# 学习大纲

|  |
| --- |
|  |

## 1.什么是wchar\_t?

### 就是宽字符，他的api和char的api版本不一样

函数介绍：  
**wcslen**()   类似与char\*类型作参数的strlen()函数      用来获取wchar\_t\*变量的长度（不包含终结符）  
**wcscpy**()  类似与char\*类型作参数的strcpy()函数      用来进行wchar\_t\*变量之间的复制  
同样的还有  wcsncpy()  
**wcscmp**()  类似与char\*类型的strcmp()函数      用来对比两个wchar\_t\*变量的大小  
**wmemset**()  类似与char\*类型的memset()函数     用来初始化内存  
**wprintf**()  类似与char\*类型的printf()函数        用来输出wchar\_t\*字符串  
  
  
**简单记忆**：  
一般把char类型函数中的" str"替换为" wcs "即可  
wcs：wide char string

## 2宽字符处理函数函数与普通函数对照表

### 1)字符分类： 宽字符函数普通C函数描述

#### iswalnum（） isalnum（） 测试字符是否为数字或字母

#### iswalpha（） isalpha（） 测试字符是否是字母

#### iswcntrl（） iscntrl（） 测试字符是否是控制符

#### iswdigit（） isdigit（） 测试字符是否为数字

#### iswgraph（） isgraph（） 测试字符是否是可见字符

#### iswlower（） islower（） 测试字符是否是小写字符

#### iswprint（） isprint（） 测试字符是否是可打印字符

#### iswpunct（） ispunct（） 测试字符是否是标点符号

#### iswspace（） isspace（） 测试字符是否是空白符号

#### iswupper（） isupper（） 测试字符是否是大写字符

#### iswxdigit（） isxdigit（）测试字符是否是十六进制的数字

### 2)大小写转换：

#### 宽字符函数普通C函数描述

#### towlower（） tolower（） 把字符转换为小写

#### towupper（） toupper（） 把字符转换为大写

### 2.2)字符比较： 宽字符函数普通C函数描述

#### wcscoll（） strcoll（） 比较字符串

### 3)日期和时间转换： 宽字符函数描述

#### strftime（） 根据指定的字符串格式和locale设置格式化日期和时间

#### wcsftime（） 根据指定的字符串格式和locale设置格式化日期和时间， 并返回宽字符串

#### strptime（） 根据指定格式把字符串转换为时间值， 是strftime的反过程

### 4)打印和扫描字符串： 宽字符函数描述

#### fprintf（）/fwprintf（） 使用vararg参量的格式化输出

#### fscanf（）/fwscanf（） 格式化读入

#### printf（）/wprintf() 使用vararg参量的格式化输出到标准输出

#### scanf（）/ wscanf从标准输入的格式化读入

#### sprintf（）/swprintf（） 根据vararg参量表格式化成字符串

#### sscanf（）/ swscanf() 以字符串作格式化读入

#### vfprintf（）/vfwprintf（） 使用stdarg参量表格式化输出到文件

#### vprintf（）/ vwprintf() 使用stdarg参量表格式化输出到标准输出

#### vsprintf（）/vswprintf（） 格式化stdarg参量表并写到字符串

### 5)数字转换：

#### 宽字符函数普通C函数描述

#### wcstod（） strtod（）   把宽字符的初始部分转换为双精度浮点数

#### wcstol（） strtol（）   把宽字符的初始部分转换为长整数

#### wcstoul（） strtoul（） 把宽字符的初始部分转换为无符号长整数

### 6)多字节字符和宽字符转换及操作：

#### 宽字符函数描述

#### mblen（） 根据locale的设置确定字符的字节数

#### mbstowcs（） 把多字节字符串转换为宽字符串

#### mbtowc（）/btowc（）把多字节字符转换为宽字符

#### wcstombs（） 把宽字符串转换为多字节字符串

#### wctomb（）/wctob（） 把宽字符转换为多字节字符

### 7)输入和输出：

#### 宽字符函数普通C函数描述

#### fgetwc（） fgetc（） 从流中读入一个字符并转换为宽字符

#### fgetws（） fgets（） 从流中读入一个字符串并转换为宽字符串

#### fputwc（） fputc（） 把宽字符转换为多字节字符并且输出到标准输出

#### fputws（） fputs（） 把宽字符串转换为多字节字符并且输出到标准输出串

#### getwc（） getc（） 从标准输入中读取字符， 并且转换为宽字符

#### getwchar（） getchar（） 从标准输入中读取字符， 并且转换为宽字符

#### None gets（） 使用fgetws（）

#### putwc（） putc（） 把宽字符转换成多字节字符并且写到标准输出

#### putwchar（） putchar（） 把宽字符转换成多字节字符并且写到标准输出

#### None puts（） 使用fputws（）

#### ungetwc（） ungetc（） 把一个宽字符放回到输入流中

### 8)字符串操作：

#### 宽字符函数普通C函数描述

#### wcscat（） strcat（） 把一个字符串接到另一个字符串的尾部

#### wcsncat（） strncat（） 类似于wcscat（）， 而且指定粘接字符串的粘接长度.

#### wcschr（） strchr（） 查找子字符串的第一个位置

#### wcsrchr（） strrchr（） 从尾部开始查找子字符串出现的第一个位置

#### wcspbrk（） strpbrk（） 从一字符字符串中查找另一字符串中任何一个字符第一次出现的位置

#### wcswcs（）/wcsstr（） strchr（） 在一字符串中查找另一字符串第一次出现的位置

#### wcscspn（） strcspn（） 返回不包含第二个字符串的的初始数目

#### wcsspn（） strspn（） 返回包含第二个字符串的初始数目

#### wcscpy（） strcpy（） 拷贝字符串

#### wcsncpy（） strncpy（） 类似于wcscpy（）， 同时指定拷贝的数目

#### wcscmp（） strcmp（） 比较两个宽字符串

#### wcsncmp（） strncmp（） 类似于wcscmp（）， 还要指定比较字符字符串的数目

#### wcslen（） strlen（） 获得宽字符串的数目

#### wcstok（） strtok（） 根据标示符把宽字符串分解成一系列字符串

#### wcswidth（） None 获得宽字符串的宽度

#### wcwidth（） None 获得宽字符的宽度

### 9)另外还有对应于memory操作

#### wmemcpy（）， wmemchr（）， wmemcmp（）， wmemmove（）， wmemset（）．

## 3.项目演练

## 1.新建一个控制台项目，取名：Lesson9-wchar\_t，然后新建一个源文件取名wchar\_t.cpp,代码如下

|  |
