Mr. C can Float!

ESC101: Foundations of Computing

Purushottam Kar

Announcements - Holiday

- No lecture, no lab this Wednesday (institute holiday)
- Extra lecture this Saturday 18 August 2018
  - 12 noon, L20 (same as usual)
- Extra lab for B10, B11, B12, B14 this Saturday 18 August
  - 2PM – 5PM, New Core Labs CC-01 and CC-02 (same as usual)

Announcements – Marks

- Lab marks and minor quiz marks for a particular week will be released after all labs for next week are over
  - Marks for last week will be released after Saturday lab this week
  - Lab problems already released as practice problems – practice!
  - Hidden test cases for these lab-practice problems will also be released
- Your questions were auto-graded – no grace marks!
- If you discover, autograder made a mistake (hidden test case), you may request for regrading
- Useless regrading requests **will be given negative marks** – be very careful before making request!

A handy Tip for Prutor

A handy Tip for Prutor

How to check if you have extra spaces or not ☺

A handy Tip for Prutor

How to check if you have extra spaces or not ☺

10:05:40 AM

Evaluation Results

Your program passed 0 out of 1 visible test case(s).

#	INPUT	EXPECTED OUTPUT	ACTUAL OUTPUT	
1	5	/---\ |%%%| |%5%| |%%%| \---/	/---\ |%%%| |%5%| |%%%| \---/	✖

```
1  #include<stdio.h>
2
3  int main(){
4      int val;
5      scanf("%d", &val);
6
7      printf("/---\ \n");
8      printf("|%%%%%%| \n");
9      printf("|%%%d%%| \n",val);
10     printf("|%%%%%%|\n");
11     printf("\\---/\n");
12     return 0;
13 }
```

A handy Tip for Prutor

How to check if you have extra spaces or not ☺

10:05:40 AM

Evaluation Results

Your program passed 0 out of 1 visible test case(s).

#	INPUT	EXPECTED OUTPUT	ACTUAL OUTPUT	
1	5	/---\ |%%%| |%5%| |%%%| \---/	/---\ |%%%| |%5%| |%%%| \---/	✖

```
1  #include<stdio.h>
2
3  int main(){
4      int val;
5      scanf("%d", &val);
6
7      printf("/---\\ \n");
8      printf("|%%%%%%| \n");
9      printf("|%%%d%%| \n",val);
10     printf("|%%%%%%|\n");
11     printf("\\---/\n");
12     return 0;
13 }
```

A handy Tip for Prutor

How to check if you have extra spaces or not ☺

10:05:40 AM

Evaluation Results

Your program passed 0 out of 1 visible test case(s).

#	INPUT	EXPECTED OUTPUT	ACTUAL OUTPUT	
1	5	/---\ |%%%| |%5%| |%%%| \---/	/---\ |%%%| |%5%| |%%%| \---/	✖

```
1  #include<stdio.h>
2
3  int main(){
4      int val;
5      scanf("%d", &val);
6
7      printf("/---\\ \n");
8      printf("|%%%%%%| \n");
9      printf("|%%%d%%| \n",val);
10     printf("|%%%%%%|\n");
11     printf("\\---/\n");
12     return 0;
13 }
```

A handy Tip for Prutor

How to check if you have extra spaces or not ☺

10:05:40 AM

Evaluation Results

Your program passed 0 out of 1 test case(s).

#	INPUT	EXPECTED OUTPUT	ACTUAL OUTPUT	
1	5	/---\ |%%%| |%5%| |%%%| \---/	/---\ |%%%| |%5%| |%%%| \---/	✖

Select both the expected output and your output. Any extra spaces will get revealed.

```
1  #include<stdio.h>
2
3  int main(){
4      int val;
5      scanf("%d", &val);
6      
                -\\ \n");
                %%%%| \n");
                %d%%| \n",val);
                %%%%|\n");
11     printf("\\---/\n");
12     return 0;
13 }
```

Hidden trick with Integer division

Hidden trick with Integer division

Integer division might seem useless 2 / 3 = 0, 5 / 2 = 2 ☹

Hidden trick with Integer division

Integer division might seem useless 2 / 3 = 0, 5 / 2 = 2 ☹

Have to use % to obtain remainder 2 % 3 = 2, 5 % 2 = 1 😐

Hidden trick with Integer division

Integer division might seem useless 2 / 3 = 0, 5 / 2 = 2 ☹

Have to use % to obtain remainder 2 % 3 = 2, 5 % 2 = 1 😐

Be careful when doing the above with negative integers

Hidden trick with Integer division

Integer division might seem useless 2 / 3 = 0, 5 / 2 = 2 ☹

Have to use % to obtain remainder 2 % 3 = 2, 5 % 2 = 1 😐

Be careful when doing the above with negative integers

- Different C compilers handle sign a bit differently

Hidden trick with Integer division

Integer division might seem useless 2 / 3 = 0, 5 / 2 = 2 ☹

Have to use % to obtain remainder 2 % 3 = 2, 5 % 2 = 1 😐

Be careful when doing the above with negative integers

- Different C compilers handle sign a bit differently
- In Prutor, 5 % 2 = 1 but 5 % -2 = 1 as well

Hidden trick with Integer division

Integer division might seem useless 2 / 3 = 0, 5 / 2 = 2 ☹

Have to use % to obtain remainder 2 % 3 = 2, 5 % 2 = 1 😐

Be careful when doing the above with negative integers

- Different C compilers handle sign a bit differently
- In Prutor, 5 % 2 = 1 but 5 % -2 = 1 as well
- However, all compilers ensure that a = (a/b) * b + a % b

Hidden trick with Integer division

Integer division might seem useless 2 / 3 = 0, 5 / 2 = 2 ☹

Have to use % to obtain remainder 2 % 3 = 2, 5 % 2 = 1 😐

Be careful when doing the above with negative integers

- Different C compilers handle sign a bit differently
- In Prutor, 5 % 2 = 1 but 5 % -2 = 1 as well
- However, all compilers ensure that a = (a/b) * b + a % b

Integer division can be used to extract digits of an integer

Hidden trick with Integer division

Integer division might seem useless 2 / 3 = 0, 5 / 2 = 2 ☹

Have to use % to obtain remainder 2 % 3 = 2, 5 % 2 = 1 😐

Be careful when doing the above with negative integers

- Different C compilers handle sign a bit differently
- In Prutor, 5 % 2 = 1 but 5 % -2 = 1 as well
- However, all compilers ensure that a = (a/b) * b + a % b

Integer division can be used to extract digits of an integer

12345 % 10 = 5 (the last digit)

Hidden trick with Integer division

Integer division might seem useless 2 / 3 = 0, 5 / 2 = 2 ☹

Have to use % to obtain remainder 2 % 3 = 2, 5 % 2 = 1 😐

Be careful when doing the above with negative integers

- Different C compilers handle sign a bit differently
- In Prutor, 5 % 2 = 1 but 5 % -2 = 1 as well
- However, all compilers ensure that a = (a/b) * b + a % b

Integer division can be used to extract digits of an integer

12345 % 10 = 5 (the last digit)

12345 / 10 = 1234 (the remaining digits)

Hidden trick with Integer division

Integer division might seem useless 2 / 3 = 0, 5 / 2 = 2 ☹

Have to use % to obtain remainder 2 % 3 = 2, 5 % 2 = 1 😐

Be careful when doing the above with negative integers

- Different C compilers handle sign a bit differently
- In Prutor, 5 % 2 = 1 but 5 % -2 = 1 as well
- However, all compilers ensure that a = (a/b) * b + a % b

Integer division can be used to extract digits of an integer

12345 % 10 = 5 (the last digit)

12345 / 10 = 1234 (the remaining digits)

Hidden trick with Integer division

Integer division might seem useless 2 / 3 = 0, 5 / 2 = 2 ☹

Have to use % to obtain remainder 2 % 3 = 2, 5 % 2 = 1 😐

Be careful when doing the above with negative integers

- Different C compilers handle sign a bit differently
- In Prutor, 5 % 2 = 1 but 5 % -2 = 1 as well
- However, all compilers ensure that a = (a/b) * b +

Integer division can be used to extrac eger

12345 % 10 = 5 (the last digit)

12345 / 10 = 1234 (the remaining digits)

Hidden trick with Integer division

Integer division might seem useless 2 / 3 = 0, 5 / 2 = 2 ☹

Have to use % to obtain remainder 2 % 3 = 2, 5 % 2 = 1 😐

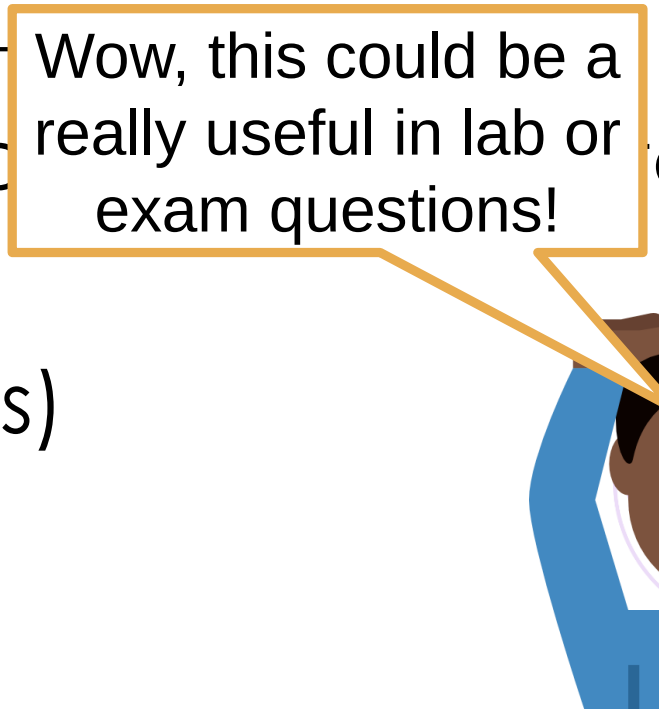
Be careful when doing the above with negative integers

- Different C compilers handle sign a bit differently
- In Prutor, 5 % 2 = 1 but 5 % -2 = 1 as well
- However, all compilers ensure that a = (a/b) * b +

Integer division can be used to extrac eger

12345 % 10 = 5 (the last digit)

123 10 = 1234 (the remaining digits)



Hidden trick with Integer division

Integer division might seem useless 2 / 3 = 0, 5 / 2 = 2 ☹

Have to use % to obtain remainder 2 % 3 = 2, 5 % 2 = 1 😐

Be careful when doing the above with negative integers

- Different C compilers handle sign a bit differently
- In Prutor, 5 % 2 = 1 but 5 % -2 = 1 as well
- However, all compilers ensure that a = (a/b) * b + (a%b)

Integer division can be used to extract digits of an integer

12345 % 10 = 5 (the last digit)

12345 / 10 = 1234 (the remaining digits)

Yup, my solution for the bonus problem used this exact same trick ☺

Wow, this could be a really useful in lab or exam questions!

Hidden trick with Integer division

Integer division might seem useless 2 / 3 = 0, 5 / 2 = 2 ☹

Have to use % to obtain remainder 2 % 3 = 2, 5 % 2 = 1 😐

Be careful when doing the above with negative integers

- Different C compilers handle sign a bit differently
- In Prutor, 5 % 2 = 1 but 5 % -2 = 1 as well
- However, all compilers ensure that a = (a/b) * b + (a%b)

Integer division and % can be used to extract digits from an integer

12345 % 10 = 5 (the last digit)

12345 / 10 = 1234 (the remaining digits)

Yup, my solution for the bonus problem used this exact same trick ☺

Wow, this could be a really useful in lab or exam questions!

Oh, Hi Puru. Nice of you to drop by

Rounding errors with Integer division

Rounding errors with Integer division

Recall that integer division throws away decimal portion

Rounding errors with Integer division

Recall that integer division throws away decimal portion

Errors can accumulate if done too many times

Rounding errors with Integer division

Recall that integer division throws away decimal portion

Errors can accumulate if done too many times

Minimize the number of times you do integer division

Rounding errors with Integer division

Recall that integer division throws away decimal portion

Errors can accumulate if done too many times

Minimize the number of times you do integer division

Rounding errors with Integer division

Recall that integer division throws away decimal portion

Errors can accumulate if done too many times

Minimize the number of times you do integer division

```
#include <stdio.h>
```

Rounding errors with Integer division

Recall that integer division throws away decimal portion

Errors can accumulate if done too many times

Minimize the number of times you do integer division

```
#include <stdio.h>
int main(){
```

Rounding errors with Integer division

Recall that integer division throws away decimal portion

Errors can accumulate if done too many times

Minimize the number of times you do integer division

```
#include <stdio.h>
int main(){
    int a = 5, b = 7;
```

Rounding errors with Integer division

Recall that integer division throws away decimal portion

Errors can accumulate if done too many times

Minimize the number of times you do integer division

```
#include <stdio.h>
int main(){
    int a = 5, b = 7;
    printf("Average %d", a/2 + b/2);
```

Rounding errors with Integer division

Recall that integer division throws away decimal portion

Errors can accumulate if done too many times

Minimize the number of times you do integer division

```
#include <stdio.h>
int main(){
    int a = 5, b = 7;
    printf("Average %d", a/2 + b/2);
    return 0;
```

Rounding errors with Integer division

Recall that integer division throws away decimal portion

Errors can accumulate if done too many times

Minimize the number of times you do integer division

```
#include <stdio.h>
int main(){
    int a = 5, b = 7;
    printf("Average %d", a/2 + b/2);
    return 0;
}
```

