東南大學 SCHOOL OF INTEGRATED CIRCUITS, SEU

集成电路学院

计算机科学基础I

——全局变量、静态局部变量、宏

东南大学 集成电路学院 朱彬武

E-mail: bwzhu@seu.edu.cn

回顾——字符串初始化、输入和输出

- 字符串常量初始化：
  - 字符数组：`char s[] = "Hi"; //在栈区分配内存，可读可写`
  - 字符指针：`char *s = "Hi"; //在常量区分配内存，可读不可写，最好加const`
- 字符串输出：
  - `cout << <char *>`：碰到\0就停止输出
- 字符串输入：
  - `cin >> <char *>`：读取到空格/换行/Tab就停
  - `cin.getline(字符数组名，写入长度，结束标志)`：若写入长度是n，最多读入n-1个字符，最后自动补\0，不需要手动输入，结束标志不写则默认换行结束

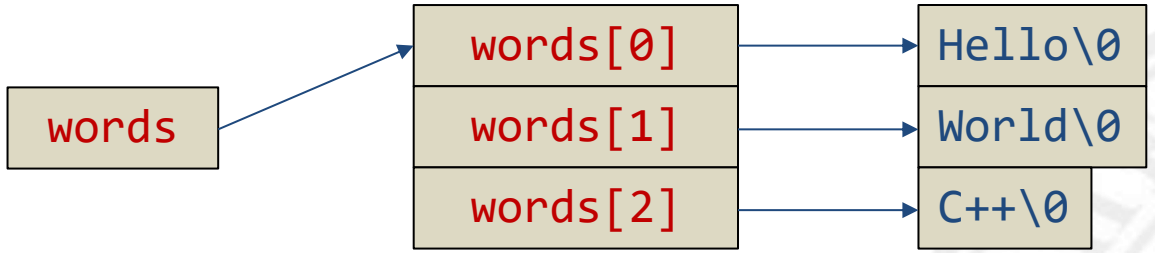
回顾——字符串函数

- `size_t strlen(const char *s)`：返回字符串s的有效长度，注意和sizeof的区别
- `int strcmp(const char *s1, const char *s2)`：相同返回0
- `char* strcpy(char* dst, const char* src)`：把src的字符串拷贝到dst，以\0为结束标志，拷贝后自动补\0，函数返回dst
- `char* strcat(char* dst, const char* src)`：从dst原有\0位置开始，拷贝src，直到\0，拼接后自动补\0，函数返回dst

```
char s[] = "abcd";          // a b c d \0
char t[] = "c\0d";          // c \0 d \0
cout << strcpy(s, t);       // c \0 c d \0
s[1] = 'x';
cout << s;                  // c x c d \0
char s2[] = "a\0bcd";       // a \0 b c d \0
char t2[] = "e\0f";         // e \0 f \0
cout << strcat(s2, t2);     // a e \0 c d \0
s2[2] = 'x';
cout << s2;                 // a e x c d \0
```

回顾——字符串数组

- 字符串数组采用字符指针数组作为容器 `char* word[] = { "Hello", "World", "C++" };`
- 数组中的每个元素都是char *（指针），字符串存在常量区
- 相比二维字符数组，不需要预留固定长度空间



- 理解words, words[0], *(words+1), *words+1, *words[1], (*words) [1]分别是什么。

回顾——引用

- 变量的引用，相当于该变量的“别名”，操纵引用相当于操纵被引用的变量
- 语法格式：<类型> &<引用名> = <变量名>　(非const的左值引用)

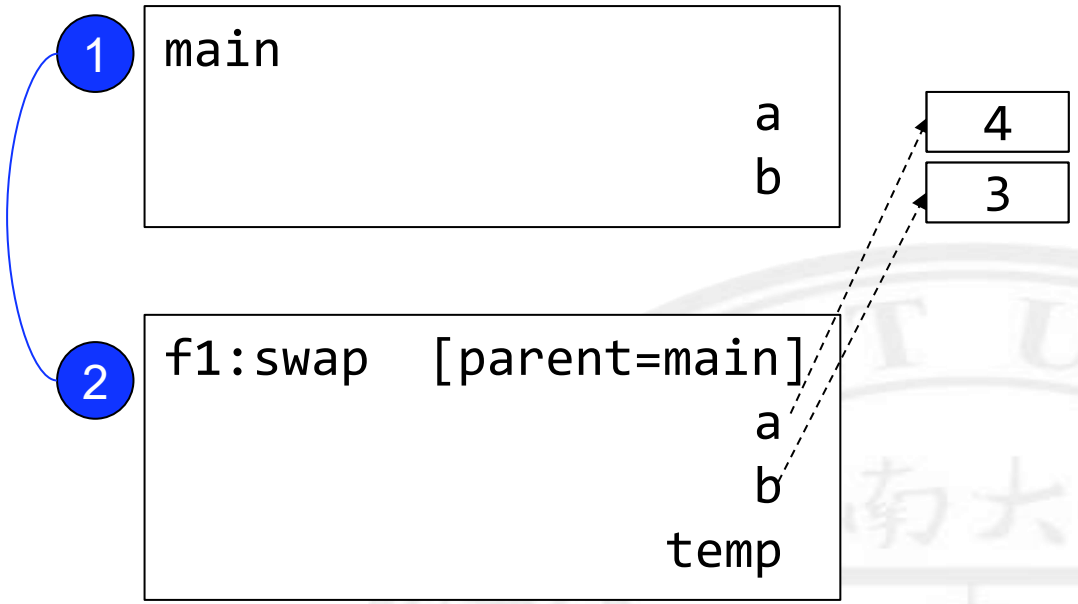
符号	*	&
类型标志	指针	（左值）引用
双目运算	乘	位与
单目运算	解引用	取地址

