计算机程序设计基础(1)
--- C语言程序设计(6)

孙甲松
sunjiasong@tsinghua.edu.cn
电子工程系 信息认知与智能系统研究所
罗姆楼6-104
电话: 13901216180/62796193
2022.10.

第6章 编译预处理

6.1 文件包含命令

6.2 条件编译命令

6.3 #pragma

6.4 #line

编译预处理

● 编译预处理是指C语言编译系统进行编译时，首先对程序模块中的编译预处理命令进行处理。

C 语言提供的编译预处理命令主要有以下五种：

1） #define 宏定义

2） #include 文件包含命令

3） #if… 条件编译命令

4） #pragma 杂注

5） #line

编译预处理命令一般是放在函数体的外面。

在C语言中，为了与一般的C语句相区别，所有的编译预处理命令都是以"#"开始的。

6.1 文件包含命令

- 一个C语言程序可以由多个函数组成。一个C程序中的多个函数模块可以放在同一个文件中，也可以将各函数模块分别放在若干个文件中。C语言的这种机制有利于进行模块化程序设计。
- 文件包含是指一个源文件可以将另一个指定的源文件包括进来。文件包含命令的一般形式为:

```
#include <文件名>
```

或

```
#include "文件名"
```

其功能是将指定文件中的全部内容读(插入)到该命令所在的位置后再一起被编译。

例如，文件file1.h的内容如下：

```
int  x，y，z;
float a，b，c;
char c1，c2;
```

文件file2.c的内容如下：

```
#include "file1.h"
main( )
{
  …
}
```

- 在对文件file2.c进行编译处理时，将首先对其中的#include命令进行“文件包含”处理，将文件file1.h中的全部内容插入到文件file2.c中的

#include "file1.h"

处代替此行，也可以说是将文件file1.h中的内容包含到文件file2.c中。

经过编译预处理后，最终实际编译的内容为:

```
int  x，y，z;
float a，b，c;
char c1，c2;
main( )
{
  …
}
```

- 在文件包含命令中，

如果被包含的文件名是用双撇号"括起来的，则编译系统首先在源程序文件所属的文件目录中寻找所包含的文件。如果没有找到，再按系统规定的标准方式检索其他目录；

如果被包含的文件名是用尖括号即小于号< 与大于号> 括起来的，则编译系统直接按系统规定的标准方式检索文件目录寻找所包含的文件。

因此，使用双撇号的**#include**命令的检索路径包含了使用尖括号的**#include**命令的检索路径。

- 头文件

- 在C编译系统中，有许多以.h为扩展名的文件，这些文件一般被称为头文件。在这些头文件中，对相应函数的原型与符号常量等进行了说明和定义。因此，如果要在程序中使用C编译系统提供的库函数，则在源程序的开头应包含相应的头文件。例如，如果在一个程序模块中要用到输入或输出函数时，则在该程序模块前要用如下的包含命令将相应的头文件包含进来：

```
#include  <stdio.h>
```

- 使用不同的C库函数，将需要包含不同的头文件。在本书的附录B中，列出了一些常用头文件中所包含的库函数。

```
#include  <math.h>
#include  <string.h>
#include  <stdlib.h>
#include  <conio.h>
```

在使用文件包含命令时，要注意以下几个问题：

(1) 当#include命令指定的文件中的内容改变时，包含这个头文件的所有源文件都应该重新进行编译。

(2) 一个#include命令只能指定一个被包含文件，如果需要包含多个文件，则要用多个#include命令实现。

(3) 被包含的文件应该是源文件，不能是经编译后的目标文件。

(4) 文件包含可以嵌套使用，即被包含的文件中还可以使用#include命令。但不能出现递归包含，也就是A文件用#include命令文件包含B文件，则B文件不能通过#include命令直接或间接文件再包含A文件。

(5) 由#include命令所指定的文件中可以有任何语言成分，因此，通常可以将经常使用的、具有公用性质的符号常量、带参数的宏定义以及外部变量等集中起来放在这种头文件中，以尽量避免一些重复操作。

(6) 一般不要用#include命令引用.c文件，这会被人鄙视的……

6.2 条件编译命令

一般情况下，C源程序中的所有命令行与语句都要进行编译。如果希望对C源程序中的部分内容只在满足一定条件时才进行编译；或者希望当满足某条件时对一部分语句进行编译，而当条件不满足时对另一部分语句进行编译。这就是“条件编译”。C语言的编译预处理程序提供了条件编译能力，以便使同一个源程序在不同的编译条件下能够产生不同的目标代码文件。便于程序在不同平台（操作系统）上的移植（porting），使程序具有通用性和普适性。