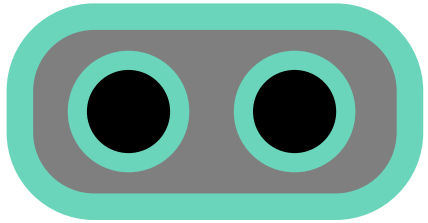
Rounding errors with Integer division

Recall that integer division throws away decimal portion

Errors can accumulate if done too many times

Minimize the number of times you do integer division

```
#include <stdio.h>
int main(){
    int a = 5, b = 7;
    printf("Average %d", a/2 + b/2);
    return 0;
}
```



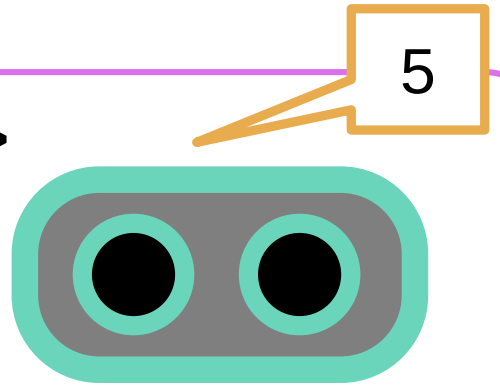
Rounding errors with Integer division

Recall that integer division throws away decimal portion

Errors can accumulate if done too many times

Minimize the number of times you do integer division

```
#include <stdio.h>
int main(){
    int a = 5, b = 7;
    printf("Average %d", a/2 + b/2);
    return 0;
}
```

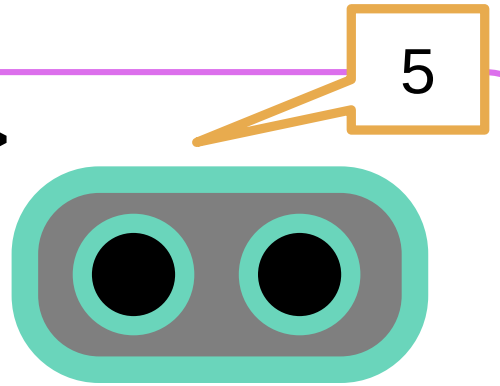


Rounding errors with Integer division

Recall that integer division throws away decimal portion

Errors can accumulate if done too many times

Minimize the number of times you do integer division




```
#include <stdio.h>
int main(){
    int a = 5, b = 7;
    printf("Average %d", a/2 + b/2);
    return 0;
}
```

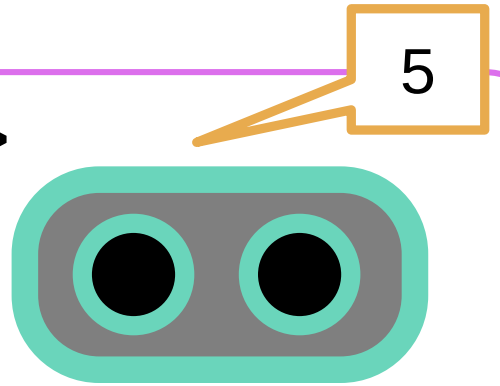
Rounding errors with Integer division

Recall that integer division throws away decimal portion

Errors can accumulate if done too many times

Minimize the number of times you do integer division

```
#include <stdio.h>
int main(){
    int a = 5, b = 7;
    printf("Average %d", a/2 + b/2);
    return 0;
}
```




```
#include <stdio.h>
```

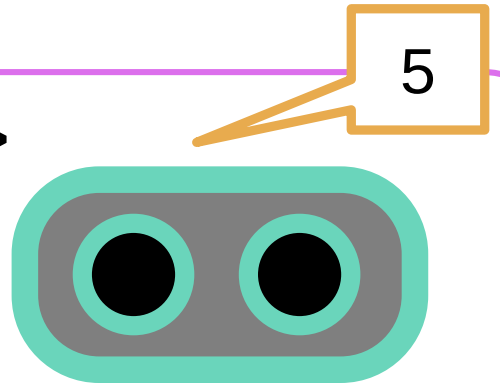
Rounding errors with Integer division

Recall that integer division throws away decimal portion

Errors can accumulate if done too many times

Minimize the number of times you do integer division

```
#include <stdio.h>
int main(){
    int a = 5, b = 7;
    printf("Average %d", a/2 + b/2);
    return 0;
}
```




```
#include <stdio.h>
int main(){
```

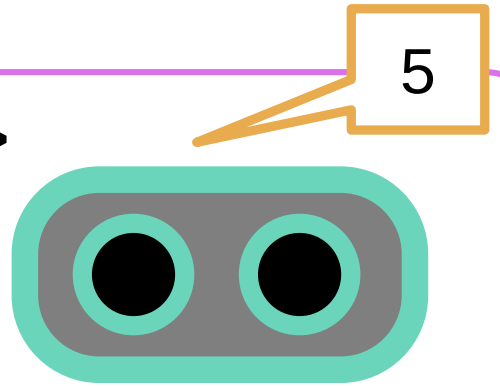
Rounding errors with Integer division

Recall that integer division throws away decimal portion

Errors can accumulate if done too many times

Minimize the number of times you do integer division

```
#include <stdio.h>
int main(){
    int a = 5, b = 7;
    printf("Average %d", a/2 + b/2);
    return 0;
}
```




```
#include <stdio.h>
int main(){
    int a = 5, b = 7;
```

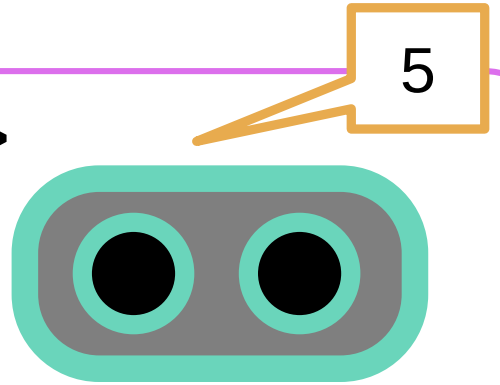
Rounding errors with Integer division

Recall that integer division throws away decimal portion

Errors can accumulate if done too many times

Minimize the number of times you do integer division

```
#include <stdio.h>
int main(){
    int a = 5, b = 7;
    printf("Average %d", a/2 + b/2);
    return 0;
}
```




```
#include <stdio.h>
int main(){
    int a = 5, b = 7;
    printf("Average %d", (a + b)/2);
```

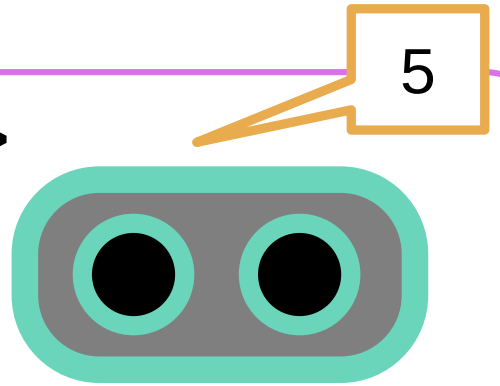
Rounding errors with Integer division

Recall that integer division throws away decimal portion

Errors can accumulate if done too many times

Minimize the number of times you do integer division

```
#include <stdio.h>
int main(){
    int a = 5, b = 7;
    printf("Average %d", a/2 + b/2);
    return 0;
}
```




```
#include <stdio.h>
int main(){
    int a = 5, b = 7;
    printf("Average %d", (a + b)/2);
    return 0;
```

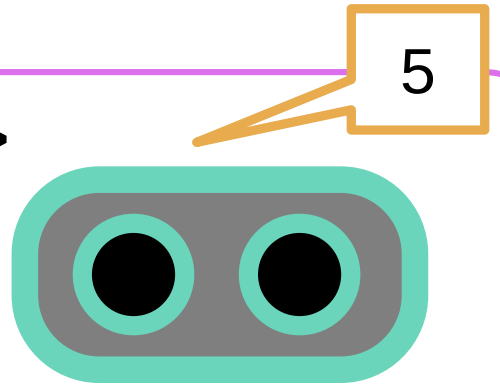
Rounding errors with Integer division

Recall that integer division throws away decimal portion

Errors can accumulate if done too many times

Minimize the number of times you do integer division

```
#include <stdio.h>
int main(){
    int a = 5, b = 7;
    printf("Average %d", a/2 + b/2);
    return 0;
}
```




```
#include <stdio.h>
int main(){
    int a = 5, b = 7;
    printf("Average %d", (a + b)/2);
    return 0;
}
```

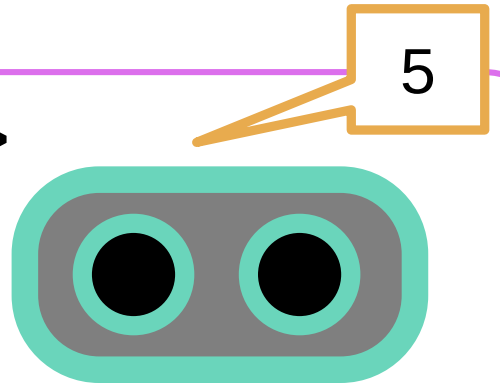
Rounding errors with Integer division

Recall that integer division throws away decimal portion

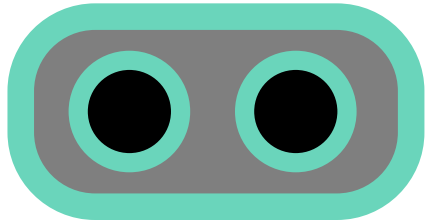
Errors can accumulate if done too many times

Minimize the number of times you do integer division

```
#include <stdio.h>
int main(){
    int a = 5, b = 7;
    printf("Average %d", a/2 + b/2);
    return 0;
}
```




```
#include <stdio.h>
int main(){
    int a = 5, b = 7;
    printf("Average %d", (a + b)/2);
    return 0;
}
```



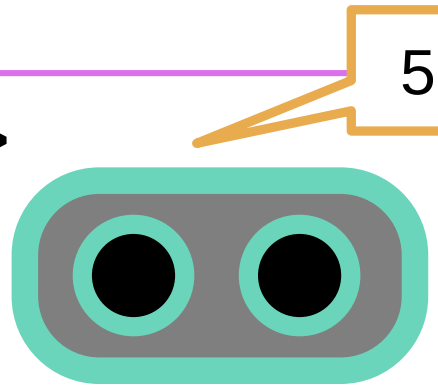
Rounding errors with Integer division

Recall that integer division throws away decimal portion

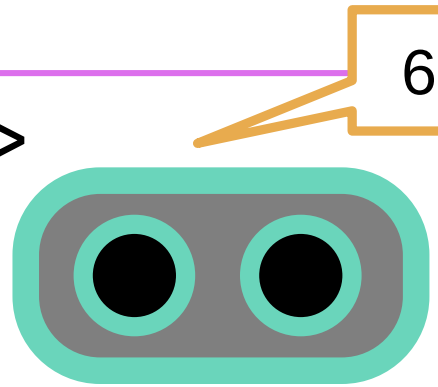
Errors can accumulate if done too many times

Minimize the number of times you do integer division

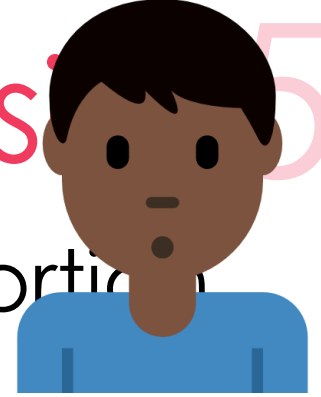
```
#include <stdio.h>
int main(){
    int a = 5, b = 7;
    printf("Average %d", a/2 + b/2);
    return 0;
}
```




```
#include <stdio.h>
int main(){
    int a = 5, b = 7;
    printf("Average %d", (a + b)/2);
    return 0;
}
```



Rounding errors with Integer divisi

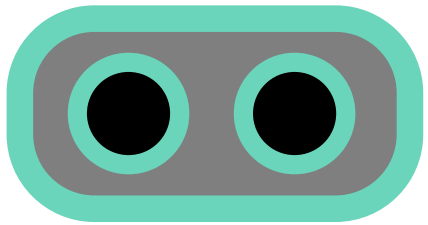


Recall that integer division throws away decimal portion

Errors can accumulate if done too many times

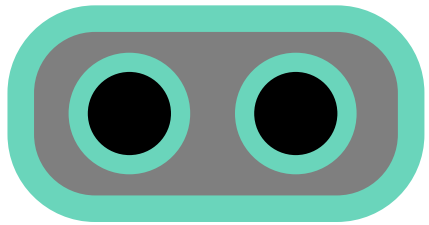
Minimize the number of times you do integer division

5



```
#include <stdio.h>
int main(){
    int a = 5, b = 7;
    printf("Average %d", a/2 + b/2);
    return 0;
}
```

6



```
#include <stdio.h>
int main(){
    int a = 5, b = 7;
    printf("Average %d", (a + b)/2);
    return 0;
}
```

Rounding errors with Integer divisi

Recall that integer division throws away decimal por

Errors can accumulate if done too many times

Minimize the number of times you do integer division

5

```
#include <stdio.h>
int main(){
    int a = 5, b = 7;
    printf("Average %d", a/2 + b/2);
    return 0;
}
```

6

```
#include <stdio.h>
int main(){
    int a = 5, b = 7;
    printf("Average %d", (a + b)/2);
    return 0;
}
```

Rounding er visi

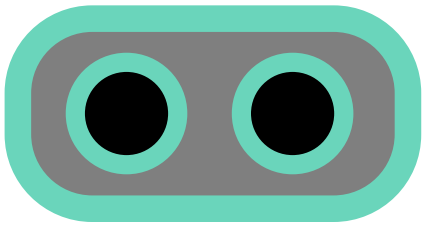
In the first program 5/2 = 2 (loss of 0.5) and 7/2 = 3 (loss of 0.5). Total loss of 1.0

Recall that integer division throws away decimal por

Errors can accumulate if done too many times

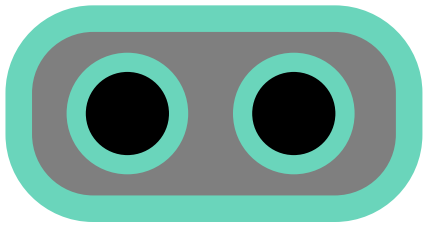
Minimize the number of times you do integer division

5



```
#include <stdio.h>
int main(){
    int a = 5, b = 7;
    printf("Average %d", a/2 + b/2);
    return 0;
}
```

6



```
#include <stdio.h>
int main(){
    int a = 5, b = 7;
    printf("Average %d", (a + b)/2);
    return 0;
}
```

Int type

Can store integers b/w -2,147,483,648 and 2,147,483,647

```
#include <stdio.h>
int main(){
int a;
scanf("%d", &a);
printf("My first int %d", a);
return 0;
}
```


Integer arithmetic applies to long int as well +, -, /, *, %, ()

Have worked with them a lot so far

Long int type

Really long – can store integers between -9,223,372,036,854,775,808 and 9,223,372,036,854,775,807

```
#include <stdio.h>
int main(){
long a; //long int also
scanf("%ld", &a);
printf("My first long int %ld", a);
return 0;
}
```


Integer arithmetic applies to long int as well +, -, /, *, %, ()

Try them out on Prutor

How does long work with int int + long, int * long?

