

**Additional Explanations – Lecture 6**

**Dynamic Memory – Lecture 6**

```

int factorial(int x)
{
    /**** A ****/
    if (x == 0)
    {
        return 1;
    }
    return x * factorial(x-1);
}

void main()
{
    int n = 3;
    printf ("%d", factorial(n));
    /**** B ****/
    return;
}

```

Stack /\*\*\*\* A \*\*\*\*/ first time:

3	<b>x</b> ( <i>parameter</i> )
3	<b>n</b>

Stack /\*\*\*\* A \*\*\*\*/ second time:

2	X <i>second time (parameter)</i>
3	X <i>first time (parameter)</i>
3	<b>n</b>

Stack /\*\*\*\* A \*\*\*\*/ third time:

1	X <i>third time (parameter)</i>
2	X <i>second time (parameter)</i>
3	X <i>first time (parameter)</i>
3	<b>n</b>

Stack /\*\*\*\* B \*\*\*\*/

3	<b>n</b>
---	----------

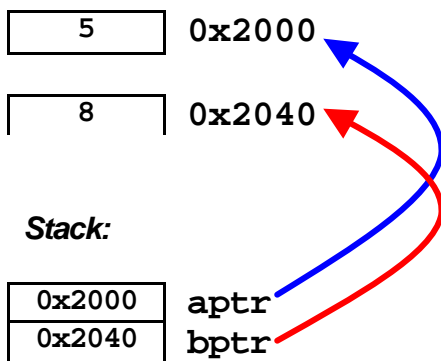
## CSE1303 Part A – Data Structures and Algorithms

**Note:** Parameters go on the stack, and dynamically allocated variables (using malloc) go in the heap.

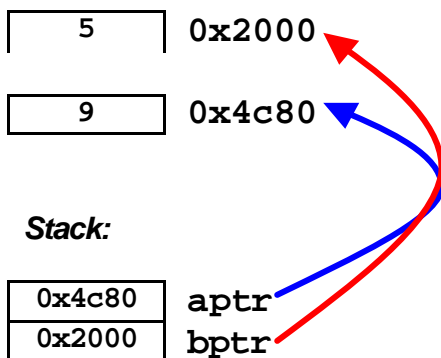
```
int *aptr = NULL;
int *bptr = NULL;
```

NULL	aptr
NULL	bptr

```
aptr = (int*)malloc(sizeof(int));
*aptr = 5;
bptr = (int*)malloc(sizeof(int));
*bptr = 8;
```

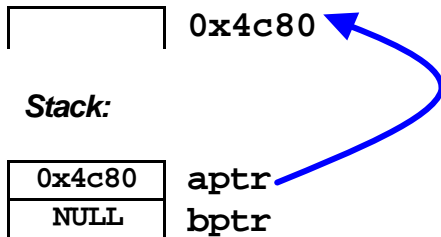


```
free(bptr);
bptr = aptr;
aptr = (int*)malloc(sizeof(int));
*aptr = 9;
```



*Be careful of pointing to freed memory:*

```
bptr = aptr;
free(bptr);
bptr = NULL;
```

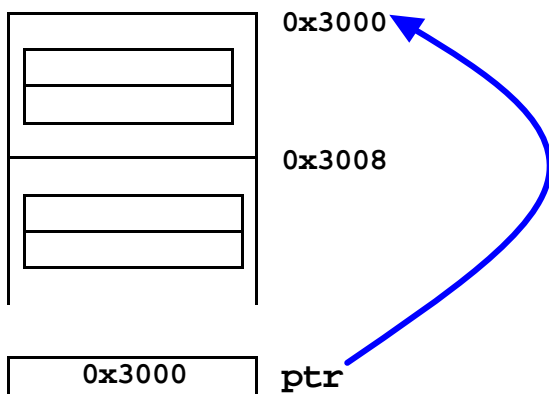


**Array dynamic allocation:**

Memory is allocated in a block, so you can use pointer arithmetic to step through to the next element. When you create a variable by `Item array[5]`, the name of the array is the address of the first item. When dynamically allocating space in the heap for the array, `malloc` returns the address of the first item.

```
struct ItemRec
{
    float price;
    int ID;
};
typedef struct ItemRec Item;

Item* ptr;
ptr = (Item*)malloc(2 * sizeof(Item))
```



**Note:** to get to a member of the structure through a pointer you need to use the `->` operator:

```
ptr->ID = 5;
ptr->price = 10.50;
```