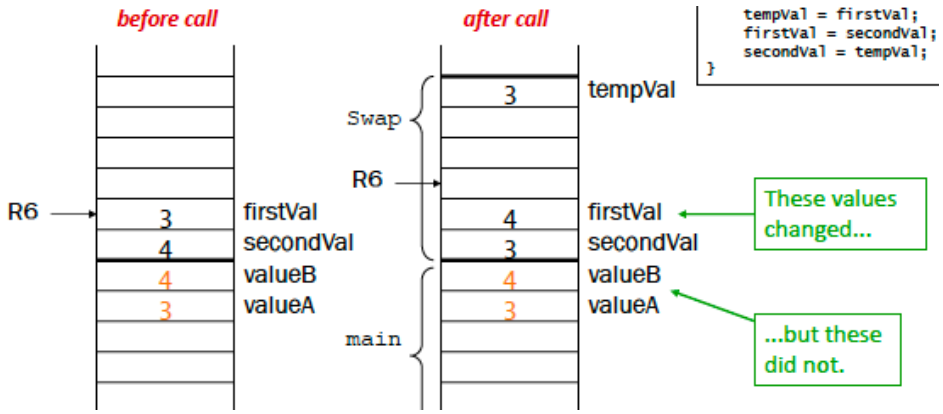
Lec 09 Pointers and Arrays

交换函数（Swap Function）

该函数展示了**值传递（Call by Value）**和**指针传递（Call by Reference）**的区别。

```
void Swap(int firstVal, int secondVal) {
    int tempVal = firstVal;
    firstVal = secondVal;
    secondVal = tempVal;
}
int main() {
    int valueA = 3, valueB = 4;
    Swap(valueA, valueB); // valueA仍是3，valueB仍是4
}
```

- 问题：
  - 使用值传递的Swap函数无法改变调用者的原始变量。
  - 原因：`firstVal`和`secondVal`是`valueA`和`valueB`的副本，函数内部的修改不会影响原始变量。
- 用运行栈的方式理解：在调用函数的时候，首先把`valueA`和`valueB`压入栈中，然后压入一系列书签变量，最后压入局部变量`firstVal`和`secondVal`和`tempVal`，而函数的操作仅仅是把`firstVal`和`seconVal`调换，而没有把这个操作传递到`valueA`和`valueB`中，因此实际上两者并没有交换。



- 解决方案：
  - 使用指针传递，通过操作变量的地址直接修改原始值。
  - 这里firstVal和secondVal通过解引用操作符`*`访问指针指向的实际值。

```c
void NewSwap(int *firstVal, int *secondVal) {
    int tempVal = *firstVal;
    *firstVal = *secondVal;
    *secondVal = tempVal;
}
int main() {
    int valueA = 3, valueB = 4;
    NewSwap(&valueA, &valueB); // valueA变为4，valueB变为3
}
```

C语言中的指针

指针是C语言的核心特性，用于操作内存地址。

- 声明：
  - 语法：`int *ptr;`（`ptr`是一个指向`int`类型的指针）。
  - 指针总是指向特定类型的数据，例如`int`、`double`、`char`等。
- 操作符：
  - `*ptr`（解引用操作符）：返回指针所指向的值，也可用于修改该值。
  - `&val`（取地址操作符）：返回变量val的内存地址。

LC-3中的指针操作

示例代码：

```c
int object = 4;
int *ptr = &object;
```

对应的LC-3代码：

```
AND R0, R0, #0   ; 清空R0
ADD R0, R0, #4   ; R0 = 4
STR R0, R5, #0   ; 将4存储到object的地址
ADD R0, R5, #0   ; R0 = object的地址
STR R0, R5, #-1  ; ptr = &object
```

解决交换问题

值传递 vs 指针传递：

- 值传递只修改副本，指针传递直接修改原始值。
- `NewSwap`的活动记录展示了堆栈如何存储指针参数`&valueA`和`&valueB`，以及局部变量`tempVal`。

更多关于指针的内容

- 空指针：

- ptr = NULL;表示指针不指向任何有效内存。
- 示例：

```
int *ptr = &valueA;
printf("%X\n", ptr); // 输出valueA的地址，如0x1B2F5F3C
ptr = NULL;
printf("%X\n", ptr); // 输出0
```

