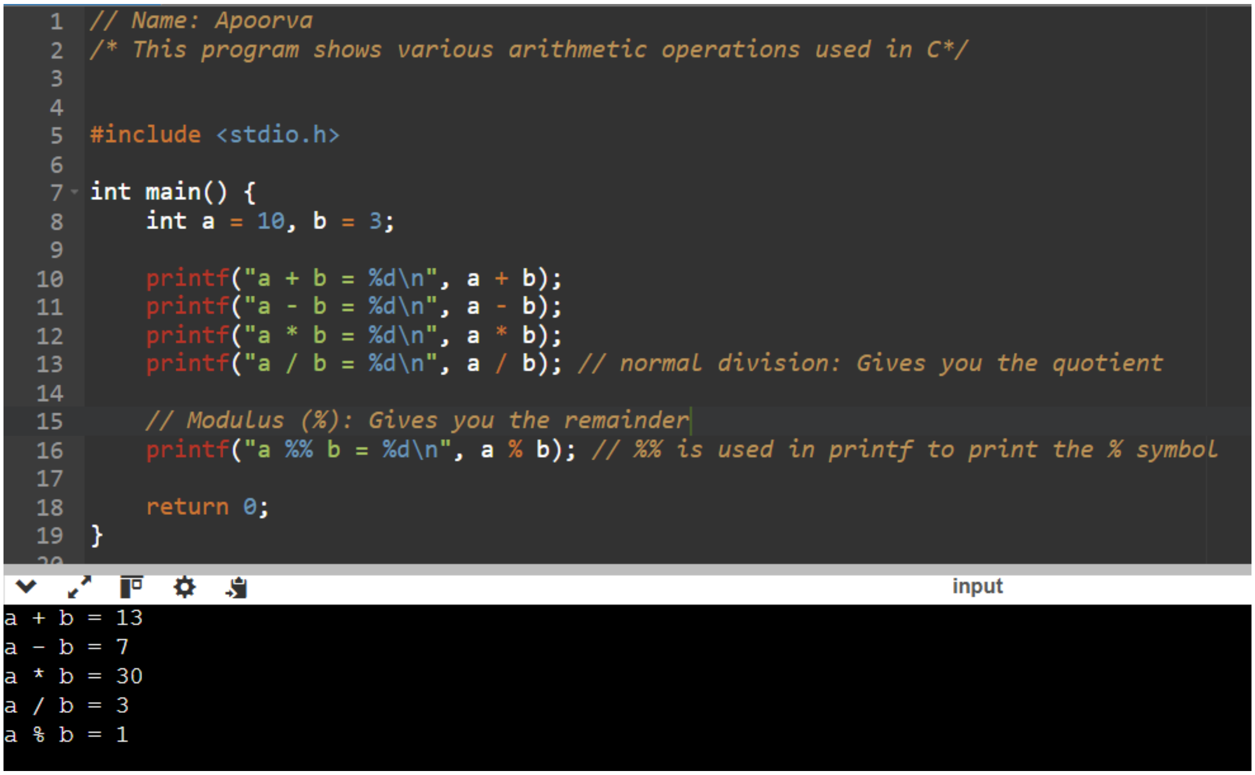
Intro to C (Lab 2: Jan 29th): Arithmetic Operations and Integer Division

Arithmetic Operators in C

C has the basic arithmetic operators:

- Addition (+)
  Example: a+b
- Subtraction (-)
  Example: a-b
- Multiplication (*)
  Example: a*b
- Division (/)
  Example: a/b
- Modulus (%): Gives you the remainder
  Example: a%b



