CSC111 - Admin

Joe Krysl

Marking Scheme

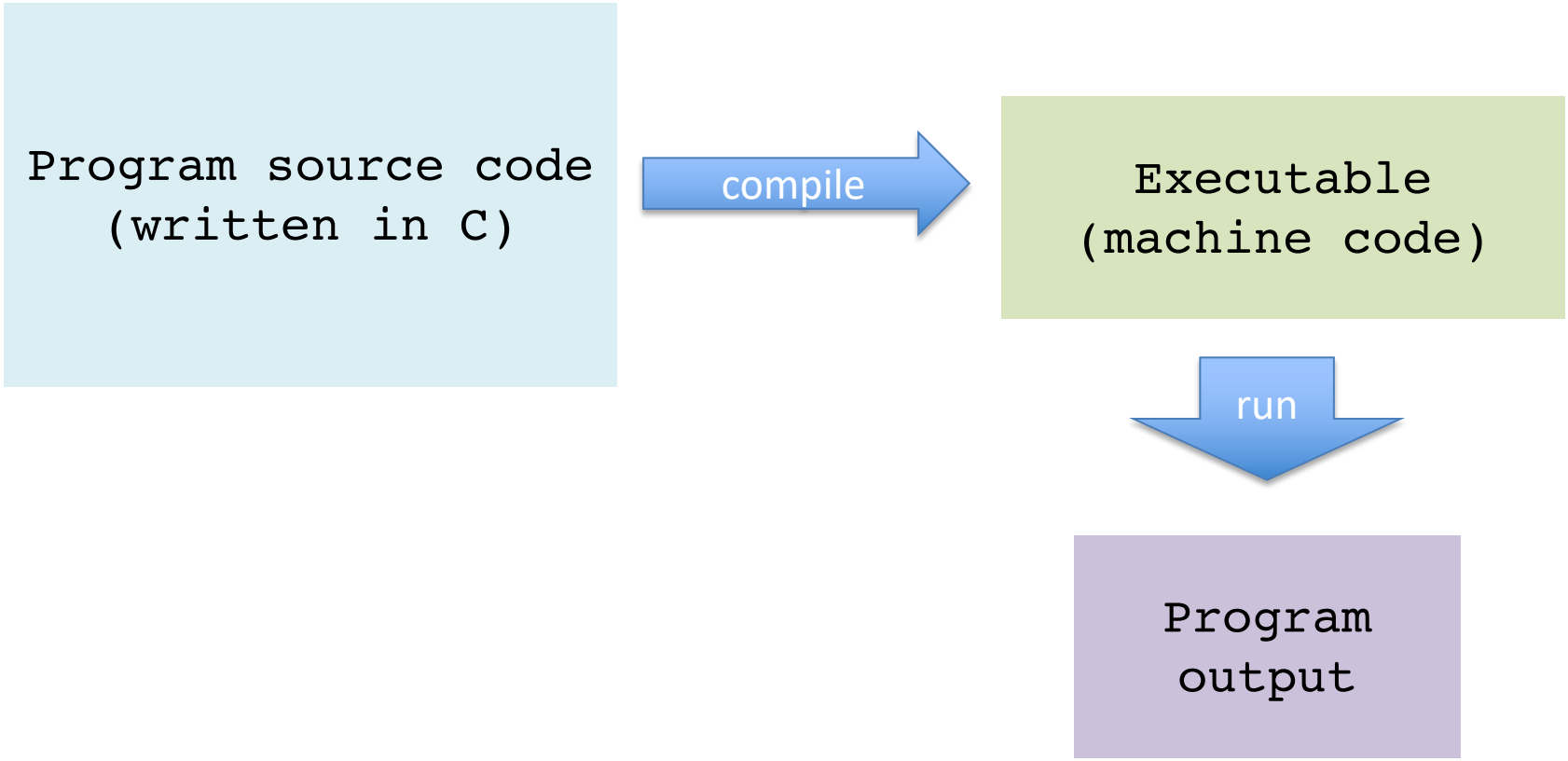
- 3 schemes –> you will automatically get the highest mark from the three.
- Labs –> More information Monday, please plan for a Zoom Lab next week
- Pre-Lecture work –> watch the videos and complete the quiz.
- Questions -> post to BrightSpace forums, unless it is of a personal nature.
  - I will not be replying to emails without CSC111 in the subject line
  - I will not be replying to emails originating from a non UVIC address

