

Variables Lesson Outline

1. Variables Lesson Outline
2. Data Types
3. What is a Variable?
4. What is a Variable? (With Examples)
5. What Does a Variable Have?
6. Who Chooses Each Variable Property?
7. The Value of a Variable Can Vary
8. Jargon: Compile Time and Runtime
9. Variable Declaration: Name & Data Type
10. Variable Declaration: Address
11. Variable Declaration: Initial Value #1
12. Variable Declaration: Initial Value #2
13. Variable Declaration: Initial Value #3
14. Variable Garbage Value Exercise
15. Declaration Section & Execution Section
16. Setting the Value of a Variable
17. Variable Assignment
18. Variable Assignment Example
19. Variable Assignment Example Program #1
20. Variable Assignment Example Program #2
21. The Same Source Code without Comments
22. Assignment is an Action, NOT an Equation #1
23. Assignment is an Action, NOT an Equation #2
24. Assignment is an Action, NOT an Equation #3
25. Changing a Variable's Contents
26. Changing a Variable's Contents: Example #1
27. Changing a Variable's Contents: Example #2
28. The Same Source Code without Comments
29. Setting the Value of a Variable
30. Variable Initialization
31. Variable Initialization Example #1
32. Variable Initialization Example #2
33. Initialize, Then Assign
34. The Same Source Code without Comments
35. C Variable Names
36. Favorite Professor Rule for Variable Names



Data Types

A **data type** is (surprise!) a type of data:

- Numeric
 - int: *integer*
 - float: *floating point* (also known as *real*)
- Non-numeric
 - char: *character*

Note that this list of data types **ISN'T** exhaustive –
there are many more data types (and you can define your own).

```
#include <stdio.h>
int main ()
{ /* main */
    float standard_deviation, relative_humidity;
    int    count, number_of_silly_people;
    char   middle_initial, hometown[30];
} /* main */
```



What is a Variable?

A ***variable*** is an **association** among:

- a **name**,
- an **address**,
and
- a **data type**.



What is a Variable? (With Examples)

A **variable** **is** an **association** among:

- a **name** (for example, `number_of_students`),
- an **address** (that is, a location in memory, such as 123456), and
- a **data type** (for example, `int`, `float`, `char`).



What Does a Variable Have?

Every variable has:

- a name (for example, `number_of_students`),
- an address (that is, a location in memory, such as 123456),
- a data type (for example, `int`, `float`, `char`),

AND

- a value, also known as the contents of the variable – specifically, the value is the contents of (what's inside) the variable's memory location.
(The value might be undefined, also known as garbage – more on this point soon.)



Who Chooses Each Variable Property?

Every variable has:

- a **name** (for example, `number_of_students`), chosen by the programmer;
- an **address** (that is, a location in memory, such as 123456), chosen by the compiler;
- a **data type** (for example, `int`, `float`, `char`), chosen by the programmer;
- a **value**, sometimes chosen by the programmer, and sometimes determined while the program is running (at **runtime**), for example based on one or more inputs. (The value might be **undefined**, also known as **garbage**.)



The Value of a Variable Can Vary

The value of a variable can vary; that is,
it can be changed at runtime.

We'll see how in a moment.



Jargon: Compile Time and Runtime

- Events that occur while a program is being compiled are said to happen at *compile time*.
- Events that occur while a program is running are said to happen at *runtime*.

For example:

- the **address** of a variable is chosen at **compile time**;
- the **value** of a variable typically is determined at **runtime**.



Variable Declaration: Name & Data Type

```
int x;
```

Remember: A program is a description of (1) a collection of data and (2) a sequence of actions on that data.

Before a program can use a variable, the program has to know

(a) that the variable **exists**, (b) what the variable's **name** is, and (c) what **type** of data the variable can have.

A **declaration** is a **statement** that tells the compiler all of these things: the variable **exists**, its **name**, and its **data type**.

For example, the declaration statement above tells the compiler to

- **choose a location** in memory for a variable,
- **name** that variable `x`,

and

- **think of that variable as** an `int`.

Note that the declaration above **doesn't specify a value** for `x`.



Variable Declaration: Address

```
int x;
```

The compiler might decide that `x` will live at, say,
address 3980 or address 98234092 or address 56436.

