CS1010 Tutorial 4

Agenda for Today

- Problem Set 11 (25 min)
- Problem Set 12 (15 min)
- Problem Set 13 (15 min)
- Assignment 1 Comments / Marking Scheme (15 min)
- Assignment 2 (Reminders) (15 min)

Problem Set 11

Repeats and gotos

Problem 11.1

```
long factorial(long n)
{
    long i = n - 1;
    long product;
    for (product = n; i >= 2; product *= i) {
        i -= 1;
    }
    return product;
}
```

Does this function run correctly?

Problem 11.1

```
long factorial(long n)
{
    long i = n - 1;
    long product;
    for (product = n; i >= 2;
            product *= i) {
        i -= 1;
    }
    return product;
}
```

Does this function run correctly?

Problem 11.1 – A possible fix

```
long factorial(long n)
{
    long i = n + 1;
    long product;
    for (product = 1; i >= 2; product *= i) {
        i -= 1;
    }
    return product;
}
```

Problem 11.2(a)

- Rewrite the "Guess A Number" program so that it shows the user the number of guesses made before the correct guess is entered.
- Do we need a new variable?
- What kind of variable?

```
int main()
{
    srandom(times(0));
    long answer = (random() % 100) + 1;
    long guess;
    do {
        guess = cs1010_read_long();
        if (guess > answer) {
            cs1010_println_string("too high");
        } else if (guess < answer) {
            cs1010_println_string("too low");
        }
    } while (guess != answer);

    cs1010_println_string("you got it. congrats!");
}
```

Problem 11.2(a)

- Rewrite the "Guess A Number" program so that it shows the user the number of guesses made before the correct guess is entered.
- What values should you put there?

```
int main()
{
    srandom(times(0));
    long answer = (random() % 100) + 1;
    long guess;
    long counter = ___;
    do {
        guess = cs1010_read_long();
        counter += ___;
        if (guess > answer) {
            cs1010_println_string("too high");
        } else if (guess < answer) {
            cs1010_println_string("too low");
        }
    } while (guess != answer);

    cs1010_println_long(counter);
    cs1010_println_string("you got it. congrats!");
}
```

Problem 11.2(a)

- Rewrite the "Guess A Number" program so that it shows the user the number of guesses made before the correct guess is entered.
- Initialise with 0, increment every iteration
- If user guesses on first try, he made 1 guess (correct)
- Otherwise, they make some number of tries, and each try increments by 1

```
int main()
{
    srandom(times(0));
    long answer = (random() % 100) + 1;
    long guess;
    long counter = 0;
    do {
        guess = cs1010_read_long();
        counter += 1;
        if (guess > answer) {
            cs1010_println_string("too high");
        } else if (guess < answer) {
            cs1010_println_string("too low");
        }
    } while (guess != answer);

    cs1010_println_long(counter);
    cs1010_println_string("you got it. congrats!");
}
```

Problem 11.2(b)

- Rewrite the "Guess A Number" program with a **while** loop
- A **do-while** loop guarantees that the loop body is executed at least once
- A **while** loop is a do-while loop without such a guarantee
- **What should you do to this code?**

```
int main()
{
    srandom(times(0));
    long answer = (random() % 100) + 1;
    long guess;
    do {
        guess = cs1010_read_long();
        if (guess > answer) {
            cs1010_println_string("too high");
        } else if (guess < answer) {
            cs1010_println_string("too low");
        }
    } while (guess != answer);

    cs1010_println_string("you got it. congrats!");
}
```

Problem 11.2(b)

- Rewrite the "Guess A Number" program with a **while** loop
- **Option 1**
- Just copy the loop body outside the main do-loop, then change the do-loop directly into a while loop
- This is ugly
  - Code duplication should be avoided
  - How can we improve this?

```
int main()
{
    srandom(times(0));
    long answer = (random() % 100) + 1;
    long guess = cs1010_read_long();
    if (guess > answer) {
        cs1010_println_string("too high");
    } else if (guess < answer) {
        cs1010_println_string("too low");
    }
    while (guess != answer) {
        guess = cs1010_read_long();
        if (guess > answer) {
            cs1010_println_string("too high");
        } else if (guess < answer) {
            cs1010_println_string("too low");
        }
    }
    cs1010_println_string("you got it. congrats!");
}
```

Problem 11.2(b)

- Rewrite the "Guess A Number" program with a **while** loop

- **Option 2**

- Just leave the printing inside

```
int main()
{
    srandom(times(0));
    long answer = (random() % 100) + 1;
    long guess = cs1010_read_long();

    while (guess != answer) {
        if (guess > answer) {
            cs1010_println_string("too high");
        } else if (guess < answer) {
            cs1010_println_string("too low");
        }
        guess = cs1010_read_long();
    }

    cs1010_println_string("you got it. congrats!");
}
```

Problem 11.2(b)

- Rewrite the "Guess A Number" program with a **while** loop
- **Option 3**
- The “idiomatic” C way that utilizes assignment expressions
- **Don’t write code like this in CS1010**
- Just showing as an FYI ☺

```
int main()
{
    srandom(times(0));
    long answer = (random() % 100) + 1;
    long guess;

    while ((guess = cs1010_read_long()) != answer) {
        if (guess > answer) {
            cs1010_println_string("too high");
        } else if (guess < answer) {
            cs1010_println_string("too low");
        }
    }

    cs1010_println_string("you got it. congrats!");
}
```

Problem 11.2(c)

- Extend the "Guess A Number" program so that it plays the game for five rounds with the user, and at the end, shows the user the average number of guesses over five rounds.
- (Hint: you should put the loop that reads the guess and prints feedback to the user into another function.)

```
int main()
{
    srandom(times(0));
    long answer = (random() % 100) + 1;
    long guess;
    do {
        guess = cs1010_read_long();
        if (guess > answer) {
            cs1010_println_string("too high");
        } else if (guess < answer) {
            cs1010_println_string("too low");
        }
    } while (guess != answer);

    cs1010_println_string("you got it. congrats!");
}
```

Problem 11.2(c)

```
count = 0;
for (i = 0; i < 4; i += 1) {
    count += guess_a_number();
}
cs1010_println_double(count / 5.0)
```

```
long guess_a_number()
{
    long counter = 0;
    long answer = (random() % 100) + 1;
    long guess;
    do {
        guess = cs1010_read_long();
        counter += 1;
        if (guess > answer) {
            cs1010_println_string("too_high");
        } else {
            cs1010_println_string("too low");
        }
    } while (guess != answer);
    return counter;
}
```

Problem 11.2(d)

- What is the optimal strategy to play the game?

- Guess 50
- If answer is lower, guess 25
  - If answer is lower, guess 12
  - If answer is higher, guess 37
- If answer is higher, guess 75
  - If answer is lower, guess 62
  - If answer is higher, guess 87

- …etc

```
int main()
{
    srandom(times(0));
    long answer = (random() % 100) + 1;
    long guess;
    do {
        guess = cs1010_read_long();
        if (guess > answer) {
            cs1010_println_string("too high");
        } else if (guess < answer) {
            cs1010_println_string("too low");
        }
    } while (guess != answer);

    cs1010_println_string("you got it. congrats!");
}
```

Problem 11.3(a)

- What is the return value when:
  - $n = 8$ and $k = 2$?
    - 3
  - $n = 81$ and $k = 3$?
    - 4
  - $n = 100$ and $k = 5$?
    - 2

```
long mystery(long n, long k)
{
    long something = n;
    long count = -1;
    while (something >= 1) {
        something /= k;
        count += 1;
    }
    return count;
}
```

Problem 11.3(b)

- What is the mathematical expression that our mystery function here is trying to compute based on the examples above?

- It computes $\lfloor \boldsymbol{\log_k n} \rfloor$

```
long mystery(long n, long k)
{
    long something = n;
    long count = -1;
    while (something >= 1) {
        something /= k;
        count += 1;
    }
    return count;
}
```

Problem 11.3(c)

- Give a pair of inputs that would cause the function to return the wrong answer.
- Any $n \leq 0$ will give a wrong answer
- $k = 0$ will throw a *Floating point exception*

```
long mystery(long n, long k)
{
    long something = n;
    long count = -1;
    while (something >= 1) {
        something /= k;
        count += 1;
    }
    return count;
}
```

Problem 11.3(d)

- Give a pair of inputs that would cause the function to loop forever.

- Any $n > 1$ and $k = 1$ suffices

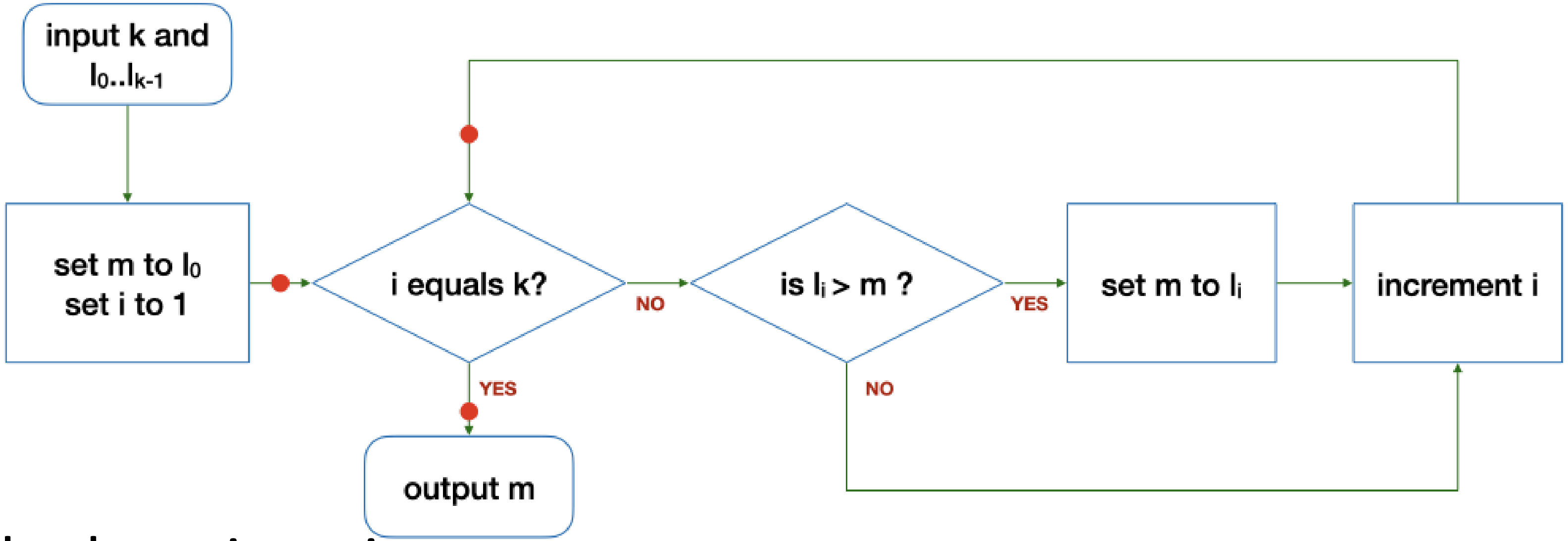
```
long mystery(long n, long k)
{
    long something = n;
    long count = -1;
    while (something >= 1) {
        something /= k;
        count += 1;
    }
    return count;
}
```

