**宽字符概念**

        宽字符又称多字节字符又称 Unicode ，在我们在日常写代码的过程中，大部分接触的是ASCII 字符，那么对于字符变量，我们常用的数据类型是 char ，但是对于中文字符，用 char 变量就不行了，那么就需要使用宽字符了。

        也就是说一般的字符是一个字节组成的，如 0x0a,0x38, 那么对于普通字符， 0x00 就可以用来表示字符串结束符，而宽字符通常是由两个字节组成的，例如:0x0061(97) 依然是用来表示字符 a, 不过它是由两个字节组成，这样一个宽字符最多可表示 65535 个字符，包含了世界上大多数的字样表示。它的类型应该声明为 **wchar\_t**在 VC++6.0 之前一直以 unsigned short 代替。

        在定义宽字符或宽字符串时使用前缀 L 标识即可，例如：

1. wchar\_t ch = L'你';
2. wchar\_t str[128] = L"你好";

        使用宽字符变量定义中文变量之前，先要使用 setlocale 函数来设置地域，他的作用相当于在控制台窗口设置好当前代码页的编码方式。

**设置或读取地域化信息 api**

**头文件：**

*#include <locale.h>*

**头文件中的 lconv 结构体：**

1. typedef struct {
2. char \*decimal\_point; //用于非货币值的小数点字符
3. char \*thousands\_sep; //用于非货币值的千位分隔符
4. char \*grouping; //一个表示非货币量中每组数字大小的字符串。
5. char \*int\_curr\_symbol; //国际货币符号使用的字符串。
6. char \*currency\_symbol; //用于货币的本地符号
7. char \*mon\_decimal\_point; //用于货币值的小数点字符。
8. char \*mon\_thousands\_sep; //用于货币值的千位分隔符
9. char \*mon\_grouping; //用于货币值的千位分隔符
10. char \*positive\_sign; //用于正货币值的字符
11. char \*negative\_sign; //用于负货币值的字符
12. char int\_frac\_digits; //国际货币值中小数点后要显示的位数
13. char frac\_digits; //货币值中小数点后要显示的位数
14. char p\_cs\_precedes; //如果等于 1，则 currency\_symbol 出现在正货币
15. 值之前。如果等于 0，则 currency\_symbol 出现在正货币值之后。
16. char p\_sep\_by\_space; //如果等于 1，则 currency\_symbol 和正货币值
17. 之间使用空格分隔。如果等于 0，则 currency\_symbol 和正货币值之间不使用
18. 空格分隔。
19. char n\_cs\_precedes; //如果等于 1，则 currency\_symbol 出现在负货币
20. 值之前。如果等于 0，则 currency\_symbol 出现在负货币值之后。
21. char n\_sep\_by\_space; //如果等于 1，则 currency\_symbol 和负货币值
22. 之间使用空格分隔。如果等于 0，则 currency\_symbol 和负货币值之间不使用
23. 空格分隔。
24. char p\_sign\_posn; //表示正货币值中正号的位置。
25. char n\_sign\_posn; //表示负货币值中负号的位置。
26. } lconv

https://csdnimg.cn/release/blogv2/dist/pc/img/newCodeMoreWhite.png

**setlocale 函数**

**函数原型：**

char \*setlocale(int category, const char \*locale)

**描述：**

        设置或读取地域化信息。

**参数** ：

**category**: 这是一个已命名的常量，指定了受区域设置影响的函数类别,类别宏如下





**locale**: 用来设置地域设置的名称（字符串），也就是设置为哪种地域，对于不同的平台和不同的 编译器，地域设置的名称可能会不同，C 语言标准没有干预太多。C 语言标准只是规定，各个组织在实现编译器时至少要支持以下**三**个名称



**返回值** ：

        1. 如果 setlocale() 执行 **成功** ，那么返回一个指向字符串的指针，该字符串包含了当前地域设置的名称。也就是说，setlocale() 会将当前地域设置的名称返回

        2.如果 setlocale() 执行 **失败** （例如为 locale 指定的名称不存在，就会导致地域设置失败），那么返回空指针 NULL

**备注** ：

        无

**使用示例：**

**1.** **查看地域设置**

1. int main (int argc, char \*argv[])
2. {
3. char \*p = setlocale(LC\_ALL, NULL);
4. printf("%s\n", p);
5. }

https://i-blog.csdnimg.cn/blog_migrate/95961eeded640f75d97e68953b2e2ba3.png

        这样的地域设置如果打印宽字符中文将不会打印出我们需要的文字，这时我们就要设置操作系统为中文地域，setlocale(LC\_ALL,””); 就可以了。

**localeconv 函数**

**函数原型：**

struct lconv \*localeconv(void)

**描述：**

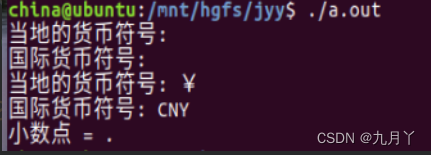
        设置或读取地域化信息。它会返回一个 **lconv**结构类型的对象。查看与货币相关的地域信息

**返回值** ：

        该函数返回一个指向当前区域 **struct lconv**的指针

**使用示例** ：

1. int main ()
2. {
3. struct lconv \* lc;
4. setlocale(LC\_MONETARY, "C");
5. lc = localeconv();
6. printf ("当地的货币符号: %s\n",lc->currency\_symbol);
7. printf ("国际货币符号: %s\n",lc->int\_curr\_symbol);
8. setlocale(LC\_MONETARY, "");
9. lc = localeconv();
10. printf ("当地的货币符号: %s\n",lc->currency\_symbol);
11. printf ("国际货币符号: %s\n",lc->int\_curr\_symbol);
12. printf("小数点 = %s\n", lc->decimal\_point);
13. return 0;
14. }



