话题二

Arrays and Pointers, Pt 2

- 薛浩
- stickmind.com

今日话题

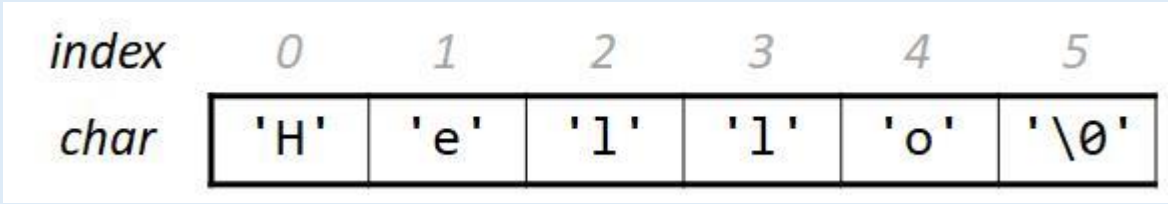
- 字符串作为数组
- string.h
- 字符串作为指针

推荐阅读：

- K&R (5.2-5.5)

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，以特殊字符 '\0' 结尾。
- 例如，字符串 "Hello" 使用字符数组表示如下：



- 以数组观点看待字符串，可以使用如下方式声明和初始化：

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

今日话题

- 字符串作为数组
- string.h
- 字符串作为指针

推荐阅读：

- K&R (5.2-5.5)

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。

比较 strcmp

- 比较操作符，比如 ==，< 或 > 等，无法比较字符串。下面的写法，实际上是在比较字符串**首字符的地址**。

```
// e.g. str1 = 0x7f42, str2 = 0x654d
void doSomething(char str1[], char str2[]) {
    if (str1 > str2) { // compares 0x7f42 > 0x654d!
        // …
    }
}
```

比较 strcmp

- 比较两个字符串可以使用 strcmp(str1, str2)，该函数会依次比较字符数组中的每个字符，并返回一个整型。
  - 两个字符串相等时，返回 0
  - str1 中的字符比 str2 中的字符靠前，返回负值
  - str1 中的字符比 str2 中的字符靠后，返回正值

```
int compResult = strcmp(str1, str2);

if (compResult == 0) {
    // equal
} else if (compResult < 0) {
    // str1 comes before str2
} else {
    // str1 comes after str2
}
```

拷贝

- 赋值操作 = 并不能将一个字符串拷贝给另一个字符串变量。
- 下面的写法，实际上是拷贝字符串数组首字符的地址。

```
// e.g. param1 = 0x7f42, param2 = 0x654d
void doSomething(char param1[], char param2[]) {
    param1 = param2;   // copies 0x654d. Points to same string!
    param2[0] = 'H';   // modifies the one original string!
}
```

拷贝整体字符串 strcpy

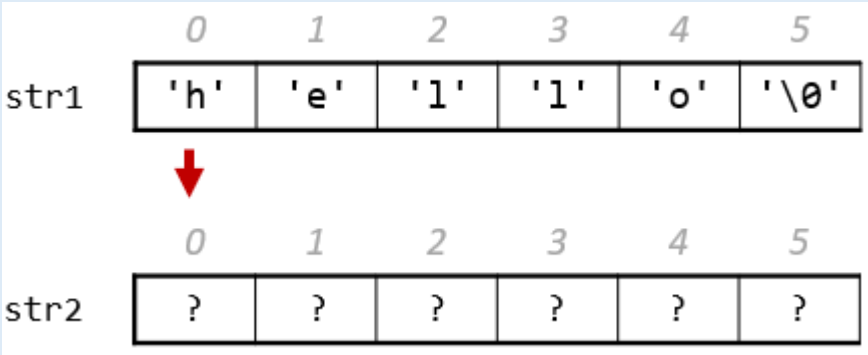
- 拷贝整体字符串可以使用 strcpy(dst, src)
- 该函数会将 src 字符数组拷贝到 dst 字符数组中，包括 '\0' 终止字符。
- 注意该函数是有返回值的，返回 dst 字符串。

```
char str1[6];
strcpy(str1, "hello");

char str2[6];
strcpy(str2, str1);
str2[0] = 'c';

printf("%s", str1);    // hello
printf("%s", str2);    // cello
```

