Exam 2

Graded

Student

Adam Fenjiro

Total Points

42.5 / 70 pts

Question 1

Q1 — **4** / 4 pts

1.1 a — **2** / 2 pts

✔ **– 0 pts** Correct

– 1 pt Parentheses correct, result missing/incorrect

– 2 pts Incorrect (correct answer is 25)

1.2 b — **2** / 2 pts

✔ **– 0 pts** Correct

– 1 pt Parentheses correct, result missing/incorrect

– 2 pts Incorrect (correct answer is -11)

Question 2

Q22 — **3** / 3 pts

2.1 a — **1** / 1 pt

✔ **– 0 pts** Correct

– 1 pt Incorrect (Correct answer is 10)

2.2 b — **1** / 1 pt

✔ **– 0 pts** Correct

– 1 pt Incorrect (Correct answer: -44)

2.3 c — **1** / 1 pt

✔ **– 0 pts** Correct

– 0.5 pts 4 bytes off (Correct answer: -28)

– 1 pt Incorrect (Correct answer: -28)

Question 3

Q3 **4** / 4 pts

✔ **– 0 pts** Correct

– 1 pt a incorrect (should be 4)

– 1 pt b incorrect (should be 5)

– 1 pt c incorrect (should be 4)

– 1 pt d incorrect (should be 5.0)

Question 4

Q4 **3** / 3 pts

4.1 a **1** / 1 pt

✔ **– 0 pts** Correct

– 1 pt Incorrect

4.2 b **1** / 1 pt

✔ **– 0 pts** Correct

– 1 pt Incorrect

4.3 c **1** / 1 pt

✔ **– 0 pts** Correct

– 1 pt Incorrect

Question 5

Q5 **3** / 5 pts

5.1 **a** **3** / 3 pts

✔ **– 0 pts** Correct

– 1 pt Missing two required labels / jump or goto style instructions

– 1 pt Missing or incorrect (a <= -100) (or equivalent) check

– 1 pt Missing or incorrect (a > 50) (or equivalent) check

– 3 pts Incorrect or not attempted

5.2 **b** **0** / 2 pts

– 0 pts Correct

– 0.5 pts andl (or equivalent) instruction incorrect

– 0.5 pts Comparison prep instruction incorrect (i.e. movl $1, ...), also acceptable to use a preexisting register with correct value or a literal in cmpl

– 0.5 pts Comparison instruction incorrect (cmpl or equivalent)

– 0.5 pts Jump instruction is incorrect

✔ **– 2 pts** Incorrect or not attempted

Question 6

Q6 **3** / 6 pts

6.1 a **1** / 1 pt

✔ **– 0 pts** Correct

– 1 pt Incorrect (should be 200)

6.2 b **2** / 2 pts

✔ **– 0 pts** Correct

– 1 pt `a[-5][4]` is guaranteed to be next to `a[-5][5]` in memory

– 1 pt `a[-4][9]` is NOT guaranteed to be next to `a[-3][-10]`

– 2 pts Incorrect or not attempted

6.3 c **0** / 3 pts

– 0 pts Correct

✔ **– 0.5 pts** Line 2 ($-88)

✔ **– 0.5 pts** Line 6 ($-5)

✔ **– 0.5 pts** Line 7 ($8)

✔ **– 0.5 pts** Line 10 (movq (%rax), %rax)

✔ **– 0.5 pts** Line 14 ($-10)

✔ **– 0.5 pts** Line 17 (addq %rcx, %rax)

– 3 pts Not attempted

Question 7

Q7 **6** / 8 pts

7.1 **a** **1** / 1 pt

✔ **– 0 pts** Correct

– 1 pt Incorrect (400 bytes)

7.2 **b** **1** / 1 pt

✔ **– 0 pts** Correct

– 1 pt Incorrect (408 bytes)

7.3 **c** **0** / 1 pt

– 0 pts Correct

✔ **– 1 pt** Incorrect (-416)

7.4 **d** **0** / 1 pt

– 0 pts Correct

✔ **– 1 pt** Incorrect (-408)