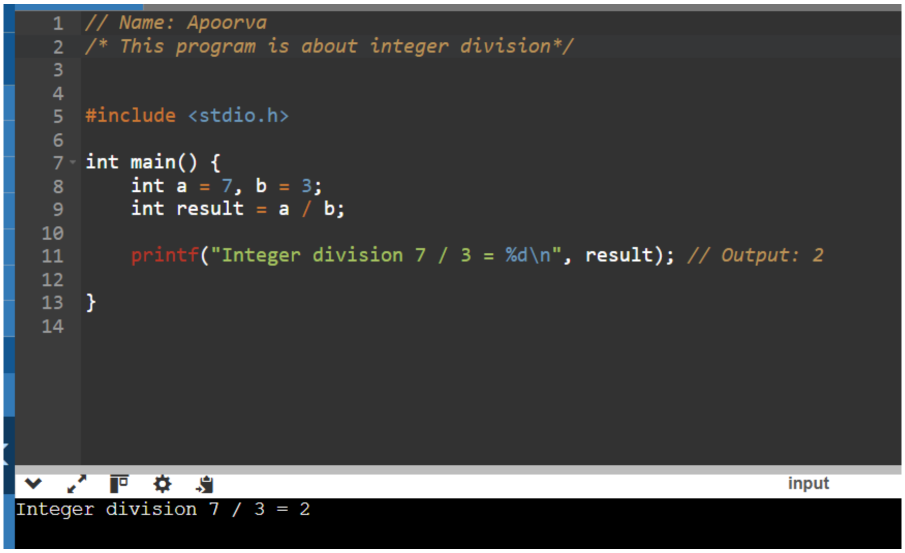
```c
// Name: Apoorva
/* This program shows various arithmetic operations used in C*/


#include <stdio.h>

int main() {
    int a = 10, b = 3;

    printf("a + b = %d\n", a + b);
    printf("a - b = %d\n", a - b);
    printf("a * b = %d\n", a * b);
    printf("a / b = %d\n", a / b); // normal division: Gives you the quotient

    // Modulus (%): Gives you the remainder
    printf("a %% b = %d\n", a % b); // %% is used in printf to print the % symbol

    return 0;
}
```

```
a + b = 13
a - b = 7
a * b = 30
a / b = 3
a % b = 1
```

Integer Division vs Float Division

Integer Division

- Happens **when both numbers are integers** (int type).



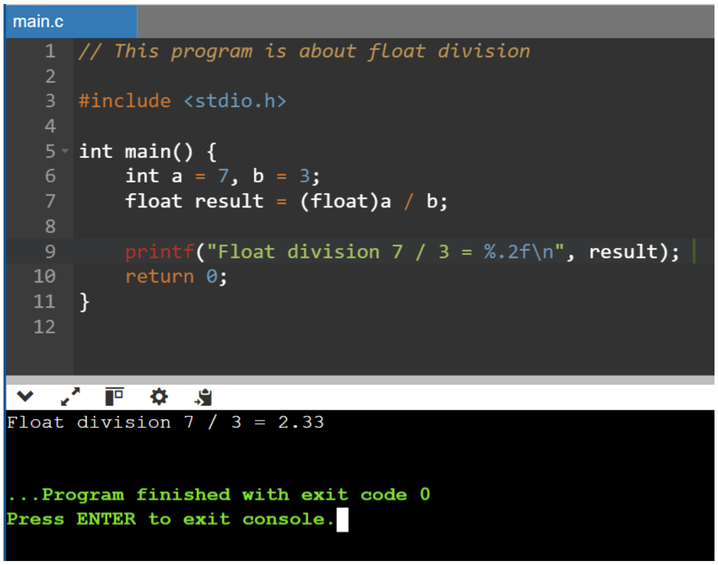
```c
// Name: Apoorva
/* This program is about integer division*/


#include <stdio.h>

int main() {
    int a = 7, b = 3;
    int result = a / b;

    printf("Integer division 7 / 3 = %d\n", result); // Output: 2

}
```

```
Integer division 7 / 3 = 2
```

Note: 7 divided by 3 is **2.333**, but integer division **drops the .333** and gives **2**.

Float Division

- Happens **when at least one operand is a float or double**.
- Gives the **exact decimal value**.