Problem Set 12

Loop invariants

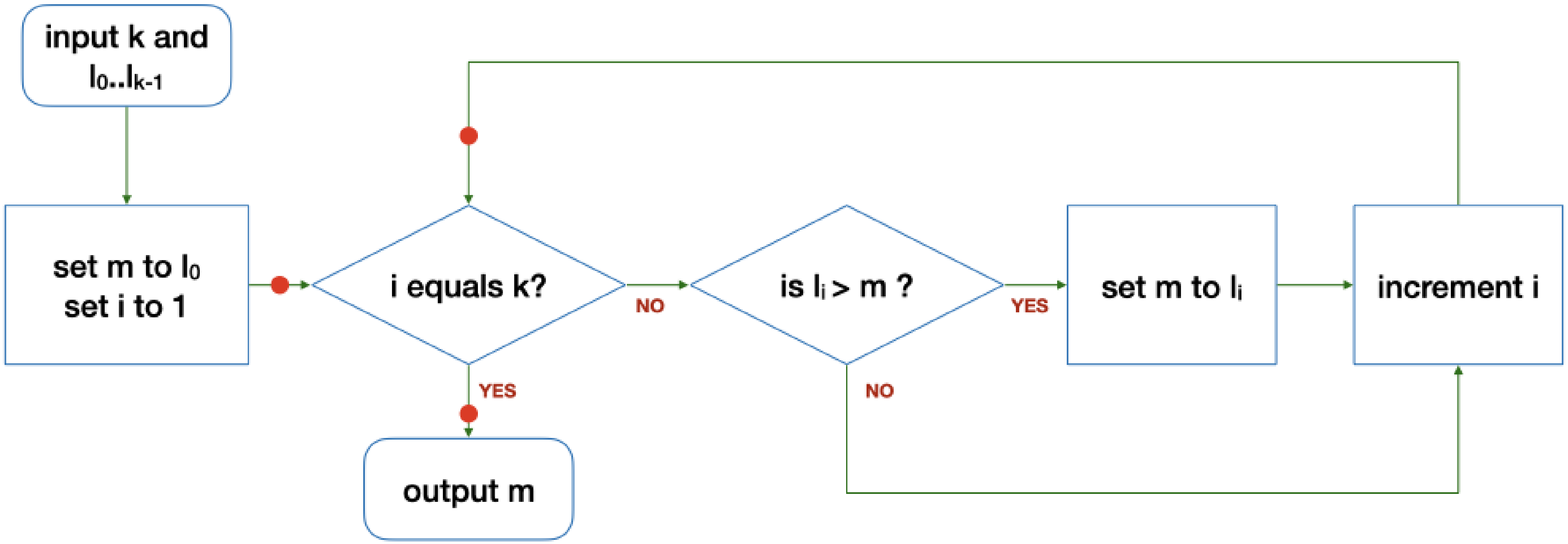
Problem 12.1

- Input: list L containing k integers where $k > 0$



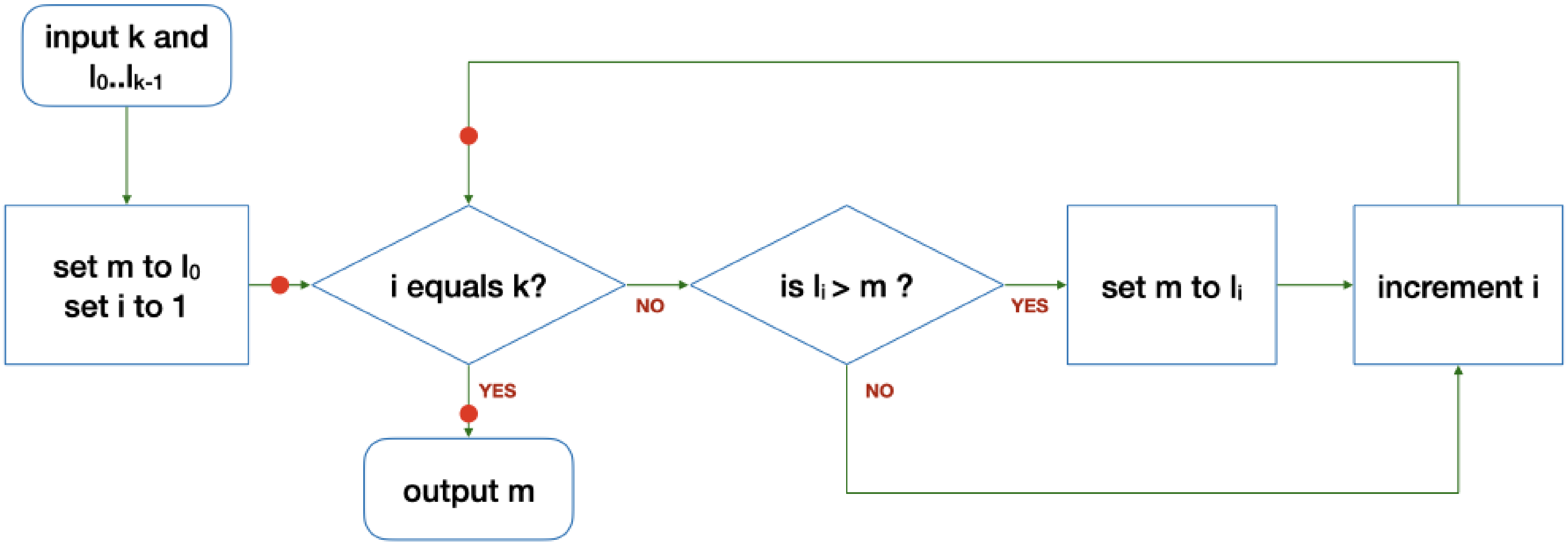
1. State the loop invariant
2. Why does it hold at the before, during and after the loop?
    - Intuitive explanations suffices
3. Argue that the loop correctly finds the maximum in L

Problem 12.1



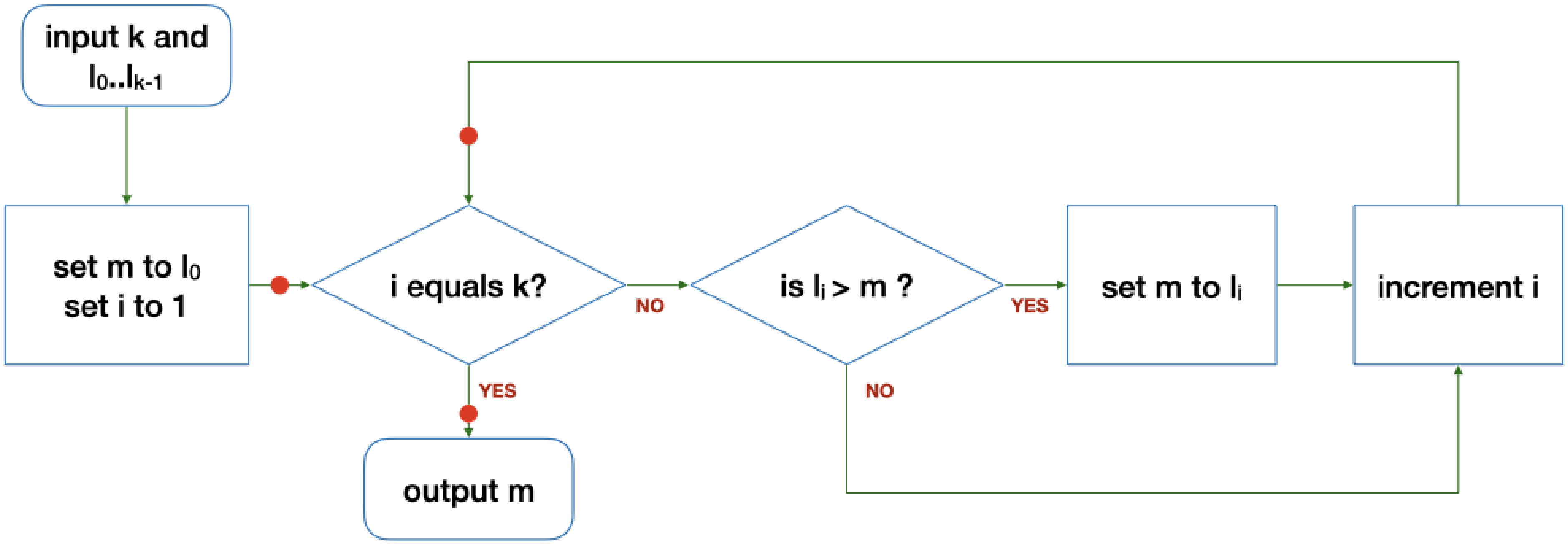
- What is the state of the variables before the loop begins?
    - m contains the first element of L
- What does the loop do with every iteration?
    - Checks if the current element is greater than m
    - Assigns it to m if it is so
    - Increments i
- After the loop terminates, m contains the max in the list of k elements

Problem 12.1



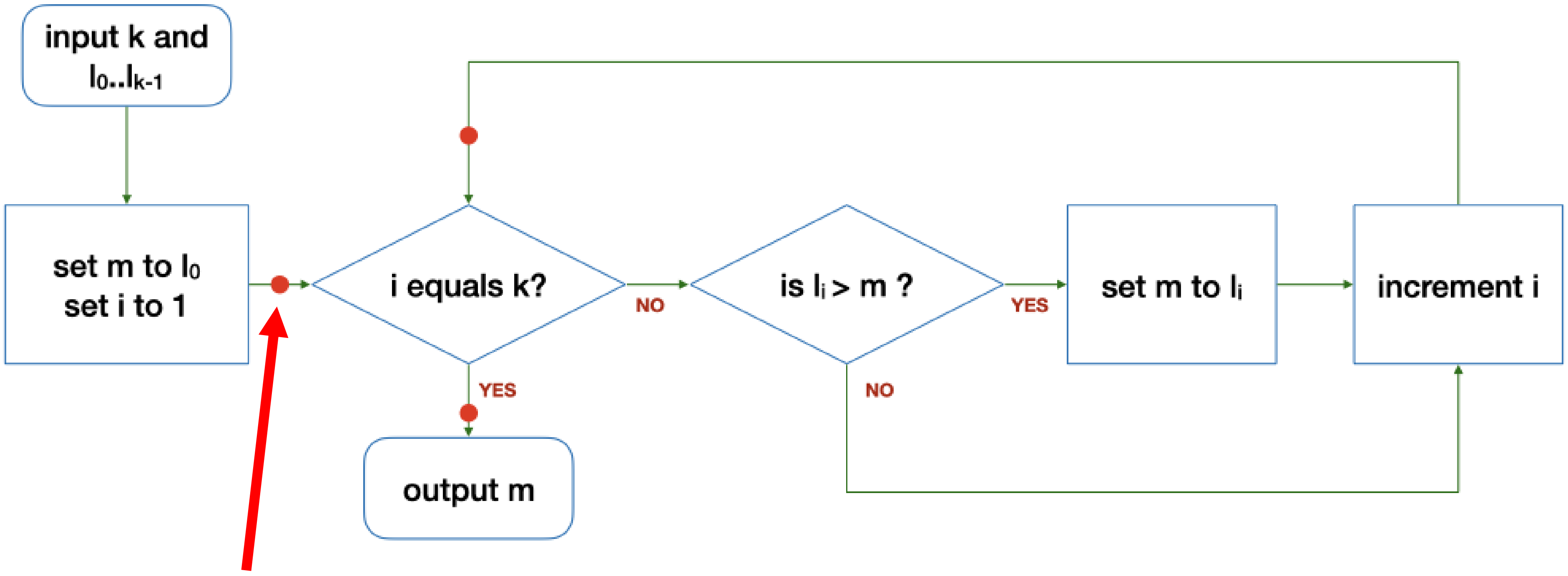
- At the end of the loop, we want m to be the max in the list L with k elements
- The property that we want to be true after the loop ends:
  - $\{m \in L \ \ \&\& \ \ m \geq [l_0 \dots l_{k-1}]\}$

Problem 12.1



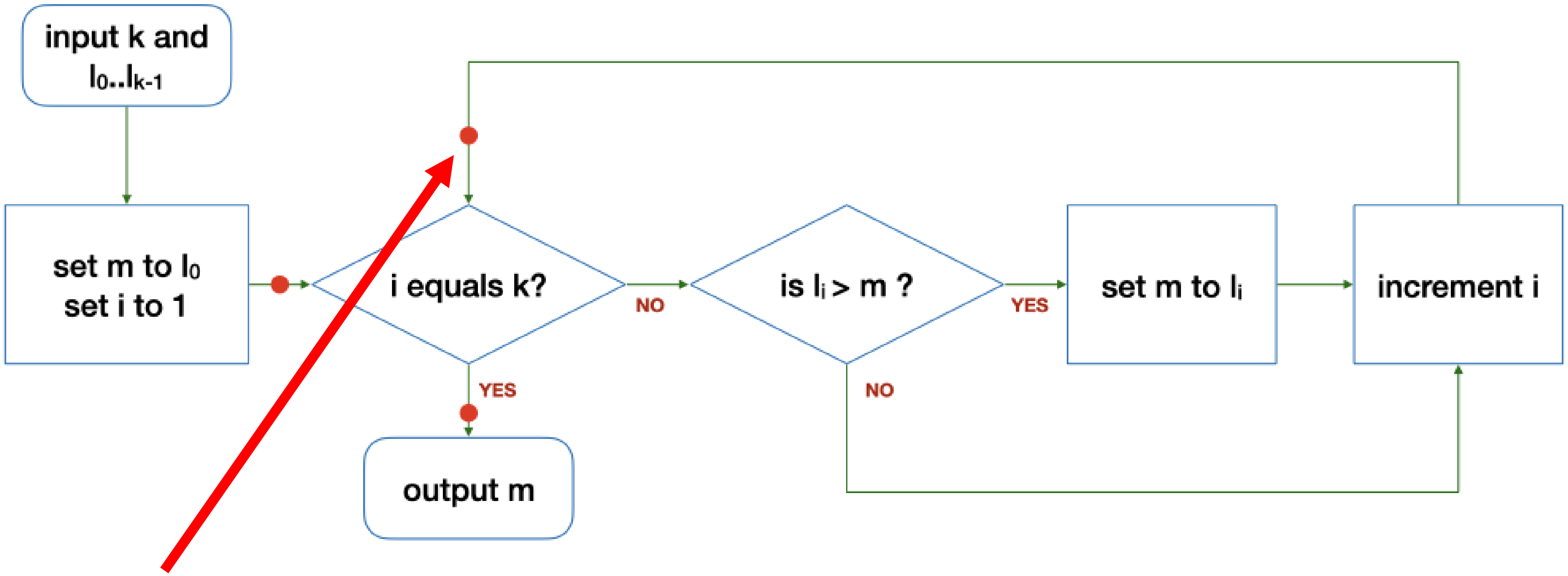
- At every iteration, the value of m always has the “max so far”
- A possible loop invariant is
  - $\{m \in L \quad \&\& \quad m \geq [l_0 \dots l_{i-1}]\}$