7.5 **e** **2** / 2 pts

✔ **– 0 pts** Correct

– 0.5 pts Missing $\times 4$ only

– 0.5 pts Minor arithmetic error

– 1 pt Major part of equation incorrect

– 2 pts Incorrect $((i-1) \times (6-2+1) + (j-2)) \times 4$, simplified is OK

7.6 **f** **2** / 2 pts

✔ **– 0 pts** Correct

– 0.5 pts Missing $\times 4$ only

– 0.5 pts Minor arithmetic error

– 1 pt Major part of equation incorrect

– 2 pts Incorrect $((j-2) \times (20-1+1) + (i-1)) \times 4$, simplified is OK

Question 8

Q8

6 / 6 pts

✔ **– 0 pts** Correct

– 1 pt Jump table declaration incorrect

– 0.5 pts $(a < 300)$ check incorrect

– 0.5 pts $(a > 304)$ check incorrect

– 1 pt Jump to table index incorrect

– 0.5 pts Jump to table index has incorrect offset

– 0.5 pts Missing break after clause 301

– 0.5 pts Missing break after clause303&304

– 0.5 pts Missing default label

– 0.5 pts Missing after switch label

– 0.5 pts Incorrect breaks in unintended positions

– 5 pts Not a jump table implementation (but otherwise reasonably correct)

– 6 pts Incorrect or not attempted

Question 9

Q9 **3** / 5 pts

9.1 **a** **0** / 1 pt

– **0 pts** Correct

– **0.5 pts** Dynamic Link incorrect (should point to AR of staticscope)

– **0.5 pts** Static Link incorrect (should point to AR of staticscope)

✔ – **1 pt** Incorrect

9.2 **b** **1** / 2 pts

– **0 pts** Correct

– **1 pt** x incorrect (refers to local of h)

✔ – **1 pt** z incorrect (refers to z of staticscope)

– **2 pts** Incorrect

9.3 **c** **1** / 1 pt

✔ – **0 pts** Correct

– **0.5 pts** x incorrect (x = 6)

– **0.5 pts** y incorrect (y = 7)

– **1 pt** Incorrect

9.4 **d** **1** / 1 pt

✔ – **0 pts** Correct

– **1 pt** Incorrect (should be 29)

Question 10

Q10 **3** / 8 pts

10.1 **a: b= (by value)** **0.5** / 0.5 pts

✔ **– 0 pts** Correct

– 0.5 pts Incorrect (should be 2)

10.2 **a: b= (by reference)** **0** / 0.5 pts

– 0 pts Correct

✔ **– 0.5 pts** Incorrect (should be 10)

10.3 **a: b= (by value-result)** **0** / 0.5 pts

– 0 pts Correct

✔ **– 0.5 pts** Incorrect (should be 2)

10.4 **a: b= (by name)** **0** / 0.5 pts

– 0 pts Correct

✔ **– 0.5 pts** Incorrect (should be 10)

10.5 **a: c= (by value)** **0.5** / 0.5 pts

✔ **– 0 pts** Correct

– 0.5 pts Incorrect (should be 5)

10.6 **a: c= (by reference)** **0** / 0.5 pts

– 0 pts Correct

✔ **– 0.5 pts** Incorrect (should be 5)

10.7 **a: c= (by value-result)** **0.5** / 0.5 pts

✔ **– 0 pts** Correct

– 0.5 pts Incorrect (should be 5)

10.8 **a: c= (by name)** **0** / 0.5 pts

– 0 pts Correct

✔ **– 0.5 pts** Incorrect

10.9 **a: a= (by value)** **0.5** / 0.5 pts

✔ **– 0 pts** Correct

– 0.5 pts Incorrect (should be 10)

10.10 **a: a= (by reference)** **0** / 0.5 pts

– **0 pts** Correct

✔ – **0.5 pts** Incorrect (should be 15)

10.11 **a: a= (by value-result)** **0** / 0.5 pts

– **0 pts** Correct

✔ – **0.5 pts** Incorrect (should be 15)

10.12 **a: a= (by name)** **0** / 0.5 pts

– **0 pts** Correct

✔ – **0.5 pts** Incorrect (should be 15)

10.13 **b** **1** / 2 pts

– **0 pts** Correct

– **1 pt** Doesn't mention that (a+b) gets evaluated in place of c

✔ – **1 pt** Doesn't correctly indicate what a and b in the (a+b) expression would be bound to (both would be bound to globals)

– **2 pts** Incorrect or not attempted

Question 11

Q11 **4** / 12 pts

11.1 **a** **2** / 3 pts

– **0 pts** Correct

– **2 pts** Incorrect symbol table structure (should be three layers, from bottom to top: staticscope, h, and g).

