This is Section.

Week 1: C

Gabe LeBlanc

This is Real. This is Me. 🎵

gleblanc@college.harvard.edu

heads@cs50.harvard.edu

github.com/gblanc25/cs50

Office Hours: Saturdays at 1pm in Cabot Dhall

harvard.cs50.me/hours

More help!

asynchronous questions

sundays 3-5pm

immediate response

Grading: Design

5 4 3 2 1

Grading: Design

5 **4** **3** 2 1

Grading: Correctness

check50

Grading: Style

style50

what ultimately matters in this course is not so much where you end up relative to your classmates but where you end up relative to yourself when you began

Honesty

"... be reasonable…"

When in doubt, ask.

*It is **always** better to submit an incomplete problem set than to submit one completed with the help of an online solution or walkthrough.*

The regret clause is available to you within 72 hours.

Show up to section early for an informal chance to chat about things outside the course—I hope we get to know each other beyond CS50!

Think.
Pair.
Share.

- Why are we using **C**?
- How can we **read** and **write** code that includes **variables**, **conditionals**, and **loops**?
- Why do we care about **data types**?
- What does it mean to **compile** a C program?

Part 1

Variables and Types

Input and Printing

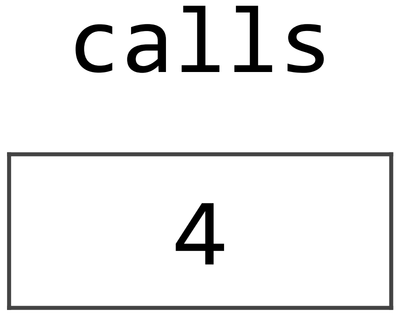
Variables

`calls`

$\boxed{4}$

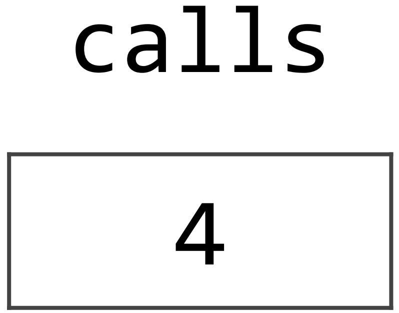
Variables

```
int calls = 4;
```



Variables

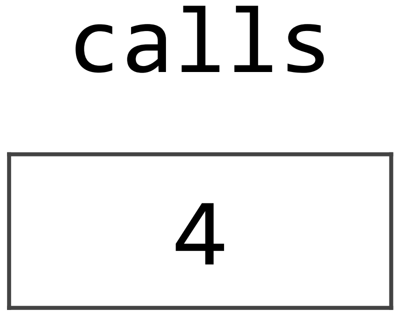
$$\texttt{int }\underbrace{\texttt{calls}}_{\text{name}}\texttt{ = 4;}$$



