CSE 30: Computer Organization and Systems Programming

Lecture 14: Midterm review

Diba Mirza
University of California, San Diego
Q2: More fun with pointers
For the C code below make the following assumptions
   i. The base address of array is 0x10000000
   ii. The byte ordering is Little Endian
   iii. int is 4 bytes

```
int array[] = {-1, 1, 2, 3, 4, 5};
int main()
    char *ptr1 = (((char *)array) + 5);
    char val;
    int *ptr2 = array + 3;
    val = *ptr1;
    return 0;
```

- Show the byte level representation of ‘array’
- What is the value of *ptr2 and val before main returns?
Compile by hand

- \( g = h + A[8] \)

\[ A : r0 \]
\[ h : r1 \]
\[ g : r2 \]
void foo (int *p, int size) {
    *p = 0;
    *(p + 1) = 0;
    *(p + size - 1) = 0;
}

In ARM:

    mov r2, #0
    str r2, [r0] @ *p
    mov r3, #4 @ size
    str r2, [r0, r3] @ *(p + 1)
    sub r1, r1, #1 @ r1 = size - 1
    mul r4, r1, r3 @ r4 = (size - 1) * 4
    sir r2, [r0, r4] @ * (p + size - 1)
How would the code change if int was replaced by short?

```
void swap (int *x, int *y){
  int tmp = *x;
  *x = *y;  <-- Need to break this up into two instructions
  *y = tmp;
}
```

In ARM:

```asm
LDR  r2, [r0]  @  r2 = *x
LDR  r3, [r1]  @  r3 = *y
STR  r3 , [r0]  @  *x = r3  effectively  *x = *y
STR  r2 , [r1]  @  *y = r2  or  *y = tmp
```
Q: Assume the linked list has already been created, what do the following expressions evaluate to?

1. head->value
   - A. 10
   - B. 5
   - C. 15
   - D. NULL
   - E. Run time error

2. head->next->value
   - A. 10
   - B. 5
   - C. 15
   - D. NULL
   - E. Run time error

3. head->next->next->value
   - A. 10
   - B. 5
   - C. 15
   - D. NULL
   - E. Run time error

4. head->next->next->next->value
   - A. 10
   - B. 5
   - C. 15
   - D. NULL
   - E. Run time error
struct Node {
    int value;
    struct Node *next;
};

Q: Write ARM Assembly for each of the following. Assume head is in r0
- head->value=100;
- head->next->value=100;

ARM

MOV r1, #100
STR r1, [r0] @ head->value = 100

LDR r2, [r0, #4] @ r2 = head->next
STR r1, [r2] @ *r2 = r1, effectively head->next->value = 100;
Let’s look at an example of using structures, pointers, `malloc()`, and `free()` to implement a linked list.

typedef struct Node node;

struct Node {
    int *int;
    _____ next;
};

Q: What should be the data type of the variable ‘next’?

A. struct Node
B. Node
C. node
D. node *
Adding a node to the list

node *list_add(node* head, int new_value)
{
    /* Add a new node on the head of the list,
       Populate it with new_value and return the new head */
    node * new-node = (node *) malloc (sizeof(node));
    if (new-node) { /* Return if malloc was not successful */
        new-node->value = new_value
        /* Complete the rest to rewrite the linked list */
        return new-node;
    } else
        return NULL;
}
What should Line 5 be to achieve the pointer diagram (below)?

A. `new_node->next = head;`
B. `next = head;`
C. `head = new_node;`
D. `new_node->next = *head;`
node *list_add(node* head, int new_value)
{
    node *new_node=(node*) malloc(sizeof(node));
    if (!new_node)
        return NULL;
    new_node->value = new_value;
    new_node->next = head;
    return ______________;
}