Functions/Problem Solving

Lecture Overview-Functions

- Lecture
  - Functions (Foundations)
    - What is a Function?
      - Math
      - Programming
    - Motivation for Functions
    - Anatomy of a Function
    - Keeping our Functions
    - C Standard Library (Built-in Functions)
- Before We Code
  - General Info
- First Programs

LECTURE

What is a function?

Motivation for Functions

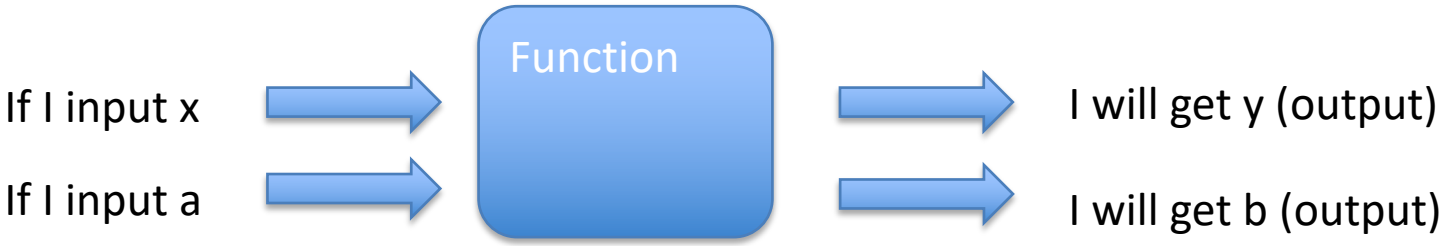
Anatomy of a Function

Keeping our Functions

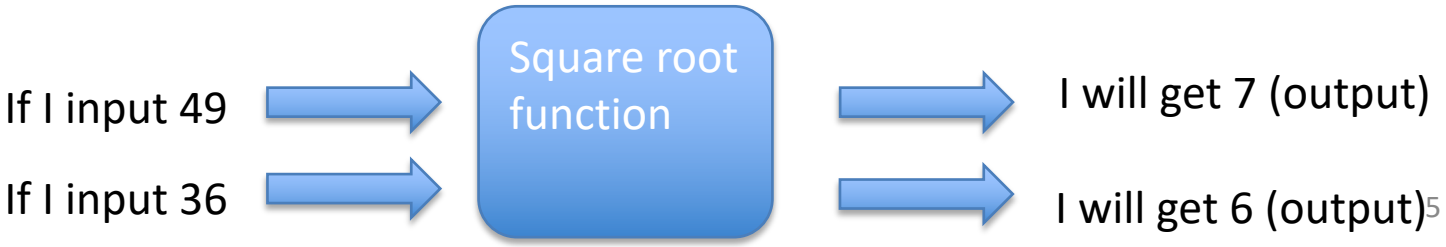
C Standard Library (Built-in Functions)

What is a Function in Math?

- In math, a function is the relationship between a set of inputs and outputs
  - Formal definition: http://mathworld.wolfram.com/Function.html
  - You can think of it like this:

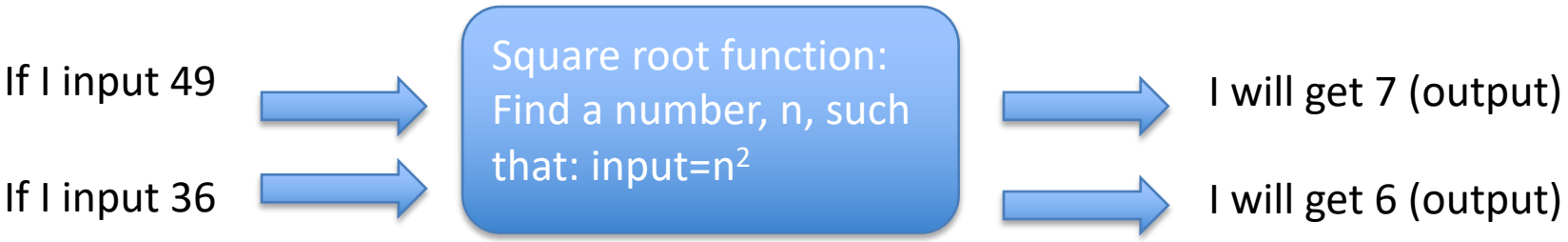


Example:



What is a Function in Math?

- When using a function, the input has certain actions done on it to produce the output
  - This simply means:
    - Use the input to help you arrive at the output



What is a Function in Math?

- You will hear phrases like:
  - y is a function of x
    - This simply means that the value of y relies on the value of x
      - **If x=9, then y=3. If x=16, y=4.**
  - z is a function of a and b
    - This simply means the value of z relies on the value of a and b
      - **If a=1 and b=2, then z=4. If a=2 and b=3, then z=6.**

What is a Function in Math?

- Remember this when we talk about function declarations in C later on in this lecture:
  - y is a function of x — y=f(x)
  - z is a function of a and b — z=g(a, b)

English

Math equivalent

What is a Function in Programming?

- In programming, a function is a collection of programming statements (each themselves doing a task) that are combined to perform a specific task
  - Sometimes you need to give an input (or inputs) to your function in order for it to work (called ***parameters***)
  - Your function can have an output which we signify with a ***return type*** (void means nothing is returned)
  - In 1310, using Java, we called this concept ***methods***

Function analogy:

Input into the function (ingredients)

Inputs (arguments, parameters)

2. We name these wrapped up actions and refer to all these actions as the single overall name

Name: Make cake

1. Mix ingredients
2. Put in oven
3. Take out of oven
4. Put frosting

1. Several individual actions are wrapped up together.

Output from function (cake result of actions on the inputs)

Output (return types, return value)

Function:

2. We name these wrapped up actions and refer to all these actions as the single overall name

Input (a vector):
1 2 1

```
int wallet_total (std::vector <int> money)
{
   if(money.size()==0)
   {
    std::cout<<"Nothing in your wallet yet."<<std::endl;
    return 0;
   }
   int total=0;
   for(int i=0;i<money.size();i++)
   {
    total+=money[i];
   }
   return total;
}
```

1. Several individual actions are wrapped up together.

Note: this is in C++, not C. The concept is the same

4 Output (an integer-the return type of this function is an int)

There are 2 ways to write functions in your code (you will see me doing both in this class):

1

```
#include <stdio.h>

void exp_function(int base, int power)
{
        int i=0;
        int answer=1;

        for(i=0;i<power;i++)
        {
                answer=answer*base;
        }

        printf("answer: %d\n", answer);
}

int main (int argc, char **argv)
{

        exp_function(3, 4);
        exp_function(5, 3);

}
```

2

```
#include <stdio.h>

void exp_function(int base, int power);

int main (int argc, char **argv)
{
        exp_function(3, 4);
        exp_function(5, 3);
}

void exp_function(int base, int power)
{
        int i=0;
        int answer=1;

        for(i=0;i<power;i++)
        {
                answer=answer*base;
        }

        printf("answer: %d\n", answer);
}
```

What is a function?