We don't know, and don't care, what address `x` lives at,
because the compiler will keep track of that for us.

It's enough to know that `x` has an address and that
the address of `x` will stay the same throughout
a given run of the program.



Variable Declaration: Initial Value #1

```
int x;
```

x: ???????? (address 56436)

When `x` is first declared, we don't know what its value is, because we haven't put anything into its memory location yet, so we say that its value is *undefined*, or, informally, *garbage*.

We'll see in a moment how to put values into our variables.



Variable Declaration: Initial Value #2

When x is first declared, we don't know what its value is, because we haven't put anything into its memory location yet, so we say that its value is undefined, or, informally, garbage.

Note: Some compilers for some languages automatically initialize newly declared variables to default values (for example, all integers might get initialized to zero), but not every compiler does automatic initialization.

You should NEVER NEVER NEVER assume that the compiler will initialize your variables for you.

You should ALWAYS ALWAYS ALWAYS explicitly give values to your variables in the body of the program, as needed.

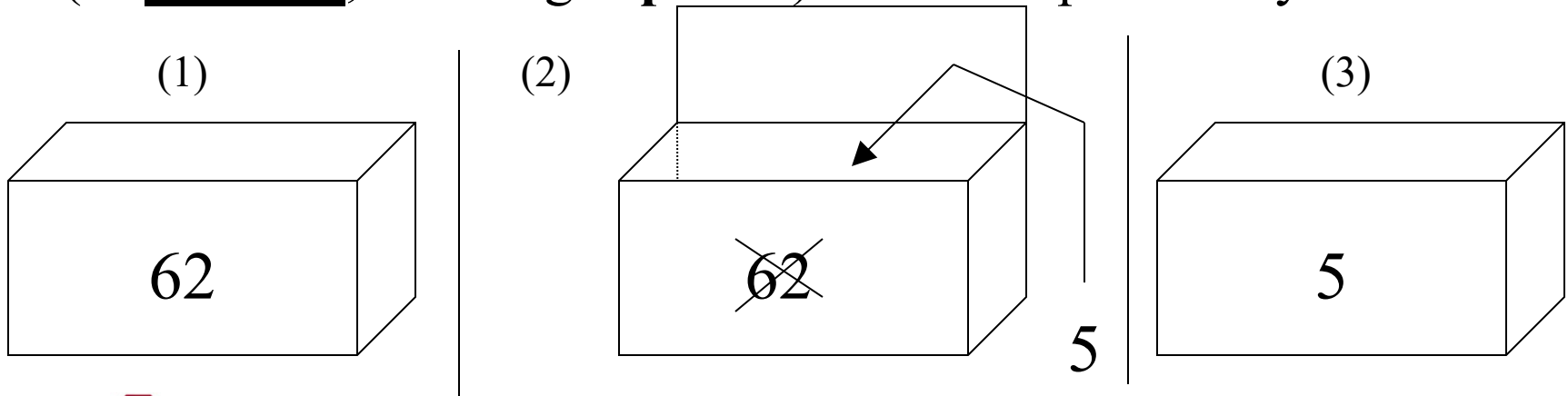


Variable Declaration: Initial Value #3

You can think of a variable's memory location as a box that always contains EXACTLY ONE THING AT A TIME.

So, if you haven't put anything into the box yet, then the contents of that box is whatever was left in it when the previous user finished with it.

You don't know what that value meant, so to you it's garbage. When you put your value into that box, the new value overwrites (or clobbers, meaning **replaces**) what was previously there.



Variable Garbage Value Exercise

- Think of an integer between 0 and 100 that is meaningful to you (for example, how many siblings you have, or your dog's age, or whatever).
- Take out a blank sheet of notebook paper (or share from a neighbor).
- Cut that sheet of paper in half, and then cut it in half again. (You can share the leftover quarter sheets with your neighbors.)
- On your quarter sheet of paper, write the integer you thought of.
- Fold your quarter sheet in half, and then fold it in half again.
- When everyone is ready, hand your foler quarter sheet to the person sitting to your left, but don't say anything.
- Let's see what happens!



Declaration Section & Execution Section

The **declaration section** of a program is the section of the program that contains all of the program's declarations.