- **scanf中的'&'**：
  - `scanf("%d", &input);`使用`&`传递变量地址，以便`scanf`直接修改`input`的值。

指针的常见错误

- 类型不匹配：
  - 如`char *ptr = &val;`（`val`是`int`），可能导致数据访问错误。
- 未初始化指针：
  - 使用未初始化的指针（如int *ptr; *ptr = 5;）会导致未定义行为。
- 指针赋值错误：
  - ptr = 4;（直接赋值整数给指针）是错误的，正确应为ptr = &val;。

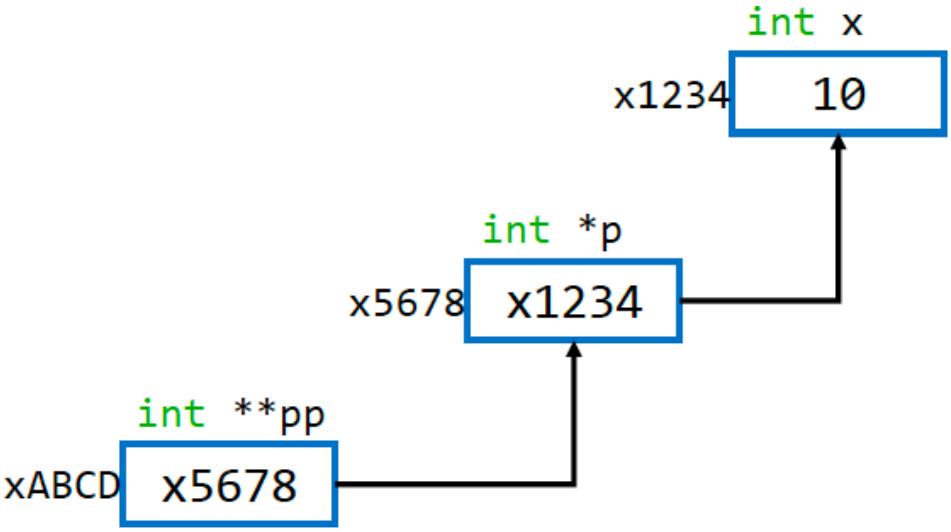
双重指针

声明和使用：

- `int **p;`声明一个指向指针的指针。
- 示例：

```
int x = 10;
int *p = &x;
int **pp = &p;
```

- 双重指针常用于动态分配二维数组或指针数组。



数组

数组是**存储多个同类型数据的结构**。

- 声明：
  - `type variable[num_elements];`
  - 例如：`int stdn[100];`声明一个包含100个整数的数组。
- 引用：
  - `variable[index];`访问第`index`个元素（索引从0开始）。
  - 注意：C语言不检查数组越界，需程序员确保索引有效。

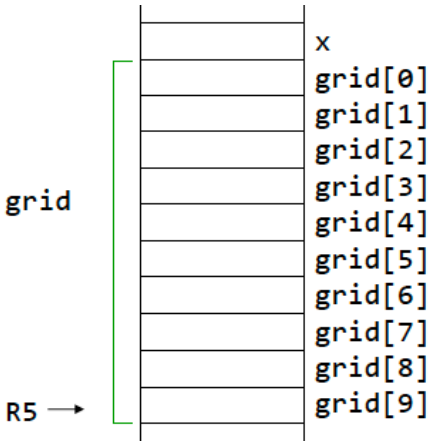
初始化和访问：

```
int grid[10] = {2, 10, 21, 3, 6, 1, 0, 9, 11, 12};
grid[6] = grid[3] + 1; // grid[6]从0变为4
for (int i = 0; i < 10; i++) {
    printf("%d\n", grid[i]);
}
```

LC-3中的数组引用

示例：`grid[x+1] = grid[x]+2`

```
LDR R0, R5, #-10   ; 加载x的值
ADD R1, R5, #-9    ; R1 = grid基地址
ADD R1, R0, R1     ; R1 = grid[x]地址
LDR R2, R1, #0     ; R2 = grid[x]
ADD R2, R2, #2     ; R2 = grid[x] + 2
ADD R0, R0, #1     ; R0 = x + 1
ADD R1, R5, #-9    ; R1 = grid基地址
ADD R1, R0, R1     ; R1 = grid[x+1]地址
STR R2, R1, #0     ; grid[x+1] = R2
```



指针和数组的关系

- 数组名：

    - 数组名（如word）是指向数组第一个元素的指针。
    - 示例：`char word[10]; char *cptr = word;`（`cptr`指向`word[0]`）。
- 区别：
    - 数组名是常量，不能重新赋值（如`word = word + 1;`会报错）。
    - 指针是变量，可以改变指向（如`cptr = cptr + 1;`）。
- 等价表达式：

```
char word[10];
char *cptr = word;
```

`cptr`≡`word`≡`&word[0]`

`cptr+n`≡`word+n`≡`&word[n]`

`*cptr`≡`*word`≡`word[0]`

`*(cptr + n)`≡`*(word + n)`≡`word[n]`

详解指针概念

1. 指针是什么?

