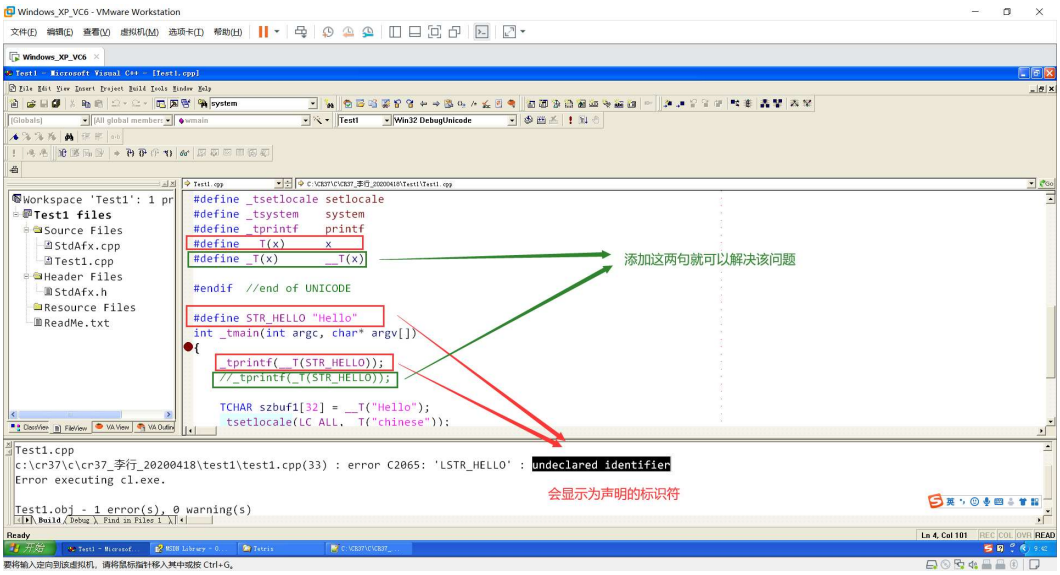
2020/04/17_第13课_预处理条件判断的使用、项目的工程组织

笔记本： C
创建时间： 2020/4/17 星期五 15:46
作者： ileemi
标签： 条件判断的使用, 预处理

课前回顾



在使用Unicode编码输出中文的时候，需要使用setlocale()函数（头文件<locale.h>）对其进行地域设置

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

#ifdef UNICODE

#define TCHAR wchar_t
#define _tmain  wmain
#define _tsetlocale _wsetlocale
#define _tsystem    _wsystem
```

```
#define _tprintf    wprintf
#define __T(x)      L##x
#define _T(x)       __T(x)

#else
#define TCHAR char
#define _tmain  main
#define _tsetlocale setlocale
#define _tsystem    system
#define _tprintf    printf

#define __T(x)      x
#define _T(x)       __T(x)

#endif  //end of UNICODE

#define STR_HELLO "Hello"
int _tmain(int argc, char* argv[])
{
    //_tprintf(__T(STR_HELLO));
    _tprintf(_T(STR_HELLO));