Will see today

Float type

int, long allow us to store, do math formulae with integers

float allows us to store, do math formulae with reals

```
#include <stdio.h>
int main(){
float a;
scanf("%f", &a);
printf("My first real %f", a);
return 0;
}
```

Very large range ± 3.4e+38

Arithmetic operations apply to float as well +, -, /, *, ()

Try them out on Prutor

Double type

भारतीय प्रौद्योगिकी संस्थान कानपुर
INDIAN INSTITUTE OF TECHNOLOGY KANPUR

Double type

Double can also handle real numbers but very large ones

Double type

Double can also handle real numbers but very large ones

Similar relation to float as long has to int

Double type

Double can also handle real numbers but very large ones

Similar relation to float as long has to int

```
#include <stdio.h>
int main(){
double a;
scanf("%f", &a);
printf("My first real %f", a);
return 0;
}
```

Double type

Double can also handle real numbers but very large ones

Similar relation to float as long has to int

```
#include <stdio.h>
int main(){
double a;
scanf("%f", &a);
printf("My first real %f", a);
return 0;
}
```

Double type

Double can also handle real numbers but very large ones

Similar relation to float as long has to int

```
#include <stdio.h>
int main(){
double a;
scanf("%f", &a);
printf("My first real %f", a);
return 0;
}
```


a

%lf works too!

Double type

Double can also handle real numbers but very large ones

Similar relation to float as long has to int

```
#include <stdio.h>
int main(){
double a;
scanf("%f", &a);
printf("My first real %f", a);
return 0;
}
```


%lf works too!

Very large range ± 1.79e+308

Double type

Double can also handle real numbers but very large ones

Similar relation to float as long has to int

```
#include <stdio.h>
int main(){
double a;
scanf("%f", &a);
printf("My first real %f", a);
return 0;
}
```


a

%lf works too!

Very large range ± 1.79e+308

Arithmetic operations apply to double as well +, -, /, *, ()

Double type

Double can also handle real numbers but very large ones

Similar relation to float as long has to int

```
#include <stdio.h>
int main(){
double a;
scanf("%f", &a);
printf("My first real %f", a);
return 0;
}
```


a

%lf works too!

Very large range ± 1.79e+308

Arithmetic operations apply to double as well +, -, /, *, ()

There is something called long double as well ☺

Double type

Double can also handle real numbers but very large ones

Similar relation to float as long has to int

```
#include <stdio.h>
int main(){
double a;
scanf("%f", &a);
printf("My first real %f", a);
return 0;
}
```


a

%lf works too!

Very large range ± 1.79e+308

Arithmetic operations apply to double as well +, -, /, *, ()

There is something called long double as well ☺

Use %Lf to work with long doubles

Double type

Double can also handle real numbers but very large ones

Similar relation to float as long has to int

```
#include <stdio.h>
int main(){
double a;
scanf("%f", &a);
printf("My first real %f", a);
return 0;
}
```


a

%lf works too!

- Very large range ± 1.79e+308
- Arithmetic operations apply to double as well +, -, /, *, ()
- There is something called long double as well ☺
- Use %Lf to work with long doubles
- Try these out on Prutor

Fun with Printing floats and doubles

Fun with Printing floats and doubles

Both %f and %lf work for float and double

Fun with Printing floats and doubles

Both %f and %lf work for float and double

For long double, %Lf needed

Fun with Printing floats and doubles

Both %f and %lf work for float and double

For long double, %Lf needed

Can use %e if want answer in exponential notation

Fun with Printing floats and doubles

Both %f and %lf work for float and double

For long double, %Lf needed

Can use %e if want answer in exponential notation

Fun with Printing floats and doubles

Both %f and %lf work for float and double

For long double, %Lf needed

Can use %e if want answer in exponential notation

```
#include <stdio.h>
```

Fun with Printing floats and doubles

Both %f and %lf work for float and double

For long double, %Lf needed

Can use %e if want answer in exponential notation

```
#include <stdio.h>
int main(){
```

Fun with Printing floats and doubles

Both %f and %lf work for float and double

For long double, %Lf needed

Can use %e if want answer in exponential notation

```
#include <stdio.h>
int main(){
    double a = 123.4567;
```

Fun with Printing floats and doubles

Both %f and %lf work for float and double

For long double, %Lf needed

Can use %e if want answer in exponential notation

```
#include <stdio.h>
int main(){
    double a = 123.4567;
    printf("Value of a = %f", a);
```

Fun with Printing floats and doubles

Both %f and %lf work for float and double

For long double, %Lf needed

Can use %e if want answer in exponential notation

```
#include <stdio.h>
int main(){
    double a = 123.4567;
    printf("Value of a = %f", a);
    return 0;
```

Fun with Printing floats and doubles

Both %f and %lf work for float and double

For long double, %Lf needed

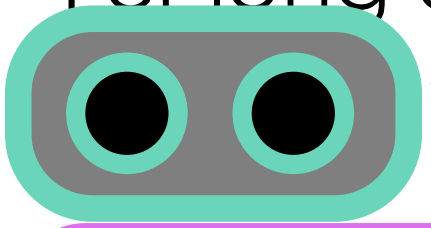
Can use %e if want answer in exponential notation

```
#include <stdio.h>
int main(){
    double a = 123.4567;
    printf("Value of a = %f", a);
    return 0;
}
```

Fun with Printing floats and doubles

Both %f and %lf work for float and double

For long double, %Lf needed



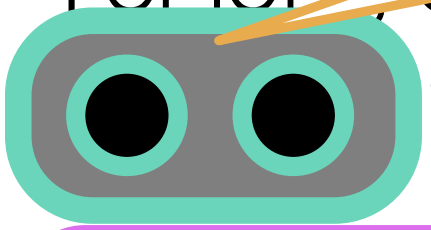
%e if want answer in exponential notation

```
#include <stdio.h>
int main(){
    double a = 123.4567;
    printf("Value of a = %f", a);
    return 0;
}
```

Fun with Printing floats and doubles

Both %f and %lf can be used for float and double

For long double, %Lf needed



%e if want answer in exponential notation

```
#include <stdio.h>
int main(){
    double a = 123.4567;
    printf("Value of a = %f", a);
    return 0;
}
```

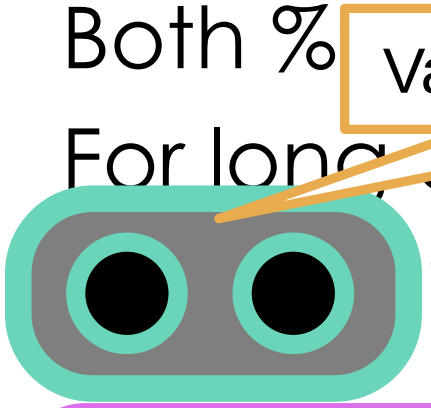
Value of a = 123.456700

Fun with Printing floats and doubles

Both %f and %lf for float and double

For long double, %Lf needed

%e if want answer in exponential notation



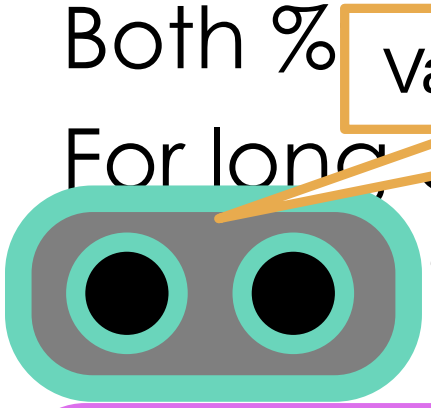

```
#include <stdio.h>
int main(){
    double a = 123.4567;
    printf("Value of a = %f", a);
    return 0;
}
```

Fun with Printing floats and doubles

Both %f and %lf for float and double

For long double, %Lf needed

%e if want answer in exponential notation




```
#include <stdio.h>
int main(){
    double a = 123.4567;
    printf("Value of a = %f", a);
    return 0;
}
```

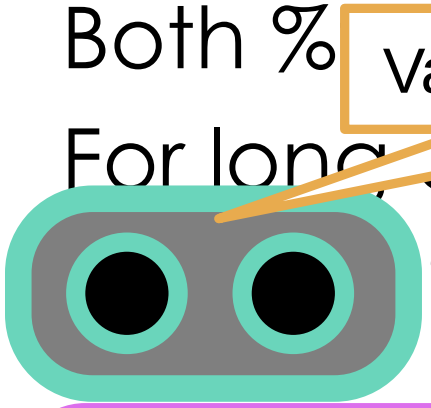
```
#include <stdio.h>
```

Fun with Printing floats and doubles

Both %f and %lf for float and double

For long double, %Lf needed

%e if want answer in exponential notation




```
#include <stdio.h>
int main(){
    double a = 123.4567;
    printf("Value of a = %f", a);
    return 0;
}
```

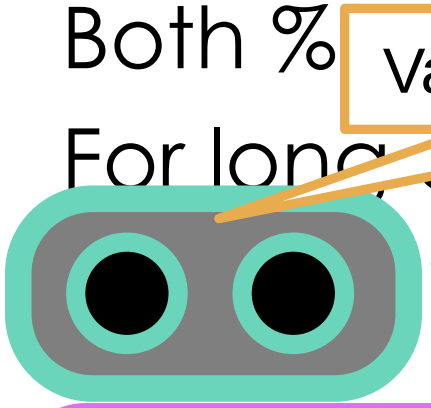
```
#include <stdio.h>
int main(){
```

Fun with Printing floats and doubles

Both %f and %lf for float and double

For long double, %Lf needed

%e if want answer in exponential notation




```
#include <stdio.h>
int main(){
    double a = 123.4567;
    printf("Value of a = %f", a);
    return 0;
}
```

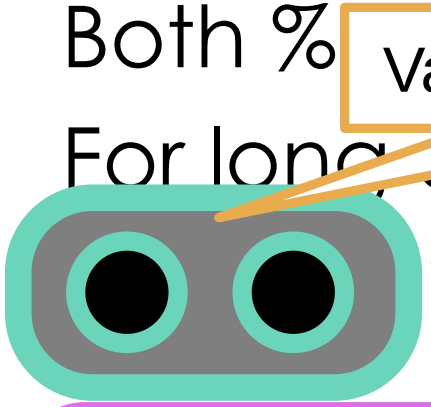
```
#include <stdio.h>
int main(){
    double a = 123.4567;
```

Fun with Printing floats and doubles

Both %f and %lf for float and double

For long double, %Lf needed

%e if want answer in exponential notation




```
#include <stdio.h>
int main(){
    double a = 123.4567;
    printf("Value of a = %f", a);
    return 0;
}
```

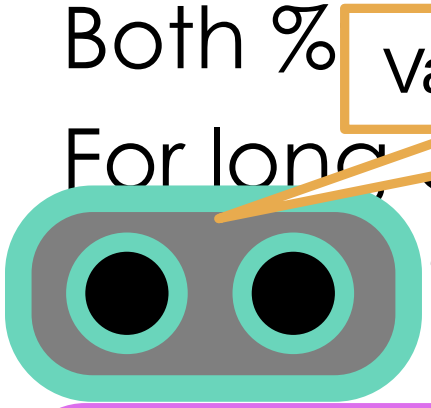
```
#include <stdio.h>
int main(){
    double a = 123.4567;
    printf("Value of a = %e", a);
```

Fun with Printing floats and doubles

Both %f and %lf for float and double

For long double, %Lf needed

%e if want answer in exponential notation




```
#include <stdio.h>
int main(){
    double a = 123.4567;
    printf("Value of a = %f", a);
    return 0;
}
```

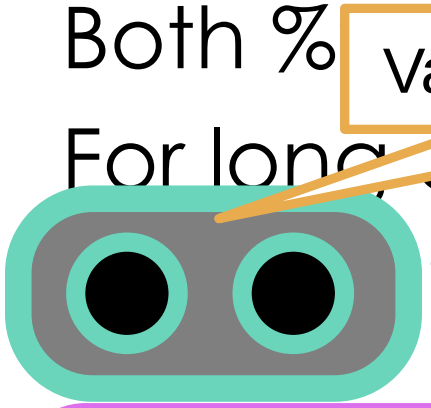
```
#include <stdio.h>
int main(){
    double a = 123.4567;
    printf("Value of a = %e", a);
    return 0;
}
```

