

Operating Systems Lab (C+Unix)

Enrico Bini

University of Turin

1/12E. Bini (UniTo)

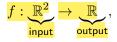
Outline

① C: functions

E. Bini (UniTo) OS Lab (C+Unix) 2/12

Functions

- Functions are used to break down a complex problem into smaller ones
- If you find yourself copying/pasting lines of code which "do something", then you may need a function for that code
- Functions are not parametric macros
- As in mathematics with



- a C function gets an input and produces an output
- A C function is characterized by
 - the <u>declaration of the function</u> (aka function prototype), which holds information about
 - ★ the name of the function (mandatory)
 - * the list of types of input parameters (optional)
 - ★ the type of one output parameter (optional)
 - ② the **body** of the function, which is the code that processes the inputs to produce the output
- the void type is specified for missing input, output or both

Functions: declaration (or prototype)

- The compiler requires that a function is declared before being used
- The declaration of a function (or prototype) is a line of code with
 - the type of one output parameter (void if none)
 - ▶ the *name* of the function (mandatory)
 - a comma-separated list of types of input parameters within round brackets (optional)
 - ▶ terminated by a semi-colon ";"

Notice: the compiler (step "2" of gcc) allows using a function by only knowing its declaration!!

Example of function declaration

```
/*
 * Sorting the array v of length num
 */
void sort(int * v, unsigned int num);
```

and an equivalent way without the parameter names

```
void sort(int *, unsigned int);
```

Functions: definition (or body)

- The definition of a function includes
 - 1 its declaration and
 - its body

```
int min(int a, int b)
{
        if (a<b) {
            return a;
        } else {
            return b;
        }
}</pre>
```

- The body is needed by the linker only (step "4" of gcc)
- Why may it be useful to have a declaration without a body?
 - ► Libraries of functions expose to the user the declaration of the functions only (in the header file, such as stdio.h)
 - ► The body may be intentionally hidden to protect the code
 - ► The function body and the code using the function may be both developed and compiled separately (by different teams)
- test-declare-fun. c

Functions: invocation

- The declaration or the full definition of a function must appear above its first usage
 - otherwise the compiler doesn't recognize the function name
 - ▶ try to move #include <stdio.h> at the bottom in hello. c
- A function is invoked by passing the parameters in accordance to the declaration

```
int min(int, int);
int main() {
   int a;
   a = min(4,-2);
}
```

- at compile time, only the function declaration is needed
- a function fun with void list of parameters is invoked by fun()

Functions: passing parameters

- When a function is invoked, the invocation parameters are copied into additional variables
- A function can use and modify the paramaters
- These modifications, however, have no effect outside the function

```
int mul(int x, int y) {
    x *= y; /* we can use variable x */
    return x;
}
int main() {
    int a, b=4;
    a = mul(b,3);
    /* what is the value of b? */
}
```

Fuctions: how to modify a parameter?

- Often times it is needed that a function modifies one or more parameters. Example: to sort an array
- However, parameters are always copies: any change to a parameter is lost after returning
- Solution: if some data needs to be modified by a function, then we
 declare a function that receives a pointer to the data, not the data
 itself
- Through the pointer the original data may be modified
- Example

```
void sort(int * v, unsigned int n)
{
/*
   * Sorting elements v[0],...,v[n-1]
   */
}
```

- The ponter only is copied internally to the function. The function can then access and modify the data through the (copied) pointer
 - often time called "call by reference"

Functions: passing const parameters

- Sometimes it is needed to pass a large amount of data to functions (a long vector, etc)
- To avoid copying all the data as parameters (which is inefficient), it is advisable to pass only a reference to the data (a pointer)
- In this way, however, the function may accidentally (or maliciously) modify the data
- To pass a pointer to a data structure that we don't want to modify we use the keyword const before the parameter
- For example (man 3 printf)

```
int printf(const char *format, ...);
```

Functions: returning

- the keyword return is used to return the value of a function
- once return is executed, no other statement of the function is executed
- there may be more than one return in the function body: the first one that is encountered is the one executed
- functions with void output:
 - has not return statement: it completes once the closing bracket "}" is reached
 - may have a return; with no value

Functions vs. parametric macros

- stage of gcc: macros expanded by preprocessor, functions are compiled
- type checking: in macros, the type of operands is not checked
- **efficiency**: macros may be more efficient than functions, no parameters passing, no call instruction
- **size of executable**: if macros are used used the size of the executable grows
- parameters: macros are expanded by the pre-processor, if a
 paramenters is modified it remains modified after the macro as well.
 A modification of parameters within functions isn't seen outside
- return value: macros do not return any value. Still, a macro may be an expression
- recursion: obviously, no recursion with macros
- debugging: programs with many macros may be harder to debug

E. Bini (UniTo) OS Lab (C+Unix) 11/12

Functions vs. parametric macros: conclusion

- Macros may be a good replacement of functions when:
 - 1 the lines of code are few (say, 10)
 - 2 the function code is used many times
 - high efficiency is needed
 - 4 no return value, nor recursion is used
 - we are ready to hard-to-debug errors
 - o gcc -E is your friend
- Macros may be good for:
 - computing the minimum between two values
- Functions may be good for:
 - sorting an array