**1. pass1()**

pass1() 是組譯過程的第一遍（Pass 1），其主要目的是建立符號表（symtab），並計算各標籤的地址。以下是每個部分的功能：

* **程式起始檢測**：此處首先檢查操作碼是否為 START，用於定義程式的起始地址。若是，則會設定程式名稱（prog\_name）和起始地址（start\_addr），並將地址計數器（locctr）初始化為 start\_addr。否則，默認將 locctr 設為 0。
* **符號表構建**：每讀到一行指令，若該行有標籤（label），則會檢查該標籤是否已存在於符號表中。如果存在，則輸出錯誤訊息「重複符號」。如果不存在，則將該標籤及當前的地址存入符號表中。
* **地址計算**：對於操作碼進行查詢，如果該操作碼存在於操作碼表中，則標準指令長度增加3（指令長度固定為3）。對於其他伺服指令（例如 WORD、RESW、RESB 和 BYTE），根據指令和操作數來更新 locctr 的值，這些都會影響指令長度並調整下一行的地址。
* **程式長度計算**：當遇到 END 時，會結束 pass1 過程並計算程式的總長度（prog\_len）。

**2. wr\_header()**

wr\_header() 是用於輸出目標檔案的「標頭記錄」（Header Record），它將生成的目標文件中包含程式的名稱、起始地址和程式長度。

具體內容為：

* H + prog\_name(6位) + start\_addr(6位) + prog\_len(6位)：將程式名稱、起始地址和程式長度格式化成 H 開頭的行，並寫入到目標文件中。

**3. wr\_text()**

wr\_text() 用於輸出「文本記錄」（Text Record），它包含指令的目標代碼，並寫入至目標檔案中。文本記錄的格式為：

* T + startaddress(6位) + len(object code)(2位) + 指令目標代碼。

具體步驟：

1. 計算文本記錄的長度，即目標代碼的字節數，並將其轉換為十六進制數填入位置。
2. 將已存的文本記錄輸出到目標檔案中，並在每條指令之間插入空白字符來分隔。

**4. wr\_end()**

wr\_end() 用於在目標檔案中寫入「結束記錄」（End Record），包含程式起始地址。此記錄的格式為 E + start\_addr(6位)，其作用是告知組譯器和連結器程式的結束位置。

**5. pass2()**

pass2() 是組譯過程的第二遍，用於產生最終的目標代碼。主要流程包括：

* **標頭記錄輸出**：呼叫 wr\_header()，將標頭記錄寫入目標檔案。
* **處理每行指令**：逐行讀取指令，對操作碼進行查詢。如果是有效的操作碼，則生成該操作的目標代碼。如果指令中包含操作數，會查找符號表，若找到相應的地址，則生成目標代碼；如果未找到，則提示錯誤「未定義的符號」。針對不同的伺服指令，例如 BYTE、WORD 等，根據指令的長度生成相應的目標代碼並添加到文本記錄中。
* **文本記錄輸出**：當文本記錄滿了或者遇到新的一段指令時，呼叫 wr\_text() 將當前的文本記錄寫入目標檔案，然後重新初始化文本記錄。
* **結束記錄輸出**：最後，呼叫 wr\_end() 在目標檔案中寫入程式的結束地址。