第17讲：字符函数和字符串函数

目录：

正文开始

在编程的过程中，我们经常要处理字符和字符串，为了方便操作字符和字符串，C语言标准库中提供了一系列库函数，接下来我们就学习一下这些函数。

1. 字符分类函数

C语言中有一系列的函数是专门做字符分类的，也就是一个字符是属于什么类型的字符的。

这些函数的使用都需要包含一个头文件是 `ctype.h`

函数	如果他的参数符合下列条件就返回真
iscntrl	任何控制字符
isspace	空白字符：空格‘ ’，换页‘\f’，换行'\n'，回车‘\r’，制表符'\t'或者垂直制表符'\v'
isdigit	十进制数字 ‘0’ ~ ‘9’ 字符
isxdigit	十六进制数字，包括所有十进制数字字符，小写字母a~f，大写字母A~F
islower	小写字母a~z
isupper	大写字母A~Z
isalpha	字母a~z或A~Z
isalnum	字母或者数字，a~z,A~Z,0~9
ispunct	标点符号，任何不属于数字或者字母的图形字符（可打印）
isgraph	任何图形字符
isprint	任何可打印字符，包括图形字符和空白字符

这些函数的使用方法非常类似，我们就讲解一个函数的事情，其他的非常类似：

代码块

```
int islower ( int c );
```

`islower` 是能够判断参数部分的 `c` 是否是小写字母的。

通过返回值来说明是否是小写字母，如果是小写字母就返回非0的整数，如果不是小写字母，则返回0。

练习：

> 写一个代码，将字符串中的小写字母转大写，其他字符不变。

代码块

```
#include <stdio.h>
#include <ctype.h>
int main ()
{
    int i = 0;
    char str[] = "Test String.\n";
    char c;
    while (str[i])
    {
        c = str[i];
```

```
11          if (islower(c))
12              c -= 32;
13          putchar(c);
14          i++;
15      }
16      return 0;
17  }
```

2. 字符转换函数

C语言提供了2个字符转换函数：

代码块

```
1  int tolower ( int c ); //将参数传进去的大写字母转小写
2  int toupper ( int c ); //将参数传进去的小写字母转大写
```

上面的代码，我们将小写转大写，是-32完成的效果，有了转换函数，就可以直接使用 `tolower` 函数。

代码块

```
1  #include <stdio.h>
2  #include <ctype.h>
3
4  int main ()
5  {
6      int i = 0;
7      char str[] = "Test String.\n";
8      char c;
9      while (str[i])
10     {
11         c = str[i];
12         if (islower(c))
13             c = toupper(c);
14         putchar(c);
15         i++;
16     }
17     return 0;
18 }
```

3. strlen

代码块

```
size_t strlen ( const char * str );
```

功能：统计参数 `str` 指向的字符串的长度。统计的是字符串中 `'\0'` 之前的字符的个数。

参数：

- `str`：指针，指向了要统计长度的字符串。

返回值：返回了 `str` 指向的字符串的长度，返回的长度不会是负数，所以返回类型是 `size_t`。

3.1 代码演示

代码块

```
#include <stdio.h>
#include <string.h>

int main()
{
    const char* str = "abcdef";
    printf("%zd\n", strlen(str));
    return 0;
}
```

使用注意事项：

- 字符串以 `'\0'` 作为结束标志，strlen函数返回的是在字符串中 `'\0'` 前面出现的字符个数（不包含 `'\0'`）。
- 参数指向的字符串必须要以 `'\0'` 结束。
- 注意函数的返回值为 `size_t`，是无符号的（`易错`）
- `strlen` 的使用需要包含头文件 `<string.h>`

3.2 strlen返回值

代码块

```
#include <stdio.h>
#include <string.h>

int main()
{
    const char* str1 = "abcdef";
    const char* str2 = "bbb";
    if(strlen(str2) - strlen(str1) > 0)
```

```
    {
        printf("str2 > str1\n");
    }
    else
    {
        printf("srt1 > str2\n");
    }
    return 0;
}
```

