COMP1511 PROGRAMMING FUNDAMENTALS

LECTURE 4

Loop the loop

LAST LECTURE...

ON TUESDAY

- Conditionals - running out code based on some sort of condition being met
- More complex IF statements
- Introducing the struct

IN THIS LECTURE

TODAY...

- Let's loop the loop **`while()`**

“

WHERE IS THE CODE?

Live lecture code can be found here:

HTTPS://CGI.CSE.UNSW.EDU.AU/~CS1511/22T1/LIVE/WEEK02/

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

```
// 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
```

WHILE

CONTROL THE WHILE LOOP

```
// 1. Initialise the loop control variable
// before the loop starts

while (expression) { // 2. Test the loop
                     // control variable,
                     // done within the
                     // (expression)

    // 3. Update the loop control variable
    // usually done as the last statement
    // in the while loop
}
```

TO INFINITY AND BEYOND

TERMINATING YOUR LOOP

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

```
while (1 < 2) {
    printf("It is time for some Messina ice-cream");
}
```

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

```
// 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("It is time for some Messina ice-cream");
}
```

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 count = 0;

while (count < 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 of ice-
    cream, with a total of %d scoops\n",
    scoops, sum);
    count = count + 1; // 3. Update the loop
                       //  control variable
}
```

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

COUNT LOOPS

```
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 to
    add to your daily consumption: ");
    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 to
    add to your daily consumption: ");
    scan("%d", &scoops);

    // 3. Update the loop control variable
    sum = sum + scoops;
}
```

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`

BREAK TIME

TIME TO STRETCH

There are 50 motor bikes, each has a petrol tank holding enough petrol to go 100km. Using these motor bikes, what is the maximum distance you can go?

WHILE INSIDE A WHILE

PUTTING A LOOP INSIDE A LOOP

- If we put a loop inside a loop . . .
- Each time a loop runs
- It runs the other loop
- The inside loop ends up running a LOT of times

PROBLEM TIME

PRINT OUT A GRID OF NUMBERS

- Print out a grid of numbers:

1 2 3 4 5
1 2 3 4 5
1 2 3 4 5
1 2 3 4 5
1 2 3 4 5

- Break down the problem...
- Get it down to a component that you can do...

PROBLEM TIME

PRINT OUT A PYRAMID OF NUMBERS

- What if we now print out a half pyramid of numbers:

1
1 2
1 2 3
1 2 3 4
1 2 3 4 5

- Break down the problem...
- Get it down to a component that you can do...

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/fwucrz4rwx

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

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

grid: print_grid.c

pyramid: print_pyramid.c

REACH OUT

CONTENT RELATED QUESTIONS

Check out the forum

ADMIN QUESTIONS

cs1511@cse.unsw.edu.au