CH-230-A

Programming in C and C++

C/C++

Tutorial 5

Dr. Kinga Lipskoch

Fall 2020

The const Keyword

- The modifier const can be applied to variable declarations
- It states that the variable cannot be changed
  - i.e., it is not a variable but a constant
- When applied to arrays it means that the elements cannot be changed

const Examples

```
1  const double e = 2.71828182845905;
2  const char str[]= "Hello world";
3  e = 3;              /* error */
4  str[0] = 'h';       /* error */
```

- You can also use `#define` of the preprocessor
- But defines do not have type checking, while constants do

More const Examples

- `const char *text = "Hello";`
  - Does not mean that the variable `text` is constant
  - The data pointed to by `text` is a constant
  - While the data cannot be changed, the pointer can be changed
- `char *const name = "Test";`
  - `name` is a constant pointer
  - While the pointer is constant, the data the pointer points to may be changed
- `const char *const title = "Title";`
  - Neither the pointer nor the data may be changed

Dealing with Big Projects

- Functions are a first step to break big programs in small logical units
- A further step consists in breaking the source into many files
  - Smaller files are easy to handle
  - Objects sharing a context can be put together and easily reused
- C allows to put together separately compiled files to have one executable

Declarations and Definitions

- **Declaration**: introduces an object. After declaration the object can be used
  - Example: functions' prototypes
- **Definition**: specifies the structure of an object
  - Example: function definition
- Declarations can appear many times, definitions just once

Building from Multiple Sources

- C compilers can compile multiple sources files into one executable
- For every declaration there must be one definition in one of the compiled files
  - Indeed also libraries play a role
  - This control is performed by the linker
- `gcc -o name file1.c file2.c file3.c`

Libraries

- Libraries are collection of compiled definitions
- You include header files to get the declarations of objects in libraries
- At linking time libraries are searched for unresolved declarations
- Some libraries are included by `gcc` even if you do not specifically ask for them

Linking Math Functions: Example

```
1  #include <math.h>
2  #include <stdio.h>
3
4  int main() {
5    double n;
6    double  sn;
7
8    scanf("%lf\n", &n); /* double needs %lf */
9    sn = sqrt(n);
10   /* conversion from double to float ok */
11   printf("Square root of %f is %f\n", n, sn);
12   return 0;
13 }
14
15      gcc -lm -o compute compute.c
```

Compilers, Linkers and More

- Different compilers differ in many details
  - Libraries names, ways to link against them, types of linking
- Check your documentation
- But preprocessing, compilation and linking are common steps

Recursive Functions (1)

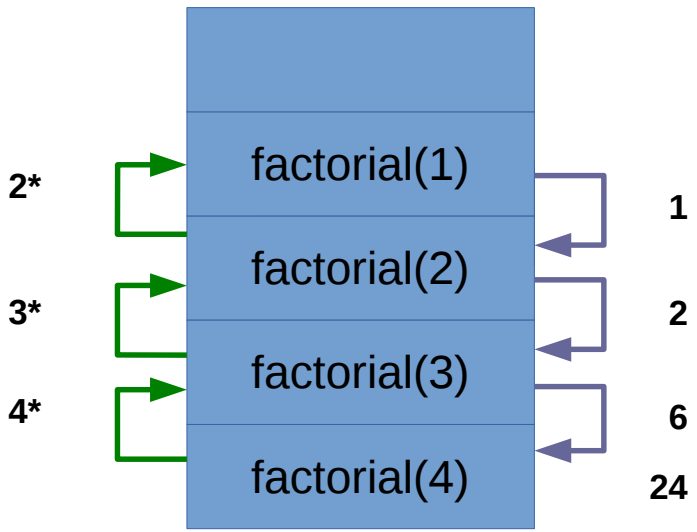
- Can a function call other functions?
  - Yes, indeed function calls appear only inside other functions (and everything starts with the execution of `main`)
- Can a function call itself?
  - Yes, but in this case special care should be taken
- A function which calls itself is called a **recursive function**
- Function A calls function A
- At a certain point function B calls A
  - A calls A then A calls A then A calls A ...
- When coding recursive functions attention should be paid to avoid endless recursive calls

Recursive Functions (2)

- Recursion theory can be studied for a longer time: here we will just scratch its surface from a basic coding standpoint
- Every recursive function must contain some code which allows it to terminate without entering the recursive step
  - Usually called **inductive base** or **base case**
- When recursion is executed, the new call should be driven "towards the inductive case"

Stack of Calls: Example

```
1 int factorial(int n) {
2   if ((n == 0) || (n == 1))
3     return 1;
4   else
5     return n * factorial(n - 1);
6 }
```