条件编译命令有以下几种形式：

#ifdef，#else，#endif

其一般形式为：

```
#ifdef 标识符
  程序段1
#else
  程序段2
#endif
```

其作用是，如果“标识符”已经定义过（一般是指用#define命令定义过），则程序段1参加编译，而程序段2不参加编译；否则（即“标识符”没有定义过）程序段2参加编译，而程序段1不参加编译。其中程序段1和程序段2均可以包含任意条语句（不需要用花括号括起来）。

#else部分可以省略，即可以写为：

```
#ifdef 标识符
    程序段1
#endif
```

其作用是，如果“标识符”已经定义过，则程序段1参加编译，否则程序段1不参加编译。

【例6-1】　有下列C程序：

```
#define  LOW      1
#include  <stdio.h>
main( )
{ char ch;
  printf("input  ch:");
  scanf("%c",  &ch);
#ifdef   LOW
  if (ch>='A' && ch<='Z')
            ch=ch - 'A' + 'a';     /*大写字母转换成小写字母*/
#else
  if (ch>='a' && ch<='z')
            ch=ch - 'a' + 'A';     /*小写字母转换成大写字母*/
#endif
  printf("%c\n",  ch);
}
```

标识符

这个程序的功能是，对于由键盘输入的字符，将英文大写字母转换成小写字母，其他字符不变。

在上述程序中，由于开头有一个宏定义命令

```
#define  LOW  1
```

即定义了一个常量LOW，这个常量表示什么是无关紧要的（在现在的程序中为1）。甚至如下形式：

```
#define  LOW
```

LOW未定义为任何内容，但是LOW已经被定义过了，这会使得 #ifdef LOW 为真。因此，条件编译命令中的程序段1（即大写字母转换成小写字母的程序段）：

```
if (ch>='A' && ch<='Z')
        ch=ch - 'A' + 'a';
```

被保留并参加编译，而程序段2（即小写字母转换成大写字母的程序段）

```
if (ch>='a' && ch<='z')
        ch=ch - 'a' + 'A';
```

被舍弃。

在这种情况下，编译预处理后，C编译系统相当于编译了如下的C源程序：

```
#include <stdio.h>
main( )
{ char ch;
  printf("input ch:");
  scanf("%c", &ch);
  if (ch>='A' && ch<='Z')
          ch=ch - 'A' + 'a';
                /* 大写字母转换成小写字母 */
  printf("%c\n", ch);
}
```

如果将例6-1源程序中的宏定义命令去掉，即程序变为：

```
#include  <stdio.h>
main( )
{ char ch;
  printf("input ch:");
  scanf("％c"，  &ch);
#ifdef   LOW
  if (ch＞＝'A' && ch＜＝'Z')
            ch=ch - 'A' + 'a';     /*大写字母转换成小写字母*/
#else
  if (ch＞＝'a' && ch＜＝'z')
            ch=ch - 'a' + 'A';     /*小写字母转换成大写字母*/
#endif
  printf("％c\n"，  ch);
}
```

程序的功能变为：将键盘输入的小写字母转换成大写字母，其他字符不变，然后输出。

此段会被舍弃

编译预处理后，源程序变为：

```
#include  <stdio.h>
main( )
{ char ch;
  printf("input ch:");
  scanf("%c",  &ch);
  if (ch>=’a’ && ch<=’z’)
        ch=ch - ’a’ + ’A’;    /*小写字母转换成大写字母*/
  printf("%c\n",  ch);
}
```

- 条件编译命令利用“标识符”是否定义作为条件，在两个程序段中选择一个进行编译。
- 条件编译不同于根据条件选择执行不同的程序段。
- 选择结构中的各程序段不管最后是否被执行，都需要进行编译，形成的目标程序就会很长。而且，在实际运行时，要对条件进行测试后才能决定执行哪个程序段，因而运行时间也长。
- 采用条件编译命令来处理，由于在编译过程中就根据条件决定对哪一段程序进行编译，另外的程序段就不参加编译了，从而减少了实际被编译的语句，也减少了目标程序的长度。并且，在实际执行过程中不必再测试条件，减少了运行时间，提高了程序的执行效率。
- 当条件编译段比较多时，会大大提高程序的运行效率。

#ifndef，#else，#endif

其一般形式为:

```
#ifndef 标识符
  程序段1
#else
  程序段2
#endif
```

其作用是，如果“标识符”没有定义过，则程序段1参加编译，而程序段2不参加编译；否则（即“标识符”定义过）程序段2参加编译，而程序段1不参加编译。程序段1和程序段2均可以包含任意条语句（不需要用花括号括起来）。

