Intro to Programming - Variables and Types

CSC 100 - Lyon College - Spring 2025

Marcus Birkenkrahe

February 1, 2025

What are you going to learn?

Topics:

- ☐ Variables (`height`, `width`)
- ☐ Types (`int`, `float`, `char`)
- ☐ Assignment (`=`)
- ☐ Keywords (`return`, `void`, `main`)
- ☐ Operators (`*`, `+`, `/`, `-`)
- ☐ Formatting integers and floats

In-class and home assignments

- **Reading**: Chapter 2 (**Think C**) - pp. 13-22.
- **Practice exercises** (in-class):
  1. Declaration of variables
  2. Changing literals
- **Programming assignments (home)**:
  1. Temperature converter.
  2. Bonus: Algebra with C

Variables

- Just like your decision-making depends on what you know (and remember), the computer's ability to compute depends on memory.
- When you give it a job, the computer needs to carefully plan its memory - very much like when you schedule your day, except not in units of time but in bits.
- One strength of C is that you have all the power of scheduling the computer's memory. One weakness is that you must do this.
- The price: To decide how much memory to reserve for its operations, you need to assign a *data type* to every number or string that you need.
- The prize: Once you've done this (picked types), you can change the *value* in the reserved memory cells, which are also called *variables*.
- Examples of data type declarations:

```
int height;   // an integer variable called ‘height‘
float profit;  // a floating-point variable called ‘profit‘
char name;  // a character variable called ‘name‘
```

- After this point, the compiler will have reserved enough memory for these three variables. If you don't reserve memory, you cannot use the variable.
- To test that, write just `height;` in the `main` program and run it. This will not work unless `height` was declared first.

  Wrong:

```
height;  // cannot use ‘height‘ without declaring it first
```

  Right:

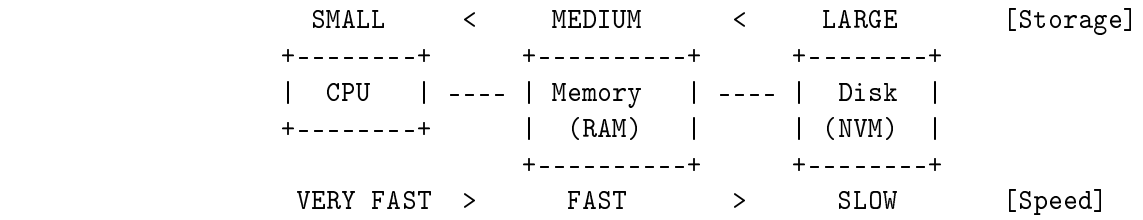
```
int height; // declare integer variable ‘height‘
height;  // cannot use ‘height‘ without declaring it first
```

- If you use an editor, you can see the syntax highlighting, which is chosen to indicate keywords and variable names (not always useful).
- You can also declare several variables of the same type on one line:

```
int height, width;
```

Why is the computer so serious about memory?

- The computer has dementia built into its very fabric. This is also called von-Neumann architecture - the way all computers are built:



- When you write a program, it is stored on the disk (NVM) and has to be loaded into memory (RAM) so that the brain (CPU) can get to it. And every time the computer is switched off, the memory is erased completely, and only the disk (Non-Volatile Memory) remains, so everything has to be written to it that you wish to keep.

- The faster the device (CPU $>$ RAM $>$ NVM) the more expensive the memory. The memory determines the performance of your computations more than anything - even though most programming languages never make you think about it!

- The less information the computer has about the data you intend to feed to it to work with, the lower its performance. This makes no difference if you play Candy Crush, but if you play an online multiplayer game, or if your computer is on board of an autonomous vehicle, or on a space ship, or on an airplane... it matters much.

PRACTICE Variable types and declarations

Write a complete program that declares two floating-point variables, `fahrenheit` and `celsius`, and print their values.

Example output:

```
0.000000 Fahrenheit = 0.000000 Celsius
```

Solution:

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

```
int main(void)
{
  float fahrenheit, celsius;
  printf("%f Fahrenheit = %f Celsius\n", fahrenheit, celsius);
  return 0;
}
```

Variable assignment: integers

- A variable gets its value through **assignment** with the `=` *operator*.
- In the code block below, the variable `height` gets the value `8` with the assignment operator `=`. The number `8` is also called a **"literal"** because it cannot change.

  ```
  height = 8;
  ```

- If you would try to run the code above, you would get an error. Can you see why?

  > **Answer:** An assignment counts as use of a variable. The type of a variable must be declared before you can use it.

- The example below would throw another error. What's wrong now?

  ```
  height = 8;
  int height;
  ```

  > **Answer:** The declaration of variable must precede its first use.

- Phew! The next block finally works, that is, it compiles and runs. But what does the code actually do?

  ```
  int height;
  height = 8;
  ```

  > **Answer:** `int height;` reserves memory for an integer variable called `height`, and `height = 8;` puts the numeric integer value `8` into the corresponding memory cell. From now on, whenever you use `height`, the computer will substitute the value `8` for it.

PRACTICE Changing literals

1. Try to change the value of the literal (constant) `8` by assigning another integer to it, e.g. `8 = 18`. What's the result?

