Executable Statements

COP 3223C – Introduction to Programming with C

Fall 2025

Yancy Vance Paredes, PhD

Terminology /1

An **expression** is a piece of code that evaluates to a value

It can be a simple **literal** 10 5 'a'

It can be a **variable** x y

It can be a **combination** 10 + x

Terminology /2

A **statement** is a complete unit of execution and must be terminated by a `;` (semicolon)

May contain an expression but not all statements are expressions

Notes

- Expressions are something that computes a value (i.e., produces)

- Statements are something the program does (i.e., performs)

Assignment Statements /1

- Store **a single value** or **a result** to a variable
- The assignment operator is the **=** (as in the equal sign)
- Unlike in math, the **=** symbol is **not symmetric!**
- Operates from right to left; look at the **right-hand side** (RHS) first!

Assignment Statements /2

- Syntax is:

```
variable_name = expression;
```

- The expression can either be a value or a computation

- Example:

```
int num;
num = 10;
```

Assignment Statements /3

```
int num1, num2;
int result;
num1 = 10;
num2 = 20;
result = num1 + num2;
```

10 20 30

num1 = 10

num2 = 20

result = 30

Assignment Statements /4

```
int num1, num2;
int result;
num1 = 10;
num2 = 20;
result = num1 + num2 + 50;
```

result = 80

Assignment Statements /5

```
int num1, num2;
int result;
num1 = 10;
num2 = 20;
result = num1 + num2 + 50;
result = result + 10;
```

result += 1;

result = 80

result = 90

80 + 10

90

Assignment Statements /6

```
int num1, num2;
int result;
num1 = 10;
num2 = 20;
result = num1 + num2 + 50;
result = result + 10;
result = result + 10;
```

