第13章

字符串（Strings）

引言

- 字符串：
  - 字符数组
  - 以特殊字符——空字符（'\0'）结尾
- 字符串常量
  - 字面量string constants,或者literals
- 字符串变量
  - string variables
- C库提供了用于操作字符串的一系列函数

字符串字面量（String Literals）

- 字符串变量：`char name[20];`
- 字符串常量：双引号括起来的字符序列

```
"When you come to a fork in the road, take it."
```

- 可包含转义序列

```
"Candy\nIs dandy\nBut liquor\nIs quicker.\n  --Ogden Nash\n"
```

- 结果：

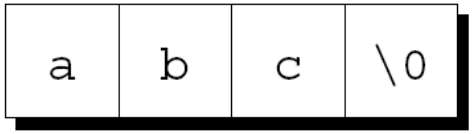
```
Candy
Is dandy
But liquor
Is quicker.
  --Ogden Nash
```

字符串字面量的存储

- 编译器遇到一个长度为n的字符串字面量时，分配n+1个字节内存空间
- 空字符标志字符串结束
  - 所有比特全为0的字节
  - 用转义序列 '\0'表示

字符串字面量的存储

- 字符串字面量 “abc” 存放如图：



- 以数组方式存储，
  - (char [4]){'a','b','c','\0'}
  - 编译器看作char * ：指针（数组名本质）
- printf、scanf第一个参数

字符串字面量的操作

- 可在任何使用 char*指针的地方使用字符串字面量

```
char *p;
p = "abc";
```

- p指向字符串的第一个字符

实验问题——变长数组

- 变长数组：指用整型变量或表达式声明或定义的数组

```
int n;
scanf("%d",&n);
int fib[n];
```

  - n在声明数组前确定

- 变长数组在其生存期内的长度固定不变。

```
int i=0;
char a[i+1];
While((ch=getchar())!='\n'){
        a[i++]=ch;
}//将语句存入变长数组
```

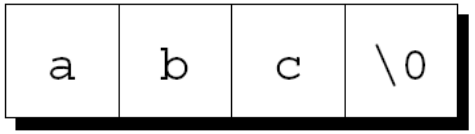
- 编译器对C99支持问题

温故——多维数组与指针

- 嵌套（降维）：多维（N维）数组可视作一维数组，数组名代表该一维数组地址（首元素（N-1维）地址），eg.
  - char name[M][L]，视作name[M]
  - name：首行地址，指向一行的指针。
  - 类型：char (*)[L]的指针，指向长度L的char数组
- 行指针：
  - char (*p)[L]=name或&name[0];
  - p++：移动一行（实质也是元素指针，元素为行）

温故而知新——字符串

- 字符串常量—（字面量）"字符序列"
  - "how are you, guys"
  - "abc"
  - 以字符数组方式存放：



  - 以空字符'\0'结尾
- 字符常变量—以空字符结尾的一维字符数组

字符串字面量的操作

- 指针可添加下标

```
char ch;
ch = "abc"[1];//ch = b.
```

- 把0到15转换为等价16进制数字的函数：

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F

```
const char a[16]={'0','1',……'F'};
给i，求a[i]
char digit_to_hex_char(int digit)
{
   return "0123456789ABCDEF"[digit];
}
```

字符串字面量的操作

- 字符串字面量是常量
- 对字符串字面量的操作会导致未定义的行为：

```
char *p = "abc";
*p = 'd';    /*** WRONG ***/
```

- 可修改p，不可修改*p

字符串字面量 vs 字符常量

- 单字符字符串字面量与字符常量不同
  - “a” 是以指针表示
  - ‘a’ 是以整数表示
  - if(ch==“M”)……
- 对printf的合法调用为

```
printf("\n");
```

- 非法的调用：

```
printf('\n');    /*** WRONG ***/
```

字符串变量

- 以空字符结尾的一维字符数组
- 一维字符数组不一定是字符串
  - 有无空字符尾巴
- 尾巴意义：
  - 只需给出字符串头部（地址），不需长度
  - 求字符串长度，通过搜索空字符

字符串变量

- 声明字符串变量
  - 字符数组：char array[STR_LEN];
  - 确保给空字符留出空间
    - 常见方法：采用+1方式定义字符数组
    - char name[M+1];如

```
#define STR_LEN 80
  …
char str[STR_LEN+1];
```

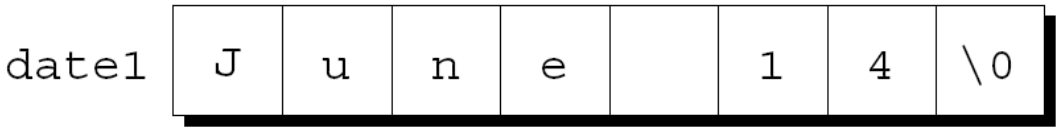
- 字符串长度VS 数组长度：
  - 字符数组：容器，STR_LEN是最大长度
  - 实际长度：取决于结束符的位置，0到STR_LEN

