16 C预处理器和C库

内容提要

- 翻译程序的第一步
- 明示常量：#define
- 在#define中使用参数
- 宏和函数的选择
- 文件包含：#include
- 其他指令
- 内联函数(C99)
- _Noreturn函数(C11)
- C库
- 数学库
- 通用工具库
- 断言库
- string.h库中的memcpy()和memmove()
- 可变参数：stdarg.h

翻译程序的第一步

1 翻译程序的第一步

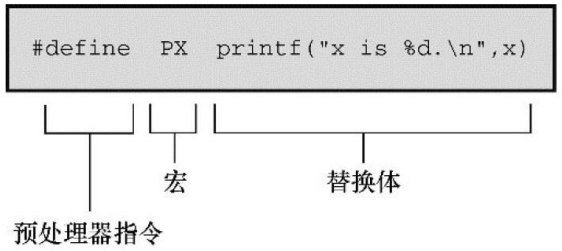
- 在预处理之前，编译器必须对该程序进行一些翻译处理
- 首先，编译器把源代码中出现的字符映射到源字符集。该过程处理多字节字符和三字符序列一字符扩展让C更加国际化
- 第二，编译器定位每个反斜杠后面跟着换行符的实例，并删除它们
- 第三，编译器把文本划分成预处理记号序列、空白序列和注释序列

```
1. printf("That's wond\
2.     erful!\n");
3. printf("That's wonderful\n!");


4. int/* 这看起来并不像一个空格*/fox;
5. int fox;
```

明示常量：#define

2 明示常量：#define

- 程序清单16.1 preproc.c
- 使用#define指令来定义明示常量(manifest constant)(也叫作符号常量)
- 预处理器指令从#开始运行，到后面的第1个换行符为止。也就是说，指令的长度仅限于一行
- 每行#define(逻辑行)都由3部分组成
  - 第1部分是#define指令本身
  - 第2部分是选定的缩写，也称为宏。有些宏代表值，被称为类对象宏(object-like macro)
  - 第3部分(指令行的其余部分)称为替换列表或替换体



- 一旦预处理器在程序中找到宏的示实例后，就会用替换体代替该宏
  - 也有例外，稍后解释
  - 从宏变成最终替换文本的过程称为宏展开(macro expansion)
- 预处理器不做计算，不对表达式求值，它只进行替换

```
1.  /* preproc.c -- simple preprocessor examples */
2.  #define TWO 2
3.  #define OW "Consistency is the last refuge of the
    unimaginetive. - Oscar Wilde"
4.  #define FOUR  TWO*TWO
5.  #define PX printf("X is %d.\n", x)
6.  #define FMT  "X is %d.\n"

7.  int main(void)
8.  {
9.      int x = TWO;
10.
11.     PX;
12.     x = FOUR;
13.     printf(FMT, x);
14.     printf("%s\n", OW);
15.     printf("TWO: OW\n");
16.
17.     return 0;
18. }
```

2.1 记号

- 从技术角度来看，可以把宏的替换体看作是记号（token）型字符串，而不是字符型字符串
- 替换体中有多个空格时，字符型字符串和记号型字符串的处理方式不同
  - 解释为字符型字符串，把空格视为替换体的一部分
  - 解释为记号型字符串，把空格视为替换体中各记号的分隔符

```
1. #define FOUR 2*2
2. #define SIX 2 * 3
3. #define EIGHT 4 * 8
```

2.2 重定义常量

- 如果需要重定义宏，使用#undef指令
- 如果确实需要重定义常量，使用const关键字和作用域规则

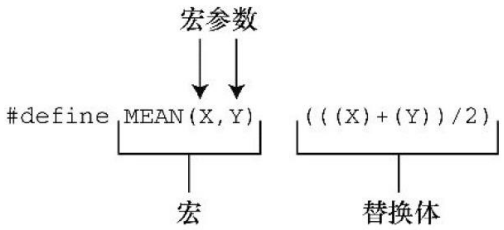
```
1. #define SIX 2 * 3

2. #define SIX 2*3
```

