GO CLASSES

Memory Layout of a program

Storage classes in C language

auto
register
static
extern

Memory Layout of a program


hello.c
hello.out
load it into MM
till Compilation
MM

GO CLASSES

printf ("%u", &a)

logical address

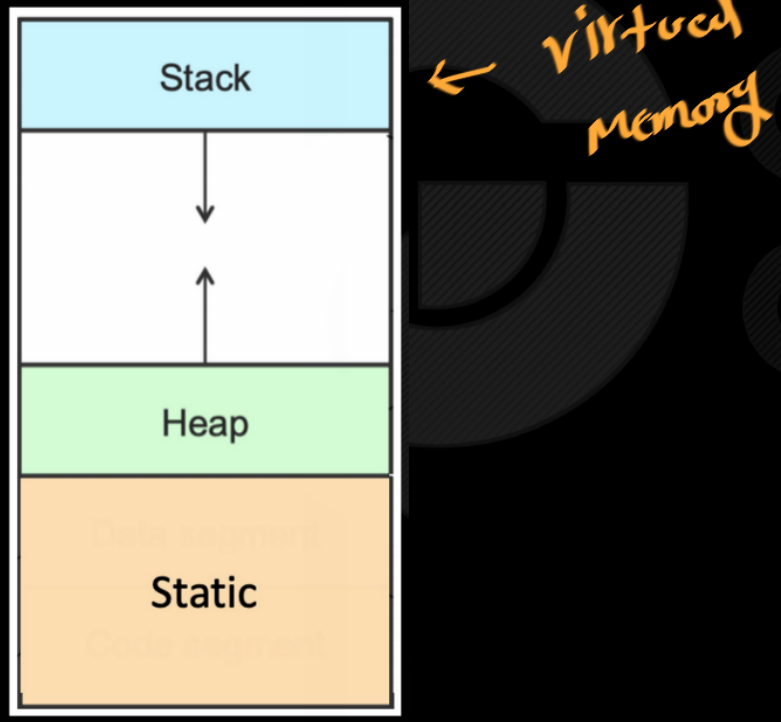
Compiler assume that whole memory is available to just one program.