初始化字符串变量

- 声明同时初始化：

```
char date1[8] = "June 14";
```

- 编译器自动增加一个空字符：

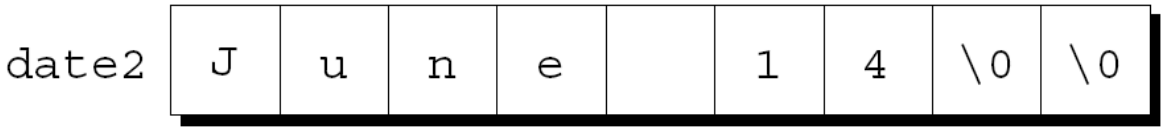


- “June 14” 不是字符串字面量
- 而是初始化式{‘J’, ‘u’, ‘n’, ‘e’, ‘ ’, ‘1’, ‘4’, ‘\0’}的缩写.

初始化字符串变量

- 初始化式太短，编译器自动补零（空字符）：
  - 补零(char)'\0'=(int)0

char date2[9] = "June 14";表示为:



字符串：字符数组 vs 字符指针

- 声明date为一个数组：

```
char date[] = "June 14";
```

初始化式{'J'……}

- 声明date为一个指针

```
char *date = "June 14";
```

字面量

- 均可作为一个字符串，存在区别：
  - 数组版：内容（字符）可修改；date：数组名（地址）不变；
  - 指针版：内容（字符串字面量）不可修改，date指向可变。

字符数组 vs 字符指针

- 声明：

```
char *p;//未初始化
p[0] = 'a';//wrong
```

- 把p作为字符串前，必须指向一个字符数组.
  - 做法0：指向字符串字面量

```
*p = "June 14";
```

  - 做法1：使p指向字符串变量：

```
char str[STR_LEN+1], *p = str;
```

  - 做法2：使p指向动态分配的字符串

```
p = (type *)malloc(STR_LEN+1);
void *malloc(size_t size)//分配size字节，返回首地址（通用地址）
```

读写字符串

- 写（输出）：printf + %s或者puts.
- 读字符串：scanf + %s或者gets

用printf和puts写字符串

- Printf函数用 %s 转换说明符来写一个字符串：

```
char str[] = "Are we having fun yet?";
printf("%s\n", str);
```

- 输出为：

```
Are we having fun yet?
```

- printf逐字符写字符串，直至遇到空字符.
  - 有尾巴，无需参数传递数组或字符串长度

用printf和puts写字符串

- puts函数用于写字符串：

```
puts(str);
```

- 与printf不同，写完字符串后，puts函数会自动换行

用scanf 和 gets读字符串

- Scanf函数用%s 转换说明符读字符到一个字符数组：

```
scanf("%s", str);
```

- str是字符串（数组），不必放置&.
- scanf自动加一个空字符在字符串的后面

用scanf 和 gets读字符串

- scanf遇换行符、空白和tab符停止读取
- 读取整行输入用gets函数
  - gets(str);
- 特点：
  - 读取输入不会跳过开始的空白.
  - 读到换行符（回车）才停止读入.
  - 不存储回车，自动加尾巴（空字符），等价于：

```
i=0;
while((ch= getchar())!='\n')
  {str[i++] = ch;}
  str[i]='\0';
```

用scanf 和 gets读字符串

- 程序片段：

```
char sentence[SENT_LEN+1];
printf("Enter a sentence:\n");
scanf("%s", sentence);
```

- 输入：To C, or not to C:
  that is the question.

  sentence = “To”.

- gets(sentence);

  sentence = “To C, or not to C:
  that is the question.”

用scanf 和 gets读字符串

- scanf、gets不检查目标存储空间是否存满
- 可能造成数组越界，导致未定义的行为
- 解决方法：
  - scanf通过转换说明符%ns来在一定程度上避免上述问题.
  - n指明能够存放的最多字符数.
  - gets本身不安全；fgets是一个更好的替代.
    - #include <stdio.h>
    - char *fgets(char *str, int num, FILE *stream); //stdin
  -

逐字符读入字符串

- scanf,gets不够灵活且有风险
- 程序员常编写自己的输入函数，需考虑的问题：
- 是否跳过开头空白？
- 怎么停止：
  - 换行符、空白字符或者其它字符？
  - 该字符是存放到字符串还是丢弃？
- 是否检测长度，超长怎么处理：
  - 丢弃额外的字符或者留给下一次输入操作？

逐字符读入字符串

- 假如我们需要函数：
  - (1)不跳过开始的空白字符
  - (2)读到第一个换行符则停止（不存入字符串）
  - (3)最多读n个字符，舍弃额外的字符
- 该函数的一种原型为：

```
int read_line(char str[], int n);
```

逐字符读入字符串

- 循环调用 getchar来实现：

```
int read_line(char str[], int n)
{
  int ch, i = 0;
  while ((ch = getchar()) != '\n')
    if (i < n)
      str[i++] = ch;
  str[i] = '\0';
  return i;
}
```