Intro To C

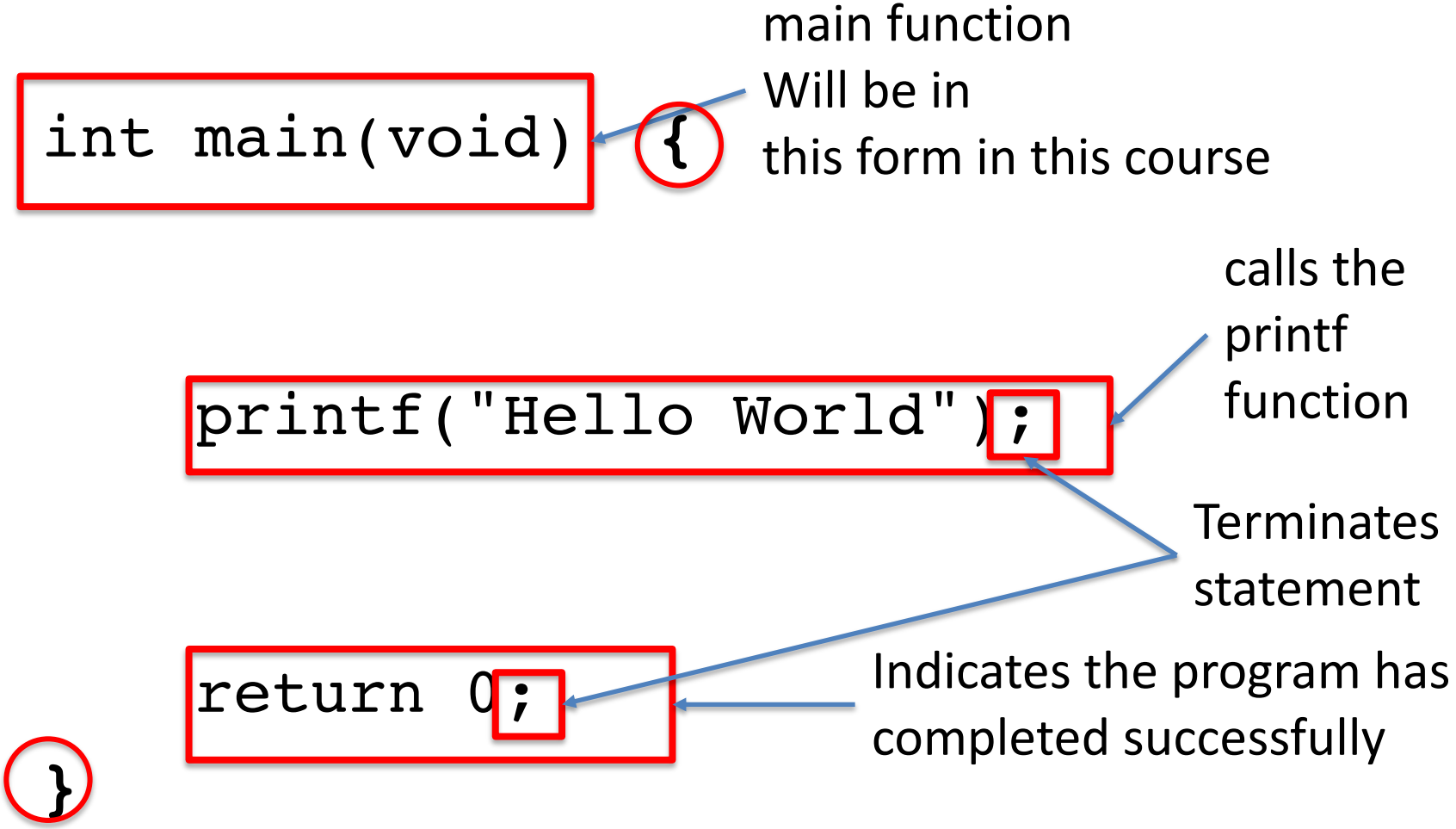
Learning outcomes

- Compilation process
- Writing a basic C program
- Compiler directives
- Variables
- Basic calculations
- Code quality best practices
- Primitive types

