计算机科学基础I

—— 指针I

东南大学 集成电路学院 朱彬武

E-mail: bwzhu@seu.edu.cn

回顾——数组初始化

- 列表初始化：
```
int mat[2][3] = {{2,3,5},      // {2,3,5,7,11,13}
                 {7,11,13}};
```
- 部分初始化：
```
int mat[2][3] = {{2},          // {{2,0,0}
                 {7,11}};          {7,11,0}}
```
- 全部元素初始化为0：
```
int mat[2][3] = {};
```
- 自动推断大小：
```
int mat[][3] = {{2,3,5},
                {7,11,13}};
```

多维数组只能省略最高维

注意：不要对数组名做赋值，也不要用数组名初始化其他数组

回顾——数组遍历


```
int mat[2][3] = {{2,3,5},
                 {7,11,13}};

for(int i = 0; i < sizeof(mat)/sizeof(mat[0]); i++){
    for(int j = 0; j < sizeof(mat[0])/sizeof(mat[0][0]); j++){
        cout << mat[i][j] << "\t";
    }
    cout << "\n";
}
```

- 数组遍历是数组各种操作的基础，依赖for循环，注意循环起点和终止条件
- 自动推断数组大小：sizeof运算符

回顾——数组作为函数参数

假设棋盘输入如下：

```
int board[3][3] = {{0, 1, 1},
                   {0, 0, 1},
                   {1, 0, 0}};
```

```
//检查正对角线
int checkMainDiag(int board[][3], int size) {
    int numOfX = 0, numOfO = 0;

    for (int i = 0; i < size; i++) {
        if (board[i][i] == 1) {
            numOfX++;
        } else if (board[i][i] == 0) {
            numOfO++;
        }
    }

    if (numOfX == size) return 1; // X 赢
    if (numOfO == size) return 0; // O 赢

    return -1;
}
```

遍历行和列


```
// 检查行
int checkRow(int board[][3], int size){
    for (int i = 0; i < size; i++) {
        int numOfX = 0, numOfO = 0;

        for (int j = 0; j < size; j++) {
            if (board[i][j] == 1) {
                numOfX++;
            } else if (board[i][j] == 0) {
                numOfO++;
            }
        }

        if (numOfX == size) {
            return 1;   // X 赢
        }
        if (numOfO == size) {
            return 0;   // O 赢
        }
    }

    return -1;
}
```

```
// 检查列
int checkCol(int board[][3], int size){
    for (int i = 0; i < size; i++) {
        int numOfX = 0, numOfO = 0;

        for (int j = 0; j < size; j++) {
            if (board[j][i] == 1) {
                numOfX++;
            } else if (board[j][i] == 0) {
                numOfO++;
            }
        }

        if (numOfX == size) {
            return 1;   // X 赢
        }
        if (numOfO == size) {
            return 0;   // O 赢
        }
    }
    return -1;
}
```

Q：两个函数能否合并起来，或者说能否用一个check函数同时完成行和列的检查？

二维数组转置


```
int checkLines(int board[][3], int size, bool checkRow) {
    for (int i = 0; i < size; i++) {
        int numOfX = 0, numOfO = 0;
        for (int j = 0; j < size; j++) {
            int v = checkRow ? board[i][j] : board[j][i];
            if (v == 1) numOfX++;
            else if (v == 0) numOfO++;
        }
        if (numOfX == size) return 1;
        if (numOfO == size) return 0;
    }
    return -1;
}
```

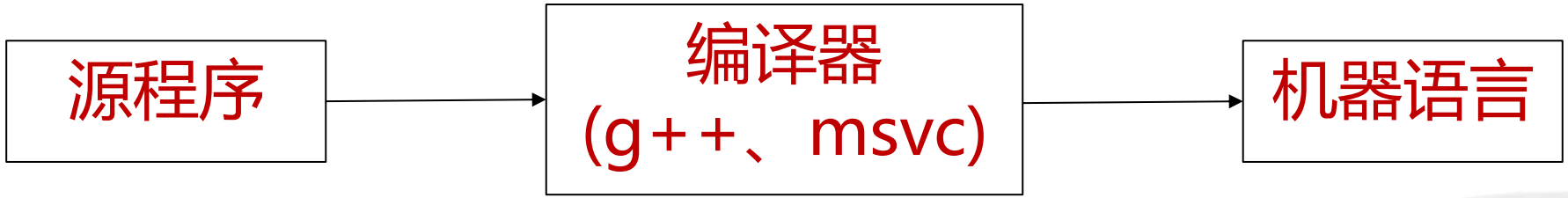
checkRow = true

checkRow = false

a[0][0]	a[0][1]	a[0][2]
a[1][0]	a[1][1]	a[1][2]
a[2][0]	a[2][1]	a[2][2]