    TCHAR szbuf1[32] = __T("Hello");
    /*
    使用unicode编码输出中文时需要添加setlocale(), 设置时区
    */
    _tsetlocale(LC_ALL,__T("chinese"));
    TCHAR szbuf2[64] = __T("Hello你好World世界\r\n");
    _tprintf(__T("Hello World!\n"));
    _tsystem(__T("pause"));
    return 0;
}
```

工程组织

加 /P不产生.obj文件
只能编译，不能链接

.cpp包含.cpp是不可取的
会重复定义，多个obj包含多个函数名

宏不能出文件，将其定义到头文件内

原则（规范）

1、产生任何处理器行为（产生二进制的代码）的代码，都在.c .cpp这类源码文件内定义
2、不产生处理器行为的代码以及各种外部声明，函数声明和宏定义等（结构体、共同体等），函数声明默认extern，应在 h 类头文件中定义，hpp 是C++头文件

main函数开始处产生处理器行为
函数声明不产生处理器行为，函数定义实现产生处理器行为

- 文加内过次对同名宏做出定义的，编译报错，#undef后，最后一次定义生效
- 宏定义不出文件，跨文件访问宏，则该宏不存在
- 包含后，形成多次定义同一个宏的，编译报错，#undef后，最后一次定义生效
- 重复定义或者重复包含的后果，要么编译不通过，报告重复定义错误，要么直接按最后一次定义生效

struct只能唯一，不能重复定义

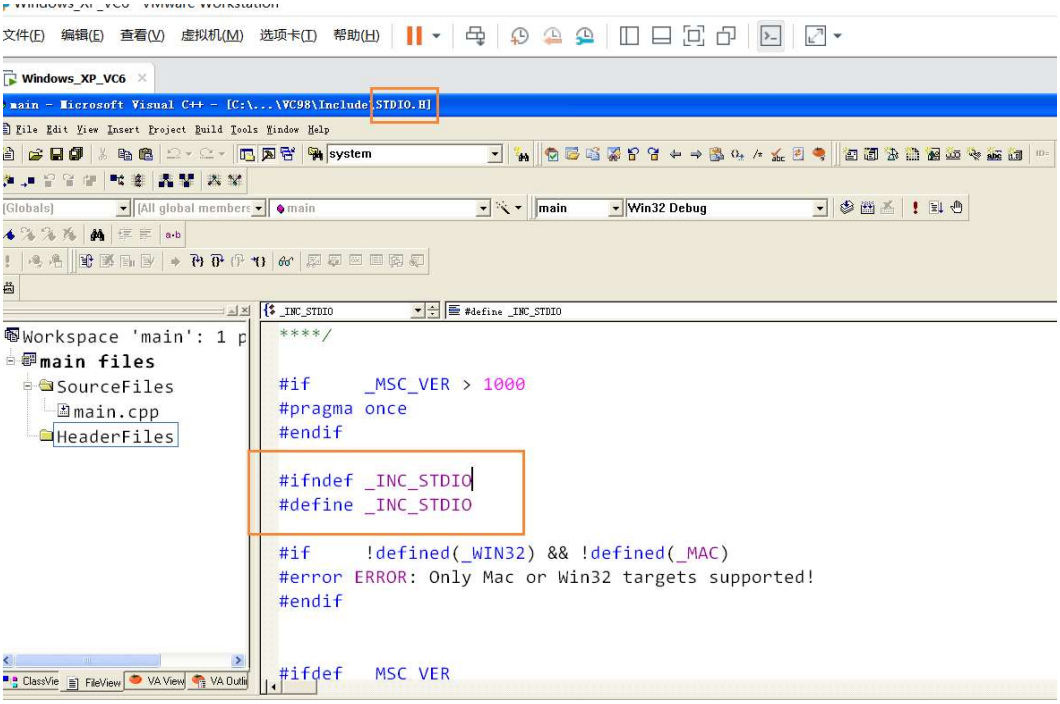
3、尽量设计宏名不重复，，并且避免
如果宏名重复很容易出现错误
重复包含头文件，就会造成一些重复定义问题

防止多次使用头文件时文件重复包含，相关函数出现重定义等可使用下面的方式对其做出相应的处理：
每个头文件都应该添加条件判断

标准做法：

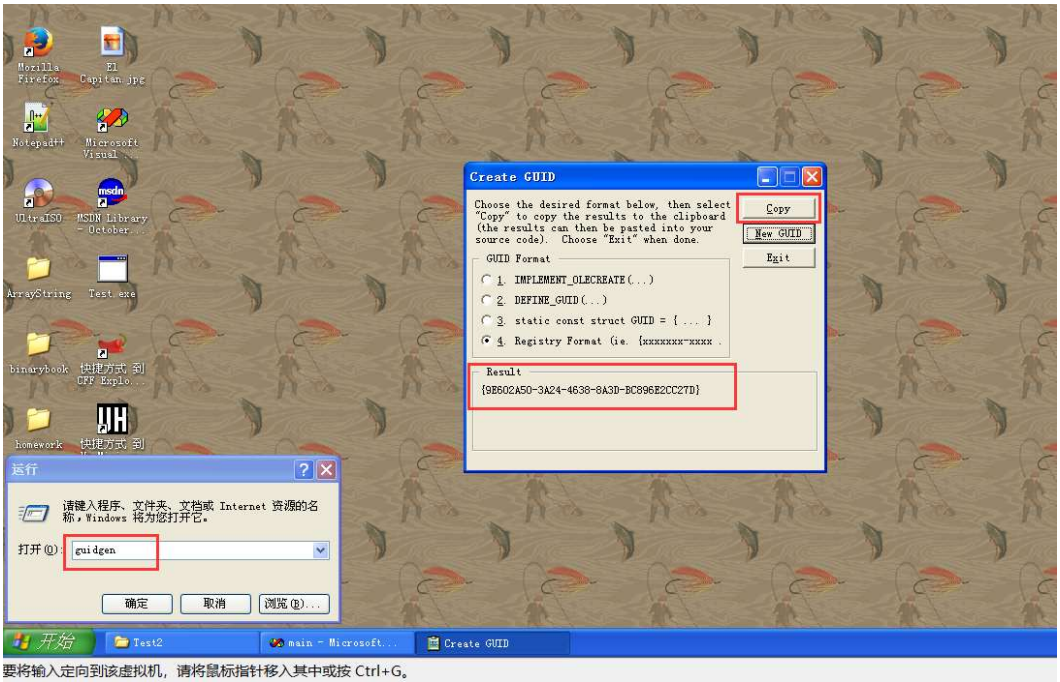
```
#ifndef xxx          //判断xxx是否定义
#define xxx          //没有定义，就定义
...
#endif               //添加注释：例如：end of xxx
 /*
 头文件内的内容过长需要在对应的条件判断后添加#endif，必要时添加注释说明
 可查看官方的TCHAR相关的宏定义源码是怎么做的
 */
```

查看官方的STDIO.H头文件，就可以看出为了防止重复包含做出相应的处理：



如何防止宏名重复

定义宏名字的时候最好不要添加自己的名字信息
可使用GUID(Globally Unique Identifier)：全球唯一标识符
在Windows下使用快捷键"win + R"在运行窗口输入 "guidgen"，如下图所示：