← Virtual memory

Contigious

GO CLASSES

Memory Layout of a program



virtual memory

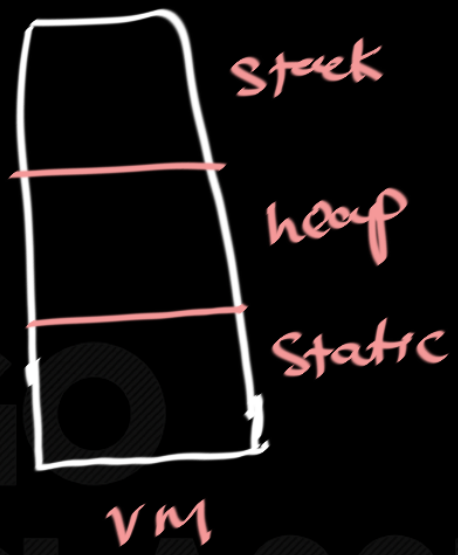
instructions

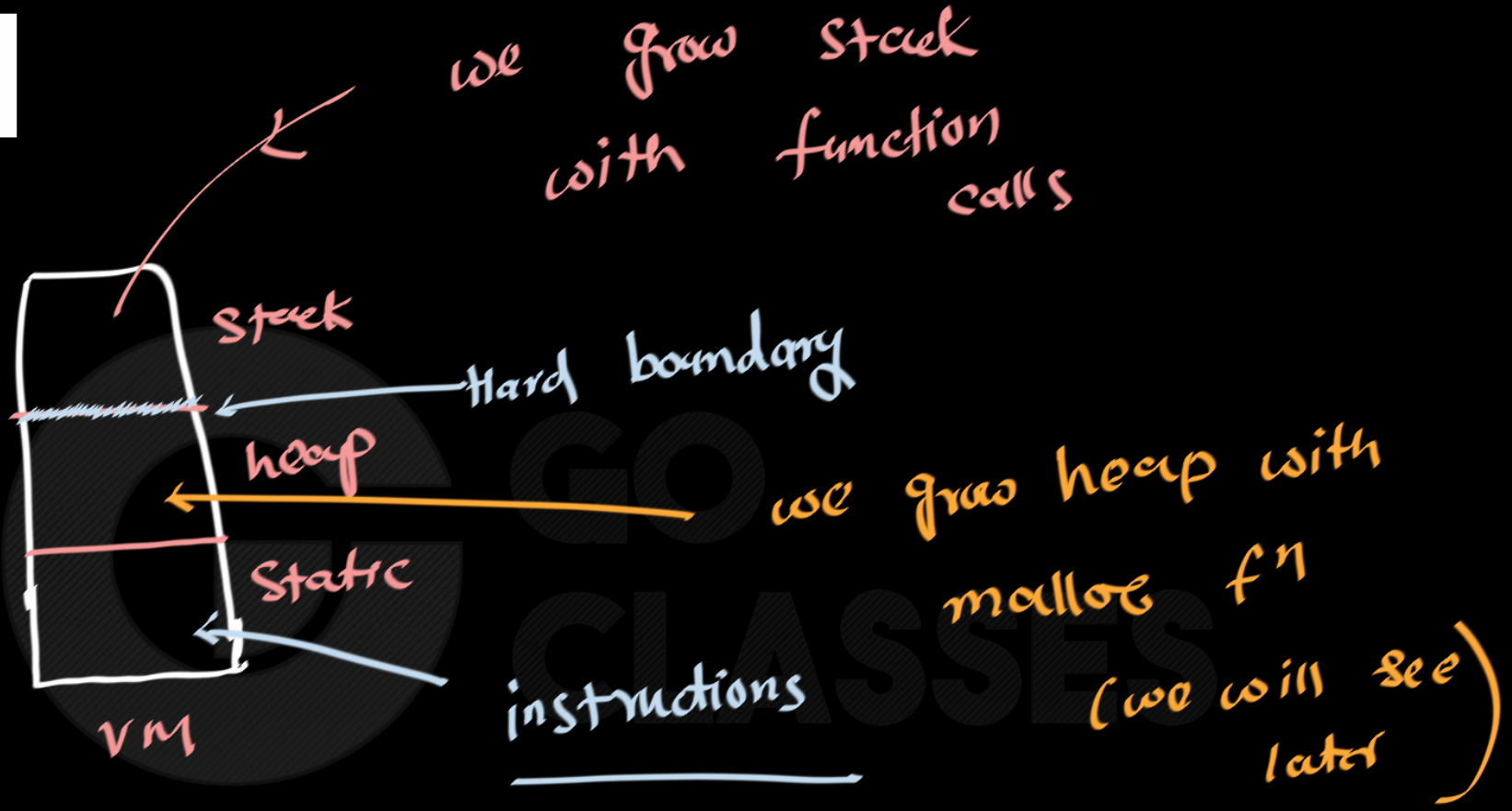
hello.out

Stack

heap

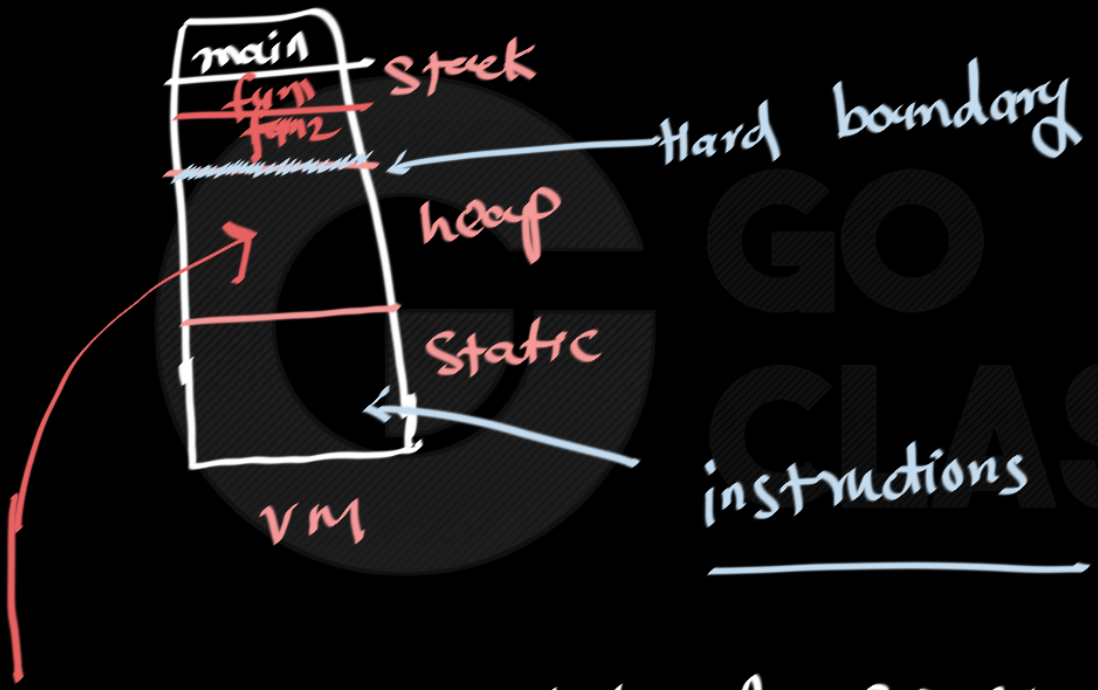
Static ~ to Store Code (instructions)




we grow stack with function calls
stack
Hard boundary
heap
we grow heap with malloc fn (we will see later)
static
instructions
VM

GO CLASSES

Design choice



there is a lot of space in heap, still you can not call any more function

GO CLASSES

Design choice

fix boundary design choice is not a good idea

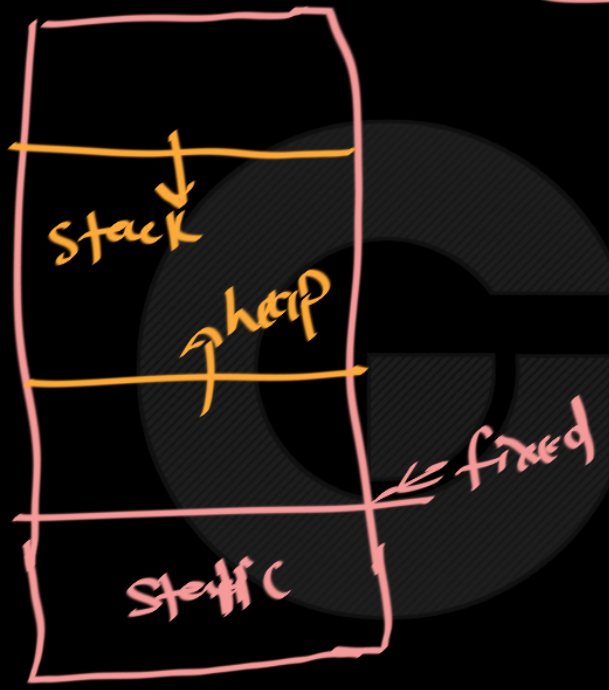
```
main( )
{
  fun1( );
  fun2( );
  fun3( );
}
```

Stack is full, and heap is empty

Still you can not call fun3( );

GO CLASSES

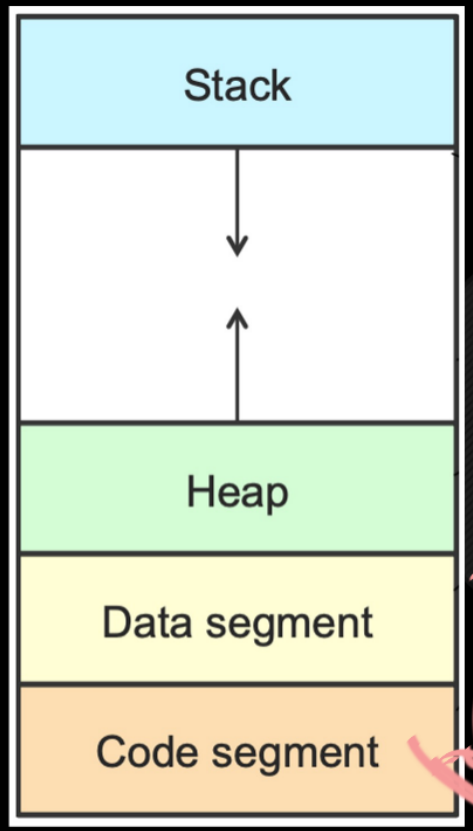
Design Choice ?



