数据结构与程序设计（信息类）

Data Structure and Programming

北航软件学院 林广艳

第 1 章

数据结构基础
（复杂数据程序设计）

目录
CONTENTS

第 1 章

数据结构基础
（复杂数据程序设计）

目录
CONTENTS

字符串（string）

- 字符串是一种常见的数据类型：一组字符排列
  - 单词、语句、段落、文章
- C语言没有提供相应的基本数据类型，需设计一种数据结构来实现该数据类型

串的定义

串是由n≥0个字符组成的有限序列，记为

$S = "a_1 a_2 a_3 \ldots a_{n-1} a_n"$

其中，S表示串名（也称串变量），一对引号括起来的字符序列称为串值，a_i可以是字母、数字或其他允许的字符。n为串的长度，长度为0的串称为空串

串的基本操作（算法）

C函数

	C函数
1.给串变量赋值 ASSIGN(S1,S2)	strcpy(S1,S2)
2. **判断两个串是否相等 EQUAL(S1,S2)**	strcmp(S1,S2)
3.两个字符串连接 CONCAT(S1,S2)	strcat(S1,S2)
4.求字符串的长度 LEN(S)	strlen(S)
5.求子串 SUBSTR(S,i,k)	
6. **求子串在主串中的位置 INDEX(S1,S2)**	strstr(S1,S2)
7.串的替换 REPLACE(S,S1,S2)	
8.串的复制 COPY(S1,S2)	strcpy(S2,S1)
9.串的插入 INSERTS(S1,i,S2)	
10.串的删除 DELETES(S,i,k)	
……	

C语言中的字符串实现

$S = "a_1 a_2 a_3 \ldots a_{n-1} a_n"$

两种实现方式：

1. 字符数组：char s[N];

0	1	2	…	…	n-2	n-1	n		N-1
a_1	a_2	a_3	…	…	a_{n-1}	a_n	\0	…	…

2. 字符指针：char *s; //注意：需要分配存储空间

由于每个字符串的长度各不相同，如何表示字符串的结束？

1. 记录当前字符串的长度（普通数组中常用的方法）

   缺点：需要单独的变量存储长度，管理和使用比较麻烦

2. 在有效字符串的最后添加一个特殊字符：'\0'

   缺点：浪费一个存储空间

C语言中的字符串处理函数

- C语言提供了一系列字符串处理功能函数，所有这些函数只能处理字符串（即一定要有\0结尾）
- <stdio.h>，提供各类字符串输入输出转换函数
  - ◆scanf、printf中提供%s操作字符串
  - ◆puts、gets字符串输入和输出函数
  - ◆sscanf、sprintf提供字符串和普通数据类型的转换
- <string.h>，以str开头，提供各类操作函数
  - ◆字符串长度：strlen
  - ◆字符串拷贝：strcpy、strncpy
  - ◆字符串连接：strcat、strncat
  - ◆字符串比较：strcmp、strncmp
  - ◆字符串查找：strstr、strchr、strrchr

C语言中的字符串处理函数

- 字符串基本构成单位为字符，字符处理函数可以简化字符处理
- <ctype.h>：字符处理函数
- 判断各类字符类型
  - ◆判断字母、数字等：isalpha、islower、isupper、isdigit、isalnum
  - ◆判断特殊字符：isprint、isspace、isxdigit、iscntrl、ispunct、isgraph
- 字符大小写转换
  - ◆将字符c转换为小写字母：tolower
  - ◆将字符c转换为大写字母：toupper

示例：C语言中字符串的定义和赋值

```
//定义一个长度为32的字符数组，可以最大长度为31的字符串
char arr1[32];
//定义一个字符指针，需要指向某个有效的连续存储空间，该空间内容可看做字符串
char *str1;

strcpy(arr1, "Data Structure");
str1 = (char *) malloc( strlen(arr1) +1 ); //根据arr1的大小(14个字符)分配字符串空间
strcpy(str1, arr1);
free(str1);  //malloc分配的空间，必须通过free释放
//定义一个字符数组，初始化数组成员
char arr2[] = "Data Structure";
//定义一个字符指针，指向某个字符串常量
char *ptr2 = "Data Structure";
```

