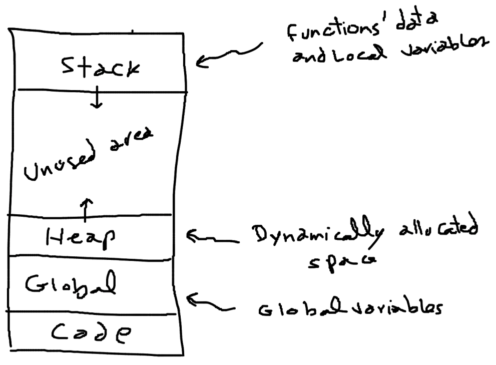
Application Memory

- There are different sections of memory for any program:
- When a function is called (executed), all its local variables will reside in the stack (pushed into the stack), then local variables go away when the function ends.
- Only data allocated using malloc, stays in the heap and must be freed manually.


Stack
Unused area
Heap
Global
Code
Functions' data and Local variables
Dynamically allocated space
Global variables

Dynamic Memory Allocation

- In C, dynamic memory is allocated from the heap using some standard library (stdlib) functions; such as: malloc() and free().
- The malloc() function takes a single parameter that is the size of the requested memory area in bytes and returns a pointer to the allocated memory. If the allocation fails, it returns NULL.

```
void *malloc(size_t size);
```

- The function free() is used to deallocate the memory by taking the pointer returned by malloc().
- Sample code:

```
int *p;
p = malloc(10 * sizeof(int));
// The same memory can be deallocated using free(p).
```

Dynamic Arrays:

- Arrays can be static arrays (stored in the stack) or dynamic arrays created dynamically using malloc (stored in the heap) with a fixed size that can be modified. For example:

```
//static array - has fixed size
//in the stack
//track the actual size
int a[10]={0};

//dynamic array - has fixed size
// in the heap
//track the actual size
 int* b= malloc(sizeof(int) * 10);

//dynamic array - has modified size
//in the heap
//track the actual size and the capacity
 int capacity=1;
int actual_size=0;
int* c=malloc(sizeof(int)*1);
```

- With all arrays in c, you have to track the actual number of items stored in the array. And you need to track the actual size and the capacity for dynamic arrays.

- With dynamic arrays, it is strongly recommended to create a function to add items that can extend the array size if needed. This can be done by reallocating the space dedicated for the array in the heap.

```
// add item function
void additem(int* arr, int* capacity, int* actual_size, int value ){
   if(*capacity==*actual_size){
      (*capacity)*=2;           //increase the capacity
      arr=(int*) realloc(arr,sizeof(int)* *capacity);
   }
   arr[*actual_size]=value;
   (*actual_size)++;
}

int* c=(int*) malloc(sizeof(int)*1);
int cap=1;
int size=0
additem(c,&cap,&size, 100);
```