Variables

```
int calls = 4;
```

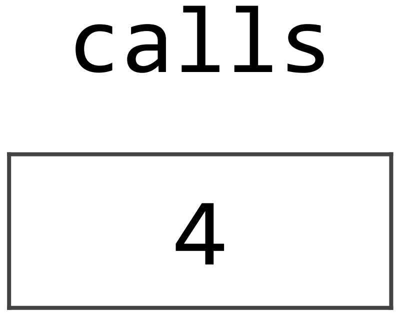
type



Variables

```
int calls = 4;
```

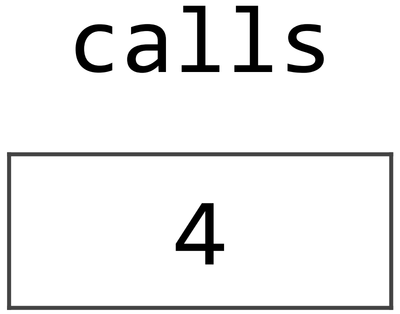
value



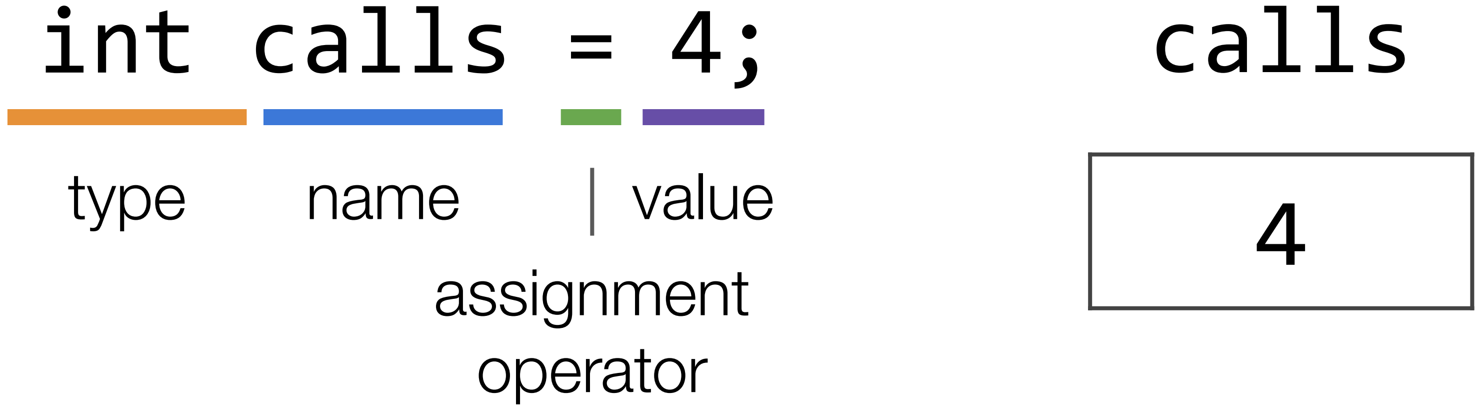
Variables

```
int calls = 4;
```

assignment operator



Variables

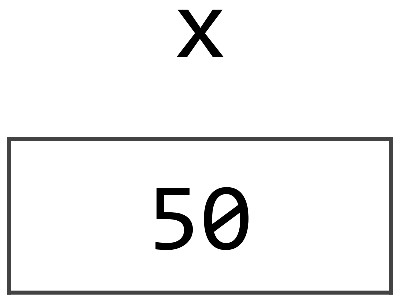


```
int calls = 4;
```

"Create an **integer** named **calls** that **gets** the **value 4**."

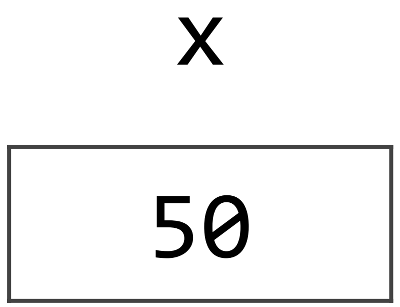
Variables

```
int x = 50;
```



Variables

```
int x = 50;
```



"Create an **integer** named **x** that **gets** the **value 50**."

Think.
Pair.
Share.

Why does C care about data types?

01000001

`int`

65

`01000001`

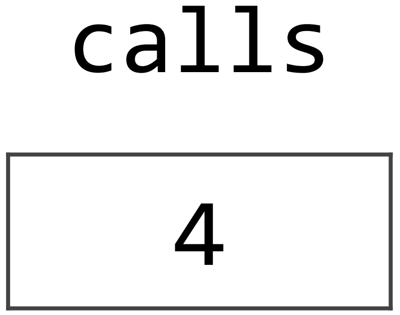
```
char

'A'

01000001
```

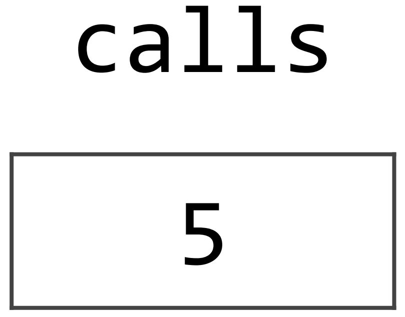
Variables

```
int calls = 4;
calls = 5;
```



