Executing the same code more than once

Sometimes we need to repeat our work

C normally executes in order, line by line

if statements allow us to “turn on or off” parts of our code

But up until now, we don’t have a way to repeat code

Copy-pasting the same code again and again is not a feasible solution

While Loop

• We often need to execute code (statements) many times.

• if statements only allow us to execute or not execute code. in other words they allow us to execute code 0 or 1 times

  while statements allow us to execute code 0 or more times

• Like if, while statements have a controlling expression but while statements execute their body until the controlling expression is false

While Loops

The format is very similar to if statements

```
// expression is checked at the start of every loop
while (expression) {
    // this will run again and again
    // until the expression is evaluated as false
    // When the program reaches this }, it will jump
    // back to the start of the while loop
}
```

The “question” in the (brackets) functions very similarly

If it’s true (non-zero), the body of the while loop will run

If it’s false (zero), the body won’t run and the program will continue

Once a while reaches the end of its {} it will start again

While Statements

• C has other looping constructs - but while is all you need

• for loops can be a little more concise/convenient we’ll see them later - for now use while

• Often use a loop counter variable to count loop repetitions

• Can then have a while loop execute n times.

Format is very similar to an if statement
While Loop Control

We can use a variable to control how many times a while loop runs

We call this variable a “loop counter”

It’s an int that’s declared outside the loop

It’s “termination condition” can be checked in the while expression

It will be updated inside the loop

We can also use a variable to decide to exit a loop at any time

We call this variable a “sentinel”

It’s like an on/off switch for the loop

While Loop with a Loop Counter

// an integer outside the loop
int counter = 0;
while (counter < 10) {
    printf("We have looped %d times.\n", counter);
    counter = counter + 1;
}
// When counter hits 10 and the loop’s test fails
// the program will exit the loop

While Loop that never stops

It’s actually very easy to make a program that goes forever

Consider the following while loop:

while (1 < 2) {
    // Never going to give you up
    // Never going to let you down . . .
}

while Loop - Loop Counter Example #1

// read an integer n
// print n asterisks
int n;
printf("How many asterisks? ");
scanf("%d", &n);
int loop_counter = 0;
while (loop_counter < n) {
    printf("*");
    loop_counter = loop_counter + 1;
}
printf("\n");

source code for asterisks.c
While Statements - Termination

- Can control termination (stopping) of while loops in many ways.
- Easy to write while loop that do not terminate.
- Often a sentinel variable is used to stop a while loop when a condition occurs in the body of the loop.

```
int stop_loop, number;
printf("Enter numbers, 0 to stop\n");
stop_loop = 0;
while (stop_loop != 1) {
    scanf("%d", &number);
    if (number == 0) {
        stop_loop = 1;
    } else if (number % 2 == 1) {
        printf("%d is odd.\n", number);
    } else {
        printf("%d is even.\n", number);
    }
}
```

A loop within a loop

```
int size;
printf("Enter size: ");
scanf("%d", &size);
// print `size` lines
int row = 0;
while (row < size) {
    // print a row of `size` asterisks
    int column = 0;
    while (column < size) {
        printf("*");
        column = column + 1;
    }
    // the row is finished, start the next line
    printf("\n");
    row = row + 1;
}
```

YO DAWG, HEARD YOU LIKE LOOPS

SO I PUT A LOOP INSIDE YOUR LOOP SO YOU CAN LOOP WHILE YOU LOOP
### Drawing a grid of stars

The previous slide's code:
- Sets up a loop using $y$.
- In each loop of $y$, sets up a loop using $x$.
- The $x$ loop writes multiple *s to the terminal.
- Then the $y$ loop finishes, writing \n so the line ends.

### What do the curly braces do?

- **What goes on inside the curly braces stays inside the curly braces**
  - Look closely at the declaration of `int x` in the grid drawing code.
  - The use of $x$ is contained inside a set of curly braces `{}`.
  - This means that $x$ will only exist inside those braces.
  - The variable $x$ will actually disappear each time the $y$ loop finishes!
  - **Curly braces create the “scope” of a program**
  - Anything created inside them only lasts as long as they do!

### While Loops, If Statements - All code

- While loops, if statements etc, it’s all code!
  - An if statement is some code.
  - A while loop is also some code.

- **This means that you can:**
  - Put if s inside while loops.
  - Put while loops inside if s or else s.
  - Put while loops inside while loops inside if statements etc etc etc!

- **Just watch out for confusing ourselves!**

### Dice Statistics, a Looping Program

- The following program:
  - I need a program that will show me all the different ways to roll two dice.
  - If I pick a number, it will tell me all the ways those two dice can reach that total.
  - It will also tell me what my odds are of rolling that number.
Break it down
What components will we need?
We need all possible values of the two dice
We need all possible totals of adding them together
Seems like we’re going to be looping through all the values of one die and adding
them to all the values of the other die
Let’s start with a simple program then go for our bigger goals later