拷贝整体字符串 strcpy

- 使用 strcpy 要特别注意，dst 数组大小一定不能小于 src 数组。
- 以下代码是可以正常拷贝的，数组的大小恰好可以容下字符串 "hello" 和一个终止符。

```
char str1[6];
strcpy(str1, "hello");

char str2[6];
strcpy(str2, str1);
```



缓冲区溢出

- 下述代码，str2 无法容下 "hello, world!"，但是拷贝操作依然会进行，这样造成的后果是 str2 后面的字节将被覆盖。

```
char str1[14];
strcpy(str1, "hello, world!"); // correct
char str2[6];
strcpy(str2, str1);  // overwrites other memory!
```

如果 str2 后面的字节被系统中其他程序使用，那么覆盖写入将会造成严重的安全问题。

我们将这种越界写入问题，称作**缓冲区溢出**（buffer overflow）。

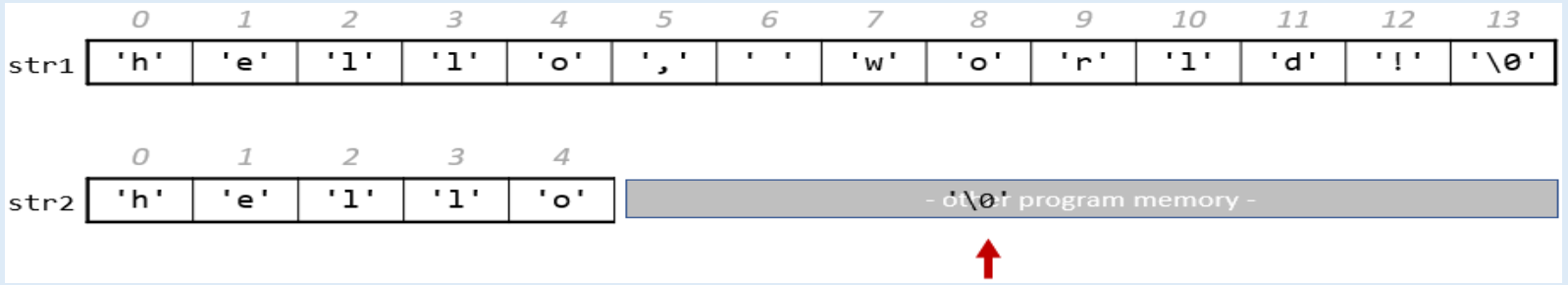
拷贝部分字符串 strncpy

- 拷贝部分字符串可以使用 strncpy(dst, src, n)，与 strcpy 不同的是 strncpy 并**不保证 dst 字符串以 '\0' 结尾**。

```
// copying "hello"
char str2[5];
strncpy(str2, "hello, world!", 5);  // doesn't copy '\0'!
```

- 没有 '\0' 终止符的字符数组：如果字符数组没有 '\0' 终止符，那么系统就无法确定字符串的结尾。
- 由于没有终止字符，在调用 strlen 时，会一直计算到随后的系统其他部分的 '\0' 处。

```
char str2[5];
strncpy(str2, "hello, world!", 5);  // doesn't copy '\0'!
int length = strlen(str2);
```



拷贝部分字符串 strncpy

- 拷贝部分字符串可以使用 strncpy(dst, src, n)，与 strcpy 不同的是 strncpy 并**不保证 dst 字符串以 '\0' 结尾**。

```
// copying "hello"
char str2[5];
strncpy(str2, "hello, world!", 5);  // doesn't copy '\0'!
```

- 没有 '\0' 终止符的字符数组：如果字符数组没有 '\0' 终止符，那么系统就无法确定字符串的结尾。
- 由于没有终止字符，printf 也会一直打印到有 '\0' 的地方，输出的结果可能就会包含乱码：hello□□J□□□。

```
char str2[5];
strncpy(str2, "hello, world!", 5);  // doesn't copy '\0'!
printf("%s\n", str1);
```

拷贝部分字符串 strncpy

- 拷贝部分字符串可以使用 strncpy(dst, src, n)，与 strcpy 不同的是 strncpy 并**不保证 dst 字符串以 '\0' 结尾**。

```
// copying "hello"
char str2[5];
strncpy(str2, "hello, world!", 5);  // doesn't copy '\0'!
```

- 为了避免这样的问题，一般需要程序员多分配一个字符，单独追加一个终止符。

```
// copying "hello"
#define NUM 5

char str2[NUM + 1];  // extra room for string and '\0'
strncpy(str2, "hello, world!", NUM);  // doesn't copy '\0'!
str2[5] = '\0';  // add null-terminating char
```