Variables

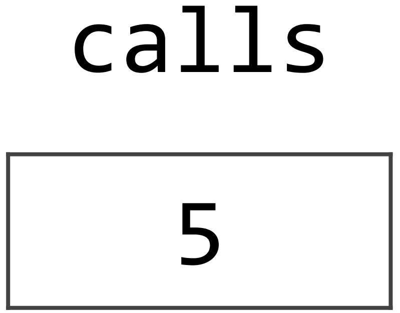
```
int calls = 4;
calls = 5;
```



Variables

```
int calls = 4;
```

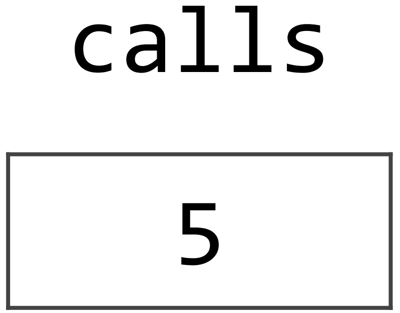
$$\underbrace{\texttt{calls}}_{\text{name}}\ \underbrace{\texttt{=}}_{\text{assignment operator}}\ \underbrace{\texttt{5;}}_{\text{value}}$$



"**Calls gets 5**."

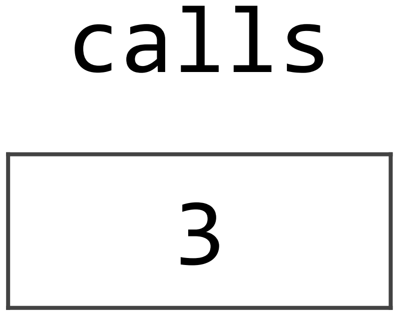
Operators

```
int calls = 4;
calls = calls + 1;
```



Operators

```
int calls = 4;
calls = calls - 1;
```



Operators

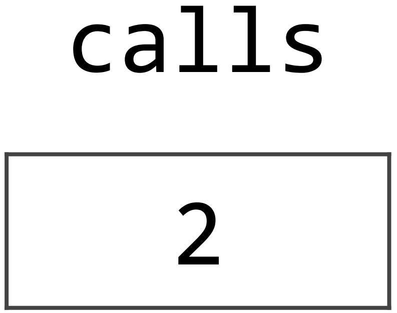
```
int calls = 4;
calls = calls * 2;
```

calls

$\boxed{8}$

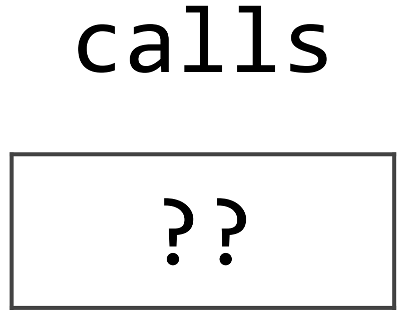
Operators

```
int calls = 4;
calls = calls / 2;
```



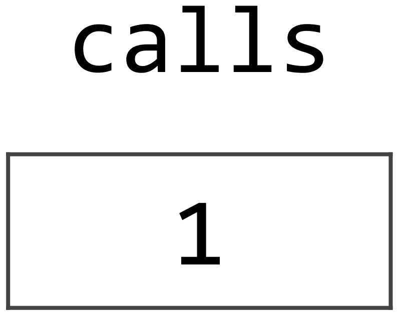
Operators

```
int calls = 4;
calls = calls / 3;
```

