MACROS AND C PREPROCESSOR: A FIRST CONTACT

Description
The C preprocessor is a very simple but powerful tool in the C programming language. Every C program is processed by the preprocessor prior to compiling it. The preprocessor allows to combine files using the `#include` or to define constants and macros using `#define`.

Most often when you use the C preprocessor you will not have to invoke it explicitly: the C compiler will do so automatically. However, the preprocessor is sometimes useful by itself. It is started using the command cpp. The following program excerpt defines two macros `max` and `square` and shows how these may be called:

```
#define max(A, B)  ((A) > (B) ? (A) : (B))
#define square(x) x * x
max(a, b);
max(a+1, b+1);
square(x);
square(x+1);
```

Consider that this program is written to the file `'pp.txt'`. Note that this file is not a C program, but a small excerpt of a program to illustrate the working of the preprocessor: The line command

```
% cpp pp.txt
```

will preprocess the file `'pp.txt'` and write the result to the console.

Exercise 1
Test the above scenario on machine and explain the output. Hint: Explain what is wrong with the macro `'square'` and write a correct version.

Exercise 2
Write a macro `swap(t,x,y)` that exchanges the values of the two variables `x` and `y` assuming that both are of type `t`, e.g. `int`, and test it on machine. Hint: Use a block structure and test your macro with the program:

```
#include <stdio.h>
#define swap(t,x,y)   /* complete this macro */
main() {
   int a=1, b=2;
   swap(int,a,b);
   printf("%d %d\n", a, b);
}
```

Exercise 3 *(tricky)*
If you found a working solution for your macro `swap(t,x,y)` in the previous exercise, this solution will probably not work in the following situation:

```
(1)   if (a>b) swap(int,a,b); /* whoops */
(2)   else a = b;
```

a) Why? Hint: write down the code of line (1) once the macro has been expanded, and you will see the problem (if not, compile your code and understand the compiler's complaint).

b) Adapt the code of your macro in order that the above lines (1)-(2) become a correct C statement. Hint: the solution is very, very tricky ! Nevertheless, try to find a solution by your own, e.g. without "google search".

Exercise 4 *(tricky)*
A common approach to generating a single source code that is suitable for both development and release is done with the help of the following macro:

```
#ifdef DEBUG
   # define DEBUG_PRINT(x) printf x
#else
   # define DEBUG_PRINT(x) do {} while (0)
#endif
```

Use it like:

```
DEBUG_PRINT(("var1: %d; var2: %d; str: %s\n", var1, var2, str));
```

Write a little program to test it.

Note: the program must be compiled with the `-D` option to define `DEBUG`:

```
gcc -D DEBUG prog.c -o prog
```