一. 字符串的定义

（由于C没有字符串的定义，所以我们不得不用数组和指针来模拟字符串）

1. 字符串字面量：

①在C中，用 ‘’ 表示字符，用“”表示字符串，编译器会自动在字符串后面添加 ‘\0’ ②**字符串的相加**：C语言中的字符串是不能用操作符相加的，应该在字符串中的间隔中添加空格或不加任何字符。

③在C的字符串中，如果要使用双引号，必须使用转义符\

④字符串字面量的另一种独特的使用方式——**作为地址**（**可以认为字符串是地址的另一种表现形式**）：const char * p = "Hello"，那么p存储的就是“Hello”字符串**第一个字符的地址**。此外，在printf的函数中，还可以把字符串字面量用%p的格式打印，打印出来的就是**第一个字符的地址**。

⑤字符串字面量还有一个比较重要的内容，那就是它是**静态存储类别**：该字符串只会被存储一次，在整个生命周期内存在。

2. 字符串数组：

①在使用数组时，应保证数组有足够的空间容纳字符串。**最小也要比字符串的长度多一**：

```
char a[40] = "Hello World!" ;      (注意未被使用的元素都将初始化为\0)
```

②还有一种方法：让编译器自动确定数组的大小：

```
char a[] = "Hello World!";
```

3. 字符串指针：

①使用格式： const *p = "Hello World!"; (p指向的类型仍然是字符类型)

②指针数组的使用：const char *p[LIM] = {"a", "b", "c"}; //可用来表示字符串数组 ②打印：可以用%s转换说明直接转换指针本身（这跟%s特殊的性质有关）

```
printf("%s", p);      // 这里打印出来的结果就是一个字符串
```

③一些说明：为什么指针指向字符串字面量的时候我要加**const**?

原因：加const是为了防止我们修改字符字面量中的内容；为什么不让我们修改？因为这是一个未定义的行为。（**在使用相同的字符串的时候，编译器可能会引用同一地址中的内容，这样我们修改了一处就把所有内容都修改了；然而有些编译器却不那么做**）

4. 数组和指针的比较：（然而现在有道云笔记都不原生支持表格了，呵呵☺）

①指针和数组都可以使用相同的表示方法：const char * p = "a"; a[0].....

（哪怕是指针数组的使用也是一样：用一个下标表示字符串，用两个下标表示字符）

②指针指向存储与静态内存中的字符串，而数组只是保存了字符串的副本，所以数组占的空间更大。③注意数组名是一个常量，它不可修改，但指针是一个变量，他可以修改（使用递增运算符等操作）④我们已经说明，使用const来修饰指针，这表明指针难以**修改**字符串，而数组则恰好相反。

二. 字符串输入

1. 说在前面的：想要把一个字符串读入程序，必须要**预留空间**。下面的做法是不行的： **char * name; scanf("%s", name); 为什么不行？因为name可能指向任何地方。** 标准的做法——使用数组表示法：char name[81]; //这样就预留了空间

2. 历史的遗珠：gets():（末尾一定有\0）

①记忆方法：get-string;

②作用：获取**一整行**输入并将其转化为字符串（这里的转化为字符串指的是在数组末尾添加\0） ③使用格式：gets(*数组名/指针*);

④致命的缺陷：可能会导致缓冲区溢出（buffer overflow）——多余的字符超出了指定空间（使用尚未使用的内存，就还可以接受；否则，就是砸出数据了）

3. 全新升级：gets_s():

①记忆方法：get-string-safely;

②作用：同样是获取**一整行**输入（至于换行符怎么处理，自己稍微想一想），但当读到最大字符处还没有换行符，毁了数组+程序。

③格式：gets_s(*数组名/指针名，最大字符数*);

4. 坚挺的难兄难弟：scanf():

①记忆方法和使用格式：这还用说吗？老朋友了；

②作用：读取**单词**

③同样的问题：使用%s可能会造成buffer overflow，如scanf("%s", "fahsdfhlsedhfnlskaj"); ④解决方案：使用最大字段宽度：scanf("%5s", "szhszhszh");

(最大字段宽度的作用：**scanf将读到最大字段宽度处或空白字符时停止)**

⑤可能出现的问题：多次读取可能会造成有部分数据留在缓冲区；这时要用getchar抛弃字符串。

5. 另类的函数：fgets() （**这个函数有一个很迷的操作：如果碰到换行符，不会给你加上\0；如果超过了最大的容量，在最后一位上添加\0**）

①记忆方法：file-get-string;

②作用：读取**文件中的字符串**；（读到**n-1**个字符，或换行符停止）。

③格式：fgets(*数组名/指针，最大字符数，读入的文件*); //如果时键盘输入就是标

准输入流stdout

④返回值：**一切顺利则返回数组的地址**；如果读取到文件结尾则返回NULL**(NULL不是C的关键字，而是宏)**；

⑤注意事项：由于是对文件内容的处理，所以保留换行符，仅在最大字符数减一（为了存储\0）处停止读取。

⑥一个技巧：利用循环可以读取玩全部内容，而不用在意最大字符数的问题。

```
puts("Enter strings (empty line to quit):");
while (fgets(words, STLEN, stdin) != NULL && words[0] != '\n') {
 fputs(words, stdout);
} // 这段程序可以照样输出你敲入的字符串
```