在#define中使用参数

3 在#define中使用参数

程序清单16.2 mac_arg.c

- 类函数宏定义的圆括号中可以有一个或多个参数，随后这些



- 看上去像函数调用，但是它的行为和函数调用完全不同
  - SQUARE(x+2) 展开为 x+2*x+2
    - #define SQUARE(x) (x)*(x)
  - 100/SQUARE(2) 展开为 100/2*2
    - #define SQUARE(x) ((x)*(x))

```
1.  #define SQUARE(X) X*X
2.  #define PR(X)   printf("The result is %d.\n", X)
3.  int main(void){
4.      int x = 5, z;
5.      z = SQUARE(x);
6.      printf("Evaluating SQUARE(x): ");
7.      PR(z);
8.      z = SQUARE(2);
9.      printf("Evaluating SQUARE(2): ");
10.     PR(z);
11.     printf("Evaluating SQUARE(x+2): ");
12.     PR(SQUARE(x+2));
13.     printf("Evaluating 100/SQUARE(2): ");
14.     PR(100/SQUARE(2));
15.     printf("x is %d.\n", x);
16.     printf("Evaluating SQUARE(++x): ");
17.     PR(SQUARE(++x));
18.     printf("After incrementing, x is %x.\n", x);
19.     return 0;
20. }
```

3.1 用宏参数创建字符串：#运算符

- 在类函数宏的替换体中，#号作为一个预处理运算符，可以把记号转换成字符串
- 如果x是一个宏形参，那么#x就是转换为字符串"x"的形参名。这个过程称为字符串化（stringizing）
- 程序清单16.3 subst.c

```c
/* subst.c -- substitute in string */
#include <stdio.h>
#define PSQR(x) printf("The square of " #x " is %d.\n",((x)*(x)))

int main(void)
{
    int y = 5;

    PSQR(y);
    PSQR(2 + 4);

    return 0;
}
```

3.2 预处理器黏合剂：##运算符

- ##运算符把两个记号组合成一个记号
- #define XNAME(n) x ## n
  - 宏XNAME(4)将展开为x4
- 程序清单16.4 glue.c

```
1.  // glue.c -- use the ## operator
2.  #include <stdio.h>
3.  #define XNAME(n) x ## n
4.  #define PRINT_XN(n) printf("x" #n " = %d\n", x ## n);

5.  int main(void)
6.  {
7.      int XNAME(1) = 14;  // becomes int x1 = 14;
8.      int XNAME(2) = 20;  // becomes int x2 = 20;
9.      int x3 = 30;
10.     PRINT_XN(1);  // becomes printf("x1 = %d\n", x1);
11.     PRINT_XN(2);  // becomes printf("x2 = %d\n", x2);
12.     PRINT_XN(3);  // becomes printf("x3 = %d\n", x3);
13.     return 0;
14. }
```

3.3 变参宏：...和__VA_ARGS__

- 一些函数（如printf()）接受数量可变的参数。stdvar.h头文件（本章后面介绍）提供了工具，让用户自定义带可变参数的函数
- 通过把宏参数列表中最后的参数写成省略号（即，3个点...）来实现这一功能
- 程序清单16.5 variadic.c

```c
#include <stdarg.h>
double sum(int, ...);
int main(void){
    double s,t;
    s = sum(3, 1.1, 2.5, 13.3);
    t = sum(6, 1.1, 2.1, 13.1, 4.1, 5.1, 6.1);
    return 0;
}

double sum(int lim,...){
    va_list ap; // declare object to hold arguments
    double tot = 0;
    va_start(ap, lim); // initialize to argument list
    for (int i = 0; i < lim; i++)
        tot += va_arg(ap, double);
    va_end(ap);  // clean up
    return tot;
}
```

翻译程序的第一步

4 宏和函数的选择

- 使用宏比使用普通函数复杂一些，稍有不慎会产生奇怪的副作用、
  - 宏定义成一行
- 宏和函数的选择实际上是时间和空间的权衡
  - 宏生成内联代码，即在程序中生成语句。而程序中只有一份函数语句的副本，节省了空间
  - 程序的控制必须跳转至函数内，随后再返回主调程序。比内联代码花费更多的时间
- 宏的一个优点是，不用担心变量类型
- C99提供了第3种可替换的方法—内联函数
- 注意
  - 记住宏名中不允许有空格，但是在替换字符串中可以有空格。ANSI C允许在参数列表中使用空格
  - 用圆括号把宏的参数和整个替换体括起来。这样能确保被括起来的部分在下面这样的表达式中正确地展开：
    - `forks = 2 * MAX(guests + 3, last);`
  - 用大写字母表示宏函数的名称。大写字母可以提醒程序员注意，宏可能产生的副作用
  - 在嵌套循环中使用宏更有助于提高效率。许多系统提供程序分析器以帮助程序员压缩程序中最耗时的部分。