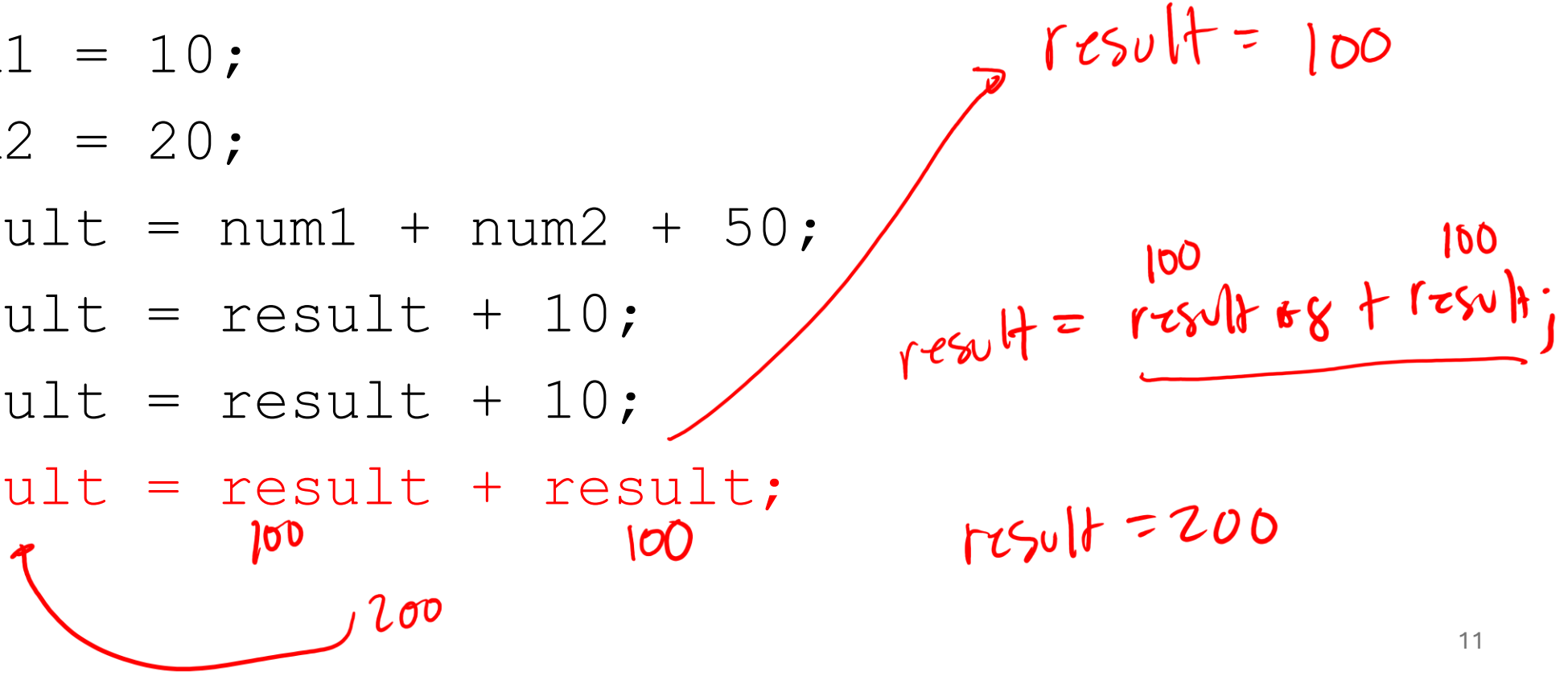
result = 90

90 + 10 = 100

result = 100

Assignment Statements /7

```
int num1, num2;
int result;
num1 = 10;
num2 = 20;
result = num1 + num2 + 50;
result = result + 10;
result = result + 10;
result = result + result;
```



Notes /1

- When you read a code, do so from top to bottom

- For **assignment statements**, look at the **right-hand side first**, then assign to the left-hand side

Notes /2

- A good technique when tracing a code manually is to not rely on a compiler right away
- You can simply keep track of the different variables on the side of your worksheet to determine their current values
- Do not forget the lifecycle of these variables

Your Turn!

```
#include <stdio.h>

int main(void)
{
        int sum = 0;
        int num;
        num = 5;
        sum = sum + num;
        sum = sum + num;
        sum = sum + num;

        return 0;
}
```

What is the final value of `sum`?

Input/Output (I/O) Operations

- **Input operation**
  - Instructions that copy data from input device (e.g., keyboard) into memory
- **Output operation**
  - Instructions that display information stored in memory

Input/Output Functions in C

- Functions that enable I/O operations are defined in the `stdio.h` library

- Output

  `printf()`

printf(" Hello World");

- Input

  `scanf()`

Task

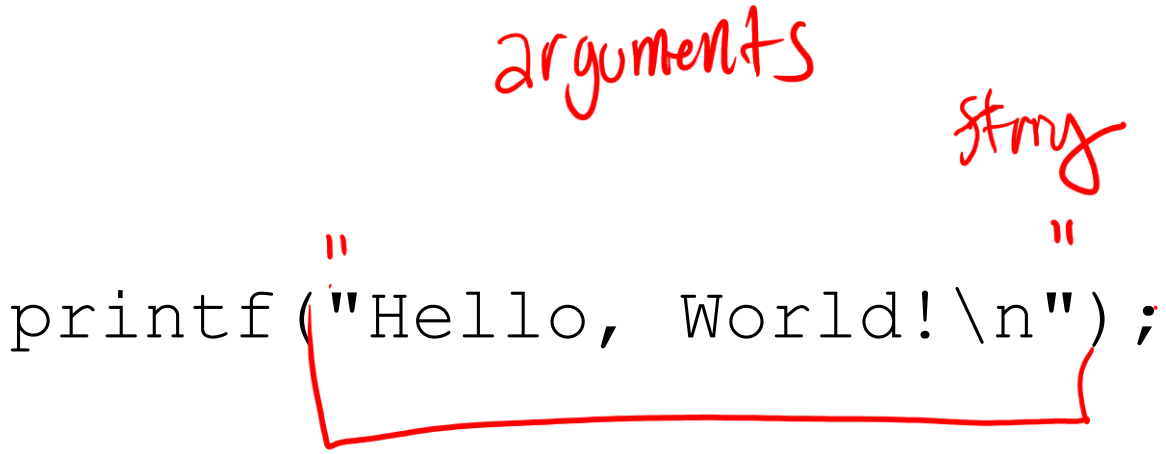
Suppose you want to print the value being held by a particular variable. How do we do that?