C proj (PS

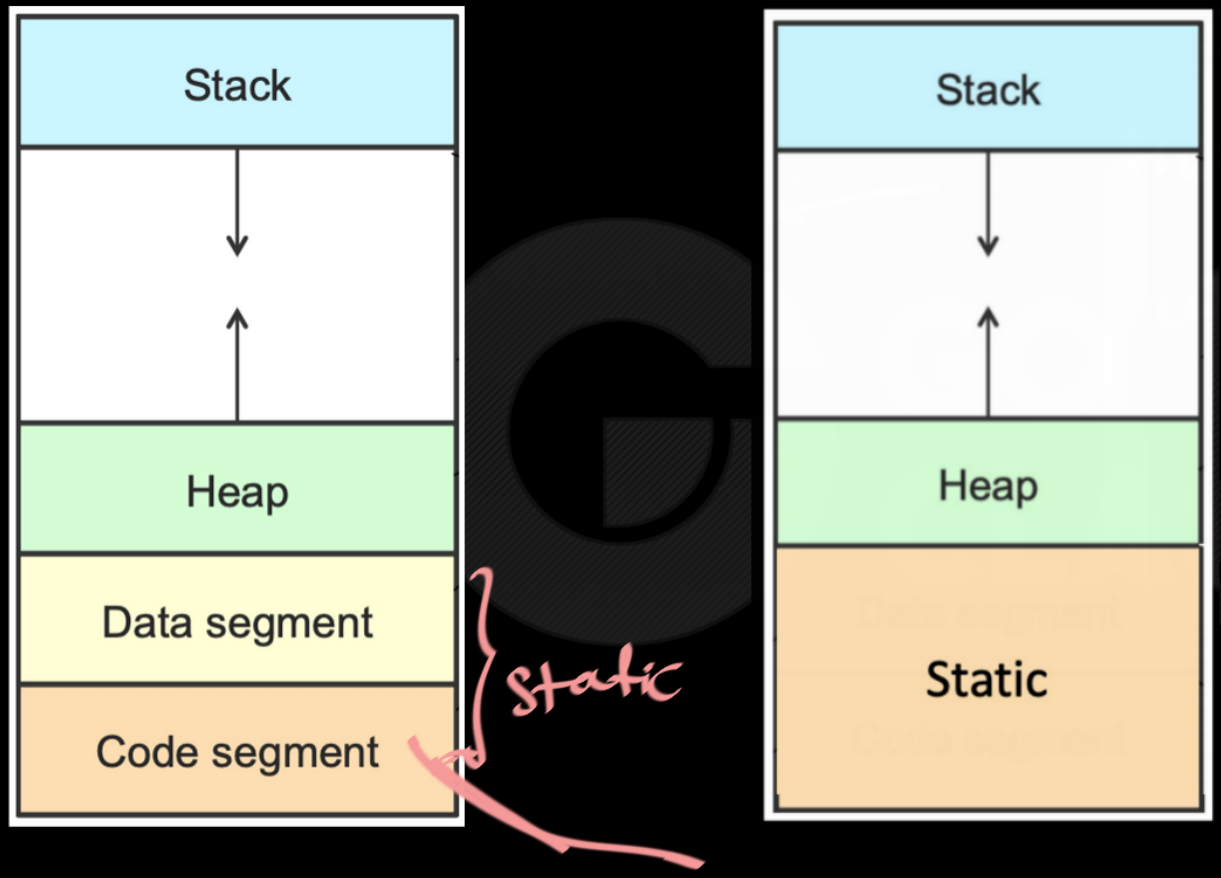
50 50

30 pages 60 pages



this is when Program (hello.out) will fit in.

static


Stack
Heap
Data segment
Code segment
Static
Stack
Heap
Static

- **The Stack -** This is where local variables and Function Activation records are stored. Size changes at runtime.

- **The Heap -** This is where dynamically allocated data is stored. Size changes at runtime.

↳ malloc ( )

- **The Static area -** This is where statically declared data (static and global variables) and code is stored. Size fixed at runtime

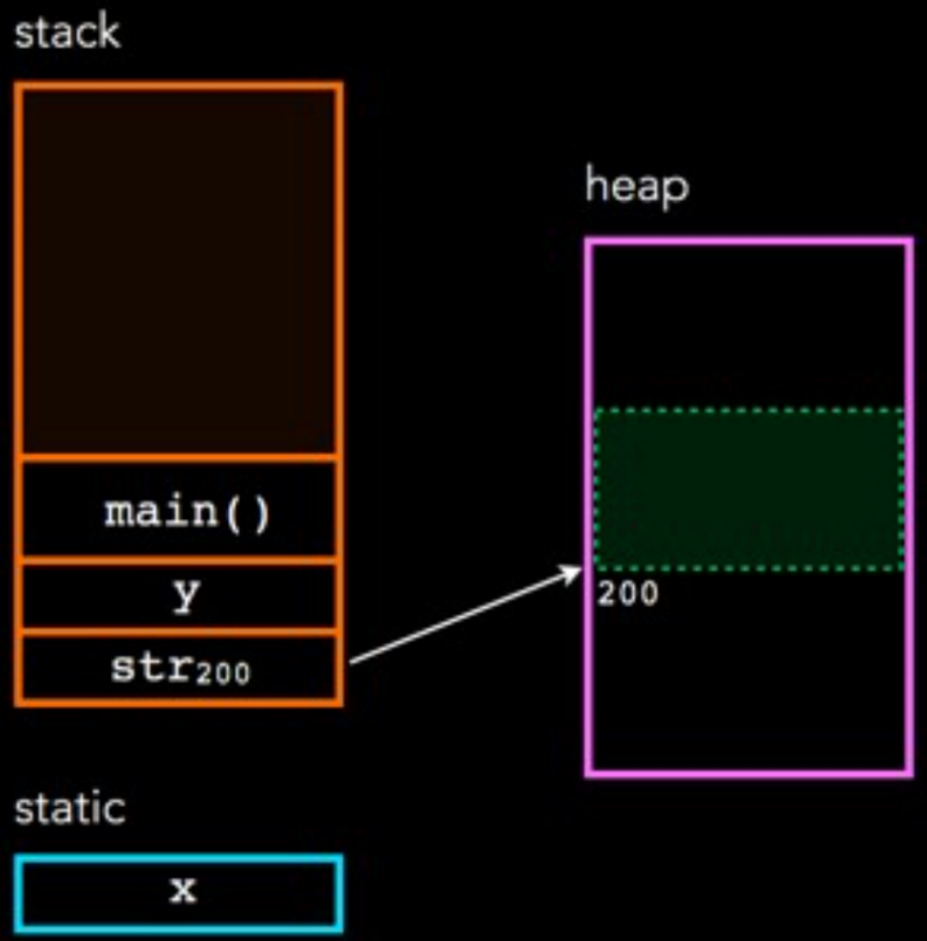
GO CLASSES

From Now on, Just know

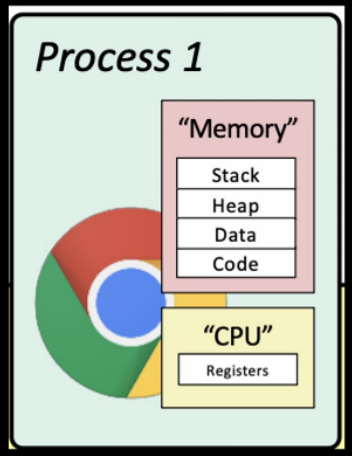
- there is some stack
- there is some heap.
- there is some static area.

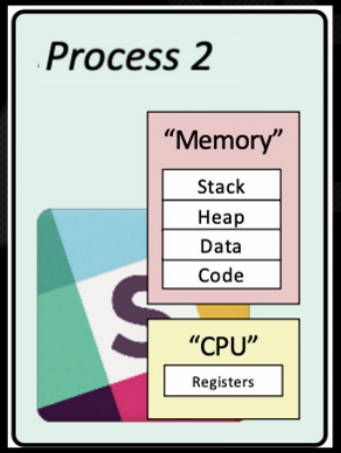
Enough for C prg. Course

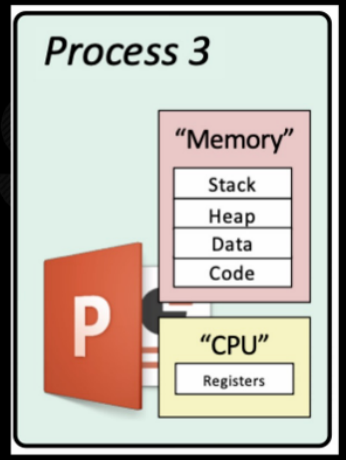
```c
#include <stdio.h>


int x;
int main(void)
{
    int y;
    char *str

    y = 4;
    printf("stack memory: %d\n", y);

    str = malloc(100*sizeof(char));
    str[0] = 'm';
    printf("heap memory: %c", str[0]);
    free(str);
    return 0;

}
```



It's an illusion!







GO CLASSES

Virtual memory is one of the great ideas in computer systems. A major reason for its success is that it works silently and automatically, without any intervention from the application programmer.