拼接

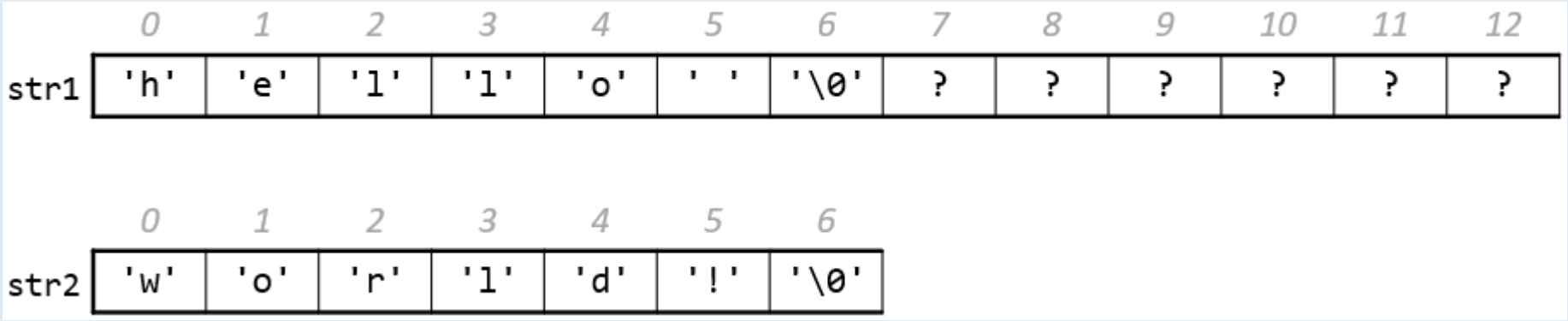
- 同样，拼接字符串也不能直接使用 + 操作符。
- 下面的写法，实际上是相加两个字符串首字符的地址。

```
// e.g. param1 = 0x7f, param2 = 0x65
void doSomething(char param1[], char param2[]) {
    printf("%s", param1 + param2);  // adds 0x7f and 0x65!
    // ...
}
```

拼接 strcat strncat

- 拼接两个字符串可以使用 strcat(str1, str2)，拼接指定的几个字符可以使用 strncat(str1, str2, n)。
- 方便的是，这两个函数在拼接过程中，会**先去除原始字符串的终止字符**，并能够保证返回的结果字符串以 '\0' 结尾。
- 以下示例创建了两个字符串，其中 str1 所在字符数组**留有足够空间**用于拼接字符串 str2。

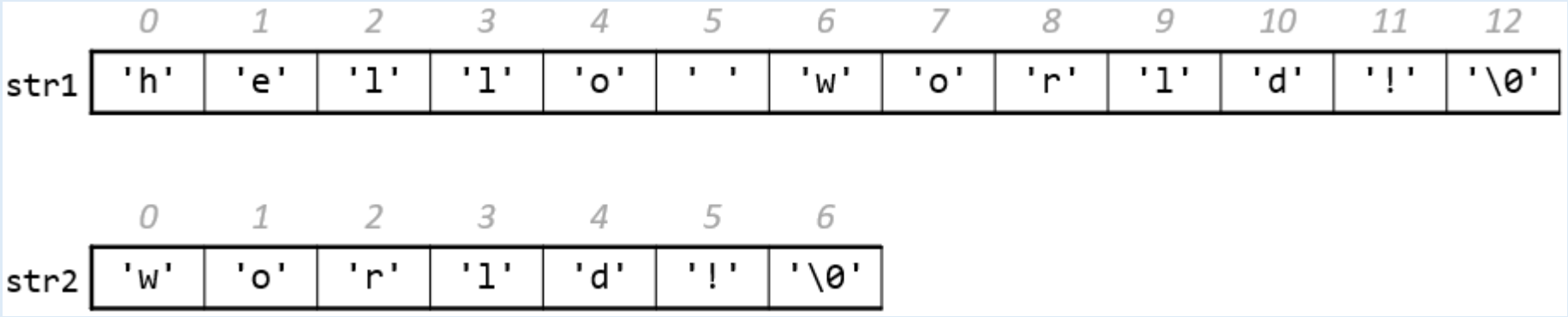
```
char str1[13];  // enough space for strings + '\0'
strcpy(str1, "hello ");
char str2[7];
strcpy(str2, "world!");
strcat(str1, str2);  // hello world!
```