Motivation for Functions

Anatomy of a Function

Keeping our Functions

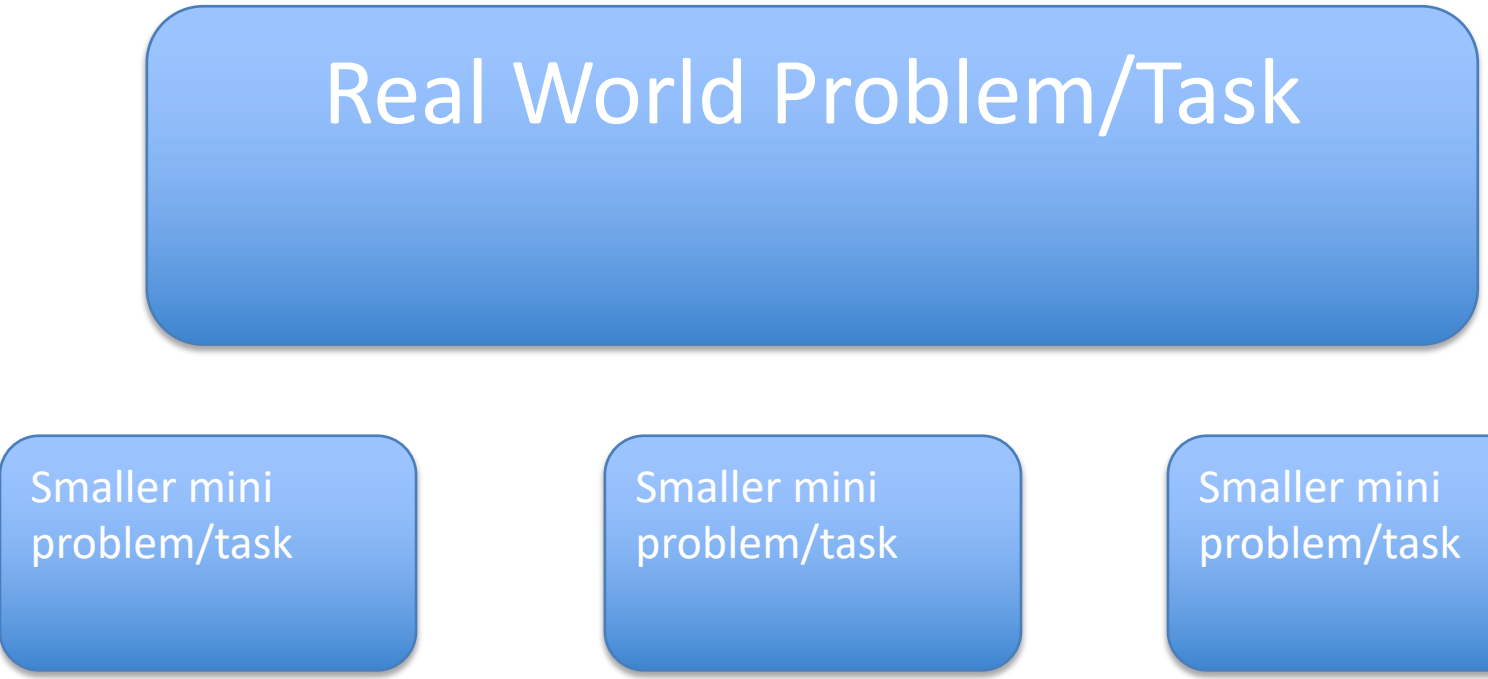
C Standard Library (Built-in Functions)

Motivation for Functions

- Why do we even have functions?
  - Why do we need them?
    - Reuse code
    - Modularize code
- Let's say we have the following problem:
  - Chef Francois has four spots for ingredients in his pantry. Create a program that allows him to enter four ingredients and then allow his sous chef to check if a certain ingredient is in the pantry.

Motivation for Functions

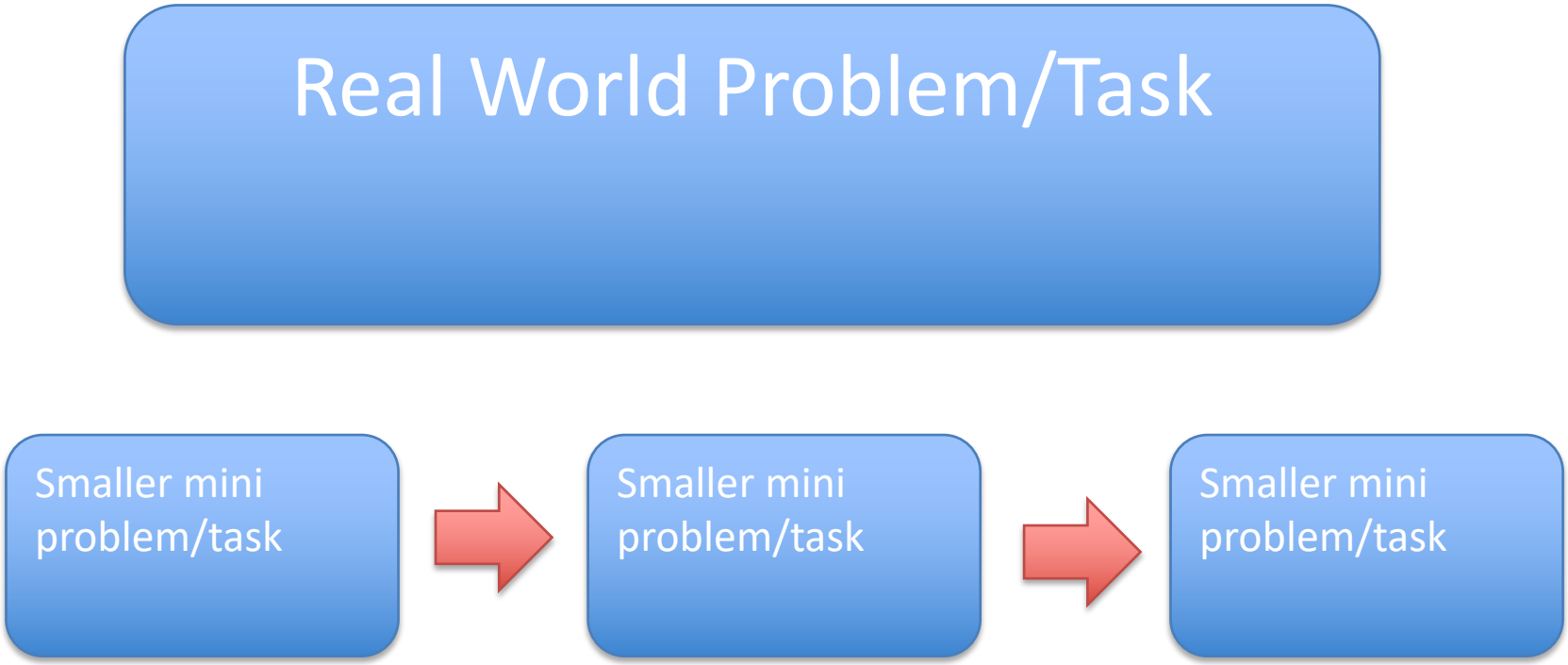
- Before we start, lets look at a general problem solving framework:



Split the problem into smaller subtasks. Solve those smaller subtasks. These will be our **functions.**