For example:


```
int num = 10;
```

How do we print the value assigned to the variable `num`?

The `printf()` Anatomy /1



Discussion

To help you remember, try the following in your code editor so you can clearly see that this is not the correct way to solve the task. Why do you think so?

```
int num = 10;


printf("num");
```

The `printf()` Anatomy /2

```
int num = 10;
```

%d

```
printf("The value of num is: %d\n", num);
```

format string

Discussion

- Do I really need to have a `\n` in every `printf()`?
- Well, it depends. Your experience and the task will guide you
- Try tracing the code in the next slide

Code Tracing

```c
#include <stdio.h>

int main(void) {
    int num = 10;
    float area = 5.575;

    printf("%d", num);
    printf("%f", area);

    return 0;
}
```

Notes

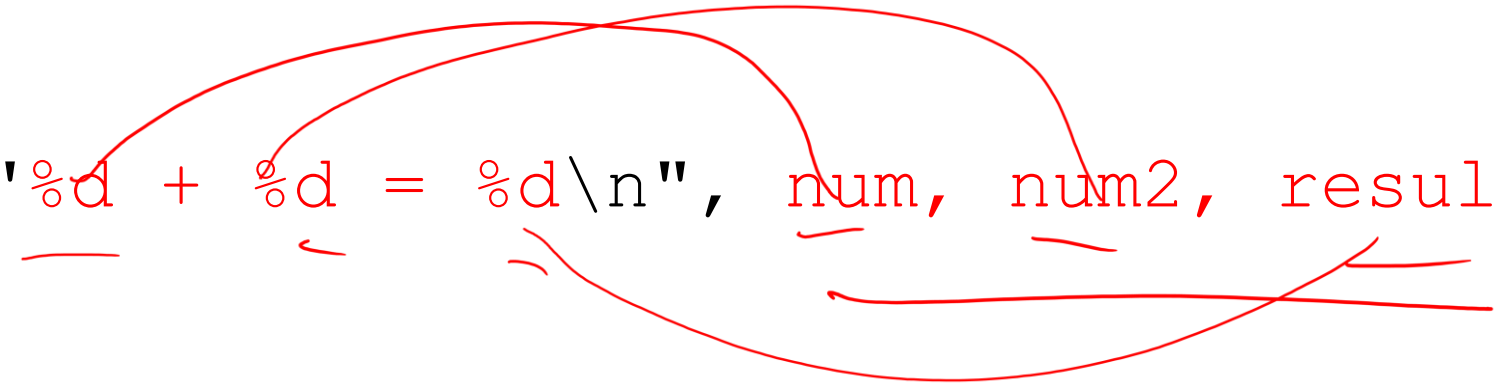
- It turns out that the newline character `\n` is not a requirement if you want to print the value of a variable

- However, adding it ensures that the value being printed is accurate especially if another `printf()` function follows it

The `printf()` Anatomy /3

```
int num = 10;
int num2 = 20;
int result = num + num2;

printf("%d + %d = %d\n", num, num2, result);
```



The `printf()` Anatomy /4

```
int num = 10;
int num2 = 20;
int result = num + num2;

printf("%d + %d = %d\n", num, num2, num+num2);
```

Placeholders or Format Specifiers

Type	Description	Placeholder
`int`	integer (whole numbers)	`%d`
`double`	number with fractional part (real numbers)	`%lf` *small L*
`float`	similar to double but smaller memory requirement	`%f`
`char`	a single character (ASCII)	`%c`

Practice

What if you want to print the value of a floating-point variable? For example:

```
float area = 5.575;
```

How do we limit the number of digits to the right of the decimal point? Say, none? How about 2 digits only?