#else部分可以省略，即可以写为:

```
#ifndef 标识符
  程序段1
#endif
```

其作用是，如果“标识符”没有定义过，则程序段1参加编译，否则程序段1不参加编译。

这种形式的条件编译命令与上一种类似，只是条件刚好相反，在实际应用中，可以根据具体情况任选一种。

将例6.1程序中的条件编译命令#ifdef改成#ifndef，即程序改成如下：

```
#define  LOW  1
#include <stdio.h>
main( )
{ char ch;
  printf("input ch:");
  scanf("%c", &ch);
#ifndef  LOW
  if (ch>='A' && ch<='Z')
        ch=ch - 'A' + 'a'; /* 大写字母转换成小写字母 */
#else
  if (ch>='a' && ch<='z')
        ch=ch - 'a' + 'A'; /* 小写字母转换成大写字母 */
#endif
  printf("%c\n", ch);
}
```

这个程序的功能是，对于由键盘输入的字符，将英文小写字母转换成大写字母，其他字符不变。

编译预处理后，源程序变为：

```
#include <stdio.h>
main( )
{ char ch;
  printf("input ch:");
  scanf("%c", &ch);
  if (ch>='a' && ch<='z')
          ch=ch - 'a' + 'A';
  printf("%c\n", ch);
}
```

#if，#else，#endif, #elif

一般形式为：

```
#if 常量表达式
  程序段1
#else
  程序段2
#endif
```

其作用是，如果常量表达式的值为“真”（值非0），则程序段1参加编译，而程序段2不参加编译；否则（即常量表达式的值为0）程序段2参加编译，而程序段1不参加编译。程序段1和程序段2均可以包含任意条语句（不需要用花括号括起来）。

其中，#else部分可以省略，即

```
#if 常量表达式
  程序段1
#endif
```

其作用是，如果常量表达式的值为“真”（值非0），则程序段1参加编译，否则程序段1不参加编译。

#if，#else，#endif，#elif

其一般形式为：

```
#if 常量表达式1
  程序段1
#elif 常量表达式2
  程序段2
……
#elif 常量表达式n
  程序段n
#else
  程序段n+1
#endif
```

其作用是，如果常量表达式1的值为“真”（值非0），则程序段1参加编译，而其余程序段不参加编译；否则如果常量表达式2的值为“真”（值非0），则程序段2参加编译，而其余程序段不参加编译；…….以此类推，最终可能程序段n+1参加编译。

最终是从n+1个程序段中选择一段参加编译，生成相应可执行程序代码。

注意将：#else if 合并写为 #elif

如果将例**6.1**程序中的条件编译命令**#ifdef**改成**#if**，即程序改成：

```
#define  LOW  1
#include <stdio.h>
main( )
{ char ch;
  printf("input ch:");
  scanf("%c", &ch);
#if  LOW
  if (ch>='A' && ch<='Z')
        ch=ch - 'A' + 'a'; /* 大写字母转换成小写字母 */
#else
  if (ch>='a' && ch<='z')
        ch=ch - 'a' + 'A';/* 小写字母转换成大写字母 */
#endif
  printf("%c\n", ch);
}
```

这个程序的功能是，对于由键盘输入的字符，将英文大写字母转换成小写字母，其他字符不变。

编译预处理后，源程序变为：

```
#include <stdio.h>
main( )
{ char ch;
  printf("input ch:");
  scanf("%c", &ch);
  if (ch>='A' && ch<='Z')
        ch=ch - 'A' + 'a';
  printf("%c\n", ch);
}
```

条件编译还用来防止多个文件引用同一个头文件时，出现多重定义同一个外部变量或说明的问题。

例如，有头文件**EXAMPLE.h**：

```
/* EXAMPLE.h - Example header file */
 #if !defined(EXAMPLE_H) 或者写为: #ifndef EXAMPLE_H
  #define EXAMPLE_H
  struct Example { ... };
 #endif
```

若多个**C**文件都文件包含头文件**EXAMPLE.h**，加上此条件编译后，每个**C**文件编译时，首先判断开关量**EXAMPLE_H**是否已经宏定义过了，若没定义过，则宏定义**EXAMPLE_H**，同时说明结构体**Example**。一旦某个文件引用过**EXAMPLE.h**，则此开关量**EXAMPLE_H**已经宏定义过了，结构体**Example**也说明过了。若此文件再出现包含文件**EXAMPLE.h**，则不会再重复定义**EXAMPLE_H**，也不会重复说明结构体**Example**。

#undef

其一般形式为：

#undef 标识符

其作用是，将已经定义的标识符变为未定义。

例如：

```
#define  WIDTH          80
#define  ADD( X, Y )    ((X) + (Y))
  . . .
#undef   WIDTH
#undef   ADD
```

