CS/CE 4337 - Assignment#6
Due Date: 12/8/19, 11:59 pm

1.A) Show the stack with all activation record instances, including static and dynamic chains, when execution reaches position 1 in the `Bigsub` program, as shown in Figure 1. Assume `Bigsub` is at level 1.

Yu Feng
Robert van Tilburg

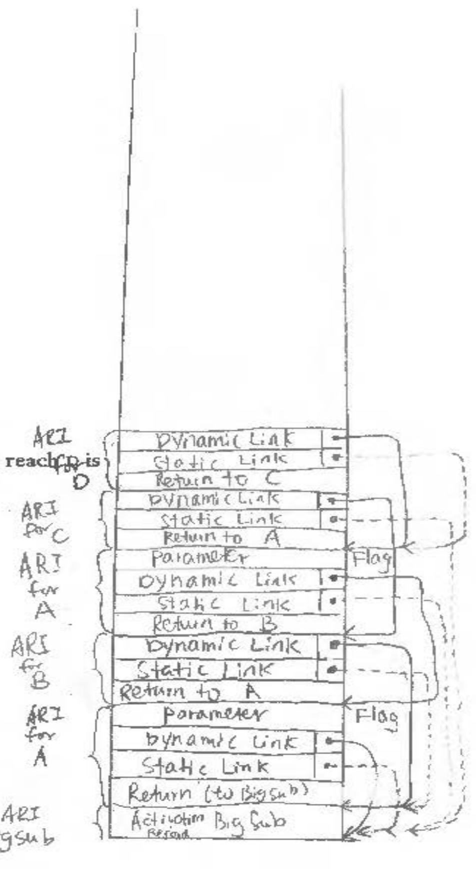
1- A) Show the stack with all activation record instances, including static and dynamic chains, when execution reaches position 1 in the following skeletal program. Assume `Bigsub` is at level 1.

```
0 procedure Bigsub is
1   procedure A(Flag : Boolean) is
2     procedure B is
       . . .
       A(false);
       end; -- of B
     begin -- of A
     if flag
      then B;
      else C;
     . . .
     end; -- of A
1   procedure C is
2     procedure D is
       . . . <------------ 1
       end; -- of D
     . . .
     D;
    end; -- of C
   begin -- of Bigsub
   . . .
   A(true);
   . . .
   end; -- of Bigsub
```

The calling sequence for this program for execution to reach D is

```
Bigsub calls A
A calls B
B calls A
A calls C
C calls D
```

Static : → closest parent
Dynamic : → the caller



1.B) Show the stack with all activation record instances, including the dynamic chain, when execution reaches position 1 in the `Bigsub` program, as shown in Figure 2. This program uses the deep access method to implement dynamic scoping.

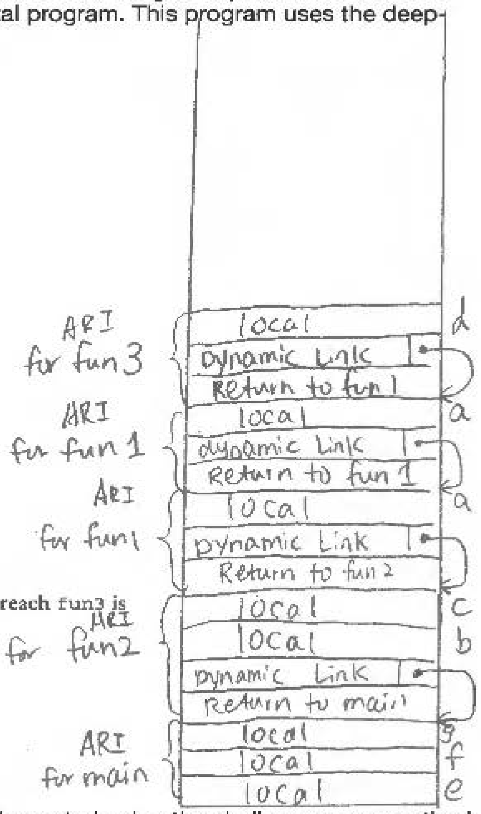
1.C) Assume that the program illustrated in Figure 2 is implemented using the shallow-access method using a stack for each variable name. Show the stacks for the time of the execution of `fun3`

1.B) Show the stack with all activation record instances, including the dynamic chain, when execution reaches position 1 in the following skeletal program. This program uses the deep-access method to implement dynamic scoping.