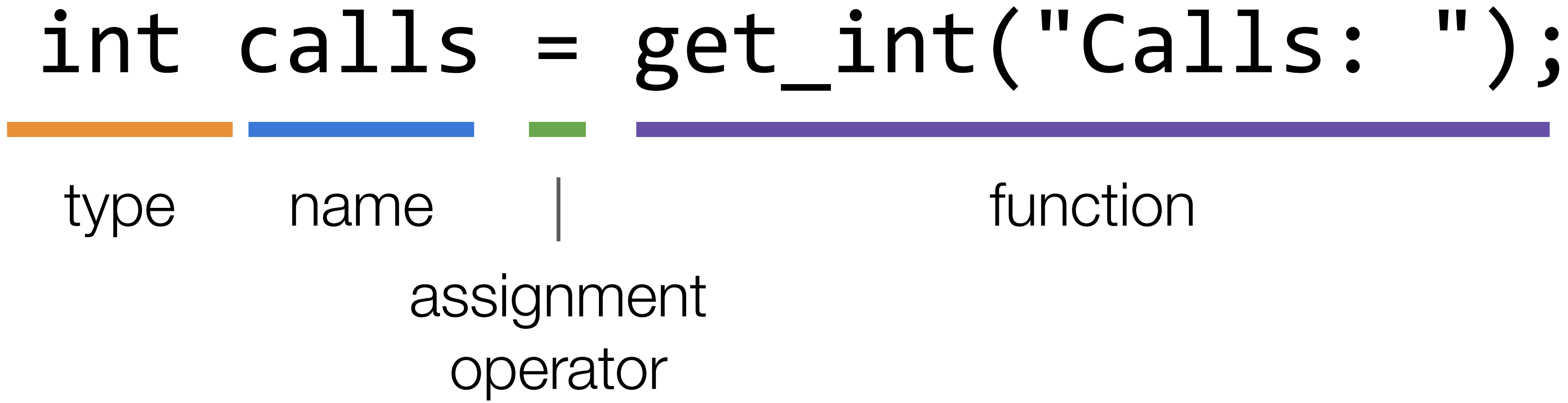


Operators

```
int calls = 4;
calls = calls / 3;
```



Getting input



```
int calls = get_int("Calls: ");
```

Functions

```
int calls = get_int("Calls: ");
```

function

Functions

```
int calls = get_int("Calls: ");
```

function name

Functions

```
int calls = get_int("Calls: ");
```

$\underbrace{\texttt{"Calls: "}}_{\text{function input}}$

Functions

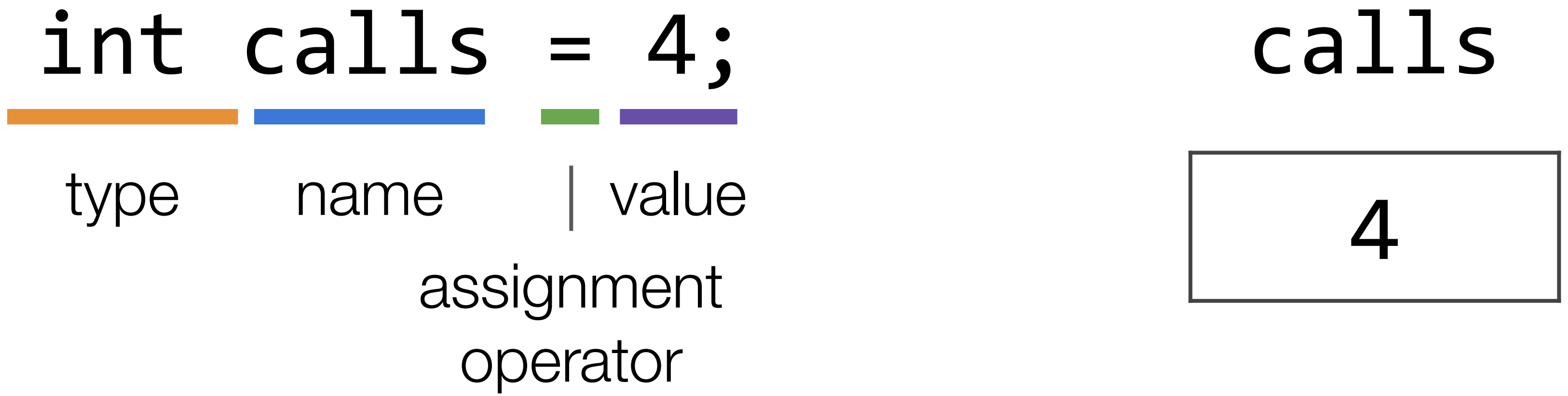
```
int calls = get_int("Calls: ");
```

$\underbrace{\texttt{get\_int("Calls: ");}}_{\text{function}}$

Return values

$$\texttt{int calls = }\underbrace{\texttt{4;}}_{\text{value}}$$

Storing return values



```
int calls = 4;
```

"Create an **integer** named **calls** that **gets** the **value 4**."