- getchar返回int：ASCII码
  - ch可以是int也可是char

访问字符串中的字符

- 字符串以数组方式存储，访问方式：
  - 下标，指针
- 函数参数声明
  - 字符串数组或指针，等价
- 计算字符串中空白字符个数的函数：

```
int count_spaces(const char
s[])
int count_spaces(const char
*s)
```

访问字符串中的字符

- 数组下标版：

```
int count_spaces(const char s[]){
  int count = 0, i;
  for (i = 0; s[i] != '\0'; i++)
    if (s[i] == ' ') count++;
  return count;
}
```

- 指针运算版本：

```
int count_spaces(const char *s){
  int count = 0;
  for (; *s != '\0'; s++)
    if (*s == ' ') count++;
  return count;
}
```

使用C字符串库

- 字符串作为数组处理（本质：字符数组）
- 不能直接拷贝（赋值）和比较

```
char str1[10], str2[10];
str1 = "abc";
str2 = str1;
if (str1 == str2) …//error
```

- 赋值：

```
for(i=0;i<10;i++) str2[i]=str1[i];
```

- 比较：

```
for(bool eq=true,int i=0;i<10;i++)
   if(str1[i]!=str2[i]){
         eq = false;
         break;
}
```

使用C字符串库

- C函数库提供了丰富的函数处理字符串

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

- 函数以字符串为形式参数：
  - 类型char *：
- 实际参数：
  - 字符数组
  - char指针
  - 字符串字面量
- 在后面的例子中，假设

```
char str1[],str2[];或
char *s1, *s2, *str;
```

strcpy （string copy）函数

- strcpy 函数原型：

```
char *strcpy(char *s1, const
char *s2);
```

- 拷贝字符串s2到字符串s1，返回s1

```
strcpy(str2, "abcd");
/* str2 now contains "abcd" */
strcpy(str1, str2);
/* str1 now contains "abcd" */
```

- str2长于str1，未定义行为

strcpy （string copy）函数

- strncpy: //量体裁衣
  - 较慢但安全.
  - 原型：c-free C\C++库函数参考

```
strncpy(str1,str2,sizeof(str1));
```

- 隐忧：str1结束符问题，更安全的方式

```
strncpy(str1, str2, sizeof(str1) - 1);
str1[sizeof(str1)-1] = '\0';
```

- 保证str1总是以空字符结尾的.

strlen (String Length)函数

- 函数原型：

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

- 返回字符串s的长度，不包括空字符.

```
int len;
len = strlen("abc"); /*3*/
len = strlen("");    /*0*/
strcpy(str1, "abc");
```

strl：数组，容器，sizeof求容积

装字符串（溶液），strlen求溶液的体积

strcat (String Concatenation) 函数

- 函数原型：

```
char *strcat(char *s1, const char *s2);
```

- 追加s2的内容到s1末尾，返回s1：

```
strcpy(str1, "abc");
strcat(str1, "def");
/*str1 now contains "abcdef" */
strcpy(str1, "abc");
strcpy(str2, "def");
strcat(str1, str2);
/*str1 now contains "abcdef" */
```

strcat (String Concatenation) 函数

- **str1**数组应足够长，以容纳**str1+str2**的内容，否则导致未定义的行为.
- **strncat**
  - 第三个参数(**int**)：限制要拷贝字符数
  - 安全但较慢版本.
- 调用：

```
strncat(str1, str2,
sizeof(str1) - strlen(str1) -
1);
```

strcmp (String Comparison) 函数

- 函数原型：

```
int strcmp(const char *s1, const char *s2);
```

- 返回：
  - 小于0：s1<s2
  - 等于0：s1==s2
  - 大于0：s1>s2
- 测试str1是否小于str2：

```
if (strcmp(str1, str2) < 0)
```

- 测试 str1 是否小于或者等于 str2：

```
if (strcmp(str1, str2) <= 0)
```

strcmp (String Comparison) 函数

- 比较实质：从头逐字符比较ASCII码
  - uestc vs ustc
- s1小于s2：
  - s1、s2前i个字符相同，但s1第(i+1)个字符小于s2第(i+1)个字符
    - “abc”小于”bcd”,“abd”小于“abe”;
  - s1的所有字符都匹配s2，但是s1比s2短（空字符与s2其余字符比）.
    - “abc”小于”abcd”

温故而知新——字符串声明

- 声明为数组：

```
char date[] = "June 14";
```

初始化式{'J'……}

- 声明为指针 ：

```
char *date = "June 14";
```

字面量

温故而知新——字符串指针

- char *p;//必须指向一个字符串
- 做法0：指向字符串字面量
  - *p = “June 14”;
- 做法1：指向字符串变量：

  char str[STR_LEN+1], *p=str;
- 做法2：指向动态分配空间：

  p = malloc(STR_LEN+1);

  处理链表，解决数组大小固定的问题

温故而知新——字符串操作

- 循环逐个访问字符串字符
- 数组版：

```
char str[LEN+1];
for(i=0;str[i]!='\0';i++){
   str[i]……}
```