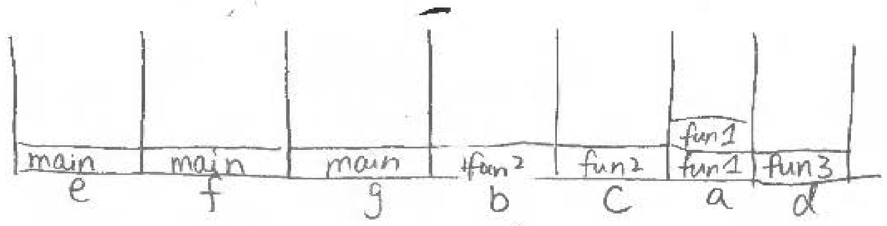
```
void fun1() {
  float a;
  . . .
}

void fun2() {
  int b, c;
  . . .
}

void fun3() {
  float d;
  . . . <------------ 1
}

void main() {
  char e, f, g;
  . . .
}
```

The calling sequence for this program for execution to reach fun3 is

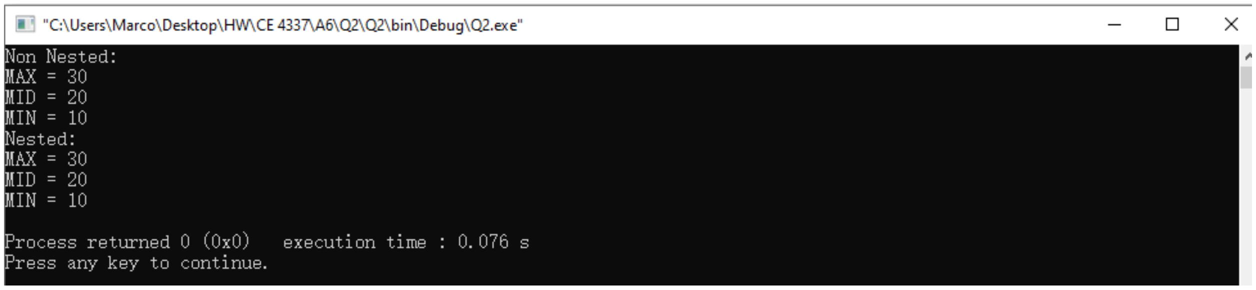
```
main calls fun2
fun2 calls fun1
fun1 calls fun1
fun1 calls fun3
```



1.C) - Assume that the above program (1.B) is implemented using the shallow-access method using a stack for each variable name. Show the stacks for the time of the execution of fun3



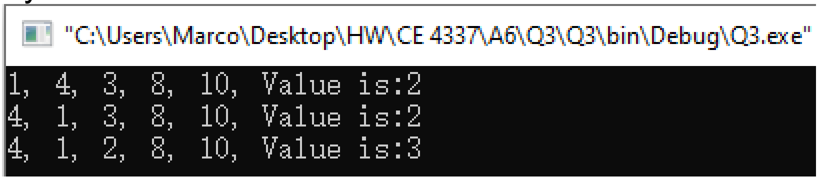
2- Consider the following programming problem: The values of three integer vari- ables—`first`, `second`, and `third`—must be placed in the three variables `max`, `mid`, and `min`, with the obvious meanings, without using arrays or user-defined or predefined subprograms. Write two solutions to this problem, one that uses nested selections and one that does not. Compare the complexity and expected reliability of the two.



```
"C:\Users\Marco\Desktop\HW\CE 4337\A6\Q2\Q2\bin\Debug\Q2.exe"
Non Nested:
MAX = 30
MID = 20
MIN = 10
Nested:
MAX = 30
MID = 20
MIN = 10

Process returned 0 (0x0)   execution time : 0.076 s
Press any key to continue.
```

3- Write a program, using the syntax of whatever language you like, that produces different behavior depending on whether pass-by-reference or pass-by-value-result is used in its parameter passing

By Reference:



```
"C:\Users\Marco\Desktop\HW\CE 4337\A6\Q3\Q3\bin\Debug\Q3.exe"
1, 4, 3, 8, 10, Value is:2
4, 1, 3, 8, 10, Value is:2
4, 1, 2, 8, 10, Value is:3
```

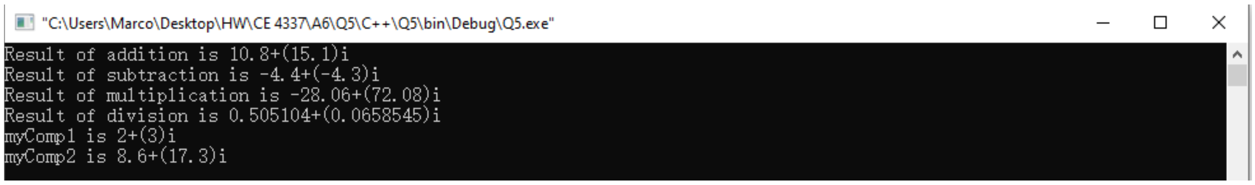
By Value-Result: Not supported by C++. The result list will be [4, 1, 3, 2, 10], value = 3, as ls[value] changes once value changes.

