话题二

Arrays and Pointers, Pt I

- 薛浩
- stickmind.com

今日话题

- 字符
- 字符串

推荐阅读：

- K&R (5.2-5.5)

补充 枚举

- 计算机表示非数值型数据，可以采用**枚举类型**的方式
- **枚举类型**是通过列出值域中所有元素来定义类型的一种方式。
  - 把枚举类型的每一个元素都绑定一个整型值
  - 行为类似于整型的类型，也称为**标量类型**（Scalar Type）

```
typedef enum {false, true} bool;

typedef enum {North=1, East, South, West} direction;
```

ASCII

- 每个字符都有一个特定的**字符代码**（char code）
- 目前已被标准化为 **ASCII 编码标准**。
  - 数字/大写/小写字母的整型值连续排列
  - 大写字母位模式通过对第 6 个位取反得到小写字母位模式（2^5）

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0	NUL	SOH	STX	ETX	EOT	ENQ	ACK	BEL	BS	HT	LF	VT	FF	CR	SO	SI
1	DLE	DC1	DC2	DC3	DC4	NAK	SYN	ETB	CAN	EM	SUB	ESC	FS	GS	RS	US
2	SP	!	“	#	$	%	&	‘	(	)	*	+	,	-	.	/
3	0	1	2	3	4	5	6	7	8	9	:	;	<	=	>	?
4	@	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O
5	P	Q	R	S	T	U	V	W	X	Y	Z	[	\	]	^	_
6	`	a	b	c	d	e	f	g	h	i	j	k	l	m	n	o
7	p	q	r	s	t	u	v	w	x	y	z	{	|	}	~	DEL

Hexadecimal to ASCII conversion table

字符类型 char

- 字符是所有文本数据处理的基础，是构建其他文本数据的“原子类型”
- char 是 C 中的基本类型，可以当作内建的枚举类型，用于表示单个字符。

```
char letterA     = 'A';
char plus        = '+';
char zero        = '0';
char space       = ' ';
char newLine     = '\n';
char tab         = '\t';
char singleQuote = '\'';
char backSlash   = '\\';
```

- 转义字符是由反斜杠和后面的字符组合而成，用于表示一些特殊字符

ASCII

- 字符的表示本质上和其他标量类型一致，基本思想是将每个字符映射到一个整型集。

```
char uppercaseA = 'A';      // Actually 65
char lowercaseA = 'a';      // Actually 97
char zeroDigit  = '0';      // Actually 48
```

字符运算与控制

- char 是标量类型，其操作行为类似整型

```
bool areEqual             = ‘A’ == ‘A’;      // true
bool earlierLetter        = ‘f’ < ‘c’;       // false
char uppercase            = ‘A’ + 1;
 int diff                 = ‘c’ - ‘a’;       // 2
 int numLettersInAlphabet = 'z’ - 'a' + 1;
 int numLettersInAlphabet = 'Z’ - 'A' + 1;
```

- 也可以用于 if/switch/while 等控制语句

```
// prints out every lowercase character
for (char ch = 'a'; ch <= 'z'; ch++) {
    printf("%c", ch);
}
```

ctype.h

- 在实际开发中，更多情况下不会直接操作字符，而是使用标准库接口。
- ctype.h 提供了一些常用的谓词函数和转换函数

Function	Description
`isalpha(`***ch***`)`	true if *ch* is `'a'` through `'z'` or `'A'` through `'Z'`
`islower(`***ch***`)`	true if *ch* is `'a'` through `'z'`
`isupper(`***ch***`)`	true if *ch* is `'A'` through `'Z'`
`isspace(`***ch***`)`	true if *ch* is a space, tab, new line, etc.
`isdigit(`***ch***`)`	true if *ch* is `'0'` through `'9'`
`toupper(`***ch***`)`	returns uppercase equivalent of a letter
`tolower(`***ch***`)`	returns lowercase equivalent of a letter

ctype.h

- 在实际开发中，更多情况下不会直接操作字符，而是使用标准库接口。
- ctype.h 提供了一些常用的谓词函数和转换函数

```
bool isLetter   = isalpha('A');        // true
bool capital    = isupper('f');        // false
char uppercaseB = toupper('b');
bool isADigit   = isdigit('4');        // true
```