```
//一般定义
#ifndef MY_HEADER
#define MY_HEADER
...
#endif //end of MY_HEADER

//为了防止上面定义的宏名重复，可将上面的宏定义更改成如下所示
```

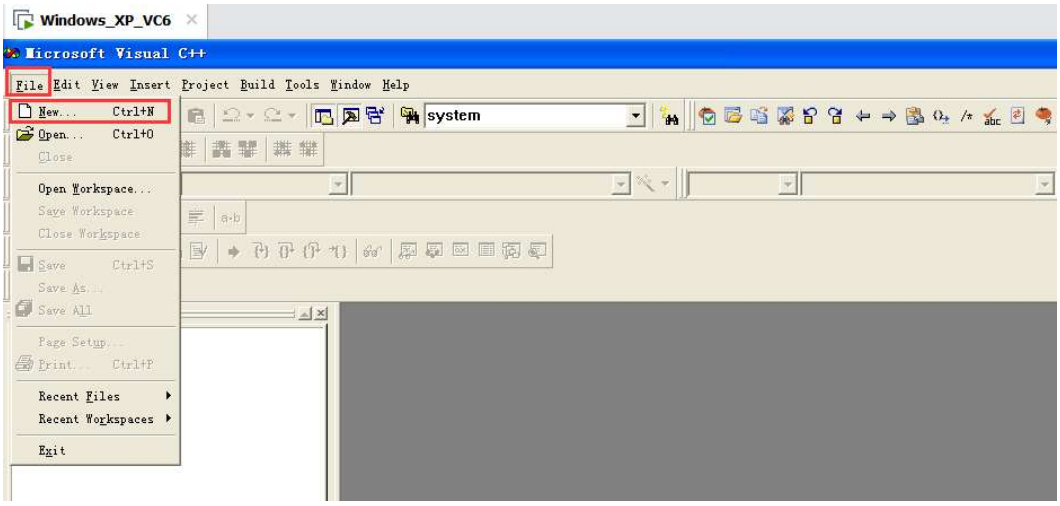
```
#ifndef MY_HEADER_9E602A50_3A24_4638_8A3D_BC896E2CC27D
#define MY_HEADER_9E602A50_3A24_4638_8A3D_BC896E2CC27D
...
#endif //end of MY_HEADER_9E602A50_3A24_4638_8A3D_BC896E2CC27D

