在C语言中，我们使用char来定义字符，占用一个字节，最多只能表示128个字符，也就是ASCII码中的字符。计算机起源于美国，char 可以表示所有的英文字符，在以英语为母语的国家完全没有问题。

但是世界上存在很多不同的语言，例如汉语、汉语、日语等有成千上万个字符，需要用多个字节来表示，称之为宽字符(Wide Character)。Unicode 是宽字符编码的一种，已经被现代计算机指定为默认的编码方式，Windows 2000以后的操作系统，包括Windows 2000、XP、Vista、Win7、Win8、Win10、Windows Phone、Windows Server 等（它们统称为 Windows NT）都从底层支持Unicode，存取效率比 char 要高。

更多内容请查看：ASCII编码与Unicode编码

C语言中的宽字符

在C语言中，使用wchar.h头文件中的wchar_t来定义宽字符，例如：

```
wchar_t ch = 'A';
```

wchar_t 被定义为typedef unsigned short wchar_t，和一个无符号整型一样，占用两个字节。

如果定义宽字符串，需要加前缀L，例如：

```
wchar_t *str = L"C语言中文网";
```

L是必须要加的，并且与字符串之间不能有空格，只有这样编译器才知道每个字符占用两个字节。

宽字符示例：

```
1. #include <stdio.h>
2. #include <wchar.h>
3. int main(){
4.     char ch = 'A';
5.     wchar_t wch = 'A';
6.     char str[] = "C语言中文网";
7.     wchar_t wstr[] = L"C语言中文网";
8.     printf("ch=%d, wch=%d, str=%d, wstr=%d\n", sizeof(ch), sizeof(wch),
sizeof(str), sizeof(wstr));
9.     return 0;
10. }
```

运行结果：

```
ch=1, wch=2, str=12, wstr=14
```

wstr 之所以比 str 多两个字节是因为：字符 'C' 占用两个字节，字符串结束标志 '\0' 也占用两个字节。

宽字符串的长度

计算ASCII字符串长度使用 strlen 函数，计算宽字符串长度使用 wcslen 函数：

```
1. #include <stdio.h>
2. #include <wchar.h>
3. #include <string.h>
4. int main(){
5.     char str[] = "C语言中文网";
6.     wchar_t wstr[] = L"C语言中文网";
7.     printf("strlen(str)=%d, wcslen(wstr)=%d\n", strlen(str), wcslen(wstr));
8.     return 0;
9. }
```

运行结果：

```
strlen(str)=11, wcslen(wstr)=6
```

strlen 的运行结果显然不正确，因为它把一个字节作为一个字符计算，而 wcslen 把两个字节作为一个字符计算。

注意：wcslen 在 string.h 和 wchar.h 头文件中均有说明。

维护一个版本的源代码

在 Windows NT 以前的操作系统中，甚至包括 Windows 98，对宽字符的支持都不是很好，所以大多情况下使用ASCII编码。Windows NT 推出以后，已经从底层支持了Unicode，所以在 Windows NT 上的程序大多使用Unicode。

如果你希望程序能够在各种版本的Windows操作系统中运行，那么就需要维护两个版本的源代码，ASCII 版和 Unicode 版。ASCII 字符和 Unicode 字符的定义、使用都不一样，要想在一个版本的源代码中做兼容处理会非常困难，要做大量的工作，对程序员来说简直是噩梦。

不过，Windows 又为我们做了一件好事，已经处理了兼容性问题。它是怎么做到的呢？

例如对于字符串，ASCII 中使用 char 来定义，而 Unicode 中使用 wchar_t 来定义，并且需要添加前缀L。那么在 windows.h 头文件中（或者是它包含的其他头文件）就这样来处理：

```
#ifdef UNICODE
typedef wchar_t TCHAR;
#define TEXT(quote) L##quote
#else
typedef char TCHAR
#define TEXT(quote) quote
#endif
```

我们在源码中可以这样来使用：

```
TCHAR str[] = TEXT("C语言中文网");
```

如果是Unicode版，也就是定义了UNICODE宏，那么上面的语句等价于：

```
wchar_t str[] = L"C语言中文网";
```

如果是ASCII，也就是没有定义UNICODE宏，那么等价于：

```
char str[] = "C语言中文网";
```

在Windows中，随处可见这样的处理。虽然现代操作系统都已经支持Unicode，无需再考虑与ASCII的兼容性问题，但是依然要为这些历史问题付出代价。