Problem 12.1



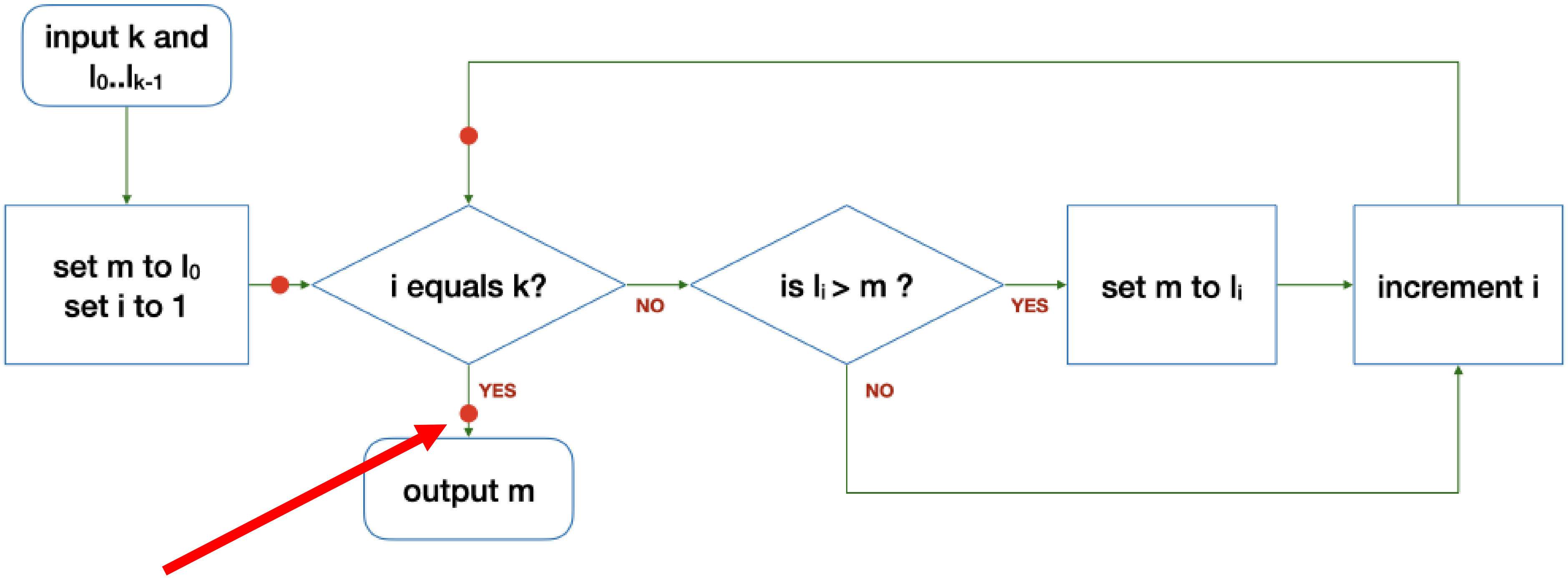
- $\{m \in L \quad \&\& \quad m \geq [l_0 \dots l_{i-1}]\}$
  - $m = l_0$ and $m \geq l_0$ (since $i = 1$)
  - The max of a list with 1 element, is the 1 element itself
  - The invariant in true

Problem 12.1



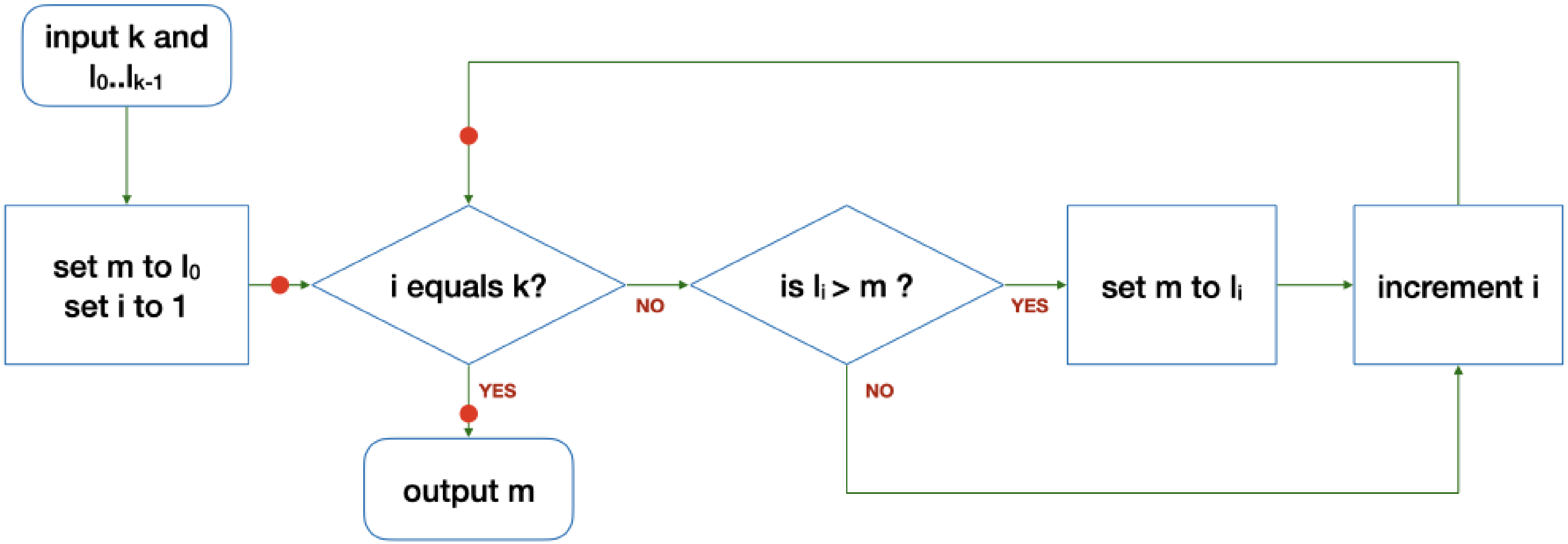
- $\{m \in L \quad \&\& \quad m \geq [l_0 \dots l_{i-1}]\}$
  - Before each iteration, the algorithm has the max of the first i elements
  - Consider the i-th element. If it is larger than m, set $m = l_i$. Else, m is unchanged
  - **Increment i**
  - Now m contains the max of the first i elements, once again.

Problem 12.1



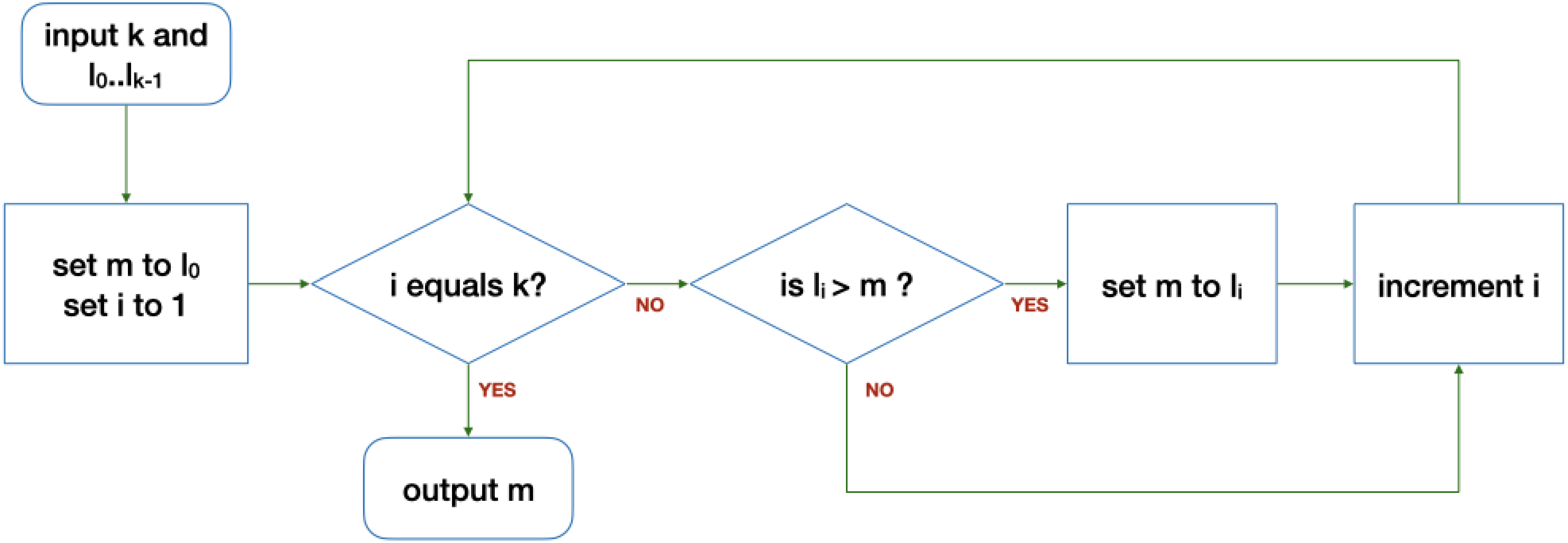
- $\{m \in L \;\; \&\& \;\; m \geq [l_0 \ldots l_{i-1}]\}$
  - i incremented until k. Now $i = k$
  - Therefore, we have the condition we want to be true at the end of the loop
    - $\{m \in L \;\; \&\& \;\; m \geq [l_0 \ldots l_{k-1}]\}$

Problem 12.1



- $\{m \in L \;\; \&\& \;\; m \geq [l_0 \ldots l_{i-1}]\}$
  - The assertion $\{i == k\}$ must be true at the end of the loop
  - Therefore, $\{m \in L \;\; \&\& \;\; m \geq [l_0 \ldots l_{k-1}]\}$ is true
  - QED

“Formal” Proof for $FIND\_MAX$ (Optional)



“Formal” Proof for $FIND\_MAX$ (Optional)

- Claim: **FIND_MAX** is correct for k where $k \in N$ and $k > 0$.
    - Prove correctness by induction on k.

- **Base case:** $k = 1$. The algorithm returns the singular element in the list, which must be the max. The base case is correct.

- **Inductive Hypothesis:** The algorithm is correct for a list of size m

- Consider a list of size $m + 1$
    - By the inductive hypothesis, we can find the max from a list of size m, call it $\boldsymbol{a}$
    - After considering m elements, one element remains, call it $\boldsymbol{b}$
    - The algorithm compares a and b and outputs the larger of the two values, or a if $a = b$

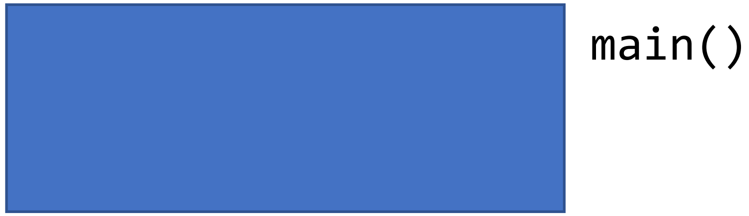
- Therefore, **FIND_MAX** is correct

Problem 13.1

Ready to draw?

Problem 13.1

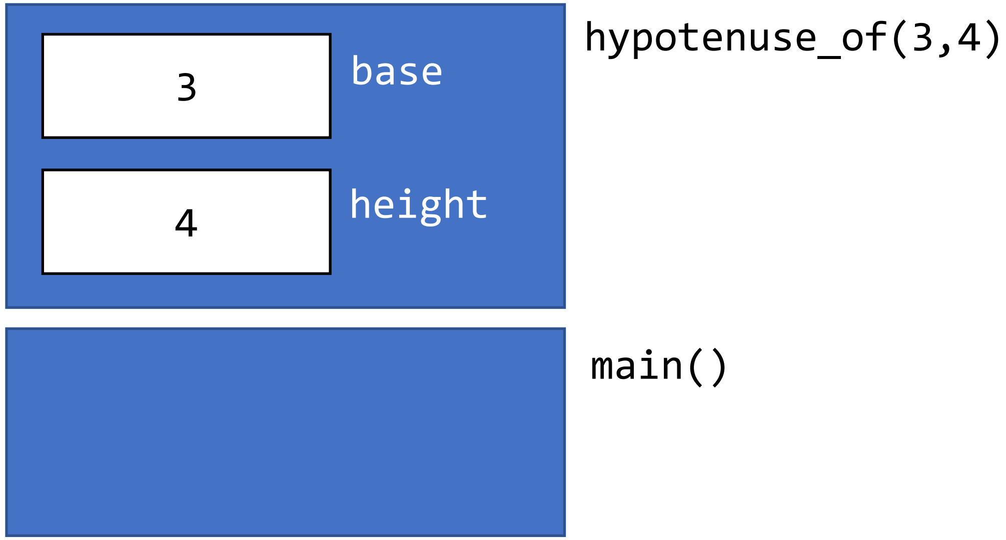
```
#include <math.h>

long square(long x)
{
    return x * x;
}

double hypotenuse_of(long base, long height)
{
    return sqrt(square(base) + square(height));
}

int main()
{
    hypotenuse_of(3, 4);
}
```



main()

Problem 13.1

```
#include <math.h>

long square(long x)
{
    return x * x;
}

double hypotenuse_of(long base, long height)
{
    return sqrt(square(base) + square(height));
}

int main()
{
    hypotenuse_of(3, 4);
}
```



hypotenuse_of(3,4)