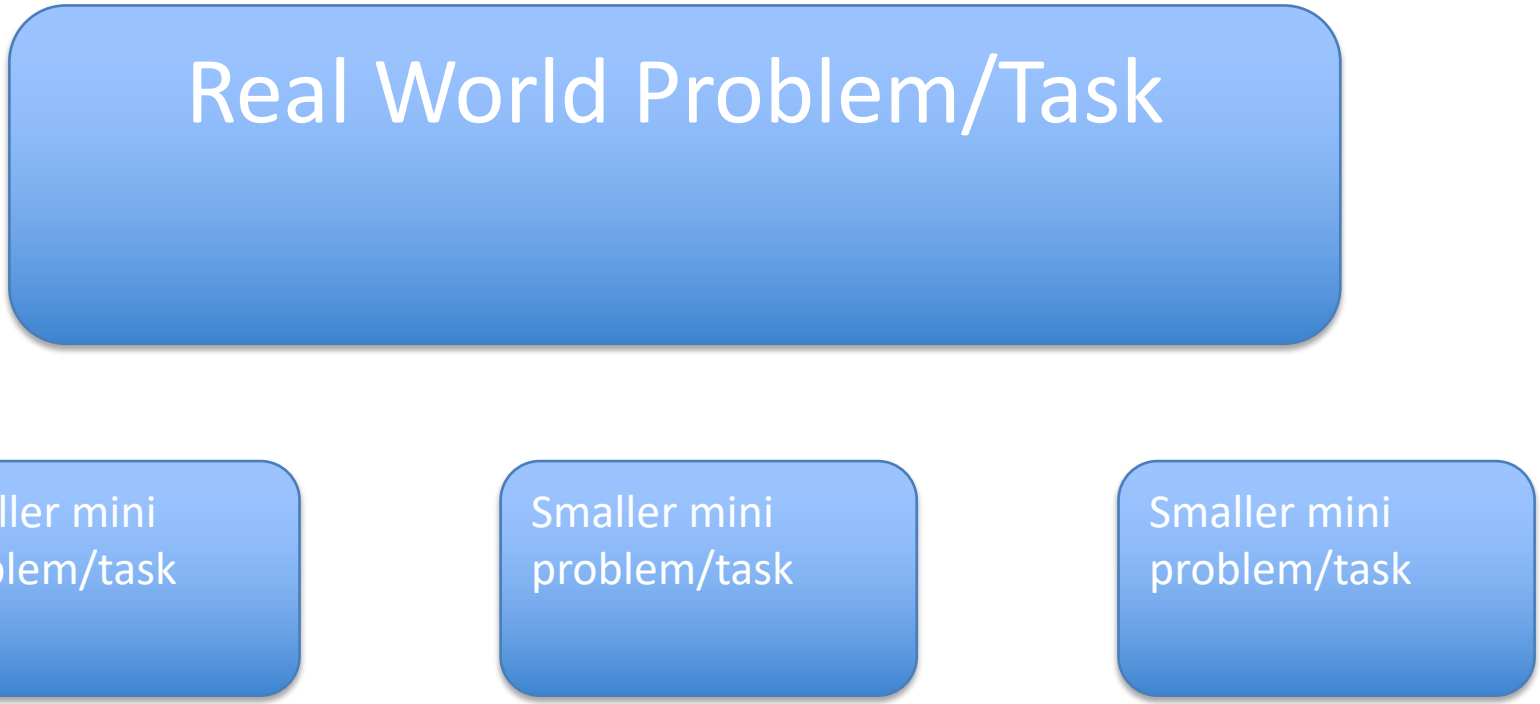
Motivation for Functions

Motivation for Functions



Sometimes we need to solve these tasks in a specific order (i.e. the input of one function relies on the output of another one.)

Motivation for Functions



Other times, we just need them available (they might not even be used in a specific program run). You see this with menu options.

Create a program that allows you to add money to your wallet. The program should keep adding money until it hits a specified goal.

Split the problem into smaller subtasks. Solve those smaller subtasks. These will be our **functions.**

Calculate total money in wallet

If goal was hit, did you go over?

Write lines of code that do this task.

Write lines of code that do this task.

Note: this is in C++, not C. The concept is the same

```
int wallet_total(std::vector <int> money)
{
   if(money.size()==0)
   {
     std::cout<<"Nothing in your wallet yet."<<std::endl;
     return 0;
   }
   int total=0;
   for(int i=0;i<money.size();i++)
   {
     total+=money[i];
   }
   return total;
}
```

```
bool check_total(int total, int goal)
{
  bool b;
  if(total<=goal)
  {
    std::cout<<"You hit your goal without going over."<<std::endl;
    b= true;
  }
  else
  {
    std::cout<<"You went over your goal."<<std::endl;
    b= false;
  }
  return b;
}
```

Create a program that allows you to add money to your wallet. The program should keep adding money until it hits a specified goal.

We can now solve the problem and use the functions…

```
int main(int argc, char **argv)
{
  int goal;
  int add;
  std::vector <int> wallet;

  std::cout<<"Enter your goal amount:"<<std::endl;
  std::cin>>goal;

  while(wallet_total(wallet) <goal)
  {
    std::cout<<"Enter amount to add to wallet: "<<std::endl;
    std::cin>>add;
    wallet.push_back(add);
  }

   check_total(wallet_total(wallet),goal);
}
```

Calculate total money in wallet

Note: this is in C++, not C. The concept is the same

If goal was hit, did you go over?

Create a program that allows you to add money to your wallet. The program should keep adding money until it hits a specified goal.

Notice by using the functions how much cleaner, direct and easier to understand our code is. We wrap up lines of code in a function, give it a name and use it by the name

```
int main(int argc, char **argv)
{
 int goal;
 int add;
 std::vector <int> wallet;

 std::cout<<"Enter your goal amount:"<<std::endl;
 std::cin>>goal;

 while(wallet_total(wallet) <goal)
 {
  std::cout<<"Enter amount to add to wallet: "<<std::endl;
  std::cin>>add;
  wallet.push_back(add);
 }

  check_total(wallet_total(wallet),goal);
}
```

```
int wallet_total(std::vector <int> money)
{
  if(money.size()==0)
  {
   std::cout<<"Nothing in your wallet yet."<<std::endl;
   return 0;
  }
  int total=0;
  for(int i=0;i<money.size();i++)
  {
   total+=money[i];
  }
  return total;
}
```

Note: this is in C++, not C. The concept is the same

```
bool check_total(int total, int goal)
{
 bool b;
 if(total<=goal)
 {
  std::cout<<"You hit your goal without going over."<<std::endl;
  b= true;
 }
 else
 {
  std::cout<<"You went over your goal."<<std::endl;
  b= false;
 }
 return b;
}
```

Motivations

A final consideration for today is the following:

- Notice the fact that solving a problem usually means reducing a problem to information/data and ways to manipulate (actions on) that information/data
- We store the information/data in variables
  - If we have many variables, we can keep them in data structure
- We can then perform actions on these variables
  - It can be a few lines of code
  - If the lines of code combine to do one task, we can store them in a function.

Motivation for Functions

- On the following slides, I give a possible way on how to think to split up problems
  - You don't have to do it this way, just an idea

Motivation for Functions

Real World Problem/Task

1. **What info/data are we dealing with?**
2. **Is there a lot? Should we use a data structure?**

Smaller mini problem/task

Smaller mini problem/task

Smaller mini problem/task

Motivation for Functions

Chef Francois has four spots for ingredients in his pantry. Create a program that allows him to enter four ingredients and then allow his sous chef to check if a certain ingredient is in the pantry.

Think about

1. **What info/data are we dealing with?** Strings (ingredient names)
2. **Is there a lot? Should we use a data structure?** Yes-an array

Enter ingredients	Check if an ingredient is available	Let user know if ingredient is available or not
Do we need a function? Yes	Do we need a function? Yes	Do we need a function? No

Motivation for Functions

Thinking in terms of info/data and doing stuff with that:

Enter ingredients	Check if an ingredient is available	Let user know if ingredient is available or not
Info/data type: strings Stored in array. **Actions:**Go through array and input ingredients.	**Info/data type:** strings Stored in array. **Actions:** Go through array and check if something is present	No need for function
What do we need to do this? An array and size (parameter)	**What do we need to do this?** An array, size, word looking for (string) (parameters)	
Return? Nothing	**Return?** Nothing	
Function declaration: void enter_info(char a [][20], int size)	**Function declaration:** void check (char food_name, char [][20], int size)	

Info/data type: strings
Stored in array.
Actions: Go through array and input ingredients.

What do we need to do this?
An array and size (parameter)

Return? Nothing

Function declaration:
void enter_info(char a [][20], int size)

This is our input (parameter) to the function-what do we need so our function works?

This is our return type for the function-what does the function give us back when it is done?

