# ESP_LEC_7

Functions in C

Functions in C are more general than mathematical functions. They can be viewed as small programs on their own. Functions have several advantages:

• Less code duplication – easier to read /update programs

• Simplifies debugging – each function can be verified separately

• Reusable code – the same functions can be used in different programs (e.g. printf)

```
return-type function-name(parameters) {
   declarations
   statements
   return value;
}
```

*return-type* type of value returned by function or void if none function-name unique name identifying function *parameters* comma-separated list of types and names of parameters *value* returned upon termination (not needed if *return-type* void)
The list of parameters is a declaration on the form

```
type_1 par_1, ..., type_n par_n
```

and represents external values needed by the function. The list of parameters can be empty. All declarations and statements that are valid in the main program can be used in the function definition, where they make up the *function body*.
Example, computing averages

```
double average(double x, double y) {
return (x+y)/2;
}
```

The return statement forces the function to return immediately, possibly before reaching the end of the function body.
Like variables a function must be declared before it can be used (called).
The declaration of a function resembles the definition,

```
return-type function-name(parameter
types);
```

The function body is replaced by a semi-colon. Parameters need not be named, it is sufficient to specify their types.
Example, declaring average

```
double average(double, double);
```

Declaration is not necessary if the function definition precedes the first call.

## Function calls

```
variable=function-name(arguments);
```

*variable* is assigned the return-value of the function. The arguments are values with types corresponding to the function parameters.
For example,

```
int main (int argc, char **argv) {
double a=1;
double avg;
avg = average(a, 3.0);
return 0;
}
```