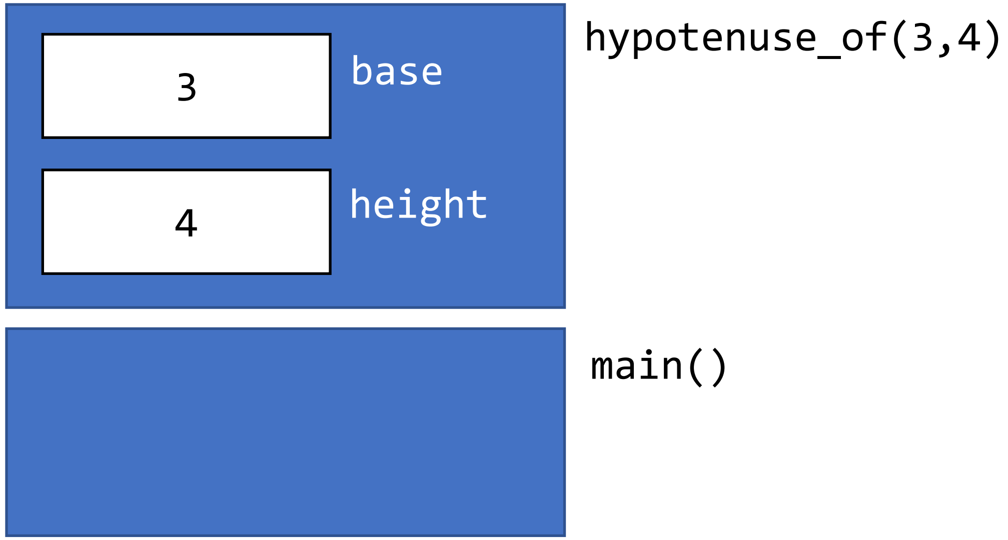
3 base

4 height

main()

Problem 13.1

```
#include <math.h>

long square(long x)
{
    return x * x;
}

double hypotenuse_of(long base, long height)
{
    return sqrt(square(base) + square(height));
}

int main()
{
    hypotenuse_of(3, 4);
}
```



3 base

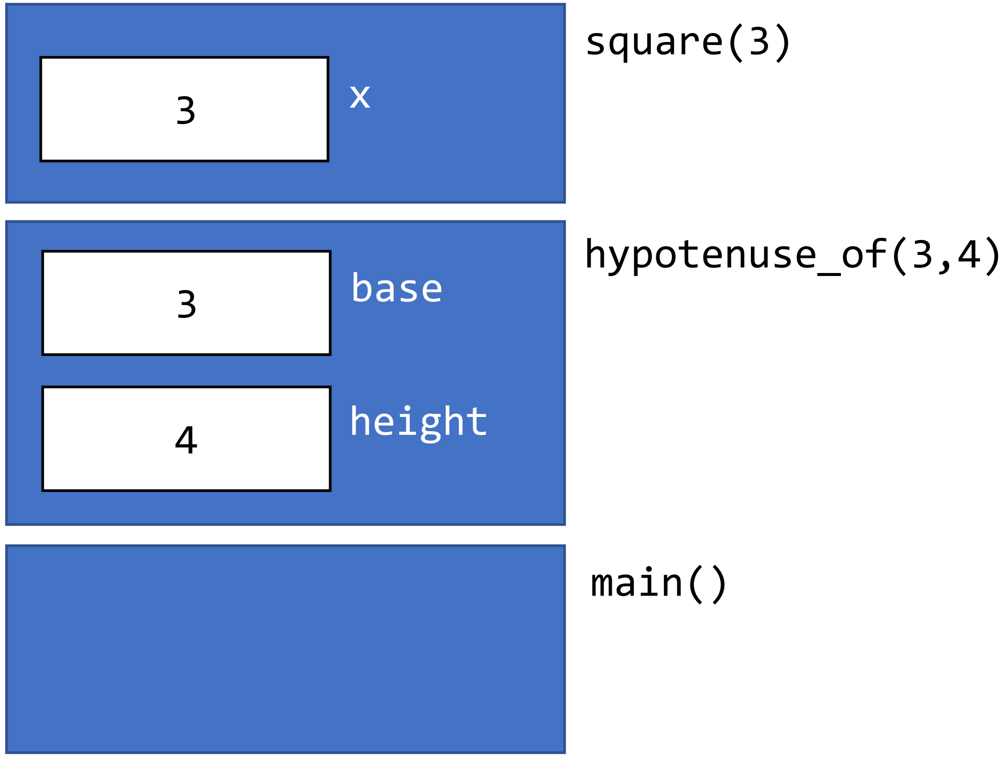
4 height

hypotenuse_of(3,4)

main()

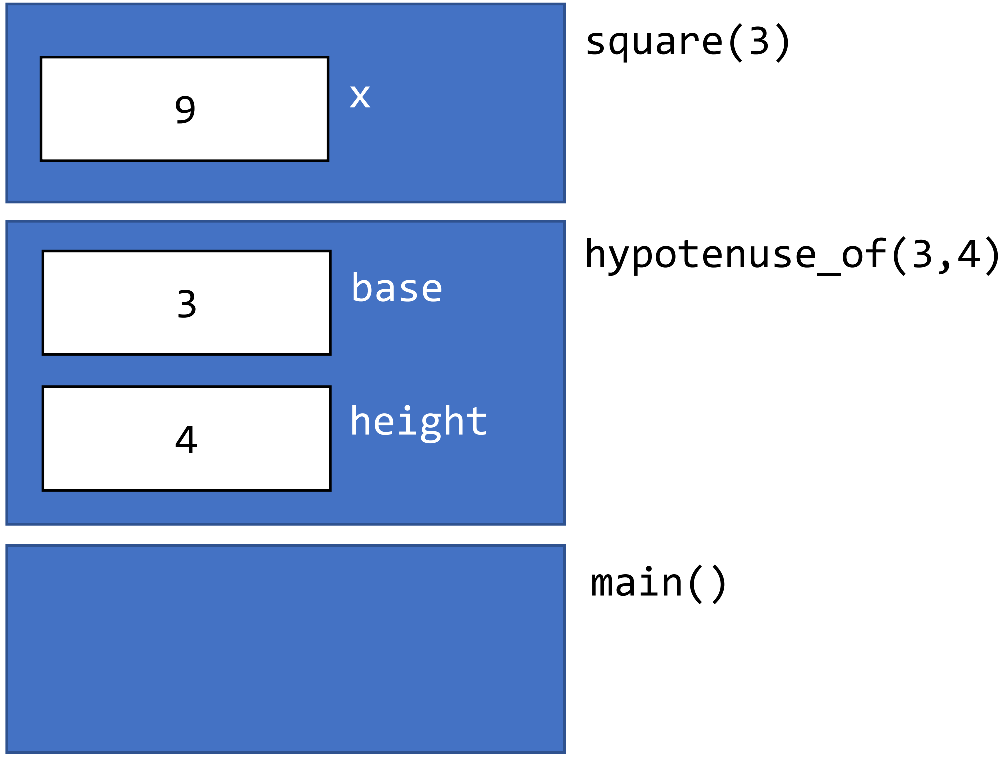
Problem 13.1

```c
#include <math.h>

long square(long x)
{
    return x * x;
}

double hypotenuse_of(long base, long height)
{
    return sqrt(square(base) + square(height));
}

int main()
{
    hypotenuse_of(3, 4);
}
```



Problem 13.1

```
#include <math.h>

long square(long x)
{
    return x * x;
}

double hypotenuse_of(long base, long height)
{
    return sqrt(square(base) + square(height));
}

int main()
{
    hypotenuse_of(3, 4);
}
```



square(3)

9 x

hypotenuse_of(3,4)

3 base

4 height

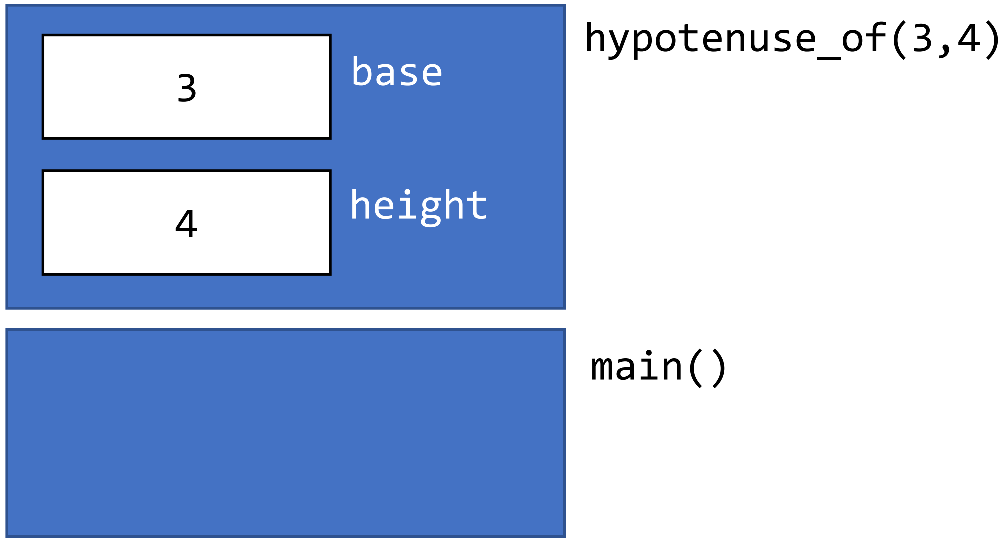
main()

Problem 13.1

```
#include <math.h>

long square(long x)
{
    return x * x;
}

double hypotenuse_of(long base, long height)
{
    return sqrt(square(base) + square(height));
}
```

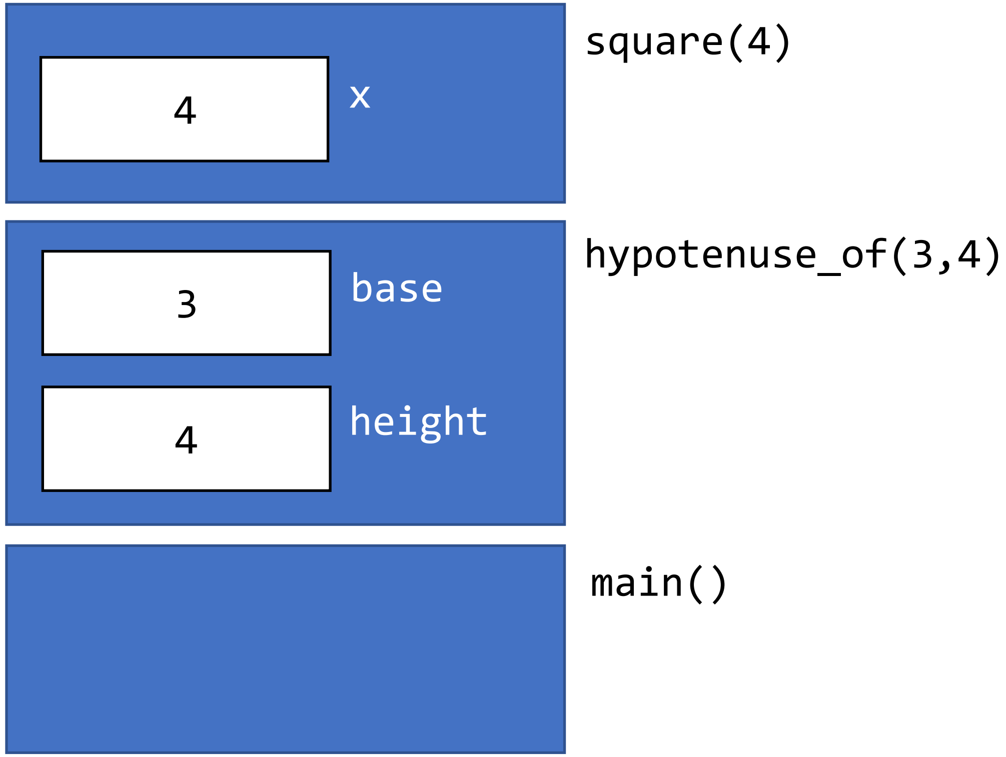
9

```
int main()
{
    hypotenuse_of(3, 4);
}
```



Problem 13.1

```
#include <math.h>

long square(long x)
{
    return x * x;
}

double hypotenuse_of(long base, long height)
{
    return sqrt(square(base) + square(height));
}                  9

int main()
{
    hypotenuse_of(3, 4);
}
```



square(4)

4 x

hypotenuse_of(3,4)

