Conditional Statements

COP 3223C – Introduction to Programming with C

Fall 2025

Yancy Vance Paredes, PhD

Control Structures /1

A construct that determines the **flow of execution** of a program

Control Structures /2

Sequential Structure

Execute instructions in the order they appear

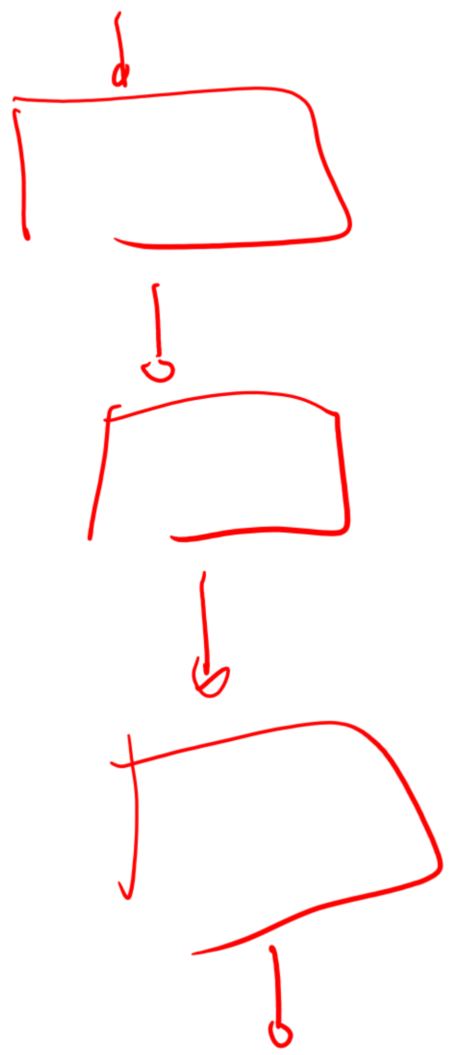
Selection Structure

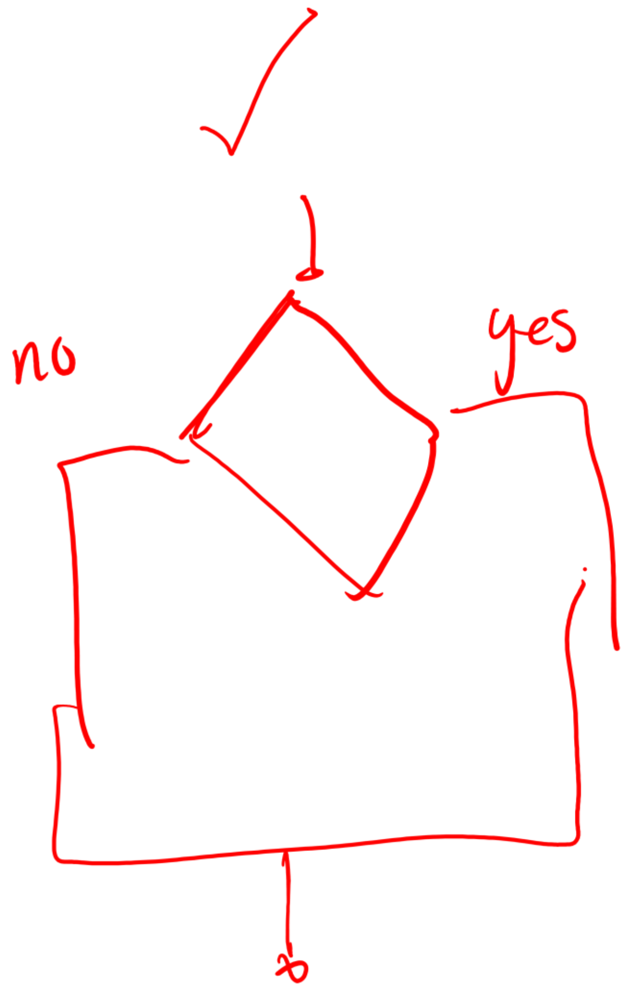
Choose different execution paths based on conditions