What goes in-what do we need to do this? (parameter)

What comes out (return)

Motivation for Functions

- There is such a thing as function overkill
  - If you have 20 functions and they each have one line of code, you are probably not breaking up the problem effectively
  - If you have a function that has one single function inside, it's probably overkill
- A good rule of thumb is making sure each function does a meaningful task
  - Just printing out a word is not a good reason for a function
- Different people will see different ways to break up a problem

Motivation for Functions

Final notes:

- Solving a smaller task is a much simpler undertaking
- It makes it easier to work in groups (which you will inevitably be doing in the future)
  - Each person takes a different function
- We can re-use code
  - If we have to solve a task, maybe there is already a function available for us to use

What is a function?

Motivation for Functions

Anatomy of a Function

Keeping our Functions

C Standard Library (Built-in Functions)

Anatomy of a Function

- Let's discuss the general anatomy of a function
  - Declarations
  - Definitions
  - Parameters
  - Arguments
  - Return Types

There are 2 ways to write functions in your code (you will see me doing both in this class):

1

```
#include <stdio.h>

void exp_function(int base, int power)
{
        int i=0;
        int answer=1;

        for(i=0;i<power;i++)
        {
                answer=answer*base;
        }

        printf("answer: %d\n", answer);
}

int main (int argc, char **argv)
{

        exp_function(3, 4);
        exp_function(5, 3);

}
```

Function definition- basically all lines of the actual function

2

Function declaration: Overview of the function

```
#include <stdio.h>

void exp_function(int base, int power);

int main (int argc, char **argv)
{
        exp_function(3, 4);
        exp_function(5, 3);
}


void exp_function(int base, int power)
{
        int i=0;
        int answer=1;

        for(i=0;i<power;i++)
        {
                answer=answer*base;
        }

        printf("answer: %d\n", answer);
}
```

```
#include <stdio.h>

void exp_function(int base, int power);

int main (int argc, char **argv)
{
        exp_function(3, 4);
        exp_function(5, 3);
}

void exp_function(int base, int power)
{
        int i=0;
        int answer=1;

        for(i=0;i<power;i++)
        {
                answer=answer*base;
        }

        printf("answer: %d\n", answer);
}
```

Argument-when you actually use the function and put values in- in this example, 3 and 4 are arguments for the first function call 5 and 3 are arguments in the second function call

Return Type-What type of value the function returns

Parameter -part of the definition. Think of them as reminders to the user that they need to actually give values when using the function.

A quick way to know if it is a parameter is whether or not a variable type (such as int) is present before a variable name.

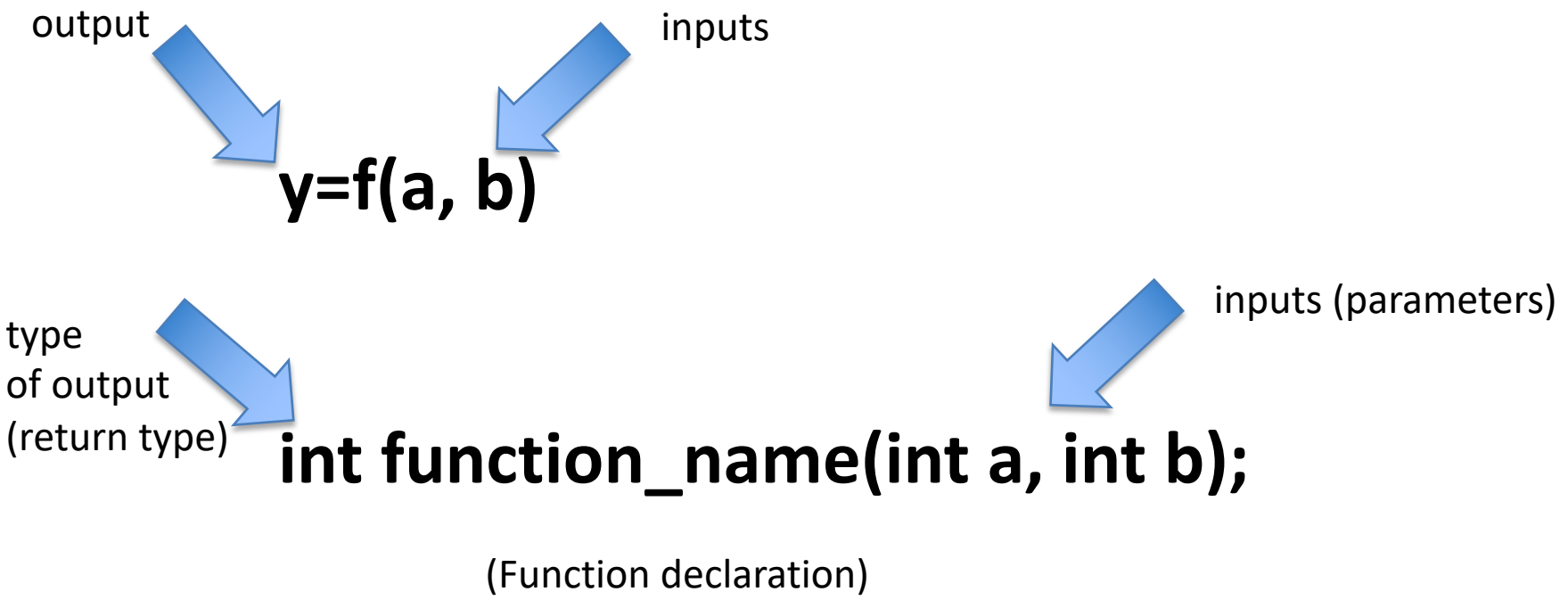
```
#include <stdio.h>

void exp_function(int base, int power);

int main (int argc, char **argv)
{
        exp_function(3, 4);
        exp_function(5, 3);
}


void exp_function(int base, int power)
{
        int i=0;
        int answer=1;

        for(i=0;i<power;i++)
        {
                answer=answer*base;
        }

        printf("answer: %d\n", answer);
}
```

Remember: main is also a function-it has a return type (int) and parameters. Sometimes, you will see main written without a return type or parameters:

```
#include<stdio.h>

main()
{
   printf("Hello World");

}
```

Anatomy of a Function

- Finally, notice how the function declaration is laid out the same way we saw with functions in math:



What is a function?

Motivation for Functions

Anatomy of a Function

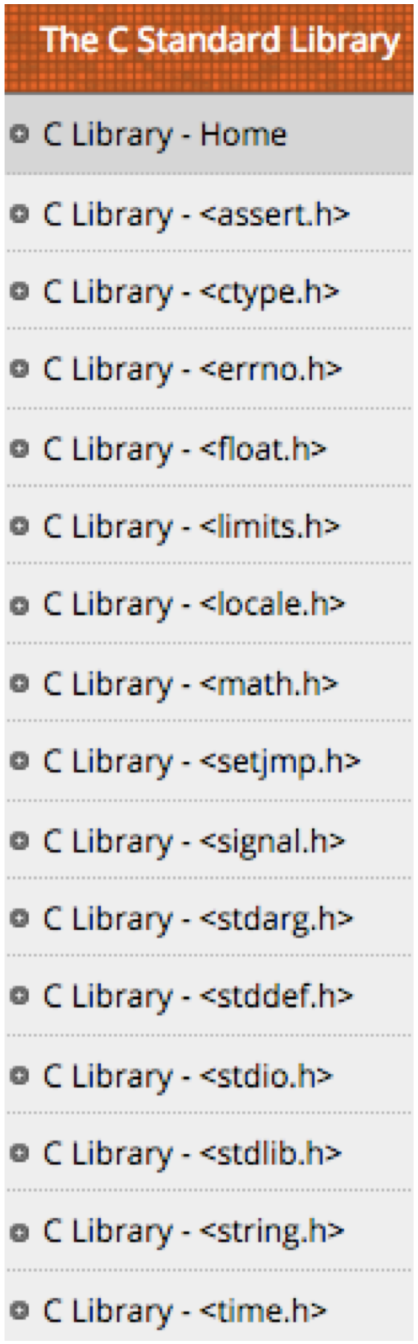
Keeping our Functions

C Standard Library (Built-in Functions)