6. 自己写一个函数：

在这里我们写一个函数，他能够截取我们输入字符串的前n位，不够的话就算了。

```
char * s_gets(a, n) {
 char * ret_val;
 int index = 0;
 ret_val = fgets(a, n, stdin);
 // 如果没有读到文件末尾（这个基本没什么用0）
 if (ret_val) {
 // 找到第一个输入字符数不够或太多的索引
 while (a[index] != '\n' && a[index] != '\0') {
 ++index;
  }
  if (a[index] == '\0') {
  // ‘\0’代表读到最大字符处，剩下的就是过剩的了
  // 利用getchar不断丢弃直到行末'\n'
  while (getchar() != '\n') {
  continue
  }
  }
  else {
  // fgets最大的问题就是读到\n就停止了，也不做什么变化
  // 将已经读取的字符整合成完整的字符串
  a[index] = '\0';
  }
  }
  return ret_val;
}
```

三. 字符串的输出

1. puts:（**会输出换行符**）

①记忆方法：put-string

②使用格式：puts(*待打印字符串/**地址***)；

③作用：输出打印的字符串并在末尾添加\n（类似与python的print）；

④puts从给定的地址开始打印，直到遇到\0停止；

2. fputs（**不会输出换行符**）：

①记忆方法: file-put-string

②使用格式：fputs(*地址，文件名*)；　　// 如果要输出到屏幕上，用标准输出流stdout　　③作用：原样输出；

3. printf：

老朋友了，注意传入一个地址是可以用%s输出字符串的。

4. 配对：

Ⅰ。gets----puts:一个在输入后面**丢弃**换行符，一个在输出后面**添加**换行符

Ⅱ。fgets----fputs:一个在输入后面保留换行符，一个在输出后面不添加换行符

5. 建议使用：

```
char line[81];
// fgets在读到文件结尾返回NULL（不会返回EOF，因为其返回的始终是指针）
while (fgets(line, 81, stdin)) {
 fputs(line, stdout)
}
```

四. 自定义输入/输出

这个其实没什么好讲的，我们直接来看下面一段程序：

```
void put1(const char * string) {
 /* 不输出换行符的输出 */
 while (*string != '\n') {
 putchar(*string++)
 }
}
```

稍微分析一下：我们都知道不能对数组名进行自增操作，那么为什么可以对string进行自增操作呢？原来，其中有一个传参的过程，也就是说，string只是一个与数组名有相同指向的指针，故可以进行递增操作；至于其他的，之前都已经讲过了。

五. 字符串函数

（小提示：①在写代码的时候什么时候要在变量名和*加空格，什么时候又不用？**想加就加，想不加就不加**）

Ⅰ. strlen:

1. prototype: size_t strlen(const char * s);
2. 作用：返回给定字符串的字符个数；
3. 技巧：在给定字符串（一般是字符串常量）的SIZE索引处赋个\0，就可以用puts输出SIZE个大小的字符串了；

Ⅱ.strcat:

1. prototype: char * strcat(char * restrict s1, const char * restrict s2);
2. 作用：将s2拼接到s1的后面（s2已经被修改），返回这个结果；（s2的第一个字符会覆盖s1字符末尾的空字符）

Ⅲ. strncat:

1. prototype: char * strncat(char * restrict s1, const char * restrict s2, size_t n);
2. 作用：在strcat的基础上引入了最大长度这一个参数；
3. 题外话：为什么同样会导致溢出，gets被废弃，而strcat会被保留呢？原因是，gets会不会溢出取决于用户的输入；而strcat的溢出原因来自于程序员，与用户无关；

Ⅳ. strcmp

1. prototype: int strcmp(const char * s1, const char * s2);
2. 作用：比较字符串的机器排序序列（machine collating sequence），s1在s2前面，返回正数；s1与s2相同，返回0；s1在s2后面返回一个负数； **(s2 - s1)**
3. 注意：strcmp比较的是整个字符串而不是数组，这意味着结果与开辟的数组大小无关；另外，strcmp只能用于比较字符串，不能用于比较字符；

Ⅴ. strncmp

1. int strncmp(const char * restrict s1, const char * restrict s2, size_t n);
2. 作用：在strcmp的基础上，比较n个字符后停止

Ⅵ. strcopy

1. prototype: char * strcopy(char * s1， const char * s2);
2. 作用：将第二个字符串拷贝到s1中；
3. 骚操作：s1不必指向数组的第一个元素，也就是说：可以实现”部分“复制；
4. 警告：不能给未初始化的指针进行copy；**声明数组将分配储存空间，而声明指针只分配储存一个地址的空间。**

Ⅶ. strncopy

1. prototype: char * strncopy(char * restrict s1, const char * restrict s2, size_t n);

2. 作用：在strncopy的基础上，提供最大拷贝字符数。

3. 注意：在copy的过程中，copy的实际上是字符，**这样就有可能拷不到\0**，为了能拷到\0，我们选择在最后一位上加上\0；

Ⅷ. sprintf

1. prototype: 没提到；

2. 使用格式：sprintf(*数组/指针，格式字符串， 要转换的量的列表*)；

3. 作用：把格式化后的字符串赋给相应的数组；

（以上的函数充分地为我们展示了字符串的**拼接，比较和复制**，当然还有其他比较重要的，比如：**查找字符（strchr和strrchr），查找字符串（strpbrk和strstr），**这些都比较重要（尤其是查找字符）；对于上面一些函数还有要说的就是size_t是单位为字节的整型，可以理解为**字符个数**的类型）