**Assignment 4**

Problem set questions:

**Chapter 3**

**24. Compute the weakest precondition for each of the following sequences of assignment statements and their postconditions:**

**a)**

a = 2 \* b + 1;

b = a - 3

{b < 0}

Replacing b on second statement

{a - 3 < 0}

{a < 3}

Therefore {a < 3} is our weakest precondition for second statement, now we use it for as a post condition for the first assignment statement

2 \* b + 1 < 3}

{2 \* b < 2}

{b < 1}

Therefore **{b < 1}** is the weakest precondition of the two statements

**b)**

a = 3 \* (2 \* b + a);

b = 2 \* a - 1

{b > 5}

Again replacing b on second statement

{2 \* a - 1 > 5}

{2 \* a > 6}

{a > 3}

Therefore {a > 3} is the weakest precondition for the second statement, now we use it for as a post condition for the first assignment statement

{3 \* (2 \* b + a) > 3}

{2 \* b + a > 1}

{a > 1 - 2 \* b}

Therefore **{a > 1 - 2 \* b}** is the weakest precondition of the two statements

**25. Compute the weakest precondition for each of the following selection constructs and their postconditions:**

**a)**

**if** (a == b)

b = 2 \* a + 1

**else**

b = 2 \* a;

{b > 1}

Solving precondition {b > 1}

2 \* a + 1 > 1

a > 0 for first statement (if)

2 \* a > 1

a > ½ for second statement (else)

Since a > ½ implies that a > 0 but not the other way around. It is correct to choose **a > 1/2** as the weakest precondition

**b)**

**if** (x < y)

x = x + 1

**else**

x = 3 \* x

{x < 0}

Solving precondition {x < 0}

x + 1 < 0

x < -1 for first statement (if)

3 \* x < 0

x < 0 for second statement (else)

Since x < -1 already implies that x < 0 then we choose **x < -1** as the weakest precondition

**c)**

**if** (x > y)

y = 2 \* x + 1

**else**

y = 3 \* x - 1;

{y > 3}

Solving precondition {y > 3}

2 \* x + 1 > 3

x > 1 for first statement (if)

3 \* x – 1 > 3

x > 4/3 for second statement (else)

Since x > 4/3 already implies that x > 1 then the weakest precondition is **x > 4/3**

**28. Prove the following program is correct:**

{n > 0}

count = n;

sum = 0;

**while** count <> 0 **do**

sum = sum + count;

count = count - 1;

**end**

{sum = 1 + 2 + . . . + n}

For this loop the invariant would be:

I: {sum = n + (n-1) + … + (count + 2) + (count + 1)}

* **First Rule: Show P=> I**

P is the pre-conditions:

n > 0

sum = 0

count = n;

As, I: {sum = n + (n-1) + … + (count + 2) + (count + 1)}

Put the preconditions in I

I: {0 = n + (n-1) +…+ (n+2) + (n+1)}

Or

It can write

I: { 0 = }

To prove the rule P => I is correct, prove

n + (n-1) + … + (count + 2) + (count + 1) = 0 when count = n

**Proof:**

n + (n-1) + … + (count + 2) + (count + 1)

**Add and subtract (count + (count -1) + …. + 1):**

n + (n-1) + … + (count + 2) + (count + 1) + (count + (count – 1) + … + 1) – (count + (count – 1) + … + 1)

We can write

1+2+…(count -1) + count + (count + 1) + (count + 2) + … + (n-1) + n =

Hence,

– (count + (count – 1) + …. + 1)

Put, count = n,

(count + (count – 1) + …. + 1) = n + (n-1) + … + 1 =

Thus

= 0

**Rule 1 is satisfied**

**Rule 2: {I and B} S {I}**

As,

Sum = n + (n-1) + …+ (count+1)

P : {sum + count = n + (n-1) + … + (count+1) + count}

* I: {sum = n + (n-1) + … + (count + 1)}

For the statement:

Sum = sum + count;

Let Q is the post condition and P is the Pre-condition.

Q : {sum = n + (n-1) + … + (count+1) + count}

P : {sum = n + (n-1) + … + ((count-1)+2) + ((count-1)+1)

* {sum = n + (n-1) +… + (count+1) + count}

Hence, the rule is correct for the statement Sum = sum + count;

For the statement;

Count = count -1;

Put count = count – 1; in

P : {sum = n + (n-1) + … + ((count-1)+2) + ((count-1)+1)

P : {sum = n + (n-1) + … + (count+2) + (count+1)

So, this statement also satisfies the second rule.