| --- |
| #include<stdio.h>  #include<stdlib.h>  #include <Locale.h>  int main()  {  /\*\*  在c语言中，char类型的变量只能用来保存英文字母，保存中文字是不行的  如果想用wprintf输出中文，需要包含Locale.h头文件，还需要调用这个函数setlocale(LC\_ALL, "chs");  \*/  // setlocale(LC\_ALL, "chs");  setlocale(LC\_ALL, ""); //这样写也行  wchar\_t\* zh = L"我是中国人";  wprintf(L"zh:%s\n",zh);  wchar\_t c = L'爱';  wprintf(L"%c\n",c);  //字符数组  wchar\_t wsz[] = L"伟大的中华人民共和国";  wprintf(L"wsz:%s\n",wsz);  system("pause");  return 0;  } |

### 别看只有短短几行代码，非常坑爹，想用wprintf输出中文，需要包含Locale.h头文件，还需要调用这个函数setlocale(LC\_ALL, "chs");也就是是c语言默认不能显示中文，显示中文必须设置为地点字符集

### 效果：

|  |
| --- |
|  |

## 2.然后我们来学习一下wcslen函数，需要包含string.h头文件

|  |
| --- |
| #include<stdio.h>  #include<stdlib.h>  #include <Locale.h>  #include<string.h>  int main()  {  /\*\*  在c语言中，char类型的变量只能用来保存英文字母，保存中文字是不行的  如果想用wprintf输出中文，需要包含Locale.h头文件，还需要调用这个函数setlocale(LC\_ALL, "chs");  \*/  // setlocale(LC\_ALL, "chs");  setlocale(LC\_ALL, ""); //这样写也行  wchar\_t\* zh = L"我是中国人";  wprintf(L"zh:%s\n",zh);  wchar\_t c = L'爱';  wprintf(L"%c\n",c);  //字符数组  wchar\_t wsz[] = L"伟大的中华人民共和国";  wprintf(L"wsz:%s\n",wsz); //length of wsz:10  //获取宽字符串的长度  wprintf(L"length of wsz:%d\n",wcslen(wsz));  system("pause");  return 0;  } |

### 效果：

|  |
| --- |
|  |

## 3.我们来学习一下宽字符串比较函数wcscmp

|  |
| --- |
| #include<stdio.h>  #include<stdlib.h>  #include <Locale.h>  #include<string.h>  int main()  {  /\*\*  在c语言中，char类型的变量只能用来保存英文字母，保存中文字是不行的  如果想用wprintf输出中文，需要包含Locale.h头文件，还需要调用这个函数setlocale(LC\_ALL, "chs");  \*/  // setlocale(LC\_ALL, "chs");  setlocale(LC\_ALL, ""); //这样写也行  wchar\_t\* zh = L"我是中国人";  wprintf(L"zh:%s\n",zh);  wchar\_t c = L'爱';  wprintf(L"%c\n",c);  wprintf(L"length of zh:%d\n",wcslen(zh));  //字符数组  wchar\_t wsz[] = L"伟大的中华人民共和国";  wprintf(L"wsz:%s\n",wsz); //length of wsz:10  //获取宽字符串的长度  wprintf(L"length of wsz:%d\n",wcslen(wsz));  //宽字符串比较  int ret = wcscmp(zh,wsz);  if(0 == ret)  {  printf("zh is equal to wsz\n");  }  else if(ret<0)  {  printf("zh is shorter than wsz\n");  }  else  {  printf("zh is shorter than wsz\n");  }  system("pause");  return 0;  } |

### 效果：

|  |
| --- |
|  |

## 4.宽字符串拼接

|  |
| --- |
| #include<stdio.h>  #include<stdlib.h>  #include <Locale.h>  #include<string.h>  int main()  {  /\*\*  在c语言中，char类型的变量只能用来保存英文字母，保存中文字是不行的  如果想用wprintf输出中文，需要包含Locale.h头文件，还需要调用这个函数setlocale(LC\_ALL, "chs");  \*/  // 1.setlocale(LC\_ALL, "chs");  setlocale(LC\_ALL, ""); //这样写也行  wchar\_t\* zh = L"我是中国人";  wprintf(L"zh:%s\n",zh);  wchar\_t c = L'爱';  wprintf(L"%c\n",c);  wprintf(L"length of zh:%d\n",wcslen(zh));  //2.字符数组  wchar\_t wsz[] = L"伟大的中华人民共和国";  wprintf(L"wsz:%s\n",wsz); //length of wsz:10  //3.获取宽字符串的长度  wprintf(L"length of wsz:%d\n",wcslen(wsz));  //4.宽字符串比较  int ret = wcscmp(zh,wsz);  if(0 == ret)  {  printf("zh is equal to wsz\n");  }  else if(ret<0)  {  printf("zh is shorter than wsz\n");  }  else  {  printf("zh is shorter than wsz\n");  }  //5.宽字符串拼接  wchar\_t wcs1[40] = L"我爱你";  wchar\_t wcs2[40] = L"中国";  wcscat\_s(wcs1,wcs2);  wprintf(L"wcs1+wcs2=%s\n",wcs1);  system("pause");  return 0;  } |

