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* among:

- a *name*,
- an *address*,
  and
- a *data type*. 
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 – that is, the value is the contents of the variable’s memory location. (The value might be **undefined**, also known as **garbage**.)
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; 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 \textit{compile time}.
- Events that occur while a program is running are said to happen at \textit{runtime}.

For example:

- the \textbf{address} of a variable is chosen at \textit{compile time};
- the \textbf{value} of a variable typically is determined at \textit{runtime}.
int x;

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

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`. 
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.
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 \textit{undefined}, or, informally, \textit{garbage}.

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

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 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.

62

5
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.
- Cut that sheet of paper in half, and then cut it in half again.
- On that quarter sheet of paper, write the integer you thought of.
- Fold the quarter sheet in half, and then fold it in half again.
- Hand it 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:

```c
#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 */
```

The **execution section**, also known as the **body**, comes **after** the declaration section.
There are three ways to set the value of a variable:

- assignment;
- initialization;
- input.
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.”

\[ x: \boxed{5} \text{ (address 56436)} \]
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." */
% cat assign.c
/
  *********************************************
  *** Program: assign                     ***
  *** Author: Henry Neeman (hneeman@ou.edu) ***
  *** Course: CS 1313 010 Fall 2022         ***
  *** Lab: Sec 014 Fridays 1:30pm            ***
  *** 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.**)
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);
} /* 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**. 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

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

    /*
     * Declaration section
     *
     */
    int height_in_cm;
}
```

Changing a Variable’s Contents: Example #1
Changing a Variable’s Contents: Example #2

/*
 * Execution section *
 */
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);
} /* main */

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

Remember, a program is a collection of data and a SEQUENCE of actions.
Variable Initialization

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

```c
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:

```c
int x;
x = 5;
```

means **EXACTLY THE SAME** as

```c
int x = 5;
```
Variable Initialization Example #1

```c
% cat initialize.c
/
  *********************************************
  *** Program: initialize ***
  *** Author: Henry Neeman (hneeman@ou.edu) ***
  *** Course: CS 1313 010 Fall 2022 ***
  *** Lab: Sec 014 Fridays 1:30pm ***
  *** 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;
```
/*
  ******************************************
  * 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 **obvious** that your favorite professor in your major, even if they know nothing about programming, could immediately tell what that variable name means.

![Diagram showing the hardest tasks for programmers](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*