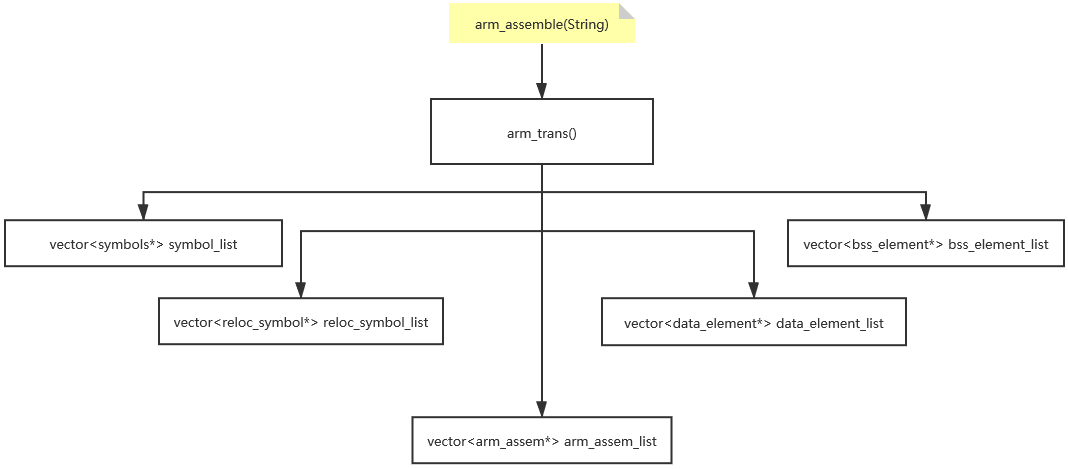
arm_analyze 模块总体设计：

读取从文件或之前生成的汇编字符串，对汇编代码的各种信息进行分析，为后续生成目标文件做准备。

模块生成 5 个列表，分别对应目标文件 5 个节区，一个列表对应一个节区，列表里有生成该节区所需的信息。

模块的整体输入输出：



每个列表的详细描述：

symbols 结构体

```
struct symbols
{
    int type;      //区分是函数还是全局变量，-1 表示 NOTYPE(未定义) 0 表示函数，1 表示全局变量，2 表示标号
    bool defined; //判断该符号是否在此文件中定义过
    string name;  //符号名称
    int value;     //如果是文件中定义的函数 或 标号 会记录函数定义的位置和标号的位置 如果是全局变量在 data 段中的偏移
    int bind;      //判断符号的作用域是 global 还是 local    0 表示 global 1 表示 local
};
```

symbol_list 存储函数及全局变量的符号，如果是函数则会记录该符号在在.text 段声明的相对偏移

在创建一个 symbols 使对其做初始化，对于可以确定的信息根据实际情况填入，不能确定的信息预定按如下规则初始化（type=-1,defined=false,value=0,bind=1）。

汇编代码中的标号比如.L0 也会在 symbol_list 中，后续模块可忽略这些标号，即判断 type==2 时可不做任何处理

reloc_symbol 结构体

```
struct reloc_symbol
{
```

```
    int type;     //需要重定位的是函数还是全局变量
    string name; //符号名称
    int type;     //需要重定位的地方在.text 中的偏移
};
```

reloc_symbol_list 存储需要重定位的符号和需要重定位的相关位置，包括未定义的函数以及需从 data 段获取的全局变量

arm_assem 结构体

```
struct arm_assem
{
    string op_name; //操作符名称 "mov" "ldr" "str"等
    string Operands1;    //操作数
    string Operands2;
    string Operands3;
    vector<string> reglist; //如果操作符是 push 或 pop   需要一个寄存器列表
};
```

arm_assem_list 存储分析好的汇编代码，里面的汇编代码与机器码处于一一对应的关系（已经将标号等替换成相对位移，对伪指令做相关处理等），使得后续模块对该链表线性遍历翻译即可得到.text 的全部内容.

Operands 需要后续模块判断是寄存器还是立即数，立即数以'#'开头，其余都表示寄存器
pop 和 push 的 Operand1,2,3 均为空值，reglist 不为空
其余指令的 reglist 为空
对于 mov 这种三操作数指令 Operands1,2,3 均不为空
cmp 这种双操作数指令 Operands3 则为空值

有一种特殊情况比如 ldr r4, =n
处理时会转化成　ldr r4, [pc , #-4]　和一条留空语句
留空语句的机器码为　0x00000000　　其实就是　.word 0x00000000
后续模块遍历到 op_name==".word"时　直接生成 0x00000000 的机器码即可

data_element 结构体

```
struct data_element
{
    string op_name; //数据声明语句 .word 或.space
    int value;       //声明数据的值
};
```

data_element 存储汇编在 data 段的数据声明语句（.word .space）
比如语句　.word 4
则 element->name="word"
element->value=4;
后续模块根据这些信息直接将数据写入 data 节区

bss_element

```
struct bss_element
{
    string name;     //未定义的变量的名称
};
```
bss 中的变量因为没有值，所以只记录变量的名称

详细设计：

需要设立一个全局变量 offset_text 用于记录当前指令或标号（或函数）在.text 中的相对位置，每处理一个正常指令（非伪指令、标号、函数），offset_text+=4，这样在处理指令时可以根据 offset 获取当前指令的位移。

还设立一个全局变量 offset_data 记录全局变量在 data 段中的偏移

arm_trans()读取汇编代码并逐行拆分，然后将每行汇编代码交给汇编指令处理模块
汇编指令处理模块对不同的指令进行处理,这里将指令分成 5 种，对各种指令的处理逻辑在函数列表中给出