Keeping Our Functions

- When we create a lot of functions, we can keep them in something called a library
  - We can have them available for use later on
  - We will do this later in the semester-for now, we will keep the functions we create in the same file with our main function

What is a function?
Motivation for Functions
Anatomy of a Function
Keeping our Functions
C Standard Library (Built-in Functions)



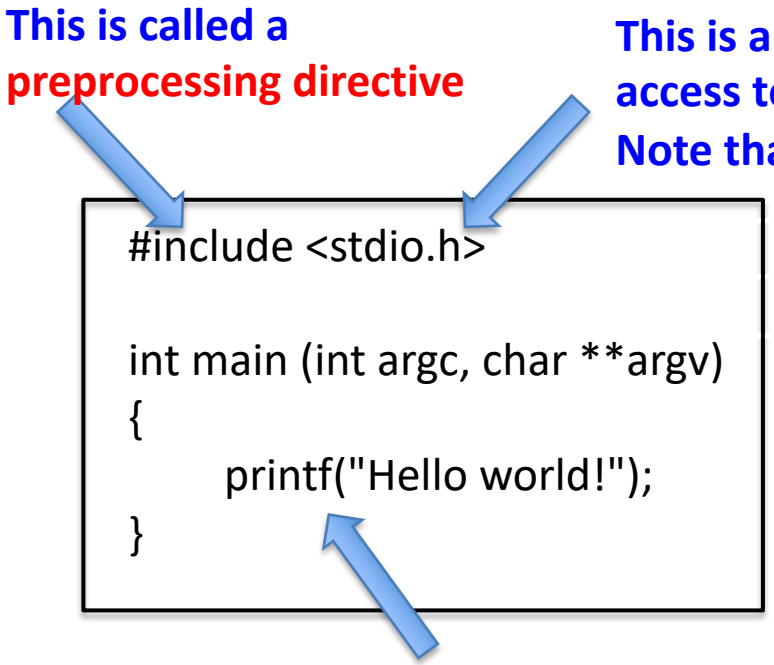
C Standard Library

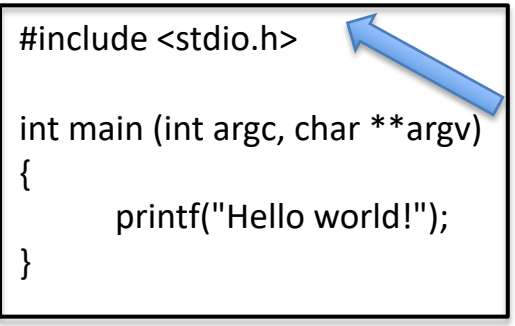
- The C Standard Library is a collection of built-in functions available for you to use

- The printf function (used to output to screen) is kept in the C Standard Library
  - Think how irritating it would be to have to include code to output to screen if we didn't have the printf function
- We can access these functions by including the appropriate header (next slide)

https://www.tutorialspoint.com/c_standard_library/

Let's break down this small program:





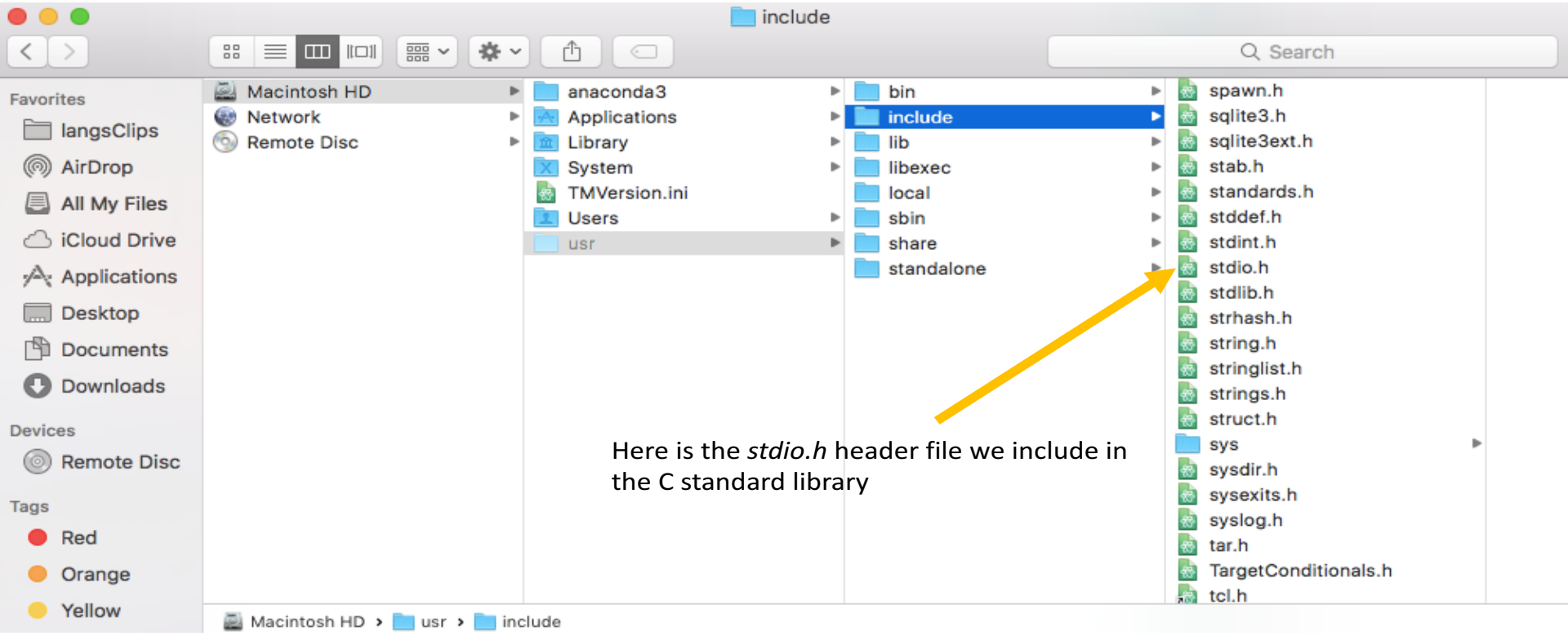
```
#include <stdio.h>

int main (int argc, char **argv)
{
        printf("Hello world!");
}
```

https://www.tutorialspoint.com/c_standard_library/

22	**int printf(const char *format, ...)** Sends formatted output to stdout.
	int sprintf(char *str, const char *format, ...) Sends formatted output to a string.
24	**int vfprintf(FILE *stream, const char *format, va_list arg)** Sends formatted output to a stream using an argument list.
25	**int vprintf(const char *format, va_list arg)** Sends formatted output to stdout using an argument list.
26	**int vsprintf(char *str, const char *format, va_list arg)** Sends formatted output to a string using an argument list.
27	**int fscanf(FILE *stream, const char *format, ...)** Reads formatted input from a stream.
28	**int scanf(const char *format, ...)** Reads formatted input from stdin.
29	**int sscanf(const char *str, const char *format, ...)** Reads formatted input from a string.

To view the complete list of functions in stdio.h :
https://www.tutorialspoint.com/c_standard_library/stdio_h.htm

On my computer:



Here is the *stdio.h* header file we include in the C standard library

Create a program to get the a number from a user. Calculate the square root of this number.

