COMP1511 PROGRAMMING FUNDAMENTALS

LECTURE 3

Control Flow

Getting harder...

IF Statements
Loop de loop

LAST LECTURE...

LAST WEEK, WE TALKED:

- Welcome and Introductions
- Started looking at C
- Our first Hello! program
- Compiling and running your code
- **`printf()`** and **`scanf()`**
- Variables (**`int`**, **`double`**, **`char`**)
- Maths :)

IN THIS LECTURE...

TODAY...

- IF statements
- Logical Operators
- Chaining **`if`** and **`else`**
- Loop, loop, loop, loop, loop
    - While

“

WHERE IS THE CODE?

Live lecture code can be found here:

HTTPS://CGI.CSE.UNSW.EDU.AU/~CS1511/22T3/LIVE/WEEK02/

DECISION TIME... ASKING THE COMPUTER TO MAKE A DECISION...

IF STATEMENTS

- Sometimes we want to make decisions based on what information we have at the time
- We can let our program branch between sets of instructions
- In C this is the `if` statement

WHAT KINDS OF PROBLEMS DO WE SOLVE WITH IF?

DECISION PROBLEMS (YES/NO)

- A decision problem is a question with a YES/NO answer
- This is the perfect time to use an IF statement to help make the decision
- Eg. Is a number even? Is a number larger than 10? Is a number prime? etc.

IF STATEMENT

IT IS LIKE A QUESTION AND AN ANSWER

- First we ask the question - this is our condition
- If the answer to our question (condition) is YES, then we run the code in the curly brackets

```
// The code inside the curly brackets
// runs IF the expression is TRUE (not zero)

if (condition) {
    do something;
    do something else;
}
```

WHAT IF THE ANSWER IS NO?

THERE ARE OPTIONS, THERE ARE ALWAYS OPTIONS

- If the answer to our question (condition) is NO, then we can add an else statement to let the computer know which other code may run

```
if (condition){
    // code to run if the condition is true OR
    // anything other than 0
} else {
    // run some other code instead
    // else is entered if the previous code
    // results in 0 (false)
}
```

WHAT IF THE ANSWER IS NO... AGAIN?

MORE OPTIONS

- If the answer to our question (condition) is NO, and the answer to our question (condition) in the else is also NO, then we can chain some if and else together to make an else if and create even more options in choosing which code to run...

```
if (condition_one){
    // code to run if the condition_one is true OR
    // anything other than 0
} else if (condition_two){
    // code to run if condition_one is FALSE (results in a 0)
    // AND condition_two is TRUE (anything other than 0)
} else {
    // code to run if both
    // condition_one AND condition_two are FALSE (0)
}

```

HOW DO WE ASK GOOD QUESTIONS?

RELATIONAL OPERATORS

NOTICE: IN C, WE HAVE == AND =

THESE ARE NOT THE SAME AND DO NOT MEAN WHAT YOU ARE USED TO IN MATHS!

USING = WHEN YOU ASSIGN VALUES
USING == WHEN YOU ARE CHECKING FOR EQUIVALENCE

- Relational Operators work with pairs of numbers:
  - < less than
  - > greater than
  - <= less than or equal to
  - >= greater than or equal to
  - == equals
  - != not equal to

- All of these will result in 0 if false and a 1 if true

I LIKE QUESTIONS, HOW DO I ASK TWO QUESTIONS AT THE SAME TIME?

LOGICAL OPERATORS

The first two are used between two questions (expressions):

- **&&** AND: if both expressions are true then the condition is TRUE (equates to 1 if both sides equate to 1)
- **||** OR: if any of the two expressions are true then the condition is TRUE (is 1 if either side is 1)

This is used in front of an expression:

- **!** NOT: reverse the expression (is the opposite of whatever the expression was)

SOME EXAMPLES

LET'S TRY THIS OUT...

- True (1) or False (0)?
- Let’s Kahoot!

IF / ELSE IF / ELSE

LET'S LOOK AT SOME CODE AND A DEMO

- IF statements with logical operators:
  `if_logic.c`
- IF statements with char:
  `lower.c`
- Harder IF logic and chaining if and else together:
  `dice_checker.c`

NOW LET'S CODE!

