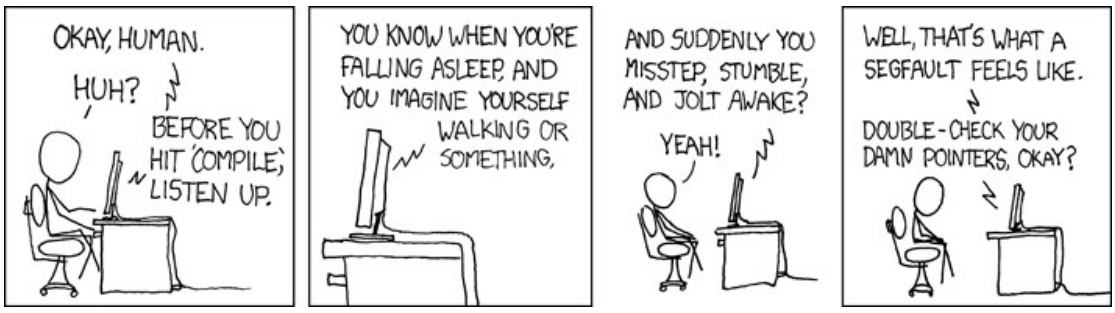
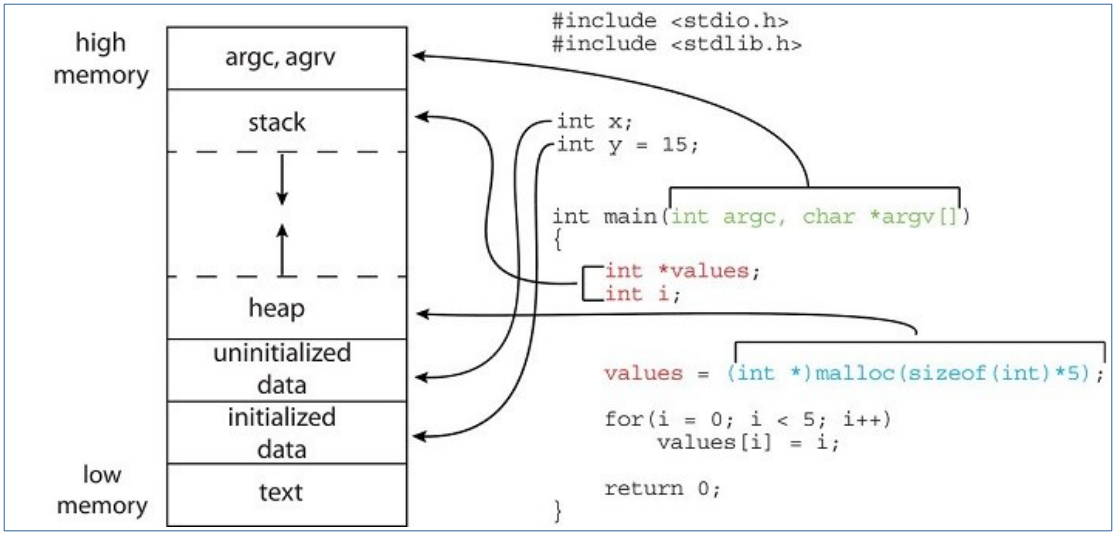
KU LEUVEN

GROUP T LEUVEN CAMPUS

Operating Systems
Programming with C: lab 2





Bachelor Electronics/ICT

Course coordinator: Bert Lagaisse
Lab coaches:

- Jeroen Van Aken
- Yuri Cauwerts
- Ludo Bruynseels

Last update: October 4, 2022

Programming with C: Lab 2

Lab target: *pointers in C, parameter passing (call-by-value/reference), understanding the function stack in C*