Computer Systems: A Programmer's Perspective.

```
int a = 5;
```

Storage classes in C language

auto
register
static
extern

auto
register
static
extern

} local variables

} for both global and local variables

auto Storage Class

- Storage – Stack
- Initial Value – Garbage
- Scope – within block
- Life Time – end of block

auto int a;

{ auto int a; }

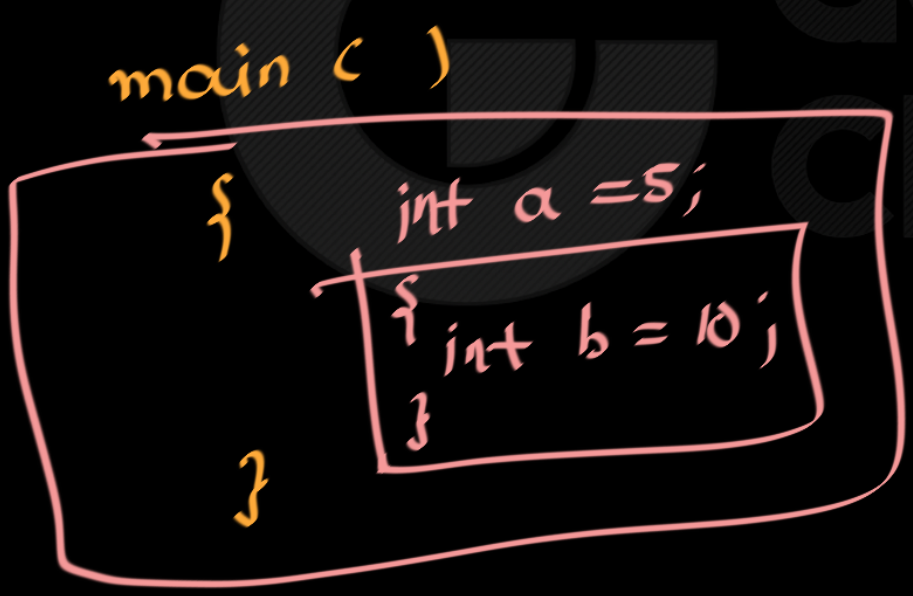
→ no scope (You can't access)
→ no lifetime (Not in memory)

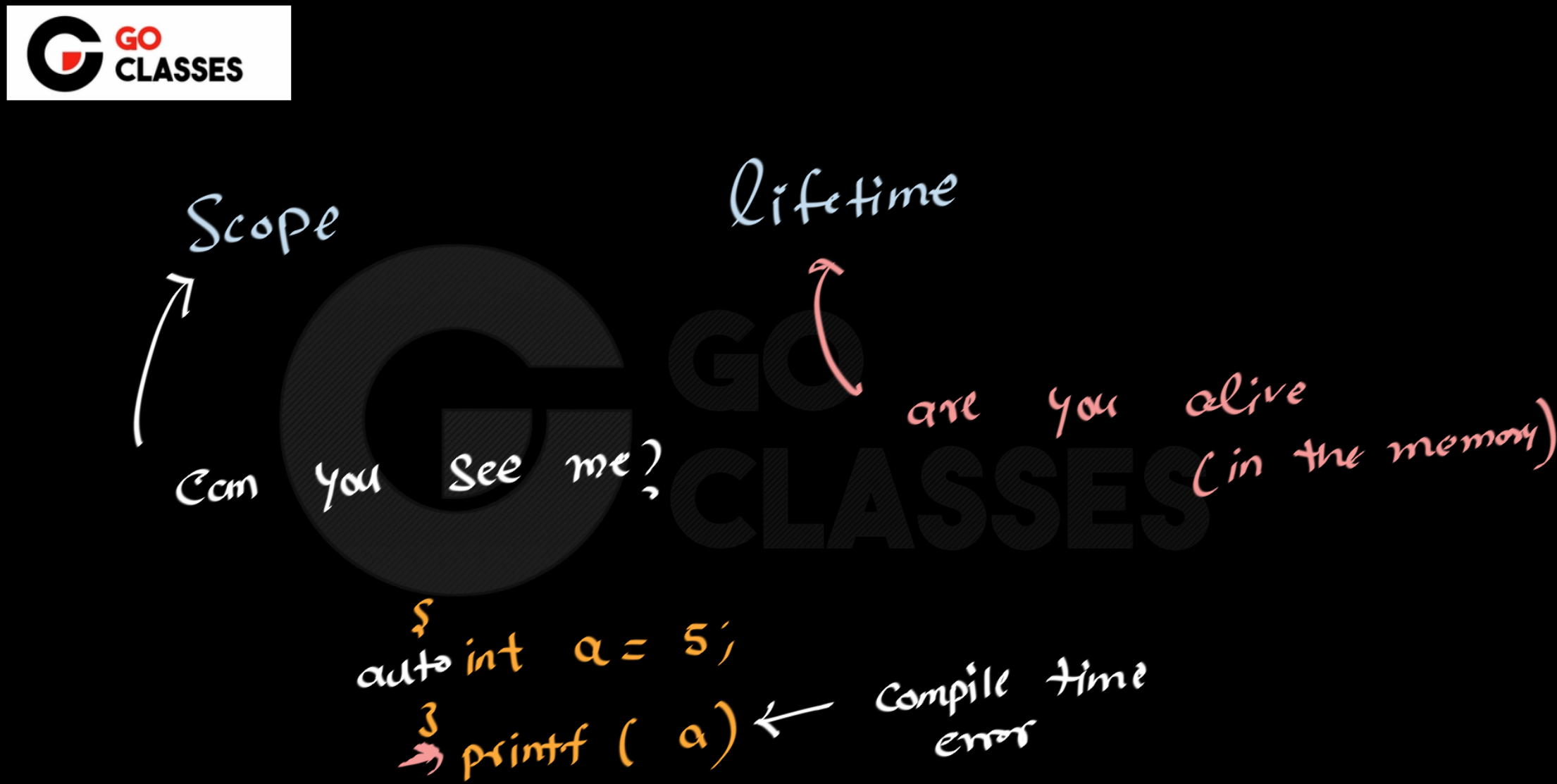
- ONLY used for local variables

GO CLASSES

Scope

lifetime



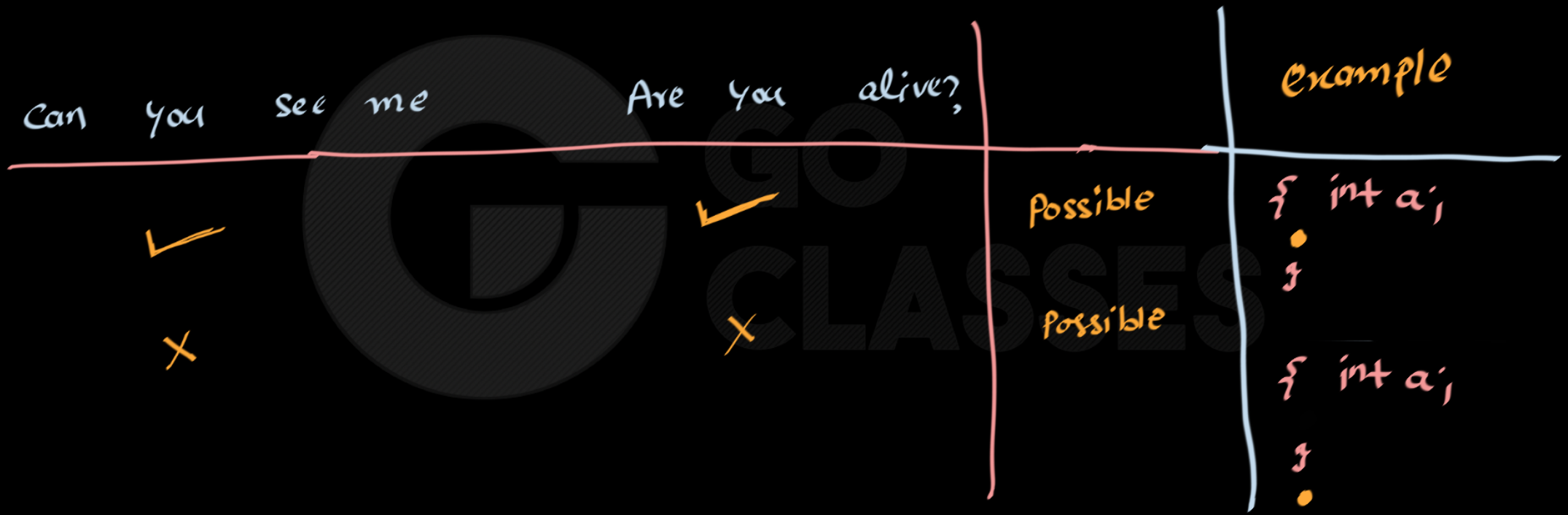

Scope
lifetime
Can you see me?
are you alive (in the memory)
{
auto int a = 5;
printf (a)
Compile time error