3.3 strlen的模拟实现

方式1：

代码块

```
//计数器方式
int my_strlen(const char * str)
{
    int count = 0;
    assert(str);
    while(*str)
    {
        count++;
        str++;
    }
    return count;
}
```

方式2：

代码块

```
//不能创建临时变量计数器
int my_strlen(const char * str)
{
    assert(str);
    if(*str == '\0')
        return 0;
    else
        return 1 + my_strlen(str+1);
}
```

方式3：

```
代码块
1  //指针-指针的方式
2  int my_strlen(char *s)
3  {
4      assert(str);
5      char *p = s;
6      while(*p != '\0')
7            p++;
8      return p - s;
9  }
```

4. strcpy

```
代码块
1  char* strcpy(char * destination, const char * source );
```

功能：字符串拷贝，拷贝到源头字符串中的 `\0` 为止。

参数：

`destination`：指针，指向目的地空间

`source`：指针，指向源头数据

返回值：

`strcpy` 函数返回的目标空间的起始地址

4.1 代码演示

```
代码块
1  #include <stdio.h>
2  #include <string.h>
3
4  int main()
5  {
6      char arr1[10] = {0};
7      char arr2[] = "hello";
8      strcpy(arr1, arr2);
9      printf("%s\n", arr1);
10
11     return 0;
12 }
```

使用注意事项：

- 源字符串必须以 `'\0'` 结束。
- 会将源字符串中的 `'\0'` 拷贝到目标空间。
- 目标空间必须足够大，以确保能存放源字符串。
- 目标空间必须可修改。

4.2 模拟实现

代码块

```c
#include <stdio.h>
#include <assert.h>

//1.参数顺序
//2.函数的功能，停止条件
//3.assert
//4.const修饰指针
//5.函数返回值
char* my_strcpy(char *dest, const char*src)
{
    char *ret = dest;
    assert(dest != NULL);
    assert(src != NULL);

    while((*dest++ = *src++))
    {
        ;
    }
    return ret;
}

int main()
{
    char arr1[10] = {0};
    char arr2[] = "hello";
    my_strcpy(arr1, arr2);
    printf("%s\n", arr1);

    return 0;
}
```

5. strcat

代码块

```
char * strcat ( char * destination, const char * source );
```

功能：字符串追加，把 `source` 指向的源字符串中的所有字符都追加到 `destination` 指向的空间中。

参数：

`destination`：指针，指向目的地空间

`source`：指针，指向源头数据

返回值：

`strcat` 函数返回的目标空间的起始地址

5.1 代码演示

代码块

```
#include <stdio.h>
#include <string.h>

int main()
{
    char arr1[20] = "hello ";
    char arr2[] = "world";
    strcat(arr1, arr2);
    printf("%s\n", arr1);
    return 0;
}
```

使用注意事项：

- 源字符串必须以 `'\0'` 结束。
- 目标字符串中也得有 `\0` ，否则没办法知道追加从哪里开始。
- 目标空间必须有足够的大，能容纳下源字符串的内容。
- 目标空间必须可修改。

5.2 模拟实现

代码块

```c
#include <stdio.h>
#include <assert.h>

char* my_strcat(char *dest, const char*src)
{
    char *ret = dest;
    assert(dest != NULL);
    assert(src != NULL);

    while(*dest)
    {
        dest++;
    }
    while((*dest++ = *src++))
    {
        ;
    }
    return ret;
}

int main()
{
    char arr1[20] = "hello ";
    char arr2[] = "world";
    my_strcat(arr1, arr2);
    printf("%s\n", arr1);
    return 0;
}
```

6. strcmp

代码块

```c
int strcmp ( const char * str1, const char * str2 );
```

功能：用来比较 `str1` 和 `str2` 指向的字符串，从两个字符串的第一个字符开始比较，如果两个字符的ASCII码值相等，就比较下一个字符。直到遇到不相等的两个字符，或者字符串结束。