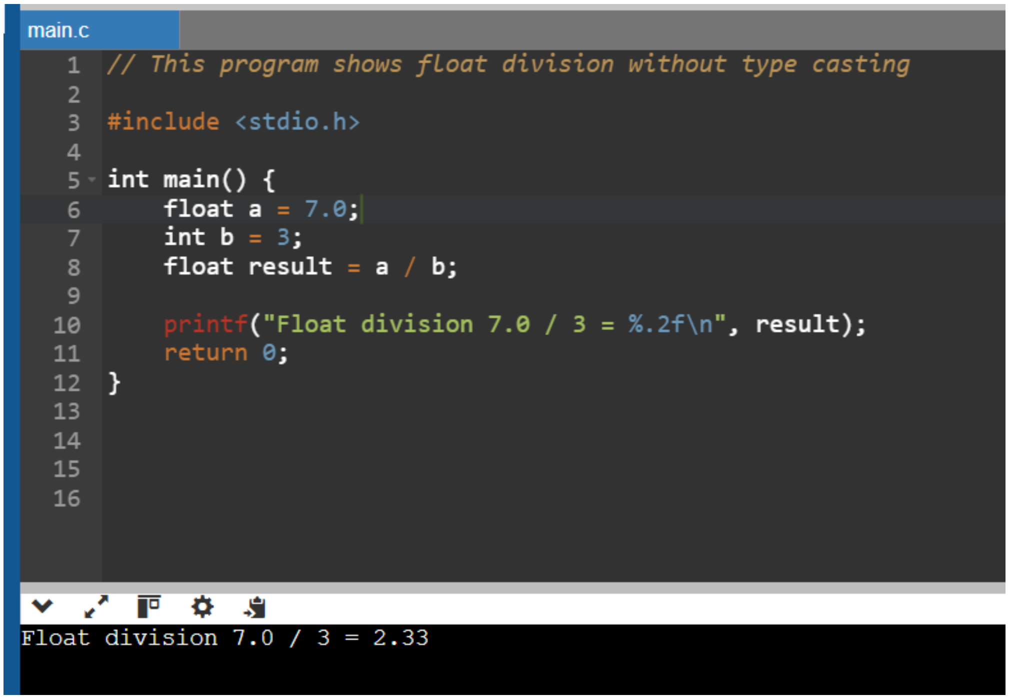


```c
// This program is about float division

#include <stdio.h>

int main() {
    int a = 7, b = 3;
    float result = (float)a / b;

    printf("Float division 7 / 3 = %.2f\n", result);
    return 0;
}
```

```
Float division 7 / 3 = 2.33


...Program finished with exit code 0
Press ENTER to exit console.
```

You **must cast** one number to float or double if both are integers; otherwise, C does integer division first.

Below is the code that uses float division **without type casting**:



```c
// This program shows float division without type casting

#include <stdio.h>

int main() {
    float a = 7.0;
    int b = 3;
    float result = a / b;

    printf("Float division 7.0 / 3 = %.2f\n", result);
    return 0;
}
```

```
Float division 7.0 / 3 = 2.33
```

Summary – Integer vs Float Division

- **Integer Division: int/int**
  Example: 7/3=2

- **Float Division:**
- **float/int**
  7.0/3=2.33

- **int/float**
  7/3.0=2.33

- **float/float**
  7.0/3.0=2.33

Type Casting:

Type casting is **super important** when teaching arithmetic in C because it directly affects division, operations between different types, and sometimes prevents unexpected results.

