# 字符串

2020年7月19日 15:05

## 字符数组

- char word[] = {'H', 'e', 'l', 'l', 'o','!'};

| | |
|---|---|
| word[0] | H |
| word[1] | e |
| word[2] | l |
| word[3] | l |
| word[4] | o |
| word[5] | ! |

这不是C语言的字符串，因为不能用字符串的方式做计算

## 字符串

- char word[] = {'H', 'e', 'l', 'l', 'o','!','\0'};

| | |
|---|---|
| word[0] | H |
| word[1] | e |
| word[2] | l |
| word[3] | l |
| word[4] | o |
| word[5] | ! |
| word[6] | \0 |

## 字符串

'0'是个字符！'\0'是整数0（一个字节），0则4个字节

- 以0（整数0）结尾的一串字符
  - 0或'\0'是一样的，但是和'0'不同
- 0标志字符串的结束，但它不是字符串的一部分
  - 计算字符串长度的时候不包含这个0
- 字符串以数组的形式存在，以数组或指针的形式访问
  - 更多的是以指针的形式
- string.h 里有很多处理字符串的函数

## 字符串变量

都是字符数组的变量，表现形式不同

- char *str = "Hello";
- char word[] = "Hello";
- char line[10] = "Hello";

占的空间是6，加了个0

## 字符串常量

- "Hello"
- "Hello" 会被编译器变成一个字符数组放在某处，这个数组的长度是6，结尾还有表示结束的0
- 两个相邻的字符串常量会被自动连接起来
- 行末的\表示下一行还是这个字符串常量

```
3  int main()
4  {
5      printf("请分别输入身高的英尺和英寸（\
6          如输入\"5 7\"表示5英尺7英寸: ");
7
8      double foot;
9      double inch;
10
11     scanf("%lf %lf", &foot, &inch);
12
13     printf("身高是%f米。\n",
14         ((foot + inch / 12) * 0.3048));
15
16     return 0;
17 }
```

表示下面依然是这个字符串里面的，但是会有tab在里面。如果下面用 "" ，相邻字符串会连在一起，不会有Tab了

```
请分别输入身高的英尺和英寸，    如输入"5 7"表示5英尺7英寸: 身高是0.000000米。
[Finished in 0.9s]
```

## 字符串

- C语言的字符串是以字符数组的形态存在的
  - 不能用运算符对字符串做运算
  - 通过数组的方式可以遍历字符串
- 唯一特殊的地方是字符串字面量可以用来初始化字符数组

## 字符串常量

我要指向这个字符串

```
char* s = "Hello, world!";
```

- s 是一个指针，初始化为指向一个字符串常量
  - 由于这个常量所在的地方，所以实际上s是 const char* s ，但是由于历史的原因，编译器接受不带const的写法
  - 但是试图对s所指的字符串做写入会导致严重的后果
- 如果需要修改字符串，应该用数组

- 通过数组的方式可以遍历字符串
- 唯一特殊的地方是字符串字面量可以用来初始化字符数组
- 以及标准库提供了一系列字符串函数

char* s，但是由于历史的原因，编译器接受不带const的写法

- 但是试图对s所指的字符串做写入会导致严重的后果
- 如果需要修改字符串，应该用数组：

```
char s[] = "Hello, world!";
```