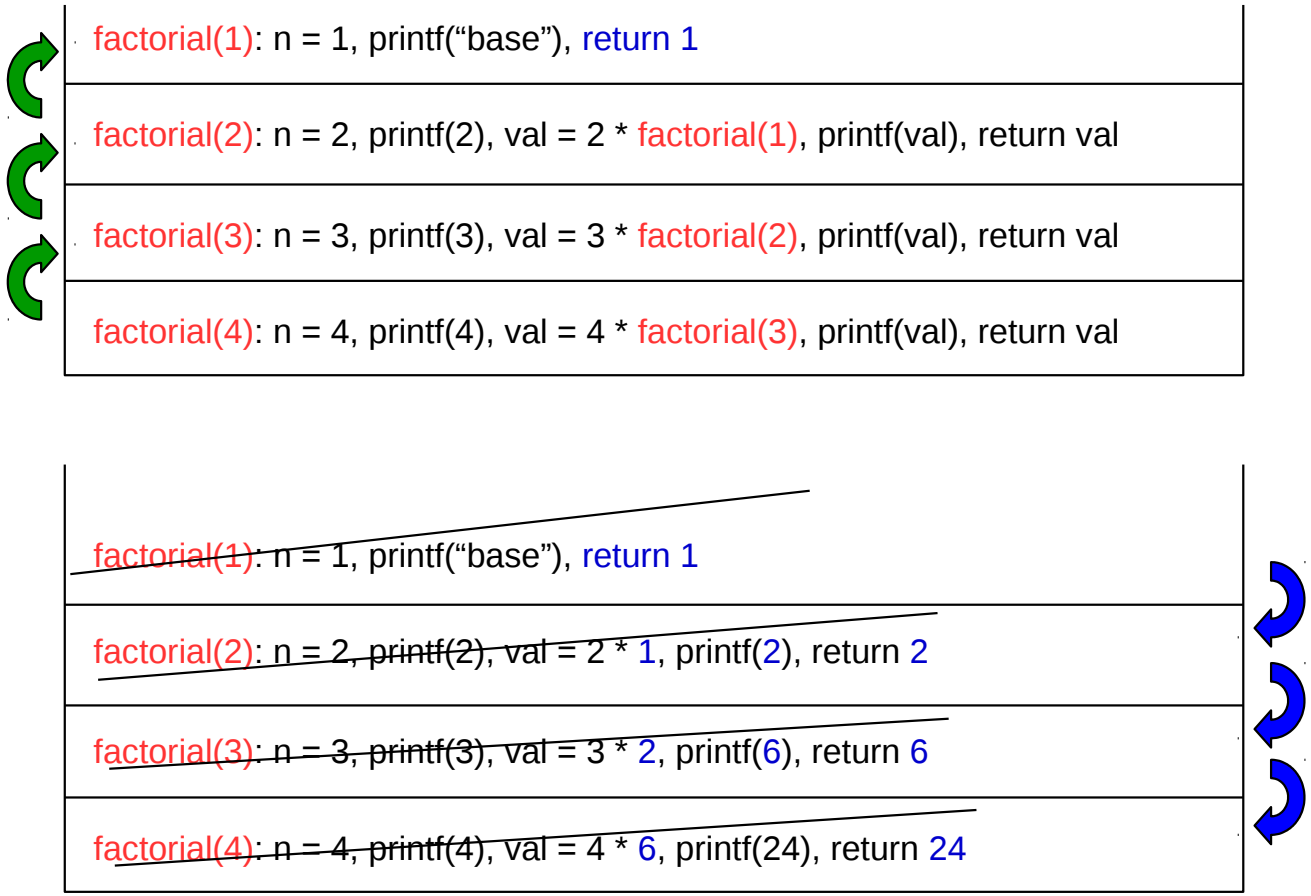


Tracing the Stack of Calls (1)

```
1  int factorial(int n) {
2    int val;
3    if ((n == 0) || (n == 1)) {
4      printf("base\n");
5      return 1;
6    } else  {
7      printf("called with par = %d\n", n);
8      val = n * factorial(n - 1);
9      printf("returning %d\n", val);
10     return val;
11   }
12 }
13 int main() {
14   printf("%d\n", factorial(4));
15   return 0;
16 }
```

Tracing the Stack of Calls (2)

From the `main`: call `factorial(4)`



One More Example: Fibonacci Numbers

$$F(N) = \begin{cases} 1, & N \leq 1 \\ F(N-1) + F(N-2), & N > 1 \end{cases}$$

```
1 int fibonacci(int n) {
2   if ((n == 0) || (n == 1))
3     return 1;
4   else
5     return fibonacci(n-1) + fibonacci(n-2);
6 }
```