The declaration section is always at the **beginning** of the program, just after the **block open** that follows the main function header:

```
#include <stdio.h>

int main ()
{ /* main */
    int height_in_cm;  ← Declaration Section

    height_in_cm = 160;
    printf("My height is %d cm.\n", height_in_cm);
} /* main */
```

Body <

The **execution section**, also known as the **body**, comes **after** the declaration section.



Setting the Value of a Variable

There are three ways to set the value of a variable:

- assignment;
- initialization;
- input.



Variable Assignment

An **assignment** statement sets the contents of a specific variable to a specific value:

$$x = 5;$$

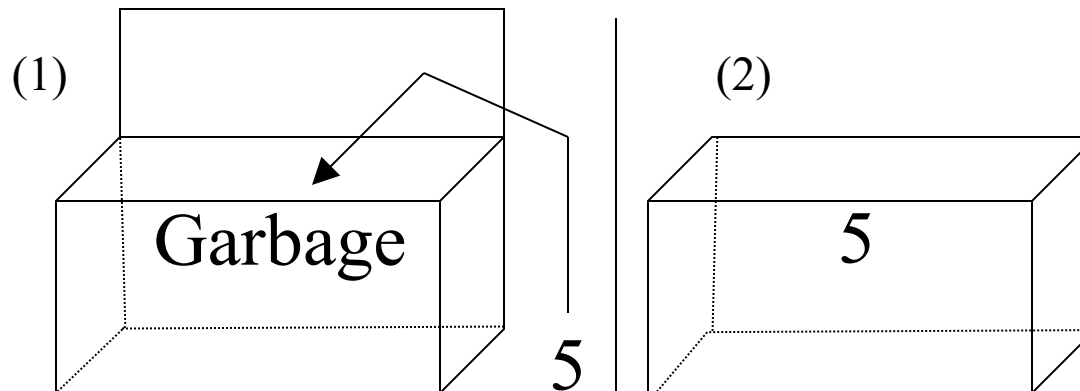
This statement tells the compiler to put the integer value 5 into the memory location named `x`, like so:

We say “`x` is assigned five” or “`x` gets five.”

`x:`

5

 (address 56436)



Variable Assignment Example

```
int x;
```

x:

????????

 (address 56436)

```
x = 5;
```

x:

5

 (address 56436)

```
x = 12;
```

x:

12

 (address 56436)

```
x = 5; /* We say "x gets 5" or "x is assigned 5." */
```

```
x = 12; /* We say "x gets 12" or "x is assigned 12." */
```



Variable Assignment Example Program #1

```
% cat assign.c
/*
*****
*** Program: assign ***
*** Author: Henry Neeman (hneeman@ou.edu) ***
*** Course: CS 1313 010 Spring 2025 ***
*** Lab: Sec 012Fridays 1:00pm ***
*** Description: Declares, assigns and ***
*** outputs a variable. ***
*****
*/
#include <stdio.h>

int main ()
{ /* main */
    /*
    *
    *****
    * Declaration section *
    *****
    *
    *****
    * Local variables *
    *****
    *
    * height_in_cm: my height in cm
    */
    int height_in_cm;
```



Variable Assignment Example Program #2

```
/*
*****
* Execution section *
*****
* Assign the integer value 160 to height_in_cm.
*/
height_in_cm = 160;
/*
* Print height_in_cm to standard output.
*/
printf("My height is %d cm.\n", height_in_cm);
} /* main */
% gcc -o assign assign.c
% assign
My height is 160 cm.
```



The Same Source Code without Comments

```
% cat assign.c
#include <stdio.h>

int main ()
{ /* main */
    int height_in_cm;

    height_in_cm = 160;
    printf("My height is %d cm.\n", height_in_cm);
} /* main */

% gcc -o assign assign.c
% assign
My height is 160 cm.
```



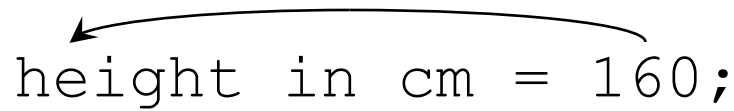
Assignment is an Action, NOT an Equation #1