参数：

`str1`：指针，指向要比较的第一个字符串

`str2`：指针，指向要比较的第二个字符串

返回值：

- 标准规定：
  - 第一个字符串大于第二个字符串，则返回大于0的数字
  - 第一个字符串等于第二个字符串，则返回0
  - 第一个字符串小于第二个字符串，则返回小于0的数字

6.1 代码演示：

代码块

```c
#include <stdio.h>
#include <string.h>

int main()
{
    char arr1[] = "abcdef";
    char arr2[] = "abq";
    int ret = strcmp(arr1, arr2);
    printf("%d\n", ret);

    if(ret > 0)
        printf("arr1 > arr2\n");
    else if(ret == 0)
        printf("arr1 == arr2\n");
    else
        printf("arr1 < arr2\n");

    return 0;
}
```

6.2 模拟实现：

代码块

```c
int my_strcmp (const char * str1, const char * str2)
{
    int ret = 0 ;
    assert(str1 != NULL);
    assert(str2 != NULL);
    while(*str1 == *str2)
    {
```

```
        if(*str1 == '\0')
            return 0;
        str1++;
        str2++;
    }
    return *str1-*str2;
}
```

7. strncpy

代码块

```
char * strncpy ( char * destination, const char * source, size_t num );
```

功能：字符串拷贝；将 `source` 指向的字符串拷贝到 `destination` 指向的空间中，最多拷贝 `num` 个字符。

参数：

`destination`：指针，指向目的地空间

`source`：指针，指向源头数据

`num`：从source指向的字符串中最多拷贝的字符个数

返回值：

`strncpy` 函数返回的目标空间的起始地址

7.1 代码演示

代码块

```
#include <stdio.h>
#include <string.h>

int main()
{
    char arr1[20] = {0};
    char arr2[] = "abcdefghi";
    char* str = strncpy(arr1, arr2, 5);
    printf("%s\n", arr1);
    printf("%s\n", str);

    return 0;
}
```

7.2 比较strcpy和strncpy函数

- `strcpy` 函数拷贝到 `\0` 为止，如果目标空间不够的话，容易出现越界行为。
- `strncpy` 函数指定了拷贝的长度，源字符串不一定要有 `\0` ，同时在设计参数的时候，就会多一层思考：目标空间的大小是否够用， `strncpy` 相对 `strcpy` 函数更加安全。

8. strncat

代码块

```
char * strncat ( char * destination, const char * source, size_t num );
```

功能：字符串追加；将 `source` 指向的字符串的内容，追加到 `destination` 指向的空间，最多追加 `num` 个字符。

参数：

`destination` ：指针，指向了目标空间

`source` ：指针，指向了源头数据

`num` ：最多追加的字符的个数

返回值：返回的是目标空间的起始地址

8.1 代码演示

代码块

```
#include <stdio.h>
#include <string.h>

int main()
{
    char arr1[20] = "hello ";
    char arr2[] = "world";
    char* str = strncat(arr1, arr2, 5);
    printf("%s\n", arr1);
    printf("%s\n", str);

    return 0;
}
```

8.2 strcat和strncat对比

- 参数不同，`strncat`多了一个参数
- `strcat`函数在追加的时候要将源字符串的所有内容，包含`\0`都追加过去，但是`strncat`函数指定了追加的长度。
- `strncat`函数中源字符串中不一定要有`\0`了。
- `strncat`更加灵活，也更加安全。

9. strncmp

代码块

```
1   int strncmp ( const char * str1, const char * str2, size_t num );
```

功能：字符串比较；比较`str1`和`str2`指向的两个字符串的内容，最多比较`num`字符。

参数：

`str1`：指针，指向一个比较的字符串

`str2`：指针，指向另外一个比较的字符串

`num`：最多比较的字符个数

返回值：

- 标准规定：
  - 第一个字符串大于第二个字符串，则返回大于0的数字
  - 第一个字符串等于第二个字符串，则返回0
  - 第一个字符串小于第二个字符串，则返回小于0的数字