Third rule: (I and (not B)) => Q

As,

Q : {sum = 1 + 2+ … + n}

I : { sum = n + (n-1) + …+ (count+2) + (count+1)

Not(B):(count=0) => ( sum = n + (n-1) + …+ 2 +1)

Hence,

( I and (not B)) => Q third rule is also satisfied.

**Rule 4: loop must terminate.**

Since

1) Before the while loop starts

n>0 and count = n

Hence, count > 0

2) In the while loop, each iteration decrements count by 1, So count’s value will become 0 after n iterations and loop will terminates.

Hence, rule 4 is also satisfied.

Thus the program is correct.

**Chapter 5**

**3. Write a simple assignment statement with one arithmetic operator in some language you know. For each component of the statement, list the various bindings that are required to determine the semantics when the statement is executed. For each binding, indicate the binding time used for the language**

For this example I chose an assignment statement with one arithmetic operator in C:

int a;

a = a + 10

|  |  |
| --- | --- |
| Various Binding Required | Binding time |
| Data type of “a” | Compile time |
| Set of possible values of “a” | Compiler design time |
| Operator + | Compile time |
| Internal representation of literal 10 | Compiler design time |
| Value of “a” | Execution time with a = a + 10 |

**6. Consider the following JavaScript skeletal program:**

// The main program

var x;

function sub1() {

var x;

function sub2() {

. . .

}

}

function sub3() {

. . .

}

Assume that the execution of this program is in the following unit order:

main calls sub1

sub1 calls sub2

sub2 calls sub3

a. Assuming static scoping, in the following, which declaration of x is the correct one for a reference to x?

i. sub1

ii. sub2

iii. sub3

I believe that the right answer would be sub 3, because in static scooping we start at the deepest of the nested programs calls and the deepest is sub 3

b. Repeat part a, but assume dynamic scoping.

1. In static scoping:
   1. In sub 1 is the var x inside sub 1
   2. In sub 2 is the var x inside sub 1
   3. In sub 3 is the var x in main
2. In dynamic scoping:
   1. In sub 1 is the var x inside sub 1
   2. In sub 2 is the var x inside sub 1
   3. In sub 3 is the var x inside sub 1

**7. Assume the following JavaScript program was interpreted using static-scoping rules. What value of x is displayed in function sub1? Under dynamic-scoping rules, what value of x is displayed in function sub1?**

**var** x;

**function** sub1() {

document.write("x = " + x + "<br />");

}

**function** sub2() {

**var** x;

x = 10;

sub1();

}

x = 5;

sub2();

In static scoping **x = 5;**

In dynamic scoping **x = 10;**

**9. Consider the following Python program:**

x = 1;

y = 3;

z = 5;

**def** sub1():

a = 7;

y = 9;

z = 11;

. . .

**def** sub2():

**global** x;

a = 13;

x = 15;

w = 17;

. . .

**def** sub3():

**nonlocal** a;

a = 19;

b = 21;

z = 23;

. . .

. . .

List all the variables; along with the program units where they are declared, that are visible in the bodies of sub1, sub2, and sub3, assuming static scoping is used.

In sub1:

a = 7 sub1

y = 9 sub1

z = 11 sub1

x = 1 main

In sub2:

a = 13 sub2

x = 15 sub2

w = 17 sub2

y = 3 main

z = 5 main

In sub3:

a = 19 sub3

b = 21 sub3

z = 23 sub3

w = 17 sub2

x = 15 sub2

y = 3 main

**10. Consider the following C program:**

**void** fun(**void**) {

**int** a, b, c; /\* definition 1 \*/

. . .

**while** (. . .) {

**int** b, c, d; /\*definition 2 \*/

. . . <------------ 1

**while** (. . .) {

**int** c, d, e; /\* definition 3 \*/

. . . <-------------2

}

. . . <------------3

}

. . . <------------4

}

For each of the four marked points in this function, list each visible variable, along with the number of the definition statement that defines it.

Answer:

**At point 1:**

a (definition 1)

b, c and d (definition2)

**At point 2:**

a (definition 1)

b (definition 2)

c, d, e (definition 3)

**At point 3:**

a (definition 1)

b, c and d (definition2)

**At point 4:**

a, b and c (definition1)

**11. Consider the following skeletal C program:**

**void** fun1(**void**); /\* prototype \*/

**void** fun2(**void**); /\* prototype \*/

**void** fun3(**void**); /\* prototype \*/

**void** main() {

**int** a, b, c;

. . .