内存与地址

C++执行方式：编译执行



C++的执行方式：借助编译器将源程序一次性转换为计算机能懂的语言（机器语言），然后这个机器语言写的程序就可以直接运行。

机器语言


```
//源程序
int a = 1;
int b = 2;
int c = a + b;
```

```
//汇编语言
mov DWORD PTR [rbp-4], 1
mov DWORD PTR [rbp-8], 2

mov eax, DWORD PTR [rbp-4]
add eax, DWORD PTR [rbp-8]
mov DWORD PTR [rbp-12]
```

地址

内存模型

	+3	+2	+1	+0
0x6dfe84	0x00	0x00	0x00	0x03
0x6dfe88	0x00	0x00	0x00	0x02
0x6dfe8c	0x00	0x00	0x00	0x01

- 每个内存地址对应 1 个字节
- 内存由多个连续的存储单元组成，每个单元都有唯一地址
- 不同类型的变量，占用的内存大小不同

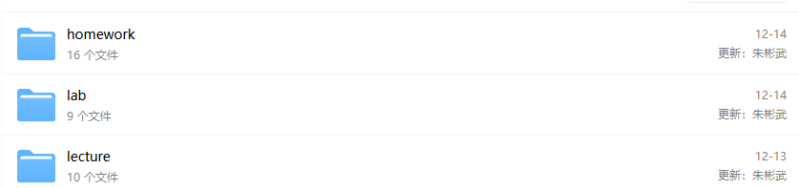
有了地址能做什么?


```
void swap(int a, int b);

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

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

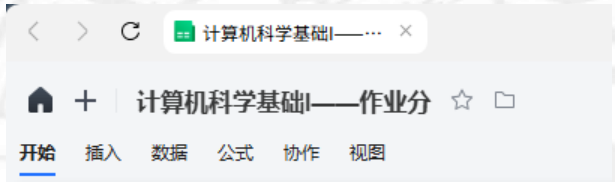
- 在操纵“地址”之前，访问变量只能通过变量名
- 函数调用采用的是传值，在很多情况下无法实现我们想要的效果，例如交换两个变量的值。
- 如果能够将取得的变量的地址传递给一个函数，情况是否有所不同?

打个比方：传值 vs 传地址

- 传值就像老师把课件传到群文件里，同学们各自下载一份到自己电脑，拿到的是一份拷贝
  - 每个人都有自己的副本
  - 小明在课件上做笔记，不影响其他人电脑里的文件



- 传地址就像老师首先建了一个云文档，然后把云文档链接发给大家，同学们操作的是同一份文档
  - 没有拷贝，只有一个原文档
  - 小明改了，别人立刻能看到



- 传地址的特点：不复制数据，效率高，但会直接影响原数据，存在风险

传地址版的swap函数


```
void swap(int *pA, int *pB);

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

void swap(int *pA, int *pB) {
    int temp = *pA;
    *pA = *pB;
    *pB = temp;
    return;
}
```

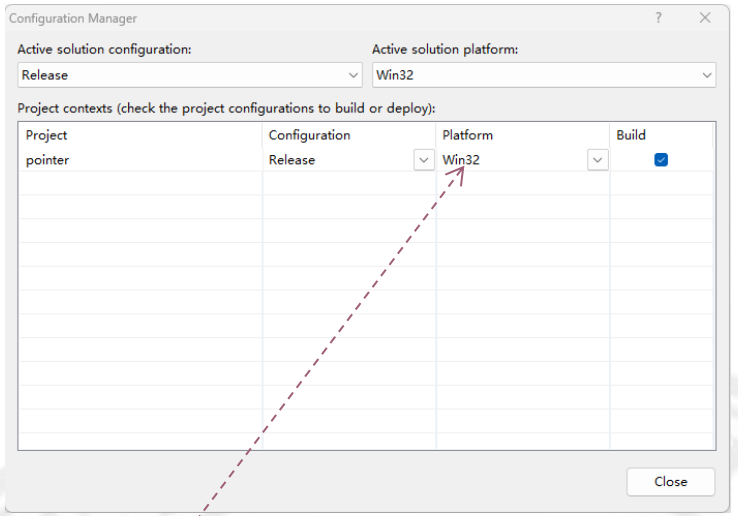
- 如何获得变量的地址?
- 如何保存变量的地址?
- 如何通过变量地址访问变量?

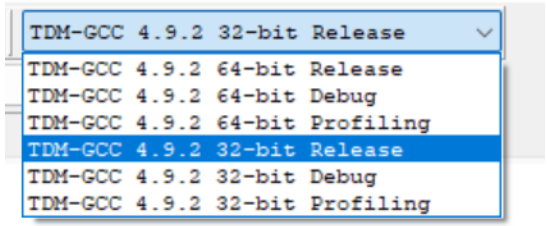
指针

取地址运算符&

- 取地址运算符： &，单目运算符，获得变量的地址