问题1：将数字字符串转换为整数

"123" ➡ 123

- 12*10 + '3'-'0' = 123
- 1*10 + '2'-'0' = 12
- '1'-'0'= 1

```
n = 0;
for(i=0; s[i]为数字字符; i++)
    n = 10*n + s[i] – '0';
```

字符	十进制 ASCII 值	十六进制 ASCII 值	字符	十进制 ASCII 值	十六进制 ASCII 值	字符	十进制 ASCII 值	十六进制 ASCII 值
0	48	30	A	65	41	a	97	61
1	49	31	B	66	42	b	98	62
2	50	32	C	67	43	c	99	63
3	51	33	D	68	44	d	100	64
4	52	34	E	69	45	e	101	65
5	53	35	F	70	46	f	102	66
6	54	36	G	71	47	g	103	67
7	55	37	H	72	48	h	104	68
8	56	38	I	73	49	i	105	69
9	57	39	J	74	4A	j	106	6A

问题1：代码实现

```
int atoi(char s[])
{
  int i, n, sign;
  for (i=0;s[i]==' '||s[i]=='\n'||s[i]=='\t'; i++)
      ; /* skip white space */
  sign = 1;
  if (s[i] = = '+' || s[i] = = '-')
      sign = (s[i++] = = '+') ? 1 : -1;
  for (n = 0; s[i] >= '0' && s[i] <= '9'; i++)
      n = 10 * n + s[i] - '0';
  return (sign * n);
}
```

字符数组

条件运算符：
<表达式1> ? <表达式2> : <表达式3>
先计算表达式1，若其值为非零，则整个表达式结果为表达式2的值，否则就为表达式3的值

读入一个以空白字符分隔的字符串到s中，以' \0' 结束(即由非空字符组成的字符串)

```
#include <stdio.h>
int atoi(char s[]);
int main()
{   char s[20];
    scanf("%s", s);
    printf("%d\n", atoi(s));
    return 0;
}
```

为何字符数组（字符串）作为函数参数传递时，通常不用传递数组长度？

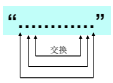
问题2：将整数转换为字符串

```
void itoa(int n, char s[])
{
    int i, sign;
    if ((sign = n) < 0)
        n = -n;
    i = 0;
    do
    {
        s[i++] = n % 10 + '0';
    } while ((n /= 10) > 0);
    if (sign < 0)
        s[i++] = '-';
    s[i] = '\0';
    reverse(s); //将字符串颠倒
}
```

```
#include <stdio.h>
void itoa(int n, char s[]);
int main()
{
    int n;
    char s[20];
    scanf("%d", &n);
    itoa(n, s);
    printf("%s\n", s);
    return 0;
}
```

提示：关于字符串和其他数据类型的转换，推荐使用sscanf和sprintf这两个函数

问题3：将字符串颠倒




```
void reverse(char s[ ])
{
    int c, i, j;
    for(i=0, j=strlen(s)-1; i < j ; i++, j--){
        c = s[i];
        s[i] = s[j];
        s[j] = c;
    }
}
```

字符数组作为函数参数传递时，不需要同时传递数组长度。因为字符数组中字符串是以’\0’结束的

提示：数组作为函数参数传递时，要不要同时传递数组长度，取决于函数中是否知道数组中元素的个数

当有多个循环变量时，要用逗号隔开逗号表达式：如e1,e2顺序求e1和e2，以e2值作为整个表达式结果的值。如，a = (t = 3, t+2); 结果为5

13

问题4：根据输入整数决定输出小数点位数

- 由参数确定输出的小数位数
  - 从标准输入读入浮点数x (-10<x<10)和整数m (0<m<13)，在标准输出上输出sin(x)的值，保留到小数点后m位数字，如：
    - 输入：3.14 3， 输出：0.002；
    - 输入：3.14 10，输出：0.0015926529
- 问题分析
  - 如果直接用printf("%.#f", sin(x))，则需要用switch或if语句，有很多判断条件（这里#是常数，值跟输入的m相同）
  - 利用sprintf函数，根据输入的m构造格式字符串