//同时还可以使用下面这样的写法
#if !define(MT_HEADER_0DA06B11_77AD_4ce8_9ADF_D9897AF87AD2)
#define MT_HEADER_0DA06B11_77AD_4ce8_9ADF_D9897AF87AD2
...
#endif //end of MT_HEADER_0DA06B11_77AD_4ce8_9ADF_D9897AF87AD2
```

这样就也可以防止包含头文件时出现重复包含情况导致函数等出现重复定义使其编译不通过的问题。

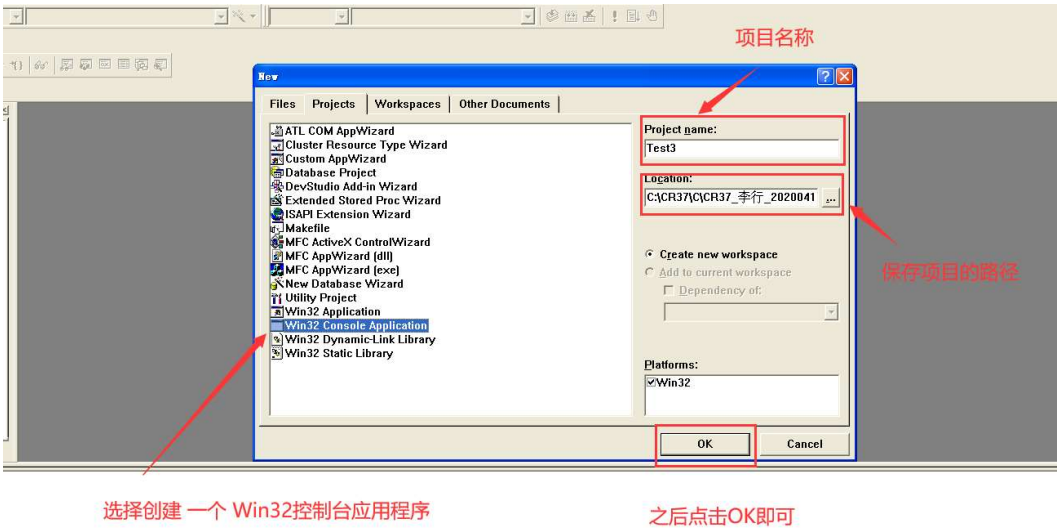
也可以使用 `#pragma once` 包含在头文件的首部即可
#pragma once 该指令是写给编译器看的，保证或者文件只包含一次非，非标准

VS++ 6.0 工程的创建

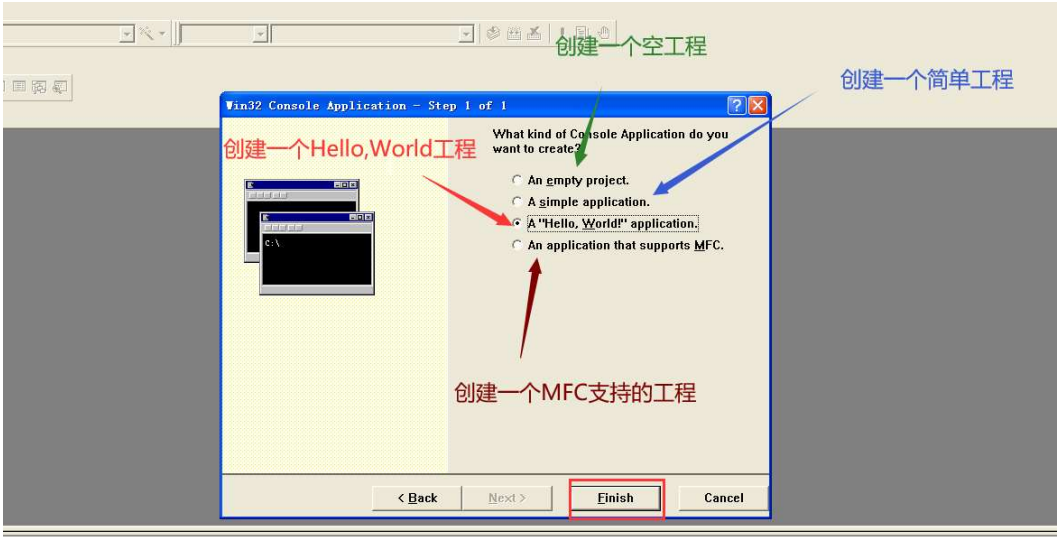
1、打开VS++ 6.0



2、在弹出的New面板下的ojects菜单中选择创建一个Win32 控制台应用程序，如下图所示：

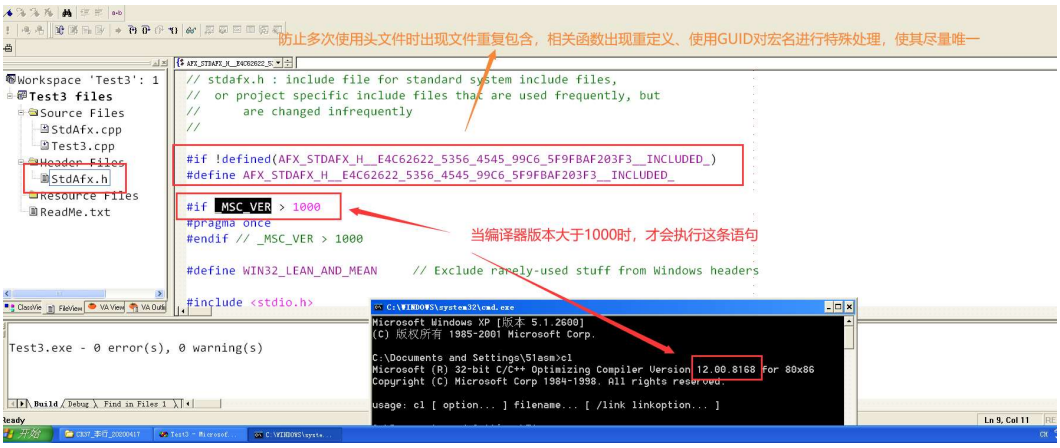


3、接着选择创建一个什么样的工程



这里我们选择创建一个Hello,World工程,之后点击finish完成即可

4、查看生成的工程文件中的头文件



寄存器变量

计算机从寄存器中读取数据比从内存中读取数据要快，所以C语言允许讲一些频繁使用的局部变量定义为寄存器变量，访问寄存器中的变量的效率由主频决定，访问内存中的变量的效率由外频决定。
