编译预处理

C编译程序的程序预处理功能是C语言特有的功能,C源程序中**以井号"#"开头的行就是预处理指令**.
预处理指令不是C语言的语法成分,当源文件进行编译时,系统将自动调用**预处理程序**对源程序中的预处理指令进行处理,处理完毕后再==由**编译程序**对预处理后的源程序进行编译.

文件包含

文件包含有两个形式:
`#include<filename>` 和 `#include"filename"`

- 尖括号或双引号中是要被包含的文件名字,预处理程序用指定文件中的全部内容替换源文件中的#include指令.
- 尖括号指在**系统标准目录**中查找被包含文件;双引号指先在**源文件或工程文件**所在目录中寻找被包含文件,若找不到再到标准目录中查照.
- 标准目录是系统头文件所在目录,它是系统默认的搜索路径.一般而言,诸如stdio.h等系统提供的头文件时,用尖括号;包含程序员自己的头文件时用双引号.
  如果指定了路径名,则两者没有区别,如 `#include<d:\ctest\my_file.h>` .
- #include指令通常置于源文件首部,故其包含的文件被称为"**头文件**",C编译系统提供的头文件拓展名为h,设计中也可以自行确定头文件的后缀,名称和位置.
- 头文件的内容通常包含**extern外部变量声明,#define指令,enum枚举类型声明,struct结构类型声明,union联合类型声明,typedef类型定义和函数原型**,这些都是不可执行代码,而是编译器用于产生可执行代码所需的信息,可执行代码在源文件中.
- 头文件stdio.h中有:EOF,getchar函数和putchar函数的宏定义,描述文件信息的FILE结构声明,size_t类型的typedef定义,I/O函数原型等.

- C语言的标准库函数(如 `printf()`，`scanf()`，`strcpy()` 等)通常是由编译器或操作系统提供的,它们的源代码通常不是公开的.这些库函数的实现可能是以编译器的形式提供的,也可能是作为操作系统的一部分.函数的**声明通常位于头文件**中，而函数的**定义则通常在库文件**中(例如msvcrt.dll).
- 当一个C程序由多个源文件组成时,**文件包含**可以将多个源文件共有的符号常量定义,宏定义,extern声明,类型声明,函数原型声明等集中在一起,单独组成一个头文件,然后在每个需要定义和声明的源文件中用#define包含这个头文件.

宏定义

无参宏定义

- 无参宏定义的一般形式为: `#define 标识符 字符串` .
  - 标识符是宏的名字,字符串可以是常数,表达式,格式串等任意字符序列;
  - 宏定义默认是一行,串太长需要换行,可以加一个**反斜线字符**"\"再换行;
  - 预处理程序对源文件中所有出现的宏名,都用宏定义中的串去代换,这称为**宏替换或宏展开**,宏替换是用由预处理程序自动完成的.符号常量的定义是简单的宏定义;
  - 宏名的作用域是从#define定义之后直到宏定义所在文件结束.
  - 无参宏定义不限于符号常量的定义,如 `#define EQ ==` .
- 预处理时只是简单的用串替代宏名,不进行任何检查.例如 `#define PI 3.14159;` ,这里多加了分号,则 `area=PI*r*r;` 进行宏替换后将变为 `area=3.14159;*r*r;` .在编译时会提醒这个赋值语句存在语法错误.
- 若宏名出现在双引号中,则预处理程序不对其进行宏替换.例如 `printf("2*PI=%d",2*PI);` 宏替换后变为 `printf("2*PI=%d",2*3.14159);` .

带参宏定义

- 宏可以有参数,带参宏定义的一般形式为: `#define MACRO_NAME(parameter_list) replacement_code` ,其中MACRO_NAME 是宏的名称;parameter_list 是宏的形参列表,以逗号分隔,可以为空但它们本质上不同于函数的形参;replacement_code 是宏的替换代码,在宏被调用时会被替换到代码中.
- 宏调用时要给出**实际参数**,宏展开时首先用replacement_code替换宏名,然后用实参替换形参.
  E.G.

```
#define SQUARE(x) ((x)*(x))
#define MIN(x,y) (((x)<(y)?(x):(y)))
#define AREA(r) (3.14159*SQUARE(r))
```

`SQUARE(a+1)` 会替换为 `((x)*(x))` 再用实参替换为 `((a+1)*(a+1))` .可以看出带参数宏定义既要进行串替换,又要进行参数替换.