```
short a = 1;
int b = 2;
long long c = 3;
cout << &a << &b << &c << endl;
```





- 地址的大小由目标平台的位数决定：32 位平台通常是 4 字节，64 位平台通常是 8 字节。（Visual C++的编译平台固定是Win32，因此指针大小是4字节）

&的作用对象

- 取地址运算符 & 的操作数必须是左值，不能是右值
- 左值：表示一个在内存中有确定位置的对象（变量、数组元素等），可以取地址，可以作为赋值号左边的操作数。
- 右值：表示没有确定内存位置的临时值（字面量或大部分表达式计算结果），不能作为赋值号左边的操作数。

```
cout << &10 << endl;     // ❌ 字面量
cout << &(a+b) << endl; // ❌ 表达式结果
```

指针：保存地址的变量

- 保存地址的变量，这类变量称为指针

```
int score = 1;
int *pScore = &score;
```

- 语法格式：<类型> *<指针名>
- 指针命名规范：“p+被指对象含义”

指针变量的值是内存的地址

连续定义多个指针

- 给如下定义`int* p1, p2;` p2是什么?
  - `int`型变量
  - `int *p1, p2`同理
- 不要把`int*`理解成一种新的类型，一般说把`*p`看作一个`int`类型变量，因此p是一个指针
- 连续定义多个指针的正确写法如下：`int *p1, *p2;`

通过指针访问变量

- 解引用运算符： *，单目运算符，用来访问指针所存的地址上的变量，进而做读或写的操作
- 注意：和定义指针的*以及做乘法运算的*含义不同
- 可以做右值也可以做左值：既可以放在赋值号的左边，也可以放右边

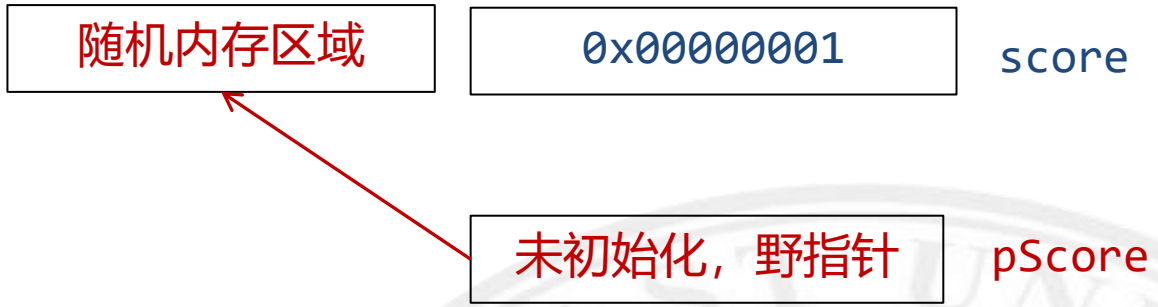
```
int score = 1;
int *pScore = &score;

int newScore = *pScore; //newScore = ?
*pScore = 2;