良好的编码风格

使用库函数是一个比较好的编码习惯。

- 库是标准的，代码可读性更好，例如 toupper 比使用'a'-'A'计算更直观
- 库函数经过大量验证，正确性更有保证
- 库函数通常比自己设计的函数效率更高

问题？

今日话题

- 字符
- 字符串

推荐阅读：

- K&R (5.2-5.5)

数组 Arrays

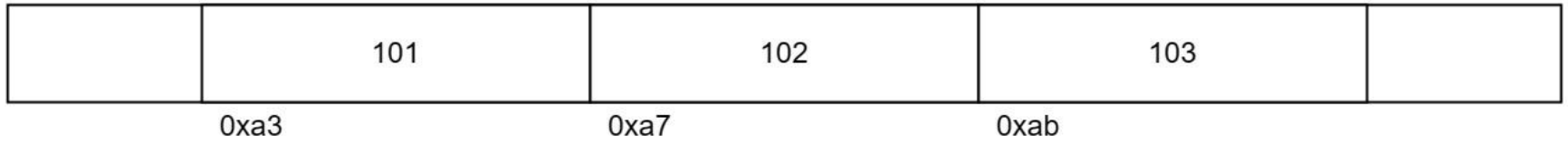
- 数组是一系列元素的集合，按顺序存储并且元素类型必须相同。
- 声明一个数组时，系统会在内存上分配一个连续的空间用于存储数组的内容。
- 数组可以通过指定数组的名字和对应元素的下标来指定某个特定元素

```
int arr[6];

arr[0] = 1;

arr[1] = 2;

// ...

arr[5] = 6;
int arr[] = {1, 2, 3, 4, 5, 6};
```

数组 Arrays

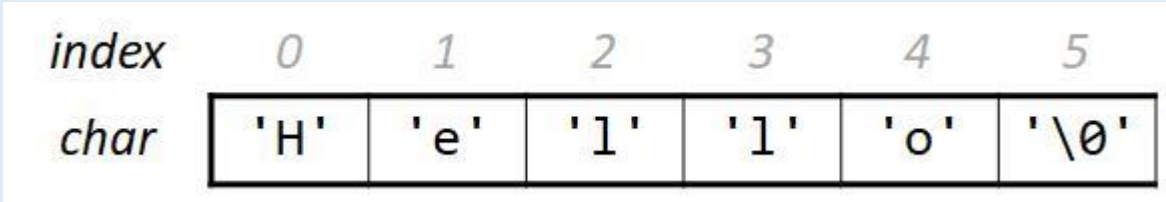
- **数据宽度**：char 表明每个元素占用 1 个字节；int 表明每个元素占用 4 个字节。
- **首元素地址**：数组变量名记录了数组首元素的地址。通过赋值给指针，我们可以将数组首元素地址保存到指针中。
- **数组长度**：数组还包含的元素个数，可以通过 sizeof 计算。

```
int arr[] = {101, 102, 103};
```



字符串

- C 语言中没有字符串类型，字符串是以字符数组的形式存在。
- 例如，字符串 "Hello" 使用字符数组表示如下：



- 特别的，字符串以特殊字符 '\0' 结尾，位模式是一个全零的字节。
- '\0' 称作 null-terminating 字符，也记作 NUL

字符串

- 以数组的观点来看待字符串，我们可以参照数组，使用 char[] 来声明字符串。
- 在分配字符串数组的时候，一定记得多分配一个字符空间用于存储终止字符。
- 访问或者修改字符串中的某个字符，可以通过字符数组下标。
- 通过数组名，也可以访问整个字符串，C 语言检测到终止字符会自动判断字符串结尾。

```
char myString[6];
myString[0] = 'H';
myString[1] = 'e';
myString[2] = 'l';
// …
myString[5] = '\0'; // don't forget it!!!
```