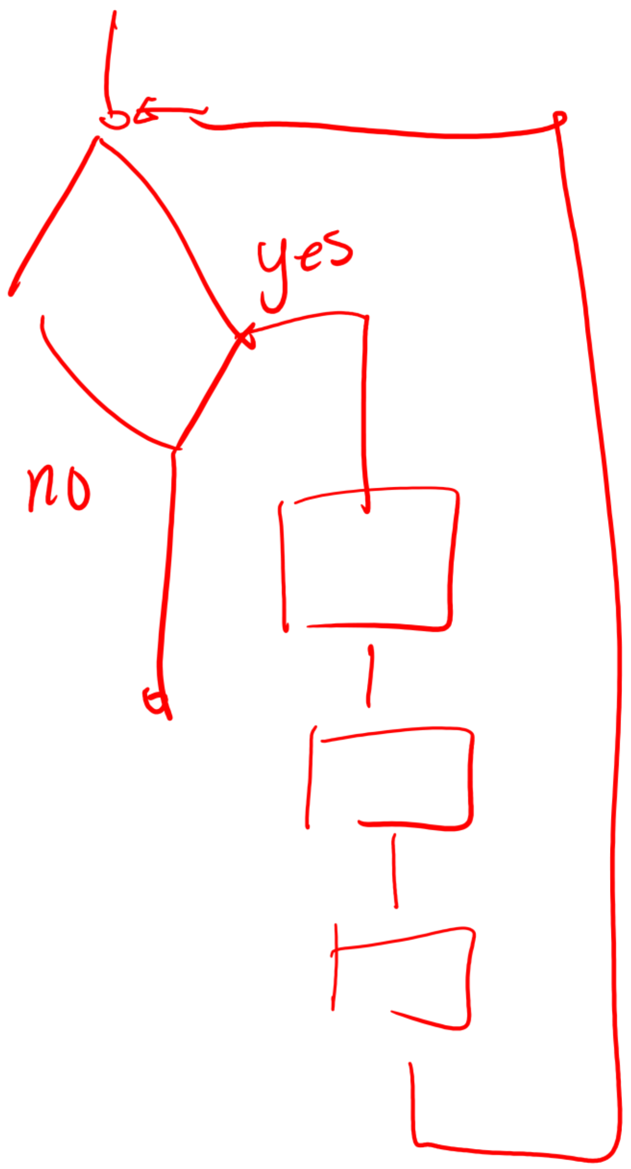
Repetition Structure

Repeat a set of instruction while a condition is met

Visualization







Making Decisions /1

- You are essentially **asking a question**
- In programming, this **should be answerable** by “yes” or “no”
- Strictly, either `true` or `false`
- This is what we often refer to as **boolean values**

Making Decisions /2

- Another way to look at it is that it is either: **1** (one) or **0** (zero)

- Also, **on** or **off**

- Ultimately, your questions should **only have two possible answers**!*

Making Decisions in C

#define true 1

- There is a header file **`stdbool.h`** that provides a type **`bool`** and the values **`true`** and **`false`** (beginning C99)

1 0

- These values are macros that expands to **`1`** and **`0`**, respectively

- C treats **zero** as false and **non-zero** as true; *prone to logic error*

Discussion

What is the difference between a **single statement** and a **compound statement**?

;

```
{
    ;        block
    ;
}
```

Compound Statements

- This is the equivalent of the **indentation** in Python
- In C, indentation does not really matter because it is a whitespace
- So, in C, we use `{ }` (the curly braces)
- We should still use indentation!

Notes

- It is often good practice to use { } even if you only have a **single statement**

- Less likely to commit a **logic error**

- **Flowcharts** can be helpful to visualize your logic

The `if` statement /1

- The questions we formulated are formally known as **condition**
- Essentially, we do something if the ***condition is met***

The **`if`** statement /2

In C, the following is the syntax of the **`if`** statement:

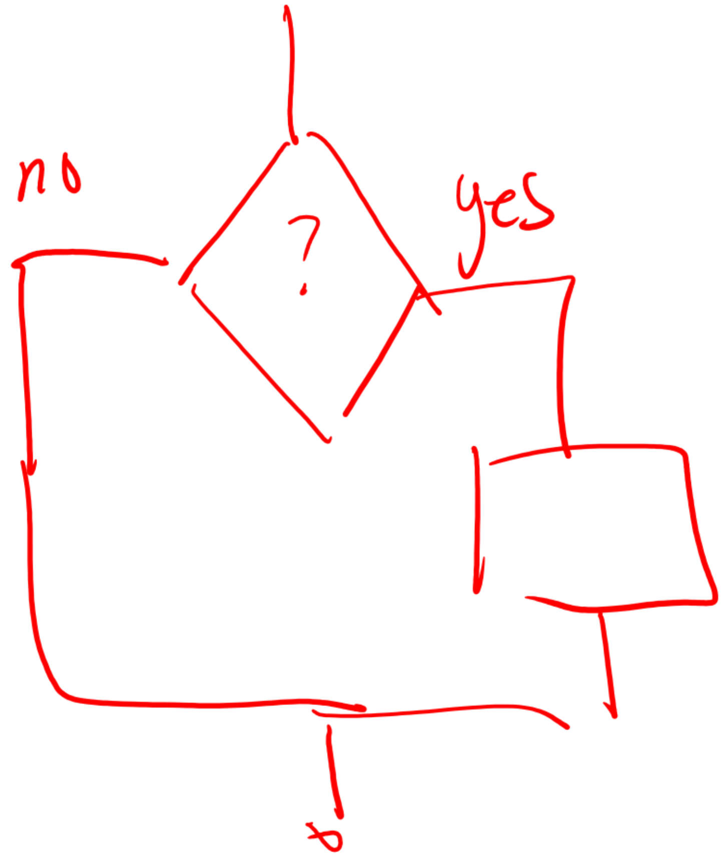
```
if ( condition )
    single statement ;

if ( condition )  single statement;
```

```
if ( condition ) {
    statement 1;
    statement 2;
}
```

$\Big]$ compound statement "block"

Notes

- In this case, we only do something **if the condition is met**
- Otherwise, we don't do anything!
- Can you visualize?



Practice

Write a program that asks the user for a whole number called `num`. If `num` is **even**, the program should inform the user that the number is even. Otherwise, the program should not perform any action.

Sample Runs

```
Enter Number: 4
Even
```

```
Enter Number: 5
```

More on the `if` statement

Say we want to perform **multiple statements** if the condition is met?

Practice

Write a program that asks the user for a whole number called `num`. If `num` is **even**, the program should inform the user that the number is even and display the number that comes immediately after it. If `num` is not even, the program should not perform any action.

Sample Runs

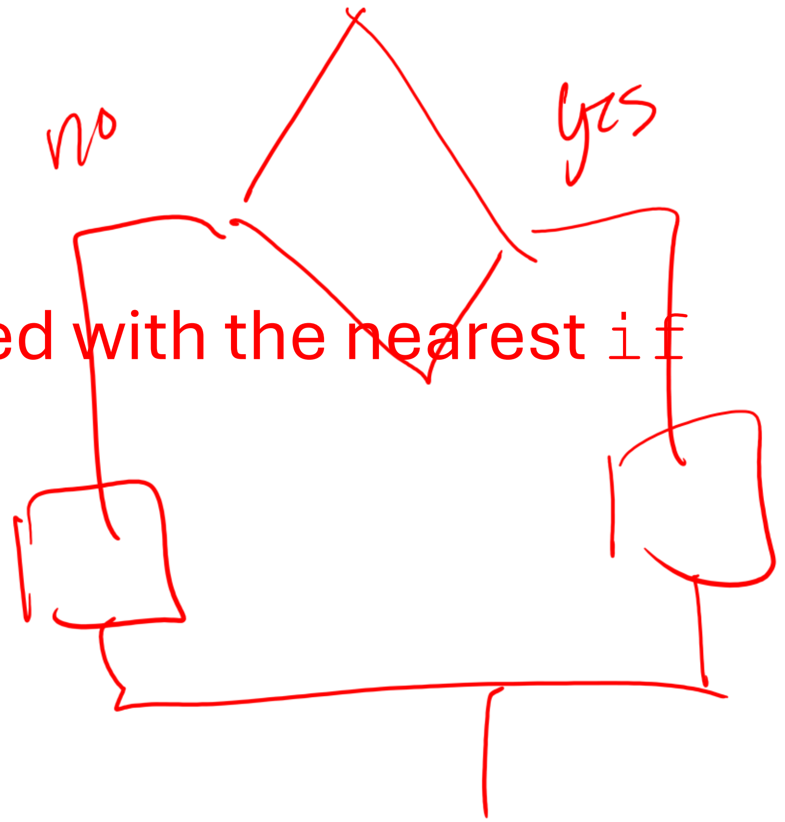
```
Enter Number: 4
Even
5
```

```
Enter Number: 5
```

