COMP1511 - Programming Fundamentals

Week 2 - Lecture 3

Recap - Relational and Logical Operators

Relational Operators

- `>, >=, <, <=, ==, !=`
- Comparisons made between numbers
- Will result in 1 for true and 0 for false

Logical Operators

- `&&, ||, !`
- Comparisons made between true and false (0 and 1) results
- Used to combine Relational Operator Questions together

Testing our Input

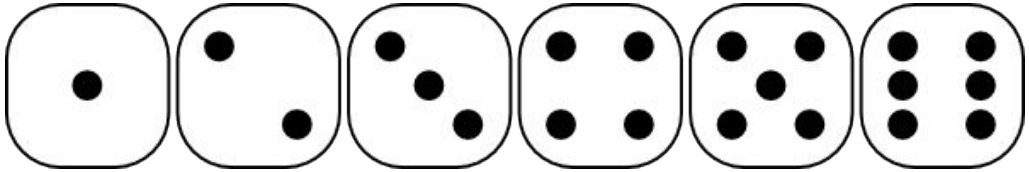
Let's assume we have this input code:

```
// Setup dice variables
int dieOne;
int dieTwo;

// we start by asking the user for their dice rolls
printf("Please enter your first die roll: ");
// then scan their input
scanf("%d", &dieOne);
// repeat for the second die
printf("Please enter your second die roll: ");
scanf("%d", &dieTwo);
```

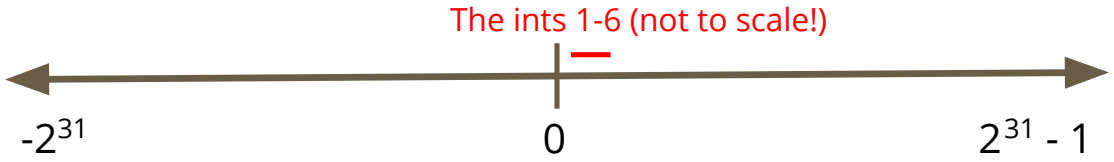
Testing Input Range

A six sided die has a specific range of inputs



We will only accept inputs in this range

But ints have a much wider range!



Testing Input Range in Code

```
// we start by asking the user for their dice rolls
printf("Please enter your first die roll: ");
// then scan their input
scanf("%d", &dieOne);

// test for proper input range
if (1 <= dieOne && dieOne <= 6) {
    // if this succeeds, we are in the right range
} else {
    // number was outside the die's range
}
```

no & means it's number

and

cd.. 上个 level

Using Constants in code

The 1 and 6 are the minimum and maximum values of the dice

- We could use something like this at the start of our program:

```
#define MIN_VALUE 1
#define MAX_VALUE 6
```

Using `MIN_VALUE` and `MAX_VALUE`:

- Makes the program much easier to modify for different dice sizes
- Also makes things much more readable by using English words instead of numbers

Reject the input

We can just end the program if the input is incorrect

```
// we start by asking the user for their dice rolls
printf("Please enter your first die roll: ");
// then scan their input
scanf("%d", &dieOne);

// test for proper input range
if (MIN_VALUE <= dieOne && dieOne <= MAX_VALUE) {
    // if this succeeds, we are in the right range
} else {
    // number was outside the die's range
    return 1;
}
```

也可以用 1. 6.

end the program

Reporting Failure

Information from the program helps the user

```
// test for proper input range
if (MIN_VALUE <= dieOne && dieOne <= MAX_VALUE) {
    // if this succeeds, we are in the right range
} else {
    // number was outside the die's range
    printf("Input for first die, %d was out of
range. Program will exit now.\n", dieOne);
    return 1;
}
```

tell user

Can we do better?

Exploring other options

Let's give the user information that helps them correct the input issues

```
// test for proper input range
if (MIN_VALUE <= dieOne && dieOne <= MAX_VALUE) {
    // if this succeeds, we are in the right range
} else {
    // number was outside the die's range
    printf("Input for first die, %d was out of the
range 1-6. Program will exit now.\n", dieOne);
    return 1;
}
```

give more information to user

Correcting the input without exiting

If we want the program to finish executing even with bad input

Imperfect, but sometimes we want the program to finish

What are our options?

夹住

- Clamping - anything outside the range gets "pushed" back into the range
- Modulus - a possibly elegant solution

简洁的

Clamping Values

Correcting the values - a brute force approach

```
// we start by asking the user for their dice rolls
printf("Please enter your first die roll: ");
// then scan their input
scanf("%d", &dieOne);

// clamp any values outside the range
if (dieOne < MIN_VALUE) {
    dieOne = MIN_VALUE;
} else if (dieOne > MAX_VALUE) {
    dieOne = MAX_VALUE;
}
```

Modulus

A reminder of what it is

- **%** - A maths operator that gives us the remainder of a division

How can we use it?

- Any number “mod” 6 will give us a value from 0 to 5
- If we change any 0 to a 6, we get the range 1 to 6
- This means the user could type in completely random numbers and be given a 1-6 dice roll result

除6 余数只能在 0-5

Using Modulus in code

```
// we start by asking the user for their dice rolls
printf("Please enter your first die roll:");
// then scan their input
scanf("%d", &dieOne);

// mod forces the result to stay within 0-5
dieOne = dieOne % MAX_VALUE;
// make any 0 into MAX_VALUE
if (dieOne == 0) {
    dieOne = MAX_VALUE;
}
```

Pros and Cons of using Modulus for dice

Pros

- We guarantee a number between 1 and 6 (or whatever the max value is)
- We don’t shut down unexpectedly due to incorrect input
- We give a very dice-like randomish result (as opposed to clamping)

Cons

- We might accept incorrect input silently
- We might make a change that affects the user's expectations

Setting up

We'll start with our description of the program

```
// The Dice Checker v2
// Marc Chee, February 2019

// Allows the user to set dice size
// Tests the rolls of two dice against a target number
// Able to deal with user reported rolls outside the range
// Will report back Success, Tie or Failure

#include <stdio.h>

#define MIN_VALUE 1
#define MAX_VALUE 6
```

Variables and Constants

Set up the Target constant and some variables

```
// The secret target number
#define SECRET_TARGET 7

int main (void) {
     int dieOne;
     int dieTwo;
```

Taking user input

Two rolls will be taken as input (only one is shown here)

```
// Process the first die roll
printf("Please enter your first die roll: ");
scanf("%d", &dieOne);
// Check and fix the die roll
if (dieOne < MIN_VALUE || dieOne > MAX_VALUE) { // dieOne is invalid
    printf("%d is not a valid roll for a D%d.\n", dieOne, MAX_VALUE);
    dieOne = (dieOne % MAX_VALUE);
    if (dieOne == 0) {
        dieOne = MAX_VALUE;
    }
}
```

Calculate and report the total

This is identical to last week's code

```
    // calculate the total and report it
    int total = dieOne + dieTwo;
    printf("Your total roll is: %d\n", total);

    // Now test against the secret number
    if (total > SECRET_TARGET) {
        // success
        printf("Skill roll succeeded!\n");
    } else {
        // failure
        printf("Skill roll failed!\n");
    }
```