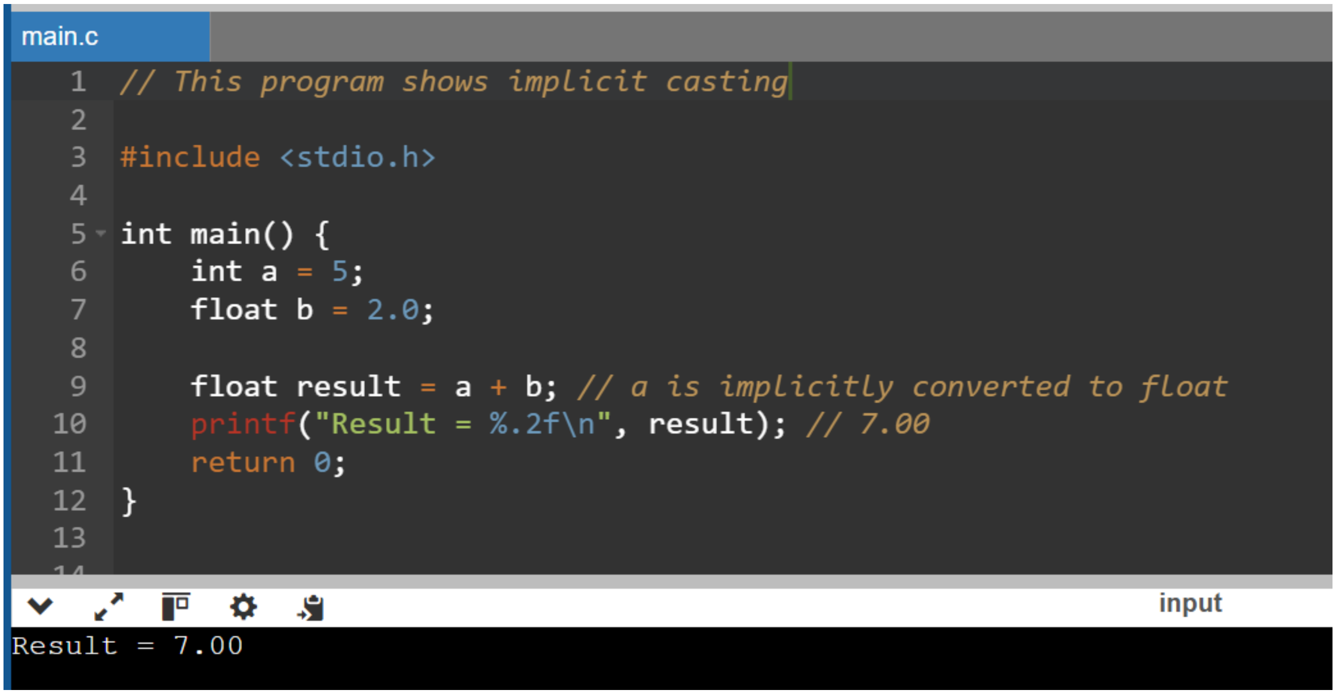
What is Type Casting?

Type casting is **converting a variable from one data type to another**.

- In C, there are **two types of type casting**:
    1. **Implicit Casting** – Done automatically by C.
    2. **Explicit Casting** – Done manually by the programmer.

Implicit Casting:

C automatically converts smaller types to bigger types when needed.



```c
// This program shows implicit casting

#include <stdio.h>

int main() {
    int a = 5;
    float b = 2.0;

    float result = a + b; // a is implicitly converted to float
    printf("Result = %.2f\n", result); // 7.00
    return 0;
}
```

```
Result = 7.00
```

Here, **a is an int** and **b is a float**. C automatically converts a to the larger data type, which is float, before performing the addition.

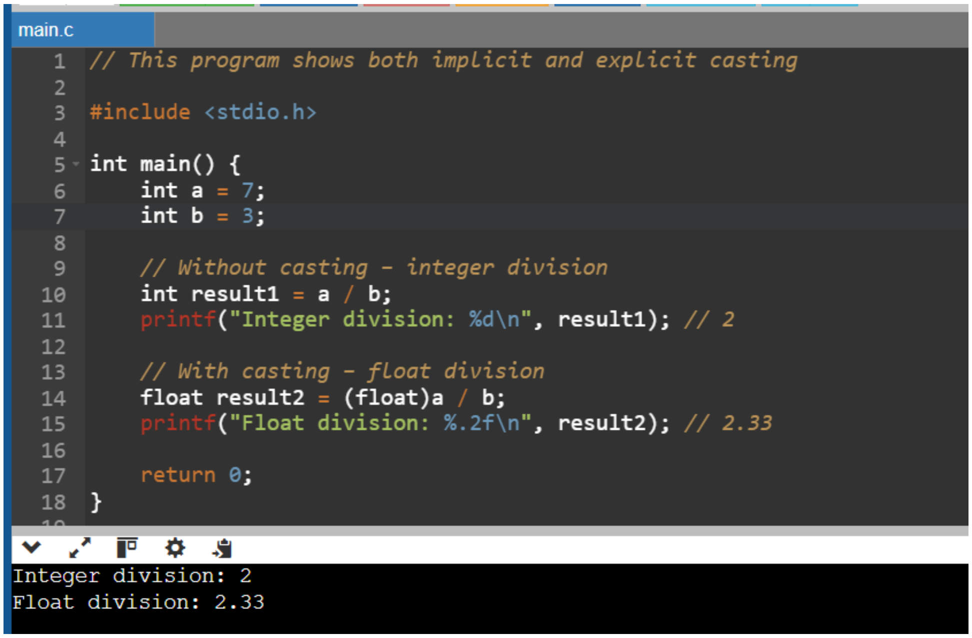
You can think of it like in math: when you add a whole number to a decimal, you treat the whole number as having .0 so the addition works correctly.