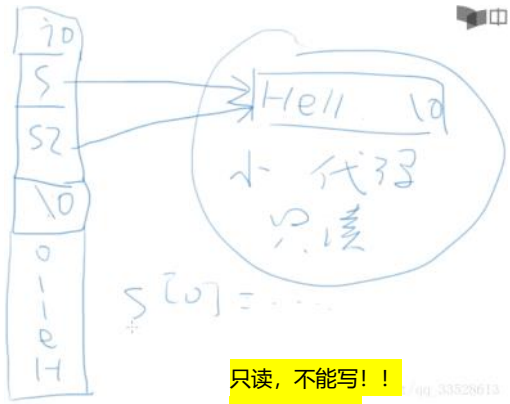
这个字符串就在我这里

```
3 int main(void)
4 {
5     char *s = "Hello World";
6     s[0] = 'B';
7
8     printf("Here!s[0]=%c\n", s[0]);
9
10    return 0;
11 }
```

```
bash: line 1: 30563 Bus error: 10
[Finished in 0.4s with exit code 138]
```

```
3 int main(void)
4 {
5     int i=0;
6     char *s = "Hello World";
7     // s[0] = 'B';
8     char *s2 = "Hello World";
9
10    printf("&i=%p\n", &i);
11    printf("s =%p\n", s);
12    printf("s2=%p\n", s2);
13    printf("Here!s[0]=%c\n", s[0]);
14
15    return 0;
16 }
```

```
&i=0xbff1fd6c
s =0xe1f82
s2=0xe1f82
Here!s[0]=H
```

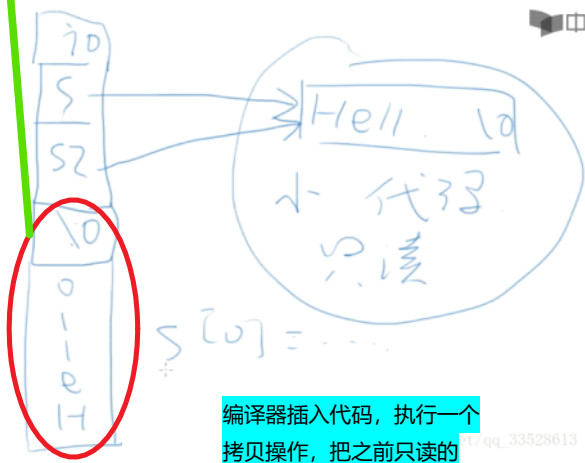



只读，不能写！！
OS的保护机制

```
3 int main(void)
4 {
5     int i=0;
6     char *s = "Hello World";
7     // s[0] = 'B';
8     char *s2 = "Hello World";
9     char s3[] = "Hello World";
10
11    printf("&i=%p\n", &i);
12    printf("s =%p\n", s);
13    printf("s2=%p\n", s2);
14    printf("s3=%p\n", s3);
15    s3[0] = 'B';
16    printf("Here!s3[0]=%c\n", s3[0]);
17
18    return 0;
19 }
```

```
&i=0xbff03d64
s =0xfdf7c
s2=0xfdf7c
s3=0xbff03d50
Here!s3[0]=B
[Finished in 0.3s]
```

修改了字符串！




编译器插入代码，执行一个拷贝操作，把之前只读的hello word拷贝到这里

## 指针还是数组?

- char *str = "Hello";
- char word[] = "Hello";

- 数组：这个字符串在这里
  - 作为本地变量空间自动被回收
- 指针：这个字符串不知道在哪里
  - 处理参数
  - 动态分配空间

- 如果要构造一个字符串—>数组
- 如果要处理一个字符串—>指针

## char*是字符串?

- 字符串可以表达为char*的形式
- char*不一定是字符串
  - 本意是指向字符的指针，可能指向的是字符的数组（就像int*一样）
  - 只有它所指的字符数组有结尾的0，才能说它所指的是字符串

## 字符串赋值?

- char *t = "title";

- char *t = “title”;
- char *s;
- s = t;
- 并没有产生新的字符串，只是让指针s指向了t所指的字符串，对s的任何操作就是对t做的


# 字符串输入输出

- char string[8];
- scanf(“%s”, string);
- printf(“%s”, string);

- scanf读入一个单词（到空格、tab或回车为止）
- scanf是不安全的，因为不知道要读入的内容的长度

```
#include <stdio.h>

int main(void)
{
    char word[8];
    scanf("%s", word);
    printf("%s##\n", word);

    return 0;
}
```

```
MOOC:cc\ $./a.out
Hello world!
Hello##
```

```
int main(void)
{
    char word[8];
    char word2[8];
    scanf("%s", word);
    scanf("%s", word2);
    printf("%s##%s##\n", word, word2);

    return 0;
}
```

```
MOOC:cc\ $./a.out
Hello world!
Hello##world!##
MOOC:cc\ $./a.out
Hello
world!
Hello##world!##
```

```
MOOC:cc\ $./a.out
12345678
12345678
##12345678##
```