14

使用字符串转换函数：sprintf

- 将多个普通变量输出为sprintf：变量->字符串
  - printf将不同变量输出到屏幕上，屏幕输出的本质就是字符串
  - sprintf用法与printf一样，只不过不是输出到屏幕，而是直接保存到字符串中
  - 比printf多一个参数（第一个参数），就是要存储的字符串数组，后面参数的使用与printf完全

```
//字符串转换函数，将其他变量转换为字符串
int sprintf(char *buf, char *format [, argument]…);
```

15

使用sprintf生成字符串

```
#include <stdio.h>
int main()
{
    int m;//小数点位数
    double x;//输入x
    char format[32];//要存储的格式串
    scanf("%lf%d", &x, &m);
    //利用sprintf生成格式串
    sprintf(format, "%%.%df\n", m);
    //利用生成的格式串控制输出格式
    printf(format, sin(x));
}
```

若输入m为3，则"%%.%df\n"变为"%.3f\n"，且该字符串存入字符数组format，利用该字符串控制输出格式

说明：斜杠是编译器级别的转义，%是printf内部的解析特殊符号，因此斜杠是不行的，只能是%%，不能是\%

sprintf 函数非常好用，任何可以输出到屏幕的数据都可以转换为字符串

16

问题5：从一行字符串中提取变量

- 提取日期和时间
  - 计算机显示的时间通常有特殊的格式，比如计算机给出的格式17/Apr/2018:10:28:28 +0800，表示北京时间2018年4月17日10时+28分28秒。给出一个这种格式表示的字符串，提取其中的每一项，并在屏幕上逐行显示出来
- 问题分析
  - 定义各种类型的输入变量
  - 编写格式控制符，从字符串中读入变量的值

17

使用字符串转换函数：sscanf

- 从字符串中转换出各种类型的变量，与sprintf对应
  - scanf从键盘读入一串符号，将其存入到不同变量中
  - sscanf用法与scanf一样，只不过不是从键盘输入，而是直接从一个给定的字符串中读入
  - 比scanf多一个参数（第一个参数），就是要存储的要读取的字符串，后面参数的使用与printf完全

```
//字符串转换函数，从字符串中提取各种变量
int sscanf(const char *buf, char *format [, arg]…);
```

18

使用sscanf从字符串中提取变量

```c
#include <stdio.h>
int main()
{
    int day, year, h, m, s;
    char mon[4], zone[6];
    char buf[] = "17/Apr/2018:10:28:28 +0800";//要转换的字符串
    //从字符串中，按照指定的格式读入日期的各个部分
    sscanf(buf, "%d/%3c/%d:%d:%d:%d %s",
           &day, mon, &year, &h, &m, &s, zone);
    mon[3] = '\0'; //理解此句话的作用
    printf("%d\n%s\n%d\n%d\n%d\n%d\n%s",
        year, mon, day, h, m, s, zone);//按照格式输出
    return 0;
}
```

```
2018
Apr
4
17
28
28
+0800
```

19

问题6：综合应用：扩展字符

【问题描述】从键盘输入包含**扩展符'-'**的字符串，将其扩展为等价的完整字符，
例如将a-d扩展为abcd，并输出扩展后的字符串。

- 要求：只处理[a-z]、[A-Z]、[0-9]范围内的字符扩展，即只有当扩展符前后的字符同时是小写字母、大写字母或数字，并且扩展符后的字符大于扩展符前的字符时才进行扩展，其它情况不进行扩展，原样输出。例如：a-R、D-e、0-b、4-B等字符串都不进行扩展。