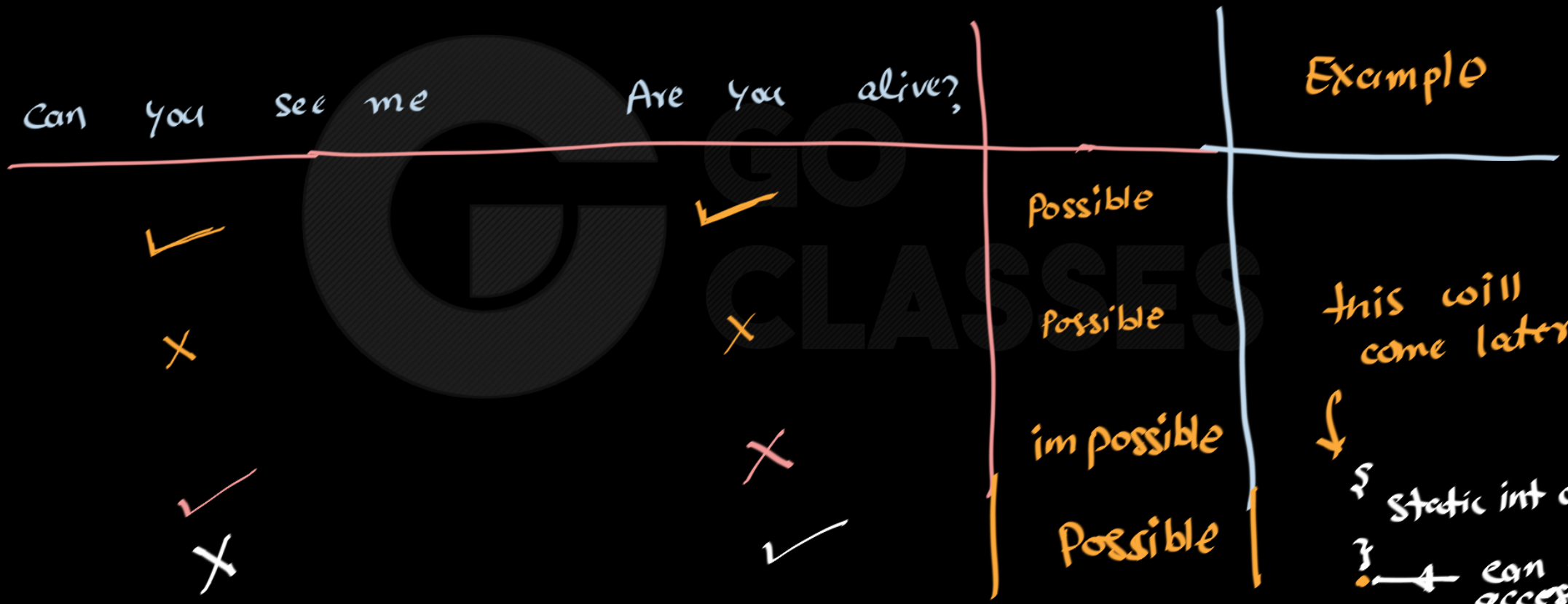
GO CLASSES

```
{
auto int a = 5;
}
```

⟹ Can you see (print) a ?

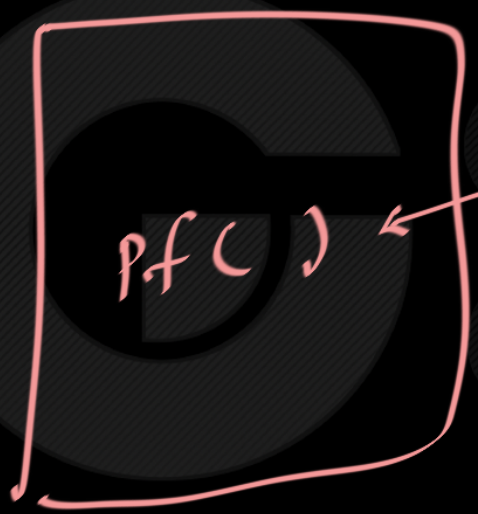
⇒ is this variable a alive ? (memory)


Can you see me
Are you alive?
Possible
Possible
Example
{ int a;
}
{ int a;
}


Can you see me
Are you alive?
Example
Possible
Possible
this will come later
impossible
Possible
static int a;
can not access

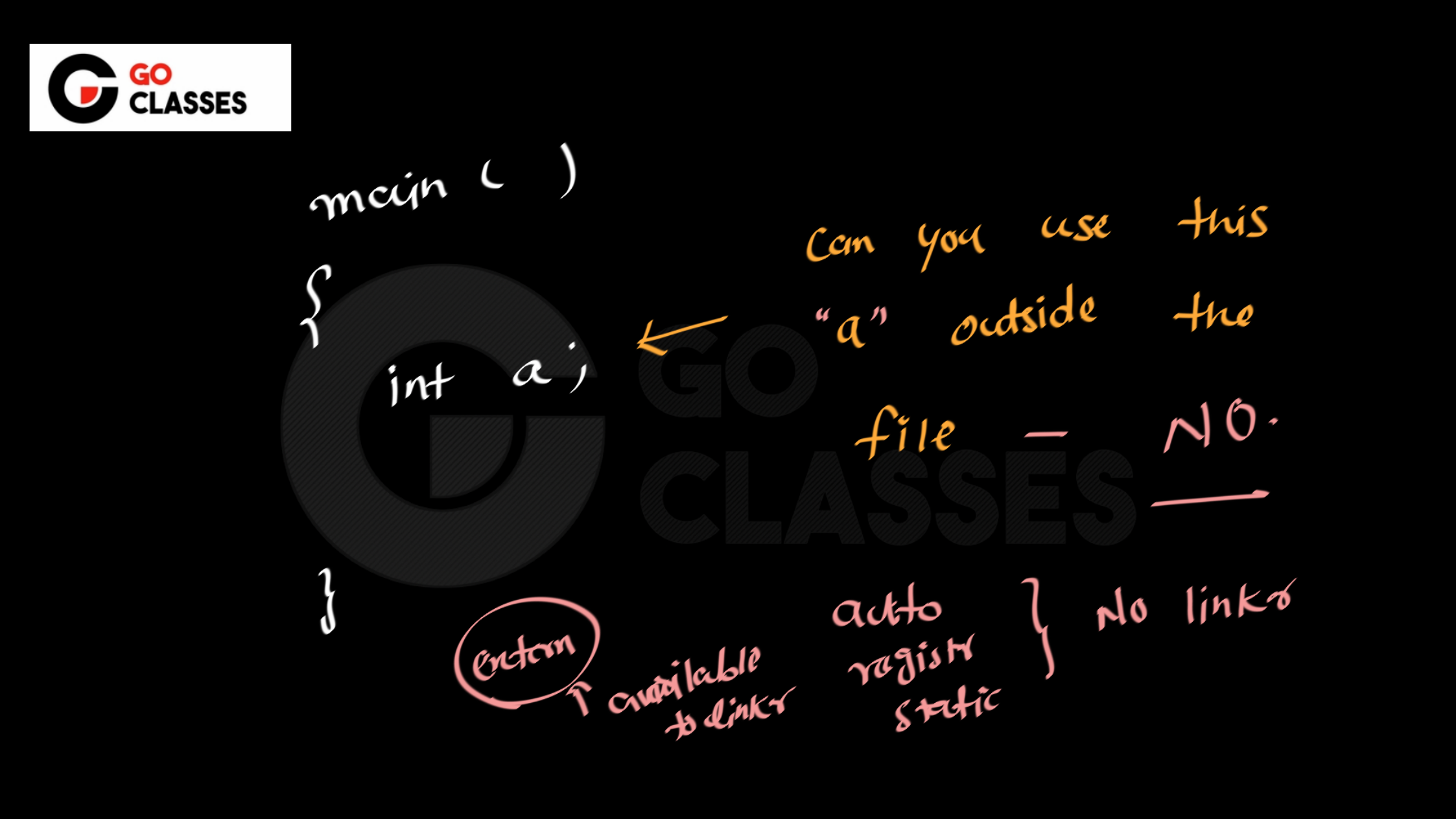
auto Storage Class (Cont..)