4- The static-chain method could be expanded slightly by using two static links in each activation record instance where the second points to the static grandparent activation record instance. How would this approach affect the time required for subprogram linkage and nonlocal references?

- It can increase the efficiency of execution, as it reduces time to access non-local variables defined two steps away from the current subprogram and make the access time the same as non-local variables defined one step away.

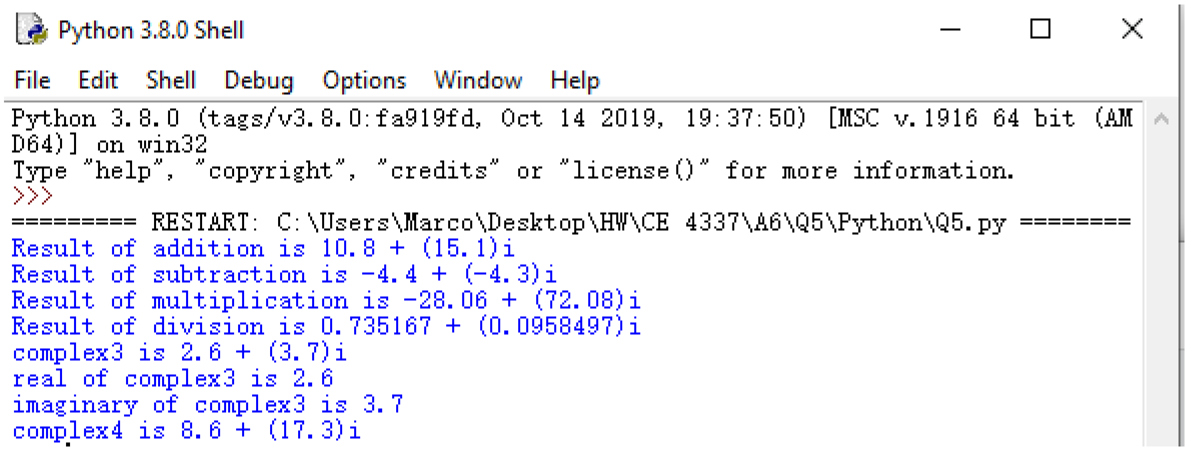
5- Write an abstract data type in C++ and python for complex numbers, including operations for addition, subtraction, multiplication, division, extraction of each of the parts of a complex number, and construction of a complex number from two floating-point constants, variables, or expressions.

C++:



```
"C:\Users\Marco\Desktop\HW\CE 4337\A6\Q5\C++\Q5\bin\Debug\Q5.exe"
Result of addition is 10.8+(15.1)i
Result of subtraction is -4.4+(-4.3)i
Result of multiplication is -28.06+(72.08)i
Result of division is 0.505104+(0.0658545)i
myComp1 is 2+(3)i
myComp2 is 8.6+(17.3)i
```

Python:



```
Python 3.8.0 (tags/v3.8.0:fa919fd, Oct 14 2019, 19:37:50) [MSC v.1916 64 bit (AM
D64)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>>
========= RESTART: C:\Users\Marco\Desktop\HW\CE 4337\A6\Q5\Python\Q5.py ========
Result of addition is 10.8 + (15.1)i
Result of subtraction is -4.4 + (-4.3)i
Result of multiplication is -28.06 + (72.08)i
Result of division is 0.735167 + (0.0958497)i
complex3 is 2.6 + (3.7)i
real of complex3 is 2.6
imaginary of complex3 is 3.7
complex4 is 8.6 + (17.3)i
```

6- The following are design issues for subprograms. Explain the design choices in python for these issues. Please also provide enough examples to illustrate the design choices.

