宏

typedef

可以使用typedef定义函数指针

```
#include <iostream>

using namespace std;
//定义一个函数指针pFUN，它指向一个返回类型为char，有一个整型的参数的函数
char (*pFun)(int);
//定义一个返回类型为char，参数为int的函数
//从指针层面上理解该函数，即函数的函数名实际上是一个指针，
//该指针指向函数在内存中的首地址
//指向函数的指针变量没有 ++ 和 -- 运算
char glFun(int a)
{
    cout << a;
    //return a;
}

int main()
{
//将函数glFun的地址赋值给变量pFun
    pFun = glFun;
//*pFun”显然是取pFun所指向地址的内容，当然也就是取出了函数glFun()的内容，然后给定参数为2。
    (*pFun)(2);
    return 0;
}
/*使用typedef函数指针并使用*/

typedef char (*PTRFUN)(int);        //定义了一种PTRFUN的类型，并定义这种类型为指向某种函数的指针
PTRFUN pFun;                    //用这个新类型定义了变量pFun
char glFun(int a){ return;}         //这种函数以一个int为参数并返回char类型
void main()
{
    pFun = glFun;
    (*pFun)(2);
}
```

#if、##ifdef、#ifndef

```
#include <stdio.h>
int main(){
    #if _WIN32
        system("color 0c");
        printf("http://c.biancheng.net\n");
    #elif __linux__
        printf("\033[22;31mhttp://c.biancheng.net\n\033[22;30m");
    #else
        printf("http://c.biancheng.net\n");
    #endif
    return 0;
}
```

#if、#elif、#else 和 #endif 都是预处理命令，整段代码的意思是：如果宏 _WIN32的值为真，就保留第 4、5 行代码，删除第 7、9 行代码；如果宏 __linux的值为真，就保留第 7 行代码；如果所有的宏都为假，就保留第 9 行代码。

#ifdef 用法的一般格式为：

```
#ifdef  宏名
    程序段1
#else
    程序段2
#endif
```

它的意思是，如果当前的宏已被定义过，则对“程序段1”进行编译，否则对“程序段2”进行编译。与 #ifdef 相比, #ifndef的意思是，如果当前的宏未被定义，则对“程序段1”进行编译，否则对“程序段2”进行编译，这与 #ifdef 的功能正好相反。

最后需要注意的是，#if 后面跟的是“整型常量表达式”，而 #ifdef 和 #ifndef 后面跟的只能是一个宏名，不能是其他的。

数据类型

void* data

无类型指针，例如对于`void *data[ ]`是包含了若干个void *类型的数组。特性有：

> - 是一种无类型指针
> - 任何指针都可以赋值给void指针
> - 转换为其他指针时，需要进行类型转换

`void *data[ ]`是一个指针的指针相当于`void **`,`void *data`是一个无类型的指针参数，任意类型指针可以赋值给data，当data赋值给其他类型时需要先指定数据类型

enum

枚举，用于定义一组具有离散值的常量。它可以让数据更简洁，更易读。

```
enum  枚举名  {枚举元素1,枚举元素2,……};

#define MON  1
#define TUE  2
#define WED  3
#define THU  4
#define FRI  5
```

```
#define SAT  6
#define SUN  7
/*等同于*/
enum DAY
{
      MON=1, TUE, WED, THU, FRI, SAT, SUN
};
```

关键字

volatile

译为直接存取原始内存地址，提醒编译器它后面所定义的变量随时都有可能改变，因此编译后的程序每次需要存储或读取这个变量的时候，告诉编译器对该变量不做优化，都会直接从变量内存地址中读取数据，从而可以提供对特殊地址的稳定访问。

假设要对一个设备进行初始化，此设备的某一个寄存器为0xff800000。for(i=0;i< 10;i++) *output = i;前面循环半天都是废话，对最后的结果毫无影响，因为最终只是将output这个指针赋值为9，省略了对该硬件IO端口反复读的操作

函数

snprintf()

C 库函数 **int snprintf(char *str, size_t size, const char *format, ...)** 设将可变参数(...)按照 **format** 格式化成字符串，并将字符串复制到 **str** 中，**size** 为要写入的字符的最大数目，超过 **size** 会被截断，最多写入 size-1 个字符。

与 sprintf() 函数不同的是，snprintf() 函数提供了一个参数 size，可以防止缓冲区溢出。如果格式化后的字符串长度超过了 size-1，则 snprintf() 只会写入 size-1 个字符，并在字符串的末尾添加一个空字符（\0）以表示字符串的结束。