Introduction to Dynamic Memory Allocation

The `malloc` statement will return a pointer to a certain amount of memory. If there is no memory available, it will return NULL. Make sure to allocate the correct amount of memory that you need for your variable.

To let go of a section of memory when you're done using it, us the `free` statement. Not freeing memory will lead to memory leaks. (See the Valgrind tutorial for how to find memory leaks.) This should be done any time you use malloc.

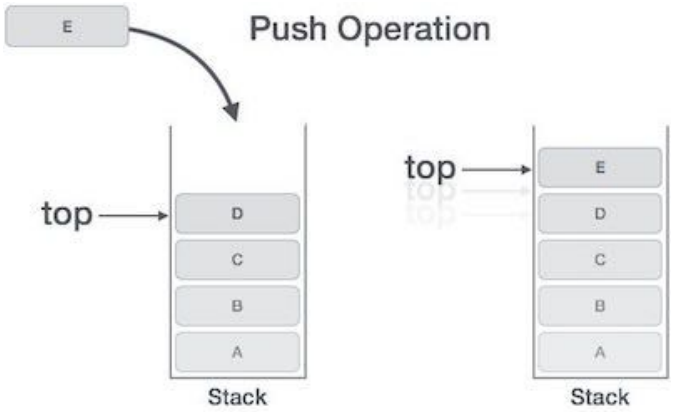
Example:

```
int * temp = malloc(10*sizeof(int));   // Allocate enough space for
                                          ten integers
free(temp);                           // Free the memory in temp
```

Note: Do not try to use a pointer again once you've freed it. This will lead to unexpected errors.

Introduction to Stack

Stack is a data structure used in computer science. It is a FILO (first-in-last-out) data structure. This means that the first value entered in the stack would be the last one coming out. The main operations on stack include `push`, `pop`, `peek`, and `empty`. `Push` operation is used to add values into stack. `Pop` operation is used to return and remove the top value in the stack. `Peek` operation is used to return the top value in stack but not remove it. `Empty` operation is used to check whether the stack is empty or not.



Example:

```
int * stack = (int*)malloc(10*sizeof(int));
empty();         // Nothing is in stack, so return value would be true
push(34);
push(78);        // After adding 2 numbers, 8 space remained
empty();         // 2 numbers in stack, so return value would be false
int x = pop();   // x = 78, because 78 is the last number we added
into
                    stack. It will be the first one popped out. The
                    only number in stack now is 34.
```