Multi–dimensional arrays
Multi–level arrays
Complex declarations in C

CSE 30: Computer Organization and Systems Programming

Diba Mirza
Dept. of Computer Science and Engineering
University of California, San Diego
Announcements

- Last quiz, due Sunday at 11:59pm
- Final review, Sun Dec 6th from 11am to 12:20pm
  WLH 2001
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

These are the options for storing into a multi-dimensional array in ARM assembly language.
Storing into a multi–dim array

- int arr[3][4];
- Assume the mapping: $r_4 = i \times 4 + j$
  arr: r0, i: r1, j: r2
  $r_4 = r_1 \times 4 + r_2$
- 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

MOV r3, #12

STR r3, [r0, r4, LSL #2]
Initializing two dimensional arrays

```c
char names[][] = {
{‘J’, ‘o’, ‘h’, ‘n’, ‘\0’},
{‘P’, ‘a’, ‘u’, ‘l’, ‘\0’},
{‘R’, ‘o’, ‘s’, ‘e’, ‘\0’}
};
```

How is “names” represented in memory?

```c
names + 1
```

\[
*(names + 1) = ‘P’
\]
Multi-level arrays

- Declaration
  ```
  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:

```c
printf("\%c\n", names[1][2]);
```

```c
char name_1[] = "John";
char name_2[] = "Paul";
char name_3[] = "Rose";
char * names[] = {name_1, name_2, name_3};
```
Multi-level arrays (ARM code)

Write ARM code to put names[1][2] in register r4:

\[
\text{names}[1][2] = *(\text{names} + 1) + 2 \\
= *(\text{ro} + 4) + 2
\]

- LDR r1, [ro, #4] @ r1 = 0x60
- LDRB r4, [r1, #2]

```
0x80 names
0x60
0x90
0x60
```

```
\text{'P'} \text{'a'} \text{'u'} \text{'l'} \text{'\0'}
\text{'j'} \text{'o'} \text{'h'} \text{'n'} \text{'\0'}
\text{'R'} \text{'o'} \text{'s'} \text{'e'} \text{'\0'}
```
Multi-level arrays

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

A. John
B. Paul
C. Rose
D. 0x60
E. None of the above

```c
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

To get names[1][2] more memory accesses are needed in which case?

A. Multilevel array “names”

B. Multi-dimensional array “names”

C. Same in both cases
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?
Complex declarations in C

How do we decipher declarations of this sort?

```
int *(*arr)[];
```

Read

* as “pointer to” (always on the left of identifier)
[] as “array of” (always to the right of identifier)
( ) as “function returning” (always to the right …)

Ref: Rick Ord
http://ieng9.ucsd.edu/~cs30x/rt_lt.rule.html
Complex declarations in C

Right-Left Rule

int *arr [];

Step 1: Find the identifier

Step 2: Look at the symbols to the right of the identifier. Continue right until you run out of symbols *OR* hit a *right* parenthesis "")"

Step 3: Look at the symbol to the left of the identifier. If it is not one of the symbols ‘*’, ‘()’, ‘[]’ just say it. Otherwise, translate it into English using the table in the previous slide. Keep going left until you run out of symbols *OR* hit a *left* parenthesis "(".

Repeat steps 2 and 3 until you've formed your declaration.
Complex declarations in C

Illegal combinations include:

[]() - cannot have an array of functions
()() - cannot have a function that returns a function
()[] - cannot have a function that returns an array
Complex declarations in C

int i;
int *i;
int a[10];
int f();
int **p;
int (*p)[];
int (*fp)();
int *p[];
int af[]();
int *f();
int fa[]();
int ff();

Ref: Rick Ord  http://ieng9.ucsd.edu/~cs30x/rt_lt.rule.html