Printing values

```
int calls = 4;
printf("calls equals %i", calls);
```

Printing values

```
int calls = 4;
printf("calls equals %i", calls);
```

format code

Printing values

```c
int calls = 4;
printf("calls equals %i", calls);
```

placeholder value

Types and format codes

Numbers	Text	True/False
`int (%i)`	`char (%c)`	`bool (%d)`
`float (%f)`	`string (%s)`	

Types and format codes

Numbers	Text	True/False
`int (%i)`	`char (%c)`	`bool (%d)`
`float (%f)`	`string (%s)`	

Part 2

Hello, cs50.dev!

Part 3

breaking down loops
and conditionals

```
if (calls < 1)
{
    printf("Call more often!");
}
```

boolean expression

```
if (calls < 1)
{
    printf("Call more often!");
}
```

conditional

```
if (calls < 1)
{
    printf("Call more often!");
}
```

```
if (calls < 1)
{
    printf("Call more often!");
}
```

conditional code

```
if (calls < 1)
{
    printf("Call more often!");
}
else
{
    printf("Thanks for calling!");
}
```

```c
if (calls < 1)
{
    printf("Call more often!");
}
else
{
    printf("Thanks for calling!");
}
```

mutually exclusive

```c
int i = 0;
while (i < 10)
{
    printf("%i\n", i);
    i = i + 1;
}
```

initialization

$\downarrow$

```
int i = 0;
while (i < 10)
{
    printf("%i\n", i);
    i = i + 1;
}
```

boolean expression

```
int i = 0;
while (i < 10)
{
    printf("%i\n", i);
    i = i + 1;
}
```

```
int i = 0;
while (i < 10)
{
    printf("%i\n", i);
    i = i + 1;
}
```

increment

```
int i = 0;
while (i < 10)
{
    printf("%i\n", i);
    i = i + 1;
}
```

```
for (int i = 0; i < 10; i++)
{
    printf("%i\n", i);
}
```

initialization

$\downarrow$

```
for (int i = 0; i < 10; i++)
{
    printf("%i\n", i);
}
```

boolean expression

```
for (int i = 0; i < 10; i++)
{
    printf("%i\n", i);
}
```

increment

```
for (int i = 0; i < 10; i++)
{
    printf("%i\n", i);
}
```

```
for (int i = 0; i < 10; i++)
{
    printf("%i\n", i);
}
```

```
int n;
do
{
    n = get_int("N: ");
}
while (n <= 0);
```

```
int n;
do
{
    n = get_int("N: ");
}
while (n <= 0);
```

```
int n;
do
{
    n = get_int("N: ");
}
while (n <= 0);
```

Part 4

“int’s a me, Mario!”

- 123

- Work an example yourself
- Write down exactly what you did
- Create a generalization (algorithm) after working multiple examples
- Test your algorithm by hand
- Translate your algorithm to code
- Find bugs in your code by running test cases
- Debug (and critique) your code

- Work an example yourself
- Write down exactly what you did
- Create a generalization (algorithm) after working multiple examples
- Test your algorithm by hand
- Translate your algorithm to code
- Find bugs in your code by running test cases
- Debug (and critique) your code

- Work an example yourself
- Write down exactly what you did
- Create a generalization (algorithm) after working multiple examples
- Test your algorithm by hand
- Translate your algorithm to code
- Find bugs in your code by running test cases
- Debug (and critique) your code

What’s up next?

- Submit pset 1, check 1
- Section reassignments on Friday
- Office Hours throughout week
- Next time: arrays!

This was CS50 section.