文件包含：#include

5 文件包含：#include

- 当预处理器发现#include指令时，会查看后面的文件名并把文件的内容包含到当前文件中，即替换源文件中的#include指令
  - 相当于把被包含文件的全部内容输入到源文件#include指令所在的位置
- 在UNIX系统中，尖括号告诉预处理器在标准系统目录中查找该文件。双引号告诉预处理器首先在当前目录中（或文件名中指定的其他目录）查找该文件，如果未找到再查找标准系统目录
  - ANSI C不为文件提供统一的目录模型，因为不同的计算机所用的系统不同。一般而言，命名文件的方法因系统而异，但是尖括号和双引号的规则与系统无关
- 为什么要包含文件？因为编译器需要这些文件中的信息
- C语言习惯用.h后缀表示头文件，这些文件包含需要放在程序顶部的信息
  - 头文件经常包含一些预处理器指令
- 包含一个大型头文件不一定显著增加程序的大小
  - 在大部分情况下，头文件的内容是编译器生成最终代码时所需的信息，而不是添加到最终代码中的材料

5.1 头文件示例

- 程序清单16.6 names_st.h
  - 包含了一些头文件中常见的内容：#define指令、结构声明、typedef和函数原型。注意，这些内容是编译器在创建可执行代码时所需的信息，而不是可执行代码
  - 通常，应该用#ifndef和#define防止多重包含头文件
- 程序清单16.7 name_st.c
  - 可执行代码通常在源代码文件而不是在头文件中
- 程序清单16.8 useheader.c
  - 使用了程序清单16.6的头文件和程序清单16.7的源文件。

```
1.  #define SLEN 32
2.  // structure declarations
3.  struct names_st{char first[SLEN];};
4.  // typedefs
5.  typedef struct names_st names;
6.  // function prototypes
7.  void get_names(names *);

8.  // names_st.c -- define names_st functions
9.  #include "names_st.h"     // include the header file
10. // function definitions
11. void get_names(names * pn){  }

12. // useheader.c -- use the names_st structure
13. #include "names_st.h"
14. // remember to link with names_st.c
15. int main(void){ return 0; }
```

5.2 使用头文件

- 明示常量
  - 例如，stdio.h中定义的EOF、NULL和BUFSIZE（标准I/O缓冲区大小）
- 宏函数
  - 例如，getchar()通常用getc(stdin)定义，而getc()经常用于定义较复杂的宏，头文件ctype.h通常包含ctype系列函数的宏定义。
- 函数声明
  - 例如，string.h头文件（一些旧的系统中是strings.h）包含字符串函数系列的函数声明。在ANSI C和后面的标准中，函数声明都是函数原型形式
- 结构模版定义
  - 标准I/O函数使用FILE结构，该结构中包含了文件和与文件缓冲区相关的信息。FILE结构在头文件stdio.h中
- 类型定义
  - 标准I/O函数使用指向FILE的指针作为参数。通常，stdio.h用#define或typedef把FILE定义为指向结构的指针。类似地，size_t和time_t类型也定义在头文件中
- 另外，还可以使用头文件声明外部变量供其他文件共享
  - `int status = 0; // 该变量具有文件作用域，在源代码文件`
  - `extern int status; // 在头文件中`
- const防止值被意外修改
- static意味着每个包含该头文件的文件都获得一份副本。因此，不需要在一个文件中进行定义式声明，在其他文件中进行引用式声明

其它指令

6 其它指令

- #undef指令
- 从C预处理器角度看已定义
- 条件编译
- 预定义宏
- #line和#error
- #pragma

6.1 #undef指令

- #undef指令用于“取消”已定义的#define指令
- 如果想使用一个名称，又不确定之前是否已经用过，为安全起见，可以用#undef指令取消该名字的定义

6.2 从C预处理器角度看已定义

- 预处理器在识别标识符时，遵循与C相同的规则：标识符可以由大写字母、小写字母、数字和下划线字符组成，且首字符不能是数字
- 当预处理器在预处理器指令中发现一个标识符时
  - 它会把该标识符当作已定义的，已定义表示由预处理器定义
  - 或未定义的

```
1. #define LIMIT 1000 // LIMIT是已定义的
2. #define GOOD // GOOD 是已定义的
3. #define A(X) ((-(X))*(X)) // A 是已定义的
4. int q; // q 不是宏，因此是未定义的
5. #undef GOOD // GOOD 取消定义，是未定义的
```

