# Semi Automatic Garbage Collection

## Semi-Automatic Garbage Collector

This is a garbage collector library written in C language by @SD (Sourav Das) for automatic collection of dynamic memory allocated by function like malloc, calloc realloc, etc.

In many programs it has been found that many dynamic allocation are left un freed, Such a situation leads to *memory leakage* in the program.
OR It is *pain to free all the memory individually* knowing exactly when we don't need them.

Memory allocated Dynamically Remain as it is even after the pointer to it is long gone leading to memory overhead and leakage.
This library has been created with that I mind, Aiming at the above problem specifically.

> All you need to do is allocate dynamic memory using our sagc_malloc or sagc_calloc function and send the pointer which holds the address to such allocated blocks to a function and we will make sure that the block is freed whenever the pointer is destroyed.

For Example.

```
#include "sagc.h"

int any_function()
{
        int * r = sagc_malloc(sizeof(int));
        sagc_RegisterVar(r);
        //Your code
        //Do anything you want
        //The allocated memory will be freed when the function exits.
        //OR more specifically when the variable 'r' is destroyed.
        return 0;
}
```

No Need to free the variable.
Sagc = semi automatic garbage collector

It is designed to be efficient and compatible with modern Abstract Data Structure and Custom made C object the next version of the library will contain specific code where destructor functions can be called upon group of dynamically allocated memory.
For Example.

```
sagc_RegisterVar(variable , destructor_function );
```

A specific data structure can have several other dynamically allocated memories like a linked list can contain a many nodes. Now we no longer need to assign or register all those nodes all we need to do is register the starting node and a function that will free all the other nodes.

# Memory Leak Detection and Demonstration

```c
#include "Gc.h"

int do_nothing()
{
	int * r = sagc_malloc(sizeof(int));
	sagc_RegisterVar(r);
	printf("\nDoNothing.r = %p ",r);
	return 0;
}

int main()
{
	do_nothing();
	int * r = sagc_malloc(sizeof(int));
	printf("\main.r = %p ",r);

	//Leak Detection Function
	sagc_Garbagecheck();
	return 0;
}
```

```
DoNothing.r =  0000002540
Main.r =  000000254f

Allocated List [
Address = 000000254f, blocks = 1 , size = 4  ]
Max GC = 2 , StillActive = 1 , Total Calls = 2
Current Dynamic Memory (in use) = 0 KiB 4 Bytes
Maximum Dynamic Memory Used = 0 KiB 8 bytes
```

**OutPut :**

Notice that do_nothing.r was freed when the function was done executing, but the r variable inside main was not as GarbageCheck() function was called inside main.
All Parameters of GarbageCheck was Self explanatory I guess.

# Extending Variable Life Span Using Scope Tolerance

```
#include "Gc.h"

int* do_nothing()
{
	int * r = sagc_malloc(sizeof(int)); //allocate memory
	sagc_RegisterVar(r); //assign and forget
	printf("\ndo_nothing.r = %p",r);
	return r;
}

int main()
{
	int * r = sagc_malloc(sizeof(int));
	printf("\nMain.r = %p",r);
	sagc_RegisterVar(r);
	r = do_nothing();
	sagc_ExtendVar(r); //Extend Life Span


	sagc_Garbagecheck(r);
	return 0;
}
```

```
DoNothing.r =  0000002540
Main.r =  000000254f

Allocated List [
Address = 0000002540, blocks = 1 , size = 4  ]
Max GC = 2 , StillActive = 1 , Total Calls = 2
Current Dynamic Memory (in use) = 0 KiB 4 Bytes
Maximum Dynamic Memory Used = 0 KiB 8 bytes
```

**OutPut :**

Notice that do_nothing.r was not freed when the function was done executing this time as its life span was extended using ExtendVar function, but the r variable inside main was, as r was replaced by the r from the do_nothing function thus making the previous allocation of r inaccessible from the current scope leaving it to be garbage collected.

If a Life Span of a variable is extended then the variable is not garbage collected within that scope.