An assignment is an **ACTION**, **NOT an equation**.

```
height_in_cm = 160;
```

An *assignment statement* means:

“Take the value on the right hand side of the single equals sign, and put it into the variable on the left hand side of the single equals sign.”



```
height_in_cm = 160;
```

(The phrase “single equals sign” will make sense in a few weeks, when we start to talk about Boolean expressions. For now, **ACCEPT IT ON FAITH.**)



Assignment is an Action, NOT an Equation #2

An assignment is an **ACTION**, **NOT an equation**.

```
#include <stdio.h>

int main ()
{ /* main */
    int height_in_cm;

    height_in_cm = 160;
    printf("My height is %d cm.\n", height_in_cm);
} /* main */
```

The **assignment statement**

```
height_in_cm = 160;
```

means “put the `int` value 160 into the memory location of the `int` variable named `height_in_cm`.”

OR, “`height_in_cm` gets 160.”



Assignment is an Action, NOT an Equation #3

An assignment is an **ACTION**, **NOT an equation** –
it means “do this,” **NOT** “this is the case.”

The variable whose value is being set by the assignment
MUST appear on the **left side** of the equals sign.

```
% cat not_an_equation.c
#include <stdio.h>
```

```
int main ()
{ /* main */
    int height_in_cm;
```

```
    160 = height_in_cm; ← ERROR!
```

```
    printf("My height is %d cm.\n", height_in_cm);
} /* main */
```

```
% gcc -o not_an_equation not_an_equation.c
```

```
not_an_equation.c: In function 'main':
```

```
not_an_equation.c:7: error: invalid lvalue in assignment
```



Changing a Variable's Contents

One way to change the value – the contents – of a variable is with another assignment statement.



Changing a Variable's Contents: Example #1

```
% cat change.c
/*
*****
*** Program: change ***
*** Author: Henry Neeman (hneeman@ou.edu) ***
*** Course: CS 1313 010 Spring 2025 ***
*** Lab: Sec 012Fridays 1:00pm ***
*** Description: Declares, assigns, changes ***
*** and outputs a variable. ***
*****
*/
#include <stdio.h>
int main ()
{ /* main */
    /*
    *****
    * Declaration section *
    *****
    *
    *****
    * Local variables *
    *****
    *
    * height_in_cm: my height in cm
    */
    int height_in_cm;
```



Changing a Variable's Contents: Example #2

```
/*
*****
* Execution section *
*****
* Assign the integer value 160 to height_in_cm.
*/
height_in_cm = 160;
/*
* Print height_in_cm to standard output.
*/
printf("My height is %d cm.\n", height_in_cm);
/*
* Assign the integer value 200 to height_in_cm.
*/
height_in_cm = 200;
/*
* Print height_in_cm to standard output.
*/
printf("My height is %d cm.\n", height_in_cm);
} /* main */
% gcc -o change change.c
% change
My height is 160 cm.
My height is 200 cm.
```



The Same Source Code without Comments

```
% cat change.c
#include <stdio.h>

int main ()
{ /* main */
    int height_in_cm;

    height_in_cm = 160;
    printf("My height is %d cm.\n", height_in_cm);
    height_in_cm = 200;
    printf("My height is %d cm.\n", height_in_cm);
} /* main */
% gcc -o change change.c
% change
My height is 160 cm.
My height is 200 cm.
```

Recall that a program is a collection of data and
a **SEQUENCE of actions.**



Setting the Value of a Variable

There are three ways to set the value of a variable:

- assignment;
- **initialization**;
- input.



Variable Initialization

To *initialize* a variable means
to declare it and assign it a value in the same statement:

```
int x = 5;
```

This statement is **EXACTLY THE SAME** as
declaring `x` in the declaration section, and then
IMMEDIATELY assigning it 5 at the beginning of
the execution section.

