東南大學 SCHOOL OF INTEGRATED CIRCUITS, SEU

集成电路学院

计算机科学基础I

—— 字符串、引用

东南大学 集成电路学院 朱彬武

E-mail: bwzhu@seu.edu.cn

回顾——指针运算

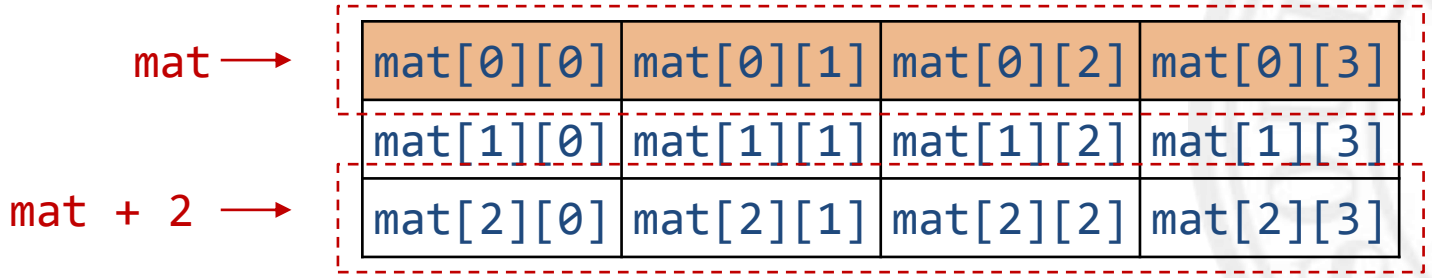
- 指针 + n：并不是简单地把地址加 n，而是地址增加 n * sizeof(指针所指向的类型）个字节
- *p++：*的优先级虽然高，但是没有++高，*p++ ==> *(p++)

```
int a[5] = {1,2,3,4,5};
int *q = a;
cout << *q++ << endl; // 1
cout << *q++ << endl; // 2
cout << *q++ << endl; // 3
```

回顾——二维数组名

对于二维数组int mat[M][N];

- mat的类型：int (*)[N]，区别于int *[N]，前者是数组指针，后者是指针数组
- mat+1：跳过一整行（N个int）
- mat[i][j] == *(*(mat+i)+j)



回顾——指针常量和常量指针

- 常量指针： `const int* p` ：不能改*p，可改p
- 指针常量： `int* const p` ：能改*p，不能改p，必须要初始化
- `const int* const p` ：都不能改
- 总结：const修饰谁，谁就不能被改

字符串

字符数组

怎么存字符串？ 可以直接用这样的字符数组吗？

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

word[0]	'H'
word[1]	'e'
word[2]	'l'
word[3]	'l'
word[4]	'o'

- 这个字符数组不是有效的C风格字符串，C语言的设计者选择以'\0'作为字符串的结束标志，这一设计被C++继承。（C++有另外自己的std::string类）
- 为什么要以'\0'作为结束标志？简单来说可以作为终止信号，且不干扰输出

C风格字符串

- C风格的字符串：`char word[] = {'H','e','l','l','o','\0'}`

word[0]	'H'
word[1]	'e'
word[2]	'l'
word[3]	'l'
word[4]	'o'
word[5]	'\0'

- '\0'的ASCII码就是0，因此也可以用字面量0替代，不是'0'
- 字符串以数组的形式存在，以数组或指针的形式访问，一般指针为主

字符串定义和初始化

- 列表初始化：`char s[] = {'H','i','\0'};`
- 部分初始化：`char s[6] = {'H','i','\0'}; // 剩余空间全补 0`
- 字符串常量初始化：`char s[] = "Hi"; // 长度自动定为3`

  `char s[5] = "Hello";` ❌ 空间不足

字符数组可以用字符串常量来初始化，且可以自动补'\0'

字符串输出

- 输出字符串：cout << <字符数组名> ; 打印到'\0'停止

```
void traverseString(const char *s){
    while(*s != '\0'){
        cout << *s++;
    }
}
```

- 假设字符数组末尾没有'\0'：

```
char s[] = {'H','e','l','l','o'};
cout << s; //紧接着是一堆乱码，直到在内存中随机撞到一个 0 为止
```

字符串输入

- 输入字符串：cin >> <字符数组名> ；遇到空格/回车/制表符就会停止读取。

```
char s[20];
cin >> s;    //输入Hello World按下回车
cout << s;   //Hello
```

- 如果想在字符串中存空格/回车/制表符，使用cin.get()或cin.getline()，默认是换行结束

```
char s[20];
cin.getline(s, 20); // 最多读取19个字符，预留1个给\0
cin.getline(s, 20, ',');
// 以逗号为结束标志，输入Hello,World再按下回车，存入Hello
```

