Functions and Flow Control

Eric McCreath

Australian National University

Why Functions?

Functions provide a way of grouping code that achieves a particular computational task.

This has a number of advantages for the programmer including:

- They can create "black boxes" (which are the functions) that do a particular task. When the programer uses these functions they do not need to think about their inner workings. They can be tested and made robust in isolation. And they can be swapped for a different function that does the same thing without modification of the containing code.
- They can make code more readable by partitioning code into logical parts.
- Functions can reduce repetition within code.
- They facilitate recursion.

Parameters

The parameters given to a function are just the evaluated expressions (parameters are passed by value). Hence if you wish a variable, in the calling context, to be modified then you need to pass a pointer to this variable into the function.

Parameters within a function are like local variable, so you can modify them but the modifications will not have any effect outside that context of the function.

```
#include<stdio.h>
void add(int a, int b, int *res) {
     *res = a + b;
}

void main() {
    int res;
    add(4,5,&res);
    printf("%d\n",res):
}
```

Designing Functions

- When designing a function think about its cohesion and coupling. Aim to make each function achieve one cohesive task, while reducing the coupling with other parts of your code.
- When possible limit side effects within your functions.

while - loop

The while loop provides a simple way of repeating a block of statements 0 or more times.

The boolean expression is calculate prior to the block of code being repeated or executed for the first time. If the expression evaluates to true then the block of code is executed. If the expression evaluates to false then the block is not executed and the loop ends.

```
while (boolean_expression) {
  // block of code
}
```

Below is an example of printing 1 to 10 with a while loop.

```
int i = 1;
while (i <= 10) {
    printf("%d ",i);
    i++;
}
```

for - loop

The for-loop, like the while-loop, provides a simple way of repeating a block of code 0 or more times. However, it is a little more structured and is often used for iterating over an array.

The for loop has: an initialization statement that is executed once before the loop starts; a boolean expression which must evaluate to true for the loop to start or repeat; and statement that is executed once the loop completes the block of code but before the boolean expression is evaluated.

```
for (init_statement ; boolean_exp; block_end_statement) {
   // block of code
}
```

Using the for-loop to count to 10:

```
for (i=1;i<=10;i++) {
   printf("%d ",i);
}
```

if-else

"if" and "if-else" statements enable you execute parts of your code when some condition is true or false.

```
if (boolean_expression) {
   // this is only executed when the boolean_expression evaluates to true
} else {
  // this is only executed when the boolean_expression evaluates to false
}
```

You can also just have the "if" part:

```
if (x > 100) {
   printf("That's a big number!\n");
}
```

They will often get joint together using the following pattern:

```
if (x < 0) {
   printf("x is negative\n");
} else if (x <= 100) {
   printf("x is in 0 to 100\n");
} else {
   printf("That's a big number!\n");
}
```

Exersizes

- Write a recursive function that calculates the gcd of two integers.
- Reformulate this gcd function so it uses a while loop rather than recursion.