预处理

Wenpeng.xing@gmail.com

c文件如何编译

- .c->a.out漫游
  .c-->.i-->.s-->.o-->a.out

  .c-->.i: cpp预处理器/gcc –E
  .i-->.s: gcc编译器/gcc –S
  .s-->.o:as汇编器/gcc –c
  .o-->a.out:ld连接器/gcc

gcc是一个集成工具编译器

预处理器工作标识

- #号是预处理器开始分析工作的标志
- 如：

```
#include <stdio.h>
#define  NULL ((void *)0)
#ifdefine  DEBUG
#else
#endif
#undef
...
```

预处理步骤

1. 三联符替换成相应的单字符；
   如：??=相当于#
2. 把用\续行的多行代码接成一行；
   如：#define STR "hello, "\
   "world"
   要求\后紧跟换行符，不能有其它字符
3. 把注释替换成空格；
4. 划分token和空白字符;
   如：#,define,空格,STR,空格，"hello, ",tab,tab,"world";

预处理步骤

5.在token中寻找#号预处理指示，作出相应的解释；
如：#include包含头文件，宏展开；

6.找出相应的字符常量和转意序列，用相应的字节来替换；
如：\n替换为0x0a;

7.把相邻的字符串连接起来；
如："hello, "和"world"-->"hello, world;

预处理步骤

8.把空白字符丢掉，把**token**交给**C**编译器做语法解析；
如：丢弃的空白字符包括空格，**tab**，换行等；

注意：把一个预处理指示写成多行需用\，因为一个预处理指示只能有一个逻辑代码行构成，**C**代码写成多行则不用\，因为换行在**C**代码中只不过是一种空白字符。

宏函数的意义

#define MAX(a,b) ((a)>(b)?(a):(b))

1.宏函数在预处理时展开，省去函数调用时的开销，提供代码执行效率；

2.简洁；

宏函数的负面影响

1.没有类型检测，只做形式上的完全替换；

2.多次调用宏函数生成的代码体积较大，而真正函数调用的代码只是一条**call**指令；

3.宏函数中的括号使用要格外小心；

如：**#define MAX(a,b) (a>b?a:b)**

a=0xf&I b=0xf&j

4.**MAX(a++,b++)**问题；

宏函数中的循环

```
#define  wake_up(val)  while(val){...}

#define  wake_up(val)  do{...}while(val)
```

结合语境考虑：

```
If (val>0)
    wake_up(val);
else
    sleep(val);
```

宏的重复定义

#define NUM (1 – 1)

#define NUM (1–1)

这种情况被预处理器视为重复定义一个宏；

取消宏定义：

#define NUM (1 – 1)

#undef NUM

#define NUM (1–1)

预处理中的#

- 宏定义中的#号用于创建字符串
  如：

  #define PRINT(x) printf(#x "=%d\n",x)

PRINT(i/j)转换后为**printf(“i/j” "=%d\n",i/j)**

预处理中的#

思考：

```
#define  STR(s)   #s
```

调用宏：

```
fputs(STR(strncmp("ab\"c\0d", "abc",
'\4"")== 0) STR(: @\n), s);
```

转换后的结果什么？

预处理中的##

- 宏定义中##可以把两个**token**连接成一个**token**；
  如：
  #define CONCAT(a, b) a##b
  CONCAT(con, cat) → concat
  #define D_HASH # ## #
  定义出两个##

宏定义辨析

翻译如下定义：

```
#define  sh(x)  printf("n" #x
"=%d,or %d\n",n##x,alt[x])
#define sub_z  26
sh(sub_z)
```

条件处理

#if
#ifdef
#ifndef
#if define
#if !define
#elif
#else
#endif