Explicit Type Casting:

You manually tell C to **treat a variable as another type** using parentheses (type). Explicit Type Casting is otherwise called “Manual Conversion”.

Syntax: (type) expression

Example: (float) a / b



```c
// This program shows both implicit and explicit casting

#include <stdio.h>

int main() {
    int a = 7;
    int b = 3;

    // Without casting - integer division
    int result1 = a / b;
    printf("Integer division: %d\n", result1); // 2

    // With casting - float division
    float result2 = (float)a / b;
    printf("Float division: %.2f\n", result2); // 2.33

    return 0;
}
```

```
Integer division: 2
Float division: 2.33
```

Casting Summary table:

- To cast or convert an **int to a float** in C, you can **use (float)**. This ensures that the division is performed as floating-point division rather than integer division.
  Example:
  float result = (float)7 / 3;

The result is 2.333333

- To cast or convert an **float to a int** in C, you can **use (int)**. This operation **truncates the decimal part**, keeping only the integer portion.
  Example:
  int result = (int) 7.89;
  Here, the result is 7

Note that the result is not 8 even though it is close to 8. When you cast a float to an int in C using (int), it **does not round**—it **truncates**. Truncation means it simply **removes everything after the decimal point**.

- ➢ To cast or convert a **char to an int** in C, you can **use (int)**. This converts the character to its corresponding **ASCII integer value**.
  Example:
  char ch = 'A';
  int ascii_value = (int)ch;

  The character 'A' has an ASCII value of 65. Casting it to int gives this numeric representation. Unlike float-to-int casting, there is **no decimal truncation involved** here—casting a char to int simply gives the ASCII code.

  What is ASCII?

- The full form of **ASCII** is **American Standard Code for Information Interchange**.
- The **ASCII value** of a character is the **numeric code assigned to that character** in the ASCII table.

- ➢ To cast or convert an int to a double in C, you can use (double). This allows the integer to be represented with **higher precision and decimal values**.
  Example:
  int a = 7;
  double result = (double)a / 3;
  The result is 2.333333

Points to remember:

- Casting to double is useful when you need **more precise decimal calculations** than float provides.
- Without casting, integer division truncates the decimal part.

===

Common Pitfalls Students Make

1. Forgetting to cast before division:

```
int a=7, b=3;

float result = a / b;
```

This is wrong because a / b is **integer division**, so the result is 2 (decimal truncated) before it is assigned to result.

Solution: Always **cast at least one operand to float or double before division** to get a decimal result.

```
float result = (float)a / b;  // result = 2.333333
```

2. Casting after division:

```
float result = (float)(a / b);
```

This is wrong because the division a / b happens **first** as integer division, giving 2. Casting it to float **afterward** only converts 2 to 2.0, it does **not recover the lost decimal**.

Solution: Always **cast before the operation**, not after, when you want a precise floating-point result.

3. Mixing too many operations without brackets:

```
int a = 7, b = 3, c = 2;

float res = (float)a + b / c;  // res = 8.0, not 8.5
```

- This is wrong because b / c is integer division → 3 / 2 = 1. Then (float)a + 1 = 8.0.
- The division result is truncated **before** adding to a.
- Use brackets to **control the order of operations**, especially when mixing int and float.