- 指针版：

```
char *p=str;
for(;*p != '\0';p++){
   *p……}
```

温故而知新——字符串处理函数

- 串拷贝
  - `char *strcpy(char *s1, const char *s2);`
  - `strncpy(str1, str2, sizeof(str1) - 1);`
- 串长度
  - `size_t strlen(const char *s);`
- 串拼接
  - `char *strcat(char *s1, const char *s2);`
  - `strncat(str1, str2, sizeof(str1) - strlen(str1) - 1);`
- 串比较
  - `int strcmp(const char *s1, const char *s2);`

Program: Printing a One-Month Reminder List：日程表

```
Enter day and reminder: 24 Susan's birthday
Enter day and reminder: 5 6:00 - Dinner with Marge and Russ
Enter day and reminder: 26 Movie - "Chinatown"
Enter day and reminder: 7 10:30 - Dental appointment
Enter day and reminder: 12 Movie - "Dazed and Confused"
Enter day and reminder: 5 Saturday class
Enter day and reminder: 12 Saturday class
Enter day and reminder: 0
```

无序输入

```
Day Reminder
  5 Saturday class
  5 6:00 - Dinner with Marge and Russ
  7 10:30 - Dental appointment
 12 Saturday class
 12 Movie - "Dazed and Confused"
 24 Susan's birthday
 26 Movie - "Chinatown"
```

有序生成

Program: Printing a One-Month Reminder List

- **Overall strategy:**
- 造表：二维字符数组存储日程表
- 录入日程：分别读日期和提醒，组合成日程
  - `scanf("%2d", &day);`
  - `read_line();`
  - `sprintf(day_str, "%2d", day);//格式化输出到串`
- 按序排列：按日期顺序
  - 排序：冒泡、快排
  - 插队：定位，让位（晚于当前日期的日程后移），插入
- 显示

remind.c

```
/* Prints a one-month reminder list */
#include <stdio.h>
#include <string.h>
#define MAX_REMIND 50 /*maximum number of reminders */
#define MSG_LEN 60    /*max length of reminder message*/

int read_line(char str[], int n);
int main(void)
{
  char reminders[MAX_REMIND][MSG_LEN+3];
  char day_str[3], msg_str[MSG_LEN+1];
  int day, i, j, num_remind = 0;

  for (;;) {//循环录入
    if (num_remind == MAX_REMIND) {
      printf("-- No space left --\n");
      break;
    }
    printf("Enter day and reminder: ");
```

日程插入过程

5 Saturday class
5 6:00 - Dinner with Marge and Russ
7 10:30 - Dental appointment
24 Susan's birthday
26 Movie - "Chinatown"

插队，请下移

12	Saturday class

数字→字符串：先定位

sprintf(char *buffer, const char *format,

插队：晚于当前日程依次下移

12 Saturday class

串拼接、插入

```
    scanf("%2d", &day);
    if (day == 0)
      break;
    sprintf(day_str, "%2d", day);//保两位日期
    read_line(msg_str, MSG_LEN);

    for (i = 0; i < num_remind; i++)//定位
      if (strcmp(day_str, reminders[i]) < 0)
        break;//应插入第i行
    for (j = num_remind; j > i; j--)//移动
      strcpy(reminders[j], reminders[j-1]);

    strcpy(reminders[i], day_str);
    strcat(reminders[i], msg_str);

    num_remind++;
  }

  printf("\nDay Reminder\n");
  for (i = 0; i < num_remind; i++)
    printf(" %s\n", reminders[i]);

  return 0;
}
```

```
int read_line(char str[], int n)
{
  int ch, i = 0;

  while (((ch = getchar()) != '\n') && (i < n))
        str[i++] = ch;

  str[i] = '\0';
  return i;
}
```

字符串惯用法-库函数实现

- 处理字符串的函数是特别丰富的惯用法资源.
- 下面将会探索其中两种最著名的惯用法，并利用它们编写`strlen`函数和`strcat`函数.