The `else` statement

- What if we want to perform something if the condition **is not met**
- Must be paired with an `if` statement
- Unless indicated otherwise using `{ }`, paired with the nearest `if`

dangling else problem



Practice

Write a program that asks the user for a whole number called `num`. The program should inform the user if `num` is even or odd.

Sample Runs

```
Enter Number: 4
Even
```

```
Enter Number: 5
Odd
```

Your Turn!

Write a program that asks the user to enter two whole numbers, `num1` and `num2`. It then informs the user which of the two is greater.

Sample Runs

```
Enter Numbers: 4 5
5
```

```
Enter Numbers: 2 1
2
```

Discussion

- In the previous example, how many **possible scenarios** are there?

- Can we simply have an `else` statement?

Multiple Alternatives /1

- Previously, we saw a version of one alternative

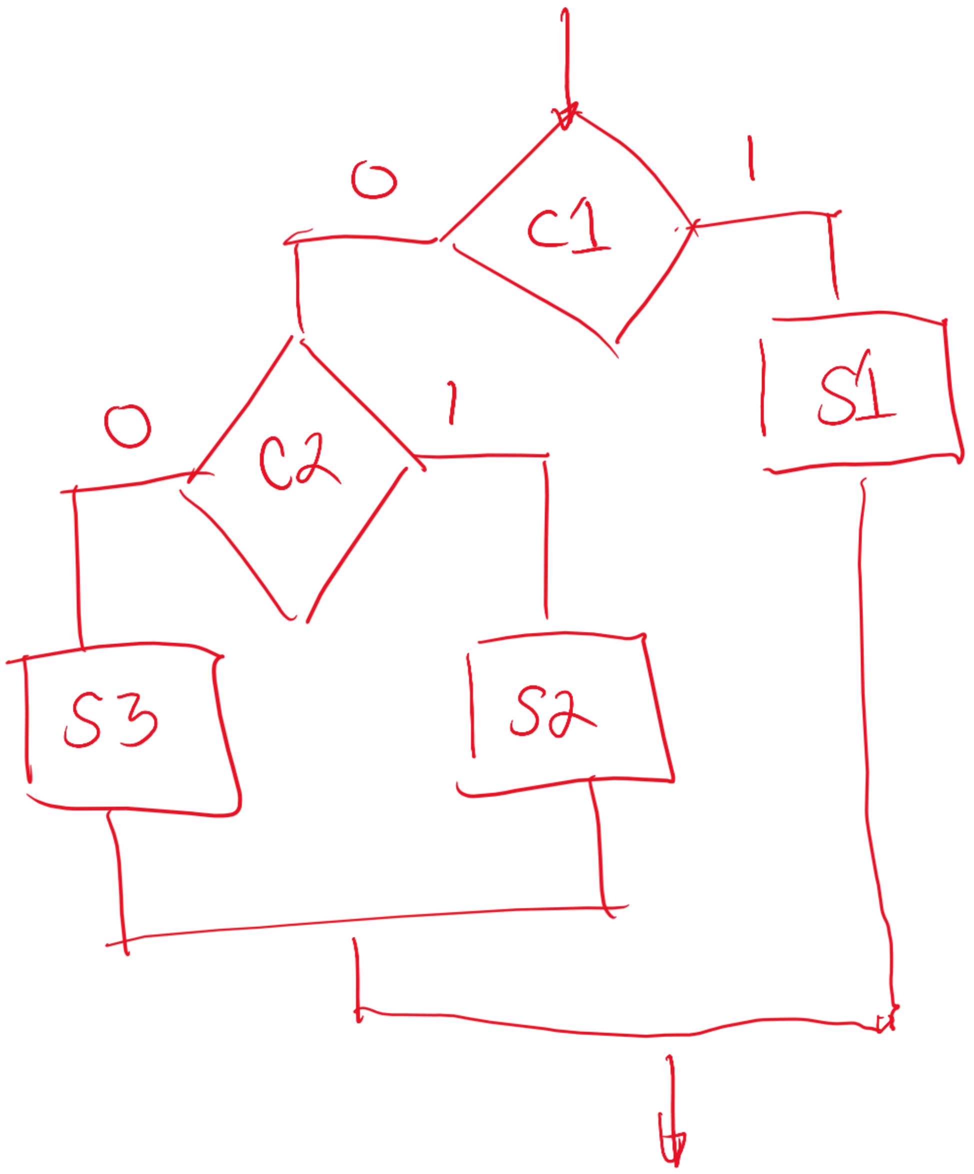
- We also saw two alternatives

- What if we have **more than two** alternatives?

Multiple Alternatives /2

Use the { } to avoid ambiguity

```
if (condition1) {
      statement1;
}
else if (condition2) {
      statement2;
}
else {
      statement3;
}
```



Notes

- If written properly, a selection structure with multiple alternatives will only follow and **perform one path**
- Refer to the visualization in the previous slide

Practice

Write a program that asks the user to enter a whole number `num`. It then informs the user if `num` is **positive**, **negative**, or **zero**.

Discussion

- Do we need to have an `if` statement for the case when `num` is `0`?
- Think: We have 3 mutually exclusive scenarios
- If we already covered the 2 scenarios (positive and negative), we can safely assume that the `else` part will cover the 3rd scenario (zero)