3 base

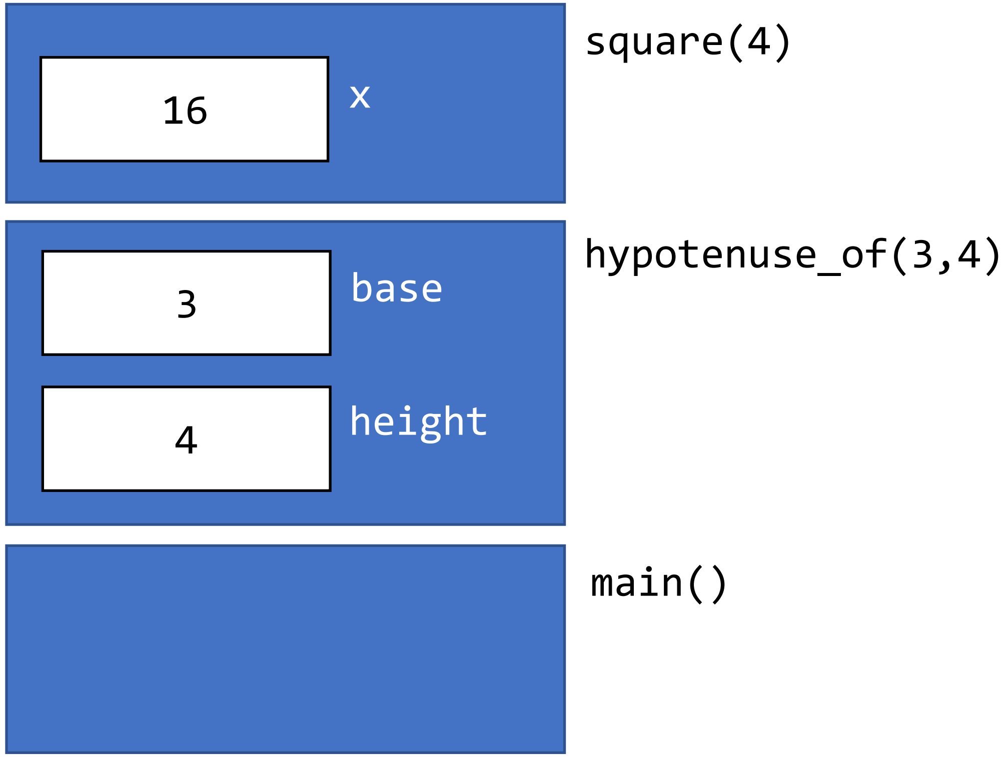
4 height

main()

Problem 13.1

```
#include <math.h>

long square(long x)
{
    return x * x;
}

double hypotenuse_of(long base, long height)
{
    return sqrt(square(base) + square(height));
}

int main()
{
    hypotenuse_of(3, 4);
}
```

9



square(4)

16 x

hypotenuse_of(3,4)

3 base

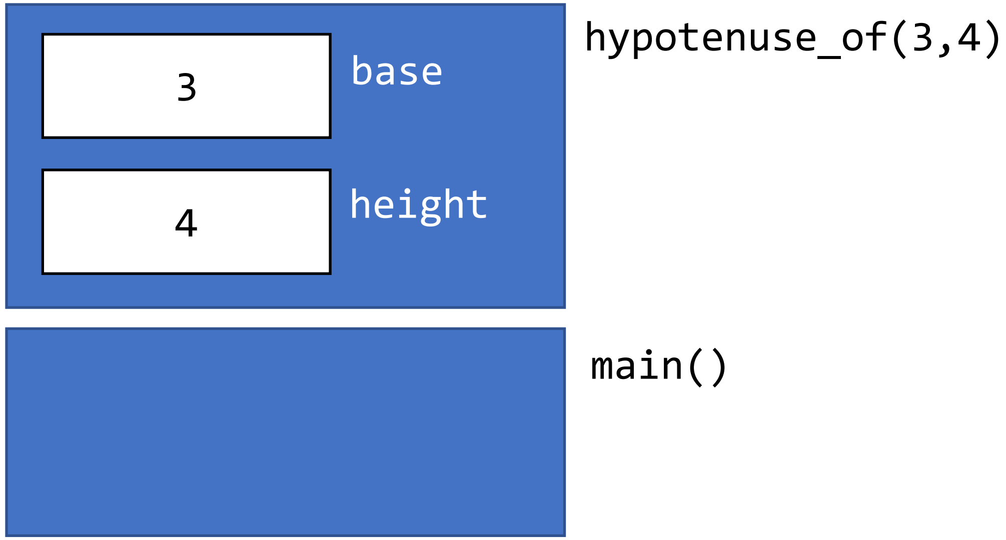
4 height

main()

Problem 13.1

```c
#include <math.h>

long square(long x)
{
    return x * x;
}

double hypotenuse_of(long base, long height)
{
    return sqrt(square(base) + square(height));
}

int main()
{
    hypotenuse_of(3, 4);
}
```

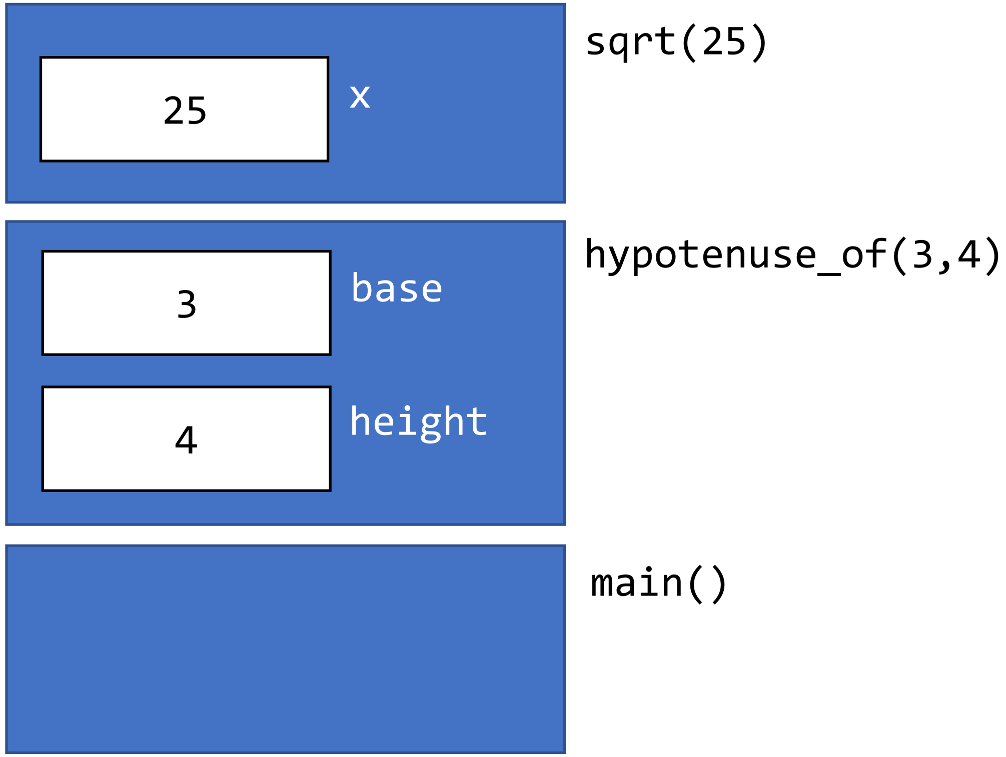
9 16



Problem 13.1

```c
#include <math.h>

long square(long x)
{
    return x * x;
}

double hypotenuse_of(long base, long height)
{
    return sqrt(square(base) + square(height));
}                    9               16

int main()
{
    hypotenuse_of(3, 4);
}
```

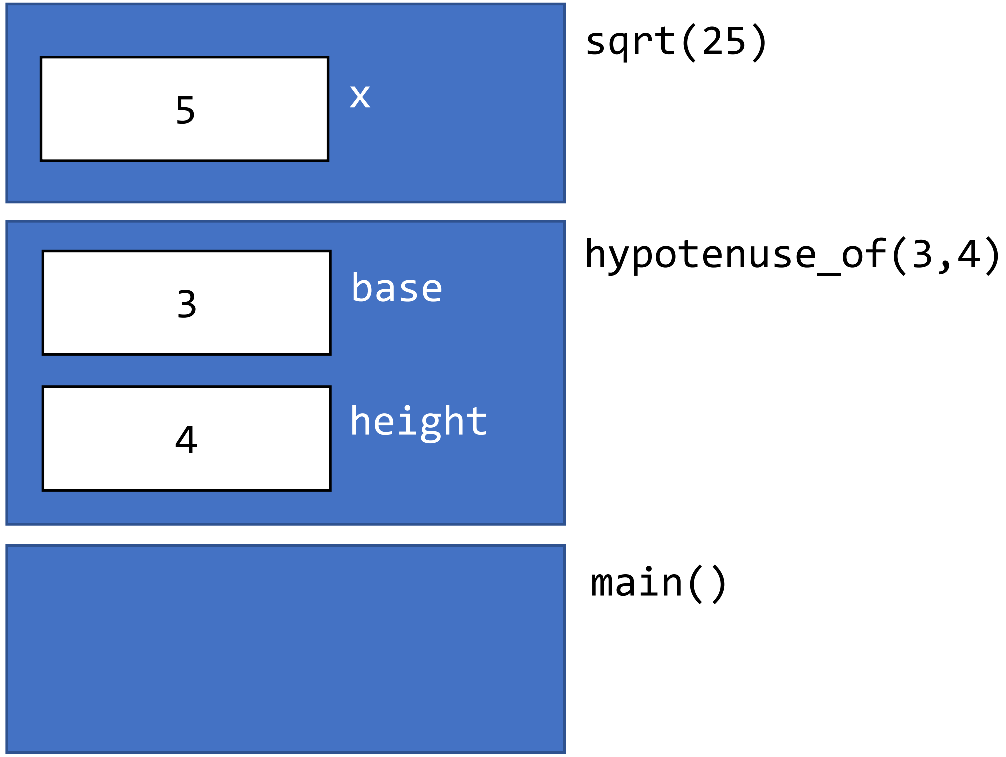
In the C library somewhere...



Problem 13.1

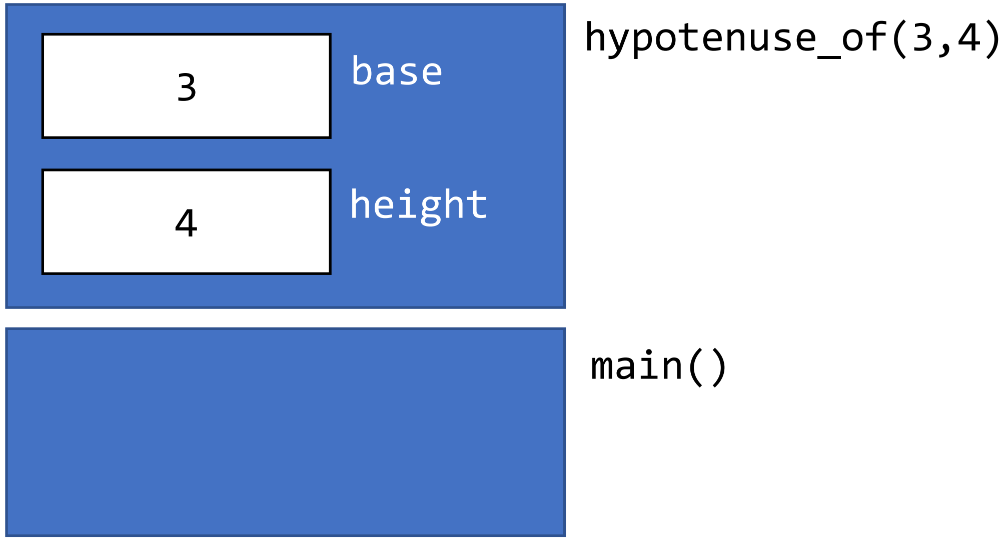
```
#include <math.h>

long square(long x)
{
    return x * x;
}

double hypotenuse_of(long base, long height)
{
    return sqrt(square(base) + square(height));
}                      9               16

int main()
{
    hypotenuse_of(3, 4);
}
```

Before **sqrt** returns



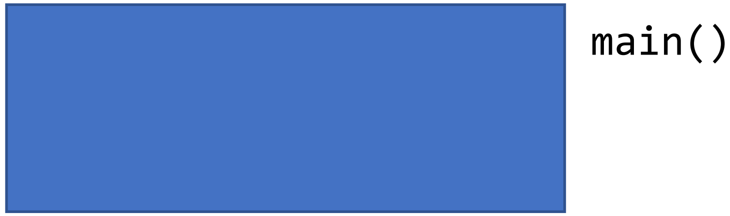
Problem 13.1

```
#include <math.h>

long square(long x)
{
    return x * x;
}

double hypotenuse_of(long base, long height)
{           5
    return sqrt(square(base) + square(height));
}                    9               16

int main()
{
    hypotenuse_of(3, 4);
}
```



Problem 13.1

```
#include <math.h>

long square(long x)
{
    return x * x;
}

double hypotenuse_of(long base, long height)
{           5
    return sqrt(square(base) + square(height));
}                    9              16

int main()
{
    hypotenuse_of(3, 4);
}
```

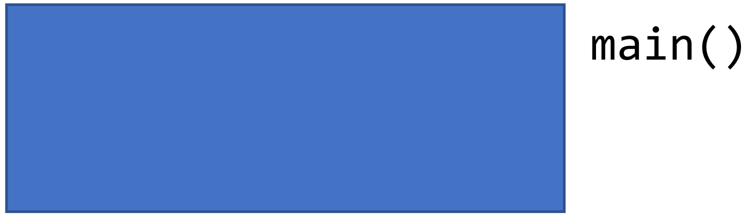


Problem 13.2

Draw some more

Problem 13.2

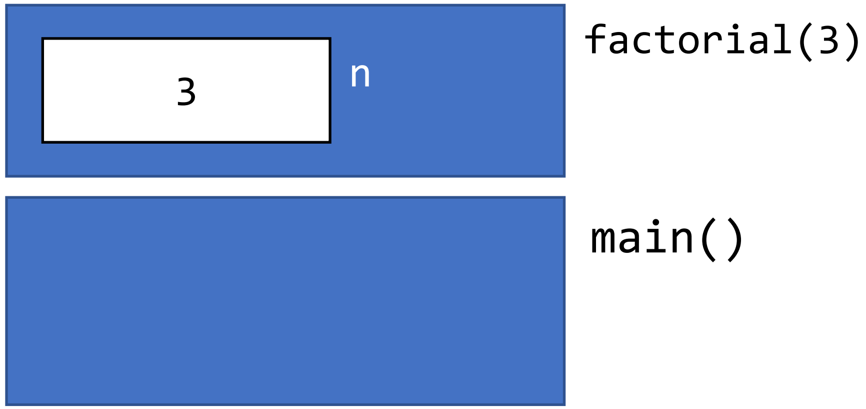
```
long factorial(long n)
{
    if (n == 0) {
        return 1;
    }
    return factorial(n - 1) * n;
}

int main()
{
    factorial(3);
}
```



main()