在C语言中，**指针（Pointer）** 是一个特殊的变量，它不像普通的变量那样直接存储数据（比如数字或字符），而是存储另一个变量的**内存地址**。简单来说，指针就像一个“地址标签”，告诉你某个数据藏在内存的哪个位置。

- **比喻**：想象内存是一排带编号的邮箱，每个邮箱里装着东西（数据）。指针就像一张纸条，上面写着某个邮箱的编号，告诉你去哪里找东西。
- **例子**：假如你有一个变量`x = 10`，它被存储在内存的某个地址，比如`0x1000`。一个指针可以存储这个地址`0x1000`，让你通过它找到`x`。

2. 怎么创建和使用指针?

让我们通过代码一步步看指针是怎么工作的。

（1）声明指针

在C语言中，声明一个指针需要告诉它会指向什么类型的数据。比如：

```
int *ptr; // 这是一个指向整数(int)的指针
```

- `int`表示指针指向的数据是整数类型。
- `*`表示这是一个指针变量。
- `ptr`是这个指针的名字。

（2）初始化指针

声明完指针后，它需要指向某个具体的地址。我们可以用`&`操作符来获取一个变量的地址：

```
int x = 10;       // x 是一个普通变量，值是 10
int *ptr = &x;    // ptr 是一个指针，存储 x 的地址
```

- `&x` 表示“`x` 的地址”。
- 现在，`ptr` 里面存的不是 `10`，而是 `x` 在内存中的地址（比如 `0x1000`）。

（3）使用指针访问数据

指针有了地址后，我们可以通过它来“访问”或“修改”那个地址里的数据。这叫做**解引用（Dereferencing）**，用 * 操作符完成：

```
int value = *ptr; // *ptr 取出 ptr 指向的地址里的值，也就是 10
```

- `*ptr` 的意思是“去 `ptr` 指向的地址，取出那里的值”。
- 这里，`value` 会得到 `10`，因为 `ptr` 指向 `x`，而 `x` 的值是 `10`。

（4）通过指针修改数据

不仅可以读取，还可以用指针改变原来的值：

```
*ptr = 20;  // 把 ptr 指向的地址里的值改为 20
```

- 这会直接把 `x` 的值从 `10` 改成 `20`，因为 `ptr` 指向的就是 `x` 的地址。

3. 指针和普通变量的区别

我们对比一下：

- 普通变量：`int x = 10;`
  - x 直接存了 10 这个值。
- 指针：`int *ptr = &x;`
  - `ptr` 存的是 `x` 的地址（比如 `0x1000`），而不是 `10`。
  - 通过 `*ptr`，你可以间接访问或修改 `x` 的值。

关键点：指针不存数据本身，它存的是“数据在哪里”。

4. 一个简单的例子

让我们用一个完整的程序来看看：

```
#include <stdio.h>

int main() {
    int x = 10;       // 定义一个变量 x，值为 10
    int *ptr = &x;    // 定义一个指针 ptr，指向 x 的地址

    printf("x 的值：%d\n", x);          // 输出：10
```

```
    printf("ptr 存的地址：%p\n", ptr);   // 输出：x 的地址（比如 0x1000）
    printf("ptr 指向的值：%d\n", *ptr);  // 输出：10

    *ptr = 20;          // 通过指针修改 x 的值
    printf("修改后 x 的值：%d\n", x);    // 输出：20

    return 0;
}
```

- `ptr` 存的是 `x` 的地址。
- `*ptr` 能拿到或改变 `x` 的值。

5. 为什么需要指针?

你可能会问："直接用变量不就好了，为什么要用指针？"其实，指针在C语言中有很多厉害的用途：

- **修改函数外的变量**：比如两个数交换值的函数，通过指针可以直接改原来的数。
- **管理内存**：像 `malloc` 这样的函数会返回指针，让你动态分配内存。
- **处理数组**：数组名实际上是一个指针，指向数组的第一个元素。
- **构建复杂结构**：比如链表、树这些数据结构，靠指针把各个部分连起来。