Code Tracing

```c
#include <stdio.h>

int main(void) {
    double res = 0.1 + 0.1 + 0.1 + 0.1 + 0.1 + 0.1 + 0.1 + 0.1 + 0.1 + 0.1;

    if( res == 1.0 )
        printf("A");
    else
        printf("B");


    return 0;
}
```

Notice how I don't have curly braces for my `if` and `else` statements. Is this an issue?

Discussion

- Checking for **equality** involving **real numbers** is generally not safe

- For example, check if a number is equal to `2.5`?

```
    num == 2.5
```

- How then?

Comparing Real Numbers

- Define a constant **EPSILON**, which is a very small number
- Used as **tolerance** to indicate if two real numbers are “equal”
- For example, **`0.000000001`** or **`1E-9`**
- You can use **`abs()`** from **`stdlib.h`** to get the absolute distance

Code Tracing

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

int main(void) {
    double res = 0.1 + 0.1 + 0.1 + 0.1 + 0.1 + 0.1 + 0.1 + 0.1 + 0.1 + 0.1;

    if( abs(res-1.0) <= 1E-9 )
        printf("A");
    else
        printf("B");


    return 0;
}
```

Practice

Write a program that accepts a single English letter from the user and outputs the same letter in the opposite case (uppercase if the input is lowercase, and lowercase if the input is uppercase).

Sample Runs

```
Enter Letter: a
A
```

```
Enter Letter: B
b
```

$\underline{A} - \underline{Z}$

$'A' <= \underset{t}{alpha} \; \boxed{\&\&} \; \underset{1}{alpha} <= '\underline{Z}'$

Practice

Write a program that asks the user to enter two English letters separated by a single space. The program should display the letter that comes first in lexicographical (alphabetical) order, regardless of case. Output the letter using the same case as it was entered by the user.

Sample Runs