- Are local variables statically or dynamically allocated?
  - All local variables in Python methods are stack dynamic. If the name of a global variable is assigned in a method, it is implicitly declared to be a local and the assignment does not disturb the global.

```
def f():
    s = "s in f"
    print s
s = "s outside"
f()
print s
```

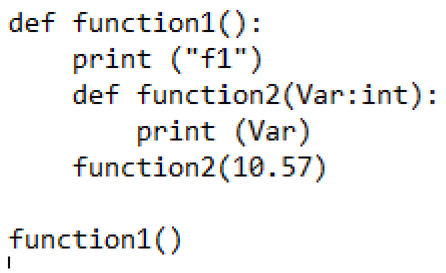
  -
- Can subprogram definitions appear in other subprogram definitions?
  - Yes. It creates a hierarchy of both logic and scopes, and provides a highly structured way to grant access to nonlocal variables in enclosing subprograms.

```
def function1():
    print ("f1")
    def function2():
        print ("f2")
    function2()
```

  - `function1()`
- What parameter-passing method or methods are used?
  - Pass-by-assignment. Because all data values are objects, every variable is a reference to an object. In pass-by-assignment, the actual parameter value is assigned to the formal parameter.
  - Suppose a reference to an array is passed as a parameter. If the corresponding formal parameter is assigned a new array object, there is no effect on the caller. However, if the formal parameter is used to assign a value to an element of the array, the actual parameter is affected. So, changing the reference of the formal parameter has no effect on the caller, but changing an element of the array that is passed as a parameter does.
- Are the types of the actual parameters checked against the types of the formal

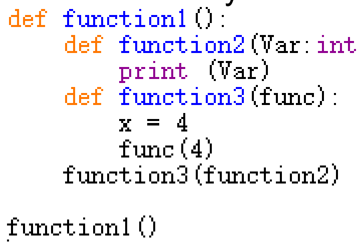
parameters?

- In Python, there is no type checking of parameters.
- Typing in these languages is a different concept. Objects have types, but variables do not, so formal parameters are typeless. This disallows the very idea of type checking parameters.



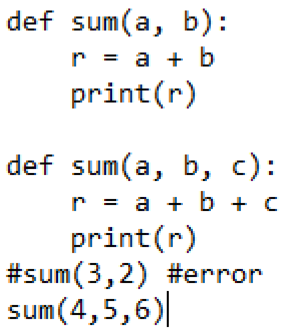
```
def function1():
    print ("f1")
    def function2(Var:int):
        print (Var)
    function2(10.57)

function1()
```

- If subprograms can be passed as parameters and subprograms can be nested, what is the referencing environment of a passed subprogram?
  - Deep binding is the choice. It is the most natural for static scoped languages.
  - Subprograms are treated like data, so they can be assigned to variables, and be referenced by another subprogram.



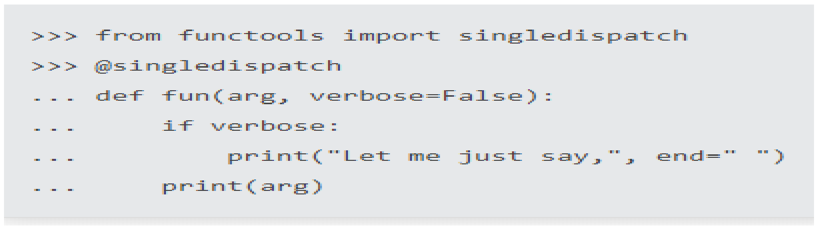
```
def function1():
    def function2(Var:int):
        print (Var)
    def function3(func):
        x = 4
        func(4)
    function3(function2)

function1()
```

- Can subprograms be overloaded?
  - Python only allows subprogram overloading when creating a class. Otherwise, overloading of subprogram is ignored as it will only execute the closest function declaration to the calling statement.



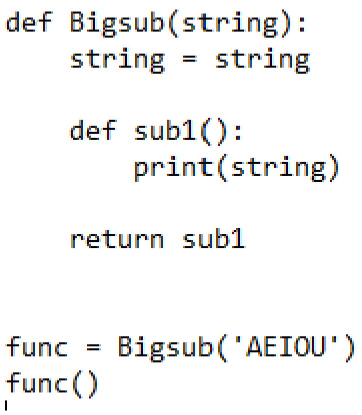
```
def sum(a, b):
    r = a + b
    print(r)

def sum(a, b, c):
    r = a + b + c
    print(r)
#sum(3,2) #error
sum(4,5,6)
```

- Can subprograms be generic?
  - A more general kind of polymorphism is provided by the methods of Python. a method will work for any type of actual parameter, as long as the operators used on the formal parameters in the method are defined.
  - Example from Python webpage:



```
>>> from functools import singledispatch
>>> @singledispatch
... def fun(arg, verbose=False):
...     if verbose:
...         print("Let me just say,", end=" ")
...     print(arg)
```

- If the language allows nested subprograms, are closures supported?
    - It is supported. The referencing environment of subprogram can include variables defined in all enclosing subprograms. Closures help to invoke function outside their scope. This helps us to reduce the use of global variables.



```
def Bigsub(string):
    string = string

    def sub1():
        print(string)

    return sub1


func = Bigsub('AEIOU')
func()
```