}

**void** fun1(**void**) {

**int** b, c, d;

. . .

}

**void** fun2(**void**) {

**int** c, d, e;

. . .

}

**void** fun3(**void**) {

**int** d, e, f;

. . .

}

Given the following calling sequences and assuming that dynamic scoping is used, what variables are visible during execution of the last function called? Include with each visible variable the name of the function in which it was defined.

a. main calls fun1; fun1 calls fun2; fun2 calls fun3.

b. main calls fun1; fun1 calls fun3.

c. main calls fun2; fun2 calls fun3; fun3 calls fun1.

d. main calls fun3; fun3 calls fun1.

e. main calls fun1; fun1 calls fun3; fun3 calls fun2.

f. main calls fun3; fun3 calls fun2; fun2 calls fun1.

**Answer:**

**(a) main calls fun1; fun1 calls fun2; fun2 calls fun3;**

Variables which are visible during their last function called: -

*d*, *e*, and *f* are defined in fun3;

*c* is defined in fun2;

*b* is defined in fun1;

*a* is defined in main;

**(b) main calls fun1; fun1 calls fun3;**

Variables which are visible during their last function called: -

*d*, *e*, and *f* are defined in fun3;

*b* and *c* are defined in fun1;

*a* is defined in main;

**(c) main calls fun2; fun2 calls fun3; fun3 calls fun1;**

Variables which are visible during their last function called: -

*b*, *c* and *d* are defined in fun1;

*e*, and *f* are defined in fun3;

*a* is defined in main;

**(d) main calls fun3; fun3 calls fun1;**

Variables which are visible during their last function called: -

*b*, *c* and *d* are defined in fun1;

*e*, and *f* are defined in fun3;

*a* is defined in main;

**(e) main calls fun1; fun1 calls fun3; fun3 calls fun2;**

Variables which are visible during their last function called: -

*c*, *d*, and *e* are defined in fun2;

*f* is defined in fun3;

*b* is defined in fun1;

*a* is defined in main;

**(f) main calls fun3; fun3 calls fun2; fun2 calls fun1;**

Variables which are visible during their last function called: -

*b*, *c* and *d* are defined in fun1;

*e* is defined in fun2;

*f* is defined in fun3;

*a* is defined in main;

**12. Consider the following program, written in JavaScript-like syntax:**

// main program

**var** x, y, z;

**function** sub1() {

**var** a, y, z;

. . .

}

**function** sub2() {

**var** a, b, z;

. . .

}

**function** sub3() {

**var** a, x, w;

. . .

}

Given the following calling sequences and assuming that dynamic scoping is used, what variables are visible during execution of the last subprogram activated? Include with each visible variable the name of the unit where it is declared.

a. main calls sub1; sub1 calls sub2; sub2 calls sub3.

b. main calls sub1; sub1 calls sub3.

c. main calls sub2; sub2 calls sub3; sub3 calls sub1.

d. main calls sub3; sub3 calls sub1.

e. main calls sub1; sub1 calls sub3; sub3 calls sub2.

f. main calls sub3; sub3 calls sub2; sub2 calls sub1.

**Answer:**

**(a) main calls sub1; sub1 calls sub2; sub2 calls sub3.**

Variables which are visible during their last function called: -

*a*, *x,* and *w* are defined in sub3;

*b, z* are defined in sub2;

*y* is defined in sub1;

**(b) main calls sub1; sub1 calls sub3.**

Variables which are visible during their last function called: -

a, x, and w, in sub3;

y, and z in sub1.

**(c) main calls sub2; sub2 calls sub3; sub3 calls sub1.**

Variables which are visible during their last function called: -

a, y, and z in sub1;

x, and w in sub3;

b in sub2;

**(d) main calls sub3; sub3 calls sub1.**

Variables which are visible during their last function called: -

a, y, and z in sub1;

x, and w in sub3;

**(e) main calls sub1; sub1 calls sub3; sub3 calls sub2.**

Variables which are visible during their last function called: -

a, b, and z in sub2;

x, and w in sub3;

y in sub1.

**(f) main calls sub3; sub3 calls sub2; sub2 calls sub1.**

Variables which are visible during their last function called: -

a, y, and z in sub1;

b in sub2;

x, and w in sub3;