```
Enter Letters: a B
a
```

```
Enter Letters: z Y
Y
```

Practice /1

Write a program that accepts a **single character** input from the user, called `choice`. Display a message based on the value of `choice`.

Value of `choice`	**Display Message**
`'A'`	`Apple`
`'B'`	`Banana`
`'C'`	`Cherry`

Practice /2

Write a program that accepts a **single character** input from the user, called `choice`. Display a message based on the value of `choice`. If it is **not a valid** option, inform the user.

Value of `choice`	**Display Message**
`'A'`	`Apple`
`'B'`	`Banana`
`'C'`	`Cherry`

Discussion

- In the previous example, we performed a series of **checking for equality**
- What if the program supports value for `choice` from `A` to `Z`?
- Multiple `if-else` statements?

The **`switch`** Statement

- A cleaner version of `if-else` if it only deals with `==` operation
- Note, it does not use relational operators for comparison
- Works only for **integral values** (`char` is included)
- Syntax is best demonstrated by an example

Practice

Write a program that accepts a **single character** input from the user, called `choice`. Display a message based on the value of `choice`. If it is **not a valid** option, inform the user.

Value of `choice`	**Display Message**
`'A'`	`Apple`
`'B'`	`Banana`
`'C'`	`Cherry`

```c
char choice;
printf("Enter Letter: ");
scanf("%c", &choice);
switch (choice) {
      case 'A': printf("Apple");
                break;
      case 'B' : printf("Banana");
                break;
      case 'C' : printf("Cherry");
                break;
      default:   printf("Invalid");
                break;
}
```

Value of choice	Display Message
'A'	Apple
'B'	Banana
'C'	Cherry

Practice

Write a program that accepts an **integer** input from the user, called `num`. Display a message based on the value of `num`. If it is **not a valid** option, inform the user.

Value of `num`	**Display Message**
`1`	`Apple`
`2`	`Banana`
`3`	`Cherry`

```
int num;
printf("Enter Number:  ");
scanf("%d", &num );
switch ( num ){
      case 1  :  printf("Apple");
                 break;

      case 2  :  printf("Banana");
                 break;

      case 3  :  printf("Cherry");
                 break;
      default:   printf("Invalid");
                 break;
}
```

Value of `num`	Display Message
1	Apple
2	Banana
3	Cherry

Your Turn!

Write a program that accepts a single lowercase letter (**character**), `alpha`. It then informs the user if it is a **vowel** or not.

Hint: If you use the **`switch`** statement, take advantage of it works (i.e., behavior)

Nested `if` statements

- It is possible for an `if` statement to be written in the body of another `if` statement

- Often seen in cases involving **multiple levels** of decision-making

Equivalent Logic

```
// Version 1
if( condition1 ) {
    if( condition2 ) {
        // statements if condition1 is true
        // AND condition2 is true
    }
}

// Version 2
if( condition1 && condition2 ) {
    // statements if condition1 is true
    // AND condition2 is true
}
```

Practice

Ask the user to enter a whole number `num`. Afterward, check if this `num` is a positive **even** number.

```
int num;
printf("Enter Number: ");
scanf("%d", &num);

if (num > 0 ) {
    if (num % 2 == 0 ){
        printf(" Positive Even");
    }
}
```

```
int num;
printf("Enter Number: ");
scanf("%d", &num);

if (num > 0 && num % 2 == 0 ) {
    printf(" Positive Even");
}
```

Practice

Ask the user to enter a whole number `num`. Afterward, check if this `num` is between the range of **1 to 10, inclusive**. If it is, display if `num` is an **even** or an **odd** number.

```
int num;
printf("Enter number");
scanf("%d", &num);