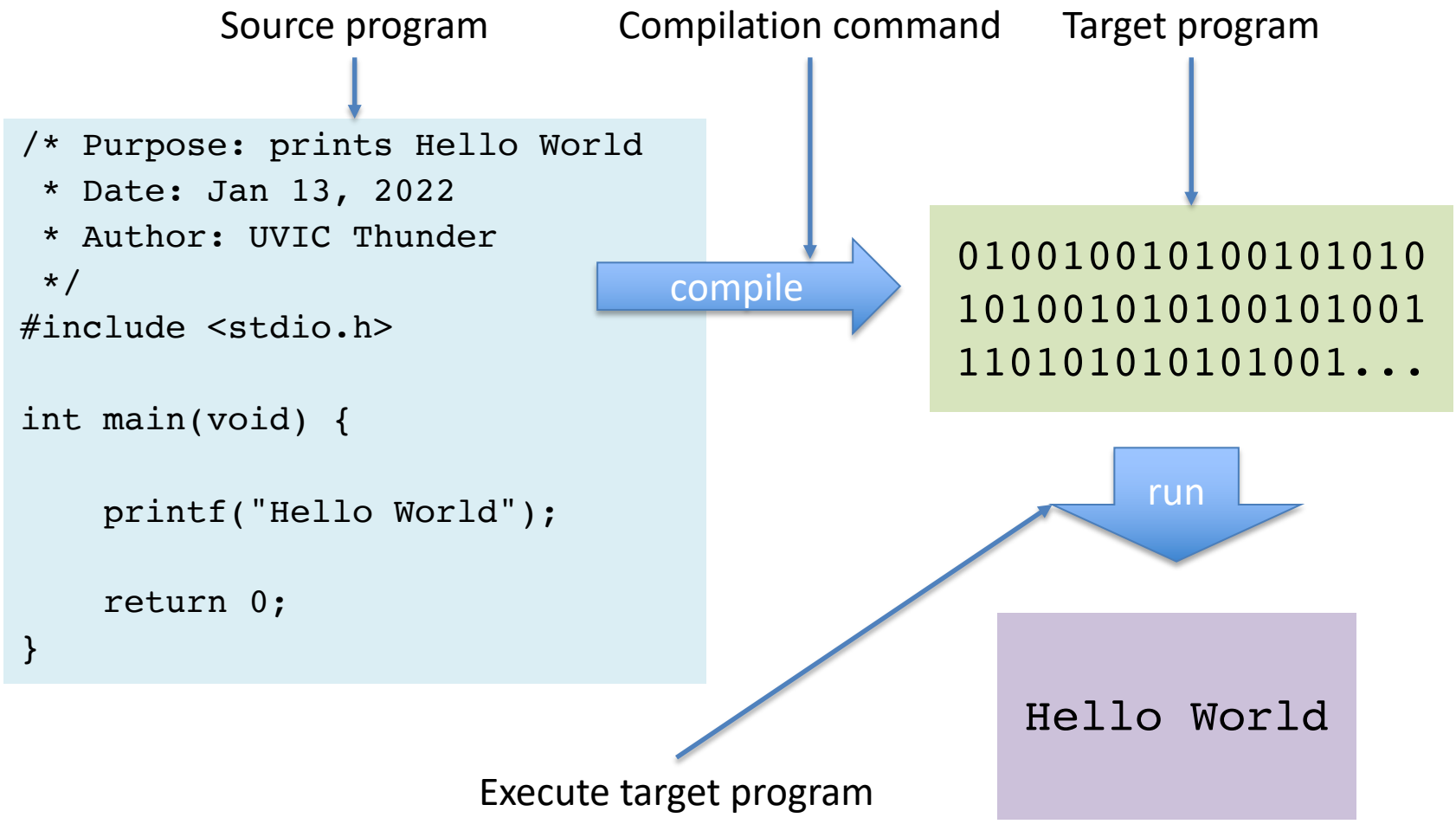
The translation process



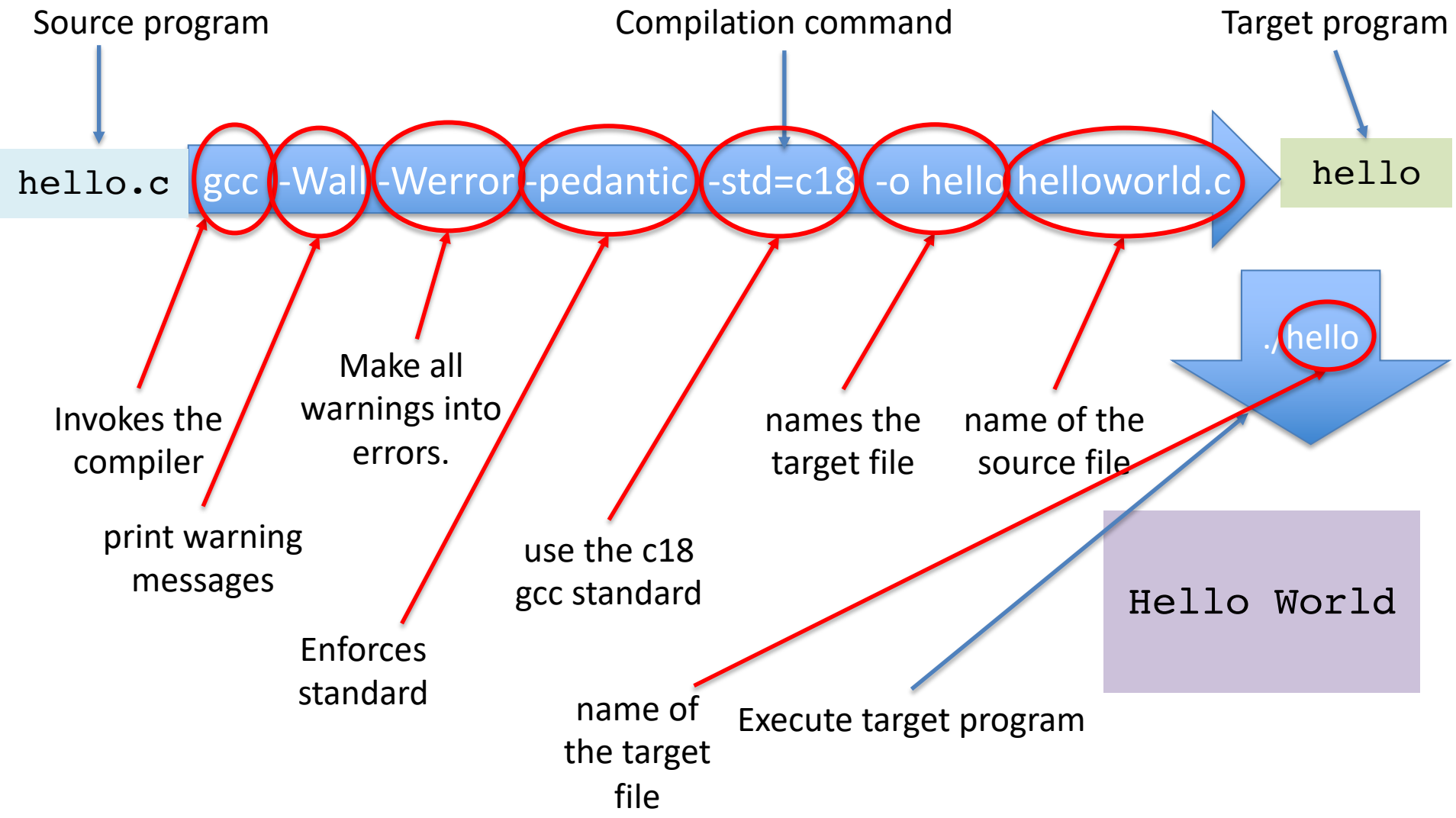
Zooming in on main



Sample C program



Dissecting the compilation & running



Demo

Variables

- A name that represents a value stored in memory is declared by specifying the type of value it represents and a variable name (identifier):

```
double x;
double y;
```

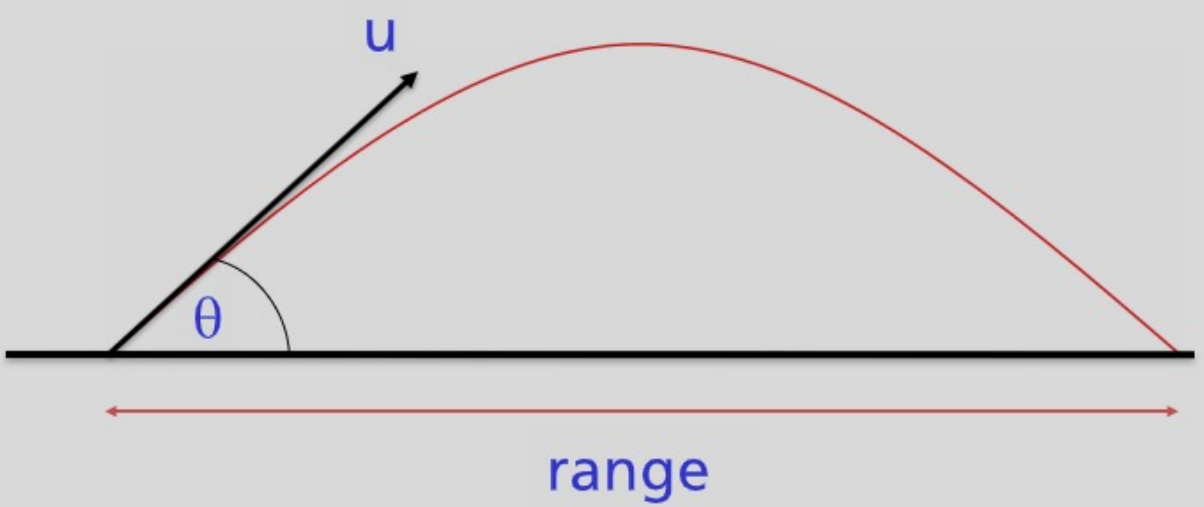
- At this point x and y do not have an assigned value.

- You can assign a value to a variable using the = operator
  the result of the expression on the right hand side of the =
  is stored to the variable specified on the left hand side of the =

```
x = 5.2;
y = 3.1 + 2.4;
x = x + y;
```

Constructing more useful programs

Projectile Problem



$$R = \frac{u^2}{g}\sin(2\theta)$$

```c
/* Purpose: Computes the range of a projectile
     going 250.7m/s at an angle of 0.52 Radians
   Date: Jan 6, 2022
   Author: CSC111 Instructor
*/
#include <stdio.h>
#include <math.h>
#define GRAV_ACEL 9.81

int main(void) {
    double initial_speed;
    double angle_of_elevation;
    double range;

    initial_speed = 250.7;
    angle_of_elevation = 0.52;

    range = (initial_speed * initial_speed)
            / GRAV_ACEL * sin(2 * angle_of_elevation);

    printf("Range is: %f meters\n", range);

    return 0;
}
```

special characters

Escape Sequence	Effect
`\n`	Forces output to print a newline (advances output to the next line)
`\t`	Forces output to print a tab (advances output to next horizontal tab position)
`\"`	Forces double quote to print
`\\`	Forces backslash to print