Correct way: float res = (float)a + (float)b / c; // res = 8.5

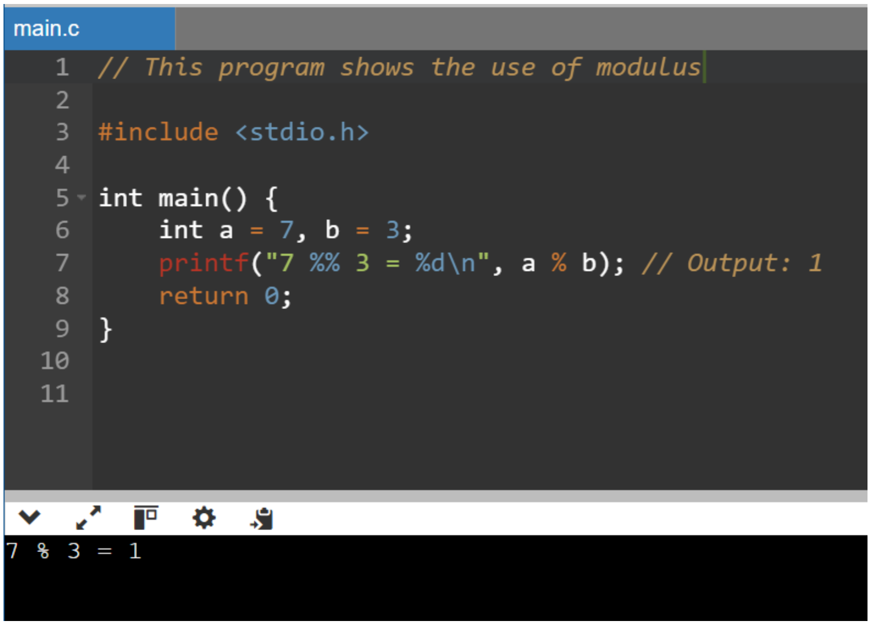
Tips to remember:

- Cast **before** the operation, not after.
- Integer division truncates decimal; float division keeps it.
- Use brackets generously when mixing integers and floats.
- (float) and (double) are your friends for precision.

==

Modulus Operator %

- Gives **remainder** after division.
- Only works with **integers**.



```c
// This program shows the use of modulus

#include <stdio.h>

int main() {
    int a = 7, b = 3;
    printf("7 %% 3 = %d\n", a % b); // Output: 1
    return 0;
}
```

```
7 % 3 = 1
```

Modulus Operator % – Common Mistakes

1. Confusing % with division

- **Mistake:** Students sometimes expect % to give the quotient.
- **Tip:** % gives **remainder**, / gives **quotient**.

2. Ignoring operator precedence

- **Mistake:** % and * have same precedence (left-to-right).
- **Tip:** Use parentheses.

3. Using negative numbers without knowing behavior

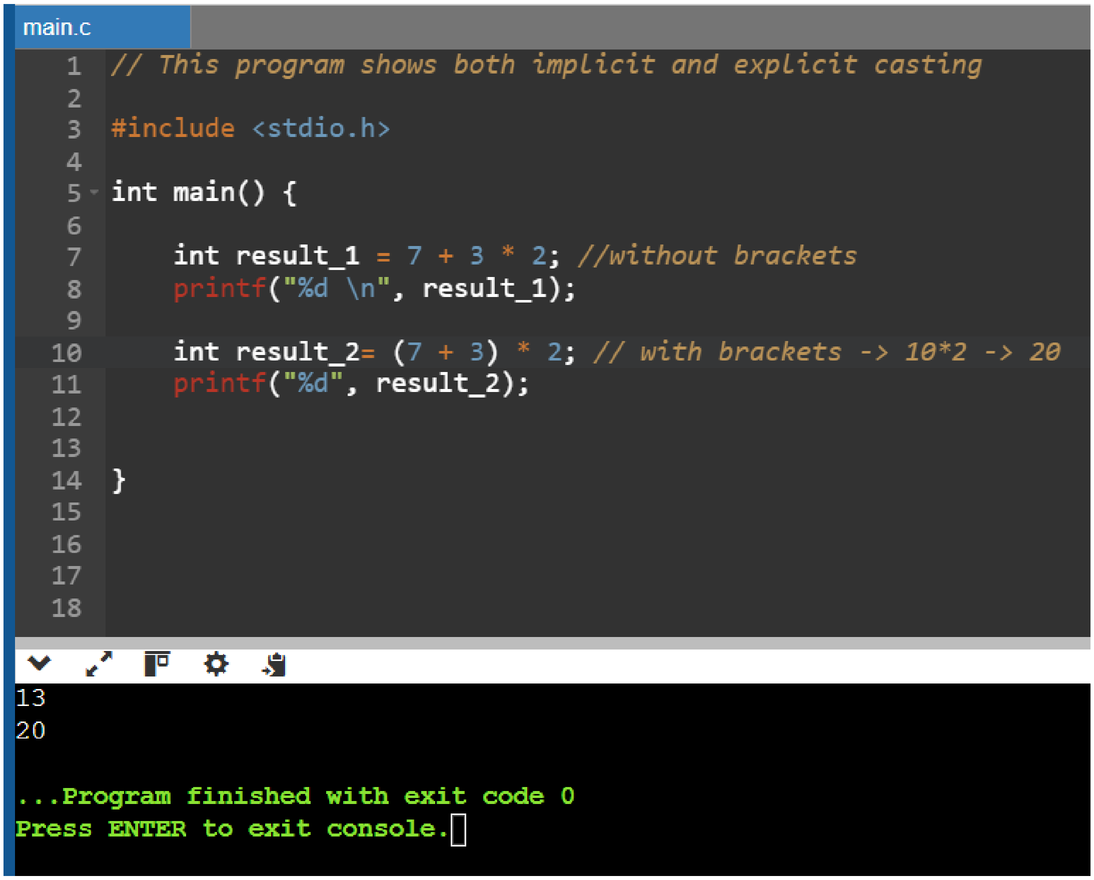
- **Mistake:** % result can be negative if the numerator is negative.
- **Tip:** Remember the remainder **takes the sign of the numerator** in C.

