一、C语言由源代码生成的各阶段如下：

C源程序－>编译预处理－>编译－>优化程序－>汇编程序－>链接程序－>可执行文件

       其中 编译预处理阶段，读取c源程序，对其中的伪指令（以#开头的指令）和特殊符号进行处理。或者说是扫描源代码，对其进行初步的转换，产生新的源代码提供给编译器。预处理过程先于编译器对源代码进行处理。

       在C 语言中，并没有任何内在的机制来完成如下一些功能：在编译时包含其他源文件、定义宏、根据条件决定编译时是否包含某些代码。要完成这些工作，就需要使用预处理程序。尽管在目前绝大多数编译器都包含了预处理程序，但通常认为它们是独立于编译器的。预处理过程读入源代码，检查包含预处理指令的语句和宏定义，并 对源代码进行响应的转换。预处理过程还会删除程序中的注释和多余的空白字符。

二、伪指令（或预处理指令）定义

      预处理指令是以#号开头的代码行。#号必须是该行除了任何空白字符外的第一个字符。#后是指令关键字，在关键字和#号之间允许存在任意个数的空白字符。整行语句构成了一条预处理指令，该指令将在编译器进行编译之前对源代码做某些转换。下面是部分预处理指令：

[复制代码](javascript:void(0);)

指令  用途  
 # 空指令，无任何效果  
 #include 包含一个源代码文件  
 #define 定义宏  
 #undef 取消已定义的宏  
 #if 如果给定条件为真，则编译下面代码  
 #ifdef 如果宏已经定义，则编译下面代码  
 #ifndef 如果宏没有定义，则编译下面代码  
 #elif 如果前面的#if给定条件不为真，当前条件为真，则编译下面代码，其实就是else if的简写  
 #endif 结束一个#if……#else条件编译块  
 #error 停止编译并显示错误信息

[复制代码](javascript:void(0);)

三、预处理指令主要包括以下四个方面：

1、宏定义指令

      宏定义了一个代表特定内容的标识符。预处理过程会把源代码中出现的宏标识符替换成宏定义时的值。宏最常见的用法是定义代表某个值的全局符号。宏的第二种用 法是定义带参数的宏（宏函数），这样的宏可以象函数一样被调用，但它是在调用语句处展开宏，并用调用时的实际参数来代替定义中的形式参数。

1.1 #define指令  
1.1.1 #define预处理指令用来定义宏。该指令最简单的格式是：声明一个标识符，给出这个标识符代表的代码（比如像圆周率这样的数）。在后面的源代码中，我们就可以使用定义的宏取代要使用的代码，举例如下：

//例1  
#define MAX\_NUM 10  
int array[MAX\_NUM];  
for(i=0;i<MAX\_NUM;i++)

    在这个例子中，对于阅读该程序的人来说，符号MAX\_NUM就有特定的含义，它代表的值给出了数组所能容纳的最大元素数目。程序中可以多次使用这个值。作为一种约定，习惯上总是**全部用大写字母来定义宏**，这样易于把程序的宏标识符和一般变量标识符区别开来。如果想要改变数组的大小，只需要更改宏定义并重新编译程序即可。

1.1.2 使用宏的好处有两点：

一是使用方便。如下：

//例2  
#define PAI 3.1415926

PAI显然比3.1415926写着方便。

二是定义的宏有了意义，可读性强。如例1，MAX\_NUM，望文生意便知是最大数量的意思，比单纯使用10这个数字可读性要强的多。

三是容易修改。如例1，如果在程序中有几十次会使用到MAX\_NUM，修改只需要在宏定义里面修改一次就可以，否则你会修改到崩溃。

1.1.3 宏表示的值可以是一个常量表达式，允许宏嵌套（必须在前面已定义）。例如：

//例3  
#define ONE 1  
#define TWO 2  
#define SUM(ONE+TWO)

这里需要注意两点：  
一是注意上面的宏定义使用了括号。尽管它们并不是必须的。但出于谨慎考虑，还是应该加上括号的。例如：  
            six=THREE\*TWO;  
    预处理过程把上面的一行代码转换成：  
            six=(ONE+TWO)\*TWO;  
    如果没有那个括号，就转换成six=ONE+TWO\*TWO;了。

**也就是说预处理仅是简单的字符替换，要时刻注意这一点，很多错误都会因此出现。**

二是虽然我们举例用了#define ONE 1 这个例子，但是一般要求宏定义要有其实际意义，#define ONE 1这种没意义的宏定义是不推荐的。（大概是这么个意思，忘记具体怎么说了）

1.1.4 宏还可以代表一个字符串常量，例如：  
            #define VERSION "Version 1.0 Copyright(c) 2003"

1.2 带参数的#define指令（宏函数）  
    带参数的宏和函数调用看起来有些相似。看一个例子：

//例4  
#define Cube(x) (x)\*(x)\*(x)

    可以时任何数字表达式甚至函数调用来代替参数x。这里再次提醒大家注意括号的使用。宏展开后完全包含在一对括号中，而且参数也包含在括号中，这样就保证了宏和参数的完整性。看一个用法：

//例4用法  
int num=8+2;  
volume=Cube(num);