【输入形式】：从键盘输入包含扩展符的字符串

【输出形式】：输出扩展后的字符串

【输入样例1】：ADEa-g-m02

【输出样例1】：ADEabcdefghijklm02

【输入样例2】：cdeT-bcd

【输出样例2】：cdeT-bcd

【样例说明】

- 将样例1的输入ADEa-g-m02扩展为：ADEabcdefghijklm02；样例2的输入cdeT-bcd中，扩展符前的字符为大写字母，扩展符后的字符为小写字母，不在同一范围内，所以不扩展

20

问题分析

- 基本字符（串）处理
  - ◆读写字符串：gets、puts
- 判断是否需要进行字符串扩展

```c
//判断是否需要扩展字符串
int isexpand(char start, char end);
```

- 字符串扩展

```c
//扩展字符串
for (ch = src[i] + 1; ch < src[i + 2]; ch++)
    dest[j++] = ch;
```

21

代码实现：主函数和判断是否扩展函数

```c
#include <stdio.h>
#include <ctype.h>
#include <string.h>
#define LEN 1024
//判断是否需要扩展字符串
int isexpand(char start, char end);
//将源串扩展为目标串
char *expand(const char *src, char *dest);
int main()
{
    char src[LEN], dest[LEN];
    gets(src);
    expand(src, dest);
    puts(dest);
    return 0;
}
```

```c
//判断是否需要扩展字符串
int isexpand(char start, char end)
{
  if (islower(start) && islower(end) && start < end)
      return 1;
  if (isupper(start) && isupper(end) && start < end)
      return 1;
  if (isdigit(start) && isdigit(end) && start < end)
      return 1;
  return 0;
}
```

22

代码实现：扩展字符串

```c
//将源串扩展为目标串
char *expand(const char *src, char *dest)  {
    char ch;
    int i, j;
    for (i = 0, j = 0; src[i] != '\0'; i++){
        dest[j++] = src[i];
        if (src[i + 1] == '-') {
            // 如果需要扩展，则扩展字符
            if (isexpand(src[i], src[i + 2])){
                for (ch = src[i] + 1; ch < src[i + 2]; ch++)
                    dest[j++] = ch;
                i++;
            }
        }
    }
    dest[j] = '\0'; //最后一定要补充\0
    return dest;
}
```

23

问题7：字符串替换

- 问题描述
  - ◆从标准输入读入数据中，每行最多包含一个字符串 "_xy_"。将输入行中的"_xy_"替换为"_abc_"，在标准输出上输出替换后的结果
- 问题分析
  - ◆如何找到”_xy_” ：strstr?
  - ◆找到后，如何替换为 “_abc_” ：数组操作、指针运算?
- 如何实现字符串查找函数?

24

字符串查找：库函数

- 子串查找库函数：strstr

```
//在字符串中查询是否存在子字符串sub_str
//返回中第一次出现的位置指针
//如果不存在，则返回NULL。
char *strstr(char *s, char *sub_str);
```

问题7：字符串替换

提示：BUFSIZ是一个常用的说明缓冲区大小的符号常量，实际数量取决于编译系统的版本和编译选项，常见的数值为512或4096

```
#include <stdio.h>
#include <string.h>
int main(){
    char buf[BUFSIZ], *p, *str = "_xy_";
    while (gets(buf) != NULL){ //读入字符串
        p = strstr(buf, str); //字符串中查询子串
        if (p == NULL){ //找不到子串，直接输出
            printf("%s", buf);
            continue;
        }
        *p = '\0';//将找到的位置设置为\0，从而输出前面的串
        //用%s输出时，通过buf只能输出前面的串
       //代码中直接输出_abc_子串
       //p+strlen(str)通过指针运算，输出_xy_后面的串
        printf("%s_abc_%s", buf, p + strlen(str));
    }
    return 0;
}
```

输出示例 abc_xy_ef

buf a b c _ x y _ e f \0

被替换的字符不输出
其它字符照常输出

用指针指向开始输出的位置

buf a b c \0 x y _ e f \0

p　　p+strlen(str)