Get user input

Calculate square root

Output this value to screen

Do we need a function?
We already have one
in the C standard library

It is declared in the header **stdio.h:**

int scanf(const char *format, ...)

Do we need a function?
We already have one
in the C standard library

It is declared in the header **math.h:**

double sqrt (double x)

Do we need a function?
We already have one
in the C standard library

It is declared in the header **stdio.h:**

int printf(const char *format, ...)

Lecture Overview-Problem Solving

- Quick Review
- Lecture
  - Problem Solving (Foundations)
    1. Variables and Equations
    2. Reducing a Prompt/Problem
    3. Looking up Info
- Sample Programs

Problem Solving

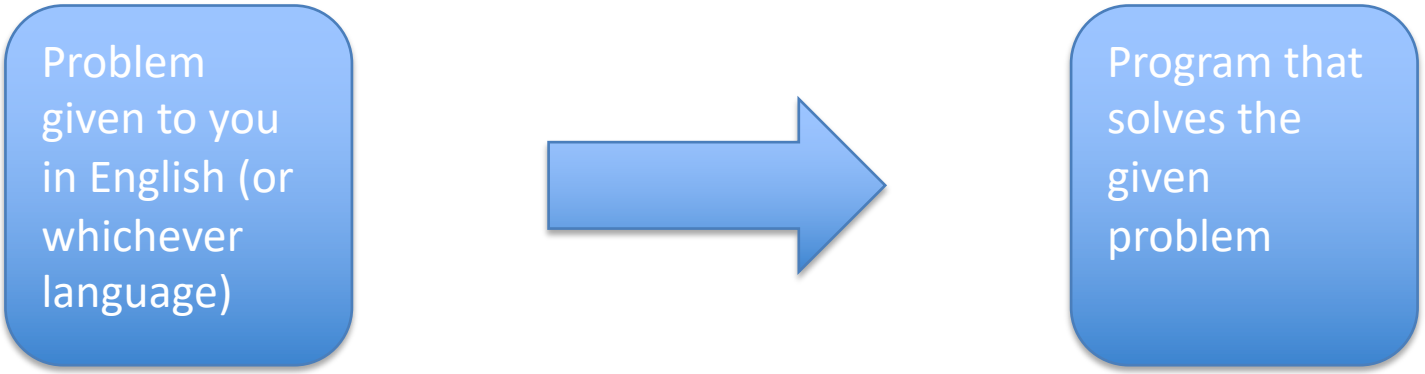
- The idea of problem solving is key to being a successful computer scientist
  - Clients wont tell you what they want
  - You need to explain the world around you to the computer
- It is the problem solving you do before hand that actually gets implemented into code
  - Code is pointless without some “thought” beforehand
  - It’s like someone rambling on and on to you incoherently

Problem Solving

- Let's look at translating a word problem into an equation
  - Remember I said most problems can be thought of as data/info and actions on that data/info?
  - We'll be making equations using that data/info

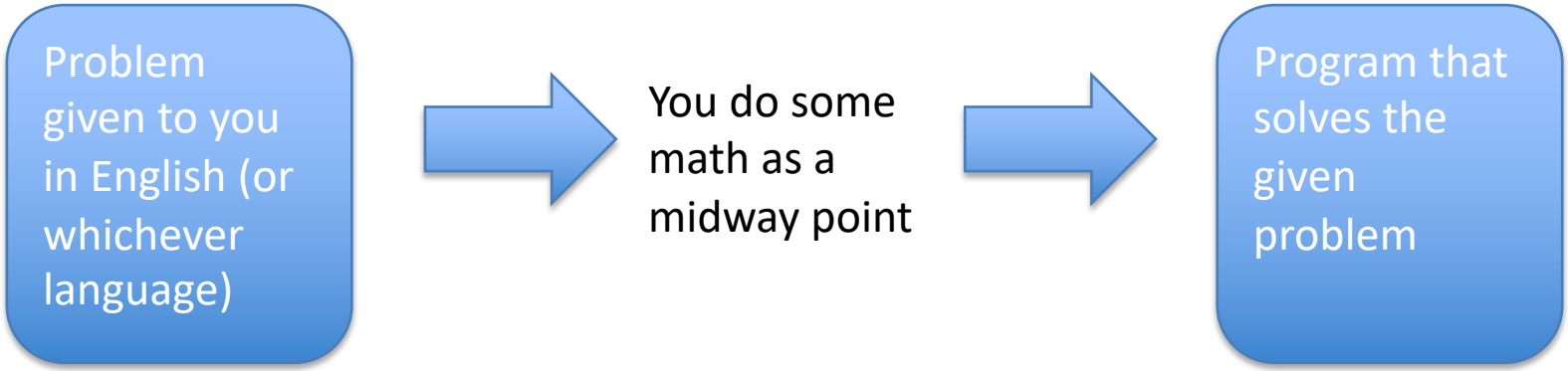
Problem Solving

- Turning English (or whichever language your problem is given in) to code
  - Example: A client telling you what they want a program to do



Problem Solving

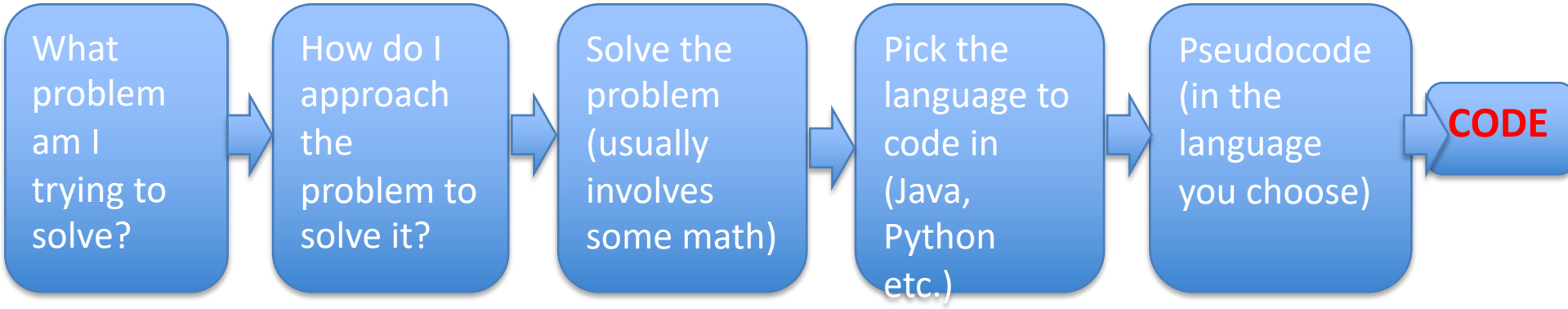
- Turning English (or whichever language your problem is given in) to code
  - Could also be a problem you observe yourself



Problem Solving

- Remember, the computer is a tool we use to solve some real world problem
- Our program is simply a list of instructions to the computer so it can solve our problem
- In order to tell our computer what to do, we ourselves need to know what we want to do

Problem Solving



- If you remember this graphic from the first lecture, notice that coding is the last step

1. **Variables and Equations**
2. Reducing a Prompt/Problem
3. Looking up Info

Variables and Equations

- What is an **expression**?
  - A math phrase containing values (numbers, variables) and operators
  - Evaluates out to some value:
    - 3x+4 4y-2s $7a+2^3$ b
  - Examples: The number of students in a class (3x), the weight of an elephant (2e+4), the age of a person (12)
  - In your programs, **you can hold the value of expressions in variables**
    - Info/data I talked about last time
    - int a=3b+4; /*3b+4 is an expression*/

Variables and Equations

- What is an **equation**?
  - A statement of equality between two expressions
    - $3^4-27x+2y = 7a+4^b$