Fun with Printing floats and doubles

Both %f and %lf for float and double

For long double, %Lf needed

%e if want answer in exponential notation



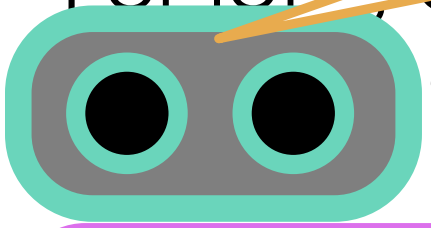

```
#include <stdio.h>
int main(){
    double a = 123.4567;
    printf("Value of a = %f", a);
    return 0;
}
```

```
#include <stdio.h>
int main(){
    double a = 123.4567;
    printf("Value of a = %e", a);
    return 0;
}
```

Fun with Printing floats and doubles

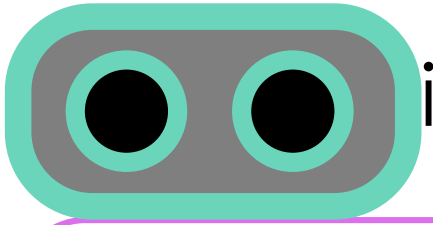
Both %f and %lf for float and double

For long double, %Lf needed



Value of a = 123.456700

%e if want answer in



exponential notation

```
#include <stdio.h>
int main(){
    double a = 123.4567;
    printf("Value of a = %f", a);
    return 0;
}
```

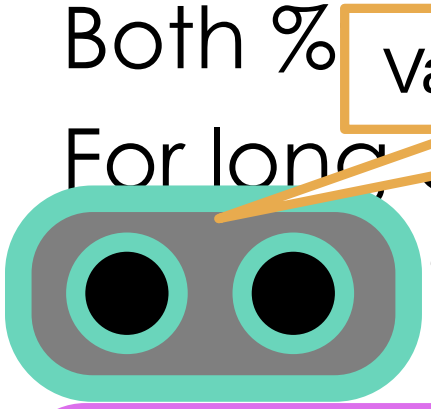
```
#include <stdio.h>
int main(){
    double a = 123.4567;
    printf("Value of a = %e", a);
    return 0;
}
```

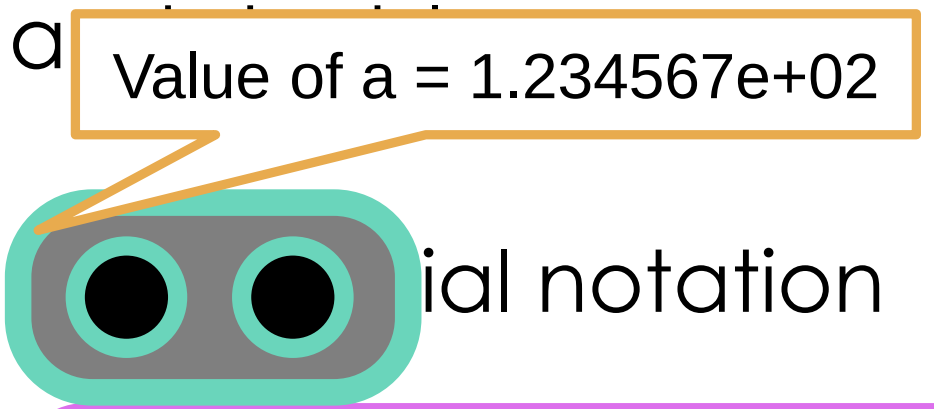
Fun with Printing floats and doubles

Both %f and %e for float and double

For long double, %Lf needed

%e if want answer in exponential notation

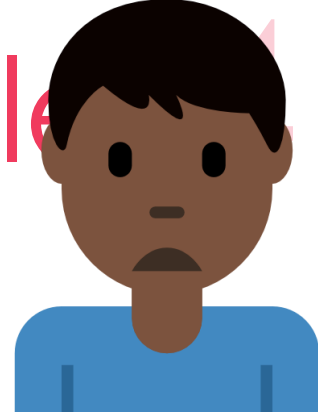





```
#include <stdio.h>
int main(){
    double a = 123.4567;
    printf("Value of a = %f", a);
    return 0;
}
```

```
#include <stdio.h>
int main(){
    double a = 123.4567;
    printf("Value of a = %e", a);
    return 0;
}
```

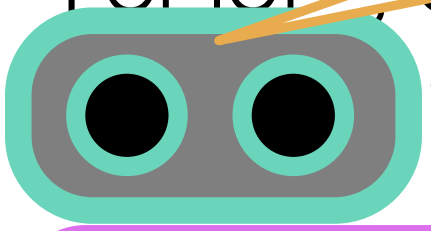
Fun with Printing floats and double



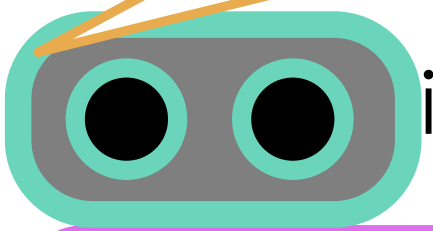
Both %f and %lf for a float a

For long double, %Lf needed

%e if want answer in exponential notation



Value of a = 123.456700



Value of a = 1.234567e+02

```
#include <stdio.h>
int main(){
    double a = 123.4567;
    printf("Value of a = %f", a);
    return 0;
}
```

```
#include <stdio.h>
int main(){
    double a = 123.4567;
    printf("Value of a = %e", a);
    return 0;
}
```

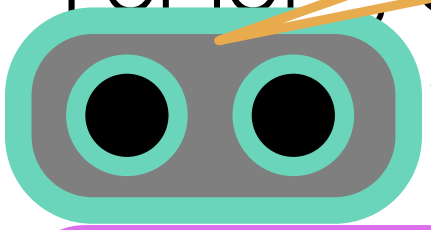
Fun with Printing floating point numbers

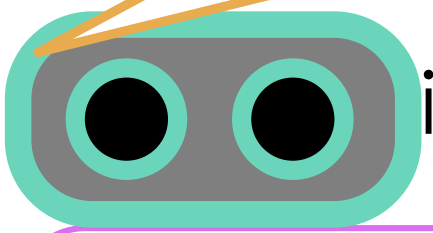
Both %f and %e are for float and double

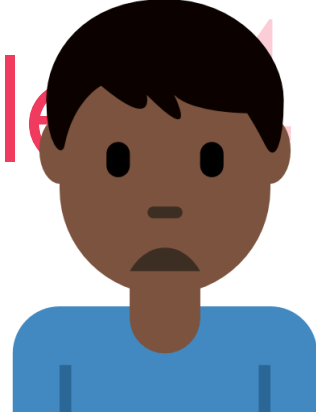
For long double, %Lf needed

%f if want answer in decimal notation

%e if want answer in exponential notation







Too many decimal digits being printed. Can I just print one or two?

Value of a = 123.456700

Value of a = 1.234567e+02

```
#include <stdio.h>
int main(){
    double a = 123.4567;
    printf("Value of a = %f", a);
    return 0;
}
```

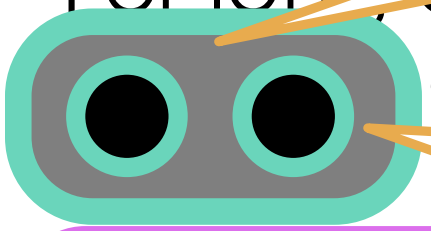
```
#include <stdio.h>
int main(){
    double a = 123.4567;
    printf("Value of a = %e", a);
    return 0;
}
```

Fun with Printing floating point numbers

Both %f and %e are for float and double

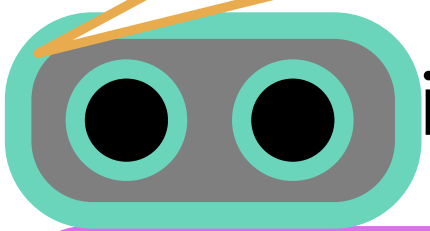
For long double, %Lf needed

%e prints in exponential notation

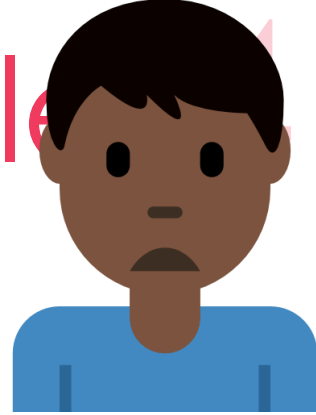


Value of a = 123.456700

Of course. Use %0.2f to print 2 decimal places



Value of a = 1.234567e+02



Too many decimal digits being printed. Can I just print one or two?

```
#include <stdio.h>
int main(){
    double a = 123.4567;
    printf("Value of a = %f", a);
    return 0;
}
```

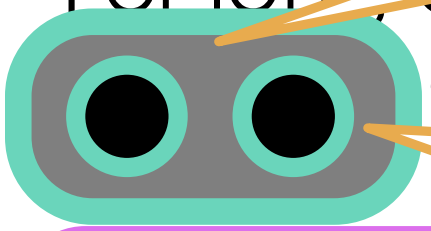
```
#include <stdio.h>
int main(){
    double a = 123.4567;
    printf("Value of a = %e", a);
    return 0;
}
```

Fun with Printing floating point numbers

Both %f and %e are used for float and double

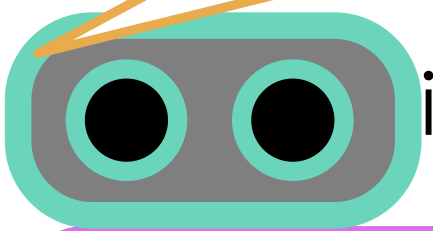
For long double, %Lf needed

%e prints in exponential notation

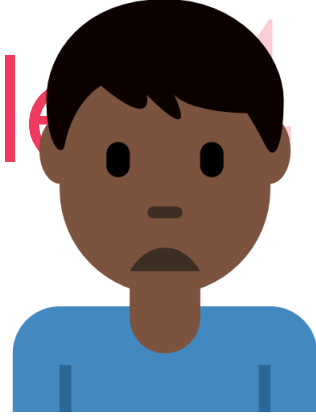


Value of a = 123.456700

Of course. Use %0.2f to print 2 decimal places



Value of a = 1.234567e+02



Too many decimal digits being printed. Can I just print one or two?

```
#include <stdio.h>
int main(){
    double a = 123.4567;
    printf("Value of a = %0.2f", a);
    return 0;
}
```

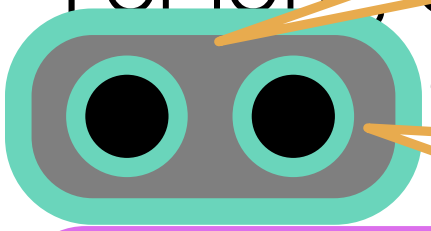
```
#include <stdio.h>
int main(){
    double a = 123.4567;
    printf("Value of a = %e", a);
    return 0;
}
```

Fun with Printing floating point numbers

Both %f and %e are used for float and double

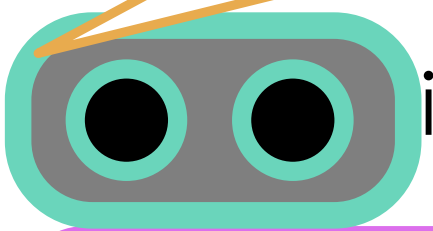
For long double, %Lf needed

%e prints in exponential notation

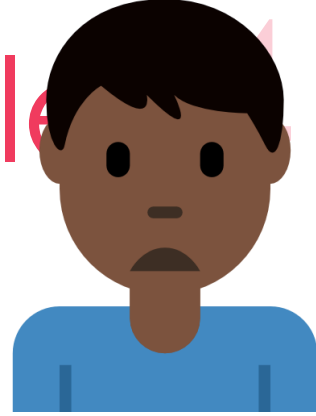


Value of a = 123.46

Of course. Use %0.2f to print 2 decimal places



Value of a = 1.234567e+02



Too many decimal digits being printed. Can I just print one or two?

```
#include <stdio.h>
int main(){
    double a = 123.4567;
    printf("Value of a = %0.2f", a);
    return 0;
}
```

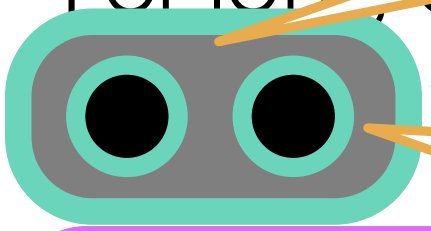
```
#include <stdio.h>
int main(){
    double a = 123.4567;
    printf("Value of a = %e", a);
    return 0;
}
```

Fun with Printing floating point numbers

Both %f and %e print a floating point number

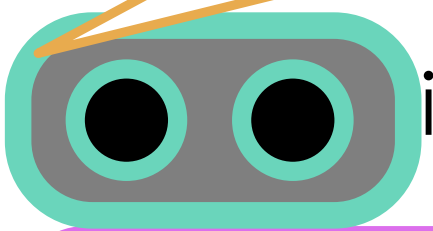
For long double, %Lf needed

%e prints in exponential notation

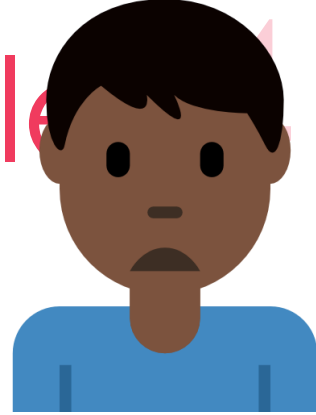


Value of a = 123.46

Of course. Use %0.2f to print 2 decimal places



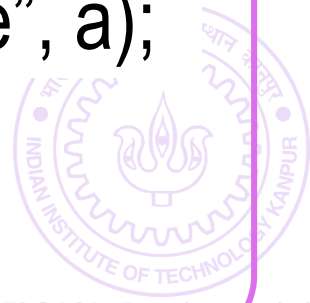
Value of a = 1.234567e+02



Too many decimal digits being printed. Can I just print one or two?

```
#include <stdio.h>
int main(){
    double a = 123.4567;
    printf("Value of a = %0.2f", a);
    return 0;
}
```

```
#include <stdio.h>
int main(){
    double a = 123.4567;
    printf("Value of a = %0.2e", a);
    return 0;
}
```