从子串 "_xy_" 的第一个字符将输入分为两部分

字符串查找：strstr函数的代码实现

- 使用C代码实现（在字符串s中查找t）

```
int index(char s[], char t[])
{
    int i, j, k;
    for (i = 0; s[i] != '\0'; i++)
    {
        for (j = i, k = 0;
             t[k] != '\0' && s[j] == t[k];
             j++, k++)  ;
        if (t[k] == '\0')
            return i;
    }
    return -1;
}
```

主串s　i　j

子串t　k

思考1-不区分大小写

- 目前实现了大小写相关的字符串查找，即字符串"the"和"The"是不同字符串
- 如何实现大小写无关的字符串查找？

在比较字符时，可将要比较字符均转换为小写或大写，即可实现大小写无关查找

设函数char tolower(char c)用于将字符c转换为相应小写字符，则上面index可改为

```
int index(char s[ ], char t[ ]){
  int i, j, k;
  for(i =0; s[i] != '\0'; i++){
    for(j=i, k=0;
        t[k]!='\0'&&tolower(s[j])== tolower(t[k]);
        j++, k++)   ;
    if(t[k] == '\0')
       return ( i);
  }
  return ( -1);
}
```

思考2-字符串的查找算法

主串S: a b a b c a b c a c b a b

模式串T: a b c a c

i=0 i=1 i=2 i=6

j=0 j=1 j=2 j=4

算法的基本思想：

从主串s的第i个字符起和模式串的第一个字符比较之：

1）若相等，则继续逐个比较后续字符；否则从**主串的下一个字符起**再重新和**模式串的第一个字符**比较之

2）依此类推，直至模式串t中的每一个字符依次和主串s中的一个连续字符序列相等，则称匹配成功，函数值为和模式T中第一个字符相等的字符在主串S中的位置

3）否则称匹配不成功，函数返回-1

思考2-简单模式匹配算法实现：暴力破解

```
int index(char s[], char t[])
{
    int i, j, k;
    for (i = 0; s[i] != '\0'; i++)
    {
        for (j = i, k = 0;
             t[k] != '\0' && s[j] == t[k];
             j++, k++ )  ;

        if (t[k] == '\0')
            return (i);
    }
    return (-1);
}
```

显然，在最坏情况下（被查找的串不存在），该算法语句的最大执行频度为：N*M。
N为源串的长度，M为要查找的串的长度

时间复杂度为 O(n*m)

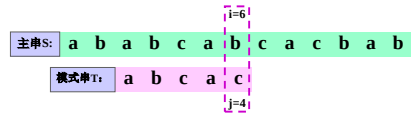
暴力破解（BF, Brute Force）法的另一种实现

- index查找函数使用两层循环，可改为只是用一层循环的字符串查找算法，但时间复杂度并没有变化

```
int index(char s[], char t[]){
    int i=0, j=0;   //设置查找的起始位置
    while(s[i] != '\0' && t[j] != '\0') {
        if(s[i] == t[j]) {  //若字符相等，继续查找下一个字符
            i++; j++;
        }
        else {      //若字符不等，则s中退回到上次查找开始的下一个位置
            i = i-j+1;
            j = 0;
        }
    }
    if(t[j] == '\0')    //查找到字符串t，返回t在s中的起始位置
        return i-j;
    else
        return -1;
}
```

31

思考3-更高效的模式匹配算法：串的模式匹配



KMP（Knuth-Morris-Pratt）算法

- 源串称为主串，定义为 S，当前匹配位置为 i；目标串称为子串，定义为T，当前匹配位置为 j
- 当前匹配在找到不匹配的字符后，重新开始匹配时：
  - 主串当前位置 **i 不回溯**，即不重置为上次匹配开始位置的一下位置
  - 子串当前位置 **j 视情况回溯**至起始串位置或子串中某一位置

32

KMP算法核心思想：计算子串回溯位置

根据子串T当前匹配的规律：　“$T_0 \cdots T_{k-1}$” = “$T_{j-k} \cdots T_{j-1}$”

