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. 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. Declaration Section & Execution Section
15. Setting the Value of a Variable
16. Variable Assignment
17. Variable Assignment Example
18. Variable Assignment Example Program #1
19. Variable Assignment Example Program #2
20. The Same Source Code without Comments
21. Assignment is an Action, NOT an Equation #1
22. Assignment is an Action, NOT an Equation #2
23. Assignment is an Action, NOT an Equation #3
24. Changing a Variable’s Contents
25. Changing a Variable’s Contents: Example #1
26. Changing a Variable’s Contents: Example #2
27. The Same Source Code without Comments
28. Variable Initialization
29. Variable Initialization Example #1
30. Variable Initialization Example #2
31. The Same Source Code without Comments
32. C Variable Names
33. 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 **ISN’T** complete.

```c
#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** between:

- a **name**,
- an **address**, and
- a **data type**.
A *variable* is an *association* between:

- 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** (which may be **undefined**, also known as **garbage**).

The value is also known as the **contents** of the variable — that is, the value is the contents of the variable’s memory location.
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** (which may be **undefined**, also known as **garbage**), sometimes chosen by the programmer, and sometimes determined while the program is running (at **runtime**).
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.
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, while its value often is determined at runtime.
A **declaration** is a **statement** that tells the compiler that an item of data (for example, a variable) **exists**, and what some of its **properties** are (specifically, its name and its data type).

For example, the declaration above tells the compiler to

- choose a location in memory,
- name it `x`,

and

- think of it as an integer.

Note that the declaration above **doesn’t specify a value** for `x`. 
Variable Declaration: Address

```c
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 take care 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

```c
int x;
```

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 get initialized to zero), but **not** every compiler does automatic initialization.

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

You should **ALWAYS ALWAYS ALWAYS ALWAYS** explicitly assign 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.

So, if you haven’t put anything into the box yet, then the contents of the box is **whatever was left in it** when the previous user got done with it.

You don’t know what that value means, so to you it’s **garbage**. When you put your value into that box, it **clobbers** what was there.

![Diagram with boxes showing the process of putting values into a box.](image)
Declaration Section & Execution Section

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

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

```c
#include <stdio.h>

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

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

```plaintext
int x;

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

x = 5;

x: 5 (address 56436)

x = 12;

x: 12 (address 56436)
```
% cat assign.c
/*
 *********************************************
 *** Program: assign ***
 *** Author: Henry Neeman (hneeman@ou.edu) ***
 *** Course: CS 1313 010 Spring 2015 ***
 *** Lab: Sec 012 Fridays 1:00pm ***
 *** Description: Declares, assigns and ***
 *** outputs a variable. ***
 *********************************************
 */
#include <stdio.h>

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

/*
 * Print height_in_cm to standard output.
 */
printf("My height is %d cm.\n", height_in_cm);

% gcc -o assign assign.c
% assign
My height is 160 cm.
The Same Source Code without Comments

```
#include <stdio.h>

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

% 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;
```
Assignment is an Action, NOT an Equation #2

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

```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);
}
```

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`.”
Assignment is an Action, NOT an Equation #3

An assignment is an **action, NOT an equation**.
The variable whose value is being set by the assignment **MUST** appear on the **left side** of the equals sign.

```c
#include <stdio.h>

int main ()
{
    int height_in_cm;

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

% **cat not_an_equation.c**

```c
#include <stdio.h>

int main ()
{
    int height_in_cm;

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

% **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 2015
 * Lab: Sec 012 Fridays 1:00pm
 * Description: Declares, assigns, changes and outputs a variable.
 */

#include <stdio.h>

int main ()
{
    /* main */
    /*
     * Declaration section
     */
    int height_in_cm;
}
Changing a Variable’s Contents: Example #2

```c
/*
 *********************************************
 * 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 ()
{
    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);
}

% gcc -o change change.c
% change
My height is 160 cm.
My height is 200 cm.
```
Variable Initialization

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

\[
\text{int } x = 5;
\]

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:

\[
\text{int } x;
\]
\[
x = 5;
\]
# Variable Initialization Example #1

```c
#include <stdio.h>

int main ()
{
    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.
```
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** is 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 \textit{obvious} that your favorite professor in your major, even if they know nothing about programming, could immediately tell what the variable name means.