Problem 13.2

```
long factorial(long n)
{
    if (n == 0) {
        return 1;
    }
    return factorial(n - 1) * n;
}

int main()
{
    factorial(3);
}
```



3

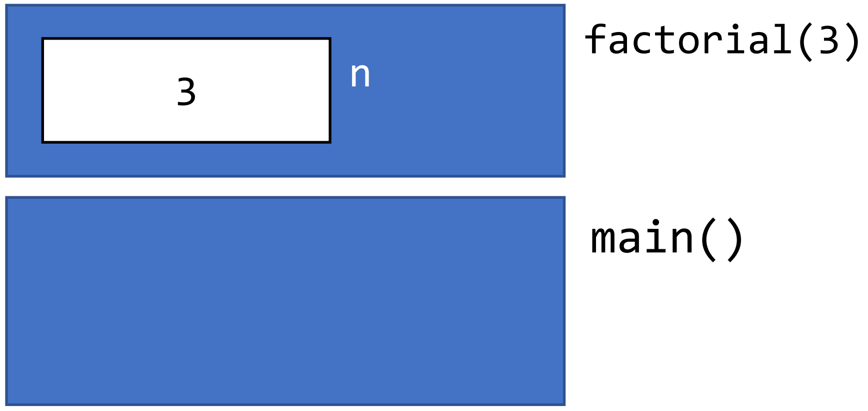
n

factorial(3)

main()

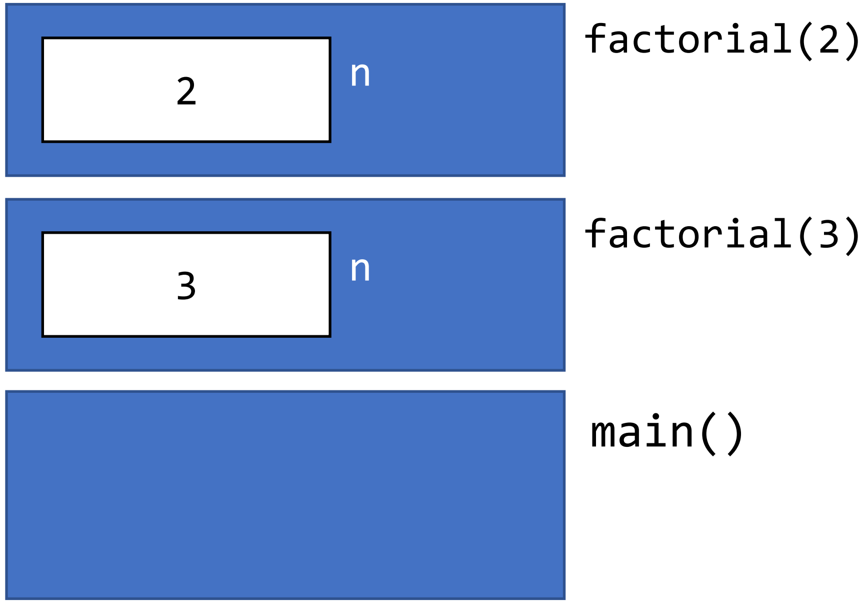
Problem 13.2

```
long factorial(long n)
{
    if (n == 0) {
        return 1;
    }
    return factorial(n - 1) * n;
}

int main()
{
    factorial(3);
}
```



Problem 13.2

```
long factorial(long n)
{
    if (n == 0) {
        return 1;
    }
    return factorial(n - 1) * n;
}

int main()
{
    factorial(3);
}
```



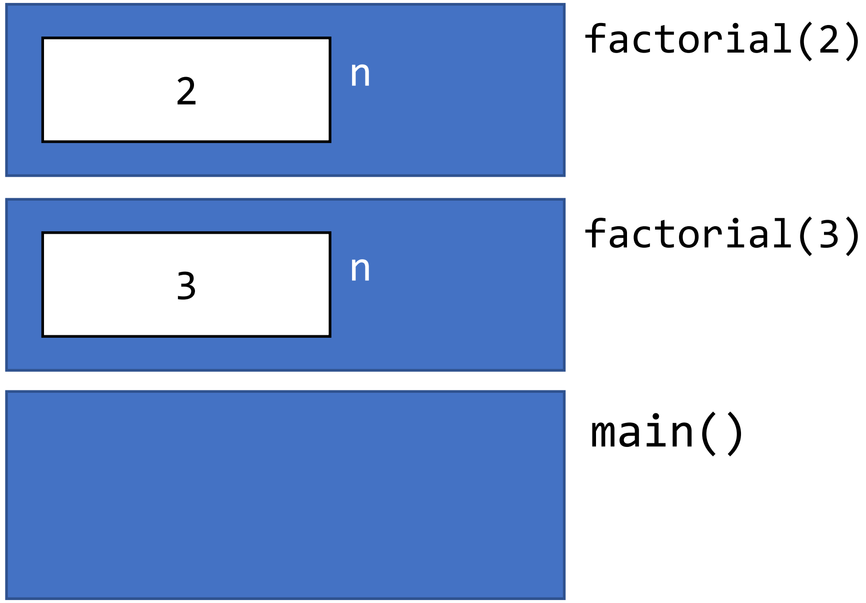
2 n factorial(2)

3 n factorial(3)

main()

Problem 13.2

```
long factorial(long n)
{
    if (n == 0) {
        return 1;
    }
    return factorial(n - 1) * n;
}

int main()
{
    factorial(3);
}
```



factorial(2)

n

2

factorial(3)

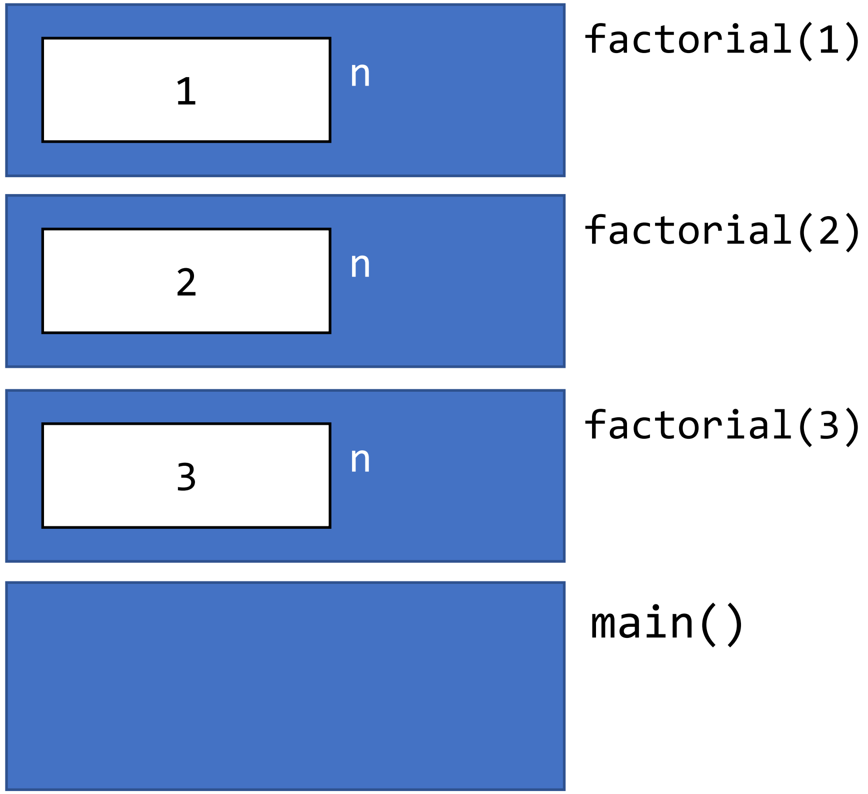
n

3

main()

Problem 13.2

```
long factorial(long n)
{
    if (n == 0) {
        return 1;
    }
    return factorial(n - 1) * n;
}

int main()
{
    factorial(3);
}
```



1 n factorial(1)

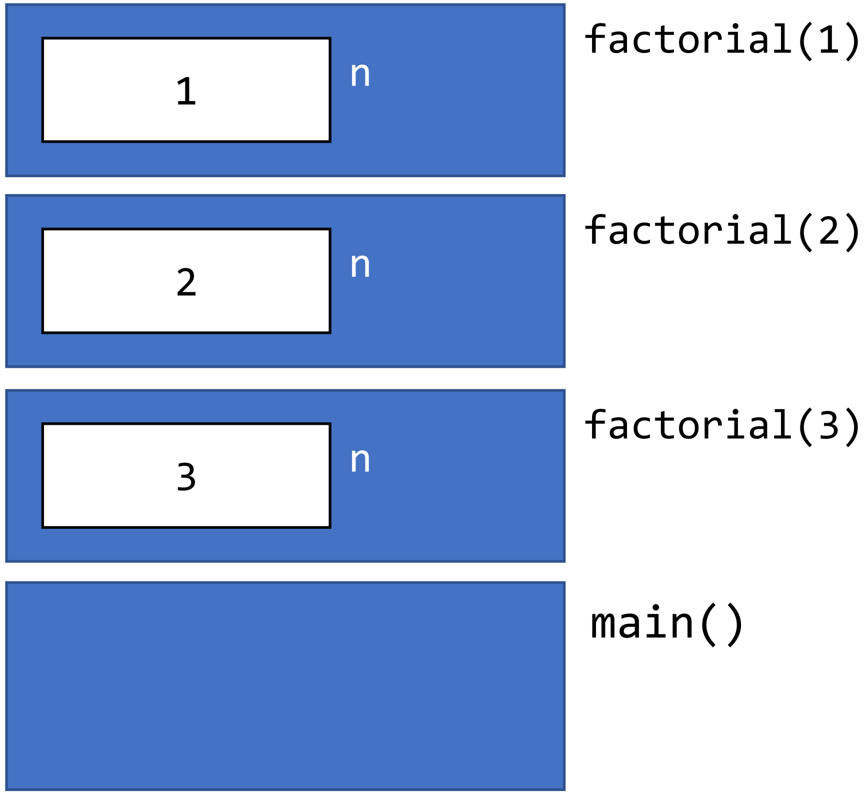
2 n factorial(2)

3 n factorial(3)

main()

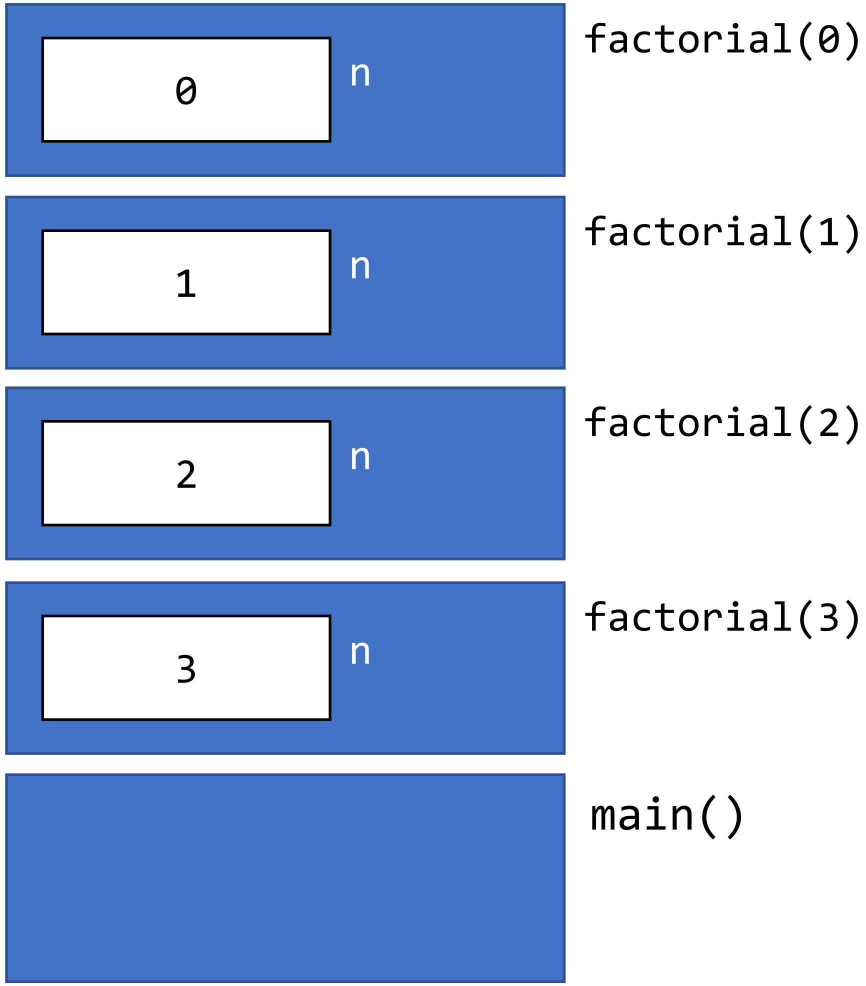
Problem 13.2

```
long factorial(long n)
{
    if (n == 0) {
        return 1;
    }
    return factorial(n - 1) * n;
}

int main()
{
    factorial(3);
}
```