**宽字符相关 API**

**宽字符串长度函数 wcslen**

**头文件：**

*#include <wchar.h>*

**函数原型** ：

size\_t wcslen(const wchar\_t \*s);

**参数：**

        s:宽字符串地址

**返回值：**

        返回宽字符的个数

**备注** ：

        宽字符的个数不等同于它的字节数

**示例** ：

1. int main ()
2. {
3. wchar\_t str[128] = L"你好 word";
4. printf("%ld\n",wcslen(str));
5. return 0;
6. }

https://i-blog.csdnimg.cn/blog_migrate/91eccc5bdd6993c24230cdbde222bdf7.png

**宽字符输出函数 wprintf**

**头文件：**

*#include <wchar.h>*

*#include <stdio.h>*

**函数原型** ：

int wprintf(const wchar\_t \*format, ...);

**参数：**

**format:**格式化宽字符串

**返回值：**

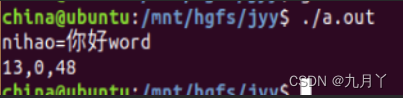
        返回打印的宽字符个数

**备注** ：

        宽字符串使用%ls, 宽字符使用 %lc.wprintf 也可以完全当成 printf 去使用但是输出的格式化字符串是宽字符而已，而且使用 wprintf 会使得 printf 变得失效。

**示例** ：

1. int main ()
2. {
3. wchar\_t ch = L'0';
4. setlocale(LC\_ALL, "");
5. wchar\_t str[128] = L"你好 word";
6. int t = wprintf(L"nihao=%ls\n",str);
7. wprintf(L"%d,%lc,%d\n",t,ch,ch);
8. return 0;
9. }



**宽字符输入函数 wscanf**

**头文件：**

*#include <wchar.h>*

*#include <stdio.h>*

**函数原型** ：

int wscanf(const wchar\_t \*format, ...);

**参数：**

**format:**格式化宽字符串

**返回值：**

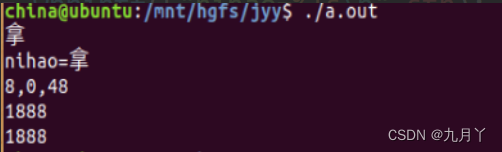
        返回输入的宽字符个数

**备注** ：

        宽字符串使用%ls, 宽字符使用 %lc.wscanf 也可以完全当成 scanf 去使用，但是输入的格式化字符串是宽字符而已，而且使用 wscanf 会使得 scanf 变得失效。

**示例** ：

1. int main ()
2. {
3. wchar\_t ch = L'0';
4. setlocale(LC\_ALL, "");
5. wchar\_t str[128];
6. wscanf(L"%ls",str);
7. int t = wprintf(L"nihao=%ls\n",str);
8. wprintf(L"%d,%lc,%d\n",t,ch,ch);
9. wscanf(L"%d",&t);
10. wprintf(L"%d\n",t);
11. return 0;
12. }



**宽字符比较函数 wcscmp**

**头文件：**

*#include <wchar.h>*

**函数原型** ：

int wcscmp(const wchar\_t \*s1, const wchar\_t \*s2);

**参数：**

        两个要比较的宽字符串

**返回值：**

        s1 == s2 返回 0

        s1 > s2 返回值大于 0

        s1 < s2 返回值小于 0

**备注** ：

        无

**宽字符拷贝函数 wcscpy**

**头文件：**

*#include <wchar.h>*

**函数原型** ：

wchar\_t \*wcscpy(wchar\_t \*dest, const wchar\_t \*src);

**描述：**

        把 src 指向的宽字符串拷贝到 dest 指向的地址

**返回值：**

        成功返回拷贝后的地址，失败返回 null

**备注** ：

        无

**宽字符连接函数 wcscat**

**头文件：**

*#include <wchar.h>*

**函数原型** ：

wchar\_t \*wcscat(wchar\_t \*dest, const wchar\_t \*src);

**描述：**

        把 src 指向的宽字符串连接到 dest 指向的宽字符串后面

**返回值：**

        成功返回连接后的宽字符地址，失败返回 null

**备注** ：

        无

**宽字符查找子宽字符串函数 wcsstr**

**头文件：**

*#include <wchar.h>*

**函数原型** ：

wchar\_t \*wcsstr(const wchar\_t \*haystack, const wchar\_t \*needle);

**描述：**

        查找 haystack 中是否有子宽字符串 needle ，如果有返回首次出现的地址

**返回值：**

        如果有返回首次出现的地址，失败返回 null

**备注** ：

        无

**宽字符和字符之间的转换**

        在头文件#include<stdlib.h> 中有几个 api 可以实现宽字符转换成字符的操作，我们称转换后的宽字符为多字节字符，其中有：

**wbstowcs 函数** ：可以将多字节字符转换为宽字符

**mblen 函数** ：可以计算多字节字符的长度

**wcstombs 函数** ：可以将宽字符转换为多字节字符

**mbtowc 函数** ：可以将多字节序列转换为宽字符

**wctomb 函数** ：可以将宽字符转换为多字节字符， 并把它存储在字符数组的开头