### 效果：

|  |
| --- |
|  |

### 注意：不要把中文字符串赋值给普通的ansi字符串，这样子不好，你在获取字符串的长度时会得到错误的结果。应该吧它赋值给宽字符串

## 5.字符串复制wcscpy

|  |
| --- |
| #include<stdio.h>  #include<stdlib.h>  #include <Locale.h>  #include<string.h>  int main()  {  /\*\*  在c语言中，char类型的变量只能用来保存英文字母，保存中文字是不行的  如果想用wprintf输出中文，需要包含Locale.h头文件，还需要调用这个函数setlocale(LC\_ALL, "chs");  \*/  // 1.setlocale(LC\_ALL, "chs");  setlocale(LC\_ALL, ""); //这样写也行  wchar\_t\* zh = L"我是中国人";  wprintf(L"zh:%s\n",zh);  wchar\_t c = L'爱';  wprintf(L"%c\n",c);  wprintf(L"length of zh:%d\n",wcslen(zh));  //2.字符数组  wchar\_t wsz[] = L"伟大的中华人民共和国";  wprintf(L"wsz:%s\n",wsz); //length of wsz:10  //3.获取宽字符串的长度  wprintf(L"length of wsz:%d\n",wcslen(wsz));  //4.宽字符串比较  int ret = wcscmp(zh,wsz);  if(0 == ret)  {  printf("zh is equal to wsz\n");  }  else if(ret<0)  {  printf("zh is shorter than wsz\n");  }  else  {  printf("zh is shorter than wsz\n");  }  //5.宽字符串拼接  wchar\_t wcs1[40] = L"我爱你";  wchar\_t wcs2[40] = L"中国";  wcscat\_s(wcs1,wcs2);  wprintf(L"wcs1+wcs2=%s\n",wcs1);  //6.宽字符串复制  wchar\_t src[] =L"大家好，欢迎学习c语言";  wchar\_t dst[40] =L"";  wcscpy(dst,src);  wprintf(L"copy result=%s\n",dst);  system("pause");  return 0;  } |