拼接 strcat strncat

- 拼接两个字符串可以使用 strcat(str1, str2)，拼接指定的几个字符可以使用 strncat(str1, str2, n)。
- 方便的是，这两个函数在拼接过程中，会**先去除原始字符串的终止字符**，并能够保证返回的结果字符串以 '\0' 结尾。
- 以下示例创建了两个字符串，其中 str1 所在字符数组**留有足够空间**用于拼接字符串 str2。

```
char str1[13];  // enough space for strings + '\0'
strcpy(str1, "hello ");
char str2[7];
strcpy(str2, "world!");
strcat(str1, str2);  // hello world!
```



间距 strspn

- strspn(str, accept) 计算字符串的开头部分**包含**第二个参数中出现的字符的长度。

```
char bailey[11];
strcpy(bailey, "Bailey Dog");
int spanLength = strspn(bailey, "aBeoi");
printf("%d\n", spanLength); // 3

char bailey[11];
strcpy(bailey, "gailey Dog");
int spanLength = strspn(bailey, "aBeoi");
printf("%d\n", spanLength); // 0
```

间距 strcspn

- strcspn(str, reject) 是互补函数，计算字符串开头部分**不包含**第二个参数中出现的字符的长度。

```
char bailey[11];
strcpy(bailey, "Bailey Dog");
int spanLength = strcspn(bailey, "aBeoi");
printf("%d\n", spanLength); // 0

char bailey[11];
strcpy(bailey, "gailey Dog");
int spanLength = strcspn(bailey, "aBeoi");
printf("%d\n", spanLength); // 1
```

搜索字符 strchr strrchr

- strchr 搜索字符串中的字符，返回一个指向该字符的指针。
  - 如果字符串中有多个相同的字符，则返回首个字符的指针；
  - 如果字符串中没有该字符，则返回 NULL。
- strrchr 区别于 strchr，如果字符串中有多个相同的字符，则返回最后一个字符的指针

```
char myString[6];
strcpy(myString, "Hello");
char* charL = strchr(myString, 'l');
char* charL2 = strrchr(myString, 'l');

printf("%s\n", myString);  // Heloo
printf("%s\n", charL);     // llo
printf("%s\n", charL2);    // lo
```

搜索子串 strstr

- strstr 搜索字符串中的子字符串，并返回一个指向该字符串的指针。
  - 如果字符串中有多个相同的子字符串，则返回**首个**子字符串的指针；
  - 如果字符串中没有该子字符串，则返回 NULL。

```
char bailey[11];
strcpy(bailey, "Bailey Dog");
char* substr = strstr(bailey, "Dog");
printf("%s\n", bailey);  // Bailey Dog
printf("%s\n", substr);  // Dog
```

