More on PA4
Multi-dimensional arrays

CSE 30: Computer Organization and Systems Programming

Diba Mirza
Dept. of Computer Science and Engineering
University of California, San Diego
void foo ( ) {
    char arr[3];
    int j=0;
    for (j=0; j<=3; j++)
        *(arr+j)=0;
}

Do you think the program would still get stuck in an infinite loop if arr was a char array?
Assume data is aligned and byte ordering is little endian.
A. Yes
B. No
void foo ( ) {
    char arr[4];
    int j=0;
    for (j=0; j<4; j++)
        *(arr+j)=0;
}

Do you think the program would still get stuck in an infinite loop for the new code? 
Assume data is aligned, byte ordering is Little Endian.
Think about why or why not
A. Yes 
B. No
int majority_count(int * arr, int len, int * result) {
    if(len == 0) {
        return 0;
    }
    if(len == 1) {
        if(result) {*result = arr[0];}
        return 1;
    }

    int left_majority, right_majority, c;
    int left_majority_count = majority_count(arr, len/2, &left_majority);
    int right_majority_count = majority_count(arr+len/2, len-len/2, &right_majority);

    if(left_majority_count) {
        c = count(arr, len, left_majority);
        if(c > len/2) {
            if(result) {*result = left_majority;}
            return c;
        }
    }
    if(right_majority_count) {
        c = count(arr, len, right_majority);
        if(c > len/2) {
            if(result) {*result = right_majority;}
            return c;
        }
    }
    return 0;
Which of the following statements correctly creates the local variables: left_majority, right_majority and c on the stack?

A. SUB sp, sp, #16
B. SUB sp, sp, #12
C. push {r4-r6}
D. STMDB sp!, {r4-r6}
E. All of the above
Assume the value of the variables arr and len are available in r4 and r5, respectively.
How do we make the first call to majority_count?
Multidimensional arrays: 2D

- Declaration

```c
int a[3][4];  /*Conceptually 2D matrix with 3 rows 4 columns */
```

- Element in row i, column j is retrieved as `a[i][j]`
- ‘a’ is a pointer to an integer array of size 4
In memory the elements of ‘arr’ are stored in a contiguous memory block.
(arr+i) increments the address of ‘arr’ by how many bytes?

A. \( i \times \text{sizeof(int)} \)

B. \( i \times \text{sizeof(int*)} \)

C. \( i \times \text{sizeof(int)} \times \text{number of columns} \)
Express \( arr[i][j] \) using ‘arr’ as a pointer

A. \*((arr+i) +j)
B. \*(*(arr+i)+j)
C. \*((arr+4*i*sizeof(int)+j*sizeof(int)))
D. None of the above
Storing into a multi-dim array

- int arr[3][4];
- Assume the base address (arr) is available in r0
- Fill in the blank to get ARM code equivalent to arr[1][2]=12;
  A. STR r3, [r0, #3]  
  B. STR r3, [r0, #12]  
  C. STR r3, [r0, #24]  
  D. None of the above

MOV r3, #12

---

MOV r3, #12
Storing into a multi-dim array

- int arr[3][4];
- Assume the mapping:
  arr: r0, i: r1, j: r2
- Fill in the blanks to get ARM code equivalent to
  arr[i][j]=12;

A. ADD r4, r2, r1, LSL #4
B. ADD r4, r2, r1, LSL #2
C. ADD r4, r1, r2, LSL #4
D. ADD r4, r1, r2, LSL #2
E. None of the above
Multi-level arrays

- Declaration
  ```c
  char name_1[] = "John";
  char name_2[] = "Paul";
  char name_3[] = "Rose";
  char * names[] = {name_1, name_2, name_3};
  ```
- How is "names" represented in memory?
Multi-level arrays:
Representation in memory

What is the output of: `printf("%c\n", names[1][2]);`

A. h
B. a
C. u
D. o
E. s

```
char name_1[]="John";
char name_2[]="Paul";
char name_3[]="Rose";
char * names[]={name_1, name_2, name_3};
```
Multi-level vs multi-dimensional arrays

1. What does names[1][2] give in each case?
2. Which one needs more memory accesses?
3. When would we prefer multi-level arrays?