✔ – **1 pt** Offsets incorrect

– **3 pts** Incorrrect or not attempted

11.2 **b** **1** / 2 pts

– **0 pts** Correct

– **1 pt** AL incorrect (points to AR of h)

✔ – **1 pt** DL incorrect (points to AR of f)

– **2 pts** Incorrect

11.3 **c** **0** / 1 pt

– **0 pts** Correct

✔ – **1 pt** Incorrect (points to AR of h)

11.4 **d** **0** / 1 pt

– **0 pts** Correct

✔ – **1 pt** Incorrect (Points to z@staticscope)

11.5 **e** **0** / 2 pts

– **0 pts** Correct

– **0.5 pts** Incorrect move to the AR of h (movq -8(%rbp), %rax)

– **0.5 pts** Incorrect move to the AR of staticscope (movq -8(%rax), %rax)

– **1 pt** Incorrect load z instruction (movl -20(%rax), %eax)

– **0.5 pts** Incorrect offset in load z

✔ – **2 pts** Incorrect or not attempted

– **0 pts** Note: Using a different register instead of %rax/%eax is OK. I also accepted treating z as a global variable like we did in the project.

11.6 **f** **1** / 1 pt

✔ – **0 pts** Correct

– **1 pt** Incorrect (y refers to parameter y of h)

11.7 **g** **0** / 1 pt

– **0 pts** Correct

– **0.5 pts** Incorrect move to h's AR (movq -8(%rbp), %rax)

– **0.5 pts** Incorrect load to register (movl 8(%rax), %eax)

✔ – **1 pt** Incorrect or not attempted

11.8 **h** **0** / 1 pt

– **0 pts** Correct

✔ – **1 pt** Incorrect (should be 55)

Question 12

Q12 **0.5** / 6 pts

12.1 **a** **0.5** / 3 pts

– **0 pts** Correct

✔ – **1 pt** x's binding incorrect (bound to x@binding)

✔ – **1 pt** y's binding incorrect (bound to y@first)

✔ – **0.5 pts** First output value incorrect (should be 9)

– **0.5 pts** Second output value incorrect (should be 20)

– **3 pts** Incorrect or not attempted

12.2 **b** **0** / 3 pts

– **0 pts** Correct

✔ – **1 pt** x's binding incorrect (bound to x@second)

✔ – **1 pt** y's binding incorrect (bound to y@second)

✔ – **0.5 pts** First output value incorrect (should be 11)

✔ – **0.5 pts** Second output value incorrect (should be 5)

– **3 pts** Incorrect or not attempted

isilioglu, G

SCAN

CS4121 Exam #2
April 7, 2025

Name: Adam FENJIRO User ID: afenjiro
(User ID is your Michigan Tech email ID. For example, put in *joe* if your email address is jeo@mtu.edu.)

1. (4 pts.) Parenthesizing an expression can enforce operator precedence and associativity. For example, 2+5*4 can be parenthesized as (2+5)*4 if + takes higher precedence over *. Evaluate the expression `6 + 5 * 4 - 3 - 2` using the following precedence and associativity schemes where the precedence goes from **lowest** to **highest**. You may assume that the operators are left associative unless stated otherwise. Show your work by **parenthesizing** the expression based on the rules in the table.

Precedence	Scheme 1	Scheme 2
lowest	+	*
	- (right associative)	+
highest	*	-

(a) (2 pts.) Scheme 1:

$6+[(5*4) - (3-2)] = 25$

(b) (2 pts.) Scheme 2:

$[(6+5) * [(4-3)-2]] = -11$

2. (3 pts.) Consider the following local variable declarations and reference.

```
VAR  i: INTEGER;
     a: ARRAY [-4..5] OF INTEGER;
     j: INTEGER;

     a[i] := j;
```

Assume that `i` is stored at offset -4 off of `%rbp`, followed by a and `j` (sizeof(INTEGER) = 4).

(a) (1 pt.) The number of elements of array a is 10 = 5 - (-4) + 1.

(b) (1 pt.) The base offset of array a with respect to `%rbp` is -44.