运气好没报错

# 安全的输入

- char string[8];
- scanf(“%7s”, string);

```
MOOC:cc\ $./a.out
123
12345678
123##1234567##
```

- 在%和s之间的数字表示最多允许读入的字符的数量，这个数字应该比数组的大小小一
  - 下一次scanf从哪里开始?

```
MOOC:cc\ $./a.out
12345678
1234567##8##
```

定义了一个指针变量！！没被初始化就使用了！！指的内存可以被输入，运行可以，如果指的内存不好，那o(╥﹏╥)o

# 常见错误

- char *string;
- scanf(“%s”, string);

- 以为char*是字符串类型，定义了一个字符串类型的变量string就可以直接使用了
  - 由于没有对string初始化为0，所以不一定每次运行都出错

# 空字符串

- char buffer[100]=””;
  - 这是一个空的字符串，buffer[0] == ‘\0’
- char buffer[] = “”;
  - 这个数组的长度只有1!

Buffer[0]就是\0，放不下任何字符串！！

```
a[0] --> char [10]
char a[][10] = {
    "Hello",
    "World"
};
```

后面每个字符串小于10，大于10编译不通过

# 字符串数组

- char **a
  - a是一个指针，指向另一个指针，那个指针指向一个字符（串）
- char a[][]
  - a是一个二维数组，第二个维度的大小不知道，不能编译
- char a[][10]
  - a是一个二维数组，a[x]是一个char[10]
- char *a[]
  - a是一个一维数组，a[x]是一个char*


# 程序参数

一个整数，一个字符串数组
整数反应字符串数组大小

- int main(int argc, char const *argv[])
- argv[0]是命令本身
  - 当使用Unix的符号链接时，反映符号链接的名字

```
int main(int argc, char const *argv[])
{
    int i;
    for ( i=0; i<argc; i++ ) {
        printf("%d:%s\n", i, argv[i]);
    }

    return 0;
}
```

```
MOOC:cc\ $./a.out
0:./a.out
MOOC:cc\ $./a.out 123
0:./a.out
1:123
MOOC:cc\ $./a.out 123 asd asd asdasd
0:./a.out
1:123
2:asd
3:asd
4:asdasd
MOOC:cc\ $
```

# putchar

- int putchar(int c);
- 向标准输出写一个字符
- 返回写了几个字符，EOF（-1）表示写失败

一个宏

# getchar

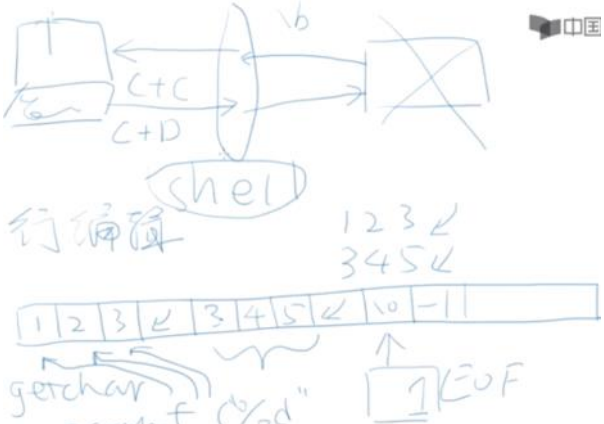
- int getchar(void);
- 从标准输入读入一个字符
- 返回类型是int是为了返回EOF（-1）
  - Windows—>Ctrl-Z
  - Unix—>Ctrl-D

```
int ch;

while ( (ch = getchar()) != EOF ) {
    putchar(ch);
}

printf("EOF\n");

return 0;
```

```
MOOC:cc\ $gcc getput.c
MOOC:cc\ $./a.out
1234546
1234546
872361823619823
872361823619823
kasdhaskdhaksjd
kasdhaskdhaksjd
EOF
EOF
-1
-1
^C
MOOC:cc\ $./a.out
uwayehkashd
uwayehkashd
EOF
MOOC:cc\ $
```