Problem 13.2

```
long factorial(long n)
{
    if (n == 0) {
        return 1;
    }
    return factorial(n - 1) * n;
}

int main()
{
    factorial(3);
}
```



0 n factorial(0)

1 n factorial(1)

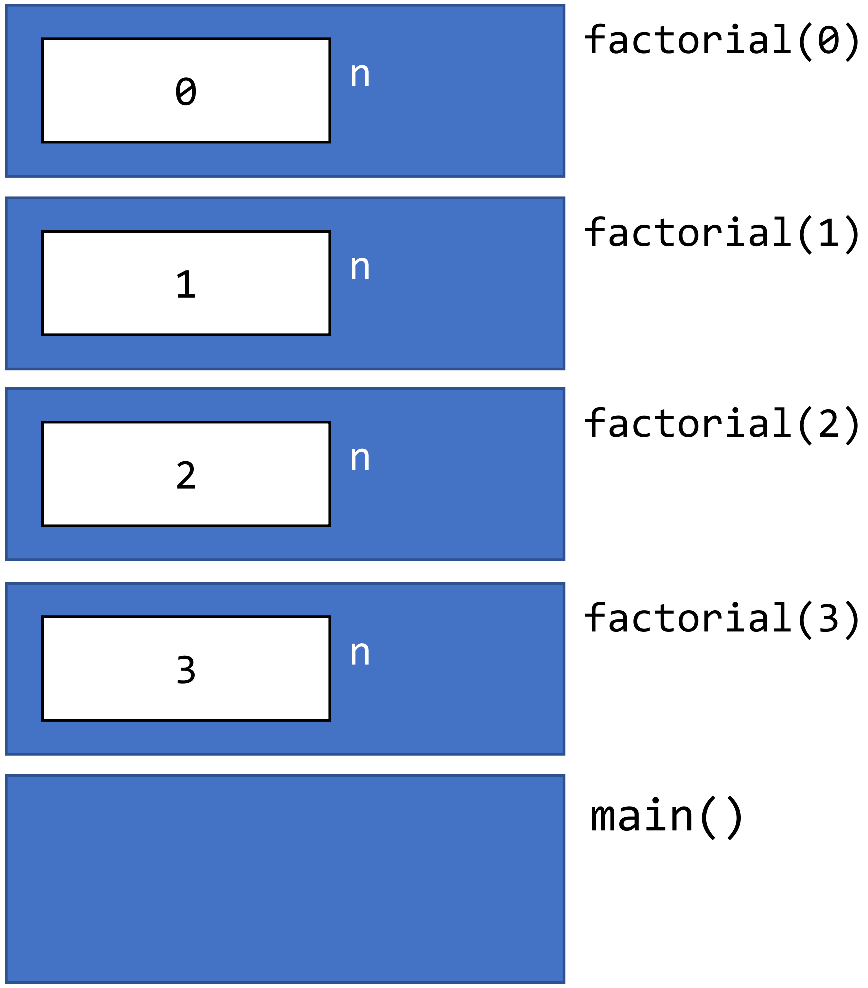
2 n factorial(2)

3 n factorial(3)

main()

Problem 13.2

```
long factorial(long n)
{
    if (n == 0) {
        return 1;
    }
    return factorial(n - 1) * n;
}

int main()
{
    factorial(3);
}
```



0 n factorial(0)

1 n factorial(1)

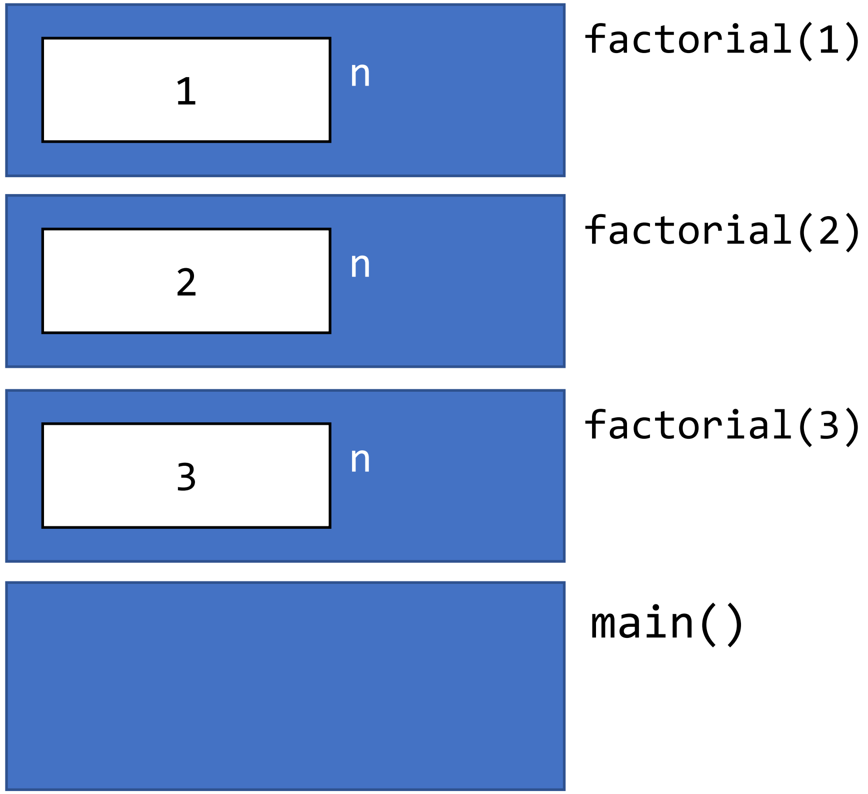
2 n factorial(2)

3 n factorial(3)

main()

Problem 13.2

```
long factorial(long n)
{
    if (n == 0) {
        return 1;
    }
    return factorial(n - 1) * n;
}

int main()
{
    factorial(3);
}
```



1 n factorial(1)

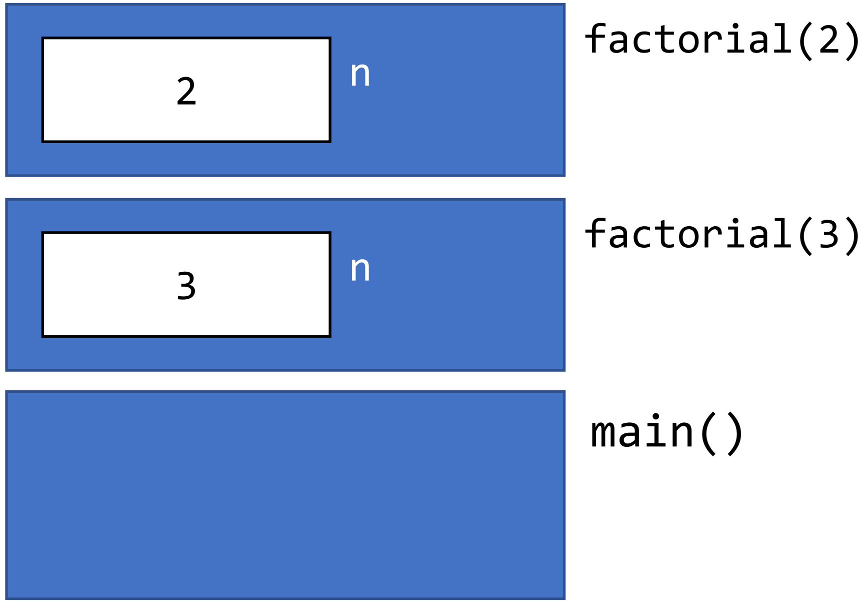
2 n factorial(2)

3 n factorial(3)

main()

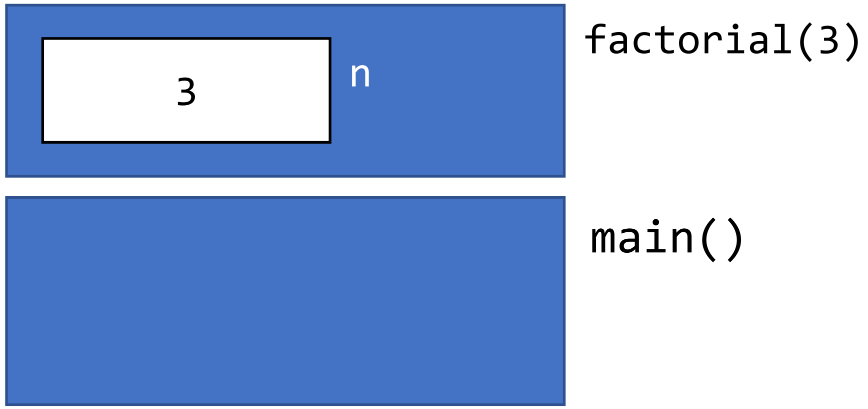
Problem 13.2

```
long factorial(long n)
{
    if (n == 0) {
        return 1;
    }
    return factorial(n - 1) * n;
}

int main()
{
    factorial(3);
}
```



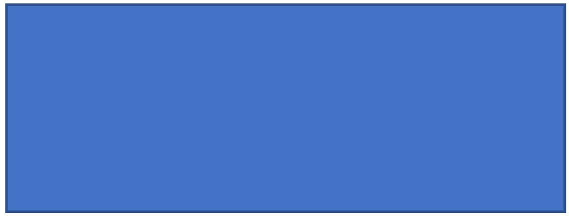
Problem 13.2

```
long factorial(long n)
{
    if (n == 0) {
        return 1;
    }
    return factorial(n - 1) * n;
}

int main()
{
    factorial(3);
}
```



Problem 13.2

```
long factorial(long n)
{
    if (n == 0) {
        return 1;
    }
    return factorial(n - 1) * n;
}

int main()
{
    factorial(3);
}
```



main()

Problem 13.3

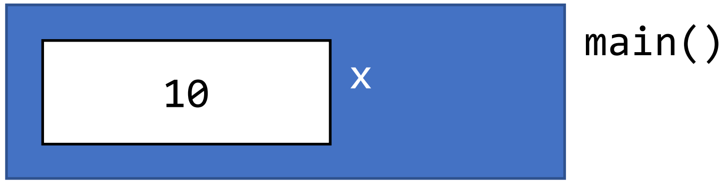
More drawing

Problem 13.3

```
void incr(long x) {
    x += 1;
}

int main()
{
    long x = 10;
    incr(x);
    incr(x);
    cs1010_print_long(x);
}
```

Problem 13.3

```
void incr(long x) {
    x += 1;
}

int main()
{
    long x = 10;
    incr(x);
    incr(x);
    cs1010_print_long(x);
}
```



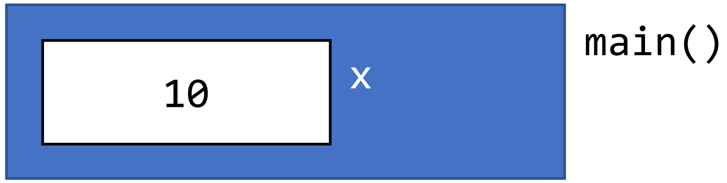
10

x

main()

Problem 13.3

```
void incr(long x) {
    x += 1;
}

int main()
{
    long x = 10;
    incr(x);
    incr(x);
    cs1010_print_long(x);
}
```



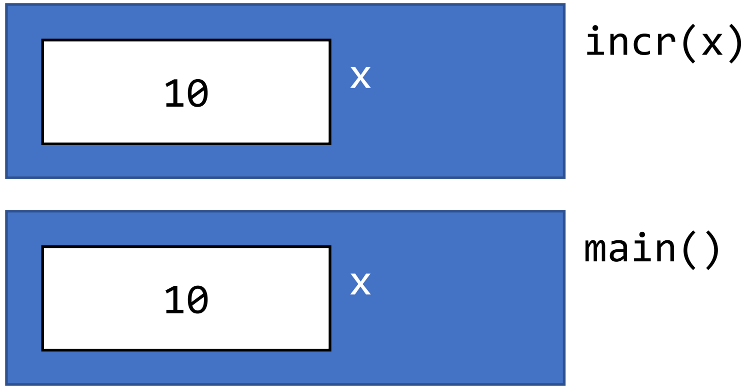
10

x

main()

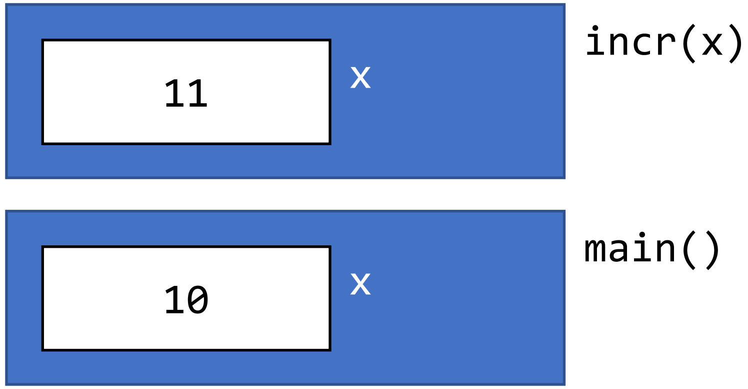
Problem 13.3

```
void incr(long x) {
    x += 1;
}

int main()
{
    long x = 10;
    incr(x);
    incr(x);
    cs1010_print_long(x);
}
```



