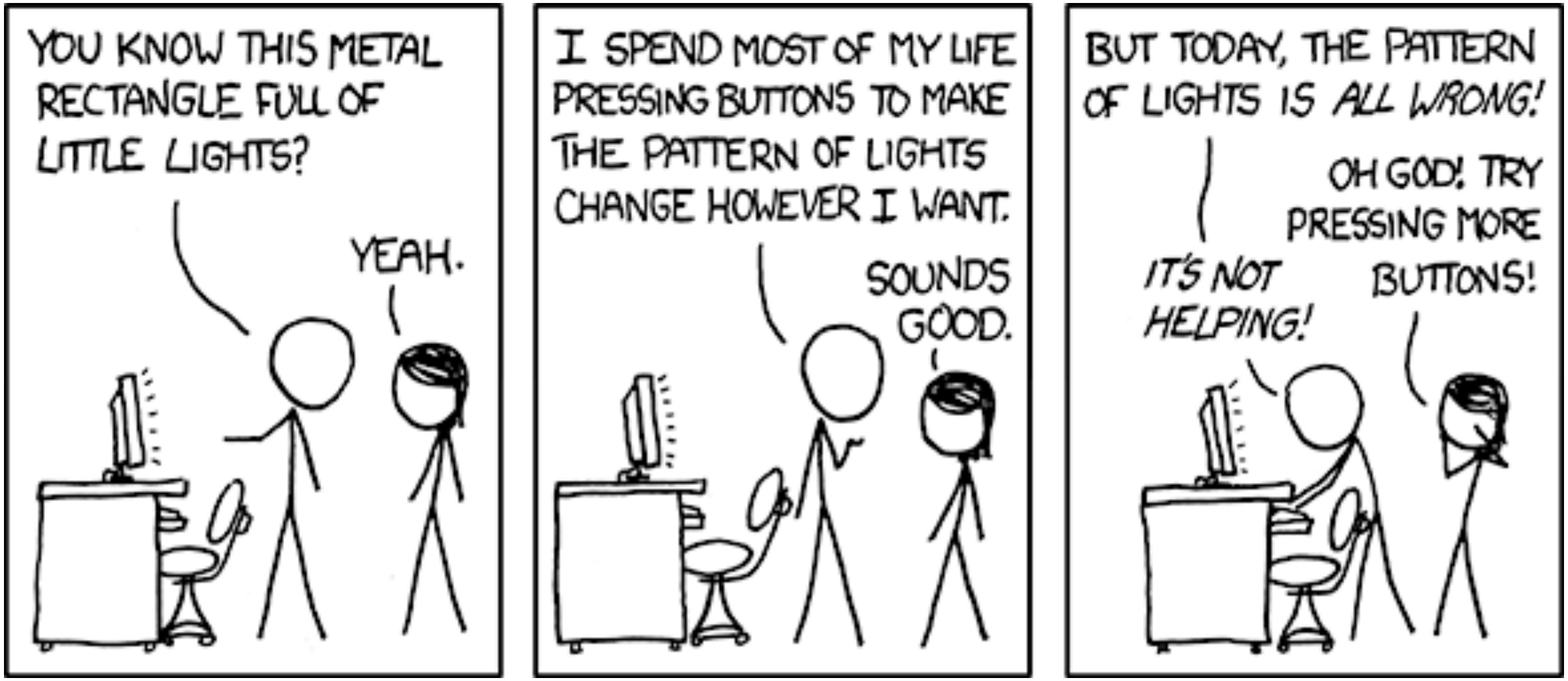
C Programming I: Lecture IV

Andreas Heinz
FYD400 - HT 2019


YOU KNOW THIS METAL RECTANGLE FULL OF LITTLE LIGHTS?
YEAH.
I SPEND MOST OF MY LIFE PRESSING BUTTONS TO MAKE THE PATTERN OF LIGHTS CHANGE HOWEVER I WANT.
SOUNDS GOOD.
BUT TODAY, THE PATTERN OF LIGHTS IS ALL WRONG!
OH GOD! TRY PRESSING MORE BUTTONS!
IT'S NOT HELPING!

xkcd.com

Recap questions

- How do you print a float with 3 digits after the comma?
- What is the key difference between printf() and scanf()?
- What does the modulus operator do?
- What is the difference between pre- and post-increment operators?
- How do you distinguish between logic and bit-wise operators?