### 效果：

|  |
| --- |
|  |

# 扩展,编写一个读写中文的c语言小程序

## 需要用到的api:fopen\_s, \_wfopen\_s

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 打开文件。 这些版本的 [fopen、\_wfopen](https://learn.microsoft.com/zh-cn/cpp/c-runtime-library/reference/fopen-wfopen?view=msvc-170) 具有安全性增强功能，如 [CRT 中的安全性功能](https://learn.microsoft.com/zh-cn/cpp/c-runtime-library/security-features-in-the-crt?view=msvc-170)中所述。 语法 C复制  errno\_t fopen\_s(  FILE\*\* pFile,  const char \*filename,  const char \*mode  );  errno\_t \_wfopen\_s(  FILE\*\* pFile,  const wchar\_t \*filename,  const wchar\_t \*mode  ); 参数 *pFile* 指向文件指针的指针，该指针接收指向打开的文件的指针。  *filename* 要打开的文件的名称。  *mode* 允许的访问类型。 返回值 如果成功，则为零；如果失败，则为错误代码。 有关这些错误代码的详细信息，请参阅 [errno、\_doserrno、\_sys\_errlist 和 \_sys\_nerr](https://learn.microsoft.com/zh-cn/cpp/c-runtime-library/errno-doserrno-sys-errlist-and-sys-nerr?view=msvc-170)。 错误条件 展开表   | ***pFile*** | ***filename*** | ***mode*** | **返回值** | ***pFile* 的内容** | | --- | --- | --- | --- | --- | | NULL | any | 任意 | EINVAL | 未更改 | | any | NULL | 任意 | EINVAL | 未更改 | | any | 任意 | NULL | EINVAL | 未更改 |  备注 **fopen\_s** 和 **\_wfopen\_s** 函数无法打开用于共享的文件。 如果需要共享文件，请将 [\_fsopen 或 \_wfsopen](https://learn.microsoft.com/zh-cn/cpp/c-runtime-library/reference/fsopen-wfsopen?view=msvc-170) 与适当的共享模式常量配合使用（例如，将 \_SH\_DENYNO 用于读/写共享）。  **fopen\_s** 函数打开由 *filename* 指定的文件。 **\_wfopen\_s** 是 **fopen\_s** 的宽字符版本，**\_wfopen\_s** 的参数是宽字符字符串。 除此以外，**\_wfopen\_s** 和 **fopen\_s** 的行为完全相同。  **fopen\_s** 接受执行时文件系统上有效的路径；**fopen\_s** 接受 UNC 路径以及包含所映射网络驱动器的路径，前提是执行代码的系统在执行时能够访问共享或映射的网络驱动器。 为 **fopen\_s** 构造路径时，请勿做出有关驱动器、路径或执行环境中网络共享的可用性假设。 可使用斜杠 (/) 或反斜杠 (\) 作为路径中的目录分隔符。  这些函数验证其参数。 如果 *pFile*、*filename* 或 *mode* 为 null 指针，则这些函数将生成无效的参数异常，如[参数验证](https://learn.microsoft.com/zh-cn/cpp/c-runtime-library/parameter-validation?view=msvc-170)中所述。  请始终先检查返回值，确定该函数是否成功，再对文件执行任何进一步的操作。 如果出现错误，则将返回错误代码并且将设置全局变量 errno。 有关详细信息，请参阅[errno, \_doserrno, \_sys\_errlist和\_sys\_nerr](https://learn.microsoft.com/zh-cn/cpp/c-runtime-library/errno-doserrno-sys-errlist-and-sys-nerr?view=msvc-170).  默认情况下，此函数的全局状态范围限定为应用程序。 若要更改此状态，请参阅 [CRT 中的全局状态](https://learn.microsoft.com/zh-cn/cpp/c-runtime-library/global-state?view=msvc-170)。 Unicode 支持 **fopen\_s** 支持 Unicode 文件流。 若要打开新的或现有的 Unicode 文件，请将指定所需编码的 **ccs** 标志传递到 **fopen\_s**，例如：  **fopen\_s(&fp, "newfile.txt", "w+, ccs=UNICODE");**  **ccs** 标志的允许值为 UNICODE、**UTF-8** 和 **UTF-16LE**。 如果未为 **ccs** 指定值，则 **fopen\_s** 将使用 ANSI 编码。  如果文件已存在并已打开以进行读取或追加，则字节顺序标记 (BOM)（如果文件中存在）将确定编码。 BOM 编码优先于 **ccs** 标志指定的编码。 仅在不存在 BOM 或如果文件是新文件时，才使用 **ccs** 编码。  **备注**  BOM 检测仅适用于在 Unicode 模式下打开的文件；即，通过传递 **ccs** 标志打开的文件。  下表汇总了提供给 **fopen\_s** 的各种 **ccs** 标志值的模式以及文件中的 BOM 的模式。 基于 ccs 标志和 BOM 使用的编码 展开表   | **ccs 标志** | **无 BOM（或新文件）** | **BOM：UTF-8** | **BOM：UTF-16** | | --- | --- | --- | --- | | UNICODE | **UTF-8** | **UTF-8** | **UTF-16LE** | | **UTF-8** | **UTF-8** | **UTF-8** | **UTF-16LE** | | **UTF-16LE** | **UTF-16LE** | **UTF-8** | **UTF-16LE** |   在 Unicode 模式下打开用于写入的文件将自动在其中写入 BOM。  如果 *mode* 为 **"a, ccs=UNICODE"**、**"a, ccs=UTF-8"** 或 **"a, ccs=UTF-16LE"**，**fopen\_s** 将先尝试使用读取和写入访问权限打开文件。 如果成功，此函数将读取 BOM 以确定文件的编码；如果失败，此函数将使用文件的默认编码。 在任一情况下，**fopen\_s** 随后均将使用只写访问权限重新打开文件。 （此行为仅适用于 **a** 模式，不适用于 **a+** 模式。）  字符串 *mode* 指定对文件请求的访问类型，如下所示。  展开表   | ***mode*** | **Access** | | --- | --- | | **"r"** | 打开以便读取。 如果文件不存在或找不到，**fopen\_s** 调用将失败。 | | **"w"** | 打开用于写入的空文件。 如果给定文件存在，则其内容会被销毁。 | | **"a"** | 在文件末尾打开以进行写入（追加），在新数据写入到文件之前不移除文件末尾 (EOF) 标记。 如果文件不存在，则创建文件。 | | **"r+"** | 打开以便读取和写入。 文件必须存在。 | | **"w+"** | 打开用于读取和写入的空文件。 如果文件存在，则其内容会被销毁。 | | **"a+"** | 打开以进行读取和追加。 追加操作包括在新数据写入文件之前移除 EOF 标记。 写入完成后，EOF 标记不会还原。 如果文件不存在，则创建文件。 |   通过使用 **"a"** 或 **"a+"** 访问类型打开文件时，所有写入操作均将在文件末尾进行。 使用 [fseek](https://learn.microsoft.com/zh-cn/cpp/c-runtime-library/reference/fseek-fseeki64?view=msvc-170) 或 [rewind](https://learn.microsoft.com/zh-cn/cpp/c-runtime-library/reference/rewind?view=msvc-170) 可重新定位文件指针，但在执行任何写入操作前，文件指针将始终被移回文件末尾，以确保不会覆盖现有数据。  在 EOF 标记追加到文件之前，**"a"** 模式会将其删除。 追加后，MS-DOS TYPE 命令仅显示原始 EOF 标记之前的数据，不显示追加到文件的任何数据。 在 EOF 标记追加到文件之前，**"a+"** 模式会将其删除。 在追加后，MS-DOS TYPE 命令显示文件中的所有数据。 需使用 **"a+"** 模式才能附加到通过 CTRL+**Z** EOF 标记终止的流文件。  当指定 **"r+"**、**"w+"** 或 **"a+"** 访问类型时，允许读取和写入。 （文件据说已经可以进行“更新”。）但是，当你从读取切换到写入时，输入操作必定会遇到 EOF 标记。 如果没有 EOF 标记，必须使用对文件定位函数的干预调用。 文件定位函数是 **fsetpos**、[fseek](https://learn.microsoft.com/zh-cn/cpp/c-runtime-library/reference/fseek-fseeki64?view=msvc-170) 和 [rewind](https://learn.microsoft.com/zh-cn/cpp/c-runtime-library/reference/rewind?view=msvc-170)。 从写入切换到读取时，必须使用对 **fflush** 或文件定位函数的干预调用。  从 C11 开始，可以将 **"x"** 追加到 **"w"**，或追加 **"w+"** 以使函数在该文件存在时失败，而不是覆盖该文件。  除了上述值之外，可以在 *mode* 中包含以下字符以指定换行符的转换模式：  展开表   | ***mode* 修饰符** | **转换模式** | | --- | --- | | **t** | 在文本（转换）模式下打开。 输入时，回车换行 (CR-LF) 组合将转换为单一的换行 (LF)；输出时，LF 字符将转换为 CR-LF 组合。 CTRL+Z 将在输入时解释为文件尾字符。 | | **b** | 在二进制（未转换）模式下打开；禁止涉及回车和换行字符的转换。 |   在文本（已转换）模式中，CTRL+**Z** 将在输入时解释为文件结尾字符。 在打开使用 **"a+"** 进行读取/写入的文件中，**fopen\_s** 将检查文件末尾的 CTRL+**Z** 并在可能的情况下将其删除。 将其删除是因为使用 [fseek](https://learn.microsoft.com/zh-cn/cpp/c-runtime-library/reference/fseek-fseeki64?view=msvc-170) 和 [ftell](https://learn.microsoft.com/zh-cn/cpp/c-runtime-library/reference/ftell-ftelli64?view=msvc-170) 在以 CTRL+**Z** 结尾的文件中移动时，可能导致 **fseek** 在文件末尾附近错误运行。  另外，在文本模式下，输入时，回车/换行 (CRLF) 组合将转换为单一的换行 (LF) 字符；输出时，LF 字符将转换为 CRLF 组合。 当 Unicode 流 I/O 函数在文本模式（默认设置）下运行时，源或目标流将假定为一系列多字节字符。 Unicode 流输入函数将多字节字符转换为宽字符（就像调用 **mbtowc** 函数一样）。 出于同一原因，Unicode 流输出函数将宽字符转换为多字节字符（就像调用 **wctomb** 函数一样）。  如果 **t** 或 **b** 在 *mode* 中未给出，则默认转换模式由全局变量 [\_fmode](https://learn.microsoft.com/zh-cn/cpp/c-runtime-library/fmode?view=msvc-170) 定义。 如果 **t** 或 **b** 是该参数的前缀，则函数将失败并返回 NULL。  有关在 Unicode 和多字节流 I/O 中使用文本和二进制模式的详细信息，请参阅[文本和二进制模式文件 I/O](https://learn.microsoft.com/zh-cn/cpp/c-runtime-library/text-and-binary-mode-file-i-o?view=msvc-170) 和[文本和二进制模式下的 Unicode 流 I/O](https://learn.microsoft.com/zh-cn/cpp/c-runtime-library/unicode-stream-i-o-in-text-and-binary-modes?view=msvc-170)。  展开表   | ***mode* 修饰符** | **行为** | | --- | --- | | **c** | 启用关联 *filename* 的提交标志，以便在调用 **fflush** 或 **\_flushall** 时将文件缓冲区的内容直接写入磁盘。 | | **n** | 将关联的 *filename* 的提交标志重置为“no-commit”。此标志为默认值。 如果将程序显式链接到 *COMMODE.OBJ*，它还将重写全局提交标志。 除非将程序显式链接到 *COMMODE.OBJ*，否则全局提交标志默认为 "no-commit"（请参阅[链接选项](https://learn.microsoft.com/zh-cn/cpp/c-runtime-library/link-options?view=msvc-170)）。 | | **N** | 指定文件不由子进程继承。 | | **S** | 指定缓存针对（但不限于）从磁盘的顺序访问进行优化。 | | **R** | 指定缓存针对（但不限于）从磁盘的随机访问进行优化。 | | **T** | 指定一个文件，除非内存压力需要它，否则不会写入磁盘。 | | **D** | 指定在关闭最后一个指向该文件的指针时删除的临时文件。 | | **ccs=UNICODE** | 指定 UNICODE 作为要用于此文件的编码字符集。 如果需要 ANSI 编码，请不要指定此字符集。 | | **ccs=UTF-8** | 指定 UTF-8 作为要用于此文件的编码字符集。 如果需要 ANSI 编码，请不要指定此字符集。 | | **ccs=UTF-16LE** | 指定 UTF-16LE 作为要用于此文件的编码字符集。 如果需要 ANSI 编码，请不要指定此字符集。 |   在 **fopen\_s** 和 [\_fdopen](https://learn.microsoft.com/zh-cn/cpp/c-runtime-library/reference/fdopen-wfdopen?view=msvc-170) 中使用的 *mode* 字符串的有效字符对应于在 [\_open](https://learn.microsoft.com/zh-cn/cpp/c-runtime-library/reference/open-wopen?view=msvc-170) 和 [\_sopen](https://learn.microsoft.com/zh-cn/cpp/c-runtime-library/reference/sopen-wsopen?view=msvc-170) 中使用的 *oflag* 参数，如下所示。  