```c
/*********************************************************
 * types:  learn how to declare and initialize variables *
 *         and how to output their values                *
 *********************************************************/

#include <stdio.h>
#include <limits.h>

int main(){
    int a;   // declaration of an integer variable
    /*
    The variable is defined utilising the assignment operator
    '=', the value on the right hand side of the operator is
    written encoded as zeros and ones into the memory location
    of a.
    */
    a = 300;

    /*
        We can output values of variables by the use of printf.
        Placeholder (format specifiers) are replaced by the
        values of the variables. The content of the variables
        is formatted depending on the conversion character
        (here d). For integers in decimal representation we
        write '%d', for the size_t type, we use '%zu'.
     */
    printf("a = %d, memory usage %zu B\n", a, sizeof(a));
    // a = 300, memory usage 4 B

    /*
        Note that printf can have multiple arguments, but the
        number of placeholders and variables must fit.
    */

    /*
        chars are small numbers. We can apply standard arithmetic
        operations to them as is presented here.
        Format specifier: '%c'
    */
    char my_char = 'a';
    printf("My first char is: %c\n", my_char);
    // My first char is: a
    printf("My first char + 2 is: %c\n", my_char + 2);
    // My first char + 2 is: c

    /*
        Strings are arrays of characters which include the
        null character '\0'. Output of the string is always
        stopped at the null character. We can initialize
        a character array utilising the following short
        hand notation
    */
    char my_string[] = "Hello!";
    // or more explicitly
    char another_string[] = {'H','e','l','l','o','!','\0'};

    // The format specifier for a string is '%s'.
    printf("%s\n",my_string); // Hello!
    printf("%s\n",another_string);// Hello!

    /*
        Double precision floating point numbers have the
        format specifier %f. Often, explicit type cast are
        necessary, when assigning integers to doubles to use
        the correct arithmetic operation. If below, no explicit
        type cast to double would have been performed, the result
        would be 0.0 as the operator '/' performs integer
        division when both operands are int's.
    */
    double my_double = ( (double) a) / 600;
    // Floating point number with two digits after the comma
    printf("my_double = %.2f\n", my_double);
    // my_double = 0.50


/*******************************************
 * CARE HAS TO BE TAKEN TO AVOID OVERFLOWS *
 *******************************************/

    // The value ranges of integer types are given in limits.h
    int big_int = INT_MAX;
    printf("The largest integer is: %d\n", big_int);
    // The largest integer is: 2147483647

    /*
        If the largest value is exceeded, the value is reset to the
        smallest possible value. This is called overflow, the
        source of many software bugs.
    */
    printf("The largest integer plus one is: %d\n", big_int + 1);
    // The largest integer plus one is: -2147483648
    // integer overflow

    /*
        If one goes below the smallest value of an integer type,
        integer underflow occurs and one starts counting from
        the largest value. This occurs especially often in the case
        of unsigned variables
    */
    unsigned int bank_account_total = 0;
    printf("Bank account total: %u Euros\n",
            bank_account_total); // 0
    printf("Bank account total minus one: %u Euros\n",
            bank_account_total -1);// 4294967295
    // underflow -> security issues

    /*
        For very large positive numbers, size_t is a good choice.
        It is also used for data size, as it's the return type
        of the sizeof() operator and for iterators in loops.
    */
    size_t huuuge_integer = 1000000000000000;
    printf("A really big integer %zu\n",
            huuuge_integer); // 1000000000000000

    return 0;
}
```