Lecture 4 Contents

- Control structures and statements
- Functions and modular programming
- Pointers – a first glance

(VtC, Chapter 6 and 7)

Control Structures

- see algorithm/tools (Lecture 2)
- 4 different control structures:
  - calculation (last lecture)
  - comparison (last lecture)
  - choice (e.g. with “if” or “switch”)
  - iteration (e.g. with “while or for”)

Statements

- executed sequentially
- all statements end with a “;”
- 6 groups: expression, selection, iteration jump, block and null statements
- compound statement enclosed by “{ }” but have no “;”
  - groups several statements into a single statement
  - common in loops
  - block: compound statement including declarations
- null statement: “;” should to be on an extra line – “no action”

```
while (!KeyHit())
        ;                 //  window left open
                          //  “;” creates an open loop
```

Selection Statements: if

- **most common selection statement**

-
  ```
  if (expression)        // if expression != 0 is true then statement 1
                         // is executed
       statement 1;      // Note the indention! Need “;” !
  ```

-
  ```
  if (expression)        // if value !=0 execute statement 1
       statement 1;
    else                 //  expression is not true
       statement 2;
  ```
- **an *if* inside *statement 1* or *2* results in nested if-statements**

Nested Selection Statements

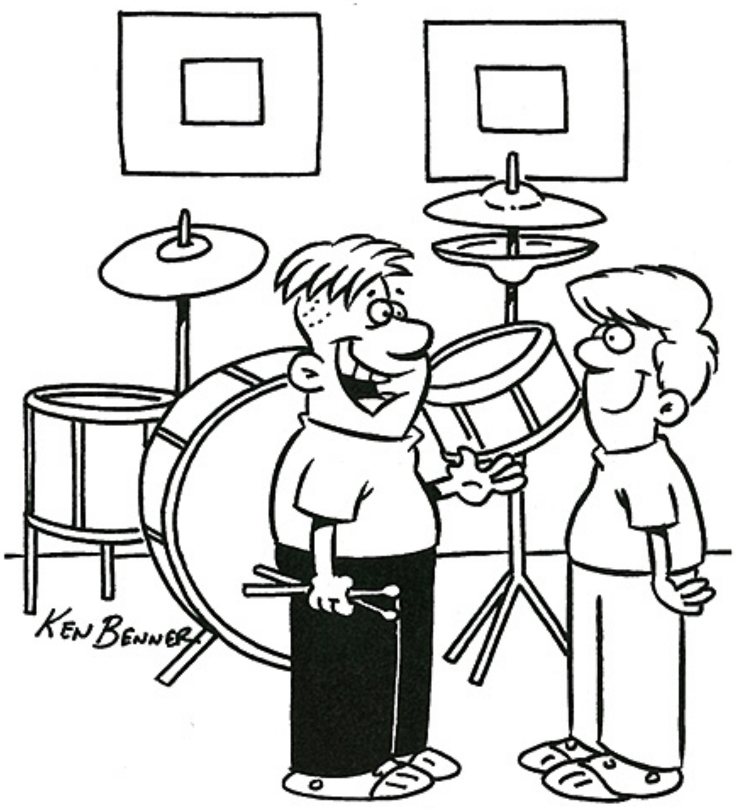
```
int main (void)               // modulus and nested if statements
{
          int x;
          printf (“Get an integer:\n”);
          scanf (“%i”, &x);
          if (x%2 == 0)       // true for all integer numbers divisible by 2
                    {
                     printf (“Divisible by 2 ”);
                     if (x%4 == 0)
                              printf (“and divisible by 4!\n”);
                    }
          else
                              printf(“Odd number!”);
          return 0;
}
```

“else” refers always to the last open “if”!

Exercise

Take a piece of paper and write down the C code for the a program that finds the largest number of 3 numbers.

This is not an exam and no one but you will see what you wrote.



"My dad says that I'm so good, I don't need to practice anymore."

