CSE 30: Computer Organization and Systems Programming

Lecture 9: Functions and pointers
Dynamic memory allocation

Diba Mirza
University of California, San Diego
Consider the following function

```c
void swap(int x, int y); //Declaration

void swap(int x, int y) {
    int tmp;
    tmp = x;
    x = y;
    y = tmp;
}
```
int main()
{
    int a=10, b=20;
    swap(a, b);
}

void swap(int x, int y) {
    int tmp;
    tmp = x;
    x = y;
    y = tmp;
}

Q: Which of the following changes would you make to interchange the values in ‘a’ and ‘b’ when swap is called?

A. In swap, return the values of ‘x’ and ‘y’ to the main function after swapping them
B. Declare ‘a’ and ‘b’ as global variables, so that they become accessible to the swap routine
C. Pass the address of ‘a’ and ‘b’ to swap instead of their value and modify the swap function
D. Move the implementation in swap to the main function
Functions: Pass by reference

```c
void swap(int *x, int *y) {
}
```

Q: What should the modified swap function do?

A. Swap the values in ‘x’ and ‘y’

B. Swap the values pointed to by ‘x’ and ‘y’

C. Both the above operations are equivalent
Q: What happens when `IncrementPtr(q)` is called in the following code:

```c
void IncrementPtr(int *p){
    p = p + 1;
}

int A[3] = {50, 60, 70};
int *q = A;
IncrementPtr(q);
```

A. ‘q’ points to the next element in the array with value 60
B. ‘q’ points to the first element in the array with value 50
Q: How should we implement Increment_Ptr(), so that ‘q’ points to 60 when the following code executes?

```c
void Increment_Ptr(int **p){
    p = p + 1;
}
```

1. `int A[3] = {50, 60, 70};`
2. `int *q = A;`
3. `Increment_Ptr(&q);`

A. `p = p + 1;`
B. `&p = &p + 1;`
C. `*p= *p + 1;`
D. `p= &p+1;`
Dynamic memory allocation

```c
void* malloc (size_t s);
```

- Allocates a block of `s` bytes on the heap and returns the starting address of the block
- If memory cannot be successfully allocated, returns a NULL.

/* Allocate memory for an integer array of size 5 */
Dynamic memory allocation

int * arr1 = malloc (8);
int arr2[2];

What is the difference between the two statements above, if both appear within function scope?
Dynamic memory allocation

```c
void* calloc (size_t n, size_t s);
```

- Allocates a block of memory on the heap to store ‘n’ elements, each of ‘s’ bytes and returns the starting address of the block.
- If memory cannot be successfully allocated, returns a NULL
- Initializes memory to zero (not guaranteed)

`/* Allocate memory for an integer array of size 5 */`
Dynamic memory allocation

```c
void* realloc (void * p, size_t s);
```

- Resize an existing block of memory pointed to by `p` to a total size of `s` bytes.
- Returns NULL on failure
- Are the contents copied over to the new block?
Dynamic memory allocation

void free(void * ptr);

• Frees the heap block pointed to by p
• What happens in the following code?

int *p=malloc(8);
free(p);
free(p);
free(p);