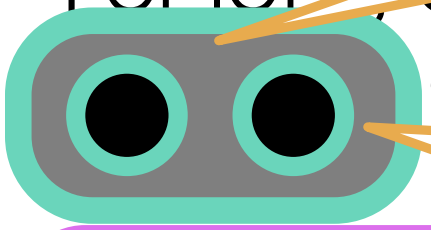


Fun with Printing flo ble

Both % oat a

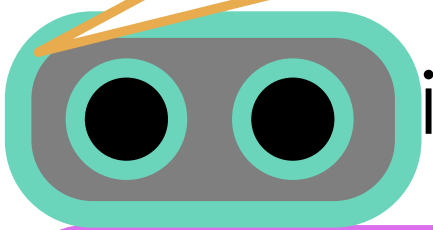
For long double, %Lf needed

%e ial notation

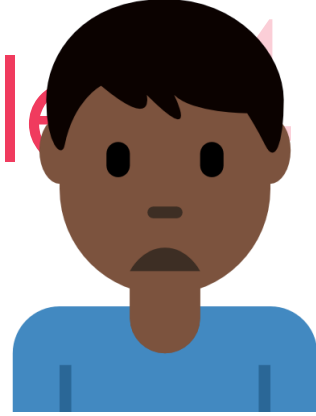


Value of a = 123.46

Of course. Use %0.2f to print 2 decimal places



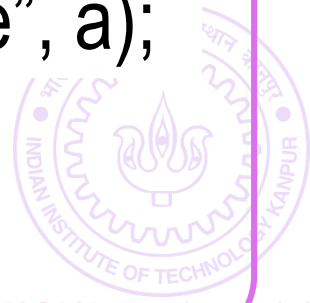
Value of a = 1.23e+02



Too many decimal digits being printed. Can I just print one or two?

```
#include <stdio.h>
int main(){
    double a = 123.4567;
    printf("Value of a = %0.2f", a);
    return 0;
}
```

```
#include <stdio.h>
int main(){
    double a = 123.4567;
    printf("Value of a = %0.2e", a);
    return 0;
}
```

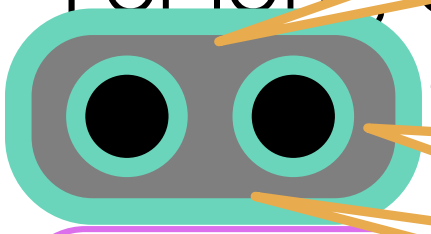


Fun with Printing floating point numbers

Both %f and %e are used for float and double

For long double, %Lf needed

%e prints in exponential notation



Value of a = 123.46

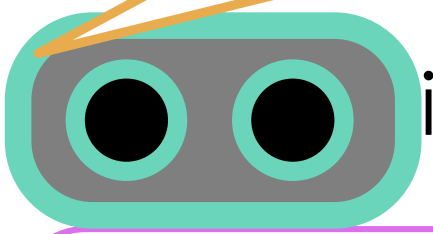
Of course. Use %0.2f to print 2 decimal places

Be careful, I am rounding while giving answer correct to 2 decimal places

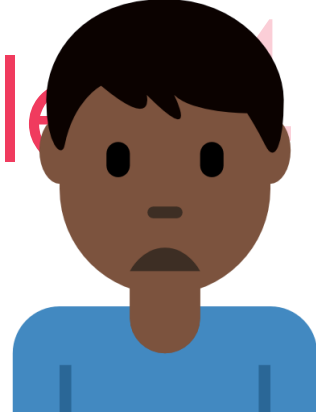
123.4567 → 123.46

1.234567 → 1.23

```
#include <stdio.h>
int main(){
    double a = 123.4567;
    printf("Value of a = %0.2f", a);
    return 0;
}
```



Too many decimal digits being printed. Can I just print one or two?



Value of a = 1.23e+02

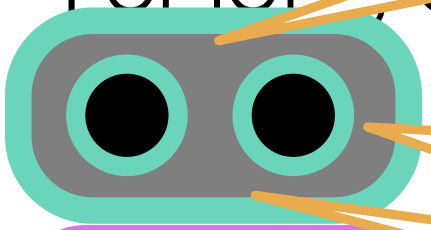
```
#include <stdio.h>
int main(){
    double a = 123.4567;
    printf("Value of a = %0.2e", a);
    return 0;
}
```

Fun with Printing flo ble

Both % oat a

For long double, %Lf needed

%e



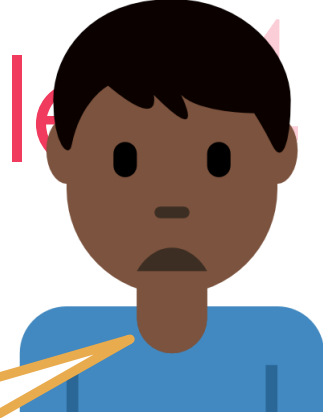
Value of a = 123.46

Of course. Use %0.2f to print 2 decimal places

Be careful, I am rounding while giving answer correct to 2 decimal places
123.4567 → 123.46
1.234567 → 1.23

```
#include <
int main()
    double
    printf("Value of a = %0.2f", a);
    return 0;
}
```

Too many decimal digits being printed. Can I just print one or two?



Value of a = 1.23e+02

Oh right. The usual rules of rounding apply here too. 1.5644 will become 1.56 if rounded to 2 places but 1.5654 will become 1.57

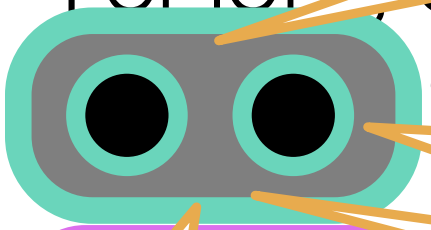
```
#
a
uble a = 123.4567;
    printf("Value of a = %0.2e", a);
    return 0;
}
```

Fun with Printing floating point numbers

Both %f and %lf are used for float and double

For long double, %Lf needed

%e



Value of a = 123.46

Of course. Use %0.2f to print 2 decimal places

Correct!

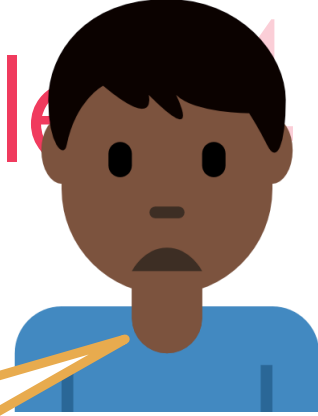
Be careful, I am rounding while giving answer correct to 2 decimal places

123.4567 → 123.46

1.234567 → 1.23

```
#include <stdio.h>
int main()
{
    double a = 123.4567;
    printf("Value of a = %0.2f", a);
    return 0;
}
```

Too many decimal digits being printed. Can I just print one or two?



Value of a = 1.23e+02

Oh right. The usual rules of rounding apply here too. 1.5644 will become 1.56 if rounded to 2 places but 1.5654 will become 1.57

```
double a = 123.4567;
    printf("Value of a = %0.2e", a);
    return 0;
}
```

Mixed-type formulae

Mixed-type formulae

int a; long b; float c; double d;

Mixed-type formulae

int a; long b; float c; double d;

Recall the care we took when working with long and int

Mixed-type formulae

int a; long b; float c; double d;

Recall the care we took when working with long and int

- We saw that b = a + a; could cause errors

Mixed-type formulae

int a; long b; float c; double d;

Recall the care we took when working with long and int

- We saw that b = a + a; could cause errors
- Mr C will first try to store a + a into a temporary integer variable

Mixed-type formulae

int a; long b; float c; double d;

Recall the care we took when working with long and int

- We saw that b = a + a; could cause errors
- Mr C will first try to store a + a into a temporary integer variable
- To force him to store a + a into a long variable, do typecasting

Mixed-type formulae

int a; long b; float c; double d;

Recall the care we took when working with long and int

- We saw that b = a + a; could cause errors
- Mr C will first try to store a + a into a temporary integer variable
- To force him to store a + a into a long variable, do typecasting
- b = (long) a + (long) a;

Mixed-type formulae

int a; long b; float c; double d;

Recall the care we took when working with long and int

- We saw that b = a + a; could cause errors
- Mr C will first try to store a + a into a temporary integer variable
- To force him to store a + a into a long variable, do typecasting
- b = (long) a + (long) a;
- b = a + (long) a;

Mixed-type formulae

int a; long b; float c; double d;

Recall the care we took when working with long and int

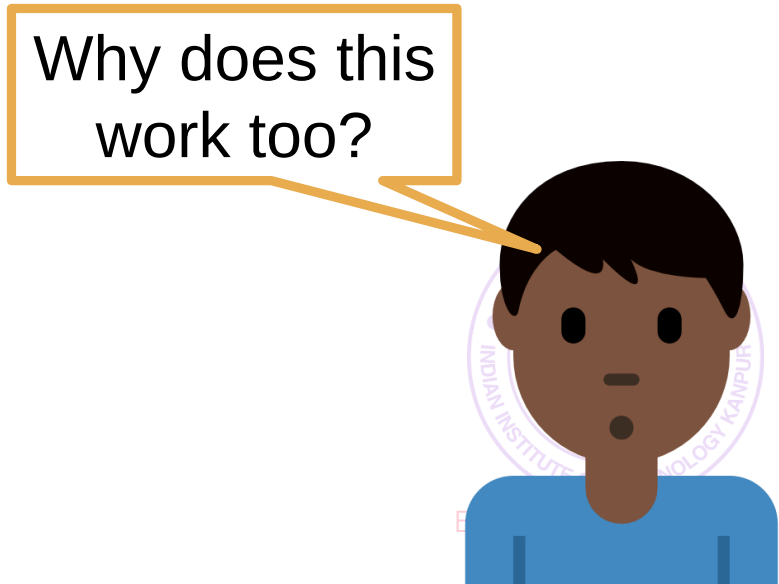
- We saw that b = a + a; could cause errors
- Mr C will first try to store a + a into a temporary integer variable
- To force him to store a + a into a long variable, do typecasting
- b = (long) a + (long) a;
- b = a + (long) a;

Mixed-type formulae

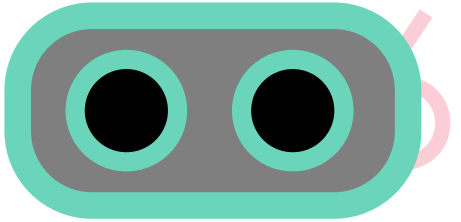
int a; long b; float c; double d;

Recall the care we took when working with long and int

- We saw that b = a + a; could cause errors
- Mr C will first try to store a + a into a temporary integer variable
- To force him to store a + a into a long variable, do typecasting
- b = (long) a + (long) a;
- b = a + (long) a;



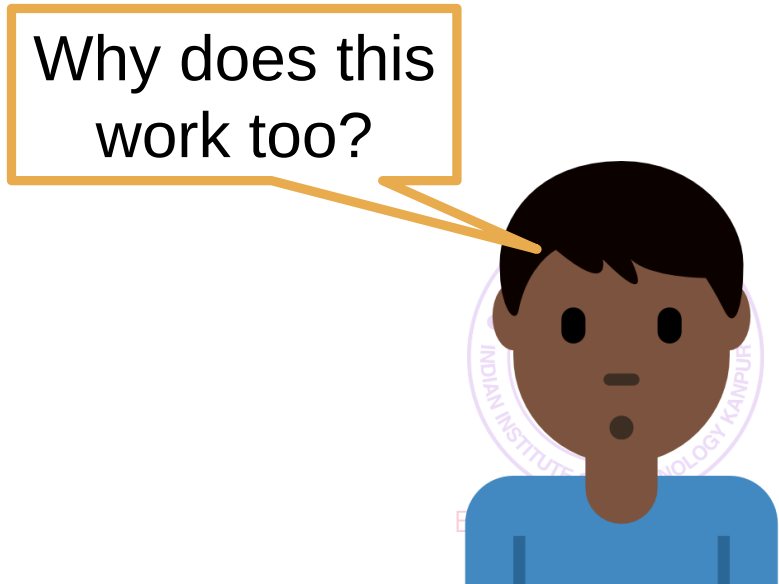
Mixed-type formulae



int a; long b; float c; double d;

Recall the care we took when working with long and int

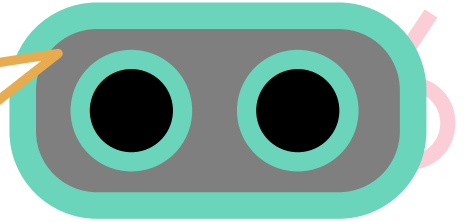
- We saw that b = a + a; could cause errors
- Mr C will first try to store a + a into a temporary integer variable
- To force him to store a + a into a long variable, do typecasting
- b = (long) a + (long) a;
- b = a + (long) a;



Mixed-type formul

int a; long b; float c; double

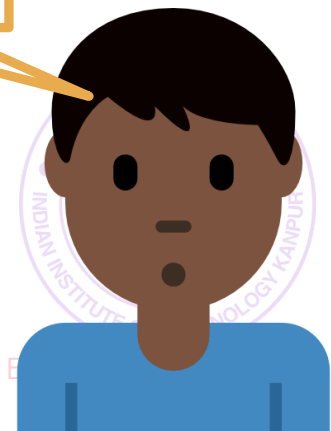
Because I see this as a mixed type formula and convert everything to long to avoid mistakes



Recall the care we took when working with long and int

- We saw that b = a + a; could cause errors
- Mr C will first try to store a + a into a temporary integer variable
- To force him to store a + a into a long variable, do typecasting
- b = (long) a + (long) a;
- b = a + (long) a;

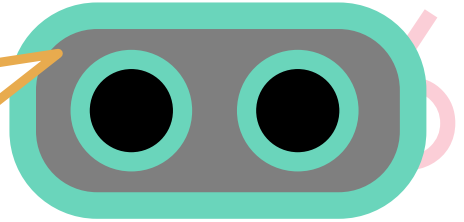
Why does this work too?