- The auto storage class is the default storage class for all local variables.
- Objects with the auto class are NOT available to the **linker**.
- You can't use for global variables.



defined in some other file

```
{
    auto int a;
}
```


GO CLASSES
main ( )
{
int a;
}
Can you use this "a" outside the file — NO.
extern
available to linker
auto
register
static
No linkage

auto Storage Class example -

```
main ()
{
    auto int b;                  /*  Valid auto declaration   */
    for (b = 0; b < 10; b++)
      {
        auto int a = b + a;      /*   Valid inner block declaration    */
      }
}
```

auto Storage Class example -

```
auto int a;          /*  Illegal -- auto must be within a block  */

main ()
{
    auto int b;                 /*  Valid auto declaration   */
    for (b = 0; b < 10; b++)
      {
        auto int a = b + a;     /*   Valid inner block declaration    */
      }
}
```

Register Storage Class

- Storage – CPU register (may)
- Initial Value – Garbage
- Scope – Within Block
- LifeTime – End of block

Same as auto

i is getting used again and again

```
for (register int i=0; i<5; i++)
{
    pf(i);
}
```

Register Storage Class example -

```
void main(){

    register int i =0;
    printf("%d", i);
}
```

suggestion

It's a hint to the compiler that the variable will be heavily used and that you recommend it be kept in a processor register if possible.

Most modern compilers do that automatically, and are better at picking them than us humans.

Register Storage Class example -

```
void main(){

    register int i =0;
    printf("%d", i);
}
```

No one use register keyword.

It's a hint to the compiler that the variable will be heavily used and that you recommend it be kept in a processor register if possible.

Most modern compilers do that automatically, and are better at picking them than us humans.

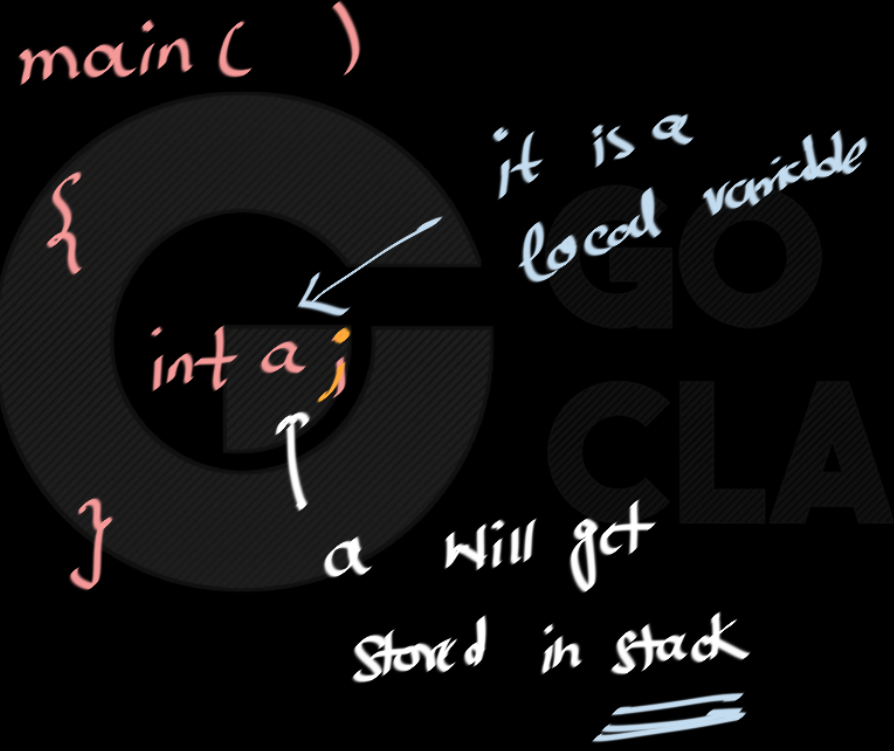
GO CLASSES

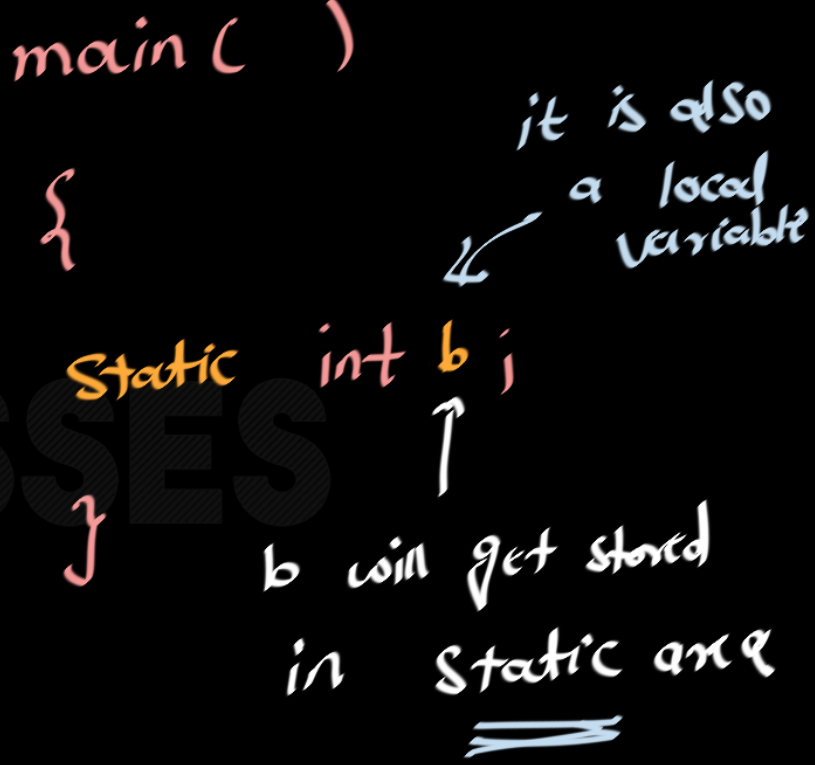
- You can't use auto or register for global variables.
- No one uses auto and register keywords.