Exercise

```
#include <stdio.h>                    /* if–else exercise */
int main(void)
        {
        float n1, n2, n3;
        printf("Enter three numbers: ");
        scanf("%f %f %f", &n1, &n2, &n3);     // read in 3 floats
        if (n1 >= n2 && n1 >= n3) // if n1 is larger or equal n2, n3 do
                                  // this
                printf("%.2f is the largest number.", n1);

        else if (n2 >= n1 && n2 >= n3) // if n2 is larger or equal n1,n3
                                       // do this
                printf("%.2f is the largest number.", n2);

        else    //otherwise do this;  n3 is largest
                printf("%.2f is the largest number.", n3);

        return 0;
}
```

Selection Statements: switch

easier to read and often faster -> select from many options without nested *ifs*

switch, case, default, break **are all reserved keywords**

```
enum operator {addition, subtraction};         // declare a list
enum operator select;                          // type?
int x=2, y = 3;
select = addition;
switch(select){
          case addition:
                    printf (“Sum: %d”, x + y);
                    break;    // Forgetting the “break” is a common mistake!
          case subtraction:
                    printf (“Difference: %d”,x-y);
                    break;
          default:
                    printf (“Wrong operator!”);
                    break;                            }
```

Selection Statements: switch

easier to read and often faster -> select from many options without nested *ifs*

switch, case, default, break **are all reserved identifiers**

```
enum operator {addition, subtraction};          // declare a list
enum operator select;                           // type is enum operator
int x=2, y = 3;
select = addition;
switch(select){                         // argument of switch has to be an int (or char)!
          case addition:
                    printf ("Sum: %d", x + y);
                    break;    // Forgetting the "break" is a common mistake!
          case subtraction:
                    printf ("Difference: %d",x-y);
                    break;
          default:
                    printf ("Wrong operator!");
                    break;                          }
```

Loops

while, for **are the most common loops**

- ***while*** **(expression) – execute loop if expression != 0; i.e. true**

  statement

  while (getchar() != EOF)
  ++ *nchar;* **// counts the number of characters…**

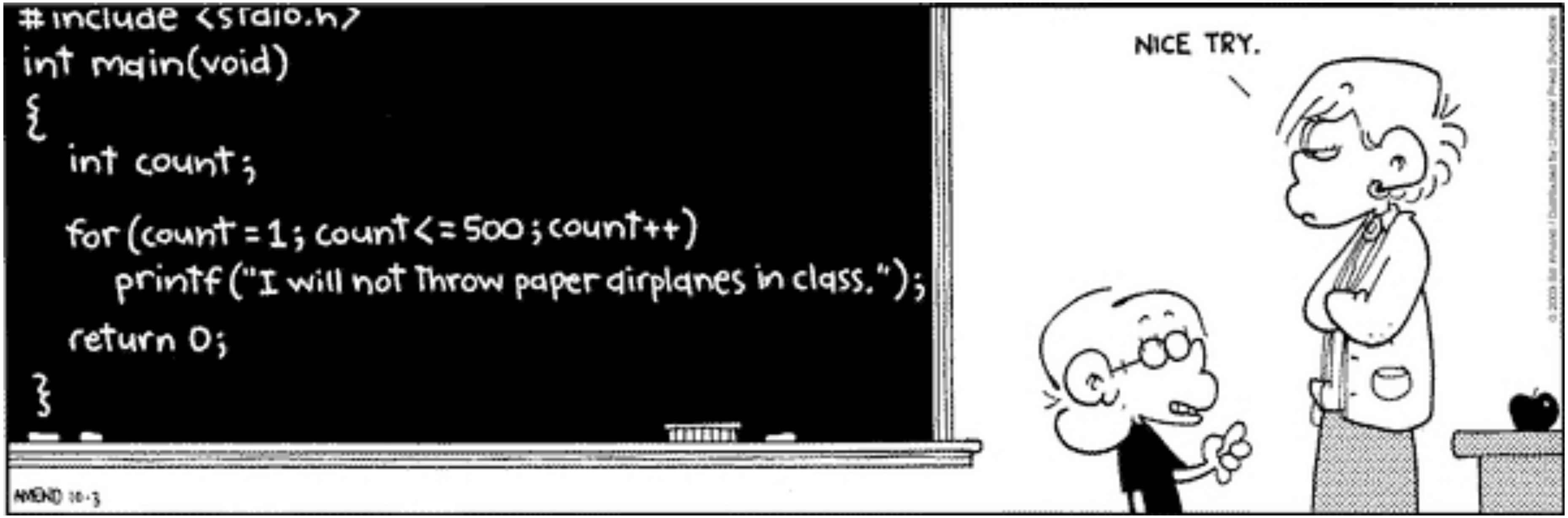
- ***do*** **– same as while but condition is checked after loop execution**