cout << score;  // score = ?
```

指针的类型，必须与它所指向对象的类型一致，否则就没办法做正确的解引用

野指针和空指针

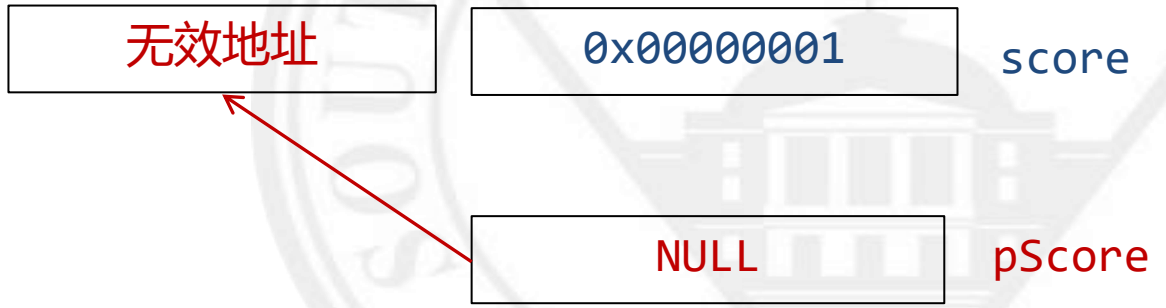
- 野指针：定义了指针变量，还没有初始化指向任何变量，就开始使用指针

```
int score = 1;
int *pScore;      //野指针

*pScore = 2;
```



- 空指针：指向一块无效地址，更安全，但是也不能访问无效地址内的数据

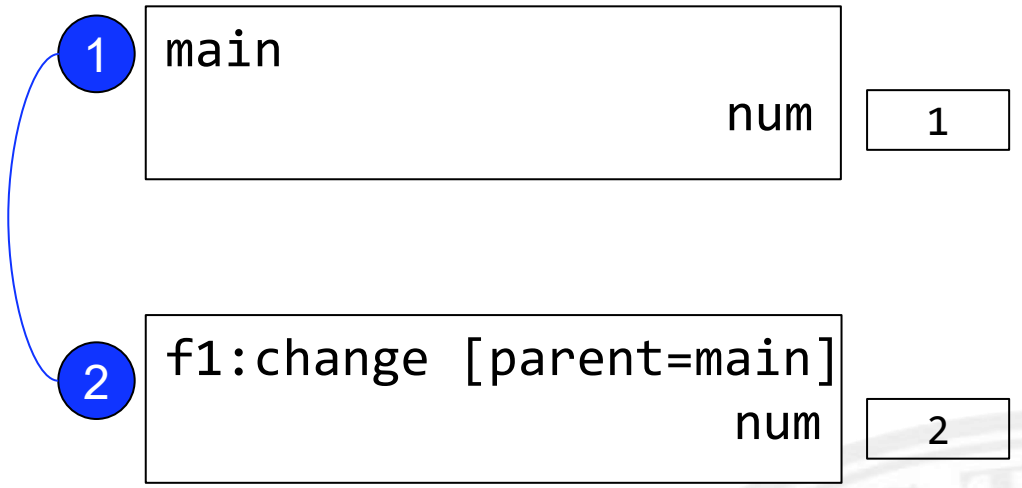
```
int *pScore = nullptr;   //C++11 空指针
int *pScore = 0;         //C++03 空指针
int *pScore = NULL;      //C++03 空指针

*pScore = 2; //❌ error
```



函数调用：传值 v.s. 传地址

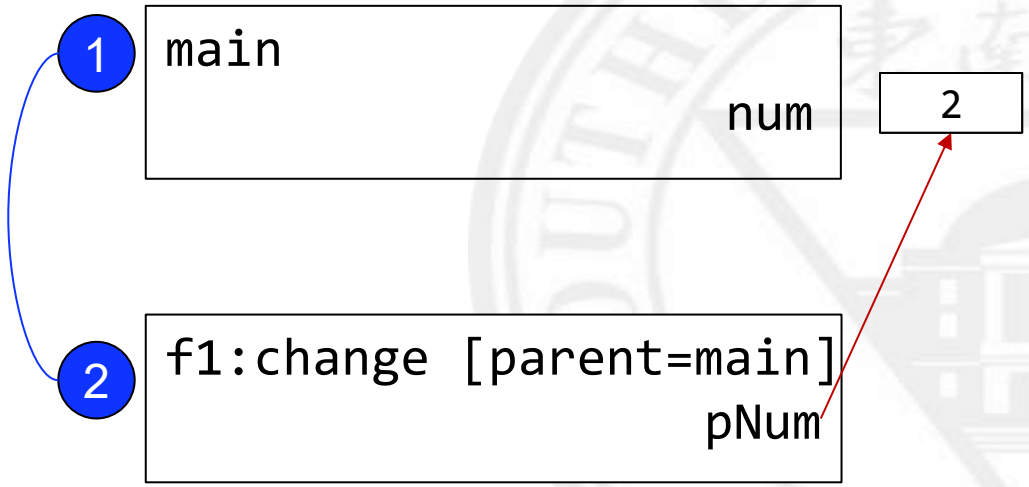
1 传值

