CMPSC 461: Programming Language Concepts, Spring 2024
Assignment 3 Practice Notes Packet

Prof. Suman Saha

September 15, 2025

Problem 1: Scopes and Bindings: Knowledge Check

Answer each question below;

1. For a non-static local variable in a C function, what is its scope and what is its lifetime?
2. For a static local variable in a C function, what is its scope and what is its lifetime?
3. For a non-static global variable in a C function, what is its scope and what is its lifetime?
4. For a static global variable in a C function, what is its scope and what is its lifetime?

Problem 2: Static and Dynamic Scoping

1. What determines an object's lifetime?
2. How do we keep track of what's visible and where in a program? Name the type of structure used and how it stores the information.
3. What are the key differences between lexical and dynamic scoping?
4. What causes the dynamic scoping symbol table to change after compilation?

Problem 3: Nested Scopes and Links

Consider the following pseudo-code, assuming nested subroutines and static scoping.

```
1  main() {
2      int a = 5;
3      int b = 3;
4      function A(int x) {
5          int b = x + 1;
6          B(b);
7      }
8      function B(int y) {
9          int c = a + b + y;
10         print c;
11     }
12     function C(int z) {
13         int a = z + b - 1;
14         B(b);
15     }
16 A(1);
17 C(2);
18 }
```

1. What does the program print?
2. Draw a diagram of the runtime stack when function `B` is last called. For each frame, show the static and dynamic links.
3. Refer to the runtime stack, briefly explain how function `B` finds variable `a and b`.
4. What does the program print when dynamic scoping is used?

Problem 4: Nested Scopes and Links

Consider the following pseudo-code, assuming nested subroutines and static scoping:

```
1  main() {
2      int g = 46;
3      int x = 61;
4      function Z(int a) {
5          int x = a * 3;
6          S(x);
7      }
8      function M(int n) {
9          int g = n
10         if(n % 2 == 0){
11             C(n / 2) ;
12         }
13     }
14     function P(int r) {
15         print x;
16         M(r);
17     }
18     function S(int k){
19         int q = k - 8;
20         M(q);
21         print k;
22     }
23     function C(int l){
24         int x = l;
25         print g;
26         P(l);
27     }
28     // body of main
29     Z(10);
30     print g;
31 }
```

1. What does the program print?
2. Draw a diagram of the runtime stack when function `M` is last called. For each frame, show the static and dynamic links.
3. Refer to the runtime stack, briefly explain how function `P` finds variable `x`.

Problem 5: Nested Scopes and Links

Consider the following pseudo-code, assuming nested subroutines and dynamic scoping:

```
1  main() {
2      int a = 1738;
3      int b = 135;
4      function good(int c) {
5          function luck(int d) {
6              int h = d + 4;
7              print h;
8          }
9          function have(int e) {
10             function fun(int f){
11                 int h = f / 2;
12             }
13             a = e * h;
14             print a;
15             fun(e);
16         }
17         int h = c + 3;
18         print a;  //// Should give 0
19         luck(c - 1);
20         print h;
21         have(h);
22         print h;
23     }
24     function fifty_fifty(int g){
25         int a = 0;
26         good(g / 5);
27         print a;
28     }
29     // body of main
30     fifty_fifty(b);
31     print a;
32 }
```

1. What does the program print?
2. Draw a diagram of the runtime stack when function `fun` is last called. For each frame, show the static and dynamic links.
3. Refer to the runtime stack, briefly explain how function `fun` finds variable `a`.

Problem 6: Binding rules

Consider following pseudo-code, assuming dynamic scoping rules:

```
1  procedure main()
2    x:int := 4
3    y:int := 6
4
5    procedure four()
6      x := x * y
7      print(x)
8
9    procedure six(f: procedure)
10     x:int := 5
11     four()
12
13   procedure one()
14     y:int := 1
15     six(four)
16
17   // main body
18   one()
19   print(x)
```

1. If the language uses deep binding, what will the output be? Explain your answer.
2. If the language uses shallow binding, what will the output be? Explain your answer.

Problem 7: Binding rules

Consider following pseudo-code, assuming dynamic scoping rules:

```
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();
```

1. If the language uses **shallow binding**, what will the output be? Justify your answer by showing the hierarchy of symbol tables at the print statement (assume each symbol table contains two columns: name and kind).

2. If the language uses **deep binding**, what will the output be? Justify your answer by explaining at **line 14**, which symbol table will be passed to function **Fire**.

Problem 8: Scoping and Binding

Consider the following pseudo-code with higher-order function support. Assume that the language has one scope for each function, and it allows nested functions (hence, nested scopes) :

```
1  function A () {
2      int x = 5;
3      function C (P) {
4          int x = 3;
5          P();
6      }
7      function D () {
8          print x;
9      }
10     function B () {
11         int x = 4;
12         C (D);
13     }
14     B ();
15 }
```

1)What would the program print if this language uses dynamic scoping and shallow binding? Justify your answer by showing the tree of symbol tables when execution reaches the display expression.

2)What would the program print if this language uses dynamic scoping and deep binding? Justify your answer using the tree of symbol tables when execution reaches the display expression.

Problem 9: Object Lifetime Tracing

Consider the following C++ code :

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

Consider one execution of the above program. The execution trace (a sequence of program statements executed at run time) of the program is

3 4 5 6 12 13 7 8

For each object associated with A, B, C, D and E, write down its lifetime using a subset of the above execution trace (e.g., "4 5 6 12 13"). Note, the answer subset might be non-strict, i.e. the whole trace.

Problem 10: Static and Dynamic Scoping

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.

```
1  int x = 10;
2  int tom(int x) {
3      {
4          int x=50;
5          jerry();
6      }
7  }
8  int jerry() {
9      print x+8;
10 }
11 tom(6);
```

1. If the language uses static scoping rules, what's the expected output from the print statement? Justify your answer.

2. If the language uses dynamic scoping rules, what's the expected output from the print statement? Justify your answer.

Problem 11: Static and Dynamic Scoping

Consider the following pseudo code:

```
1  int a;
2  int funcA(){
3      int b, c;
4      ...
5      {
6          int d, e;
7          ...
8      }
9      int f;
10     ...
11     {
12         ...
13         int g, h;
14         ...
15     }
16 }
17 int funcB(){
18     int i, j;
19     ...
20     {
21         int k, l;
22         ...
23     }
24     int m;
25 }
```

1. Draw a symbol table for each scope with entry consisting of name, type and data type. Include the global scope.
2. Order the tables in an tree based structure showcasing the hierarchy of scoping.

Problem 12: Static and Dynamic Scoping

Consider the following pseudo code:

```
1     int a = 50;
2     int b = 60;
3     int swap () {
4         int temp = a;
5         a = b;
6         b = temp;
7         print(a, b);
8     }
9     int main () {
10        int a = 10;
11        int b = 20;
12        swap();
13        print(a,b);
14    }
```

1. After executing main, what is the output of this code with static scoping?
2. After executing main, what is the output of this code with dynamic scoping?
3. What function scoping will *swap* use for variables a and b after *main* is called with dynamic scoping?