Mixed-type formul

int a; long b; float c; double

Because I see this as a mixed type formula and convert everything to long to avoid mistakes

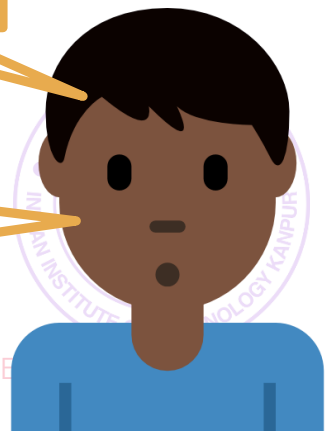


Recall the care we took when working with long and int

- We saw that b = a + a; could cause errors
- Mr C will first try to store a + a into a temporary integer variable
- To force him to store a + a into a long variable, do typecasting
- b = (long) a + (long) a;
- b = a + (long) a;

Why does this work too?

Hmm.. So just because I typecasted one of the variables in the formula, you typecasted the rest for free



Mixed-type formul

int a; long b; float c; double

Recall the care we took when working with lon

We saw that b = a + a; could cause errors

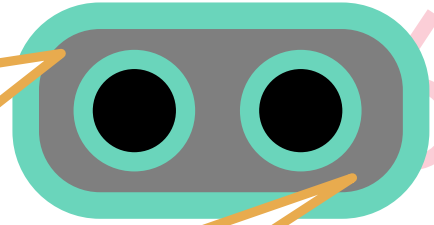
Mr C will first try to store a + a into a temporary integer variable

To force him to store a + a into a long variable, do typecasting

b = (long) a + (long) a;

b = a + (long) a;

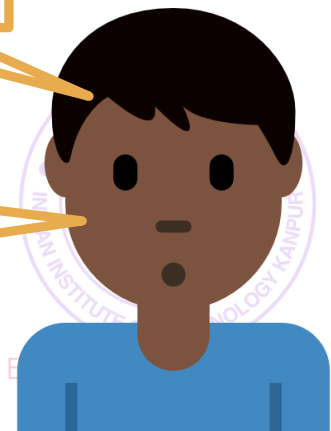
Because I see this as a mixed type formula and convert everything to long to avoid mistakes



See, I am so helpful!

Why does this work too?

Hmm.. So just because I typecasted one of the variables in the formula, you typecasted the rest for free

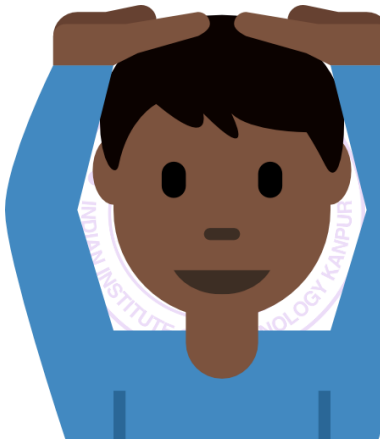


Mixed-type formul

int a; long b; float c; double

Recall the care we took when working with lon

- We saw that b = a + a; could cause errors
- Mr C will first try to store a + a into a temporary integer variable
- To force him to store a + a into a long variable, do typecasting
- b = (long) a + (long) a;
- b = a + (long) a;



Mixed-type formul

int a; long b; float c; double

Recall the care we took when working with lon

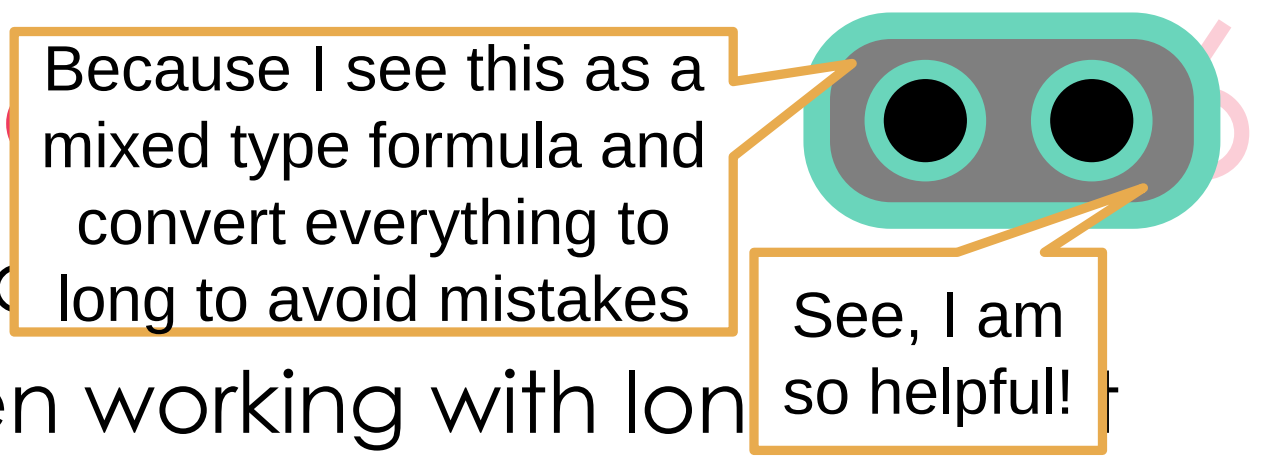
We saw that b = a + a; could cause errors

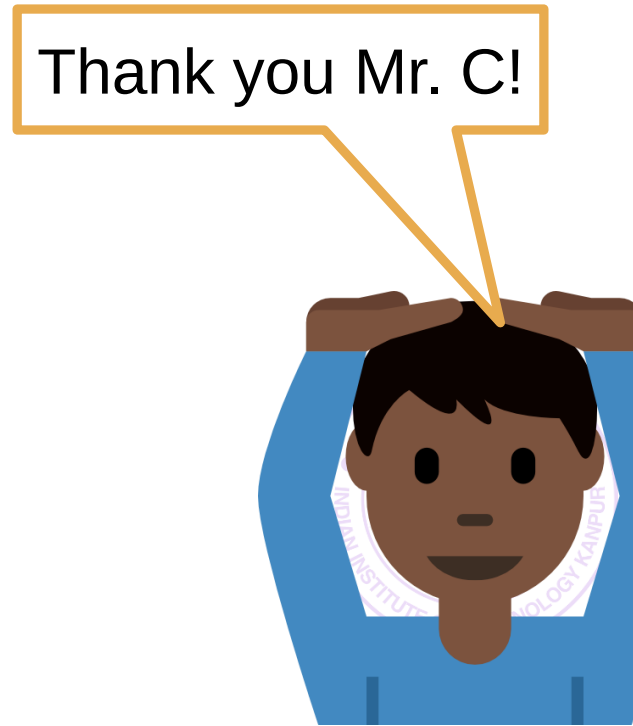
Mr C will first try to store a + a into a temporary integer variable

To force him to store a + a into a long variable, do typecasting

b = (long) a + (long) a;

b = a + (long) a;





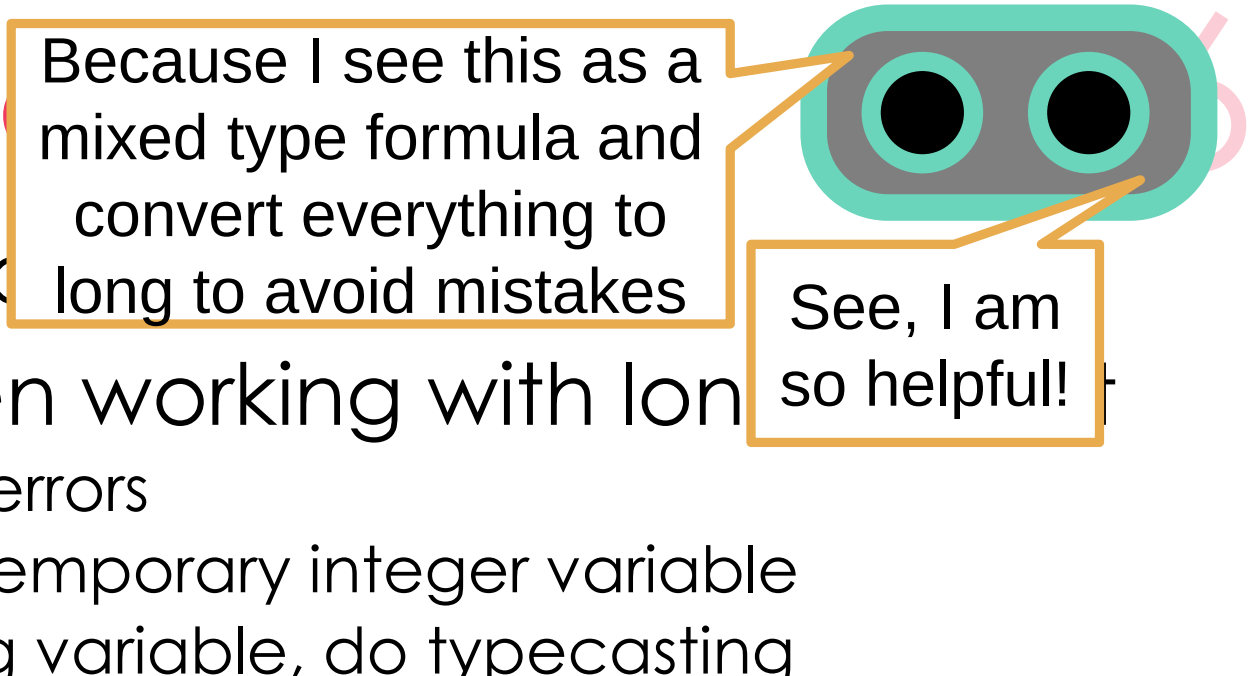
Mixed-type formul

int a; long b; float c; double

Recall the care we took when working with lon

- We saw that b = a + a; could cause errors
- Mr C will first try to store a + a into a temporary integer variable
- To force him to store a + a into a long variable, do typecasting
- b = (long) a + (long) a;
- b = a + (long) a;

Similar care must be taken with int + float, int + double



Mixed-type formul

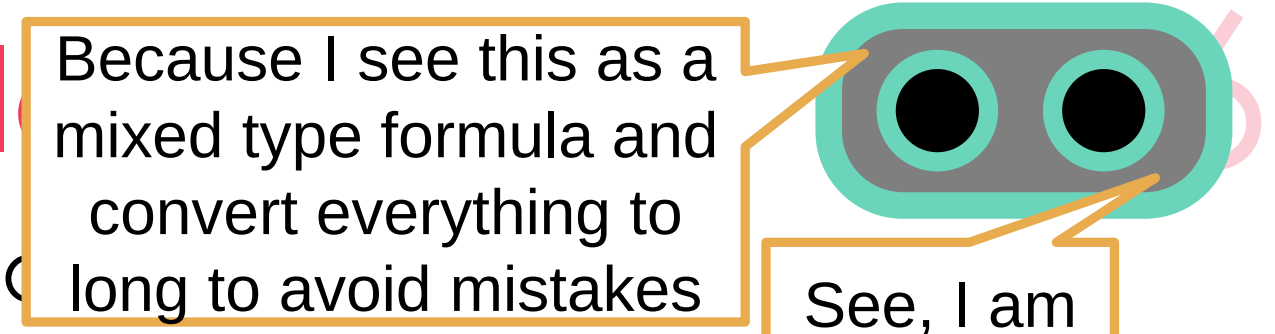
int a; long b; float c; double

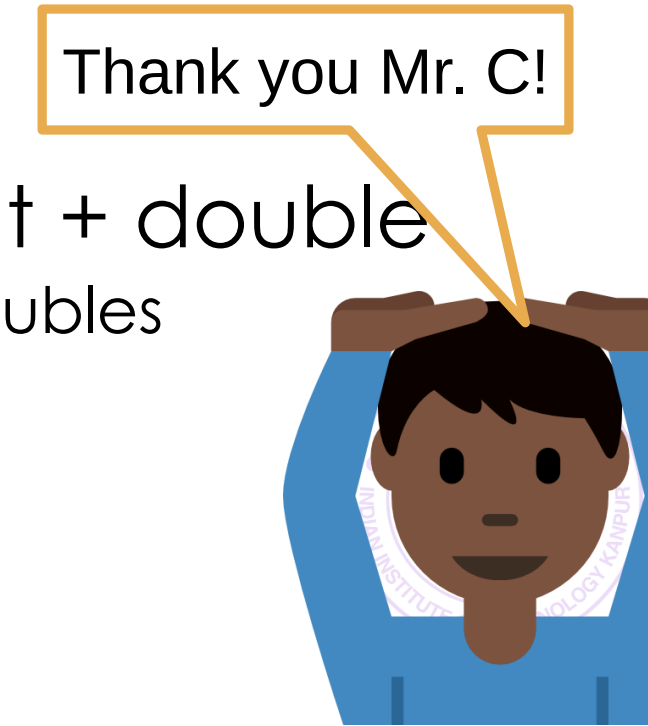
Recall the care we took when working with lon