Return Value

Returns an integral value indicating the relationship between the strings:

return value	indicates
<0	the first character that does not match has a lower value in ***str1*** than in ***str2***
0	the contents of both strings are equal
>0	the first character that does not match has a greater value in ***str1*** than in ***str2***

9.1 代码演示

代码块

```
#include <stdio.h>
#include <string.h>

int main()
{
    char arr1[] = "abcdef";
    char arr2[] = "abcqw";

    int ret1 = strncmp(arr1, arr2, 3);
    printf("%d\n", ret1);
    int ret2 = strncmp(arr1, arr2, 4);
    printf("%d\n", ret2);

    return 0;
}
```

9.2 strcmp和strncmp比较

- 参数不同
- strncmp可以比较任意长度了
- strncmp函数更加灵活，更加安全

10. strstr

代码块

```
char * strstr ( const char * str1, const char * str2);
```

功能：

`strstr` 函数，查找 `str2` 指向的字符串在 `str1` 指向的字符串中第一次出现的位置。

简而言之：在一个字符串中查找子字符串。

`strstr` 的使用得包含<string.h>

参数：

`str1`：指针，指向了被查找的字符串

`str2`：指针，指向了要查找的字符串

返回值：

- 如果str1指向的字符串中存在str2指向的字符串，那么返回第一次出现位置的指针
- 如果str1指向的字符串中不存在str2指向的字符串，那么返回NULL

10.1 代码演示

代码块

```c
/* strstr example */
#include <stdio.h>
#include <string.h>
int main ()
{
    char str[] ="This is a simple string";
    char * pch;
    pch = strstr (str,"simple");
    if (pch != NULL)
        printf("%s\n", pch);
    else
        printf("查找的字符串不存在\n");
    return 0;
}
```

10.2 strstr的模拟实现

代码块

```c
char * strstr (const char * str1, const char * str2)
{
    char *cp = (char *) str1;
    char *s1, *s2;
    //特殊情况：str2是空字符串时，直接返回str1
    if ( !*str2 )
        return((char *)str1);

    while (*cp)
    {
        s1 = cp;
        s2 = (char *) str2;

        while ( *s1 && *s2 && !(*s1-*s2) )
            s1++, s2++;

        if (!*s2)
```

```
            return(cp); //返回第一次出现的起始

        cp++;
    }

    return(NULL); //找不到则返回NULL
}
```

strstr函数的实现有多种，可以暴力查找，也有一种高效一些的算法：KMP，有兴趣的可以去学习。

11. strtok 函数的使用

代码块

```
char * strtok(char * str, const char *delim);
```

功能

- **分割字符串**：根据 `delim` 参数中指定的分隔符，将输入字符串 `str` 拆分成多个子字符串。
- **修改原始字符串**：`strtok` 会直接在原始字符串中插入 `'\0'` 终止符，替换分隔符的位置，因此原始字符串会被修改。

参数

1. `str`：首次调用时传入待分割的字符串；后续调用传入 `NULL`，表示继续分割同一个字符串。
2. `delim`：包含所有可能分隔符的字符串（每个字符均视为独立的分隔符）。

返回值

- 成功时返回指向当前子字符串的指针。
- 没有更多子字符串时返回 `NULL`。

使用步骤

1. **首次调用**：传入待分割字符串和分隔符。
2. **后续调用**：传入 `NULL` 和相同的分隔符，继续分割。
3. **结束条件**：当返回 `NULL` 时，表示分割完成。