- ***for (initialization, condition, modification)***

  Example: ***for (nchar = 0; getchar () != EOF; ++nchar)***
  ;

  - **initialization – necessary before the first loop**
  - **condition – checked before execution of the first loop**
  - **modification (usually increment) in the next loop**

Loops



```
#include <stdio.h>
int main(void)
{
  int count;

  for(count = 1; count<= 500; count++)
     printf("I will not Throw paper airplanes in class.");

  return 0;
}
```

xkcd.com

Functions

- **int main (void) // main function of C and start of execution**
  {
  }
  - **int: type of return value**
  - **main: identifier**
  - **( ): void = no parameters are passed to the function**
  - **{ }: function body – has so far contained the entire program**
- **divide a program into different functions (inside the same .c file)**
  - **generalize, systematize, clarify, prevent misuse**
- **definition, declaration, call**
- **In C functions are the building blocks of a program!**

Global and Local Variables

Global Variables

- These variables are declared **outside** all functions.
- Life time (**scope**) of a global variable spans the **entire** execution period of the program.
- Can be accessed by **any** function defined below the declaration.

Local Variables

- These variables are declared **inside** of a function.
- Life time (**scope**) of a local variable spans the time it takes to execute the function.
- Can be accessed only **within** the function where it is declared (usually).

Functions: Example “max” in one .c file

```
# include <stdio.h>
float max (float x, float y)    // function definition
         {                      // you always need { }!
                 if (x > y)
                          return x;
                 else
                          return y;
         }
int main (void){                // the program starts here!
                 float a, b;
                 printf (“Type a number:\n”);
                 scanf (“%f”, &a);
                 printf (“Give another number:\n”);
                 scanf (“%f”, &b);
                 printf (“Larger number: %f”, max (a, b)); // call of
                                                            //function max
                 return 0;
         }
```

Functions in the same .c file

- function definition (header+body}
  - type of the return value - if nothing is returned: "void”
  - type of all parameters (poss. ellipsis notation “,...”, or void)
  - needs to occur before the first call of the function (we will see later how to get around this...)
- function call - what happens?
  - function return type, name, arguments (e.g. “a” and “b”)
  - calculation of arguments (if those are expressions)
  - transfer of the arguments to the function (types ..) (“call by value”)
  - declare (“x” and “y”) = actual parameters (memory is allocated)
  - execution of the function body
    => Careful! This does NOT affect the variables that are passed as arguments!
  - return of the result (if any) (free allocated memory)

Functions – Modified Example

```
# include <stdio.h>
float max (float x, float y); //makes the function known to main, note “;”!
int main (void)                  // the program starts here!
           {
             float a, b;
             printf (“Type a number!\n”);
             scanf (“%f”, &a);
             printf (“Give another number!\n”);
             scanf (“%f”, &b);
             printf (“Larger number: %f”, max (a, b)); // call of function max
                     return 0;
           }
float max (float x, float y) // declared before; defined here!
           {
                     if (x > y)
                                return x;
                     else
                                return y;
}
```

Functions in Several Files

- function declaration

```
float mean ( float x, float y);
```

  - same syntax as the function definition

    header +";" (prototype)
  - declaration before the call
- function definition – the whole thing:

```
float mean (float x, float y)
        { return (x+y)/2.0; }
```

  - name, body, function parameters => memory allocation
  - *extern* (default) – function can be called from other files
  - *static* can "hide" a function from other files

=> Definition and declaration must match – your responsibility!

