Activation records for C on Linux (32-bit)

The contents of the stack reflect the order in which the required operations take place.

Caller stores values of any caller-save registers
Caller pushes parameters in reverse order of parameter list (last parameter first)
All values are 32-bit. Arrays, structures, etc., are passed by reference so a 32-bit address will be pushed on the stack. Anything shorter than 32 bits, such as a character, is extended to 32 bits.
The Intel architecture does not allow memory-to-memory transfers, so any values of variables used as parameters will have to be copied through a register. However constants can be pushed as immediate operands. This does not include addresses, since the absolute address is not known at compile time. The address must be calculated by adding the value of the offset to EBP.

The CALL instruction automatically pushes the return address on the stack.

The callee starts by pushing EBP and then setting EBP to the value in ESP. This sets the basis for addressing the contents of the stack frame.
The callee then pushes any additional callee-save registers that there may be. (In the sample Lab 4 code you were given, it is assumed that there is one callee-save register.)
The callee then allocates stack space for local variables in the order that they are declared.
After that the callee can make any use of the stack it wants, provided it is subsequently restored.

At the end of callee execution, local variables are de-allocated by incrementing ESP. Callee-save registers and EBP are restored by popping values back off the stack. The return value is in EAX.

The RET instruction automatically takes the return address from the stack.

The caller removes parameters from the stack, usually by incrementing ESP by the appropriate value. Then any caller-save registers are restored.