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
Explaining weird program behavior

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
```c
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?

```
mov    r0, r4
lsr    r1, r5, #1  @  r1 = len/2
mov    r2, sp
bl     majority_count
str    r0, [sp, #12] @ left_majority_count = r0
```
### Multidimensional arrays: 2D

#### Declaration

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

<table>
<thead>
<tr>
<th>Row 0</th>
<th>Row 1</th>
<th>Row 2</th>
</tr>
</thead>
<tbody>
<tr>
<td>( a[0][0] )</td>
<td>( a[1][0] )</td>
<td>( a[2][0] )</td>
</tr>
<tr>
<td>( a[0][1] )</td>
<td>( a[1][1] )</td>
<td>( a[2][1] )</td>
</tr>
<tr>
<td>( a[0][2] )</td>
<td>( a[1][2] )</td>
<td>( a[2][2] )</td>
</tr>
<tr>
<td>( a[0][3] )</td>
<td>( a[1][3] )</td>
<td>( a[2][3] )</td>
</tr>
</tbody>
</table>

- **Element in row \( i \), column \( j \) is retrieved as \( a[i][j] \)**
- ‘\( a \)’ is a pointer to an integer array of size 4
int arr[3][4];  /*Conceptually 2D matrix with 3 rows 4 columns */

In memory the elements of ‘arr’ are stored in a contiguous memory block

\[
\begin{array}{cccc}
[0,0] & [0,1] & [0,2] & [0,3] \\
[1,0] & [1,1] & [1,2] & [1,3] \\
\end{array}
\]

Row 0 elements  Row 1 elements  Row 2 elements
(arr+i) increments the address of ‘arr’ by how many bytes?

A. i*sizeof(int)

B. i*sizeof(int*)

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

\[
arr
\]

\[
\begin{array}{ccccccccccc}
\end{array}
\]

A. \( *((arr+i) + j) \)
B. \( *((arr+i)+j) \)
C. \( *(arr+4*i*sizeof(int)+j*sizeof(int)) \)
D. None of the above