展开表   | **字符串中的 *mode* 字符** | **\_open/\_sopen 的等效 *oflag* 值** | | --- | --- | | **a** | \_O\_WRONLY | \_O\_APPEND （通常为 \_O\_WRONLY | \_O\_CREAT | \_O\_APPEND ） | | **a+** | \_O\_RDWR | \_O\_APPEND （通常为 \_O\_RDWR | \_O\_APPEND | \_O\_CREAT ） | | **R** | \_O\_RDONLY | | **r+** | \_O\_RDWR | | **w** | \_O\_WRONLY （通常为 \_O\_WRONLY | \_O\_CREAT | \_O\_TRUNC ） | | **w+** | \_O\_RDWR（通常为 \*\*\_O\_RDWR | \_O\_CREAT | \_O\_TRUNC） | | **b** | \_O\_BINARY | | **t** | \_O\_TEXT（翻译） | | **c** | 无 | | **n** | 无 | | **D** | \_O\_TEMPORARY | | **R** | \_O\_RANDOM | | **S** | \_O\_SEQUENTIAL | | **T** | \_O\_SHORTLIVED | | **ccs=UNICODE** | \_O\_WTEXT | | **ccs=UTF-8** | \_O\_UTF8 | | **ccs=UTF-16LE** | \_O\_UTF16 |   **c**、、**nR**、**S**、**t**和 **T***mode* **D**选项Microsoft扩展fopen\_s\_wfopen\_s，并且不应在需要 ANSI 可移植性时使用。  如果使用 **rb** 模式，并且无需移植代码、预计将读取大量文件或者不关心网络性能，则内存映射的 Win32 文件可能也是一个选项。  关于 T 和 D：   * 只要内存压力不需要，T 就避免将文件写入磁盘。 有关详细信息，请参阅[文件属性常量](https://learn.microsoft.com/zh-cn/windows/win32/fileio/file-attribute-constants)中的 FILE\_ATTRIBUTE\_TEMPORARY，以及此博客文章[它只是暂时性的](https://learn.microsoft.com/archive/blogs/larryosterman/its-only-temporary)。 * D 指定写入磁盘的常规文件。 区别在于它在关闭时会自动删除。 可以组合 TD 来获取这两种语义。  要求 展开表   | **函数** | **必需的标头** | **C++ 标头** | | --- | --- | --- | | **fopen\_s** | <stdio.h> | <cstdio> | | **\_wfopen\_s** | <stdio.h> 或 <wchar.h> | <cstdio> |   有关 C 运行时库中的标准符合性和命名约定的详细信息，请参阅[兼容性](https://learn.microsoft.com/zh-cn/cpp/c-runtime-library/compatibility?view=msvc-170)。 一般文本例程映射 展开表   | **<tchar.h> 例程** | **\_UNICODE 和 \_MBCS 未定义** | **\_MBCS 已定义** | **\_UNICODE 已定义** | | --- | --- | --- | --- | | **\_tfopen\_s** | **fopen\_s** | **fopen\_s** | **\_wfopen\_s** |  库 [C 运行时库](https://learn.microsoft.com/zh-cn/cpp/c-runtime-library/crt-library-features?view=msvc-170)的所有版本。 示例 C复制  // crt\_fopen\_s.c  // This program opens two files. It uses  // fclose to close the first file and  // \_fcloseall to close all remaining files.  #include <stdio.h>  FILE \*stream, \*stream2;  int main( void )  {  errno\_t err;  // Open for read (will fail if file "crt\_fopen\_s.c" doesn't exist)  err = fopen\_s( &stream, "crt\_fopen\_s.c", "r" );  if( err == 0 )  {  printf( "The file 'crt\_fopen\_s.c' was opened\n" );  }  else  {  printf( "The file 'crt\_fopen\_s.c' was not opened\n" );  }  // Open for write  err = fopen\_s( &stream2, "data2", "w+, ccs=UTF-8" );  if( err == 0 )  {  printf( "The file 'data2' was opened\n" );  }  else  {  printf( "The file 'data2' was not opened\n" );  }  // Close stream if it isn't NULL  if( stream )  {  err = fclose( stream );  if ( err == 0 )  {  printf( "The file 'crt\_fopen\_s.c' was closed\n" );  }  else  {  printf( "The file 'crt\_fopen\_s.c' was not closed\n" );  }  }  // All other files are closed:  int numclosed = \_fcloseall( );  printf( "Number of files closed by \_fcloseall: %u\n", numclosed );  }  Output复制  The file 'crt\_fopen\_s.c' was opened  The file 'data2' was opened  Number of files closed by \_fcloseall: 1 另请参阅 [流 I/O](https://learn.microsoft.com/zh-cn/cpp/c-runtime-library/stream-i-o?view=msvc-170) [fclose, \_fcloseall](https://learn.microsoft.com/zh-cn/cpp/c-runtime-library/reference/fclose-fcloseall?view=msvc-170) [\_fdopen, \_wfdopen](https://learn.microsoft.com/zh-cn/cpp/c-runtime-library/reference/fdopen-wfdopen?view=msvc-170) [ferror](https://learn.microsoft.com/zh-cn/cpp/c-runtime-library/reference/ferror?view=msvc-170) [\_fileno](https://learn.microsoft.com/zh-cn/cpp/c-runtime-library/reference/fileno?view=msvc-170) [freopen, \_wfreopen](https://learn.microsoft.com/zh-cn/cpp/c-runtime-library/reference/freopen-wfreopen?view=msvc-170) [\_open, \_wopen](https://learn.microsoft.com/zh-cn/cpp/c-runtime-library/reference/open-wopen?view=msvc-170) [\_setmode](https://learn.microsoft.com/zh-cn/cpp/c-runtime-library/reference/setmode?view=msvc-170) |