  - I am saying that the two expressions above are equal to each other
  - Using **properties of equality**, we can manipulate an equation
    - Because we assume it is true our equation is equal

Variables and Equations

- Properties of equality are like the rules I use to make any "moves" on my equation
- Every "move" you make must follow a rule
- For example, you are **not** allowed to do something like this to your equation:

$$2x = 2(3x+5)$$

$$7000+yx+2x = 2(3x+5)^2$$

You can't just randomly add values to one side

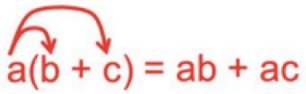
You can't just randomly square one side

Variables and Equations

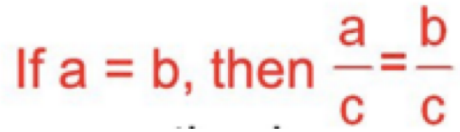
- Some commonly used properties (not all):

Addition Property: if a=b, then a+c=b+c 3x=2y -> 3x+7=2y+7

Subtraction Property: if a=b, then a-c=b-c 3x=2y -> 3x-5=2y-5

Distributive property: if 

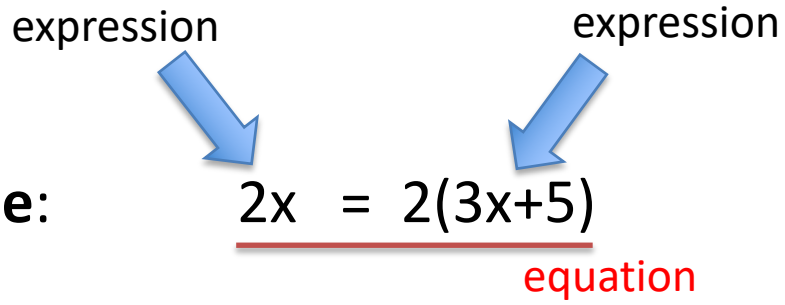
3(x+2)=3x+6 *(6=3*2)*

Division Property: 

3x=2y -> 3x/4=2y/4

(You can look up a full list)

Variables and Equations



Example:

I am saying these two expressions are equal. I can then manipulate this equation to find the unknown value of x.

Step 1:	2x=2*3x +2*5 2x=6x+10	Distributive property
Step 2:	2x-10=6x+10-10 2x-10=6x	Subtraction property
Step 3:	2x-10-2x=6x-2x -10=4x	Subtraction property
Step 4:	-10/4=4x/4 -10/4=x	Division Property

Variables and Equations

Let's now take a look at the following real world scenario:

Jane spent $42 for shoes. This was $14 less than twice what she spent for a blouse. How much was the blouse?

Variables and Equations

Jane spent $42 for shoes. This was $14 less than twice what she spent for a blouse. How much was the blouse?

We are looking for the value of our blouse. Since we do not know the value yet, let's make a variable, **b**, to represent the unknown price.

Using the variable b, let's convert this real world scenario into a mathematical equation (basically looking for quantities and operations on these quantities):

We will set our equation equal to 42 since the price of shoes equals everything else

less implies a subtraction operation

Jane spent **$42 for shoes**. This was **$14 less** than **twice what she spent for a blouse**. How much was the blouse?

Twice what she spent for b (our unknown blouse price) implies a multiplication operation on b

Variables and Equations

Jane spent $42 for shoes. This was $14 less than twice what she spent for a blouse. How much was the blouse?

42=2b-14

We can convert the given real world problem to the following math equation, using the variable b to represent the blouse price.

Variables and Equations

Jane spent $42 for shoes. This was $14 less than twice what she spent for a blouse. How much was the blouse?

42=2b-14

We can convert the given real world problem to the following math equation, using the variable b to represent the blouse price.

We can use our properties of equality to solve for our blouse price, b:

42=2b-14

42+14=2b-14+14 Addition property

56=2b

56/2=2b/2 Division property

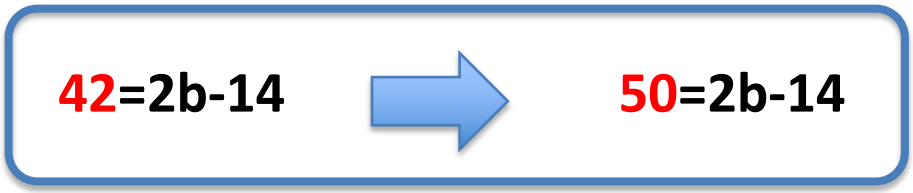
28=b

Variables and Equations

Now, what if we are also given the following scenario:

Gretel spent $50 for shoes. This was $14 less than twice what she spent for a blouse. How much was the blouse?

We can use the same equation as before, but now the total shoe price is 50 instead of 42:



How can we account for this? Instead of hardcoding the values 42 or 50, let's use a variable, **s**, to represent the shoe price:

By letting **s=42** OR **s=50** we can solve both scenarios given.

Variables and Equation

The same idea follows if want to handle this situation:

Sonya spent $60 for shoes. This was $20 less than twice what she spent for a blouse. How much was the blouse?

- We can handle the new shoe price of $60 since we have s as our shoe variable
- But now we're saying $20 less, not $14 less
- Lets create a new variable, called l:

s=2b-14 → **s=2b-l**

Variables and Equations

Finally:

Zaza spent $70 for shoes. This was $10 less than three times what she spent for a blouse. How much was the blouse?

- Now we have to handle 3x as much, no longer 2x as much. Let's create a variable called t:

s=tb-l

We now have a general equation.

By generalizing the whole equation (and using variables), we can now have a computer program that can handle **any** of the preceding situations. We simply assign values to the variables accordingly.

Our program is not limited.

Variables and Equations

$s=tb-l$

Using the properties of equality, solve for b

Step 1: $s+l=tb-l+l$ Addition property

$s+l=tb$

Step 2: $s+l/t=tb/t$ Division property

$s+l/t=b$

This equation is what we will turn into code

Variables and Equations

Additionally:

We could say (from the previous lecture):

– b is a function of s, l, t b=f(s, l, t)

(Remember, this just means that the value of b is dependent on the values of s, l and t)

Where f is the blouse price function

Variables and Equations

Additionally:

We could say (from the previous lecture):

– b is a function of s, l, t b=f(s, l, t)

(Remember, this just means that the value of b is dependent on the values of s, l and t)

Where f is the blouse price function

Note: I would only make this an actual function in my program if I do more than just use the equation (see the following two slides)

```
#include <stdio.h>

int main (int argc, char **argv)
{
	int s, l, t, b;

	/*enter values*/
	printf("Enter price of shoes: $");
	scanf("%d", &s);

	printf("Enter value less than blouse: ");
	scanf("%d", &l);

	printf("Enter value times blouse: ");
	scanf("%d", &t);

	b=(s+l)/t;

	printf("Blouse price is $: %d\n", b);

}
```

I just used this line to calculate the blouse price. I don't see a need to put this in a function. Someone else might decide to do so.

This code solves any of the previous blouse questions.

Notes:
-The more scenarios your code can handle, the better.

-Using variables instead of hardcoding

```
#include <stdio.h>

int blouse_price(int s, int l, int t);

int main (int argc, char **argv)
{
	int s, l, t, b;

	/*enter values*/
	printf("Enter price of shoes: $");
	scanf("%d", &s);

	printf("Enter value less than blouse: ");
	scanf("%d", &l);

	printf("Enter value times blouse: ");
	scanf("%d", &t);

	b=blouse_price(s, l, t);

	if(b==-1) /*blouse price too low*/
	{
		printf(”Blouse price too low to calculate.\n");
	}
	else
	{
		printf("Blouse price is $: %d\n", b);
	}
}

int blouse_price(int s, int l, int t)
{
	int ret;
	/*check that blouse price is at least 40*/
	if(s<40)
	{
		ret=-1;
	}
	else
	{
		ret=(s+l)/t;
	}

	return ret;

}
```

(I made a function here because I’m doing more than just calculating one line)

Variables and Equations

Generally for quantities:

triple, quadruple etc.

twice, four times, five times etc.

