : int ; var y : int ;
 procedure p is
    y := x ;
 end
 x := 1 ;
 call p ;
end
```

Environment

Procedure	Shifting
main	x → 4 y → 8
main.p	

Exercise 6

We consider the following program.

```
 var x : int ; var y : int ; var z : int : // global variables (in main)
 procedure p is
  begin var x : int ; var t : int :
    x := 4 ;
    y := 5 ;
    z := x + y ;
  end
 // program body (main)
  x := 5;
  call p ;
end
```

Environment

Procedure	Shifting
main	x ↦ 4 y ↦ 8 z ↦ 12
main.p	x ↦ 4 t ↦ 8

1. Generate the corresponding assembly code.

Exercise 7

Let us consider the following program:

```
var x, y ; // global variables

proc p1 is
   var y, z
in
    y := x+1 ;
    z := y+2 ;
    y := z*3 ;
end

proc p2 is
   var x, z
in
```

```
    x := y-1 ;
    call p1 ;
    z := x*2 ;
end

begin
  // program body
x := 1 ;
call p1 ;
call p2 ;
end
```

1. Draw the complete execution stack when procedure `p2` is executed
2. Give the code produced for procedure `p1`
3. Give the code produced for procedure `p2`
4. Give the code produced for the program body
5. We change the code of `p1` as follows:

   ```
   proc p1 is
      var y, z
   in
       y := x+1 ;
       if y>0 then
            x := x-1 ;
            call p1 ;
       else
           call p2 ;
       fi
   end
   ```

   1. Draw the complete execution stack when procedure `p2` is executed (starting the program from the beginning)
   2. Give the new code produced for procedure `p1`

Exercise 8

Let us consider the following program where parameters are **passed by value**:

```
var x, y ; // global variables

proc p1(y, z) is
   var t
in
    t := x+z ;
    y := t*2+y ;
end

proc p2 (t, v, w) is
   var x, z
in
    z := y+t-1 ;
    call p1 (x, w+v) ;
    w := t+z
end

begin
  // program body
```

```
x := 1 ;
call p1 (x, 42) ;
call p2 (12, x, y) ;
end
```

1. Draw the complete execution stack when procedure `p2` is executed
2. Give the code produced for procedure `p1`
3. Give the code produced for procedure `p2`
4. Give the code produced for the program body
5. We now consider that:
   - parameter `y` of `p1` is passed by **address**
   - parameter `w` of `p2` is passed by **value-result**

   Indicate what is changed on your answers for questions 2, 3 and 4 ...

Exercise 9

Let us consider the following program (where parameters are **passed by value**):

```
var z1 ;
var p proc (int) ; /* p is a procedure variable */
proc p1 (x : int) is z1 := x ;
proc p2 (q : proc (int)) is call q(2) ;

// program body
p := p2 ;
call p (p1) ;
```

1. Draw the complete execution stack when `p1` is executed
2. Give the code produced for `p1`
3. Give the code produced for `p2`
4. Give the code produced for the program body.

Exercise 10

Let us consider the following program:

```
var x : int ; var y : int ; var z : int ; var a : int ; var b : int ; var c : int ;
c := 4 ;
x := a+b+c;
y := a * b + 2 * c ;
```

1. Generate 3-address code and then assembly code by supposing an unlimited number of registers.
2. Generate 3-address code and then assembly code by supposing that we have only 4 registers R_1, R_2, R_3, R_4.