今日话题

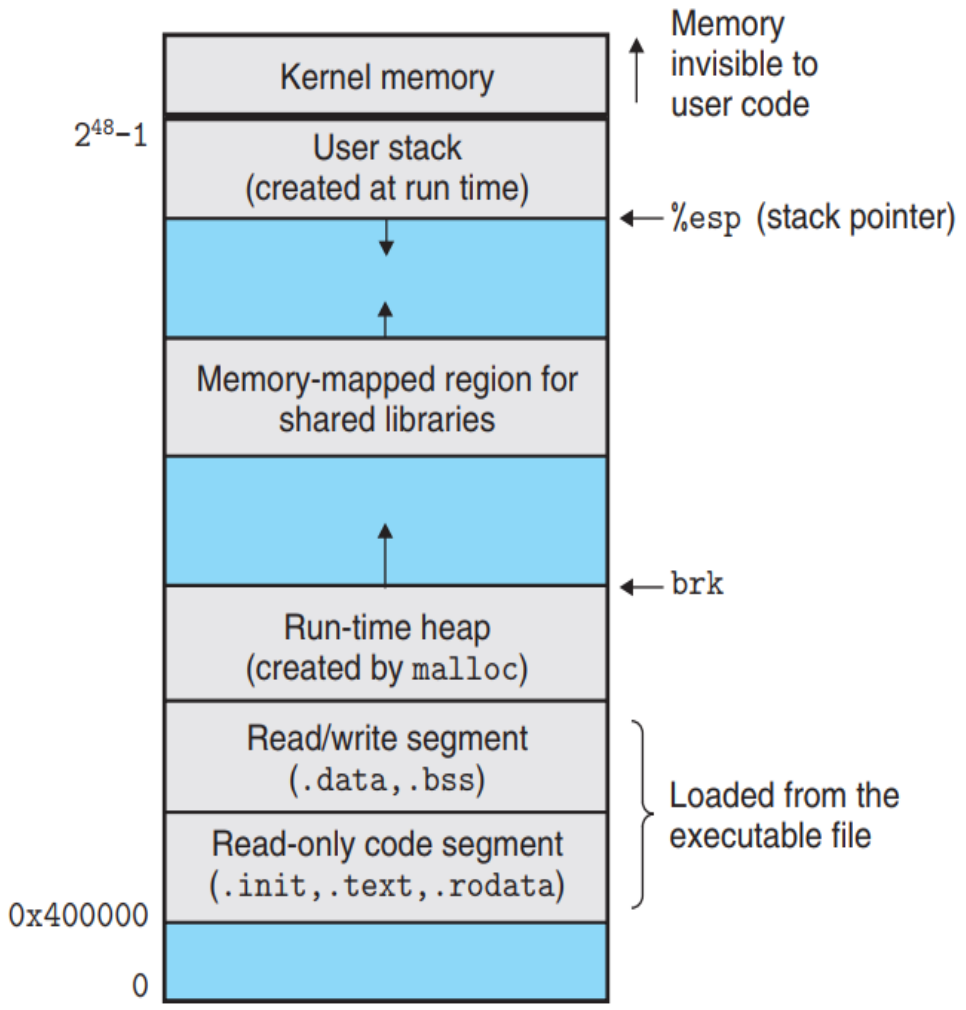
- 字符串作为数组
- string.h
- 字符串作为指针

推荐阅读：

- K&R (5.2-5.5)

内存 Memory

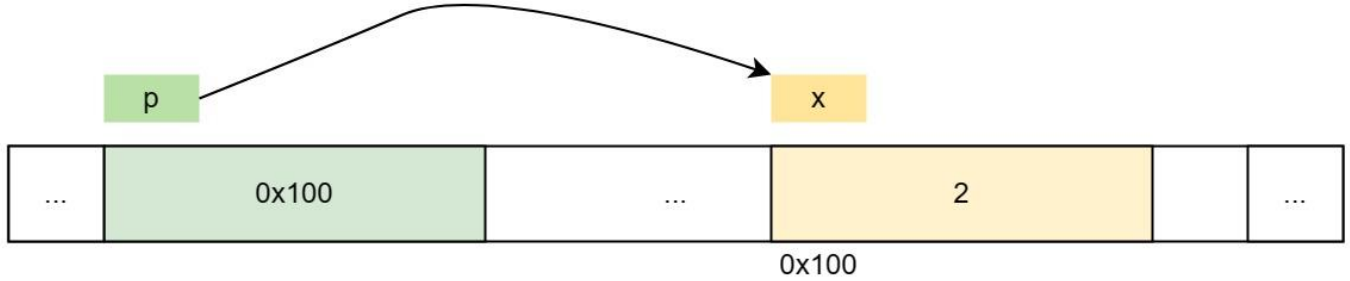
- **内存**可以看作是一个“大号数组”，以字节为单位，每个字节都有一个独特的地址，通常以十六进制形式表示。
- 在 C 语言中，使用**指针**特殊数据类型来抽象内存的地址。通过指针，我们才可以进行**内存相关的操作**。



址针

- 指针变量用于存储内存的地址，指针本身占用一个字（word）的内存空间。
- 对于任意大小的数据，都可以使用指针表示其内存地址。
  - 声明一个指针可以使用 **type* varName** 语法
  - 获取变量的地址可以使用取址 **&** 操作符
  - 获取指针指向的地址内的数据可以使用间接引用 * 操作符

```
int x = 2;
int* p = &x;
printf("%d", *p);
```



NULL 址针

- 初始化一个指针，如果暂时不指向任何数据，可以使用特殊常量 **NULL**。该常量可以赋值给任何类型的指针，在系统内部表示地址值 0。
- 一旦指针的值为 NULL，不要使用 * 操作符来间接引用该指针。

```
int* p = NULL;
int i = *p;  // ERROR
```