EXAMPLE:

```
printf("She said \"Hi\"!");
```

Output: `She said "Hi"!`

Code quality (best practices)

- documentation
- meaningful names
- indentation
- whitespace
- descriptive user output

Variable Naming Rules

- Rules for naming variables in C:
  - Variable name cannot be a C key word
  - Variable name cannot contain spaces
  - First character must be a letter or an underscore
  - After first character may use letters, digits, or underscores
  - Variable names are case sensitive
- Variable name should reflect its use
- We will use typical C variable naming conventions:
  - Begin with a lowercase letter
  - Multi-word variable names have words separated with '_'

```
my_variable = 10
square_area = 45.9
```

Magic Numbers

- A magic number is an unexplained numeric value that appears in a program's code. Example:

```
amount = balance * 0.069;
```

- What is the value 0.069? An interest rate? A fee percentage? Only the person who wrote the code knows for sure.

The Problem with Magic Numbers

- It can be difficult to determine the purpose of the number.

- If the magic number is used in multiple places in the program, it can take a lot of effort to change the number in each location, should the need arise.

- You take the risk of making a mistake each time you type the magic number in the program's code.
  - For example, suppose you intend to type 0.069, but you accidentally type .0069. This mistake will cause mathematical errors that can be difficult to find.

Symbolic Constants

- You should use symbolic constants instead of magic numbers.
- A symbolic constant is a name that represents a value that does not change during the program's execution.
- Example:
  `#define INTEREST_RATE 0.069`
- This creates a symbolic constant named `INTEREST_RATE`, assigned the value 0.069.
  It can be used instead of the magic number:

```
amount = balance * INTEREST_RATE;
```

Variables

```
double  weight;

weight = 2.1;
```

weight

2.1

type identifier;
identifier = expression;

Variables

```
double  weight = 2.1;
```

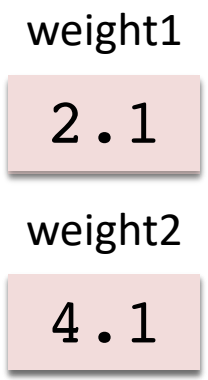
weight

2.1

type identifier = expression;

Variables

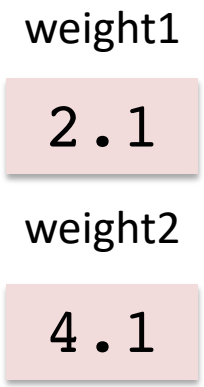
```
double  weight1, weight2;

weight1 = 2.1;
weight2 = 4.1;
```




```
type identifier1, identfier2, identifier3;
identifier1 = expression;
identifier2 = expression;
identifier3 = expression;
```

Variables

```
double  weight1 = 2.1, weight2 = 4.1;
```




```
type identifier1 = expression,
     identifier2 = expression,
     identifier3 = expression;
```

C primitive data types

type	size (bytes)	range	Example declaration + initialization	description

int
short
long
	2
	4
	4
		−32,767 to 32,767
		−2,147,483,647 to 2,147,483,647
		−2,147,483,647 to 2,147,483,647
			`int i1 = 4;`
			`int i2 = -3;`
			`short s = 11;`
			`long l = -89;`
				integer type:
				a negative or positive whole number

unsigned int
unsigned short
unsigned long
	2
	4
	4
		0 to 65,535
		0 to 4,294,967,295
		0 to 4,294,967,295
			`unsigned int ui = 4;`
			`unsigned short us = 11;`
			`unsigned long ul = 0;`
				Unsigned integer type:
				a non-negative whole number

float
double
long double
	4
	8
	12
			`float f = 11.79;`
			`double i1 = 4.23;`
			`double i2 = -3.0;`
			`long ld = -89.25;`
				floating point type:
				a negative or positive floating point number

char	1		`char c1 = 'a';`
			`char c2 = '1';`
			`char c3 = '$';`
				Character type:
				A single character wrapped in single quotes