All possible dice rolls - code for inputting values

- first we’ll read the size for the two dice
- this code is very similar to previous programs

```c
int die_one_size;
int die_two_size;
// User decides the two dice sizes
printf("Please enter the size of the first die: ");
scanf("%d", &die_one_size);
printf("Please enter the size of the second die: ");
scanf("%d", &die_two_size);
// Then loop through both dice
```

source code for all_die_rolls.c

Quick Pause for new C syntax: ++
Incrementing just got a little easier

```c
int die1 = 0;
int die2 = 0;
// The following two lines have the
// same effect on their variables
die1 = die1 + 1;
die2++;
// both variables now == 1
```

source code for all_die_rolls.c

All possible dice rolls - loops

```c
// Then loop through both dice
int die1 = 1;
while (die1 <= die_one_size) { // seen die1 - 1 values
    int die2 = 1;
    while (die2 <= die_two_size) { // seen die2 - 1 values
        int total = die1 + die2;
        printf("%d , %d total: %d\n", die1, die2, total);
        die2++;
    }
    die1++;
}
```
Extending Our Program

We have all possibilities listed
We know all the totals
We could also count how many times the dice were rolled

Let's try now isolating a single target number
Check the targets of the rolls and output only if they match our target value

Now with a target number - input

```c
int main(void) {
    int die_one_size;
    int die_two_size;
    int target_value;
    // User decides the two dice sizes and target
    printf("Please enter the size of the first die: ");
    scanf("%d", &die_one_size);
    printf("Please enter the size of the second die: ");
    scanf("%d", &die_two_size);
    printf("Please enter the target value: ");
    scanf("%d", &target_value);
    // Then loop through both dice
```

Now with a target number - loop

```c
    // Then loop through both dice
    int die1 = 1;
    while (die1 <= die_one_size) { // seen die1 - 1 values
        int die2 = 1;
        while (die2 <= die_two_size) { // seen die2 - 1 values
            int total = die1 + die2;
            if (total == target_value) {
                printf("%d , %d total: %d\n", die1, die2, total);
            }
            die2++;
        }
        die1++;
    }
```

Getting there!

We now have a program that can identify the correct rolls
If we want the odds, we just compare the target rolls vs the rest
If we count the number of rolls that added to the target value
And we count the total number of rolls
We can do some basic maths and divide the successful rolls by the total
That should give us our chances of getting that number
How do we count our successful rolls?

We can count using ints
We can keep a counting variable outside the loop
This will increment only on successes
We can either calculate or count our total
Dividing them will give us the fraction chance of rolling our target number

Measuring Successes

Adding some variables to count results
integers (die_one_size, die_two_size) for the two dice sizes
integer (target_value) for the target value
integer (num_successes) for the number of successes
integer (num_rolls) for the number of rolls

Making sure our loop records results
int num_successes = 0;
int num_rolls = 0;
// Then loop through both dice
int die1 = 1;
while (die1 <= die_one_size) {
    // seen die1 - 1 values
    int die2 = 1;
    while (die2 <= die_two_size) { // seen die2 - 1 values
        num_rolls++;
        int total = die1 + die2;
        if (total == target_value) {
            num_successes++;
            printf("%d, %d total: %d\n", die1, die2, total);
        }
        die2++;
    }
    die1++;
}

Output our Percentage
int percentage = (100 * num_successes) / num_rolls;
printf("Percentage chance of getting your target number is: %d\n", percentage);

• BTW There’s a much simpler way to list the rolls that sum to a target number
• There’s also a much simpler way to find the total number of rolls
• If we just use a bit more maths and less raw coding . . .
• See what you can come up with!
Loop Example - Checking Gauss’s Formula

```c
// check gauss's formula for sum of integers 1.. n
int n;
printf("Enter n: ");
scanf("%d", &n);
int sum = 0;
int i = 1;
while (i <= n) {
    sum = sum + i;
    i = i + 1;
}
printf("Sum of integers 1..%d = %d\n", n, sum);
int gauss = ((n + 1) * n) / 2;
printf("(%d + 1) * %d / 2 = %d\n", n, n, gauss);
source code for gauss.c
```

Loop Example - Summing 42 Numbers

```c
int sum = 0;
printf("Enter \%d numbers: \n", N_NUMBERS);
int n = 0;
while (n < N_NUMBERS) {
    int x;
    scanf("%d", &x);
    sum = sum + x;
    n = n + 1;
}
printf("Sum of the numbers is \%d\n", sum);
source code for sum_42_numbers.c
```

Loop Example - Summing n Numbers

```c
int n_numbers;
printf("How many numbers do you wish to sum: ");
scanf("%d", &n_numbers);
printf("Enter \%d numbers: \n", n_numbers);
int n = 0;
int sum = 0;
while (n < n_numbers) {
    int x;
    scanf("%d", &x);
    sum = sum + x;
    n = n + 1;
}
printf("Sum of the numbers is \%d\n", sum);
source code for sum_n_numbers.c
```

Loop Example - Printing A Triangle

```c
int size;
printf("Enter size: ");
scanf("%d", &size);
// print \$size\ lines
int row = 0;
while (row < size) {
    // print a row of \row + 1\ asterisks
    int column = 0;
    while (column <= row) {
        printf("*");
        column = column + 1;
    }
    // the row is finished, start the next line
    printf("\n");
    row = row + 1;
}
source code for triangle.c
```
// loop through numbers 1..MAX
int n = 1;
while (n < MAX) {
    // loop through numbers 1..n counting factors
    int possible_factor = 1;
    int n_factors = 0;
    while (possible_factor <= n) {
        if (n % possible_factor == 0) {
            n_factors = n_factors + 1;
        }
        possible_factor = possible_factor + 1;
    }
    if (n_factors <= 2) {
        printf("%d is prime\n", n);
    }
    n = n + 1;
}