搜索字符串的结尾strlen

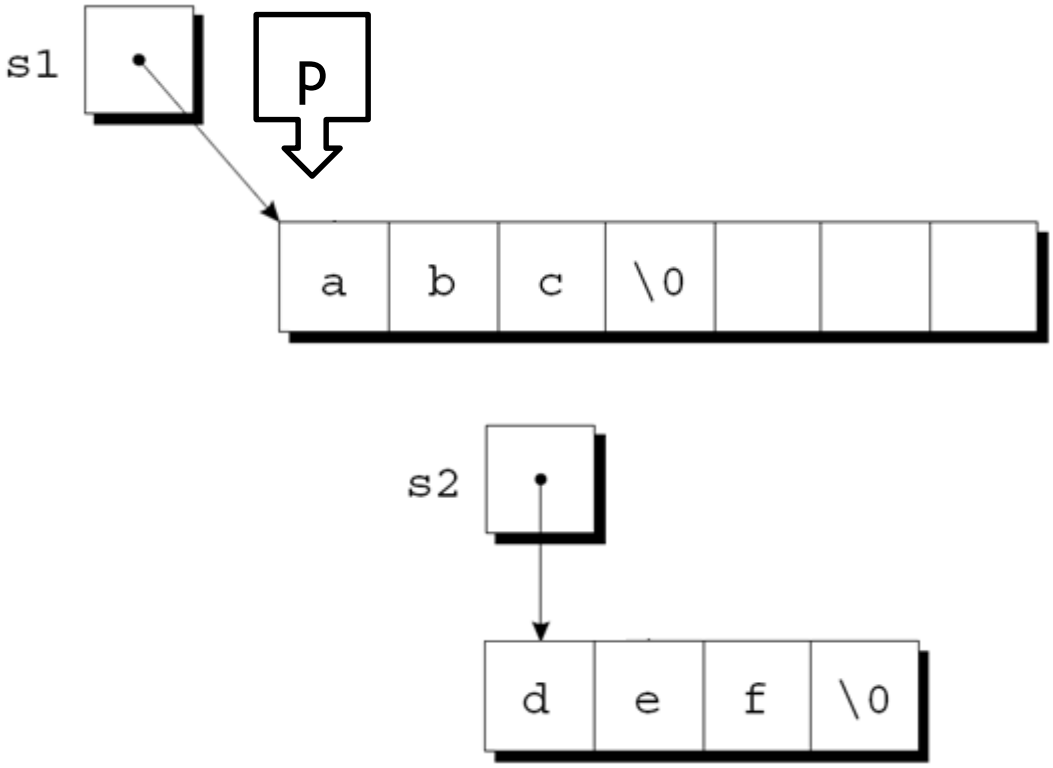
- 累计字符串长度：

```
size_t strlen(const char *s)
{
  size_t n;

  for (n = 0; *s != '\0'; s++)
    n++;
  return n;
}
```

- size_t：标准C库中定义，
  - 应为unsigned int，
  - 在64位系统中为 long unsigned int。

串联字符串strcat(s1,s2)

- 用p指向s1并搜索其结尾空字符
- 从s2逐个复制字符到p指向的位置



串联字符串strcat(s1,s2)

```
char *strcat(char *s1, const char *s2)
{
   char *p = s1;
   while (*p != '\0')p++;
   while (*s2 != '\0'){
      *p = *s2;
      p++;
      s2++;
   }
   *p = '\0';
   return s1;
}
```

字符串数组

- 存储字符串数组最佳方式：二维字符数组，每行一个字符串：

```
char planets[][8] =
{"Mercury", "Venus", "Earth",
 "Mars", "Jupiter", "Saturn",
 "Uranus", "Neptune","Pluto"};
```

- 可以忽略数组的行数，但是必须指明数组的列数.

字符串数组

- 字符串长短不一
- **planets**数组存在未用空白（额外的空字符)：

	0	1	2	3	4	5	6	7
0	M	e	r	c	u	r	y	\0
1	V	e	n	u	s	\0	\0	\0
2	E	a	r	t	h	\0	\0	\0
3	M	a	r	s	\0	\0	\0	\0
4	J	u	p	i	t	e	r	\0
5	S	a	t	u	r	n	\0	\0
6	U	r	a	n	u	s	\0	\0
7	N	e	p	t	u	n	e	\0
8	P	l	u	t	o	\0	\0	\0