1. Switch over toVLab
2. Open Terminal
3. Open a new file:

```
code dice_checker.c
```

Feel free to follow along with lecture coding, or you can also find the code here:

https://cgi.cse.unsw.edu.au/~cs1511/22T3/live/Week02/

LET'S PUT OUR SKILLS TO THE TEST

LET'S CODE! (SOLVE THE PROBLEM FIRST)

A user rolls two dice and tell us the number on each of the rolled die. Our program will add the die numbers together and check them against a target number that only the program knows. It will then report back whether the total of the dice was higher, equal or lower than the secret number.

BREAKING DOWN THE PROBLEM INTO A SUM OF SIMPLE PARTS

A user rolls two dice and tell us the number on each of the rolled die. Our program will add the die numbers together and check them against a target number that only the program knows. It will then report back whether the total of the dice was higher, equal or lower than the secret number.

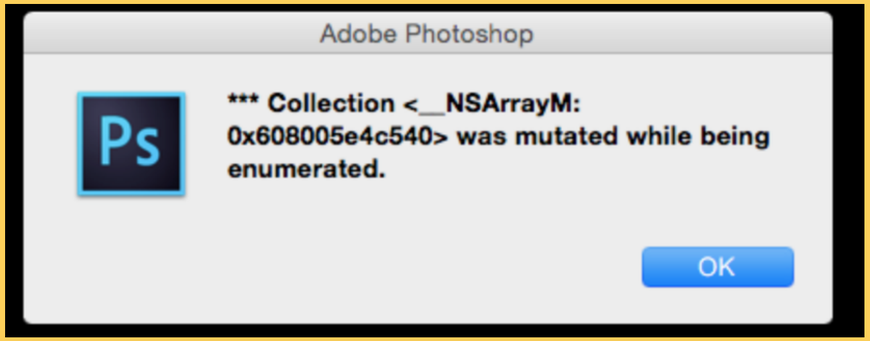
1. A user will roll two dice - done outside of our program
2. Take in the result of each die - how do we read input?
3. Add the die numbers together
4. Check them against a target number - based on steps 4 and 5, it looks like we need to make a decision - therefore IF statement
5. Output if total of the dice was higher, equal or lower than the target number - output based on the decision that we made

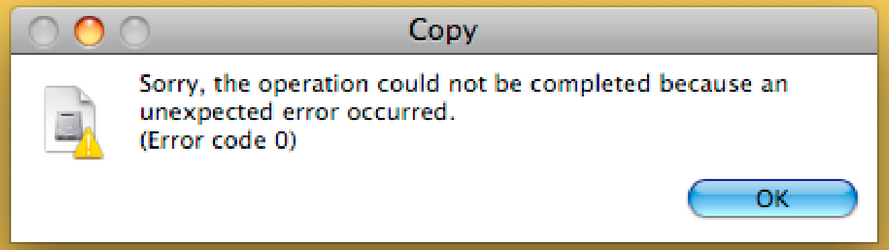
BREAKING DOWN THE PROBLEM INTO A SUM OF SIMPLE PARTS

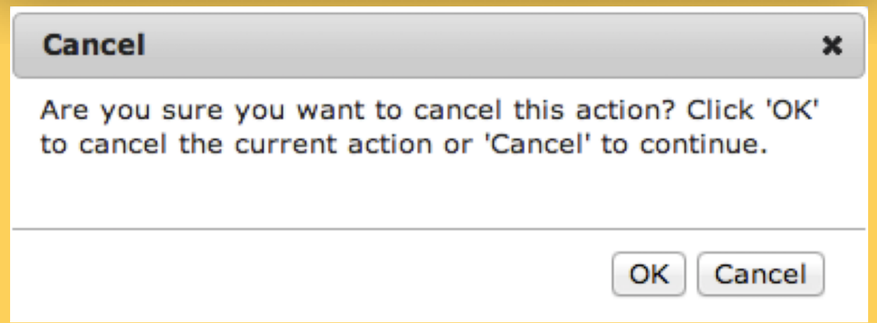
A user rolls two dice and tell us the number on each of the rolled die. Our program will add the die numbers together and check them against a target number that only the program knows. It will then report back whether the total of the dice was higher, equal or lower than the secret number.