## fgetws函数

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 从流获取字符串。 语法 C  char \*fgets(  char \*str,  int numChars,  FILE \*stream  );  wchar\_t \*fgetws(  wchar\_t \*str,  int numChars,  FILE \*stream  ); 参数 *str* 数据的存储位置。  *numChars* 要读取的最大字符数。  *stream* 指向 FILE 结构的指针。 返回值 其中每个函数都会返回 *str*。 将返回 NULL 指示错误或文件尾条件。 使用 **feof** 或 **ferror** 确定是否出错。 如果 *str* 或 *stream* 是空指针，或者 *numChars* 小于或等于零，此函数会调用无效的参数处理程序，如[参数验证](https://learn.microsoft.com/zh-cn/cpp/c-runtime-library/parameter-validation?view=msvc-170)中所述。 如果允许继续执行，则将 errno 设置为 EINVAL 并且该函数返回 NULL中所述。  有关返回代码的详细信息，请参阅 [errno、\_doserrno、\_sys\_errlist 和 \_sys\_nerr](https://learn.microsoft.com/zh-cn/cpp/c-runtime-library/errno-doserrno-sys-errlist-and-sys-nerr?view=msvc-170)。 备注 **fgets** 函数将读取输入 *stream* 参数中的一个字符串，并将其存储到 *str* 中。 **fgets** 会将字符从当前流位置读取到流的结尾（包含第一个换行符），或直到读取的字符数量等于 *numChars* - 1，以先到者为准。 将向存储在 *str* 中的结果追加一个 null 字符。 换行符（如果读取）将包括在字符串中。  **fgetws** 是 **fgets** 的宽字符版本。  当 *stream* 分别以文本模式或二进制模式打开时，**fgetws** 会将宽字符自变量 *str* 作为多字节字符串或宽字符串读取。 若要详细了解如何在 Unicode 和多字节流 I/O 中使用文本和二进制模式，请参阅[文本和二进制模式文件 I/O](https://learn.microsoft.com/zh-cn/cpp/c-runtime-library/text-and-binary-mode-file-i-o?view=msvc-170) 和[文本和二进制模式下的 Unicode 流 I/O](https://learn.microsoft.com/zh-cn/cpp/c-runtime-library/unicode-stream-i-o-in-text-and-binary-modes?view=msvc-170)。  默认情况下，此函数的全局状态范围限定为应用程序。 若要更改此行为，请参阅 [CRT 中的全局状态](https://learn.microsoft.com/zh-cn/cpp/c-runtime-library/global-state?view=msvc-170)。 一般文本例程映射 展开表   | **TCHAR.H 例程** | **\_UNICODE 和 \_MBCS 未定义** | **\_MBCS 已定义** | **\_UNICODE 已定义** | | --- | --- | --- | --- | | **\_fgetts** | **fgets** | **fgets** | **fgetws** |  要求 展开表   | **函数** | **必需的标头** | | --- | --- | | **fgets** | <stdio.h> | | **fgetws** | <stdio.h> 或 <wchar.h> |   有关兼容性的详细信息，请参阅 [兼容性](https://learn.microsoft.com/zh-cn/cpp/c-runtime-library/compatibility?view=msvc-170)。 示例 C复制  // crt\_fgets.c  // This program uses fgets to display  // the first line from a file.  #include <stdio.h>  int main( void )  {  FILE \*stream;  char line[100];  if( fopen\_s( &stream, "crt\_fgets.txt", "r" ) == 0 )  {  if( fgets( line, 100, stream ) == NULL)  printf( "fgets error\numChars" );  else  printf( "%s", line);  fclose( stream );  }  } 输入：crt\_fgets.txt Input复制  Line one.  Line two. 输出 Output复制  Line one. |