- 注意:在带参数的宏定义中,如果串是一个含有运算符的表达式,则**串的每一个参数都必须用括号括起来,字符串外也要加括号**.因为,宏在预处理程序中只会进行**替换**,不会进行运算.
  E.G.
  `#define SQUARE(x) x*x` 时,若程序引用 `SQUARE(3+2);` ,则被替换为$2+3*2+3$
  . `#define SQUARE(x) (x)*(x)` 时,引用 `27/SQUARE(3)` ,则被替换为$27/(3)*(3)$.
- 带参宏引用在形式上类似于函数调用.宏引用是在编译前将宏名用定义中的串替换,形参用实参替换,实参与形参是**替换关系**,而不是**传递关系**.宏的参数在**预处理阶段就被替换成了对应的值**,而不是在运行时使用变量进行传递和计算.因此,它们**不会在内存中存储**.
  - 函数调用是在程序运行时将控制转移到被调用函数的代码处执行,实参与形参是**赋值关系**.
  - 函数定义时对函数的值和形参都有**类型声明**,而宏定义不用进行类型说明,可以是各种数据类型,但是实参必须有数据类型.
  - 带参数的宏定义不需要进行数据类型说明是因为宏是一种**文本替换机制**,而不是函数.使用带参数的宏时,预处理器会简单地将宏的参数替换为宏调用时提供的实际参数,并且在展开宏时不会对参数进行类型检查或转换.
  - 宏执行速度比函数快,因为它没有调用,返回,参数传递.但是宏替换的结果会使源程序代码增长,占用更多存储空间.

```c
#include<stdio.h>
#define PRINT_ARR(array,start,end) for(int i=start;i<=end;i++) \
printf("%d",array[i]);
int main()
{
    int a[5]={1,2,3,4,5};
    PRINT_ARR(a,0,4);//不必进行数据类型说明，可以是各种类型
    return 0;
}
```

宏引用 `PRINT_ARR(a,0,4);` 在展开后得到如下语句: `for(int i=0;i<=4;i++) printf("%d",a[i]);`

- 带参宏在系统上以及广泛应用.getchar和putchar都可以用宏实现:

```c
#define getchar() getc(stdin)
#define putchar(c) putc((c), stdout)
```

stdin和stdout分别是编译系统预定的标准输入流和标准输出流的FILE结构指针.

取消宏定义

要终止宏名的作用域,可以使用 `#undef` 指令: `#undef 标识符` .
标识符是由 `#define` 指令定义过的宏名

条件编译

#if指令

```
#if constant_expression
    // code to include if constant_expression is true
#elif constant_expression2
    // code to include if constant_expression2 is true
#else
    // code to include if none of the above constant expressions are true
#endif
```

1. 常量表达式必须是**整型**,并且不能包含sizeof和强制类型转换运算符以及枚举常量.因为constant_expression 在预处理指令 #if 中是要求其在**编译时即可计算出结果的表达式**.因此,constant_expression 必须是一种编译时可知的,固定的值,而不能依赖于运行时才能确定的值.程序段中可以包含预处理行.
2. #if 指令的功能是依次检测每个常量表达式的值,若某个常量表达式非零,则将其后面的程序段加入源文件,其他程序段**不包含进来(不占用内存)**.若所有常量表达式都是0,则将#else后面的程序段包含进来.
3. 选择结构if语句与条件编译#if指令的**生命周期不同**,前者会对整个源程序进行编译;后者在预处理阶段起作用,只编译其中一段代码,生成的目标程序较短.
4. #if需要一个**明确的结束符号#endif**,否则系统不知道条件编译的范围.
5. #elif指令的意义与else if类似.

```
#define DEBUG_MODE 1
#if DEBUG_MODE//调试阶段
    printf("Debug mode is enabled\n");
#else//发布阶段
    printf("Debug mode is disabled\n");
#endif
```

#ifdef指令与#ifndef指令

1. #ifdef指令一般形式为:

```
#ifdef macro_name
    // 如果宏已定义，则执行这部分代码
#endif
```

- 其中macro_name是一个宏的名称.如果**在当前位置或之前的代码中已经使用 #define 定义了该宏**,则#ifdef指令下面的代码块将会被编译.如果该宏没有被定义,则#ifdef指令下面的代码块将被忽略,不会被编译.
- #ifdef 指令通常用于在不同的编译配置下包含或排除不同的代码,从而实现条件编译.

2. #ifndef指令一般形式为:

```
#ifdef macro_name
    // 如果宏没有定义，则执行这部分代码
#endif
```

define运算符

`define（标识符）`用来判断标识符**是否被#define定义过**.

```c
#include <stdio.h>
// 定义操作系统宏
#if defined(_WIN32) || defined(_WIN64)//_WIN64 是 Windows 平台上的一个预定义宏
    #define OS_WINDOWS
#elif defined(__linux__)
    #define OS_LINUX
#elif defined(__APPLE__)
    #define OS_MACOS
#else
    #error "Unsupported operating system"
#endif
int main() {
    // 根据操作系统输出不同的消息
    #if defined(OS_WINDOWS)
        printf("Hello from Windows!\n");
    #elif defined(OS_LINUX)
        printf("Hello from Linux!\n");
    #elif defined(OS_MACOS)
        printf("Hello from macOS!\n");
    #endif

    return 0;
}
```