6.3 #pragma命令

#pragma一般形式为：

#pragma token-string

其中token-string有多种，像alloc_text, auto_inline, bss_seg, check_stack, code_seg, const_seg, comment, component, data_seg, function, hdrstop, include_alias, init_seg1, inline_depth, inline_recursion, intrinsic, message, once, optimize, pack, pointers_to_members1, setlocale, vtordisp1, warning等。

其作用是，指示编译器如何进行编译，比如如何处理某一类warning错误信息，如何处理某文件被多次include，如何进行内存存放处理，比如紧缩方式pack等等。本章将仅讲once、warning和pack的使用。而且pack涉及到结构体的成员对齐方式，将留到第11章再讲。

- **#pragma once**

作用是让编译器把指定的文件只包含一次，防止此文件被多次引用出现的重复定义等错误。通常放在头文件的开始处。

例如：你打开文件stdio.h，会看到开头几行为：

```
#if  _MSC_VER > 1000
#pragma once
#endif
```

其作用是当一个文件多次include文件stdio.h时，让编译器把文件stdio.h只文件包含一次，防止多次引用出现的重复定义等错误。

- **#pragma warning(disable:4996)**

作用是将4996类警报置为失效，让编译器不再显示这类警告。

例如，对于程序：

```
#include<stdio.h>
main( )
{ int x;
  scanf("%d",&x);
  printf("%d\n",x);
}
```

编译结果：

test.c(4) : warning C4996: 'scanf': This function or variable may be unsafe. Consider using scanf_s instead. To disable deprecation, use _CRT_SECURE_NO_WARNINGS. See online help for details.

1> c:\program files (x86)\microsoft visual studio 9.0\vc\include \stdio.h(306) : 参见“scanf”的声明

程序开头加上：**#pragma warning(disable:4996)**

```
#include <stdio.h>
#pragma warning(disable:4996)
main( )
{ int x;
  scanf("%d",&x);
  printf("%d\n",x);
}
```

将不会再出现这类警告性错误，对于**VS2012**以后的版本尤其重要，否则无法通过编译。

根据编译时给出的 **warning C4996:** 警告性错误编号，也可以关闭类似**6031**（函数返回值）等其他警告错误。

6.4 #line命令

#line一般形式为

#line 数字 ["文件名"]

其中"文件名"是任选项。其作用是让编译器编译显示错误信息时，改变当前所显示的行号和文件名，便于调试。

例如：在文件test.c中，插入

#line 151

从此行后，编译信息显示将是test.c的151行开始的计数，实际上尽管#line 151所在的行可能是第1行。

而在文件test.c中，插入

#line 151 "copy.c"

从此行后，编译信息显示将是copy.c的151行开始的计数。

例如有程序：

```
#include <stdio.h>
main( )
{ int x,y;
  char c,d;
  scanf("%d%d", &x, &y)
  scanf("%c%c", &c, &d);
  printf("%d,%d,%c,%c", x, y, c, d)
}
```

编译结果是：

test.c(6) : error C2146: 语法错误 : 缺少 “;”(在标识符 “scanf”的前面)

test.c(8) : error C2143: 语法错误 : 缺少 “;”(在 “}”的前面)

如果在文件test.c中，插入 #line 151 行：

```
#include<stdio.h>
main( )
{ int x,y;
  char c,d;
#line 151
  scanf("%d%d",&x,&y)
  scanf("%c%c",&c,&d);
  printf("%d,%d,%c,%c",x,y,c,d)
}
```

编译结果是：

```
test.c(152) : error C2146: 语法错误 : 缺少“;”(在标识符“scanf”的前面)
test.c(154) : error C2143: 语法错误 : 缺少“;”(在“}”的前面)
```

从#line行后，编译信息显示将是test.c的151行开始的计数，实际上尽管#line 151所在的行可能是第1行。

如果在文件test.c中，插入 #line 151 "copy.c" 行：

```
#include<stdio.h>
main( )
{ int x,y;
  char c,d;
#line 151  "copy.c"
  scanf("%d%d",&x,&y)
  scanf("%c%c",&c,&d);
  printf("%d,%d,%c,%c",x,y,c,d)
}
```

编译结果是：

```
copy.c(152) : error C2146: 语法错误 : 缺少 “;”(在标识符 “scanf”的前面)
copy.c(154) : error C2143: 语法错误 : 缺少 “;”(在 “}”的前面)
```

从#line行后，编译信息显示将是copy.c的151行开始的计数。
虽然根本就没有copy.c这个源文件。