FUNCTIONS AND CONTROL FLOW

Klaus-Peter Zauner

ELEC1201 Programming
Retrospect

By now you should be able to:

• Open an editor, type in a minimal C-program, compile it, run it, and debug it

• Know how to declare a variable, assign it a value, and display the value on the terminal

• Know where you can look up additional information about C
Functions
Functions

C is structured programming language: Parts of code can be named and then called up by the name.

- Very important aspect to make the code understandable for humans
- Allows to modularize code: a function that has been tested can be used with confidence again and again from various places => Libraries
- Ideally: isolated with a clear interface—reality is more messy
Functions

- Functions have a Name
- Possibly several inputs: Arguments
- Possibly one output: Return Value
- They can have *side effects*
  => change the state outside the function directly (not through the return value)
Defining a Function

```c
int myname(int myin ){
    return( ++myin );
}
```

Adds one to its argument and returns the result.

- **Name:** myname
- **Return type:** int
- **Arguments:** 1 of type int
- **Function have file scope**
Defining a Function

```c
int myname( int myin ){
    return( ++myin );
}
```

Adds one to its argument and returns the result.

- Name: myname
- Return type: int
- Arguments: 1 of type int
- Function have file scope
Defining a Function

```c
int myname( int myin ){
    return( ++myin );
}
```

Adds one to its argument and returns the result.

- Name: myname
- Return type: int
- Arguments: 1 of type int
- Function have file scope
Defining a Function

```c
1 int myname( int myin ){
2       return( ++myin );
3 }
```

Adds one to its argument and returns the result.

- Name: myname
- Return type: int
- Arguments: 1 of type int
- Function have file scope
Declaring a Function

```c
int main(){
    int x = 1;

    printf("x = %d modified to %d\n",
            x, myname(x) );
}

int myname( int myin ){
    return( ++myin );
}
```
Declaring a Function

```c
int myname(int myinput);

int main(){
    int x = 1;

    printf("x = %d modified to %d\n", x, myname(x));
}

int myname(int myin){
    return( ++myin );
}
```
Local Variables

• Scope within block → ...
• Created on the function stack
  • not initialized (value unknown)
  • do not use for large amounts of data
• Valid only during execution
  use static prefix to preserve value; then created in memory
• Only return as value (never its address)
Function Arguments

- Passed “by value” → a copy
- The function cannot change the value outside its scope (it would change only the copy)
- To enable a function to change the value of its argument we need to pass it the location in memory where the argument is located (“pass by reference”) → &
Function Calls

- Redirects control from caller to called function
- Returns control flow to caller when function terminates
Control Flow
Control Flow

In the simplest case a program is a list of commands that are executed one after another. More useful is the ability to change which commands are executed depending on the state of variables. The state of the variables may be the result of inputs or could be due to internal state changes.

=> The flow of where in the code the program goes is controlled by the data.
Conditionals

Execute code only if a condition is satisfied.

```c
if( a == b ){
    printf( "a and b are identical\n" );
}
```

Note, in C any value that is not zero is regarded as true

```c
if( a ){
    printf( "a is not zero\n" );
}
```
Conditionals

```c
if( a == b ){
    printf( "When does this print?\n" );
}
```
if( !a ){
    printf( "a is zero\n" );
}

Zero is special: it can be tested very fast (often through special hardware), it is used to indicate the end of strings and to indicate invalid pointers.
Loops

• It is very common that some code should be repeated
• Functions help if code needs to be repeated at different locations in the control flow
• Repetition that is conditional on some state (variable) is very often required
for-loop

Repeat something a given number of times:

```c
int i;

for (i = 0; i < 10; i++) {
    printf("%d ", i);
}
```
while-loop

Repeat code until smoothing changes:

```c
while(!f){
    f = read_flag();
}
```
while-loop

In C this is allowed:

```c
while(!(f = read_flag()));
```