由当前失配位置 **j** (已知)，可以归纳计算下次匹配起点 **k**的表达式。

令**k = next[j]**（**函数next用子串当前位置j来计算下次开始匹配位置k**，**k** 与**j** 显然具有函数关系），则

$$next[j] = \begin{cases} -1 & \text{当 } j=0\text{时} \\ \max\{k \mid 0<k<j \text{ 且 } `T_0 \cdots T_{k-1}` = `T_{j-k} \cdots T_{j-1}`\} & \\ 0 & \text{其他情况} \end{cases}$$

取当前位置j前序串首尾最大的相同子串长度

注意：

（1）**k**值仅取决于子串本身而与相匹配的主串无关。

（2）**k**值为子串从头向后及从**j**向前的两部分子串最大相同子串的长度。

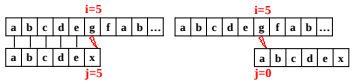
（3）这里的两部分子串可以有**部分重叠**的字符，但不可以全部重叠，即**k**最大为**j-1**。

33

KMP算法核心思想

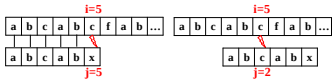
1. 子串T中无重复子串

j	0	1	2	3	4	5
子串T	a	b	c	d	e	x
next[j]	-1	0	0	0	0	0



2. 子串T中有重复子串

j	0	1	2	3	4	5
子串T	a	b	c	a	b	x
next[j]	-1	0	0	1	2	



34

KMP算法C语言实现

```
int KMPindex(char S[ ], char T[ ]) {
    int i = 0,  j=0, *next;
    next = (int *)malloc(sizeof(int)*(strlen(T)+1));
    getnext(T, next);  //求next
    while ( S[i]!='\0'  &&  T[j]!='\0') {
        if (S [i] == T[j] ) {
            i++;
            j++ ;
        }
       else
            //j回退到相应位置开始匹配，i不变
             (j == 0) ? i++ :( j = next[j]);
    }
    free(next);
    if ( T[j] == '\0')      //匹配成功，返回匹配位置
        return  i-j;
   else
        return -1;
}
```

算法分析：

由于指针i无须回溯，比较次数仅为n，加上计算next[j]时所用的比较次数m，比较总次数也仅为n+m，时间复杂度为**O(n + m)**，大大优于朴素的Brute-Force算法

35

KMP算法C代码实现：计算next

```
void getnext(char T[], int next[])
{
    int i = 0, j = -1;
    next[0] = -1;
    while (T[i] != '\0')
    {
        if (j == -1 || T[i] == T[j])
        { // i为后缀位置，j为前缀位置
            i++;
            j++;
            next[i] = j;
        }
        else
            j = next[j]; //若字符不同，则j值回溯
    }
}
```

36

问题4：思考4-更复杂的字符串查找

- 注意：index函数只能查找子字符串首次出现，如果一行中含有多个要查找的字符串怎么办？(类似Office软件中的查找功能)
  - 如何查找子字符串的最后一次出现?
  - 如果要查找一个字符串在一个文件中的所有出现（如给出所有出现的行列位置），如何实现?
- 在一个文件中查找给定串并用另一个替换串，如何实现？(Office软件中的替换功能)
- 如何实现模糊查找（如UNIX命令grep），如要查找的串形式为：comp?ter, com*er (?单字符匹配，*多字符匹配)

37

单选题 1分

有如下变量声明语句：

char *name[] = {"Zhangsan", "Lisi", "Wangwu"};

则表达式“*name[1] != *name[2]”比较的是___。

A **字符串"Zhangsan"和"Lisi"；**

B **字符'Z'和'L'；**

C **字符串"Lisi"和"Wangwu"；**

D **字符'L'和'W'。**

38

找错题

```
Main()
{
   char string1[10] = “hello”;
   char string[10] = {‘h’, ‘e’, ‘l’, ‘l’, ‘o’, ‘\0’};
   string1=“aaaaaaa”;        --错!
   ……
```

- 字符串以’\0’作为结束标志
- 字符串在定义中可以整体初始化，但不能使用赋值语句整体赋值