

# Hacking in C

## Assignment 2, Wednesday, February 8, 2017

**Handing in your answers:** Submission via Blackboard (<http://blackboard.ru.nl>)

**Deadline:** Wednesday, February 15, 23:59:59 (midnight)

1. You are given the following code fragment:

```
int main (void)
{
    short i = 0x1234;
    char x = -127;
    long sn1 = <STUDENT NUMBER OF TEAM MEMBER 1, WITHOUT LEADING S>;
    long sn2 = <STUDENT NUMBER OF TEAM MEMBER 2, WITHOUT LEADING S>;
    int y[2] = {0x11223344,0x44332211};

    ...
}
```

- (a) Write this code snippet to a file called `exercise1.c`.
- (b) Set the values of `sn1` and `sn2` to your student numbers.
- (c) Replace the `...` by code that prints the size in bytes of each of the local variables.
- (d) Extend the functionality of the program to print the memory layout of the local variables, in a byte-by-byte fashion, so a four-byte integer becomes four lines. More specifically, your program should print a table of the following form (addresses and data are fictional):

address	content (hex)	content (dec)
0x...00	0xFF	255
0x...01	0x12	18
0x...02	...	...

You do not have to sort the output.

**Hint:** To walk through memory byte-by-byte, you will want to use a pointer of type `char *`.

- (e) Compile your program with `gcc -O3 -Wall` and run the program. Write the output of the program to a file called `exercise1.out`. Explain which variable is stored at which location in memory and write this explanation to a file called `exercise1.exp`.
2. Since the C99 standard, the C programming language has a `bool` data type. Programs that use this data type have to include the file `stdbool.h`. They have to be compiled with the compiler flag `-std=c99`. Write a program (in a file called `exercise2.c`), which finds out about the internal representation of `bool`. Specifically, your program shall print the following:
  - How many bytes does a `bool` use?
  - What hexadecimal representation does a `bool` have, if you set it to `true`?
  - What hexadecimal representation does a `bool` have, if you set it to `false`?
  - Can you assign other hexadecimal values than these two to a `bool` variable? Are those interpreted as `true` or as `false` or do they cause an error?

3. This exercise is about pointer arithmetic. Write all parts of this exercise into a file called `exercise3.c`. For testing you probably need to write a `main` function; however, this should be in a separate file, which is not part of the submission.
  - (a) Consider the function `addvector.c` shown in the first werkcollege (the code is online on the course website). Rewrite this function to use pointer arithmetic instead of array indexing with bracket notation.
  - (b) Write your own version of the `memcmp` standard C library function. Don't use any array indexing with bracket notation but only pointer arithmetic.  
For documentation of this function, see `man memcmp` or <http://pubs.opengroup.org/onlinepubs/009695399/functions/memcmp.html>.
  - (c) Now write a function called `memcmp_backwards` with the same signature as `memcmp`. This function shall compare the two input byte arrays backwards, i.e., the sign of a non-zero return value shall be determined by the sign of the difference between the values of the *last* pair of bytes that differ in the objects being compared.  
Again, don't use any array indexing with bracket notation but only pointer arithmetic.
  - (d) **(optional)** For an additional challenge, think about how to make the `memcmp` function faster for long input arrays. As a hint, consider that in C it takes 1 operation to compare two values of a basic type (e.g. `char`, `int`) regardless of that type. If you decide to submit a solution to this part, write it into a function `memcmp_fast`, also in the file `exercise3.c`. Again, don't use any array indexing with bracket notation but only pointer arithmetic.
  - (e) **(optional)** For yet another challenge, think about how to ensure that the time taken by the `memcmp` function only depends on the length of the inputs, not on the values in the input arrays.  
If you decide to submit a solution to this part, write it into a function `memcmp_consttime`, also in the file `exercise3.c`. Feel free to use array indexing for this part.

4. Place the files

- `exercise1.c`,
- `exercise1.out`,
- `exercise1.exp`,
- `exercise2.c`, and
- `exercise3.c`

in a directory called `sws1-assignment2-STUDENTNUMBER1-STUDENTNUMBER2` (again, replace `STUDENTNUMBER1` and `STUDENTNUMBER2` by your respective student numbers). Write a `Makefile` that (with a single invocation of `make` in the `sws1-assignment2-STUDENTNUMBER1-STUDENTNUMBER2` directory) builds programs

- `exercise1` (from `exercise1.c`),
- `exercise2` (from `exercise2.c`), and
- `exercise3` (from `exercise3.c`).

Make sure that this `Makefile` is also in the `sws1-assignment2-STUDENTNUMBER1-STUDENTNUMBER2` directory.

Make a `tar.gz` archive of the whole `sws1-assignment2-STUDENTNUMBER1-STUDENTNUMBER2` directory and submit this archive in Blackboard.