C-Lessons

Control structures 2

Lecturers: Mirko Jantschke, Pascal Scholz

13. November 2018
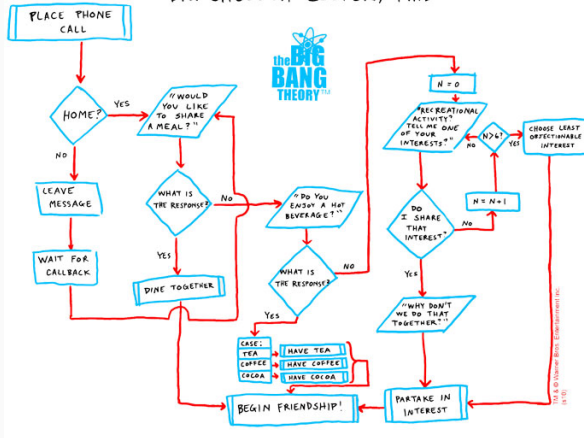
Overview

Motivation

Take a look at the right part. It is executed up to seven times.

Loops

Loops

To repeat statements as long as a certain condition is met, C offers 3 different loops.

```c
while (condition)
    statement;
```

```c
do
    statement;
while (condition);
```

```c
for (initialization; condition; statement)
    statement;
```

To use more than one *Statement* in your loop, use curly brackets like in the following example:

```c
for (int i = 5; i >= 0; --i) {
    if(i > 0)
        printf("This loop will run %i more times.\n", i);
    else {
        printf("This was the last run.\n");
        printf("Now leaving the loop...\n");
    }
}
```

The execution of a loop is a continuous alternation between
checking if the condition is still met and executing the statement(s).

```c
int i = 2;
while (i > 0)
    --i;
printf("done\n");
```

The execution of a loop is a continuous alternation between checking if the condition is still met and executing the statement(s).

```c
int i = 2;
while (i > 0)
    --i;
printf("done\n");
```

1. Check $(i > 0) \rightarrow$ **true** $\rightarrow$ go to line 3

The execution of a loop is a continuous alternation between checking if the condition is still met and executing the statement(s).

```c
int i = 2;
while (i > 0)
    --i;
printf("done\n");
```

1. Check (i > 0) → **true** → go to line 3
2. Decrement i → i now is **1**, go back to line 2

The execution of a loop is a continuous alternation between checking if the condition is still met and executing the statement(s).

```c
int i = 2;
while (i > 0)
    --i;
printf("done\n");
```

1. Check (i > 0) → **true** → go to line 3
2. Decrement i → i now is **1**, go back to line 2
3. Check (i > 0) → **true** → go to line 3

The execution of a loop is a continuous alternation between
checking if the condition is still met and executing the statement(s).

```
1  int i = 2;
2  while (i > 0)
3      --i;
4  printf("done\n");
```

1. Check (i > 0) → **true** → go to line 3
2. Decrement i → i now is **1**, go back to line 2
3. Check (i > 0) → **true** → go to line 3
4. Decrement i → i now is **0**, go back to line 2

The execution of a loop is a continuous alternation between
checking if the condition is still met and executing the statement(s).

```c
int i = 2;
while (i > 0)
    --i;
printf("done\n");
```

1. Check $(i > 0) \rightarrow$ **true** $\rightarrow$ go to line 3
2. Decrement i $\rightarrow$ i now is **1**, go back to line 2
3. Check $(i > 0) \rightarrow$ **true** $\rightarrow$ go to line 3
4. Decrement i $\rightarrow$ i now is **0**, go back to line 2
5. Check $(i > 0) \rightarrow$ **false** $\rightarrow$ go to line 4

The execution of a loop is a continuous alternation between
checking if the condition is still met and executing the statement(s).

```c
int i = 2;
while (i > 0)
    --i;
printf("done\n");
```

1. Check (i > 0) → **true** → go to line 3
2. Decrement i → i now is **1**, go back to line 2
3. Check (i > 0) → **true** → go to line 3
4. Decrement i → i now is **0**, go back to line 2
5. Check (i > 0) → **false** → go to line 4
6. Print **done**

The difference between *do…while* and *while* is the order of executing the statement(s) and checking the condition.

The *while* loop begins with checking, while the *do…while* loop begins with executing the statement(s).

```
int i = 3;
do
    --i;
while (i < 1);
```

The Statement(s) in a *do … while* loop are executed at least once.

for

The For-Loop is comfortable for iterating. It takes three arguments.

- Initialization
- Condition
- Iteration statement

For illustration, consider a program printing the numbers 1 to 10:

```
int i;
for (i = 1; i <= 10; ++i)
    printf("%d\n", i);
```

- *i* is called an *index* iterating from the given start to a given end value
- *i, j, k* are commonly used identifiers for the index

Miscellaneous

Meanwhile...

Be careful, this

```
while (1 > 0)
    printf("Did you miss me?\n");
```

runs till the end of all days.

∞ loops are common mistakes, and you will experience many of
them.
Check for conditions that are always true.

The arguments for the *for loop* are optional. E.g. if you already have defined your iterating variable:

```
int i = 1;
for (; i <= 10; ++i)
    printf("%d\n", i);
```

Or if you have the iteration statement in your loop body:

```
for (i = 1; i <= 10;)
    printf("%d\n", ++i);    /* seems more like a while loop */
```

And if you're not passing anything, it runs **for**ever:

```
for (;;)
    printf("I'm still here\n");
```

Note: the semicolons are still there.

Cancelling loops

break

- Ends loop execution
- Moves forward to first statement after loop

continue

- Ends current loop iteration
- Moves forward to next step of loop iteration
    - *while:* Jumps to condition
    - *for:* Jumps to iteration statement

Saving code lines

You can define variables inside the initialization part of a for loop.

```
for (int i = 1; i <= 10; ++i)
    printf("%d\n", i);
```

In that case, the variable is only available inside the for loop (as if it was declared in the body).

```
int guess = 0;
int solution = 42;
for (int i = 1; guess != solution; ++i)
    scanf("%d", &guess);
printf("Tries: %d\n", i);   /* invalid! */
```

This feature was added in the C99 standard.

Compiler options

When calling *gcc*, you can pass several options to it:

option	description
-std=c99	Use C99 as the standard
-o <name>	output file is *name* instead of *a.out*
-Wall	Enable all compiler warnings
-Wextra	Enable even more compiler warnings
-Werror	Treat warnings as errors

Example:

```
$ gcc -std=c99 -o main main.c
```

- Stay consistent after deciding whether to use or not to use braces on a single statement
- If you skip the loop body
  - Leave a comment in your code
  - Use an extra line for the empty statement

```c
for (i = 1; i < 9; printf("%d\n", ++i));      /* confusing */

for (i = 1; i < 9; printf("%d\n", ++i))       /* clear */
    ;/* do nothing */
```

Related Task

Task as online:
Write a program that takes two numbers a, b from the user and calculates a^b and b^a. Try using one 'for' and one 'while' loop.

Experts: Write a program that calculates the factorial (a! and b!) of the user input.