1. Take in the result of each die - how do we read input?
    a. Read input of die 1
    b. Read input of die 2
2. Add the die numbers together
    - sum = die1+die2
3. Check them against a target number - based on steps 3 and 4, it looks like we need to make a decision - therefore IF statement
    - Define the target number
4. Output if total of the dice was higher, equal or lower than the target number. - output based on the decision that we made
    - Is sum greater than target number?
    - Is sum less than target number?
    - Is sum equal to the target number?

BREAKING THINGS







It is really good practice to think about how it is possible to break your code? What can go wrong?

- Try and counter for these breaks!
- Important to have good error messages:
  - Tells the user exactly what has gone wrong
  - How can they fix it?
  - What is happening!?

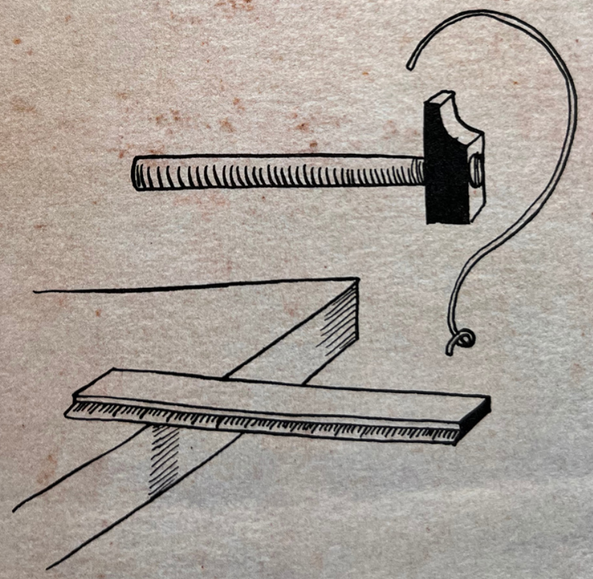
LET'S TRY IT WITH SCANF()

- Gives us the ability to scan stuff in from the terminal (standard input)
- We have to tell the computer what we expect to scanf() - is it an int, a char, or a double?
- But since scanf() is a function does it return something?
    - Yes, scanf() returns the number of input values that are scanned
    - If there is some input failure or error then it returns EOF (end-of-file) - we will look at this more later on!
    - This can be useful to check for any errors

BREAK TIME

CENTRE OF GRAVITY

Is it possible to keep a ruler flat in the position shown in the drawing below, simply using a hammer and a piece of string?



Note: you cannot place the hammer on the ruler!

WHEN DO WE NEED TO LOOP?

REPETITION

- Any time your program needs to keep doing something (repeating the same or similar action) until something happens and you may not know how many times that will be in advance

- Can you think of some examples in real life?
    - While there are songs in my playlist, keep playing the songs

WHILE

REPETITIVE TASKS SHOULDN'T REQUIRE REPETITIVE CODING

- C normally executes in order, line by line (starting with the main function after any # commands have been executed)
  - 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
- Let's see an example where it is inefficient to copy and paste code...

WHILE

WHILE SOMETHING IS TRUE, DO SOMETHING

- **`while()`** loops - can commonly be controlled in three ways:
  - Count loops
  - Sentinel loops
  - Conditional loops

```
1 while (expression) {
2     // This will run again and again until
3     // the expression is evaluated as false
4  }
5 // when the program reaches this }, it will
6 // jump back to the start of the while loop
```

WHILE

CONTROL THE WHILE LOOP

```
1 // 1. Initialise the loop control variable
2 // before the loop starts
3
4 while (expression) { // 2. Test the loop
5                       // control variable,
6                       // done within the
7                       // (expression)
8
9     // 3. Update the loop control variable
10    // usually done as the last statement
11    // in the while loop
12 }
```

TO INFINITY AND BEYOND

TERMINATING YOUR LOOP

- It's actually very easy to make a program that goes forever
- Consider the following while loop:

```c
// To infinity and beyond!

while (1 < 2) {
    printf("<3 COMP1511 <3");
}
```