11.1 代码演示

代码块

```c
#include <stdio.h>
#include <string.h>

int main()
{
    char arr[] = "192.168.6.111";
    const char* sep = ".";
    const char* str = NULL;

    char  buf[30] = {0};
    strcpy(buf, arr); //将arr中的字符串拷贝到buf中，对buf的内容进行切割
    for (str = strtok(buf, sep); str != NULL; str = strtok(NULL, sep))
    {
        printf("%s\n", str);
    }
    return 0;
}
```

11.2 注意事项

- **破坏性操作**：`strtok` 会直接修改原始字符串，将其中的分隔符替换为 `'\0'`。如果需要保留原字符串，应先拷贝一份。
- **连续分隔符**：多个连续的分隔符会被视为单个分隔符，不会返回空字符串。
- **空指针处理**：如果输入的 `str` 为 `NULL` 且没有前序调用，行为未定义。

12. strerror 函数的使用

代码块

```c
char* strerror ( int errnum );
```

功能：

1. `strerror` 函数可以通过参数部分的 `errnum` 表示错误码，得到对应的错误信息，并且返回这个错误信息字符串首字符的地址。
2. `strerror` 函数只针对标准库中的函数发生错误后设置的错误码的转换。
3. `strerror` 的使用需要包含<string.h>

> 在不同的系统和C语言标准库的实现中都规定了一些错误码，一般是放在 `errno.h` 这个头文件中说明的，C语言程序启动的时候就会使用一个全局的变量 `errno` 来记录程序的当前错误码，只不过程序启动的时候 `errno` 是 0，表示没有错误，当我们在使用标准库中的函数的时候发生了某种错误，

> 就会将对应的错误码，存放在 `errno` 中，而一个错误码的数字是整数，很难理解是什么意思，所以每一个错误码都是有对应的错误信息的。strerror函数就可以将错误码对应的错误信息字符串的地址返回。

参数：

`errnum` ：表示错误码

这个错误码一般传递的是 `errno` 这个变量的值，在C语言有一个全局的变量叫： `errno` ，当库函数的调用发生错误的时候，就会将本次错误的错误码存放在 `errno` 这个变量中，使用这个全局变量需要包含一个头文件 `errno.h` 。

返回值：

函数返回通过错误码得到的错误信息字符串的首字符的地址。

12.1 代码演示

代码块

```c
#include <errno.h>
#include <string.h>
#include <stdio.h>

//我们打印一下0~10这些错误码对应的信息
int main()
{
    int i = 0;
    for (i = 0; i <= 10; i++) {
        printf("%d: %s\n", i,  strerror(i));
    }
    return 0;
}
```

在Windows11 + VS2022环境下输出的结果如下：

代码块

```
0: No error
1: Operation not permitted
2: No such file or directory
3: No such process
4: Interrupted function call
5: Input/output error
6: No such device or address
7: Arg list too long
8: Exec format error
```

```
9: Bad file descriptor
10: No child processes
```

举例：

代码块

```c
#include <stdio.h>
#include <string.h>
#include <errno.h>
int main ()
{
    FILE * pFile = NULL;
    //fopen函数以读的形式打开文件，如果文件不存在，则打开失败。
    pFile = fopen ("unexist.ent", "r");
    if (pFile == NULL)
    {
        printf ("错误信息是: %s\n", strerror(errno));
        return 1;//错误返回
    }

    return 0;
}
```

输出：

代码块

```
错误信息是: No such file or directory
```

12.2 perror

代码块

```c
void perror ( const char * str );
```

也可以了解一下 `perror` 函数， `perror` 函数相当于一次将上述代码中的第11行完成了，直接将错误信息打印出来。 `perror` 函数打印完参数部分的字符串后，再打印一个冒号和一个空格，再打印错误信息。

代码块

```c
#include <stdio.h>
```

```c
#include <string.h>
#include <errno.h>

int main ()
{
    FILE * pFile = NULL;
    pFile = fopen ("unexist.ent", "r");
    if (pFile == NULL)
    {
        perror("错误信息是");
        return 1;
    }
    return 0;
}
```

输出：

代码块

```
错误信息是: No such file or directory
```

完