字符串

- 以数组的观点来看待字符串，我们可以参照数组，使用 char[] 来声明字符串。
- 在分配字符串数组的时候，一定记得多分配一个字符空间用于存储终止字符。
- 访问或者修改字符串中的某个字符，可以通过字符数组下标。
- 通过数组名，也可以访问整个字符串，**C 语言检测到终止字符会自动判断字符串结尾。**

```
printf("%c\n", myString[2]); // 'l'

myString[1] = 'a';

printf("%s\n", myString);    // "Hallo"
```

string.h

- 标准库 string.h 提供了大量的字符串操作函数，但是这些函数大多不会作条件检查，所以输入参数必须是合法的字符串，在必要时也可以加一些条件判断。

Function	Description
strlen(***str***)	returns the # of chars in a C string (before null-terminating character).
strcmp(***str1***, ***str2***), strncmp(***str1***, ***str2***, ***n***)	compares two strings; returns 0 if identical, <0 if ***str1*** comes before ***str2*** in alphabet, >0 if ***str1*** comes after ***str2*** in alphabet. ***strncmp*** stops comparing after at most ***n*** characters.
strchr(***str***, ***ch***) strrchr(***str***, ***ch***)	character search: returns a pointer to the first occurrence of ***ch*** in ***str***, or ***NULL*** if ***ch*** was not found in ***str***. strrchr find the last occurrence.
strstr(***haystack***, ***needle***)	string search: returns a pointer to the start of the first occurrence of ***needle*** in ***haystack***, or ***NULL*** if ***needle*** was not found in ***haystack***.
strcpy(***dst***, ***src***), strncpy(***dst***, ***src***, ***n***)	copies characters in ***src*** to ***dst***, including null-terminating character. Assumes enough space in ***dst***. Strings must not overlap. **strncpy** stops after at most ***n*** chars, and does not add null-terminating char.
strcat(***dst***, ***src***), strncat(***dst***, ***src***, ***n***)	concatenate ***src*** onto the end of ***dst***. **strncat** stops concatenating after at most ***n*** characters. Always adds a null-terminating character.
strspn(***str***, ***accept***), strcspn(***str***, ***reject***)	**strspn** returns the length of the initial part of ***str*** which contains only characters in ***accept***. **strcspn** returns the length of the initial part of ***str*** which does not contain any characters in ***reject***.

长度 strlen

- 和 C++ 中 std::string 不同的是，C 语言的字符串不是对象，不包含字符串相关的信息（例如字符串的长度）。
- 如果想计算一个字符串的长度，我们可以使用 strlen 标准函数，结尾终止字符不会统计在内。
- 对于 "Hello" 字符串，计算结果如下：

```
char myString[] = {'H', 'e', 'l', 'l', 'o'};

int length = strlen(myString);
```

辨析 数组长度 vs 字符串长度

- sizeof 除了可以计算数据类型的长度，还可以用于计算数组的大小。

  语法类似于 `sizeof(arr) / sizeof(arr[0])`

- 由于字符串会添加终止字符到末尾，所以不可以使用该语法替代 strlen。

MUSL LIB

```c
#define ALIGN (sizeof(size_t))
#define ONES ((size_t)-1/UCHAR_MAX)
#define HIGHS (ONES * (UCHAR_MAX/2+1))
#define HASZERO(x) ((x)-ONES & ~(x) & HIGHS) // Lab 1 Challenge Problem

size_t strlen(const char *s) {
    const char* a = s;
#ifdef __GNUC__
    typedef size_t __attribute__((__may_alias__)) word;
    const word* w;
    for (; (uintptr_t)s % ALIGN; s++) if (!*s) return s-a;
    for (w = (const void *)s; !HASZERO(*w); w++);
    s = (const void *)w;
#endif
    for (; *s; s++);
    return s-a;
}
```

检测全 0 字节

在 musl libc 库中很常见，目前的写法是使用宏定义，通过并行检查提高性能。

HASZERO(x) 由三个部分组成，考察对于 MSB 的置 1 操作

- (x)-ONES 可以对数集 {0x81, 0x82, 0x83, ..., 0xFF, 0x00} 的 MSB 置 1
- ~(x) 可以对数集 {0x00, 0x01, 0x02, 0x03, ..., 0x7F, 0x80} 的 MSB 置 1
- HIGHS 所有字节的 MSB 均已置 1，用于过滤 MSB 位
- 最后，(x)-ONES & ~(x) & HIGHS 彼此作交集，**仅 {0x00} 的 MSB 可以置 1**，其他情况结果全为 0。所以，只有全 0 字节的结果不等于 0。