I/O with characters

Consider the following code:

```c
#include <stdio.h>
int main(void) {
    int c;
    while ((c = getchar()) != EOF) {
        printf("char %c, ASCII code %d
", c, c);
    }
}
```

What gets printed on the screen for the following input: 0123 abc ABC

Arrays and Functions

Consider the following code:

```c
void max(int x[], int y[], int z[], int n) {
    int i;
    for(i=0; i<n; i++) {
        if( x[i]>y[i] )
            z[i] = x[i];
        else
            z[i] = y[i];
    }
}
int main(void) {
    int i, a[10], b[10], c[10];
    for(i=0; i<10; i++) {
        a[i] = 10*i;
        b[i] = 100-10*i;
    }
    max(a,b,c,10);
    return EXIT_SUCCESS;
}
```
What are the values of \(c[0], \ldots, c[9]\)?

**discuss:** What would happen if we wrote \(\max(a, b, c, 8)\); instead \(\max(a, b, c, 10)\)?

... if we wrote \(\max(a, b, c, 12)\)?

**Programming**

We would like to have a function that takes an integer number \(n\) and prints it out in hexadecimal format. Hexadecimal are base-16 numbers. Digits 0-9 have the usual meaning. We use letters a–f to stand for 10–15. *For example*, the decimal number 270 is 10e in hex.

This task is similar in principle to the task of printing decimal numbers digit-by-digit (see the *PrintDecimal* of Lecture 11). Note that the largest `int` storable is \(2^{31} - 1 = (16)^7 \times 8 - 1\).

**Hint:** Two ways of solving this.

(i) This can be done using normal arithmetic operators, similar to *PrintDecimal* (Lecture 11, 19 Oct). Slightly more tricky as we need to take care of a–f digits.

(ii) Alternatively, we can work with the *bitwise* representation of the integer, and use shifting and bit-masking. Check out the C operators & (not the same as &&), <<, >>.