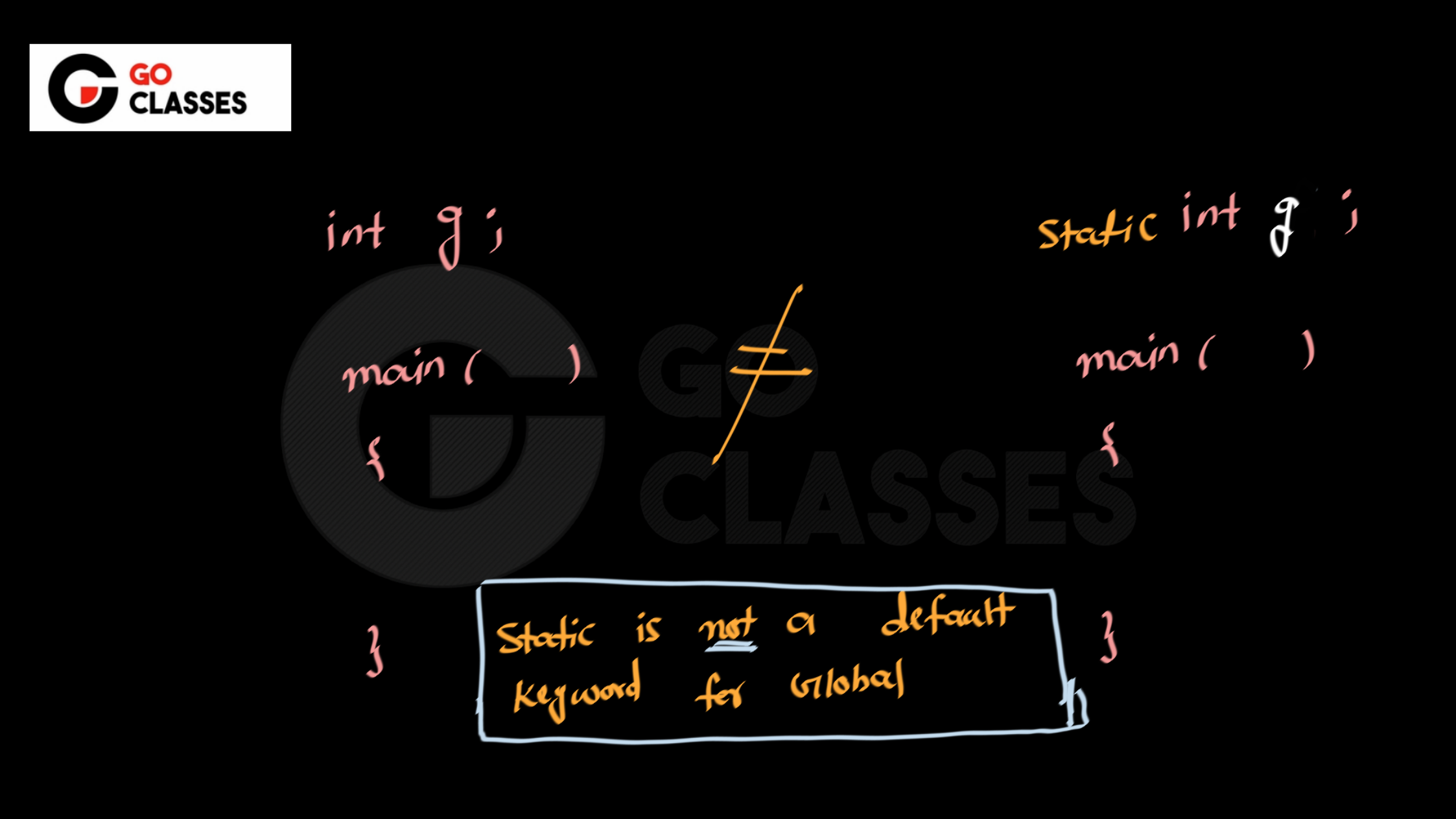
Static Storage Class

as per GATE it is very imp.

- Storage – Static Memory
- Initial Value – zero
- Scope – within block
- Life Time – Till end of program
- Objects with the Static class are also NOT available to the **linker**.


main ( )
{
int a j
}
it is a local variable
a will get stored in stack


main ( )
{
Static int b j
}
it is also a local variable
b will get stored in static area


int g;
main ( )
{
}
≠
static int g;
main ( )
{
}
Static is not a default keyword for Global