主板的频率称为外频
CPU的频率称为内频

主频 / 外频 = 倍频（越低越好）

定义寄存器变量的禁忌：

- 32位环境，寄存器一起性只能处理4字节的数据
- 变量不能超4字节
- 不能取地址
- 不能全局变量和静态局部变量
- Debug版本，register失效
- Release版 /O2，尝试把所有可以register化的变量都做成register，即使你没有register修饰（
  鸡肋）

只有自动变量，或者形式参数可以是寄存器变量，全局变量和静态局部变量不行。
函数中的寄存器变量在调用函数时占用寄存器存放变量的值，当函数结束时释放寄存器，该变量消失。
寄存器变量仅限于int型、char型、指针类型的变量使用，如果寄存器使用饱和时（Debug永远饱和），程序会将寄存器变量自动转换为自动变量处理。
寄存器变量不能取地址，只要取地址就不符合寄存器的特性

宏是一把双刃剑，它可以大大提高代码的可读性和可移植性，也可以大大减低代码的可读性和可移植性

```
#include <stdio.h>
#include <stdlib.h>
#define while if
#define int float
int main(int argc, char* argv[])
{
    int n = 1;
    while(n < 6)
    {
        printf("%d\r\n", n);
        n++;
    }
    printf("Hello World!\n");
    system("pause");
        return 0;
}
```