- We saw that b = a + a; could cause errors
- Mr C will first try to store a + a into a temporary integer variable
- To force him to store a + a into a long variable, do typecasting
- b = (long) a + (long) a;
- b = a + (long) a;

Similar care must be taken with int + float, int + double

- To avoid integer division, typecast integers to floats or doubles





Mixed-type formul

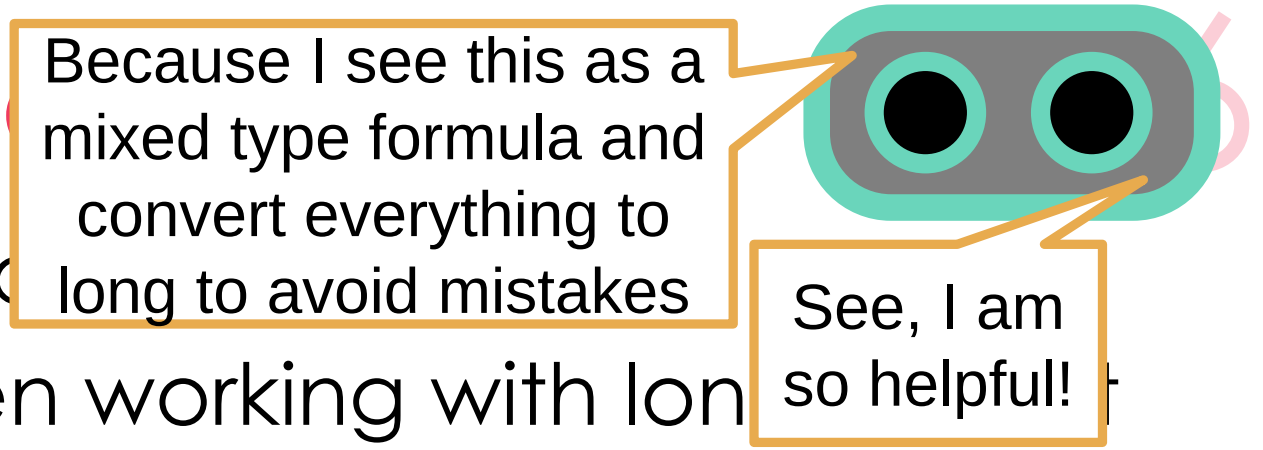
int a; long b; float c; double

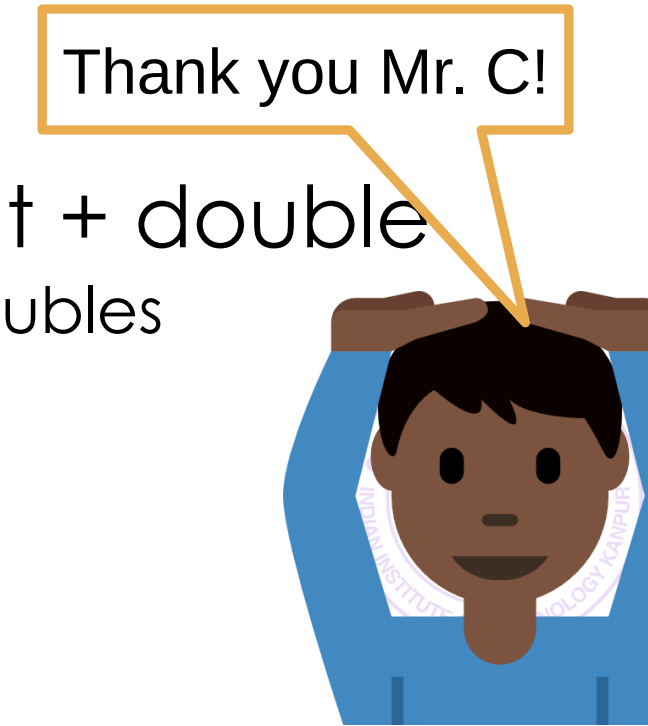
Recall the care we took when working with lon

- We saw that b = a + a; could cause errors
- Mr C will first try to store a + a into a temporary integer variable
- To force him to store a + a into a long variable, do typecasting
- b = (long) a + (long) a;
- b = a + (long) a;

Similar care must be taken with int + float, int + double

- To avoid integer division, typecast integers to floats or doubles
- int a = 2;





Mixed-type formul

int a; long b; float c; double

Recall the care we took when working with lon

We saw that b = a + a; could cause errors

Mr C will first try to store a + a into a temporary integer variable

To force him to store a + a into a long variable, do typecasting

b = (long) a + (long) a;

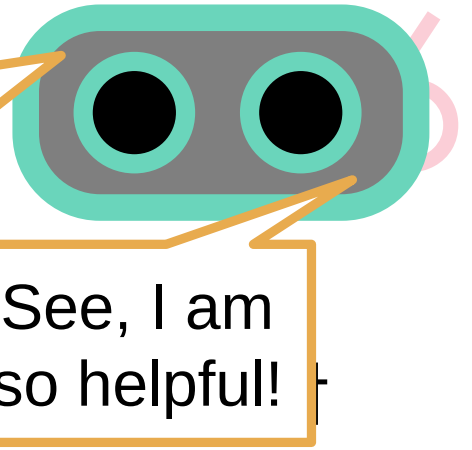
b = a + (long) a;

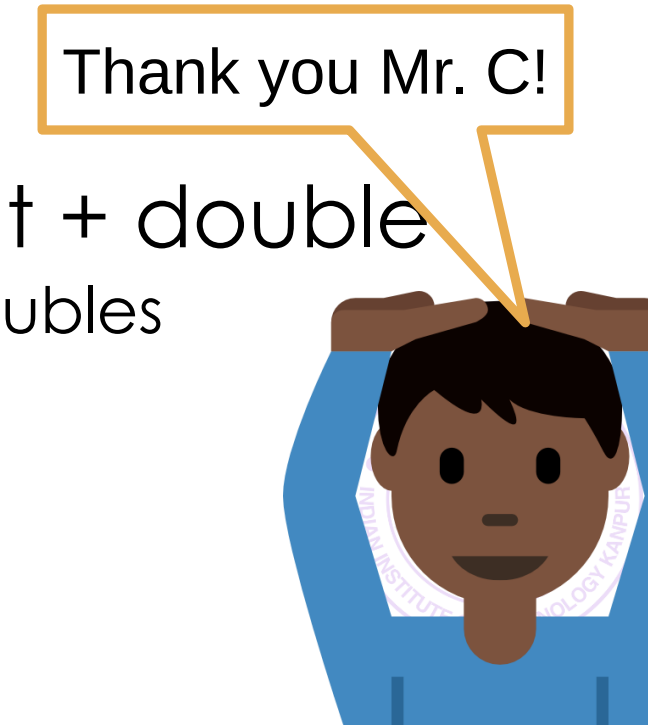
Similar care must be taken with int + float, int + double

To avoid integer division, typecast integers to floats or doubles

int a = 2;

c = a / (a+1); → 0





Mixed-type formul

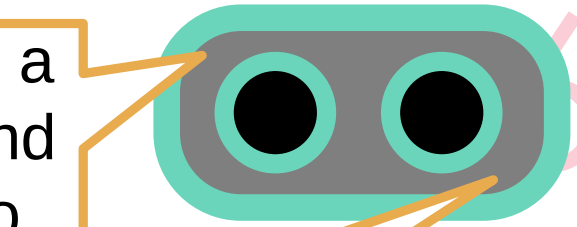
int a; long b; float c; double

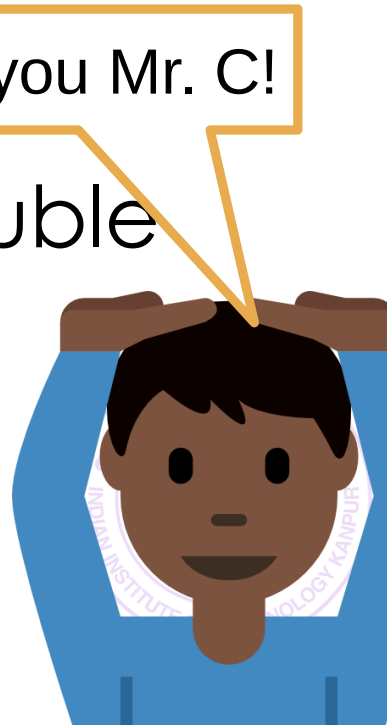
Recall the care we took when working with lon

- We saw that b = a + a; could cause errors
- Mr C will first try to store a + a into a temporary integer variable
- To force him to store a + a into a long variable, do typecasting
- b = (long) a + (long) a;
- b = a + (long) a;

Similar care must be taken with int + float, int + double

- To avoid integer division, typecast integers to floats or doubles
- int a = 2;
- c = a / (a+1); → 0
- c = (float) a / (a + 1); → 0.666666666





Mixed-type formul

int a; long b; float c; double

Recall the care we took when working with lon

We saw that b = a + a; could cause errors

Mr C will first try to store a + a into a temporary integer variable

To force him to store a + a into a long variable, do typecasting

b = (long) a + (long) a;

b = a + (long) a;

Similar care must be taken with int + float, int + double

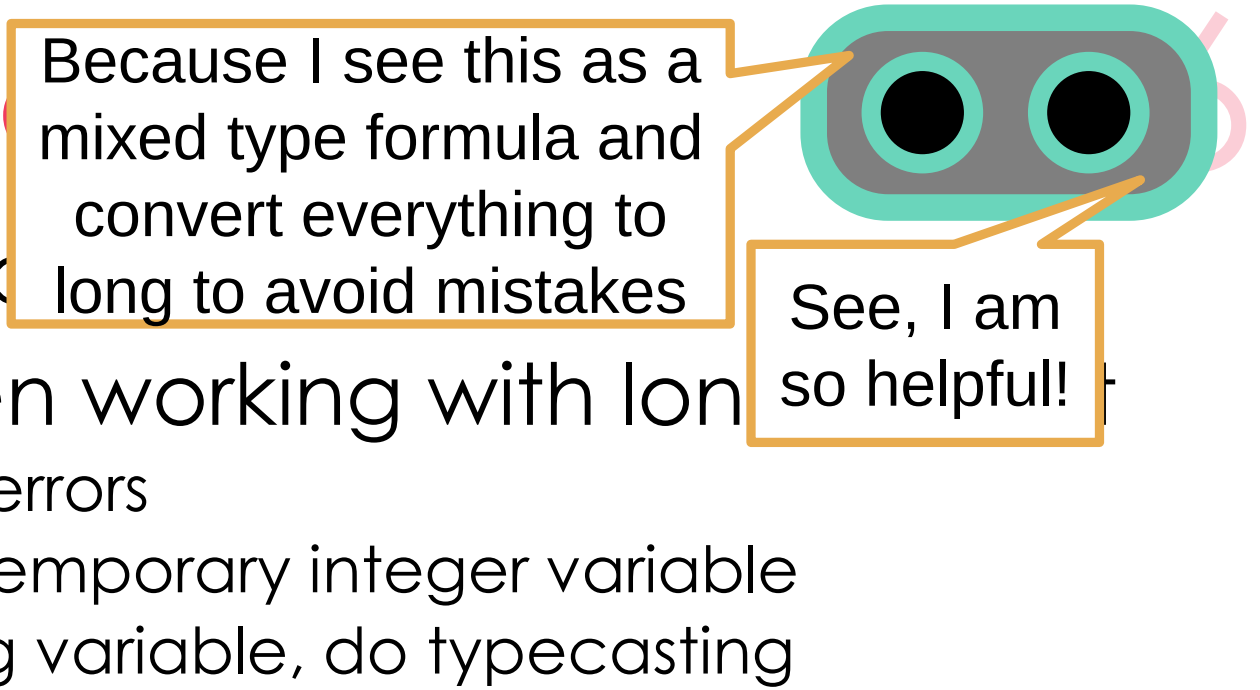
To avoid integer division, typecast integers to floats or doubles

int a = 2;

c = a / (a+1); → 0

c = (float) a / (a + 1); → 0.666666666

c = a / (a + 1.0); → 0.666666666



Mixed-type formul

int a; long b; float c; double

Recall the care we took when working with lon

We saw that b = a + a; could cause errors

Mr C will first try to store a + a into a temporary integer variable

To force him to store a + a into a long variable, do typecasting

b = (long) a + (long) a;

b = a + (long) a;

Similar care must be taken with int + float, int + double

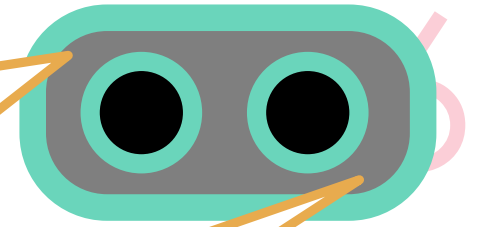
To avoid integer division, typecast integers to floats or doubles