- If you see words like double or three times, you might be multiplying: Machine language instructions are quadruple the instructions you code.
  - int your_code_instructions=60;
  - int machine_instructions=your_code_instructions*4;

- The opposite of this is division: I made 3x what he made. (so made he made 1/3 of what I did)

  double my_salary=120000;

  double his_salary=my_salary/3;

Variables and Equations

- If you see words like gained, received, got, more than, greater than you might be adding:
  She received her first lotto check yesterday.

```
double money_in_bank, lotto;
money_in_bank=10.75;
lotto=500000.99;
money_in_bank=money_in_bank+lotto;
```

Variables and Equations

- If you see words like less than, lost, gave, spent, bought you might be subtracting: He lost some of his students after the drop date.

```
int student_total=12;
int student_dropped;
printf("How many students dropped? ");
scanf("%d", &student_dropped);
student_total=student_total-student_dropped;
```

1. Variables and Equations
2. **Reducing a Prompt/Problem**
3. Looking up Info

Reducing a Prompt/Problem

- When given a prompt or real world problem (or sub-problem), it's a good idea to try and reduce the problem to the actual task you are trying to solve
  - It will help write a better program
  - It can also help with code reusability if multiple problems have the same task
    - We could write a function for the task and use it with different problems

Reducing a Prompt/Problem

- Jameson paints ceilings for $3 dollars/square foot. Create a program to help calculate the total price of a job.

- ABC Lawn will only mow lawns that 50 sq. ft or larger. Create a program to decide if a lawn can be mowed or not.

- Party Planners needs to know how many brochures are needed to cover a table. Create a program to help them with this.

(for simplicity: let the ceilings, lawns, tables and brochures be rectangular shaped for this example)

Reducing a Prompt/Problem

Even though the prompts seem completely different and unrelated, parts of what they are asking you to do are the same.

In all cases, you need to calculate the area of a rectangle at some point. You can make a function that does this for you and use it for all the problems.

- That way, we can reuse code (the function) instead of writing it again
- We can use the idea of variables (discussed in the previous section) and pass in different values that we need to calculate

Reducing a Prompt/Problem

Which ones are actually asking you to do the same task?

- See if at least one student made an A from a list of letter grades
- Given a list of yes or no answers (Y is yes, N is no) see if anyone said yes
- Given a list of letter grades, see if the total number of C students is less than 10

Which ones are actually asking you to do the same task?

- See if at least one student made an A from a list of letter grades

You are looking if a letter is present from a list of letters-yes it is or no it is not.

SAME TASK

- Given a list of yes or no answers (Y is yes, N is no) see if anyone said yes

You are looking if a letter is present from a list of letters-yes it is or no it is not.

SAME TASK

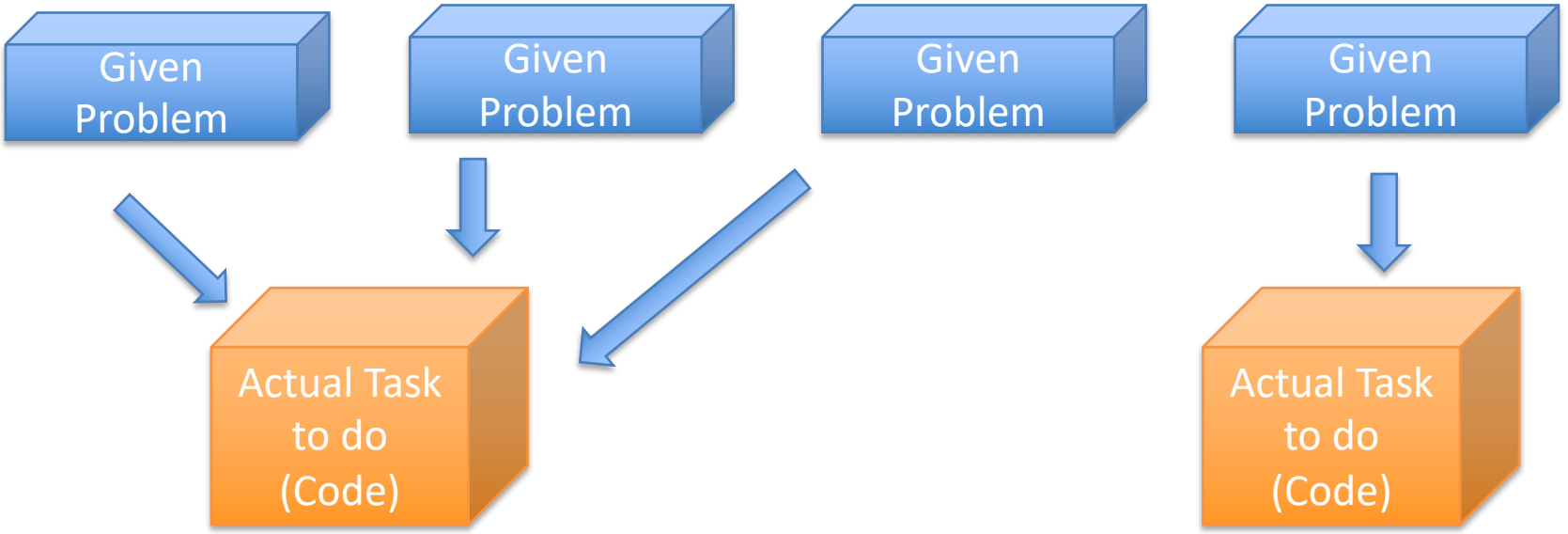
- Given a list of letter grades, see if the total number of C students is less than 10

DIFFERENT TASK

You are counting the number of times a letter appears in a list of letters-you will end up with an integer

Reducing a Prompt/Problem

- Even though tasks may look similar on the outside (same topic or same objects used), it is important to try to reduce to what you are **REALLY** trying to do



Reducing a Prompt/Problem

- Note that if you use different data structures to hold information, you may have no choice but to re-write code
  - If you use an array for one problem and a linked list for another, you will have to go through the data structure in a different manner.
    - This means the code won't be the same

1. Variables and Equations
2. Reducing a Prompt/Problem
3. **Looking up Info**

Looking Up Info

- Looking up information
- The concept of looking up is simple but important
  - I've noticed students get stuck when they are not directly given all the information in a problem
    - Worse, sometimes they blindly guess what something means
  - Being able to figure things out is the heart of computer science
- We can look up equations
- We can look up what defines something

Looking Up Info

- We can look up equations:
  - Farees drove from Arlington to Dallas, 25 miles away. He left Arlington at 7pm and arrived in Dallas at 7:30. He drove the same speed the whole time. How fast was he going?
    - We're looking for his speed.
    - We have an equation for that:
      - Speed=distance/time

Looking Up Info

- We can look up what defines something
  - Antoinette is a 13 year old girl living in Strasbourg. She wants to know how many more years until she can get her drivers license.
    - Look up-where is Strasbourg? France
    - Look up-what is the legal age to get a drivers license in France?
  - Texas might not have enough funding for the CHIP program. Would your family be affected?
    - Look up-What is CHIP?
    - Look up-What determines whether you will be affected? Family income level? Age?

Looking Up Info

- You may have some assignments where you will have to look up information
  - I will **NOT** specify to you that you need to look something up
  - I will of course **always** help with instructions and concepts for the assignments themselves
    - Help breaking up a problem, for example
  - Future clients will not always explicitly tell you what something is