数组格式字符串，指针格式？？

字符串数组

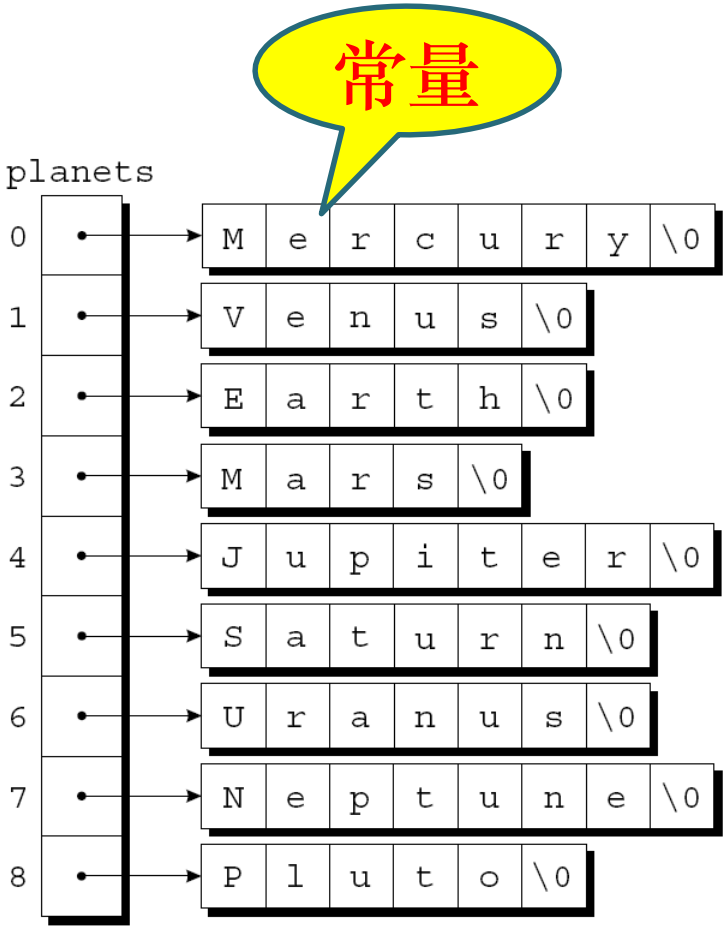
- 为节省空间，构造参差不齐的数组(ragged array)——指针数组：

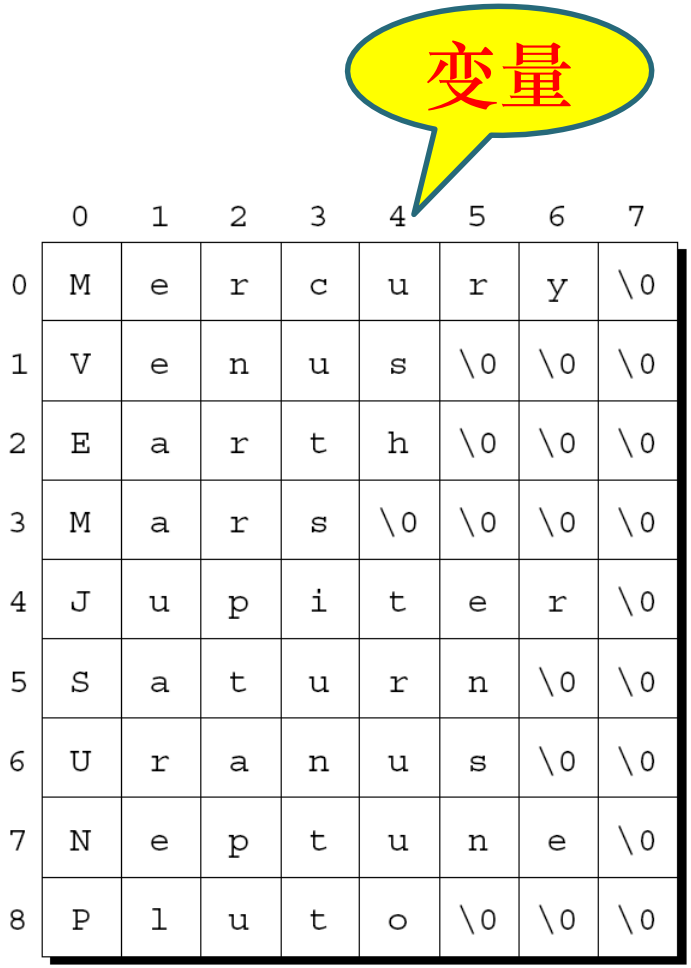
```
char *planets[] =
{"Mercury", "Venus", "Earth",
 "Mars", "Jupiter", "Saturn",
 "Uranus", "Neptune","Pluto"};
```

- {……}：初始化式
- “……”：字符串字面量
- planets[0]=“Mercury”
- char (*planets)[]

字符串数组

- **planets的存储：**





字符串数组

- planets[i]第i个指针
  - 指向（访问）第i个行星名字（字符串字面量）
- planets[i][j]
  - planets[i]指针当作数组名
  - 访问第i个行星的第j个字符
- 搜索并显示以字母M开头的字符串为：

```
for (i = 0; i < 9; i++)
  if (planets[i][0] == 'M')
    printf("%s begins with
M\n",planets[i]);
```

命令行参数

- 控制台程序常以命令方式执行，运行程序时，常需提供一些信息（程序参数）
- eg.文件名或者用于修改程序行为的选项

UNIX、Linux下的ls：

```
C:\WINDOWS\system32>ping www.uestc.edu.cn

正在 Ping www.uestc.edu.cn [202.112.14.178] 具有 32 字节的数据:
来自 202.112.14.178 的回复: 字节=32 时间=84ms TTL=33
来自 202.112.14.178 的回复: 字节=32 时间=89ms TTL=33
来自 202.112.14.178 的回复: 字节=32 时间=86ms TTL=33
来自 202.112.14.178 的回复: 字节=32 时间=84ms TTL=33

202.112.14.178 的 Ping 统计信息:
    数据包: 已发送 = 4, 已接收 = 4, 丢失 = 0 (0% 丢失),
往返行程的估计时间(以毫秒为单位):
    最短 = 84ms, 最长 = 89ms, 平均 = 85ms
```

ipconfig

命令行参数

- 命令行参数：操作系统传递给程序的数据（参数）
- ——main函数参数：

```
int main(int argc, char *argv[])
{   …   }
```