(c) (1 pt.) The offset of `a[0]` with respect to `%rbp` is -28.

base(A) + (i - low) ~~*28~~ *4.
-44 + (0 - (-4)) 4

3. (4 pts.) The output of the C code below is

a = 4

b = 5

c = 4

d = 5.0

```
int a, b, c;
float d;

b = 9;
a = 2;
a = 1/2 + b/a;
printf("a = %d\n", a);

d = 9;
b = 2;
b = (float)1/2 + d/b;
printf("b = %d\n", b);

c = 9;
a = 2;
c = (float)1/2 + c/a;
printf("c = %d\n", c);

d = 9;
b = 2;
d = 1.0/2 + d/a;
printf("d = %.1f\n", d);
```

4. (3 pts.) In the following code, which of the variables will a compiler consider to have compatible types under structural equivalence, strict name equivalence, and loose name equivalence, respectively?

```
struct SUV {
  int year;
  char make[20];
  double msrp;
};

typedef struct SUV Crossover;

struct Sedan {
  int year;
  char make[20];
  double msrp;
};

typedef struct Sedan CompactSedan;

struct SUV escape;
Crossover crosstour;
Crossover venza;
struct Sedan malibu;
struct Sedan camery;
CompactSedan civic;
```

(a) Structural equivalence:

escape, crosstour, venza, malibu, camery, civic

(b) Strict name equivalence:

crosstour, venza
malibu, camery

(c) Loose name equivalence:

escape, crosstour, venza
malibu, camery, civic.

5. (5 pts.) Consider the `do-while` loop below:

```
do {
   \\loop body
} while ((a > -100 ) && (a < 50))
\\ after loop
```

(a) (3 pts.) Show the code shape if the language supports short-circuit evaluation.

.L1
if(a $\leq$ -100) goto .L2
if(a $\geq$ 50) goto .L2
// loop body

.L2
// after loop

(b) (2 pts) The segment of x86-64 code below implement the loop assuming that the language does not support short-circuit evaluation. The code generator assumes that a is stored in the activation record with offset -4. Fill in the missing instructions. (Note: An answer with less than 4 or more than 4 instructions is acceptable as long as it works.)

```
.L0:  nop
      #loop body
      ...
      movq %rbp,%rbx
      addq $-4, %rbx
      movl (%rbx), %ecx
      movl $-100, %ebx
      cmpl %ebx, %ecx
      movl $0, %ecx
      movl $1, %ebx
      cmovg %ebx, %ecx
      movq %rbp,%rbx
      addq $-4, %rbx
      movl (%rbx), %r8d
      movl $50, %ebx
      cmpl %ebx, %r8d
      movl $0, %r8d
      movl $1, %ebx
      cmovl %ebx, %r8d
      movq %rbp, %rbx
      addq $-4, %rbx
      movl (%rbx), %r8d
      movl $0, %ecx
.L1:  nop
      #after loop
```

6. (6 pts.) Consider the following local variable declarations and reference.

```
VAR  i, j: INTEGER;
     a: ARRAY [-5..4][-10..9] OF INTEGER;

     a[i][j] := 0;
```

Assume that the language implementation applies row-pointer layout, `i` and `j` are stored at offsets -4 and -8, respectively, off of `%rbp`, and the row pointers for `a` should be put into the activation record next to `j` (sizeof(INTEGER) = 4).

(a) (1 pt) How many elements are in array a?

200

#rows = 4 - (-5) + 1 = 10
#columns = 9 - (-10) + 1 = 20
elements = 20 x 10

(b) (2 pts.) Is it guaranteed that a[-5][4] and a[-5][5] are next to each other in memory? How about a[-4][9] and a[-3][-10]?

Yes, it guaranteed that a[-5][4] and a[-5][5] are next to each other, but not a[-4][9] and a[-3][-10].