字符串常量和字符指针

- 字符串常量还可以用来初始化char *，实际上是用字符串首元素的地址初始化s

```
char *s = "Hello"; //s指向'H'的地址
```

- 此时"Hello"存储在内存的常量区
- 由于数据在常量区，只有读权限，没有写权限，直接定义为char *在现代编译器会被发出警告，严格写法如下：

```
const char *s = "Hello";
s[1] = 'a'; ❌ 只读权限，无法修改

char s[] = "Hello";
s[1] = 'a'; ✅ 在栈区，可以修改
```

字符串函数

cstring头文件

- cstring头文件中包含了许多处理字符串的函数
  - strlen
  - strcmp
  - strcpy
  - strcat
  - strchr
  - strstr
- 要使用这些函数需要包含相应的头文件：`#include<cstring>`

strlen

- 函数原型：size_t strlen(const char *s);
- 返回字符串s的有效长度（不包含结尾的0）
- 注意区分和sizeof的不同
- 函数定义：

```
size_t mylen(const char *s){
    size_t cnt = 0;
    while(*s++ != '\0'){
        cnt++;
    }
    return cnt;
}
```

```
char s[] = "Hello";
cout << sizeof(s) << endl; //6
cout << strlen(s) << endl; //5

char s[7] = "Hello";
cout << sizeof(s) << endl; //7
cout << strlen(s) << endl; //5
```

strcmp

- 字符串比较：不要想当然用s1 == s2来判断，这时比较的是地址
- 函数原型：int strcmp(const char *s1, const char *s2);
- 比较两个字符串的内容，返回两个字符串第一个不同字符的 ASCII 码差值；相同则返回 0。

```
char s1[] = "abc";
char s2[] = "abc";
char s3[] = "dbc";
cout << strcmp(s1, s2) << endl; // 0
cout << strcmp(s1, s3) << endl; //'d' - 'a' ==> 3
```

- 如果自己实现呢？放在作业里，注意考虑字符串的结束标志。

strcpy

- 字符串赋值：不要想当然用s1 = s2来做，数组名是指针常量，不能被修改
- 函数原型：char *strcpy(char* dst, const char* src);
- 把src的字符串拷贝到dst，函数返回dst

```
char src[] = "hello";
char dst[20];          // 要保证有存'\0'的空间
strcpy(dst, src);

cout << dst << endl; // hello
```

- 注意：函数的返回类型是一个指针类型char *

返回类型是指针的函数

- 函数的返回类型不仅可以是值，也可以是指针

```
T* findMax(T arr[], int n)
{
    T* maxPtr = arr;
    for (int i = 1; i < n; i++)
    {
        if (arr[i] > *maxPtr)
        {
            maxPtr = &arr[i];
        }
    }
    return maxPtr;
}
```

- 假设T是一种自定义类型，sizeof(T)所占字节数很大，返回地址效率更高

更多字符串处理函数

- 拼接：将src的字符串拼接到dst的末尾，返回拼接好的字符串

```
char* strcat(char* dst, const char* src);
```

- 查找（1）：在字符串s中查找特点字符c，返回查找到的地址，没有查到返回NULL

```
char* strchr(const char* s, int ch);
```

- 查找（2）：在字符串s中查找字符串sub，返回查找到的地址，没有查到返回NULL

```
char* strstr(const char* s, const char* sub);
```

- 更多函数包括strlwr，strupr，strrev等等大家可自行查阅CPP官方文档说明：
  https://en.cppreference.com/w/cpp/header/cstring.html

字符串数组

二维字符数组

- 如何定义字符串数组？ 二维字符数组是否可行？

```
char words[][9] = {
        "Cat",
        "Rabbit",
        "Elephant"
    };
```

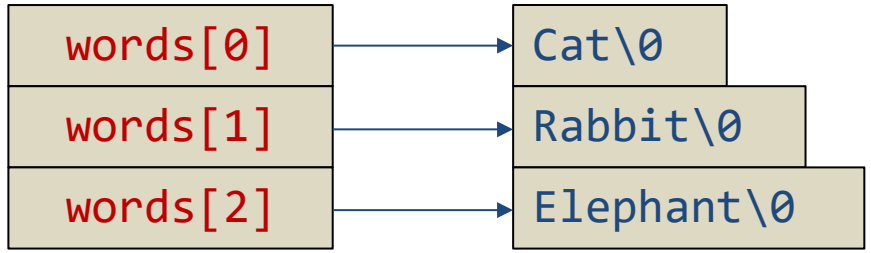
words[0]	Cat\0
words[1]	Rabbit\0
words[2]	Elephant\0