## fwprintf函数

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 将格式化数据输出到流。 这些功能有更安全的版本可用；请参阅 [fprintf\_s、\_fprintf\_s\_l、fwprintf\_s、\_fwprintf\_s\_l](https://learn.microsoft.com/zh-cn/cpp/c-runtime-library/reference/fprintf-s-fprintf-s-l-fwprintf-s-fwprintf-s-l?view=msvc-170)。  有关 \_ftprintf 和 \_ftprintf\_l，请参阅 [泛型文本函数映射](https://learn.microsoft.com/zh-cn/cpp/c-runtime-library/reference/fprintf-fprintf-l-fwprintf-fwprintf-l?view=msvc-170#generic-text-function-mappings)。 语法 C复制  int fprintf(  FILE \*stream,  const char \*format [,  argument ]...  );  int \_fprintf\_l(  FILE \*stream,  const char \*format,  \_locale\_t locale [,  argument ]...  );  int fwprintf(  FILE \*stream,  const wchar\_t \*format [,  argument ]...  );  int \_fwprintf\_l(  FILE \*stream,  const wchar\_t \*format,  \_locale\_t locale [,  argument ]...  ); 参数 *stream* 指向 FILE 结构的指针。  *format* 窗体控件字符串。  *argument* 可选参数。  *locale* 要使用的区域设置。 返回值 **fprintf** 返回已写入的字节数。 **fwprintf** 返回已写入的宽字符数。 其中每个函数在出现输出错误时返回一个负值。 如果 *stream* 或 *format* 为 NULL，这些函数会调用无效参数处理程序，如[参数验证](https://learn.microsoft.com/zh-cn/cpp/c-runtime-library/parameter-validation?view=msvc-170)中所述。 如果允许执行继续，则这些函数将返回 -1 并将 errno 设置为 EINVAL。 使用 **fprintf\_s** 或 **fwprintf\_s** 时，不会检查格式字符串原本的格式设置字符是否有效。  有关返回代码的详细信息，请参阅 [errno、\_doserrno、\_sys\_errlist 和 \_sys\_nerr](https://learn.microsoft.com/zh-cn/cpp/c-runtime-library/errno-doserrno-sys-errlist-and-sys-nerr?view=msvc-170)。 备注 **fprintf** 格式化一系列字符和值并将其输出到输出 *stream*。 每个 *argument* 函数（如果有）根据 *format* 中相应的格式规范进行转换和输出。 对于 **fprintf**，*format* 参数具有与 **printf** 中相同的语法。  **fwprintf** 是 **fprintf** 的宽字符版本；在 **fwprintf** 中，*format* 是宽字符字符串。 如果在 ANSI 模式下打开流，则这些函数行为相同。 **fprintf** 当前不支持到 UNICODE 流中的输出。  这些带有 **\_l** 后缀的函数的版本相同，只不过它们使用传递的区域设置参数而不是当前线程区域设置。  **重要**  确保 *format* 不是用户定义的字符串。  从 Windows 10 版本 2004（内部版本 19041）开始，printf 系列函数根据 IEEE 754 的舍入规则输出可精确表示的浮点数。 在早期的 Windows 版本中，以“5”结尾并且可精确表示的浮点数总是向上取整。 IEEE 754 规定它们必须舍入到最接近的偶数（也称为“四舍六入五成双”）。 例如，printf("%1.0f", 1.5) 和 printf("%1.0f", 2.5) 都应舍入为 2。 之前，1.5 舍入为 2，2.5 舍入为 3。 此更改仅影响可精确表示的数字。 例如，2.35（用于内存表示时更接近于 2.35000000000000008）仍然向上取整为 2.4。 这些函数完成的舍入现在也遵循 [**fesetround**](https://learn.microsoft.com/zh-cn/cpp/c-runtime-library/reference/fegetround-fesetround2?view=msvc-170) 设置的浮点舍入模式。 以前，舍入始终选择 FE\_TONEAREST 行为。 此更改仅影响使用 Visual Studio 2019 版本 16.2 及更高版本生成的程序。 若要使用旧的浮点舍入行为，请链接到 [**'legacy\_stdio\_float\_rounding.obj`**](https://learn.microsoft.com/zh-cn/cpp/c-runtime-library/link-options?view=msvc-170)。 泛型文本函数映射 列中的 tchar.h 函数映射到其他列中的函数，具体取决于编译时定义的字符集。  展开表   | **tchar.h 函数** | **\_UNICODE 和 \_MBCS 未定义** | **\_MBCS 已定义** | **\_UNICODE 已定义** | | --- | --- | --- | --- | | \_ftprintf | fprintf | fprintf | fwprintf | | \_ftprintf\_l | \_fprintf\_l | \_fprintf\_l | \_fwprintf\_l |   有关详细信息，请参阅[格式规范语法](https://learn.microsoft.com/zh-cn/cpp/c-runtime-library/format-specification-syntax-printf-and-wprintf-functions?view=msvc-170)。 要求 展开表   | **函数** | **必需的标头** | | --- | --- | | **fprintf**, **\_fprintf\_l** | <stdio.h> | | **fwprintf**, **\_fwprintf\_l** | <stdio.h> 或 <wchar.h> |   有关兼容性的详细信息，请参阅 [兼容性](https://learn.microsoft.com/zh-cn/cpp/c-runtime-library/compatibility?view=msvc-170)。 示例 C复制  // crt\_fprintf.c  /\* This program uses fprintf to format various  \* data and print it to the file named FPRINTF.OUT. It  \* then displays FPRINTF.OUT on the screen using the system  \* function to invoke the operating-system TYPE command.  \*/  #include <stdio.h>  #include <process.h>  FILE \*stream;  int main( void )  {  int i = 10;  double fp = 1.5;  char s[] = "this is a string";  char c = '\n';  fopen\_s( &stream, "fprintf.out", "w" );  fprintf( stream, "%s%c", s, c );  fprintf( stream, "%d\n", i );  fprintf( stream, "%f\n", fp );  fclose( stream );  system( "type fprintf.out" );  }  Output复制  this is a string  10  1.500000 另请参阅 [流 I/O](https://learn.microsoft.com/zh-cn/cpp/c-runtime-library/stream-i-o?view=msvc-170) [\_cprintf、、\_cprintf\_l\_cwprintf、、\_cwprintf\_l](https://learn.microsoft.com/zh-cn/cpp/c-runtime-library/reference/cprintf-cprintf-l-cwprintf-cwprintf-l?view=msvc-170) [fscanf、、\_fscanf\_lfwscanf、、\_fwscanf\_l](https://learn.microsoft.com/zh-cn/cpp/c-runtime-library/reference/fscanf-fscanf-l-fwscanf-fwscanf-l?view=msvc-170) [sprintf、、\_sprintf\_lswprintf、\_swprintf\_l、\_swprintf\_l](https://learn.microsoft.com/zh-cn/cpp/c-runtime-library/reference/sprintf-sprintf-l-swprintf-swprintf-l-swprintf-l?view=msvc-170) [格式规范语法：printf 和 wprintf 函数](https://learn.microsoft.com/zh-cn/cpp/c-runtime-library/format-specification-syntax-printf-and-wprintf-functions?view=msvc-170) |

## 1.新建一个控制台应用程序,取名lesson9-widechar-file-op,然后新建一个cpp文件取名:main.cpp

|  |
| --- |
|  |

## 2.main.cpp的代码如下

|  |
| --- |
| #include<stdio.h>  #include<locale.h>  #include<string.h>  #include<stdlib.h>  void SaveTextFile(const char\* fileName)  {  setlocale(LC\_ALL, "");  const wchar\_t\* wstr = L"热烈庆祝伟大的中华人民共和国成立75周年";  FILE\* fin;  fopen\_s(&fin, fileName, "w, ccs=utf-8");  fwprintf(fin, L"%s", wstr);  fclose(fin);  }  void SaveTextFileUnicode(const wchar\_t\* fileName)  {  setlocale(LC\_ALL, "");  const wchar\_t\* wstr = L"再次热烈庆祝伟大的中华人民共和国成立75周年";  FILE\* fin;  \_wfopen\_s(&fin, fileName, L"w, ccs=utf-8");  fwprintf(fin, L"%s", wstr);  fclose(fin);  }  void ReadTextFileUnicode(const wchar\_t\* fileName)  {  setlocale(LC\_ALL, "");  FILE\* fin;  \_wfopen\_s(&fin, fileName, L"r, ccs=utf-8");  wchar\_t szContent[100];  memset(szContent, 0, 100);  fgetws(szContent, 100, fin);  wprintf(L"文件内容:\n");  wprintf(L"%s\n\n", szContent);  fclose(fin);  }  int wmain()  {  //SaveTextFile("guoqing.txt");  //SaveTextFileUnicode(L"gq.txt");  ReadTextFileUnicode(L"gq.txt");  system("pause");  return 0;  } |

### 效果:调用保存函数,就可以生成一个gq.txt文件,调用读取函数,就能够在控制台显示文件内容

|  |
| --- |
|  |
|  |

# 这一节就学习到这里,当然还有许多宽字符函数，需要慢慢学习和练习