- 定义引用时一定要初始化，初始化之后不可引用其他变量，非const的左值引用对象必须是左值，不能是右值
- 常引用：定义引用时在类型前加const，表示不能通过引用修改变量

函数传引用

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
}
```



对比传值、传地址和传引用

传值

```
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;
}
```

传地址

```
void swap(int *pa, int *pb){
    int temp = *pa;
    *pa = *pb;
    *pb = temp;
    return;
}

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

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;
}
```

函数传引用的注意事项


```
void change(int &a){
    a = a + 1;
    return;
}
int main(){
    change(2);  ❌ 此时在做 int &a = 2
    double b = 2.0;
    change(b); ❌ 此时相当于 int &a = (int) b;
}
```

- 函数以“非 const 左值引用”作为形参时，不能绑定到临时量，也不能发生类型转换。

引用说明

- 通过上面的例子并非说明我们有了引用就不需要指针，只是在这样一个函数传参的场景下，引用更方便好用
- 引用和指针都有不可互相替代的使用场景：
  - 指针：数据结构中做动态内存分配
  - 引用：在运算符重载函数中返回值必须是引用，从而支持链式调用

```
string s = "hello";
s[1] = a;
```

全局变量

全局变量

- 定义在函数外面的变量是全局变量
- 全局变量具有全局的生存期和作用域
  - 它们与任何函数都无关
  - 在任何函数内部都可以使用它们

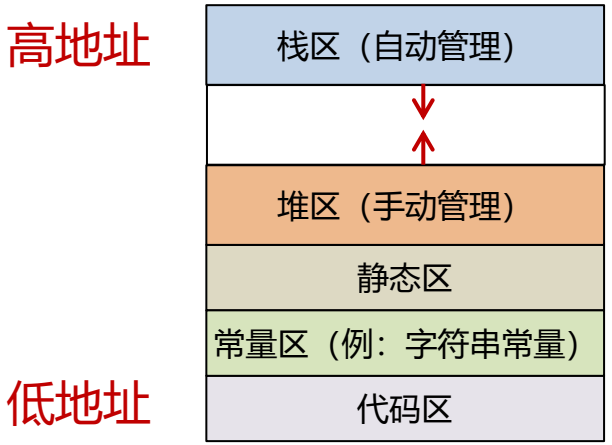
```
int global = 1;

void func(){
    cout << "in func: global=" << global;
}

int main(){
    cout << "in main: global=" << global;
    global += 1;
    func();
}
```

全局变量在内存中的位置

- 全局变量在内存的静态区，而局部变量在栈区

```
int global = 1;
int main() {
    int local1 = 1;
    int local2 = 2;
    cout << "global in memory: " << &global << endl; //0x4a7030
    cout << "local1 in memory: " << &local1 << endl; //0x6ffe4c
    cout << "local2 in memory: " << &local2 << endl; //0x6ffe48
}
```

- 内存模型



全局变量初始化

- 没有做初始化的全局变量会得到0值（区别于局部变量），如果是指针会得到NULL值

```
int global;

int main(){
    cout << "in main: global=" << global;
}
```

- 只能用编译时刻已知的值来初始化全局变量

```
//以下是全局变量

int global = 1;

int global2 = global;
```

❌ 编译时刻未知

同名隐藏

- 如果函数内部存在与全局变量同名的变量，则全局变量被隐藏（该性质在局部变量里讲过）

```
int global = 1;

int main(){
    int global = 2;
    cout << global; //2
}
```

- 慎重使用全局变量：增加代码阅读难度，各个函数都能访问修改，修改全局变量相关的代码很容易引发一系列问题

静态局部变量：具有全局的生存期

静态局部变量

- 在局部变量定义时加上static关键字就成为静态局部变量
- 静态局部变量的初始化只会在第一次进入这个函数时做，以后进入函数时会保持上次离开时的值

```
int global = 1;
void func(){
    static int x = 1;
    cout << "x in memory: " << &x << endl;
    cout << "x = " << x << endl;
    x++;
}

int main(){
    cout << "global in memory: " << &global << endl;
    func();
    func();
    func();
}
```

输出

```
global in memory: 0x472010
x in memory: 0x472014
x = 1
x in memory: 0x472014
x = 2
x in memory: 0x472014
x = 3
```

静态局部变量：特殊的全局变量

- 静态局部变量实际上是特殊的全局变量，和全局变量位于相同的内存区域
- 静态局部变量具有全局的生存期，局部作用域

变量类型	全局变量	静态局部变量	局部变量
内存区域	静态区	静态区	栈区
生存期	程序运行期	程序运行期	离开作用域
作用域	全局	局部	局部

应用场景：维护函数状态

- 为了防止盗号，常用措施是在输入密码超过k次之后，锁定账号
- 我们需要一个变量来记录“密码校验失败的次数”，这个变量应该：
  - 记录校验次数
  - 只被密码校验函数使用，不能被外部随意访问或修改
- 静态局部变量最合适

```
bool checkPassword(int input) {
    static int failCount = 0;   // 失败次数
    const int PASSWORD = 1234;

    if (failCount >= 3) {
        cout << "尝试次数过多，无法再尝试！" << endl;
        return false;
    }

    if (input == PASSWORD) {
        failCount = 0;          // 成功后清零
        return true;
    }
    else{
        failCount++;
        return false;
    }
}
```

宏

编译预处理指令

- #开头的是编译预处理指令，我们写的第一行代码就是：#include <头文件>
  - iostream：C++标准输入输出
  - iomanip：格式化输出
  - cmath：数学运算
  - cstdlib：通用工具（内存分配、退出、随机数等）
  - ctime：时间相关
  - cstring：C风格字符串的通用函数
- 更多C++标准库头文件：https://en.cppreference.com/w/cpp/headers.html

宏

- 宏（macro）：用一个名字代替一些重复内容
- #define用来定义一个宏
- 语法格式：#define <宏名> <替换内容>　不需要分号，宏名必须是一个词
- 原理：在编译之前由预处理器处理，做文本替换

```
#define PI 3.14159

int main(){
    double radius = 2.5;
    double area = PI * radius * radius;
    double perimeter = 2 * PI * radius;
}
```

替换

```
int main(){
    double radius = 2.5;
    double area = 3.14159 * radius * radius;
    double perimeter = 2 * 3.14159 * radius;
}
```

函数宏

- 宏可以带参数：

```
#define SQUARE(x) x * x

int main(){
    int a = SQUARE(3); // 3 * 3
    int b = SQUARE(1 + 2); // 1 + 2 * 1 + 2
    int c = 20 / SQUARE(3);  // 20 / 3 * 3
}
```

简单文本替换

- 正确写法是：`#define SQUARE(x) ((x)*(x))`
  - 整个被替换内容需要加括号
  - 参数出现的每个地方都要括号
- 现代编程规范不建议使用宏定义常量/函数

条件编译

- 宏主要用于：（1）条件编译（2）头文件保护

```
#define DEBUG
//在要调试的代码处加入下面的代码
#ifdef DEBUG
    cout << "[DEBUG] ..." << endl;
    cout << "[DEBUG] ..." << endl;
#endif
            ......
#ifdef DEBUG
    cout << "[DEBUG] ..." << endl;
    cout << "[DEBUG] ..." << endl;
#endif
```

```
#define _WIN32
//在和目标平台有关的代码处加入下面的代码
#ifdef _WIN32
    //Windows 代码
#endif
#ifdef __linux__
    //linux 代码
#endif
```

- 宏定义可以没有<替换内容>，只需要宏名即可，#ifdef是检查这个宏是否被定义过

说明

课程说明

- 由于本学期时间紧张，应该讲但没讲的内容：
  - 自定义类型：枚举、联合、typedef
  - 文件输入输出
  - 大型 C++ 程序：如何将多个源文件（.cpp） + 头文件（.h）构建（预处理/编译/汇编/链接）形成可执行文件

考试说明

- 笔试（2小时，所有程序代码设定在Win32目标平台下）
  - 15道单选（30分）
  - 4道程序阅读（20空，40分）
  - 3道程序填空（15空，30分）
- 机试（1小时，要求以本学期学的C++语言知识作答）
- 考试范围：本学期学的所有内容