CMPSC 461: Programming Language Concepts
Assignment 4 Solution

Problem 1 [5pt] Give an example in a programming language that you're familiar with in which a variable is alive but not in scope.

Solution: One example in C. After the function exists, the object that C links to is no longer in scope, but it is alive.

```
1  void foo()
2  {
3    myClass* C = new myClass();
4    return;
5  }
```

Problem 2 [16pt] Consider the following pseudo code. Assume that the language has one global scope, one scope per function, and one scope for each braced code block.

```
int x = 10;
int tom(int x) {
    {
        int x=50;
        jerry();
    }
}
int jerry() {
    print x+8;
}
tom(6);
```

a) (8pt) Draw the symbol tables in separation for all 4 scopes in this program. Assume that each table contains two columns: name and type.

Solution:
The global scope (Table A):

Name	Type
x	int
tom	fun (or int→int)
jerry	fun (or void→int)

The scope of function tom (Table B):

Name	Type
x	int

The nested scope in function tom (Table C):

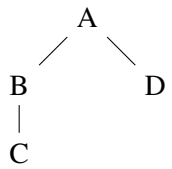
Name	Type
x	int

The scope in function jerry (Table D):

Name	Type

b) (4pt) If the language uses static scoping rules, what's the expected output from the print statement? Justify your answer by showing the hierarchy of the symbol tables at the print statement.

Solution:
The program prints out 18, using x in Table A. The hierarchy of tables is as follows:



c) (4pt) If the language uses dynamic scoping rules, what's the expected output from the print statement? Justify your answer as you did in Problem 2 b).

Solution:
The program prints out 58, using x in Table C. The hierarchy of tables is as follows:
A
|
B
|
C
|
D

Problem 3 [10pt] Consider the following pseudo-code:

```
1  int x = 2;
2
3  function Fire(f) {
4      int x = 40;
5      f();
6  }
7
8  function Earth() {
9      print x;
10 }
11
12 function Water() {
13     int x=20;
14     Fire(Earth);
15 }
16
17 Water();
```

a) (5pt) What would it print if this language uses dynamic scoping and shallow binding? Justify your answer by showing the hierarchy of symbol tables at the print statement (assume that each table contains two columns: name and type).

Solution:
The global scope (Table A):

Name	Type
x	int
Fire	func→void
Earth	void→void
Water	void→void

The scope of function Fire (Table C):

Name	Type
f	func
x	int

The scope of function Water (Table B):

Name	Type
x	int

The scope of function Earth (Table D):

Name	Type

The hierarchy when Earth is called:
A
|
B
|
C
|
D
So the program will print out the "x" in function Fire, which is 40.

b) (5pt) What would it print if this language uses dynamic scoping and deep binding? Justify your answer by explaining at line 14, which symbol table will be passed to function `Fire` in a function closure.

The symbol table passed at line 14 is Table B (the one of Water). So the program will print out the "x" in function Water, which is 20.

Problem 4 [11pt] Consider the following class instances in a C++ program:

```
1  static myClass A;
2
3  int main()
4  {
5    myClass *B = foo();
6    ...
7    delete B;
8    return 0;
9  }
10
11 myClass* foo()
12 {
13   myClass D;
14   myClass* C = new myClass();
15   return C;
16 }
```

a) (5pt) What is the storage allocation (static/stack/heap) for the `myClass` objects that are reference to by A, C and D? How about the storage for the pointers `B` and `C`?

Solution:
A is static allocated, C is heap allocated, and D is stack allocated. The pointers B and C are stack allocated.

b) (6pt) Consider one execution of the program above. The execution trace, a sequence of program statements executed at run time, of this program is
`5 13 14 15 6 7 8`

For each `myClass` object that is reference to by A, C and D, write down its lifetime (use a subset of execution trace, e.g., `6 7 8` to represent the lifetime).

Solution:
Lifetime of A: `5 13 14 15 6 7 8`
Lifetime of C: `14 15 6 7` or `14 15 6`
Lifetime of D: `13 14 15`

Problem 5 [8pt] Consider the following pseudo-code with nested functions. Draw a picture of the run-time stack when control reaches the end of the last call to mouse (i.e., the control reaches line 5, via the call at line 7, via the call at line 13). Include dynamic and static links and storage for local variables and parameters. You do not have to show the storage for any other control information, or temporaries. But you need to specify execution frames for each function.

```
1  int p = 3;
2  int q = 2;
3  int cat(int b, int c) {
4      int mouse(int n) {
5          return n+1;
6      }
7      b = 2 * mouse(c) + b;
8      return b;
9  }
10
11 int dog(int a) {
12     int b = 5;
13     return cat(b, a);
14 }
15
16 print dog(cat(p,q));
```

Solution: