Homework3

Hyunki Lee

Panther# 002-34-4677

1.

a. main - a, [b, c]

fun1 – b, [c, d]

fun2 – c, [d, e]

fun3 – d, e, f

finally

main – a

fun1 – b

fun2 – c

fun3 – d, e, f

b. main – a, [b, c]

fun1 – b, c, [d]

fun3 – d, e, f

finally

main – a

fun1 – b, c

fun3 – d, e, f

c. main – a, [b], [c]

fun2 – [c], [d, e]

fun3 – [d], e, f

fun1 – b, c, d

finally

main – a

fun3 – e, f

fun1 – b, c, d

d. main – a, [b, c]

fun3 – [d], e, f

fun1 – b, c, d

finally

main – a

fun3 – e, f

fun1 – b, c, d

e. main – a, [b, c]

fun1 – b, [c], [d]

fun3 – [d, e], f

fun2 – c, d, e

finally

main – a

fun1 – b

fun3 – f

fun2 – c, d, e

f. main – a, [b, c]

fun3 – [d, e], f

fun2 – [c, d], e

fun1 – b, c, d

finally

main – a

fun3 – f

fun2 – e

fun1 – b, c, d

e. main – a, [b], [c]

fun2 – [c, d], [e]

fun1 – b, c, [d]

fun3 – d, e, f

finally

main – a

fun1 – b, c

fun3 – d, e, f

d. main – a, [b], [c]

fun2 – [c, d], e

fun1 – b, c, d

finally

main – a

fun2 – e

fun1 – b, c, d

2.

sub1()

a = 7 declared at sub1()

y = 9 declared at sub1()

z = 11 declared at sub1()

x = 1 declared at main()

sub2()

a = 13 declared at sub2()

x = 15 declared at sub2()

w = 17 declared at sub2()

y = 3 declared at main()

z = 5 declared at main()

sub3()

a = 19 declared at sub3()

b = 21 declared at sub3()

z = 23 declared at sub3()

x = 15 declared at sub2()

y = 3 declared at main()

w = 17 declared at sub2()

3. Java Script

function nest(){

function fun1(){

var x = 10;

function fun2(){

var a = x;

function fun3(){

var b = x;

return b;

}

return fun3();

}

return fun2();

}

return fun1();

}

nest();

4. Python

def fun1():

x = 10

def fun2():

a = x

def fun3():

b = a

print("fun3", b)

fun3()

print("fun2", a)

fun2()

print("fun1", x)

fun1()

5. Java

EBNF rule

<while\_stmt> 🡪 while “(“<boolexpr>”)” <statement>

Recursive-descent subprogram

Void whilestmt(){

if (nextToken != WHILE\_CODE)

error();

else {

lex();

if (nextToken != LEFT\_PAREN)

error();

else {

boolexpr();

if (nextToken != RIGHT\_PAREN)

error();

else {

statement();

}

}

}

6.

C language

int x, y;

y = x - 3;

|  |  |
| --- | --- |
| Various binding | Binding time |
| Data type of x and y | Compile time |
| Possible value of x and y | Compile time |
| ‘=’ assignment | Language design time |
| ‘-‘ operator | Compile time |
| Value of y | Execution time |

7.

Dynamic type binding is that the type of a variable is determined by the type of the last assigned value. Implicit heap-dynamic variables are bound to heap storage only when they are assigned values. Thus, variables from implicit heap-dynamic variables are in dynamic type binding.

8.

History-sensitive variable is static variables. It means that the variables retain their values between separate executions of the subprograms. The history-sensitive variables are useful when we need fixed data. For example, maintaining students’ information, we can have history-sensitive variable for unique studentIDs then we can manipulate other data such as gpa, address, phone number etc.