A01 - Assignment 1

Student: Deidier Simone - 133020

Answers to the theory questions

1. The major pro of declaring a "*function*" as a macro, like `#define function(n) code`, is that when we "*call that function*" we are not allocating any memory in the stack for the function call activation, in fact doing that we only say to the compiler to substitute the "*function call*" in our code with the "*function definition*" we gave to the macro. This means that during the compile time the compiler has to parse all the document and do all the substitution, that can result in a slower compile time. Another important factor to remember is that they are not functions, so all the parameters passed are not copied in the stack, and during the substitution the code will work directly on the variables passed as arguments (*like functions with parameters passed with reference*).
2. We could have used the Dirichlet boundary conditions that sets the value of our function on the boundaries as a constant values (*in a domain* $[a, b]$ *we can set the boundaries values of our function* $f(x)$ *as* $f(a) = C_1$ *and* $f(b) = C_2$) othewise we could have used the Robin boundary conditions, that mixes the Dirichlet and Neumann conditions with weights (*in a domain* $[a, b]$ *we can set the boundaries values of our function* $f(x)$ *as* $w_1 \cdot f(a) + w_2 \cdot f'(a) = C_1$ *and* $w_1 \cdot f(b) + w_2 \cdot f'(b) = C_2$, *where* $f'(c) = C$ *is the Neumann boundary condition*).
3. If we don't allocate memory of the three buffers in *T1* we have ana array of three pointers that are set to *NULL* and don't point to a memory location, so we cannot have access to the memory where they are pointing. Accessing to memory of a pointer that points to *NULL* gives us a *compile error* if the compiler notices the wrong access or a *runtime error* if the compilers compile the code and during the runtime we try to access to the *NULL* location.
4. The difference between `float const* a` and `float* const a` is that in the first case the variable `a` is a pointer to a constant float value, so we canot modify the value of `a` but we can modify the lcoation to the memory where the pointer is pointing, where in the second case the variable `a` is a constant pointer to a float value, so we cannot modify the location where the pointer is pointing, but we can modify the value that there is in it.