Exercise: parameter passing and the function stack

The following code is incorrect. **Draw the function stack** before, during, and after the function call to `date_struct()` to indicate where the problem appears.

```
typedef struct {
    short day, month;
    unsigned year;
} date_t;

void date_struct( int day, int month, int year, date_t *date) {
    date_t dummy;
    dummy.day = (short)day;
    dummy.month = (short)month;
    dummy.year = (unsigned)year;
    date = &dummy;
}

int main( void ) {
    int day, month, year;
    date_t d;
    printf("\nGive day, month, year:");
    scanf("%d %d %d", &day, &month, &year);
    date_struct( day, month, year, &d);
    printf("\ndate struct values: %d-%d-%d", d.day, d.month, d.year);
    return 0;
}
```

And what if we rewrite the code such that the function `date_struct()` returns a pointer to date? Draw the function stack before, during, and after the function call to `date_struct()` to find out what really happens. Explain why the code might work if the function f is not called.

```
typedef struct {
 short day, month;
 unsigned year;
} date_t;

void f( void ) {
 int x, y, z;
 printf("%d %d %d\n", x, y, z );
}

date_t * date_struct( int day, int month, int year ) {
 date_t dummy;
 dummy.day = (short)day;
 dummy.month = (short)month;
 dummy.year = (unsigned)year;
 return &dummy;
}
```

```c
int main( void ) {
 int day, month, year;
 date_t *d;
 printf("\nGive day, month, year:");
 scanf("%d %d %d", &day, &month, &year);
 d = date_struct( day, month, year );
 //f();
 printf("\ndate struct values: %d-%d-%d", d->day, d->month, d->year);
 return 0;
}
```

Solve the problem by allocating `dummy` on the heap instead of the stack. Now, use `date_t *d` to experiment with memory leaks. Validate your memory leak with *valgrind*. Use `date_t * d` to clean up in the end. Again, validate your clean-up with *valgrind*.

Exercise: parameter passing

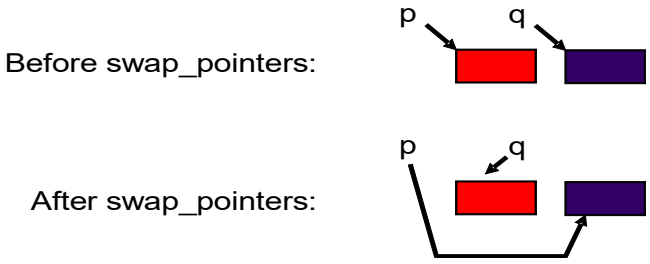
Implement the function '`swap_pointers`'. This functions takes two arguments of type void pointer and has no return value. The functions 'swaps' the two pointers as illustrated below. **Draw the function stack** before, during, and after the function call to swap_pointers(). What are the addresses when a and b are allocated on the heap?

```c
int a = 1;
int b = 2;
// for testing we use pointers to integers
int *p = &a;
int *q = &b;

printf("address of p = %p and q = %p\n", p, q);
// prints p = &a and q = &b

swap_pointers( ?p , ?q );


printf("address of p = %p and q = %p\n", p, q);
// prints p = &b and q = &a
```



Exercise: random numbers, the time() and sleep() functions

Implement a program that simulates a sensor node measuring the outdoor temperature. Use the pseudo-random number generator to simulate temperature readings and use the `sleep` function to generate temperature readings at a pre-defined frequency. The temperature values should be realistic outdoor values (not too

cold, too hot – e.g. between -10 and +35°C). Use `#define` to set the frequency, min. and max. temperature values. Print every reading as a new line on screen as follows:

```
Temperature = <temperature> @<date/time>
```

In this format, <temperature> should be printed with 1 digit before (= width) and 2 digits after (= precision) the decimal point, and <date/time> is the date and time as returned by the Linux 'date' command.

Hint 0: All functions that you could use have very informative man pages.

Hint 1: use the library function `srand`() to initialize the pseudo-random generator with the result of `time(NULL)`. You should call `srand()` only once.

Hint 2: Printing the temperature in the correct format can be easily done with the format specifier `%1.2f` i.s.o. `%f`.

Hint 3: `printf()` followed by `sleep()` could delay the output to screen due to buffered output. To avoid this use the statement `fflush(stdout)` just after `printf()`.

Hint 4: the `time.h` header file defines the functions `time(), asctime(), localtime()`. These could be of use for your implementation.