(c) (3 pts.) Filling the missing instructions/components in the following x86-64 code for the assignment `a[i][j] := 0`. (Hint: read the comments.)

```
1.  movq %rbp, %rax
2.  addq  -12 , %rax    #add the offset of a[-5]
3.  movq %rbp, %rbx
4.  addq $-4, %rbx
5.  movl (%rbx), %ecx
6.  subl  0 , %ecx      #substract base index
7.  imull  200 , %ecx   #multiply element size
8.  movslq %ecx, %rcx
9.  addq %rcx, %rax
10. load -12, %rax      #load a[i], the base address of row i to %rax
11. movq %rbp, %rbx
12. addq $-8, %rbx
13. movl (%rbx), %ecx
14. subl  0 , %ecx      #substract base index
15. imull $4, %ecx
16. movslq %ecx, %rcx
17. load -812, %rax     #address of a[i][j] to %rax
18. movl $0, (%rax)
```

7. (8 pts.) Consider the following local variable declarations and reference.

```
VAR  i,j : INTEGER;
     r: RECORD
          k: INTEGER;
          c: CHAR;
          z: ARRAY [1..20][2..6] OF INTEGER;
        END;

     ... := r.z[i][j];
```

Assume that `i` and `j` are stored at offsets -4 and -8, respectively, off of `%rbp`. The record `r` should be put into the activation record next to `j`. The language also requires 4-byte alignment for integers (sizeof(CHAR) = 1, sizeof(INTEGER) = 4). Record fields must be laid on memory from low address to high address. Field reordering is prohibited in this language. Arrays apply column- or row-major layout.

(a) (1 pt.) What is the size of array `r.z`?

400 byts

#rows = 20-1+1 = 20
#columns = 6-2+1 = 5
#elements = 20 x 5 = 100
#size = 100 x 4

(b) (1 pt.) What is the size of record `r` (you need to consider padding due to alignment)?

408 byts

(c) (1 pt.) What is the base offset of record `r` (i.e. the offset of `r.k`) with respect to `%rbp`?

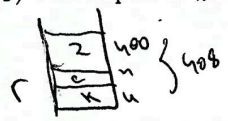
i → -4, j → -8, r.k → -12

-12

(d) (1 pt.) What is the base offset of array `r.z` (i.e. the offset of `r.z[1][2]`) with respect to `%rbp`?

-416

i → -4, j → -8, [illegible] → -416



(e) (2 pts.) Assuming row-major order for arrays, develop an equation in terms of `i` and `j` that calculates the offset of `r.z[i][j]` with respect to `r.z[1][2]`) (No x86-64 code is needed).

Base(0) + [(i - low1)(high2 - low2 + 1) + (j - low2)]4

[illegible] [(i - 1)(5) + (j - 2)]4

(f) (2 pts.) Assuming column-major order for arrays, develop an equation in terms of `i` and `j` that calculates the offset of `r.z[i][j]` with respect to `r.z[1][2]`) (No x86-64 code is needed).

Base(0) + [(j - low2)(high1 - low1 + 1) + (i - low1)]4

[illegible] [(j - 2)(20) + (i - 1)]4

8. (6 points) Consider the following C switch statement. Show the **code shape** of a jump table implementation. Use comments to represent the clauses. (X86-64 code is not needed.)

```
switch(a) {
  case 300: clause300;
  case 301: clause301;
            break;
  case 303:
  case 304: clause303&304;
            break;
  default:  clauseDefault;
}
```

```
.T[5] = {.L300, .L301, .Ld, .L303, .L304}
if(a<300) goto .Ld
if(a>304) goto .Ld
goto .T[a-300]

.L300:
  clause300

.L301:
  clause301
  jmp .Lend

.L303:

.L304:
  clause303&304
  jmp .Lend

.Ld:
  clauseDefault

.Lend:
  //after switch code
```

9. (5 pts.) Consider the following pseudocode using call-by-value parameter passing and static scoping.

```
PROGRAM staticscope;
VAR x, y, z: INTEGER;

  FUNCTION h(y : INTEGER) : INTEGER;
  VAR x: INTEGER;
  BEGIN {* h *}
    x := 6;
    z := y + x;  {* What do x and z refer to? *}
    h := z; {* return value assigned to h. This statement is similar to return z in C *}
  END;

BEGIN {* staticscope *}
  x := 3;
  y := 4;
  z := 5;
  y := h(x+y);
  WRITELN(x+y+z);
END.
```

(a) (1 pt.) When h has just been called, where should the dynamic link and the static link of the activation record of h point to, respectively?

They should both point to the function h.

(b) (2 pts) The execution now reaches z := y + x, which declarations (definitions) do variables x and z refer to, respectively?

x refers to the declaration in function h(), x:=6

y refers to the y param in function h(), which was passed from the main function h(x+y)

(c) (1 pt.) What are the values of x and y, respectively, at this point?

x = 6 , y = 7

(d) (1 pt.) What is the output of this program?