6.3 条件编译

- #ifdef #else很像C的if else
  - 两者的主要区别是，预处理器不识别用于标记块的花括号（{}）
  - 使用#else（如果需要）和#endif（必须存在）来标记指令块。这些指令结构可以嵌套。也可以用这些指令标记C语句块
- 程序清单16.9 ifdef.c

```c
#include <stdio.h>
#define JUST_CHECKING
#define LIMIT 4
int main(void){
    int total = 0;
    for (int i = 1; i <= LIMIT; i++) {
        total += 2*i*i + 1;
#ifdef JUST_CHECKING
        printf("i=%d, total = %d\n", i, total);
#endif
    }
    printf("Grand total = %d\n", total);
    return 0;
}
```

#ifndef指令

- #ifndef指令通常用于防止多次包含一个文件
- 为何要多次包含一个文件？最常见的原因是，许多被包含的文件中都包含着其他文件，所以显式包含的文件中可能包含着已经包含的其他文件
  - 这有什么问题？在被包含的文件中有某些项（如，一些结构类型的声明）只能在一个文件中出现一次。C标准头文件使用#ifndef技巧避免重复包含
- 程序清单16.10 names.h

```
1.  // names.h --revised with include protection
2.  #ifndef NAMES_H_
3.  #define NAMES_H_

4.  // constants
5.  #define SLEN 32
6.  // structure declarations
7.  struct names_st{ };
8.  // typedefs
9.  typedef struct names_st names;
10. // function prototypes
11. void get_names(names *);
12. void show_names(const names *);
13. char * s_gets(char * st, int n);

14. #endif
```

#if和#elif指令

- #if后面跟整型常量表达式，如果表达式为非零，则表达式为真。可以在指令中使用C的关系运算符和逻辑运算符
- 可以按照if else的形式使用#elif（早期的实现不支持#elif）
- defined是一个预处理运算符，如果它的参数是用#defined定义过，则返回1；否则返回0。这种新方法的优点是，它可以和#elif一起使用。

```
#if defined (IBMPC)
    #include "ibmpc.h"
#elif defined (VAX)
    #include "vax.h"
#elif defined (MAC)
    #include "mac.h"
#else
    #include "general.h"
#endif
```

6.4 预定义宏

6.6 #pragma

- #pragma把编译器指令放入源代码中

C库

9 C库

- 最初，并没有官方的C库
- 后来，基于UNIX的C实现成为了标准。
- ANSI C委员会主要以这个标准为基础，开发了一个官方的标准库
- 在意识到C语言的应用范围不断扩大后， ANSI C委员会重新定义了这个库

9.1 访问C库

- 1. 自动访问
- 2. 文件包含
  - 通过#include指令包含定义宏函数的文件
- 3. 库包含
  - 在编译或链接程序的某些阶段，可能需要指定库选项

9.2 使用库描述

- 首先，了解函数文档
  - 阅读文档的关键是看懂函数头
- 可以在多个地方找到函数文档。你所使用的系统可能有在线手册，集成开发环境通常都有在线帮助

数学库

10 数学库

- 数学库中包含许多有用的数学函数。math.h头文件提供这些函数的原型
- 三角问题
  - 程序清单16.14　rect_pol.c
- 类型变体

通用工具库

11 通用工具库

- 通用工具库包含各种函数，包括随机数生成器、查找和排序函数、转换函数和内存管理函数
- 11.1 exit()和atexit()函数
  - 程序清单16.16 byebye.c
- 11.2 qsort()函数
  - 程序清单16.17 qsorter.c

断言库

12 断言库

- assert.h头文件支持的断言库是一个用于辅助调试程序的小型库
  - 它由assert()宏组成，接受一个整型表达式作为参数。如果表达式求值为假（非零），assert()宏就在标准错误流（stderr）中写入一条错误信息，并调用abort()函数终止程序
  - （abort()函数的原型在stdlib.h头文件中）
- 程序清单16.18　assert.c

string.h库中的memcpy()和memmove()

13 string.h库中的memcpy()和memmove()

- 不能把一个数组赋给另一个数组，所以要通过循环把数组中的每个元素赋给另一个数组相应的元素。
  - 使用strcpy()和strncpy()函数来处理字符数组
  - memcpy()和memmove()函数提供类似的方法处理任意类型的数组
- 程序清单16.20　mems.c

可变参数：stdarg.h

14 可变参数：stdarg.h

- 程序清单16.21　varargs.c
- stdarg.h头文件为函数提供了可以接受可变数量的参数：
  - 1. 提供一个使用省略号的函数原型；
  - 2. 在函数定义中创建一个va_list类型的变量；
  - 3. 用宏把该变量初始化为一个参数列表；
  - 4. 用宏访问参数列表；
  - 5. 用宏完成清理工作