CONTROL THE WHILE LOOP

COUNT LOOPS

- Use a variable to control how many times a loop runs - 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

```c
// 1. Declare and initialise a loop control
// variable just outside the loop
int count = 0;

while (count < 5) { // 2. Test the loop
                    // control variable
                    // against counter
    printf("I <3 COMP1511");

    //Update the loop control variable
    count = count + 1;
}
```

CONTROL THE WHILE LOOP

COUNT LOOPS

```
int scoops = 0;
int sum = 0;

// 1. Declare and initialise a loop control
// variable just outside the loop
int serves = 0;

while (serves < 5) { // 2. Test the loop
                     // control variable
                     // against counter
    printf("How many scoops of ice cream have
    you had?");
    scan("%d", &scoops);
    sum = sum + scoops;
    printf("You have now had %d serves\n", serves);
    printf("A total of %d scoops\n", sum);
    serves = serves + 1; // 3. Update the loop
                         //  control variable
}
printf("That is probably enough ice-cream\n");
```

SENTINEL VALUES

WHAT IS A SENTINEL?

- When we use a loop counter, we assume that we know how many times we need to repeat something
- Consider a situation where you don't know the number of repetitions required, but you need to repeat whilst there is valid data
- A sentinel value is a 'flag value', it tells the loop when it can stop...
- For example, keep scanning in numbers until an odd number is encountered
    - We do not know how many numbers we will have to scan before this happens
    - We know that we can stop when we see an odd number

CONTROL THE WHILE LOOP

SENTINEL LOOPS

- Sentinel Loops: 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
- It is declared and set outside the loop
- It’s “termination condition” can be checked in the while expression
- It will be updated inside the loop (often attached to a decision statement)

CONTROL THE WHILE LOOP

SENTINEL LOOPS

```c
int scoops = 0;
int sum = 0;

// 1. Declare and initialise a loop control
// variable just outside the loop
int end_loop = 0;

while (end_loop == 0) { // 2. Test the loop
                        // control variable
    printf("Please enter number of scoops today: ");
    scan("%d", &scoops);
    if (scoops > 0) {
        sum = sum + scoops;
    } else {
        end_loop = 1; // 3. Update the loop
                      // control variable
    }
}
```

CONTROL THE WHILE LOOP

CONDITIONAL LOOPS

- Conditional Loops: can also use a condition to decide to exit a loop at any time
- This is called conditional looping
- Also do not know how many times we may need to repeat.
- We will terminate as a result of some type of calculation

CONTROL THE WHILE LOOP

COUNT LOOPS

```
int scoops = 0;

// 1. Declare and initialise a loop control variable
// Since I want the sum to be as close to 100
// as possible, that is my control condition
int sum = 0;

while (sum < 100) { // 2. Test the loop
                    // condition
    printf("Please enter number of scoops: ");
    scan("%d", &scoops);

    // 3. Update the loop control variable
    sum = sum + scoops;
}
printf("Yay! You have eaten %d scoops of ice cream", sum);
```

ACTION TIME

CODE DEMO

- While loop with a counter:
`while_count.c`
- While loop with a sentinel:
`while_sentinel.c`
- While loop with a condition:
`while_condition.c`

Feedback Please

I value your feedback and use to pace the lectures and improve your overall learning experience. If you have any feedback from today's lecture, please follow the link below. Please remember to keep your feedback constructive, so I can action it and improve the learning experience .

https://www.menti.com/aljgtermcocq

WHAT DID WE LEARN TODAY?

CONDITIONS

if /else /else if

Decision problems

Relational Operators

Logical Operators

if_logic.c

LOGICAL OPERATORS AND IF WITH CHAR

lower.c

CHAINING IF/ELSE AND ERROR CHECKING

dice_checker

WHAT DID WE LEARN TODAY?

LOOP THE LOOP WHILE (COUNTER)

while_counter.c

LOOP THE LOOP WHILE (SENTINEL)

while_sentinel.c

LOOP THE LOOP WHILE (CONDITION)

while_condition.c

WEDNESDAY: LOOP INSIDE A LOOP (CAN'T GET ENOUGH OF A LOOP)

grid: grid.c

pyramid: pyramid.c

REACH OUT

CONTENT RELATED QUESTIONS

Check out the forum

ADMIN QUESTIONS

cs1511@cse.unsw.edu.au