- 存在的问题：不灵活，每个字符串的长度固定且不能超过N，会浪费内存空间

字符指针数组


```
char *s1 = "Cat";
char *s2 = "Rabbit";
char *s3 = "Elephant";
char *words[] = {s1, s2, s3};
```

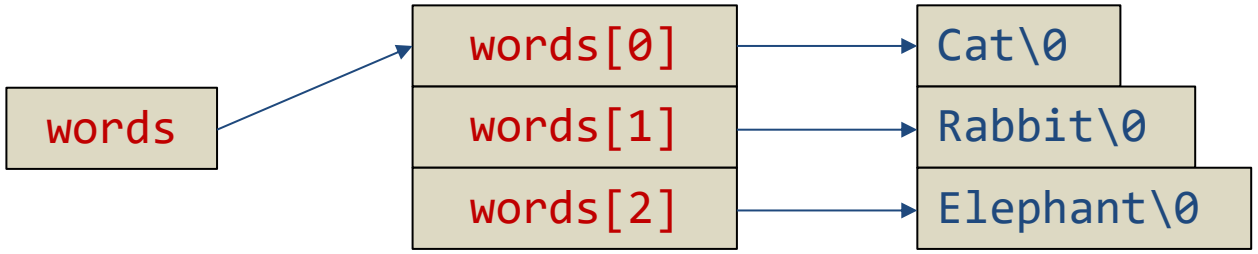
```
char *words[] = {
    "Cat",
    "Rabbit",
    "Elephant"
};
```



- 字符指针数组：数组中的每个元素都是char *（指针）
- 每个数组元素存了字符串首元素的地址，相比 char 的二维数组，不需要为每个字符串预留固定长度空间
- 这些字符串通常是字符串常量，存放在只读区，内容不允许被修改，定义时最好加上const

理解指针数组

```
char *words[] = {
        “Cat",
        “Rabbit",
        “Elephant"
    };
```



- *words和words[1]是什么？ 'C'的地址和'R'的地址
- *(words+1)和*words+1是什么？ 'R'的地址和'a'的地址
- words[0][1]和words[1][3]是什么？ 'a'和'b'
- *words[1]和(*words)[1]是什么？ 'R'和'a'

提示：[]运算符优先级高于*运算符

引用：C++指针的重要扩充

我们用指针做什么

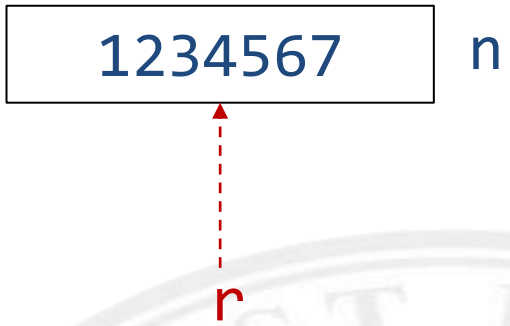
- 用函数来修改不止一个变量
- 让函数返回不止一个结果
- 传入较大的数据时使用传地址替代传值

以上场景，在C++中都可以用引用（reference）来更安全、更方便地实现

引用：变量的别名

- 某个变量的引用，等价于这个变量，相当于该变量的“别名”。

```
int n = 1234567;
int &r = n;
n = n + 1;
cout << r << endl; //1234568
r = r - 1;
cout << n << endl; //1234567
```



- 语法格式：<类型> &<引用名> = <变量名>

符号	*	&
类型标志	指针	（左值）引用
双目运算	乘	位与
单目运算	解引用	取地址

引用的注意事项

- 定义引用时一定要将其初始化
- 初始化后，它就一直引用该变量，不会再引用别的变量（类似指针常量）
- （左值）引用只能引用变量，不能引用常量和表达式（类比取地址运算）

```
int x = 1;
int y = 2;
int &r = x;
r = y;
cout << x << endl;
```

⚠ 此时没有建立新的引用关系，而是把x的值改成了y

```
int &r = 1; ❌
double &r = (double) x; ❌
```

常引用

- 常引用：不能通过引用修改其引用的内容（类似常量指针）

```
int x = 1;
const int &r = x;
r = 2; ❌ 常引用，不能修改
x = 2; ✅ x是int类型
const int y = 2;
int &r2 = y; ❌ 权限放大
```

- T类型的变量或T &类型的引用可以用来初始化const T &类型的引用
- const T 类型常变量的或const T &类型的常引用不能用来初始化T &类型的引用

函数传引用


```
void swap(int &a, int &b){
    int temp = a;
    a = b;
    b = temp;
    return;
}


int main(){
    int a = 3;
    int b = 4;
    swap(a, b);
    cout << a << " " << b;    //4 3
}
```