Loops

We know how to control the flow of execution in our programs, by using the \textit{if}-statement. BUT having to rerun the program repeatedly would not be convenient.

Consider the Problem
Reading in a positive integer \(n\) and print \(n\) asterisks.

Most non-trivial C programs contain some form of \textit{looping}. This means that the same block of code is executed:
- a certain number of times; or
- while some condition is true.

Andrew Taylor  COMP1911 Computing 1A

The \textit{while} Loop

One of the looping constructs in C is \textit{while}:
\begin{verbatim}
while (EXPR) {
    stmt1;
    stmt2;
    ...
    stmtn;
}
\end{verbatim}

Behaviour
The block of statements in the \textbf{while} loop is executed repeatedly while the condition of the loop (\textit{EXPR}) is true (i.e., evaluates to a non-zero value).

Andrew Taylor  COMP1911 Computing 1A

Template for Simple Loops

Here is the skeleton of a loop that iterates \(n\) times:
\begin{verbatim}
int i, n; // declare loop counter
i = 0; // initialise loop counter
while (i < n) {
    /* statements the loop needs to perform */
    i = i + 1; // increment loop counter
}
\end{verbatim}

IMPORTANT
Be sure to increment/decrement the loop counter so that it converges on the termination condition. An infinite loop results if you don’t and the program will never terminate!
Nested Loops

How can we print a square of $10 \times 10$ asterisks (*)?

```c
int i, j;
i = 0;
while (i < 10) {
    j = 0;
    while (j < 10) {
        printf("* ");
        j = j + 1;
    }
    printf("\n");
i = i + 1;
}
```

Open-Ended Iteration

With loops it is possible to:
- iterate a constant number of times; or
- an unspecified number of time.

PROBLEM: What if we want to iterate until we see some particular input or until an event occurs?

Task

Write a program that reads integers from the input and returns the sum. The program should stop when it reaches the end of input or it encounters an input error.

How do we approach this problem? In particular what is the loop condition?

- we must repeatedly read numbers
- the loop condition should use the return value from `scanf`

Open-Ended Iteration

The sum program:

```c
int n, ret;
int sum = 0; // what happens if you don’t init it?

printf("Please enter an integer: ");
ret = scanf("%d", &n);
while (ret == 1) {
    sum = sum + n;
    printf("Please enter another integer: ");
    ret = scanf("%d", &n);
} printf("The sum is: %d\n", sum);
```

Does this program work as expected?

Loop Termination

Can you see the logic flaw in the following code fragment?

```c
int year;
printf("Enter your year of birth: ");
while (scanf("%d", &year) != 1) {
    printf("That is not a valid year!\n");
    printf("Please try again: ");
}
```

Upon invalid input the loop will enter an infinite cycle! Try it!

This is because subsequent `scanf` statements still attempt to read the same invalid input.

Careful

You must double and triple-check that the logic of your loop conditions is sound and leads to termination in all cases!
The **for** Loop

The **while** loop is versatile enough to handle all types of looping. However, C provides a more compact looping construct, the **for** loop, which makes certain types of loops easier to write.

```c
int i;
i = 0;
while (i < n) {
    // body of loop
    i = i + 1;
}

// this is equivalent to the while loop above
for (i = 0; i < n; i = i + 1) {
    // body of loop
}
```

The **for** Loop

This is the skeleton of **for**:

```
for (INIT; EXPR; UPDATE) {
    stmt1;
    stmt2;
    ...
    stmtn;
}
```

**Behaviour**

The **INIT** part is executed upon entering the loop, this usually **initialises** the counter variable, then if the condition of the loop (**EXPR**) is **true** (i.e., evaluates to a non-zero value) the block of statements in the **for** loop is executed. Finally, the **UPDATE** part is executed, this usually **increments** the counter variable, and the **loop repeats** by skipping **INIT** and evaluating **EXPR**.

Template for Simple Loops

Here is the skeleton of a loop that iterates \( n \) times:

```c
int i; // declare counter

// initialise, test, increment
for (i = 0; i < n; i = i + 1) {
    // insert here whatever the loop needs to do
}
```

Have you noticed that this template loop iterates through values of \( i \) between 0 and \( n - 1 \)? This is the range of indices in an \( n \)-element array, we will look at arrays next week!

**Remember**

Construct your loop condition so that the loop **terminates**!

Nested Loops

How can we print the top right diagonal half of a \( 10 \times 10 \) square of asterisks?

```c
#define SIZE 10

int i, j;

for (i = 0; i < SIZE; i = i + 1) {
    for (j = 0; j < SIZE; j = j + 1) {
        if (j < i) {
            printf(" ");
        } else {
            printf("* ");
        }
    }
    printf("\n");
}
```