29

10. (8 pts.) Evaluate the following C program using call-by-value, call-by-reference, call-by-value-result and call-by-name semantics.

```
int a = 2;
int b = 3;
int c = 7;

void f(int b, int c, int d) {
  a = a * c;
  printf("b = %d\n", b);
  printf("c = %d\n", c);
  b = d + 11;
}

void main() {
  int c = 4;

  f(a, a+b, c);
  printf("a = %d\n", a);
}
```

(a) (6 pts.) Fill the output in the table below. (No steps are needed here.)

Output Text	Call-by-value	Call-by-reference	Call-by-value-result	Call-by-name
b =	2	2	15	2
c =	5	4	5	10
a =	10	8	10	8

(b) (2 pts.) Under call-by-name, describe how c is evaluated when executing statement `printf("c = %d \n", c)`.

in the main() function, we passe (a+b) as the param int c of the function f().

so basically, inside the function f() we will evaluate every mention of c as (a+b) since it is a call-by-name.

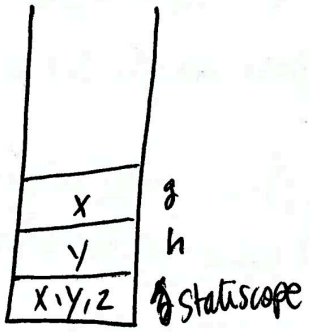
printf("c = %d \n", c) is evaluated as

printf("c = %d \n", (a+b)).

11. (12 pts.) Consider the following pseudocode using call-by-value parameter passing and static scoping. Assume that static link is stored at offset -8 followed by local variables toward low address. Also assume that dynamic link is stored at offset 0 followed by parameters in the declaration order toward high address.

```
PROGRAM staticscope;
VAR x, y, z : INTEGER;

  FUNCTION h(y : INTEGER) : INTEGER;
    FUNCTION g(x : INTEGER) : INTEGER;
    BEGIN {* g *}
      g := x + y + z;  {* return value assigned to g *}
    END;

    FUNCTION f(x, y: INTEGER) : INTEGER;
    VAR z : INTEGER;
    BEGIN {* f *}
      z := y - x + 23;
      f := g(x + z); {* return value assigned to f *}
    END;

  BEGIN {* h *}
    z := x + 11;
    h := f(y, z); {* return value assigned to h *}
  END;

BEGIN {* staticscope *}
  x := 1;
  y := 3;
  z := 5;
  WRITELN(h(y+z));
END.
```

(a) (3 pts.) Show the stacked symbol table when function g is being processed at compilation time. For each variable in the symbol table, list its offset.



(b) (2 pts.) When g has just been called (the execution now enters g), where do the access link and dynamic link of the activation record of g point to, respectively?

prints to h

(c) (1 pt.) At this point, where do the access link of the activation record of `f` point to?

points to g

(d) (1 pt.) At this point, which declaration (definition) does variable `z` in statement `g := x + y + z` refer to?

z refers to the z from the function g()

z = y - x + 23

(e) (2 pt.) Generate x86-64 assembly that loads `z`.

(f) (1 pt.) At this point, which declaration (definition) does variable `y` in statement `g := x + y + z` refer to?

y refers to the y param in the g() function

(g) (1 pt.) Generate x86-64 assembly that loads `y`.

(h) (1 pt.) What is the output of this program?

74

12. (6 pts.) Consider the following pseudocode:

```
PROGRAM binding;
VAR x, y: INTEGER;

  PROCEDURE add();
  BEGIN
    x := x + y;
  END;

  PROCEDURE second(P : PROCEDURE);
  VAR x, y: INTEGER;
  BEGIN
    x := 9;
    y := 2;
    P();
    writeln(x);
  END;

  PROCEDURE first();
  VAR y: INTEGER;
  BEGIN
    y := 3 * x;
    second(add);
  END;

BEGIN
  x := 5;
  y := 1;
  first();
  writeln(x);
END.
```

(a) (3 points) What does it print if the language uses dynamic scope with deep binding? What are x and y in add are bound to, respectively, when add is called?

the program prints 20
in add(), x is 20 and y is 15

(b) (3 points) Repeat (a) assuming the language uses dynamic scope with shallow binding.

the program prints 9
in add(), x is 9 and y is 2.