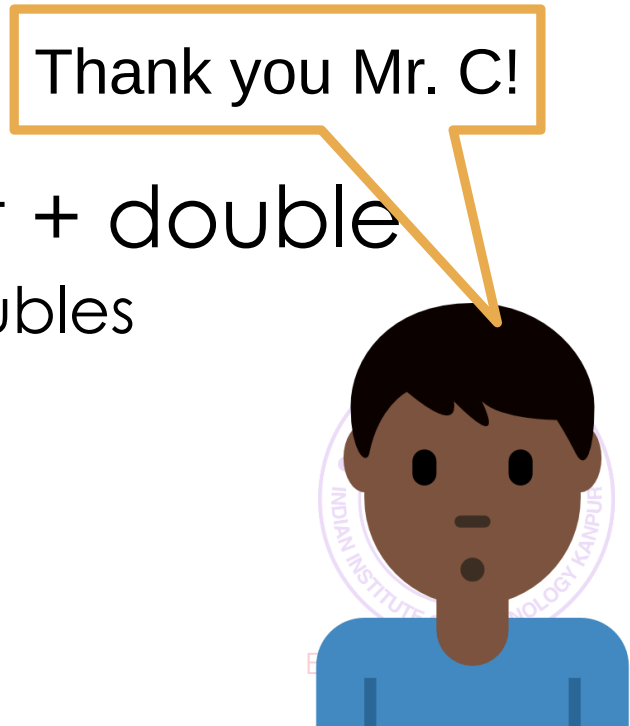
int a = 2;

c = a / (a+1); → 0

c = (float) a / (a + 1); → 0.666666666

c = a / (a + 1.0); → 0.666666666





Mixed-type formul

int a; long b; float c; double

Recall the care we took when working with lon

We saw that b = a + a; could cause errors

Mr C will first try to store a + a into a temporary integer variable

To force him to store a + a into a long variable, do typecasting

b = (long) a + (long) a;

b = a + (long) a;

Similar care must be taken with int + float ble

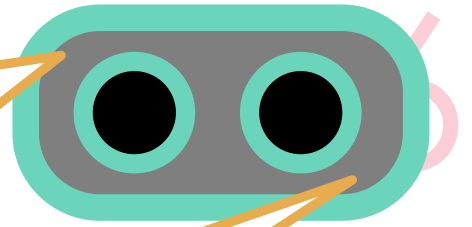
To avoid integer division, typecast integers to floats or

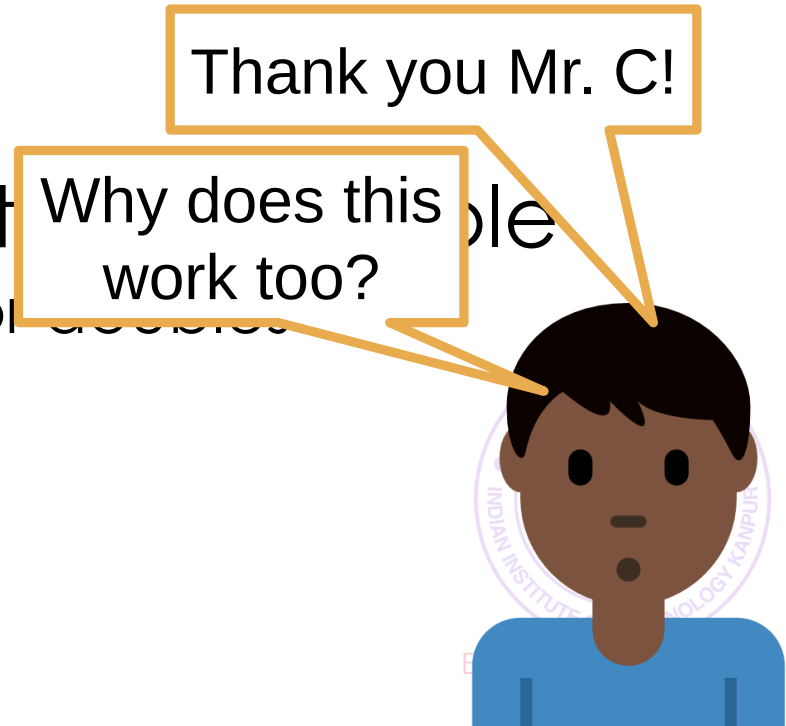
int a = 2;

c = a / (a+1); → 0

c = (float) a / (a + 1); → 0.666666666

c = a / (a + 1.0); → 0.666666666





Mixed-type formul

int a; long b; float c; double

Recall the care we took when working with lon

We saw that b = a + a; could cau

Mr C will first try to store a + a into able

To force him to store a + a into a ting

b = (long) a + (long) a;

b = a + (long) a;

Similar care must be taken with int + float ble

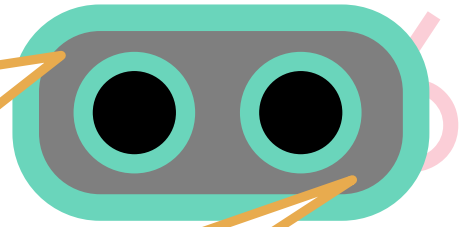
To avoid integer division, typecast integers to floats or

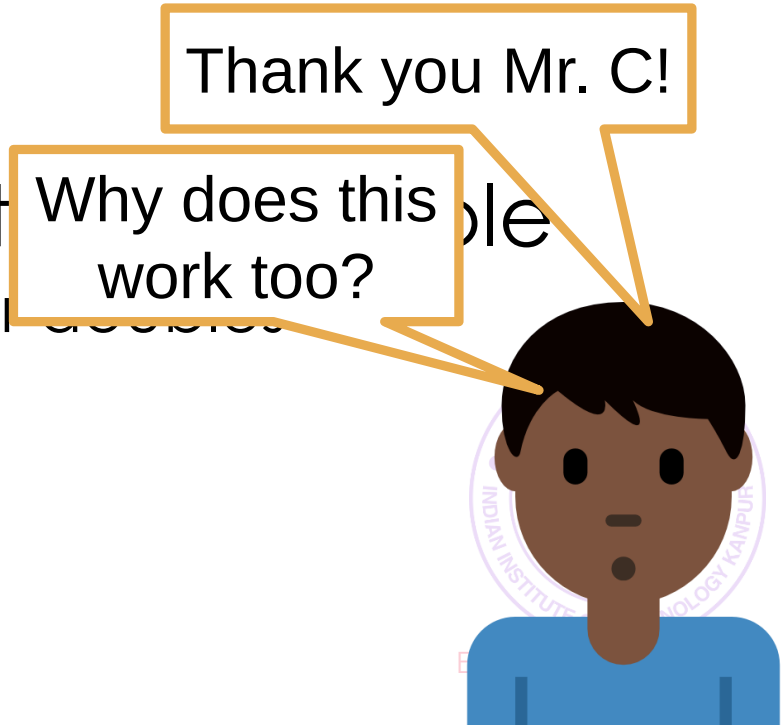
int a = 2;

c = a / (a+1); → 0

c = (float) a / (a + 1); → 0.666666666

c = a / (a + 1.0); → 0.666666666





Mixed-type formul

int a; long b; float c; double

Recall the care we took when working with lon

We saw that b = a + a; could cau

Mr C will first try to store a + a into

To force him to store a + a into a

b = (long) a + (long) a;

b = a + (long) a;

Similar care must be taken with int + float

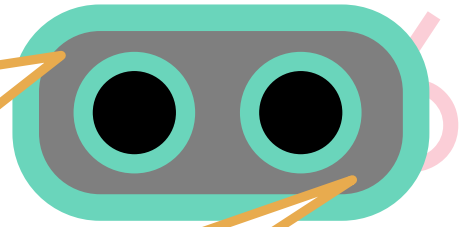
To avoid integer division, typecast integers to floats or

int a = 2;

c = a / (a+1); → 0

c = (float) a / (a + 1); → 0.666666666

c = a / (a + 1.0); → 0.666666666



Reverse typecasting problems

Reverse typecasting problems

float can store bigger numbers yet sometimes when int is typecast to float, there are errors

Reverse typecasting problems

float can store bigger numbers yet sometimes when int is typecast to float, there are errors

Details of this error will be covered later in the course

Reverse typecasting problems

float can store bigger numbers yet sometimes when int is typecast to float, there are errors

Details of this error will be covered later in the course

Mr. C has a lot of things to store for float variables

Reverse typecasting problems

float can store bigger numbers yet sometimes when int is typecast to float, there are errors

Details of this error will be covered later in the course

Mr. C has a lot of things to store for float variables

- What are the digits in the number

Reverse typecasting problems

float can store bigger numbers yet sometimes when int is typecast to float, there are errors

Details of this error will be covered later in the course

Mr. C has a lot of things to store for float variables

- What are the digits in the number
- Where is the decimal point placed

Reverse typecasting problems

float can store bigger numbers yet sometimes when int is typecast to float, there are errors

Details of this error will be covered later in the course

Mr. C has a lot of things to store for float variables

- What are the digits in the number
- Where is the decimal point placed

If your integer is too long, Mr C will approximate it when storing it as a float even though the value of the number is well within Mr C's range of numbers

math.h

भारतीय प्रौद्योगिकी संस्थान कानपुर
INDIAN INSTITUTE OF TECHNOLOGY KANPUR

math.h

A really nice library of lots of mathematical functions

math.h

A really nice library of lots of mathematical functions

abs(x): absolute value of integer x

math.h

A really nice library of lots of mathematical functions

abs(x): absolute value of integer x

fabs(x): absolute value of x if x is float or double

math.h

A really nice library of lots of mathematical functions

abs(x): absolute value of integer x

fabs(x): absolute value of x if x is float or double

ceil(x): ceiling function (smallest integer greater than x)

math.h

A really nice library of lots of mathematical functions

abs(x): absolute value of integer x

fabs(x): absolute value of x if x is float or double

ceil(x): ceiling function (smallest integer greater than x)

floor(x): floor function (largest integer smaller than x)

math.h

A really nice library of lots of mathematical functions

abs(x): absolute value of integer x

fabs(x): absolute value of x if x is float or double

ceil(x): ceiling function (smallest integer greater than x)

floor(x): floor function (largest integer smaller than x)

log(x): logarithm of x (do not give negative value of x)

math.h

A really nice library of lots of mathematical functions

abs(x): absolute value of integer x

fabs(x): absolute value of x if x is float or double

ceil(x): ceiling function (smallest integer greater than x)

floor(x): floor function (largest integer smaller than x)

log(x): logarithm of x (do not give negative value of x)

pow(x,y): x to the power y (both doubles – typecast if int)

math.h

A really nice library of lots of mathematical functions

abs(x): absolute value of integer x

fabs(x): absolute value of x if x is float or double

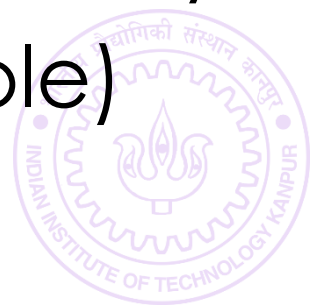
ceil(x): ceiling function (smallest integer greater than x)

floor(x): floor function (largest integer smaller than x)

log(x): logarithm of x (do not give negative value of x)

pow(x,y): x to the power y (both doubles – typecast if int)

sqrt(x): square root of double x (typecast if not double)



math.h

A really nice library of lots of mathematical functions

abs(x): absolute value of integer x

fabs(x): absolute value of x if x is float or double

ceil(x): ceiling function (smallest integer greater than x)

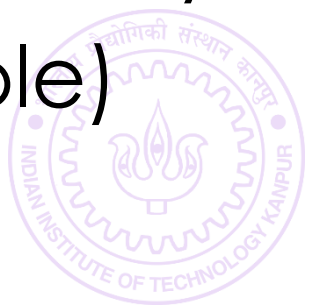
floor(x): floor function (largest integer smaller than x)

log(x): logarithm of x (do not give negative value of x)

pow(x,y): x to the power y (both doubles – typecast if int)

sqrt(x): square root of double x (typecast if not double)

cos(x), sin(x), tan(x) etc are also present – explore!



Operators

भारतीय प्रौद्योगिकी संस्थान कानपुर
INDIAN INSTITUTE OF TECHNOLOGY KANPUR

Operators

We have seen quite a few math operators till now

Operators

We have seen quite a few math operators till now

+, -, *, /, %

Operators

We have seen quite a few math operators till now

+, -, *, /, %

All take two numbers and give one number as answer

Operators

We have seen quite a few math operators till now

+, -, *, /, %

All take two numbers and give one number as answer

Called *binary operators* for this reason. Binary = two

Operators

We have seen quite a few math operators till now

+, -, *, /, %

All take two numbers and give one number as answer

Called *binary operators* for this reason. Binary = two

Many *unary operators* also exist

Operators

We have seen quite a few math operators till now

+, -, *, /, %

All take two numbers and give one number as answer

Called *binary operators* for this reason. Binary = two

Many *unary operators* also exist

Have seen two till now:

Operators

We have seen quite a few math operators till now

+, -, *, /, %

All take two numbers and give one number as answer

Called *binary operators* for this reason. Binary = two

Many *unary operators* also exist

Have seen two till now:

Unary negation int a = -21; b = -a;

Operators

We have seen quite a few math operators till now

+, -, *, /, %

All take two numbers and give one number as answer

Called *binary operators* for this reason. Binary = two

Many *unary operators* also exist

Have seen two till now:

Unary negation int a = -21; b = -a;

Typecasting c = (int) a;

Operators

We have seen quite a few math operators till now

+, -, *, /, %

All take two numbers and give one number as answer

Called *binary operators* for this reason. Binary = two

Many *unary operators* also exist

Have seen two till now:

Unary negation int a = -21; b = -a;

Typecasting c = (int) a;

Will see several more operators in the next class

Operators

We have seen quite a few math operators till now

+, -, *, /, %

All take two numbers and give one number as answer

Called *binary operators* for this reason. Binary = two

Many *unary operators* also exist

Have seen two till now:

Unary negation int a = -21; b = -a;

Typecasting c = (int) a;

Will see several more operators in the next class

Also will start expanding our programming power

Operators

We have seen quite a few math operators till now

+, -, *, /, %

All take two numbers and give one number as answer

Called *binary operators* for this reason. Binary = two

Many *unary operators* also exist

Have seen two till now:

Unary negation int a = -21; b = -a;

Typecasting c = (int) a;

Will see several more operators in the next class

Also will start expanding our programming power

Conditional statements and relational operators