Example: Functions in Several Files

```
// main.c
# include <stdio.h>

extern float avge(float x, float y);
```

declaration

```
int main (void)
{
   float a, b;
   printf("Get a number!\n");
   scanf("%f", &a);
   printf("Get another number! \n");
   scanf("%f", &b);
   printf("Average: %f:", avge(a,b));
   return 0;
}
```

no ";" here

```
// function.c
# include <stdio.h>

float avge (float x, float y)
{
   return ((x+y)/2.0);
}
```

Example: Functions in Several Files

```
// main.c
# include <stdio.h>

extern float avge(float x, float y);

int main (void)
{
   float a, b;
   printf(“Get a number!\n”);
   scanf(“%f”, &a);
   printf(“Get another number! \n”);
   scanf(“%f”, &b);
   printf(“Average: %f:”, avge(a,b));
   return 0;
}
```

declaration

no “;” here

```
// function.c
# include <stdio.h>

float avge (float x, float y)
{
   return ((x+y)/2.0);
}
```

definition

Example: Functions in Several Files

```
// main.c
# include <stdio.h>

extern float avge(float x, float y);

int main (void)
{
   float a, b;
   printf(“Get a number!\n”);
   scanf(“%f”, &a);
   printf(“Get another number! \n”);
   scanf(“%f”, &b);
   printf(“Average: %f:”, avge(a,b));
   return 0;
}
```

declaration

no “;” here

```
// function.c
# include <stdio.h>

float avge (float x, float y)
{
   return ((x+y)/2.0);
}
```

definition

How to tell main.c about function.c?

Example: Functions in Several Files

```
// main.c
# include <stdio.h>

extern float avge(float x, float y);

int main (void)
{
   float a, b;
   printf(“Get a number!\n”);
   scanf(“%f”, &a);
   printf(“Get another number! \n”);
   scanf(“%f”, &b);
   printf(“Average: %f:”, avge(a,b));
   return 0;
}
```

declaration

no “;” here

```
// function.c
# include <stdio.h>

float avge (float x, float y)
{
   return ((x+y)/2.0);
}
```

definition

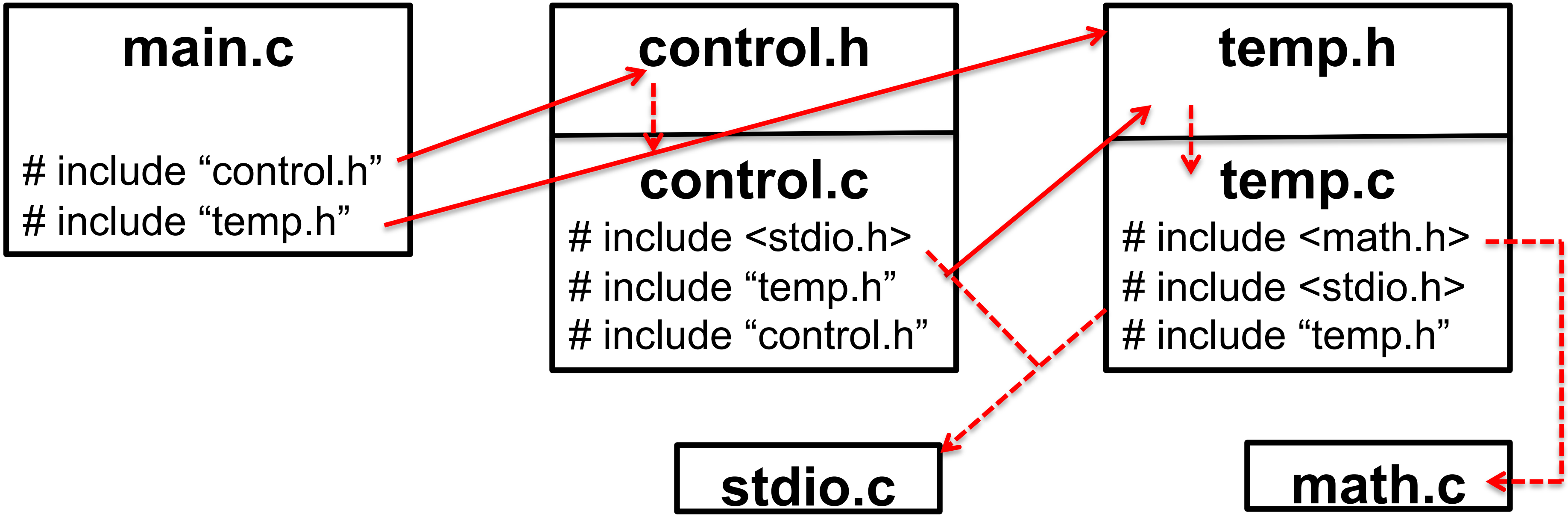
How to tell main.c about function.c?