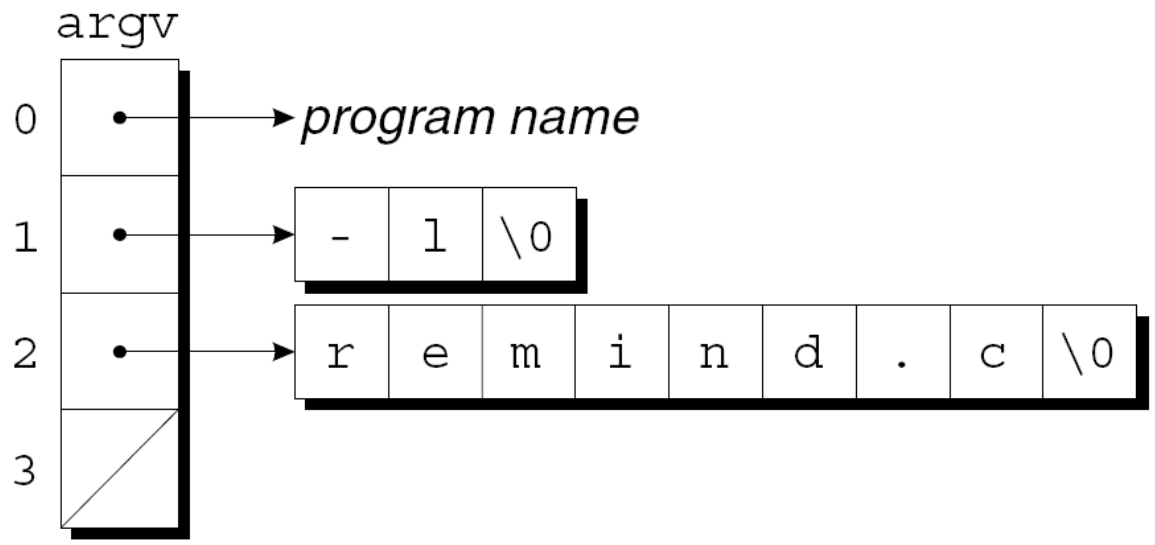
- argc：统计传给main的参数个数
- argv：参数向量，指向命令行参数（字符串方式存储）的指针数组.
  - argv[0]：指向程序名
  - argv[1]至 argv[argc-1]：指向余下的命令行参数.
  - argv[argc]：宏NULL表示空指针（不指向任何东西）

命令行参数

- 如果用户输入的命令行为：

```
ls -l remind.c
```

- 则argc为3，argv[argc+1]为如下表示：



命令行参数

- 典型地，采用循环来顺序检查每个命令行参数.

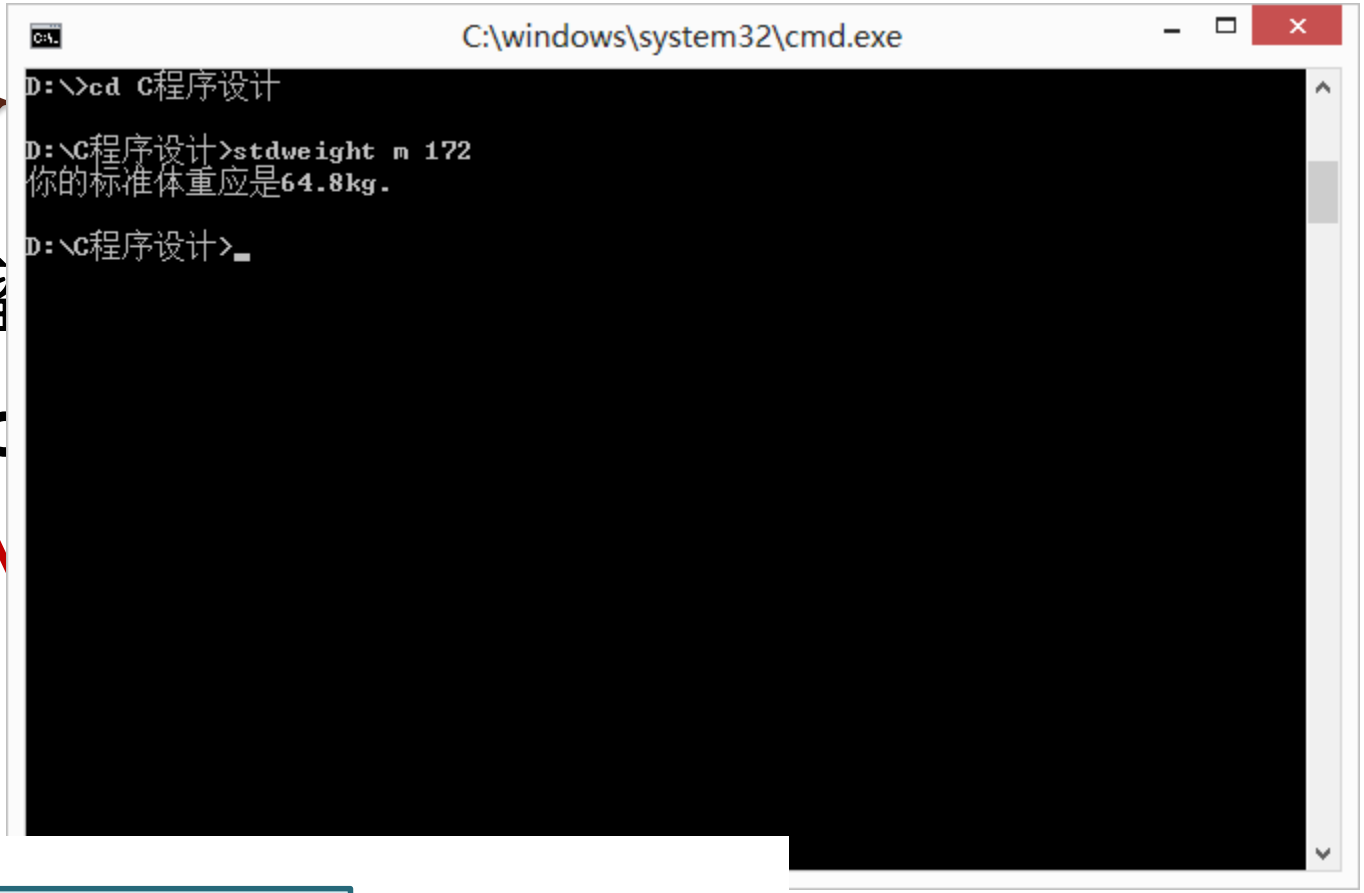
```
int i;

for (i = 1; i < argc; i++)
  printf("%s\n", argv[i]);
```