if ( num >=1 && num <=10 ) {
        if(num % 2 == 0 ) {
                printf("Even");
        }
        else { printf ("Odd");
        }
}
```

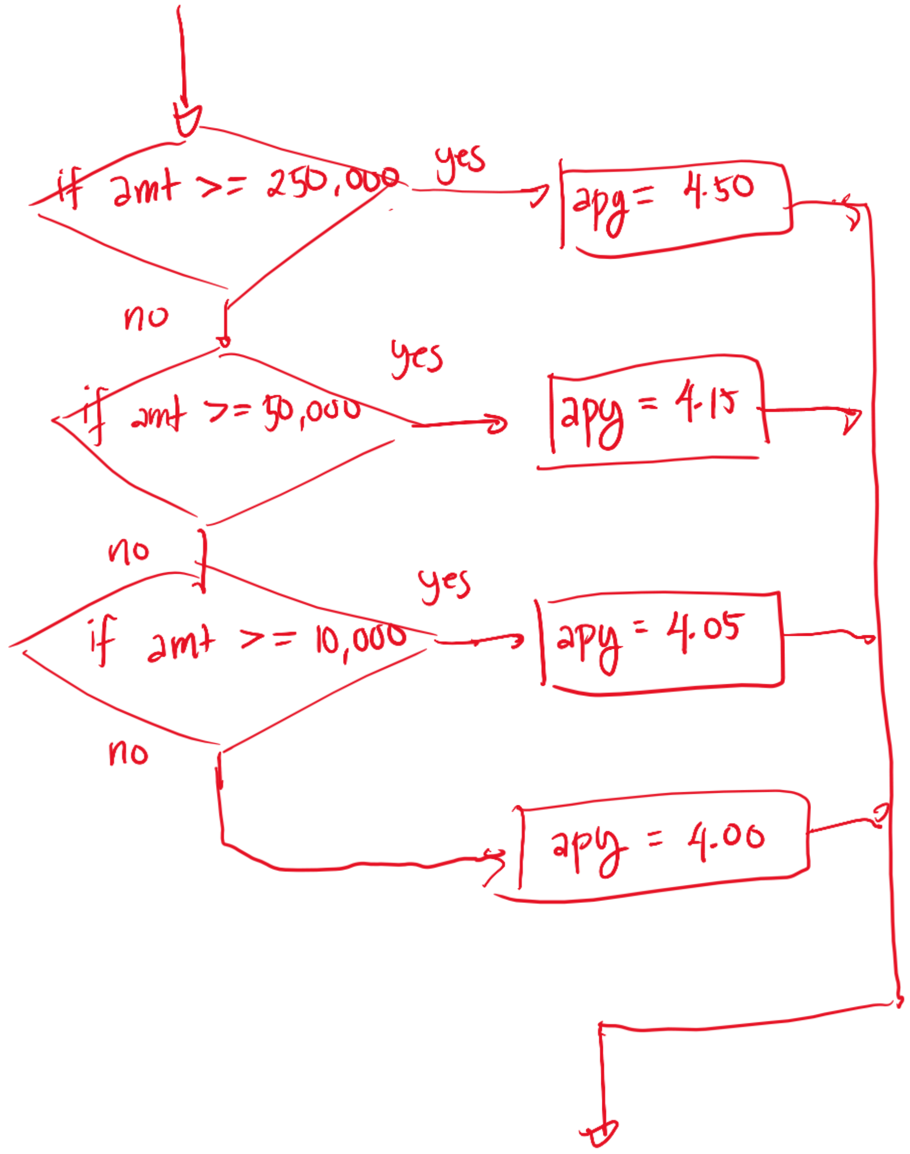
```
int num;
printf("Enter number");
scanf("%d", &num);