Discussion

- Notice what happens when rounding the number `5.575`:
  - With no digits after the decimal (`%.0f`), it becomes `6`
  - With two digits after the decimal (`%.2f`), it becomes `5.57`

- At first, this may look inconsistent. However, C follows the standard rounding rules

- The issue is primarily due to how floating-point numbers are internally represented (i.e., cannot be stored exactly in binary)

Escape Sequence Character

Escape Sequence	Meaning
`\a`	Alert
`\b`	Backspace
`\n`	Newline
`\t`	Horizontal Tab
`\v`	Vertical Tab
`\\`	Backslash
`\'`	Single Quote
`\"`	Double Quote
`\?`	Question Mark
`%%`	Percent Symbol

Practice

Write a program that prints out the following:

```
20% of 100 is 20
```

```
printf("20%% of  100 is  20");
```

Discussion

What is the output of the following code fragment?

```
int num;

printf("%d\n", num);
```

Notes

In C, if you declare a local variable but do not explicitly initialize it, its value is **indeterminate**

The memory location allocated may already contain leftover bits from previous data

Therefore, the value appears random and unpredictable, so we use the common phrase **garbage value**

Interlude

If you’re coming from a Python or Java background, getting **user input** in C is a bit different

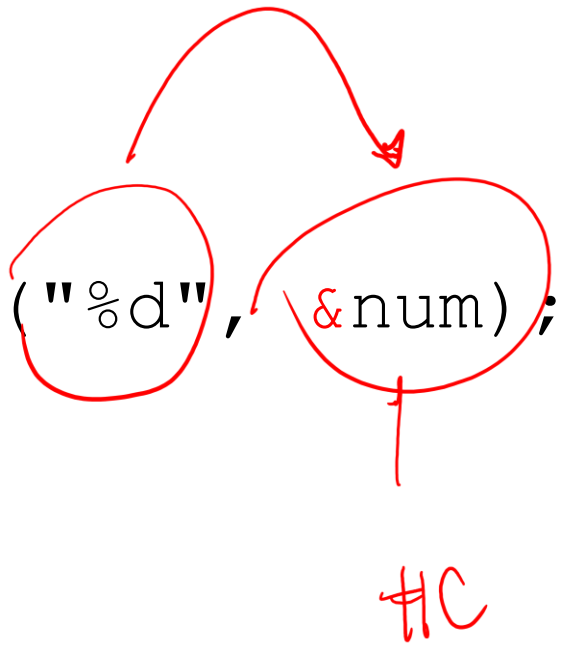
Practice

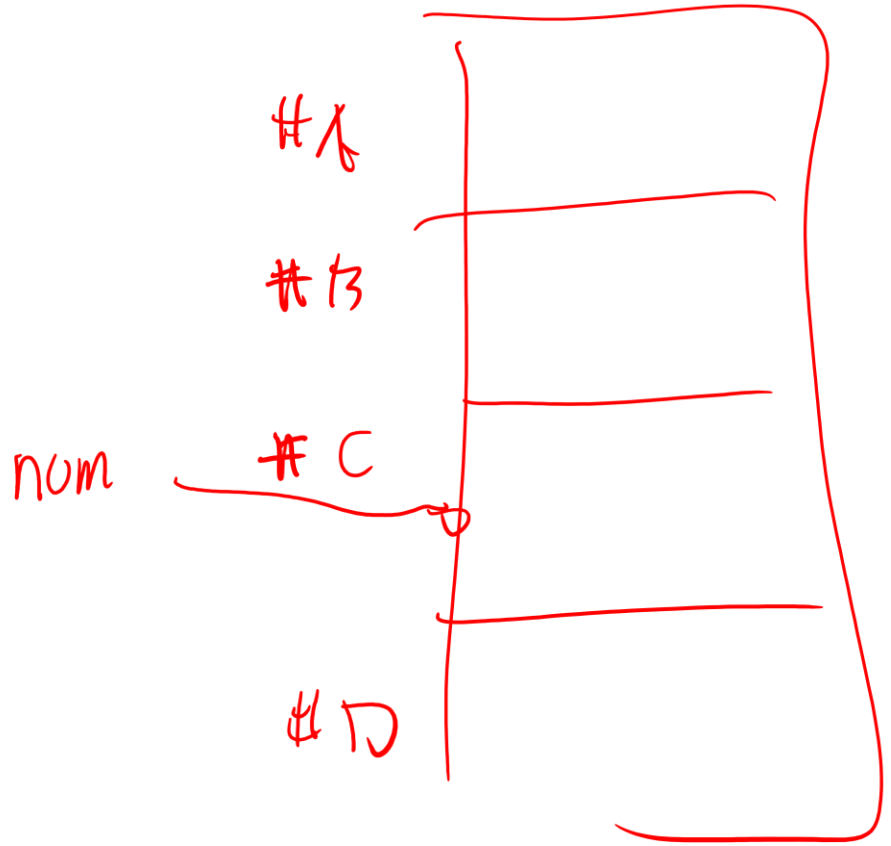
Read a whole number (integer) and store it at the address of variable **`num`**.

