GCC has attributes with which you can tell the compiler about how a lot of things should be handled by the compiler. With this feature, he functions defined as constructor function would be executed before the function main starts to execute, and the destructor would be executed after the main has finished execution. The GCC function attributes to define constructors and destructors are as follows:

\_\_attribute\_\_((constructor))

\_\_attribute\_\_((destructor))

\_\_attribute\_\_((constructor (PRIORITY)))

\_\_attribute\_\_((destructor (PRIORITY)))

\_\_attribute\_\_((constructor)) void begin (void)

{

/\* Function Body \*/

}

\_\_attribute\_\_((destructor)) void end (void)

{

/\* Function Body \*/

}

After declaring the functions as constructors and destructors as above, gcc will automatically call begin () before calling main () and call end () after leaving main or after the execution of exit () function.

Multiple constructors and destructors can be defined and can be automatically executed depending upon their priority. In this case the syntax is \_\_attribute\_\_((constructor (PRIORITY))) and \_\_attribute\_\_((destructor (PRIORITY))). In this case the function prototypes would look like.

void begin\_0 (void) \_\_attribute\_\_((constructor (101)));

void end\_0 (void) \_\_attribute\_\_((destructor (101)));

void begin\_1 (void) \_\_attribute\_\_((constructor (102)));

void end\_1 (void) \_\_attribute\_\_((destructor (102)));

void begin\_2 (void) \_\_attribute\_\_((constructor (103)));

void end\_2 (void) \_\_attribute\_\_((destructor (103)));

The constructors with lower priority value would be executed first. The destructors with higher priority value would be executed first. So the constructors would be called in the sequence: begin\_0, begin\_1 (), begin\_2 () . and the destructors are called in the sequence end\_2 (), end\_1 (), end\_0 (). Note the LIFO execution sequence of the constructors and destructors depending on the priority values.

Note that, when compiling with priority values between 0 and 100 (inclusive), gcc would throw you warnings that the priority values from 0 to 100 are reserved for implementation, so these values might be used internally that we might not know. So it is better to use values out of this range. The value of the priority does not depend, instead the relative values of the priority is the determinant of the sequence of execution.

example:

$ cat constractor.cpp

#include <stdio.h>

#include <stdlib.h>

void at\_exit()

{

printf("in the at exit\n");

}

class A

{

public:

A()

{

printf("in A constructor\n");

}

~A()

{

printf("in A destructor\n");

}

};

A a;

void \_\_attribute\_\_ ((constructor (101))) my\_init1(void)

{

printf("in my init1\n");

}

void \_\_attribute\_\_ ((destructor (101))) my\_fini1(void)

{

printf("in my fini1\n");

}

void \_\_attribute\_\_ ((constructor (102))) my\_init2(void)

{

printf("in my init2\n");

}

void \_\_attribute\_\_ ((destructor (102))) my\_fini2(void)

{

printf("in my fini2\n");

}

int main()

{

printf("in main\n");

atexit(at\_exit);

return 0;

}

$ ./constrcator

in my init1

in my init2

in A constructor

in main

in the at exit

in A destructor

in my fini2

in my fini1