Ctrl c 程序强行终止
Ctrl d 输入结束
win上面是ctrl z 结束

我们回车之后getchar才给了回答




shell完成行编辑工作。没按回车的时候，内容在shell中，按下后，放进很大的缓冲区
先输入123回车——123回车\0，还没有读，再345回车——123回车345回车\0
接下来是程序的任务了
遇到结束标志，继续等我们输入
输入ctrl d shell放入-1？？（-1为EOF)不同编译器不一样。Ctrl c，shell把程序直接关闭


遇到结束标志，继续等我们输入

输入ctrl d shell放入-1？？（-1为EOF)不同编译器不一样。Ctrl c，shell把程序直接关闭

# strlen

strlen()的返回值是无符号长整型

不修改传进去的数组

- size_t strlen(const char *s);
- 返回s的字符串长度（不包括结尾的0）

```c
int main(int argc, char const *argv[])
{
    char line[] = "Hello";
    printf("strlen=%lu\n", strlen(line));
    printf("sizeof=%lu\n", sizeof(line));

    return 0;
}
```

```
strlen=5
sizeof=6
[Finished in 0.1s]
```

```c
size_t mylen(const char* s)
{
    int cnt = 0;
    int idx = 0;
    while (s[idx] != '\0' ) {
        idx++;
        cnt++;
    }
    return cnt;
}

int main(int argc, char const *argv[])
{
    char line[] = "Hello";
    printf("strlen=%lu\n", mylen(line));
    printf("sizeof=%lu\n", sizeof(line));

    return 0;
}
```

```
strlen=5
sizeof=6
[Finished in 0.1s]
```

# strcmp

- int strcmp(const char *s1, const char *s2);
- 比较两个字符串，返回：
  - 0:s1==s2
  - >0:s1>s2
  - <0:s1<s2

```c
int main(int argc, char const *argv[])
{
    char s1[] = "abc";
    char s2[] = "abc";
    printf("%d\n", s1==s2);
    printf("%d\n", strcmp(s1,s2));

    return 0;
}
```

```
/Users/wengkai/cc/test.c:8:19: warning: array
        printf("%d\n", s1==s2);
                       ^
1 warning generated.
0
0
[Finished in 0.1s]
```

```c
int main(int argc, char const *argv[])
{
    char s1[] = "abc";
    char s2[] = "Abc";
    printf("%d\n", strcmp(s1,s2));
    printf("%d\n", 'a'-'A');

    return 0;
}
```

```
32
32
```

数组比较一定是false

因为表达的是它们是否是相同的地址

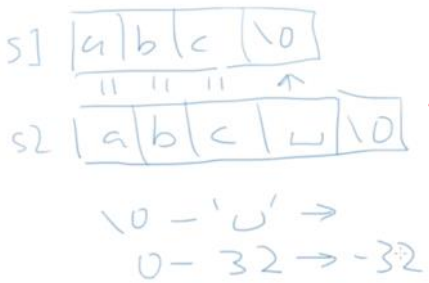
abc<bbc

不相等的时候给出不相等字符的差值

```c
int main(int argc, char const *argv[])
{
    char s1[] = "abc";
    char s2[] = "abc ";
    printf("%d\n", strcmp(s1,s2));
    printf("%d\n", 'a'-'A');

    return 0;
}
```