⇒ you need to have both .c files in the same project (IDE), or you compile separately and link the object files.

Functions in several files – large projects

Option 1: - collect all declarations in the same .h file
- include this .h file in all .c files - see Fig. 6.1 in VtC
Option 2: - couple .h and .c files according to the problem => create modules

- **in .h files: function declarations, in .c files: corresponding function definitions**
- **provides clarity (and less to compile if modified), see Fig. 6.2 in VtC**



main.c

# include “control.h”
# include “temp.h”

control.h

control.c

# include <stdio.h>
# include “temp.h”
# include “control.h”

temp.h

temp.c

# include <math.h>
# include <stdio.h>
# include “temp.h”

stdio.c

math.c

Functions: Recursion

- **Recursion**
  - **function calls itself**
  - **looks complex but simplifies sometimes a problem**
- **classic example: n! = 1 * 2 *3*.....*n**

```
int nfac (int n)     // function to compute n!
          {
            if (n <= 0)
                     return 1;
            else
                     return (n*nfac(n-1)); // All versions of nfac have their own
                                           // parameter n; n exists while function
          }                                // is executed.
```

Exercise: m = nfac (3); // how does nfac(int n) work?

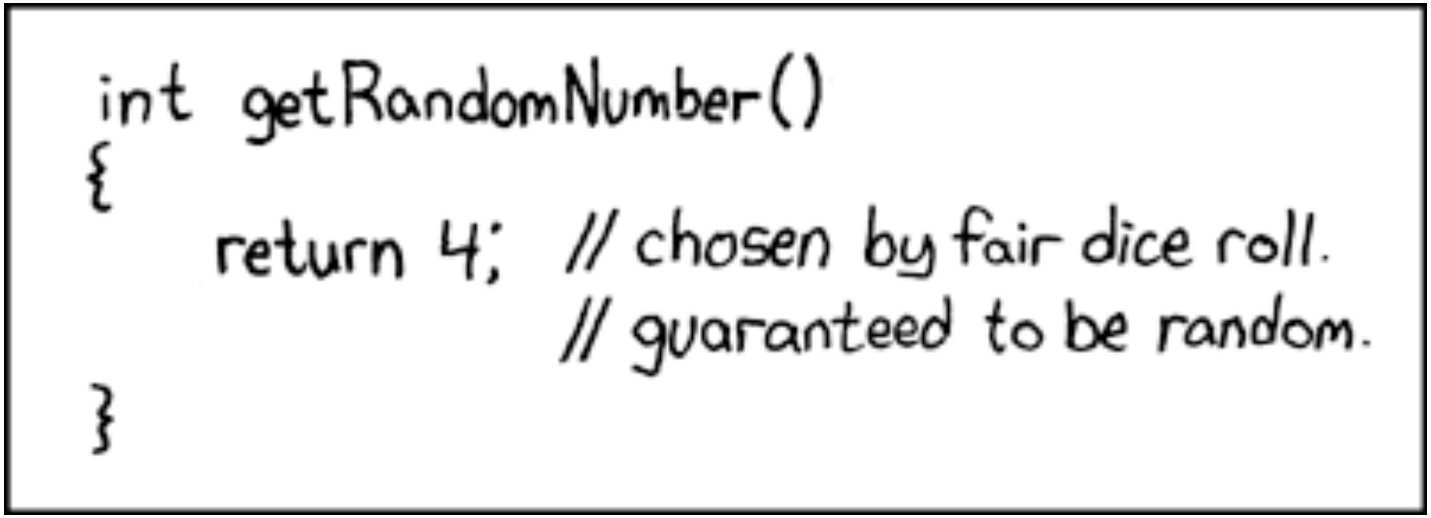
Functions: Recursion

- **Recursion**
  - **function calls itself**
  - **looks complex but simplifies sometimes a problem**
- **classic example: n! = 1 * 2 *3*…..*n**

```
int nfac (int n)     // function to compute n!
           {
             if (n <= 0)
                      return 1;
             else
                      return (n*nfac(n-1)); // All versions of nfac have their own
                                            // parameter n; n exists while function
           }                                // is executed.
```