    展开后为(8+2)\*(8+2)\*(8+2);  
    如果没有那些括号就变为8+2\*8+2\*8+2了。  
    **下面的用法是不安全的：**  
            volume=Cube(num++);  
    如果Cube是一个函数，上面的写法是可以理解的。但是，因为Cube是一个宏，所以**会产生副作用**。这里的书写不是简单的表达式，它们将产生意想不到的结果。它们展开后是这样的：  
            volume=(num++)\*(num++)\*(num++);  
    很显然，结果是10\*11\*12,而不是10\*10\*10;  
    那么怎样安全的使用Cube宏呢？**必须把可能产生副作用的操作移到宏调用的外面进行**：  
            int num=8+2;  
            volume=Cube(num);  
            num++;

宏函数使用不当会出现一些难以发现的错误，请慎重使用。

1.3 #运算符  
    出现在宏定义中的#运算符把跟在其后的参数转换成一个字符串。有时把这种用法的#称为字符串化运算符。例如：

[复制代码](javascript:void(0);)

//例5  
#define PASTE(n) "adhfkj"#n  
int main()  
{  
 printf("%s\n",PASTE(15));  
 return 0;  
}  
//输出adhfj15

[复制代码](javascript:void(0);)

宏定义中的#运算符告诉预处理程序，把源代码中任何传递给该宏的参数转换成一个字符串。所以输出应该是adhfkj15。

1.4 ##运算符（很少用）  
    ##运算符用于把参数连接到一起。预处理程序把出现在##两侧的参数合并成一个符号。看下面的例子：

[复制代码](javascript:void(0);)

//例6  
#define NUM(a,b,c) a##b##c  
#define STR(a,b,c) a##b##c  
int main()  
 {  
 printf("%d\n",NUM(1,2,3));  
 printf("%s\n",STR("aa","bb","cc"));  
 return 0;  
 }  
//最后程序的输出为:  
123  
aabbcc

[复制代码](javascript:void(0);)

2、条件编译指令。

      程序员可以通过定义不同的宏来决定编译程序对哪些代码进行处理。条件编译指令将决定那些代码被编译，而哪些是不被编译的。可以根据表达式的值或者某个特定的宏是否被定义来确定编译条件。

2.1 #if/#endif/#else/#elif指令  
    #if指令检测跟在制造另关键字后的常量表达式。如果表达式为真，则编译后面的代码，知道出现#else、#elif或#endif为止；否则就不编译。  
    #endif用于终止#if预处理指令。  
    #else指令用于某个#if指令之后，当前面的#if指令的条件不为真时，就编译#else后面的代码。

[复制代码](javascript:void(0);)

//例7  
#define DEBUG //此时#ifdef DEBUG为真  
//#define DEBUG 0 //此时为假  
int main()  
{  
   #ifdef DEBUG  
      printf("Debugging\n");  
   #else  
      printf("Not debugging\n");  
   #endif  
   printf("Running\n");  
   return 0;  
}

[复制代码](javascript:void(0);)

这样我们就可以实现debug功能，每次要输出调试信息前，只需要#ifdef DEBUG判断一次。不需要了就在文件开始定义#define DEBUG 0

#elif预处理指令综合了#else和#if指令的作用。

[复制代码](javascript:void(0);)

//例8  
#define TWO  
int main()  
{  
 #ifdef ONE  
 printf("1\n");  
 #elif defined TWO  
 printf("2\n");  
 #else  
 printf("3\n");  
 #endif  
}  
//输出结果是2。

[复制代码](javascript:void(0);)

2.2 #ifdef和#ifndef

这二者主要用于防止重复包含。我们一般在.h头文件前面加上这么一段：

//头文件防止重复包含  
//funcA.h  
#ifndef FUNCA\_H  
#define FUNCA\_H  
//头文件内容  
#end if

这样，如果a.h包含了funcA.h，b.h包含了a.h、funcA.h，重复包含，会出现一些type redefination之类的错误。  
#if defined等价于#ifdef; #if !defined等价于#ifndef

3、头文件包含指令。

采用头文件的目的主要是为了使某些定义可以供多个不同的C源程序使用。因为在需要用到这些定义的C源程序中，只需加上一条#include语句即可，而不必再在此文件中将这些定义重复一遍。预编译程序将把头文件中的定义统统都加入到它所产生的输出文件中，以供编译程序对之进行处理。

  #include预处理指令的作用是在指令处展开被包含的文件。包含可以是多重的，也就是说一个被包含的文件中还可以包含其他文件。标准C编译器至少支持八重嵌套包含。预处理过程不检查在转换单元中是否已经包含了某个文件并阻止对它的多次包含，这个的处理办法上面已经给出。

  在程序中包含头文件有两种格式：  
        #include <my.h>  
        #include "my.h"  
    第一种方法是用尖括号把头文件括起来。这种格式告诉预处理程序在编译器自带的或外部库的头文件中搜索被包含的头文件。第二种方法是用双引号把头文件括起 来。这种格式告诉预处理程序在当前被编译的应用程序的源代码文件中搜索被包含的头文件，如果找不到，再搜索编译器自带的头文件。  
    采用两种不同包含格式的理由在于，编译器是安装在公共子目录下的，而被编译的应用程序是在它们自己的私有子目录下的。一个应用程序既包含编译器提供的公共 头文件，也包含自定义的私有头文件。采用两种不同的包含格式使得编译器能够在很多头文件中区别出一组公共的头文件。