Operator Precedence and Brackets (VERY IMPORTANT)

C follows a specific order of operations, also called operator precedence:

- Parentheses () – Highest priority, expressions inside parentheses are evaluated first.
- Multiplication *, Division /, Modulus % – Evaluated next, left to right.
- Addition +, Subtraction - – Evaluated last, left to right

Why brackets are important:
When an expression contains multiple operations, the default precedence may not match what you expect, so using parentheses ensures the correct order.



```c
// This program shows both implicit and explicit casting

#include <stdio.h>

int main() {

    int result_1 = 7 + 3 * 2; //without brackets
    printf("%d \n", result_1);

    int result_2= (7 + 3) * 2; // with brackets -> 10*2 -> 20
    printf("%d", result_2);


}
```

```
13
20

...Program finished with exit code 0
Press ENTER to exit console.
```

Common Pitfalls:

1. Assuming left-to-right always works

```
int a = 2 + 3 * 4;
```

- Here, the answer for a would be 14, not 20. This is because the multiplication happens before addition.
- Tip: **Use parentheses** if you want addition first.

```
int a = (2 + 3) * 4;
```

Now, we get our desired output which is 20.

2. Mixing integer and float operations without brackets

```
int a = 7, b = 3, c = 2;

float res = (float)a + b / c;
```

- Here, the res is 8.0, not 8.5. b / c is integer division → decimal lost.
- Tip: Cast before division and use brackets.

```
float res = (float)a + (float)b / c;  // now the res is 8.5
```

3. Overlooking modulus % precedence

```
int a = 10, b = 3, c = 2;

int result = a % b * c;  // Here the result = 2, not 4
```

- Pitfall: % and * have the same precedence and are left-to-right, so 10 % 3 = 1, then 1 * 2 = 2.
- Tip: Use parentheses if needed

Correct way:

```
int result = a % (b * c);  // Now the result is 10 % 6 = 4
```

4. **Chaining many operations without parentheses**

Chaining many operations without parentheses can give unexpected results because C follows operator precedence. Use parentheses to control the order and make expressions clear and correct.

Example:

```
int result = 2 + 3 * 4 / 2 - 1;  // result = 7, not what you may expect
```

- Pitfall: Hard to read and easy to miscalculate.
- Tip: Break complex expressions using parentheses to clarify the order.

- Correct way: int result = 2 + (3 * 4) / (2 - 1); // result = 14

Tips to Remember

- **Parentheses first:** Always use () to force the order you want.
- **Know operator precedence:** () > * / % > + -
- **Type matters:** Casting floats vs ints affects the outcome.
- **When in doubt, use brackets!** It’s safer and makes code readable.