有了空格

```
-32
32
[Finished in 0.2s]
```




```c
int mycmp(const char* s1, const char* s2)
{
    int idx = 0;
    while ( 1 ) {
        if ( s1[idx] != s2[idx] ) {
            break;
        } else if ( s1[idx] == '\0' ) {
            break;
        }
        idx ++;
    }
    return s1[idx] - s2[idx];
}

int main(int argc, char const *argv[])
{
    char s1[] = "abc";
    char s2[] = "abc";
    printf("%d\n", mycmp(s1,s2));
    printf("%d\n", 'a'-'A');

    return 0;
}
```

```c
int mycmp(const char* s1, const char* s2)
{
    // int idx = 0;
    // while ( s1[idx] == s2[idx] && s1[idx]!= '\0' ) {
    //  idx ++;
    // }
    while ( *s1 == *s2 && *s1 != '\0' ) {
```

```c
int mycmp(const char* s1, const char* s2)
{
    int idx = 0;
    while ( s1[idx] == s2[idx] && s1[idx]!= '\0' ) {
        // if ( s1[idx] != s2[idx] ) {
        //  break;
        // } else if ( s1[idx] == '\0' ) {
        //  break;
        // }
        idx ++;
    }
    return s1[idx] - s2[idx];
}
```

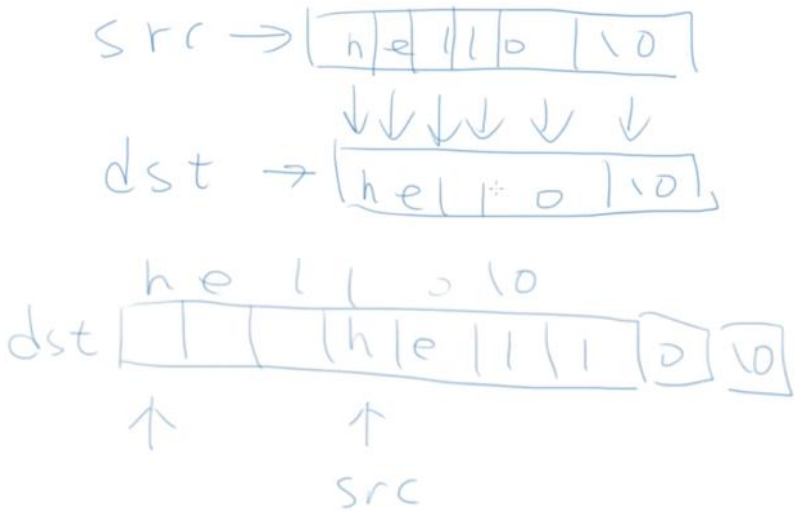
```
// int idx = 0;
// while ( s1[idx] == s2[idx] && s1[idx]!= '\0' ) {
//   idx ++;
// }
while ( *s1 == *s2 && *s1 != '\0' ) {
    s1++;
    s2++;
}
return *s1 - *s2;
}
```

```
    // }
    idx ++;
}
return s1[idx] - s2[idx];
}
```

一种方法把字符串当数组使用，一种用指针

# strcpy

- char * strcpy(char *restrict dst, const char *restrict src);
- 把src的字符串拷贝到dst
  - restrict表明src和dst不重叠（C99）
- 返回dst
  - 为了能链起代码来




就是把内存内容往前移

不能做这种事情！！

不能用strcpy

# 复制一个字符串

套路！！记得要加一！！

```
char *dst = (char*)malloc(strlen(src)+1);
strcpy(dst, src);
```

```
char* mycpy(char* dst, const char* src)
{
    int idx = 0;
    while (src[idx] != '\0' ) {
        dst[idx] = src[idx];
        idx++;
    }
    dst[idx] = '\0';
    return dst;
}
```

```
char* mycpy(char* dst, const char* src)
{
    int idx = 0;
    while (src[idx] ) {
        dst[idx] = src[idx];
        idx++;
    }
    dst[idx] = '\0';
    return dst;
}
```

```
char* mycpy(char* dst, const char* src)
{
    // int idx = 0;
    // while (src[idx] ) {
    //   dst[idx] = src[idx];
    //   idx++;
    // }
    // dst[idx] = '\0';
    char* ret = dst;
    while (*src != '\0' ) {
        *dst = *src;
        dst++;
        src++;
    }
    *dst = '\0';

    return rest;
}
```