Problem 13.3

```
void incr(long x) {
    x += 1;
}

int main()
{
    long x = 10;
    incr(x);
    incr(x);
    cs1010_print_long(x);
}
```

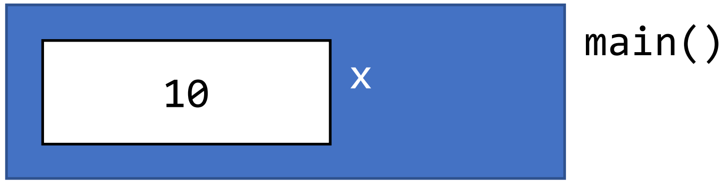


11 x incr(x)

10 x main()

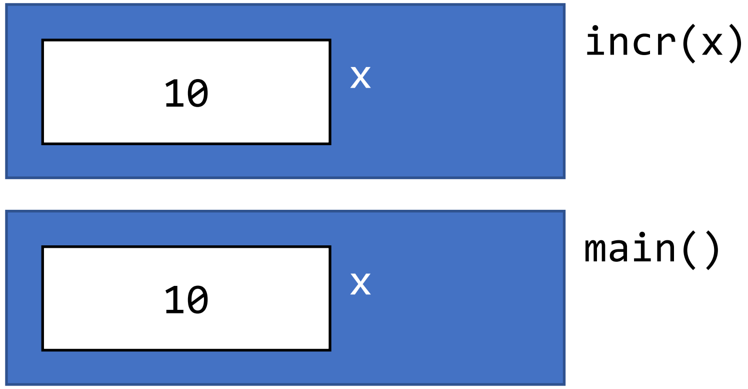
Problem 13.3

```
void incr(long x) {
    x += 1;
}

int main()
{
    long x = 10;
    incr(x);
    incr(x);
    cs1010_print_long(x);
}
```



Problem 13.3

```
void incr(long x) {
    x += 1;
}

int main()
{
    long x = 10;
    incr(x);
    incr(x);
    cs1010_print_long(x);
}
```

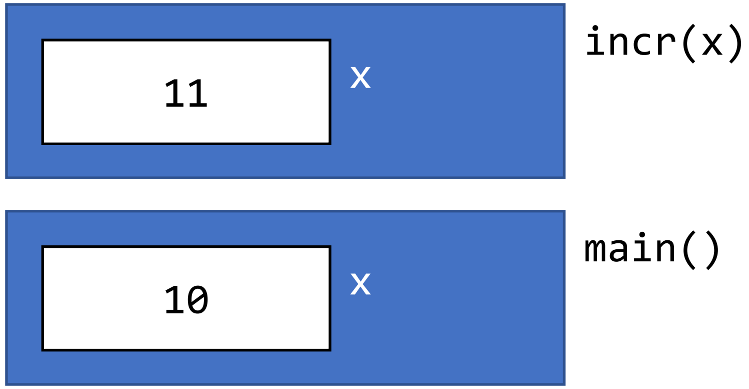


10 x incr(x)

10 x main()

Problem 13.3

```
void incr(long x) {
    x += 1;
}

int main()
{
    long x = 10;
    incr(x);
    incr(x);
    cs1010_print_long(x);
}
```

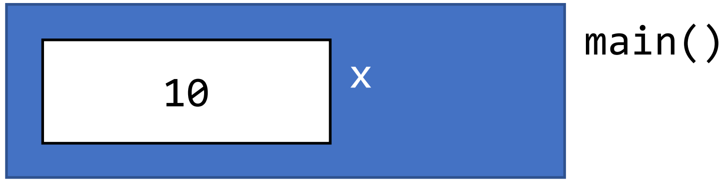


11 x incr(x)

10 x main()

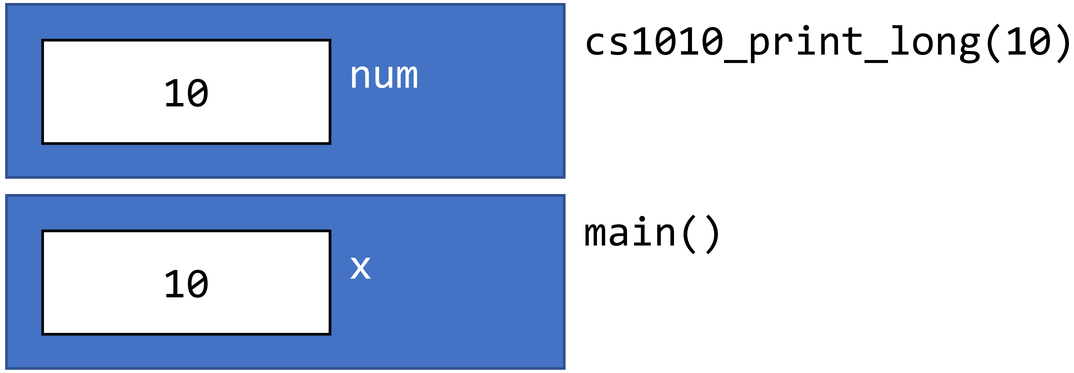
Problem 13.3

```
void incr(long x) {
    x += 1;
}

int main()
{
    long x = 10;
    incr(x);
    incr(x);
    cs1010_print_long(x);
}
```



Problem 13.3

```
void incr(long x) {
    x += 1;
}

int main()
{
    long x = 10;
    incr(x);
    incr(x);
    cs1010_print_long(x);
}
```

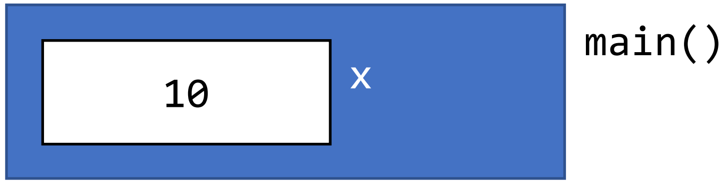


10 num cs1010_print_long(10)

10 x main()

Problem 13.3

```
void incr(long x) {
    x += 1;
}

int main()
{
    long x = 10;
    incr(x);
    incr(x);
    cs1010_print_long(x);
}
```



Assignment 1 Marking Scheme

And comments

General Marking Scheme

No submission / cannot compile	0 marks
Compiler warnings	-1 per warning
Bugs	-1 per bug
No identification via @author	-1 per occurrence
Penalization is per program	

Box Comments

- Most generally correct

- Some used the formula $D = \sqrt{l^2 + w^2 + h^2}$ to calculate the diagonal

- This is not wrong, but the question asks you to modify the **hypotenuse_of** function seen in lecture to calculate the diagonal
    - This was not penalized, however

Digits Comments

- To get the last digit from a number, use **% 10**
- To strip away the last digit from a number, using **/ 10** suffices
- Integer division will automatically truncate the fractional portion of the number

Suffix Comments

- Generally correct

- Any number that ends with **11**, **12** and **13** should have the **th** suffix.

- You need to check if the number “ends with” 11, 12 or 13, not just check if the number is 11, 12 or 13

Taxi Comments

- **long** should be used for reading in the variables
  - -1 for each wrong type

- -1 per occurrence of using **int** over **long** and **float** over **double**

- Computation should be broken down into four functions
  - **is_weekday**
  - **is_morning_peak_hour**
  - **is_evening_peak_hour**
  - **is_midnight_peak_hour**
  - -1 per missing function

Taxi Comments

- Input/output types of functions should be correct
    - -1 if you used **int** for boolean types

- **If…else** statements should be clean
    - No redundancy

Assignment 2

Roughly 15min or so

Assignment 2 Instructions

- Accept the assignment, and **ssh** into the PE nodes and run **~cs1010/get-as02**

- During the PE, you must **maximise your terminal** and are not allowed to use *any* other software, the question papers are stored as text files for them on PE hosts
  - I will try to ask Prof Zhao if we can print out the paper for you to avoid this

- If you are still uncomfortable with vim, you **must** learn how to use it by PE!

Assignment 2 Instructions

- Try to work on Assignment 2 by **only** using **a single** terminal window
  - No browsers
  - No extra terminal windows

- This is to emulate the PE environment

Exercise 2 Explanations

Loop invariants

Q1 - Binary

- Read in an **integer** containing a **binary number**

- A binary number is a number in **base-2** that only contains 1s and 0s

- Task
  - **Convert** and print out the corresponding **decimal** (base-10) number

Q1 - Binary

- Read in an **integer** containing a **binary number**

- A binary number is a number in **base-2** that only contains 1s and 0s

- Task
  - **Convert** and print out the corresponding **decimal** (base-10) number

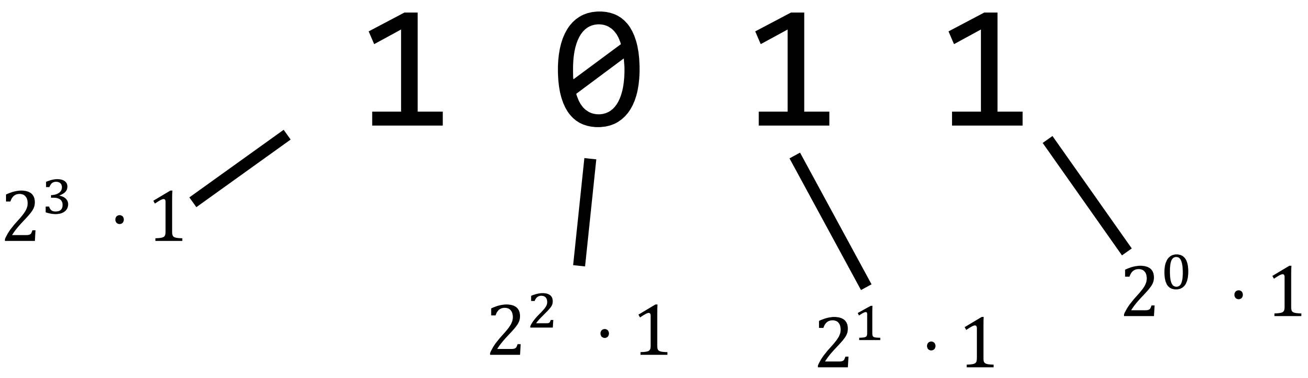
Q1 - Binary

- **Convert** and print out the corresponding **decimal** (base-10) number

- You already know how to extract the last digit from a number
- You already know how to strip the last digit from a number

- What is the “weight” of each digit in a binary number?

Q1 - Binary



1 0 1 1

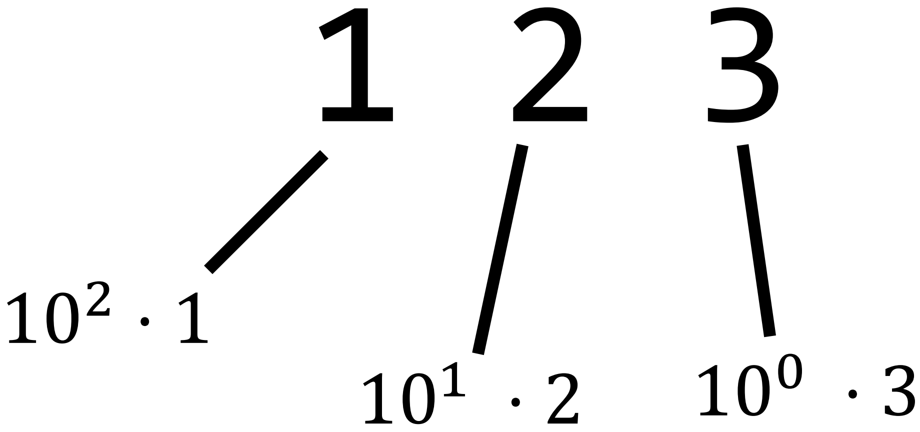
$2^3 \cdot 1$

$2^2 \cdot 1$

$2^1 \cdot 1$

$2^0 \cdot 1$

$$d = (2^0 \cdot 1) + (2^1 \cdot 1) + (2^2 \cdot 0) + (2^3 \cdot 1)$$
$$= 1 + 2 + 8$$
$$= 11$$



1 2 3

$10^2 \cdot 1$

$10^1 \cdot 2$

$10^0 \cdot 3$

$$d = (10^0 \cdot 3) + (10^1 \cdot 2) + (10^2 \cdot 1)$$
$$= 3 + 20 + 100$$
$$= 123$$

Q2 - Rectangle

- Read in two integers that correspond to the **width** and **height** of a rectangle where $width \geq 2$ and $height \geq 2$

- This one should be quite simple

- Just two loops and a bunch of **if…else** statements

Q3 - Fibonacci

- Write a **non-recursive** program that calculates the n-th Fibonacci number
- The Fibonacci numbers are the sequence

$$Fib = 1, 1, 2, 3, 5, 8, 13, \ldots$$

- $Fib(n) = F(n-1) + F(n-2)$

Q3 - Fibonacci

$$Fib = 1, 1, 2, 3, 5, 8, 13, \ldots$$

- How many variables do you need?

- What order should you perform the additions?

- When should you overwrite values?