Exercise: m = nfac (3); // how does nfac(int n) work?
3 > 0 gives else, call nfac again with (3-1) = 2, 2> 0,…(2-1) = 1, again, 1> 0, recalling with n = 0, return = 1 => 1 * 1, return 1, 2 * 1, return 2, 3 * 2, returns 6

Functions and Algorithms



```
int getRandomNumber()
{
    return 4;  // chosen by fair dice roll.
               // guaranteed to be random.
}
```

http://xkcd.com/221/

The best function does not compensate for a poor algorithm!

Exercise

www.UShumor.com

```
void salami (num)
          {
                    int num, count;
                    for (count = 1; count <= num; num++)
                              printf("O salami mio!\n");
           }
```

Is there a problem with this function?

Exercise

www.UShumor.com

```
void salami (num)
          {
                    int num, count;
                    for (count = 1; count <= num; num++)
                              printf(“O salami mio!\n“);
           }
```

Is there a problem with this function?

Yes!

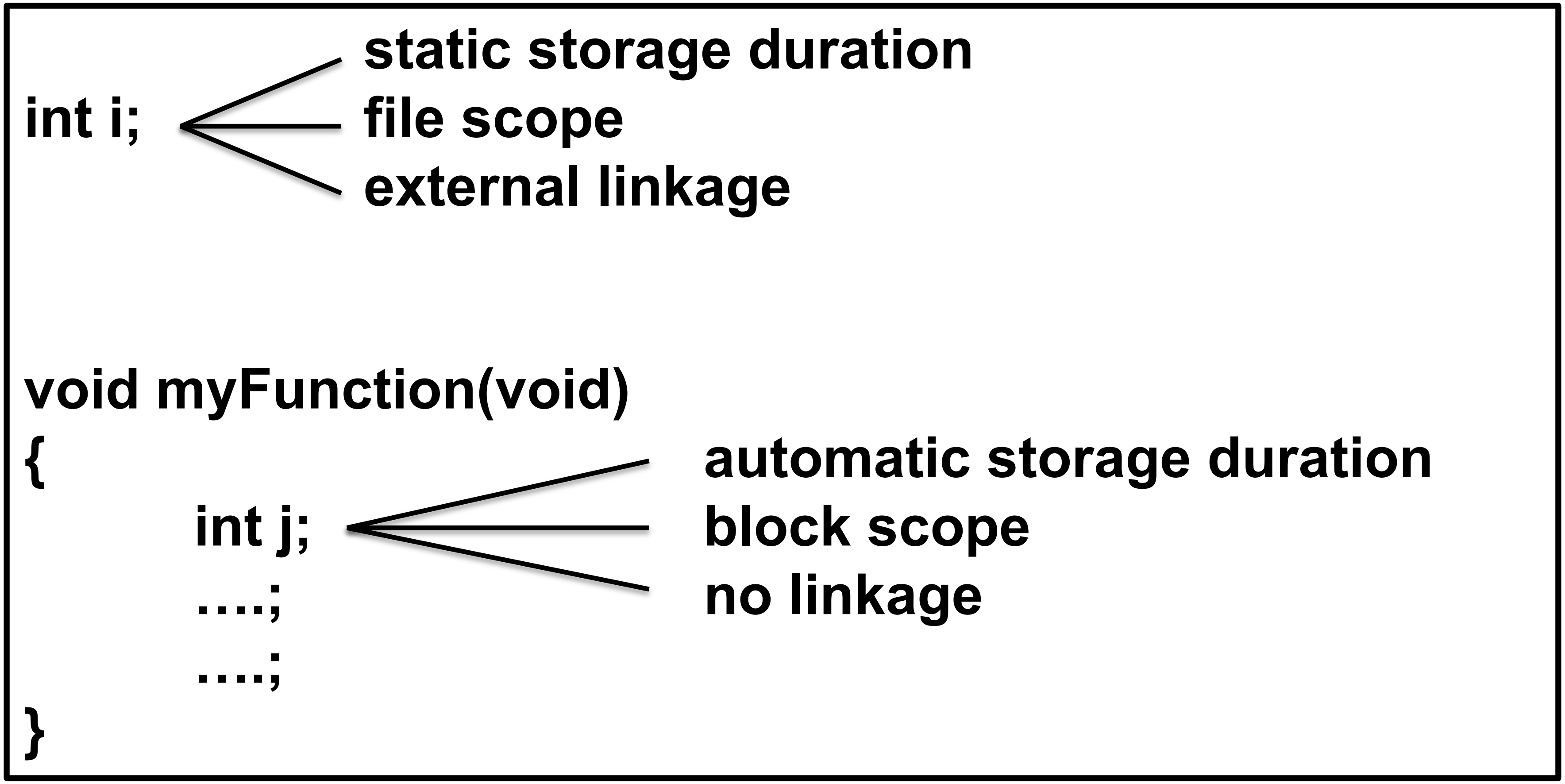
- **no type for num => automatic type => double declaration**
- **num increases during execution -> infinite loop**