The `scanf()` Anatomy /1

```
int num;

scanf("%d", &num);
```





The `scanf()` Anatomy /2

```
int num;
```

```
scanf("%d", &num);
```

We use the **&** (**address-of operator**) to indicate where (i.e., the location in the memory) to store the data we received

The `scanf()` Anatomy /3

```
int num;



scanf("%d", &num);


printf("The value of num is: %d\n", num);
```

The `scanf()` Anatomy /4

```
int num;


printf("Enter a number: ");
scanf("%d", &num);


printf("The value of num is: %d\n", num);
```

The `scanf()` Anatomy /5

```
int num;
int num2;

printf("Enter two numbers: ");
scanf("%d", &num);
scanf("%d", &num2);
```

The `scanf()` Anatomy /6

```
int num;
int num2;

printf("Enter two numbers: ");
scanf("%d", &num);
scanf("%d", &num2);

printf("%d + %d = %d\n", num, num2, num+num2);
```

The `scanf()` Anatomy /7

```
int num;
int num2;

printf("Enter two numbers: ");
scanf("%d%d", &num, &num2);


printf("%d + %d = %d\n", num, num2, num+num2);
```

Your Turn!

10

20

```
int num;
int num2;

printf("Enter two numbers: ");
scanf("%d%d", &num2, &num);

printf("%d + %d = %d\n", num, num2, num+num2);
```

Notes on `scanf()`

- The **placeholder** indicates the data type to expect (and storage size requirement)
- The **memory location** indicates the address in the memory where to store the value
- The **order** of the addresses matter (i.e., arguments of `scanf()`)!

Common Mistakes (Run-Time Errors)

Using the wrong placeholder

Example:

```
int num;
scanf("%f", &num);
```

Forgetting to include the **&** operator

Example:

```
scanf("%d", num);
```

The `return` Statement /1

Terminates the function and **transfers control back** to the activator (i.e., who called this function)

In the case of the `main()` function, control is transferred back to the **operating system**

Example:

```
return 0;
```

The `return` Statement /2

Can sometimes be used where `;` immediately follows the keyword

Example:

```
return;
```

Your Turn!

Write a program that prompts the user to input three numbers. After collecting the inputs, display the numbers in reverse order. Refer to the sample run in the next slide. Anything that is underlined represents user input.

Sample Run

```
10 20 30
30 20 10
```

```
int num1, num2, num3;
scanf("%d %d %d", &num1, &num2, &num3);
printf("%d %d %d", num3, num2, num1);
```

Practice

Write a program that asks the user to enter a single letter from the English alphabet. Then, display the ASCII value of that letter.

chr

Sample Run

```
Enter Letter: B
The ASCII value of B is 66
```

```
char alpha;
printf("Enter Letter:");
scanf("%c", &alpha);
printf("The ASCII value of %c is %d", alpha, alpha);
```

Your Turn!

Write a program that prompts the user to input three characters. After collecting the inputs, display the characters in reverse order.

Sample Run

A

B

C

C B A

Potential Workaround

```c
#include <stdio.h>

int main(void) {
    char alpha, alpha2, alpha3;

    //printf("Enter Letter: ");
    scanf("%c", &alpha);

    //printf("Enter Letter: ");
    scanf(" %c", &alpha2);

    //printf("Enter Letter: ");
    scanf(" %c", &alpha3);

    printf("%c %c %c\n", alpha, alpha2, alpha3);

    return 0;
}
```

Questions?