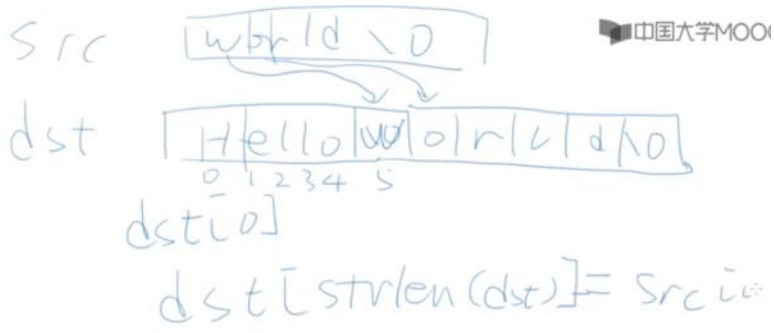
```
char* mycpy(char* dst, const char* src)
{
    // int idx = 0;
    // while (src[idx] ) {
    //   dst[idx] = src[idx];
    //   idx++;
    // }
    // dst[idx] = '\0';
    char* ret = dst;
    while ( *src ) {
        *dst++ = *src++;
    }
    *dst = '\0';

    return rest;
}
```

```
char* mycpy(char* dst, const char* src)
{
    // int idx = 0;
    // while (src[idx] ) {
    //   dst[idx] = src[idx];
    //   idx++;
    // }
    // dst[idx] = '\0';
    char* ret = dst;
    while ( *dst++ = *src++ )
        ;
    *dst = '\0';

    return rest;
}
```

# strcat

- char * strcat(char *restrict s1, const char *restrict s2);
- 把s2拷贝到s1的后面，接成一个长的字符串
- 返回s1
- s1必须具有足够的空间

也是一种拷贝




# 安全问题

- strcpy和strcat都可能出现安全问题
  - 如果目的地没有足够的空间?

# 安全版本

- char * strncpy(char *restrict dst, const char *restrict src, size_t n);
- char * strncat(char *restrict s1, const char *restrict s2, size_t n);
- int strncmp(const char *s1, const char *s2, size_t n);

# 字符串中找字符

- char * strchr(const char *s, int c);
- char * strrchr(const char *s, int c);
- 返回NULL表示没有找到

- 如何寻找第2个?

可能是看前三个字符是什么

# 字符串中找字符串

- char * strstr(const char *s1, const char *s2);
- char * strcasestr(const char *s1, const char *s2);

```
4  int main(int argc, char const *argv[])
5  {
6      char s[] = "hello";
7      char *p = strchr(s, 'l');
8      p = strchr(p+1, 'l');
9      printf("%s\n", p);
10
11     return 0;
12 }
13
```

```
llo
```

```
4  int main(int argc, char const *argv[])
5  {
6      char s[] = "hello";
7      char *p = strchr(s, 'l');
8      p = strchr(p+1, 'l');
9      printf("%s\n", p);
10
11     return 0;
12 }
13
```

```
lo
[Finished in 0.2s]
```

```
5  int main(int argc, char const *argv[])
6  {
7      char s[] = "hello";
8      char *p = strchr(s, 'l');
9      char *t = (char*)malloc(strlen(p)+1);
10     strcpy(t, p);
11     printf("%s\n", t);
12     free(t);
13
14     return 0;
15 }
16
```

```
llo
[Finished in 0.2s]
```

```c
int main(int argc, char const *argv[])
{
    char s[] = "hello";
    char *p = strchr(s, 'l');
    char c = *p;
    *p = '\0';
    char *t = (char*)malloc(strlen(s)+1);
    strcpy(t, s);
    printf("%s\n", t);
    free(t);

    return 0;
}
```

```
he
[Finished in 0.2s]
```