Storage Classes

- properties of a variable: storage duration, scope, linkage:
  - storage duration => how long does the variable exist?
  - scope => from where can a variable be accessed? (block or file?)
  - linkage => from which files can a variable be accessed?
- storage class control via the key words*:*
  - *extern, static, auto, register (volatile)* // *extern, auto* most // important
- *extern, static* – default for variables and functions at <u>external levels</u>
  - variables and functions (=> several files)
  - memory not allocated during <u>declaration</u> but during <u>definition</u>
- *auto* – default for variables in a block or formal parameters

  ⇒ variable is visible and the memory is allocated <u>only inside</u> the function

Storage Classes: Defaults



```
int i;
```

static storage duration

file scope

external linkage

```
void myFunction(void)
{
        int j;
        ….;
        ….;
}
```

automatic storage duration

block scope

no linkage

Storage Classes

- ***static***
  - **variables and functions**
  - **static variables inside a function can "live" on**
  - **hide to prevent misuse (declaration outside all functions)**

    Note: static variables are not known outside their source file (.c)!
- ***register***
  - **like auto, just inside the function**
  - **possibly faster program execution**
  - **memory address not accessible (no "&" operator)**
- ***volatile* – not a proper storage class – "opposite to const"**

  application:
  - **memory space that can be modified by external input**
  - **prevents "optimization" by the compiler**
  - **e.g. clock / signal / flag from instruments**

Storage Classes: Example

```
extern int a = 42;                   // declaration of an ext. variable, to be used in avg
extern double func (float q);  // declaration; the definition is in another file
float avg (float x, float y)        // function definition (no external access)
            {
                        a++;            // declaration outside and external, “lives on”
                        return ((x+y)/2.0);       // x, y: not available outside, auto
            }
int hexa (double z)
            {…
                        int d;                    // vanishes at the end of hexa, auto
            }
int peta (void)
            {…
                        static float w;    // inaccessible from the outside; exists while the
                                           // program runs
            }
```

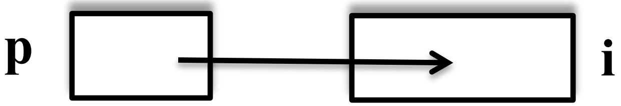
Memory and Addresses: Pointers

Address	Contents
0x0	01010011
0x1	01110101
0x2	01110011
0x3	01100011
…	
0x2000	10101010
0x2001	111110111
…	

Data are stored as one or more bytes with 8 bits each (for example).

Each variable has a memory address given by the address of the first byte.

variable i of type that needs 16 bits; address: 0x2000



"Pointer" p points to the address of i.

Summary of Lecture 4

- Recap
- Control structures and statements => realization of algorithms in C
  - if and switch for selection and while and for in loops
- Functions: solve a part of the problem and pass the result
  - modular programming
  - storage classes –or variables and functions
- Pointers: memory address of data items