if ( num >=1 && num <=10  &&  num % 2 == 0  ) {
        printf("Even");
else { printf ("Odd");
}
```

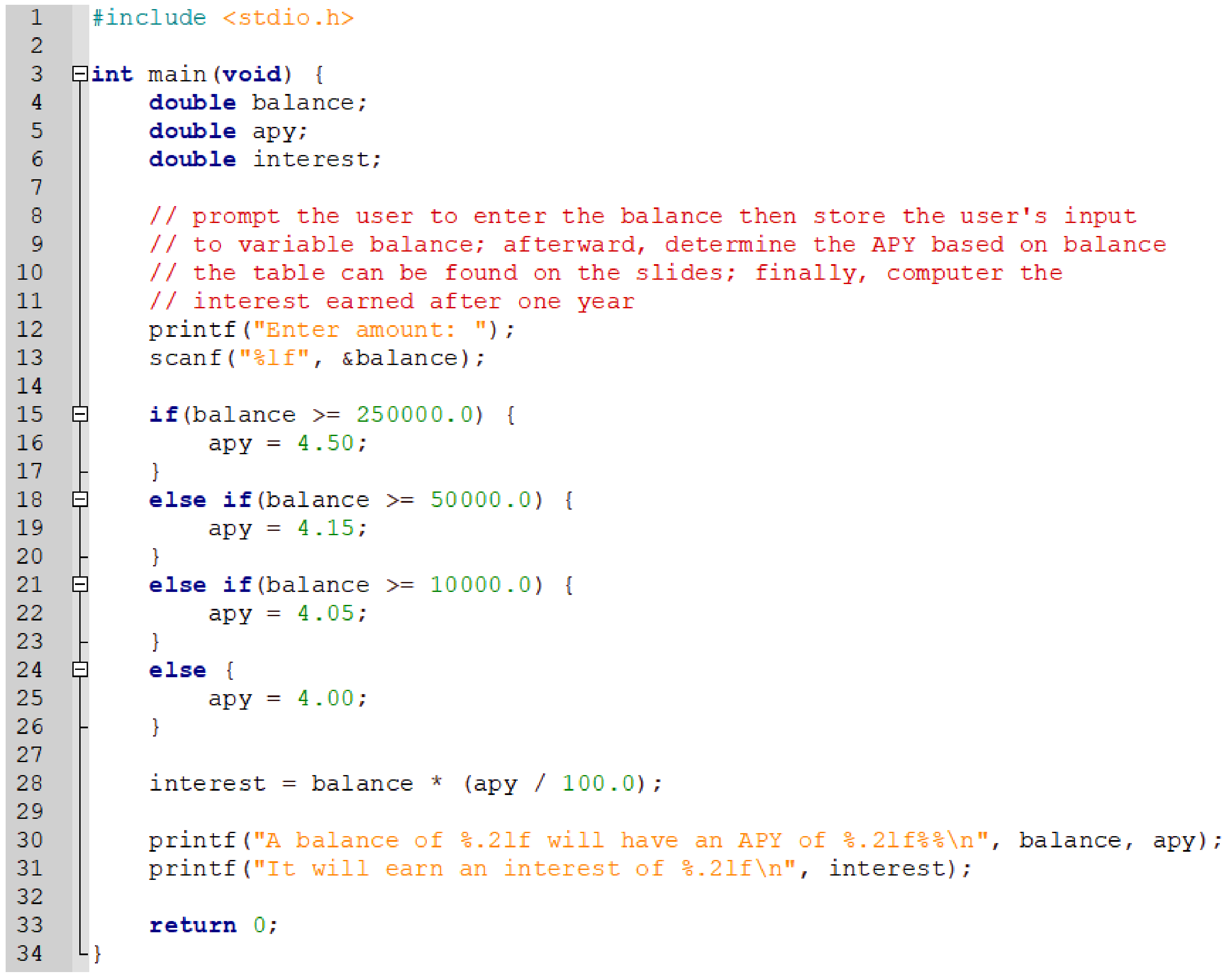
Your Turn!

`Interest = Balance * APY`

Write a program that asks the user to enter their current savings account balance. Using the simple interest formula, calculate and display how much interest the user will earn in one year.

Balance Tier	**APY**
$250,000 or greater	4.50%
$50,000 to <$250,000	4.15%
$10,000 to <$50,000	4.05%
Less than $10,000	4.00%





```c
#include <stdio.h>

int main(void) {
    double balance;
    double apy;
    double interest;

    // prompt the user to enter the balance then store the user's input
    // to variable balance; afterward, determine the APY based on balance
    // the table can be found on the slides; finally, computer the
    // interest earned after one year
    printf("Enter amount: ");
    scanf("%lf", &balance);

    if(balance >= 250000.0) {
        apy = 4.50;
    }
    else if(balance >= 50000.0) {
        apy = 4.15;
    }
    else if(balance >= 10000.0) {
        apy = 4.05;
    }
    else {
        apy = 4.00;
    }

    interest = balance * (apy / 100.0);

    printf("A balance of %.2lf will have an APY of %.2lf%%\n", balance, apy);
    printf("It will earn an interest of %.2lf\n", interest);

    return 0;
}
```

Ternary Operator

C provides a one-liner **shortcut** in writing `if-else` statements

Syntax:

condition ? expr-true : expr-false;

Practice

Write a program that prompts the user to enter a whole number. If the number is even, display the number that comes immediately after it. If the number is odd, display the number that comes immediately before it.

Questions?