```
void change(int num){
    num = 2;
}

int main(){
    int num = 1;
    change(num);
    cout << num << endl;
}
```



2 传地址

```
void change(int *pNum){
    *pNum = 2;
}

int main(){
    int num = 1;
    change(&num);
    cout << num << endl;
}
```



传地址版的swap函数解析


```
void swap(int *pA, int *pB);

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

void swap(int *pA, int *pB) {
    int temp = *pA;
    *pA = *pB;
    *pB = temp;
    return;
}
```

- 如何获得变量的地址? ✅
- 如何保存变量的地址? ✅
- 如何通过变量地址访问变量? ✅

为什么实现了变量交换?

因为函数里操作的是“变量本身的地址”，而不是值的拷贝。

指针的应用：用指针传递多个结果


```
void solve(int head, int foot, int* pChicken, int* pRabbit){
    for(int c = 0; c <= head; c++){
        int r = head - c;
        if(c * 2 + r * 4 == foot){
            *pChicken = c;
            *pRabbit  = r;
            return;
            }
        }
}

int main(){
    int head = 10;
    int foot = 28;
    int chicken, rabbit;

    solve(head, foot, &chicken, &rabbit);
    cout << "鸡: " << chicken << endl;
    cout << "兔: " << rabbit << endl;
    return 0;
}
```

- 原来采用传值的版本，函数只能通过 return 返回一个值，无法同时返回函数内计算得到的鸡和兔的数量。
- 通过指针参数，将计算结果写入 main 函数中的变量，从而实现把多个结果带回 main 函数。

数组名：特殊的指针

数组变量是特殊的指针


```
int arr[5] = {1, 2, 3, 4, 5};
cout << arr << endl;          //0x70fe30
cout << &arr[0] << endl;      //0x70fe30
cout << &arr[1] << endl;      //0x70fe34
```

- 数组名存储的是第一个元素的地址
- 数组变量本身表达地址，因此可以：`int *p = a;`
- 数组单元表达的是变量，因此需要用&取地址

[] 和*作用于数组名和指针


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

cout << *p << endl;      // 等价arr[0]
cout << p[0] << endl;    // 等价arr[0]
cout << p[1] << endl;    // 等价arr[1]
```

- 下标运算符[]可以对数组名做，也可以对指针做：p[0] == arr[0]
- *运算符可以对指针做，也可以对数组名做：*p == *arr

- 数组名是一个指针常量，不能被修改

```
int arr[5] = {1, 2, 3, 4, 5};
int arr2[5] = {0};

arr = arr2;
cout << arr[0];
```

数组传参==传地址


```
void traverse(int b[], int length) {
    for(int i = 0; i < length; i++){
        cout << b[i] << endl;
    }
}

int main() {
    int a[3] = {1, 2, 3};
    traverse(a, sizeof(a)/sizeof(a[0]));
}
```

参数表里的int b[]，实际上等价于int *b，这是一个指针变量，b存储的值是a[0]的地址

因此，传数组名就是在传地址。

右边这四种函数声明（原型）是等价的，后两种是省略了参数名的版本。

```
void traverse(int arr[], int length);

void traverse(int *arr, int length);

void traverse(int [], int);

void traverse(int *, int);
```

在数组中插入元素：insert函数


```
void insert(int a[], int len, int pos, int value)
{
    // 元素后移
    for (int i = len; i > pos; i--)
    {
        a[i] = a[i - 1];
    }

    // 插入新元素
    a[pos] = value;
}

int main(){
    int a[10] = {1, 2, 3, 4, 5};
    int value = 99, pos = 2;
    insert(a, 5, pos, value);

    cout << "in main function" << endl;
    for(int i = 0; i < 6; i++){
        cout << a[i] << " ";
    }
}
```

insert函数：将元素value插入到数组的pos处

由于传入的是数组地址，因此函数中通过该地址对数组元素进行的修改，会直接作用在主函数中的原数组上。