2. What is the error message when you compile this program:

   ```
   8 = 18;
   ```

   Error message:

   ```
   literal.c: In function ‘main’:
   literal.c:3:8: error: lvalue required as left operand of assignment
       3 |      8 = 18;
         |        ^
   ```

3. Can you understand it? If you don't know what *lvalue* means, google it.

   > **Explanation:** ‘lvalue‘ stands for ‘left value‘ and stands for an object that you can assign a value to. But in the statement, `8` is not a variable but a literal or constant. Its memory is **persistent**, and not **variable**.

4. How then can you change a variable called `foo` from the value `8` to `18`? Answer with **two lines** of code.

   ```
   int foo = 8; // declare ‘foo‘ and define its value
   foo = 18;  // change value of ‘foo‘
   ```

5. Test the code by inserting suitable print statements. Output:

   ```
   foo = 8
   foo = 18
   ```

   ```
   int foo = 8; // declare ‘foo‘ and define its value
   printf("foo = %i\n",foo);
   foo = 18;  // change value of ‘foo‘
   printf("foo = %i\n",foo);
   ```

Variable assignment: floating-point numbers

- A *floating-point literal* assigned to a `float` variable contains a decimal point and the letter `f` to indicate its "floatiness":

```
float profit;
profit = 2150.48f;  // 'f' specifies the float format
```

- Assigning a `float` to an `int` and vice versa is possible (but not safe as we will see) - the compiler will not warn you:

```
/* ASSIGNING A floating point value TO AN integer variable*/
int iProfit = 2150.48;  // Don't do this!

/* ASSIGNING AN integer TO AN floating point variable */
float profit = 2150;  // Don't do this!
```

- Variables with values can be used to compute other values:

```
// variable declarations
int height, length, width, volume;

// variable assignments
height = 8;
length = 12;
width = 10;

// variable evaluation
volume = height * length * width;
```

- How many tasks does this last little program perform?

  Answer:

  1. Declare four variables (4)
  2. Assign three variables (7)
  3. Compute one variable (8)

  8 tasks: The program messes 8 times with computer memory.

PRACTICE Declare, Assign, Compute and Print

- You can also initiate and declare several variables at once:

  1. Declare **and** assign = define `height` 8, `length` 12, `width` 10.
  2. Compute the volume **inside** the `printf` function call.
  3. How many tasks does this program perform?

  Solution:

```
int height = 8, length = 12, width = 10; // 6
printf("Volume: %i", height * length * width); // + 1
```

PRACTICE In-Class Practice 4: Variable Manipulation and Formatted Output

Objective

Practice variable declaration, assignment, arithmetic operations, and formatted output in C.

Instructions

1. Write a C program =
2. Declare an integer variable named `num` and initialize it to 10.
3. Declare an integer variable named `doubleNum` that holds twice the value of `num`.
4. Declare a floating-point variable named `half` that holds half of the value of `num`.
5. Print the following three lines using a single `printf` statementwith the correct format specifiers:
   - "The number is: 10"
   - "Double the number is: 20"
   - "Half the number is: 5.000000"
6. Ensure your program includes the proper header files and returns 0 from the `main` function.

Sample Output

```
The number is: 10
Double the number is: 20
Half the number is: 5.000000
```

Program template:

```
#include <stdio.h>

/* main program */
int main(void)
{
  // declare and define variables

  // print results

  return 0;
}
```

Submit your result to Canvas! (`In-class practice 4: Variables`)

Sample solution:

Link: `https://onecompiler.com/c/437ujapsy`

```
/*****************************************************
 * variables.c: Declare, define and print variables
 * Input: None
 * Output: two integers and one floating-point variable
 * Author: Marcus Birkenkrahe
 * Date: 02/01/2025
 *****************************************************/
#include <stdio.h>

/* main program */
int main(void)
{
  // declare and define variables
  int num = 10;
  int doubleNum = 2 * num;
```

```
  float half = 0.5 * num;

  // print results
  printf("The number is: %i\nDouble the number is: %i\nHalf the number is: %f",
 num, doubleNum, half);

  return 0;
}
```

Glossary

TERM	EXPLANATION
Variable	A memory location that stores data which can change.
Data Type	Type of data a variable can store (`int`, `float`, `char`).
Assignment	The process of giving a value to a variable using `=`.
Keyword	Reserved words in C (e.g., return, void, main).
Operator	Symbols used for calculations (e.g., `+`, `-`, `*`, `/`).
Precision	The accuracy of numerical representations in memory.
Memory	System storage used for computation (RAM, CPU cache, etc.)
Literal	A fixed value assigned to a variable (e.g., `8` in `height = 8;`).
Lvalue	A memory location that can hold a value (i.e., a variable).
Floating-Point	A number that includes a decimal component (e.g., `2.5f`).
Format Specifier	A placeholder used in `printf` to print variables (e.g., `%i`)

Summary

This section introduces key programming concepts in C, starting with variables and memory management. It explains how variables function like reserved memory spaces and the necessity of defining their types before use. Assigning values with the `=` operator is covered, along with best practices for declaring multiple variables.

A deeper dive into the importance of memory highlights how computing performance is affected by variable storage. The von Neumann architecture is briefly discussed, emphasizing the need for memory efficiency.

Exercises reinforce variable declarations, assignments, and modifying literals, encouraging hands-on practice. The section concludes with an introduction to floating-point numbers, format specifiers in `printf`, and arithmetic operations using variables.