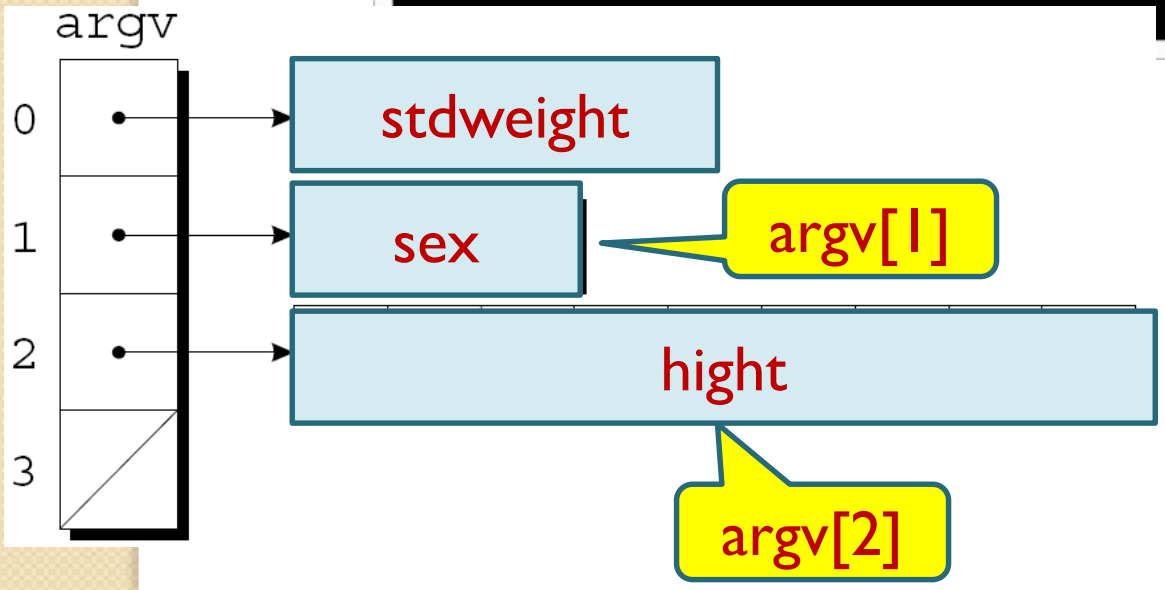
命令行参数示例——标准体重

```
#include <stdio.h>
#define FACTOR 0.9f
int main(void){
     int height;
     float weight, stdwt;
     char sex;
     puts("输入性别，男性用m表示，其它字符表女性.");
     scanf("%c", &sex);
     puts("输入身高(cm).");
     scanf("%d", &height);
     if (sex == 'm')stdwt = (height - 100) * FACTOR;
     else stdwt = (height - 100) * FACTOR -2.5;
     printf("你的标准体重应是%.1fkg\n.", stdwt);
     return 0; }
```

命令

- 期望
- cmd
  D:\


C:\windows\system32\cmd.exe
D:\>cd C程序设计
D:\C程序设计>stdweight m 172
你的标准体重应是64.8kg.
D:\C程序设计>


argv
0
1
2
3
stdweight
sex
hight
argv[1]
argv[2]

命令行参数示例——标准体重

```
#include <stdio.h>
#include <stdlib.h>
#define FACTOR 0.9f
int main(int argc, char *argv[]){
    int height;
    float weight, stdwt;
    char sex;
    sex = argv[1][0];
    height  = atoi(argv[2]);
    if (sex == 'm') stdwt = (height - 100) * FACTOR;
    else stdwt = (height - 100) * FACTOR -2.5;
    printf("你的标准体重应是%.1fkg.\n", stdwt);
    return 0;
}
```

每个参数都是字符串格式