For example:

```
int x;
```

```
x = 5;
```

means **EXACTLY THE SAME** as:

```
int x = 5;
```



Variable Initialization Example #1

```
% cat initialize.c
/*
*****
*** Program: initialize ***
*** Author: Henry Neeman (hneeman@ou.edu) ***
*** Course: CS 1313 010 Spring 2025 ***
*** Lab: Sec 012Fridays 1:00pm ***
*** Description: Declares/initializes and ***
*** outputs a variable. ***
*****
*/
#include <stdio.h>

int main ()
{ /* main */
    /*
    *****
    * Declaration section *
    *****
    *
    *****
    * Local variables *
    *****
    *
    * height_in_cm: my height in cm
    */
    int height_in_cm = 160;
```



Variable Initialization Example #2

```
/*
*****
* Execution section
*****
*
* Print height_in_cm to standard output.
*/
printf("My height is %d cm.\n", height_in_cm);
} /* main */
% gcc -o initialize initialize.c
% initialize
My height is 160 cm.
```



The Same Source Code without Comments

```
% cat initialize.c
#include <stdio.h>

int main ()
{ /* main */
    int height_in_cm = 160;

    printf("My height is %d cm.\n", height_in_cm);
} /* main */
% gcc -o initialize initialize.c
% initialize
My height is 160 cm.
```



Initialize, Then Assign

You can initialize a variable in the declaration section, and then change its value in the execution section (body) via an assignment statement.

```
% cat initialize_assign.c
#include <stdio.h>

int main ()
{ /* main */
    int height_in_cm = 160;

    printf("My height is %d cm.\n", height_in_cm);
    height_in_cm = 200;
    printf("My height is %d cm.\n", height_in_cm);
} /* main */

% gcc -o initialize_assign initialize_assign.c
% initialize_assign
My height is 160 cm.
My height is 200 cm.
```



C Variable Names

C identifiers (including variable names) have the following properties:

- Constructed using only these characters:
 - Letters (case sensitive: it matters whether it's upper case or lower case)

a b c d e f g h i j k l m
n o p q r s t u v w x y z
A B C D E F G H I J K L M
N O P Q R S T U V W X Y Z

- Digits

0 1 2 3 4 5 6 7 8 9

- Underscore (NOTE: NOT hyphen)

—

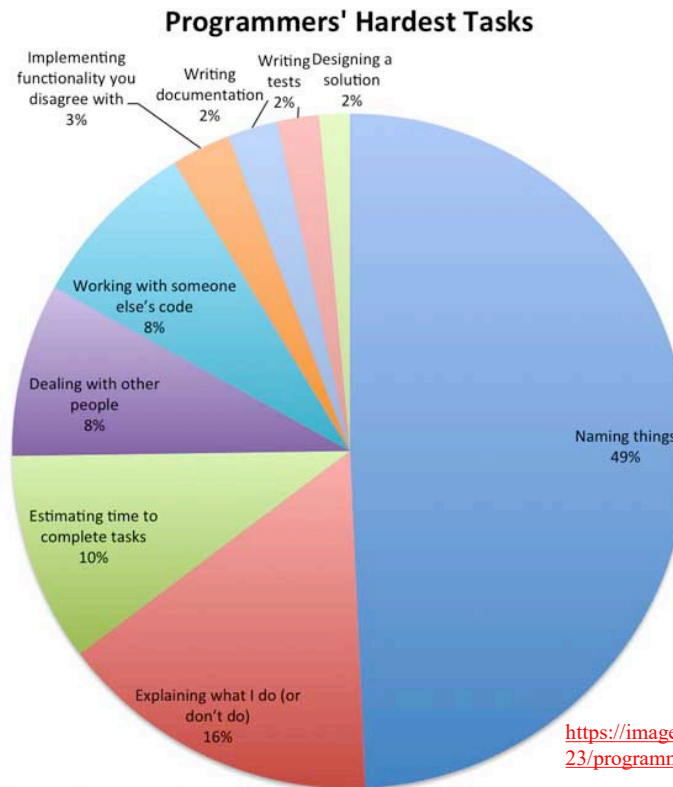
- The first character **MUST** be a letter or an underscore:

a123_456 is good, and so is _a123456,
but not 1a23_456



Favorite Professor Rule for Variable Names

A variable name should be so **obvious** that your favorite professor in your major, even if they know nothing about programming, could immediately tell what that variable name means.



https://images.techhive.com/images/idge/imported/article/itw/2013/10/23/programmers_hardest_tasks-600x700-100521914-orig.jpg

Data Source: Quora/Ubuntu Forums
Total Votes: 4,522