GO CLASSES

```
main ( )
{
int j;
}
```

=

```
main ( )
{
auto int j;
}
```

GO CLASSES

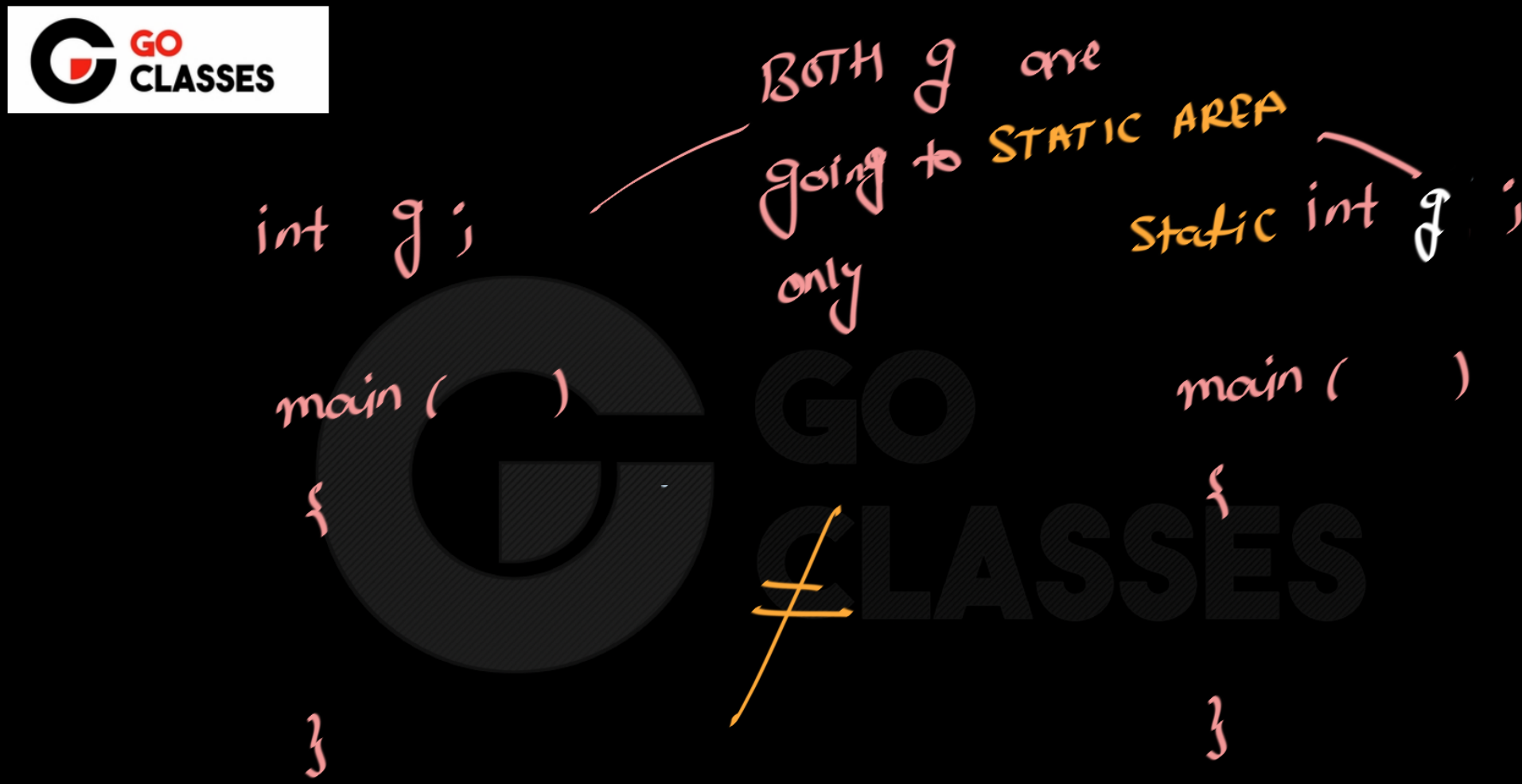
```
int g;
main ( )
{
}
```

g is a global variable and all global variables goes to static area

$\neq$

```
Static int g;
main ( )
{
}
```

a global variable and also static variable storage is static


GO CLASSES
BOTH g are going to STATIC AREA only
int g;
main ( )
{
}
≠
Static int g;
main ( )
{
}

GO CLASSES

local (auto) ⇒ stack storage

local (static) ⇒ static storage

global (static, or no static) ⇒ static storage

- Scope – within block
- Life Time – Till end of program

if local then within { }

if global then within same file (you can not access in other file)

Static variable get destroyed only if Program terminates.

GO CLASSES

```
fun ( )
{
    static int a = 5;
}
main ( )
{
    fun ( );
}
```

fun( ); ⟶ a is in memory but we cant access it.

Static Storage Class Example -

```c
#include <stdio.h>

void increment() {
    static int count = 0;
    count++;
    printf("%d ", count);
}

int main() {
    increment();
    increment();
    increment();
    return 0;
}
```

output: 1 2 3

Question: What will be the output of following program ?

```c
#include <stdio.h>

void increment() {
    static int count = 5;
    count--;
    printf("%d ", count);
}

int main() {
    increment();
    increment();
    increment();
    return 0;
}
```

What will be the output of the program?

a) 4 3 2

b) 5 4 3

c) 5 5 5

d) 4 4 4

Question: What will be the output of following program ?

```c
#include <stdio.h>

void increment() {
    static int count = 5;
    count--;
    printf("%d ", count);
}

int main() {
    increment();
    increment();
    increment();
    return 0;
}
```

4 3 2

What will be the output of the program?

a) 4 3 2

b) 5 4 3

c) 5 5 5

d) 4 4 4

5 4 3 2

```c
#include <stdio.h>

static int y = 5;

void fun() {
    static int y;
    y++;
    return;
}


int main() {

    fun();
    printf("%d", y);

    return 0;
}
```

→ 5

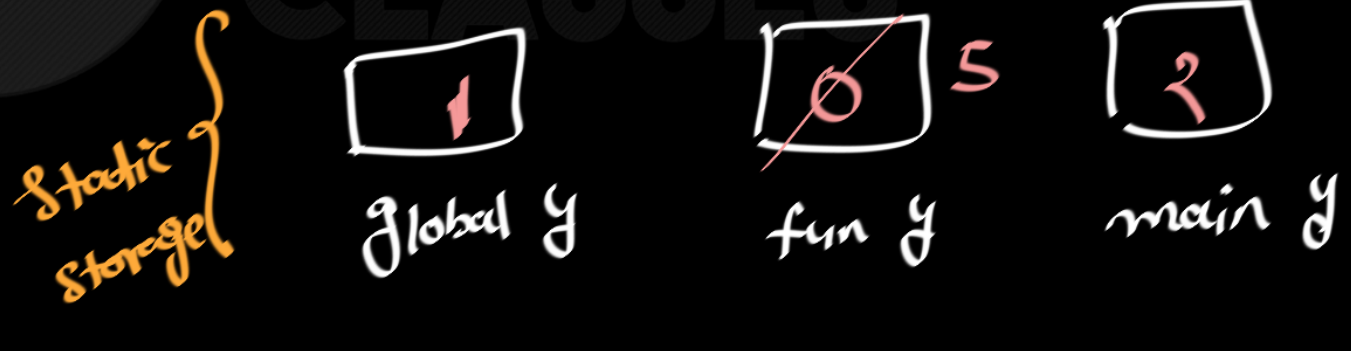
Question: What will be the output of program given?

```c
#include <stdio.h>

static int y = 1;

void fun() {
    static int y;
    y=5;
    return;
}


int main() {
    static int y = 2;

    fun();
    printf("%d", y);


    return 0;
}
```

Question: What will be the output of program given?

```c
#include <stdio.h>

static int y = 1;

void fun() {
    static int y;
    y=5;
    return;
}


int main() {
    static int y = 2;

    fun();
    printf("%d", y);


    return 0;
}
```

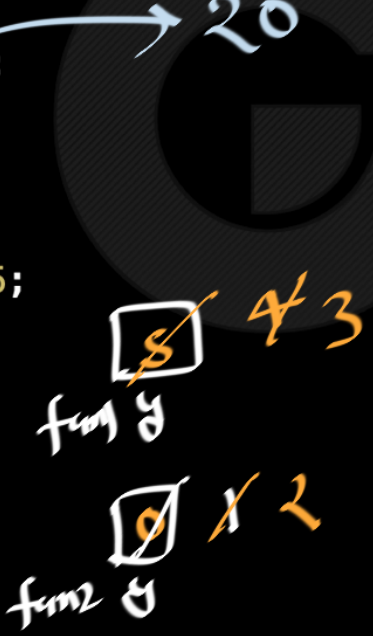
Question: What will be the output of program given?



```c
main()
{
    int x = 0;

    for (int i = 1; i<5; i++)
    {
        x += fun1()+fun2();
    }

    printf("%d", x);
}


int fun1(){
    static int y = 5;
    y--;
    return y;
}
int fun2(){
    static int y;
    y++;
    return y;
}
```

Question: What will be the output of program given?

```c
main()
{
    int x = 0;

    for (int i = 1; i<5; i++)
    {
        x += fun1()+fun2();
    }

    printf("%d", x);
}


int fun1(){
    static int y = 5;
    y--;
    return y;
}
int fun2(){
    static int y;
    y++;
    return y;
}
```



Question: What will be the output of program given?

$i=1 \quad x = 0 + \underbrace{fun1()}_{4} + \underbrace{fun2()}_{1} = 5$

$i=2 \quad x = 5 + \underbrace{fun1()}_{3} + \underbrace{fun2()}_{2} = 10$

$i=3 \quad x = 10 + 2 + 3 = 15$

$i=4 \quad x = 15 + 1 + 4 = 20$

Extern Storage Class

- Storage – Static Memory
- Initial Value – zero
- Scope – Global
- Life Time – Till end of program

extern refers to global variable

extern int a;

global variable

- Objects with the extern class are available to the **linker**.

GO CLASSES

extern int a ;

a

int a

a

Extern Storage Class example -

Example 1

```c
#include <stdio.h>

int max;

int main() {
    int len;
    extern int max;
    printf("%d", max);
    max = 5;
}
```

0

it is a way to tell compiler that there is a global variable max.

Extern Storage Class example -

```c
#include <stdio.h>

int max;

int main() {
    int len;

    extern int max;

    printf("%d", max);

    max = 5;

}
```

(Suppose we remove extern from this code)

→ 0 ← output

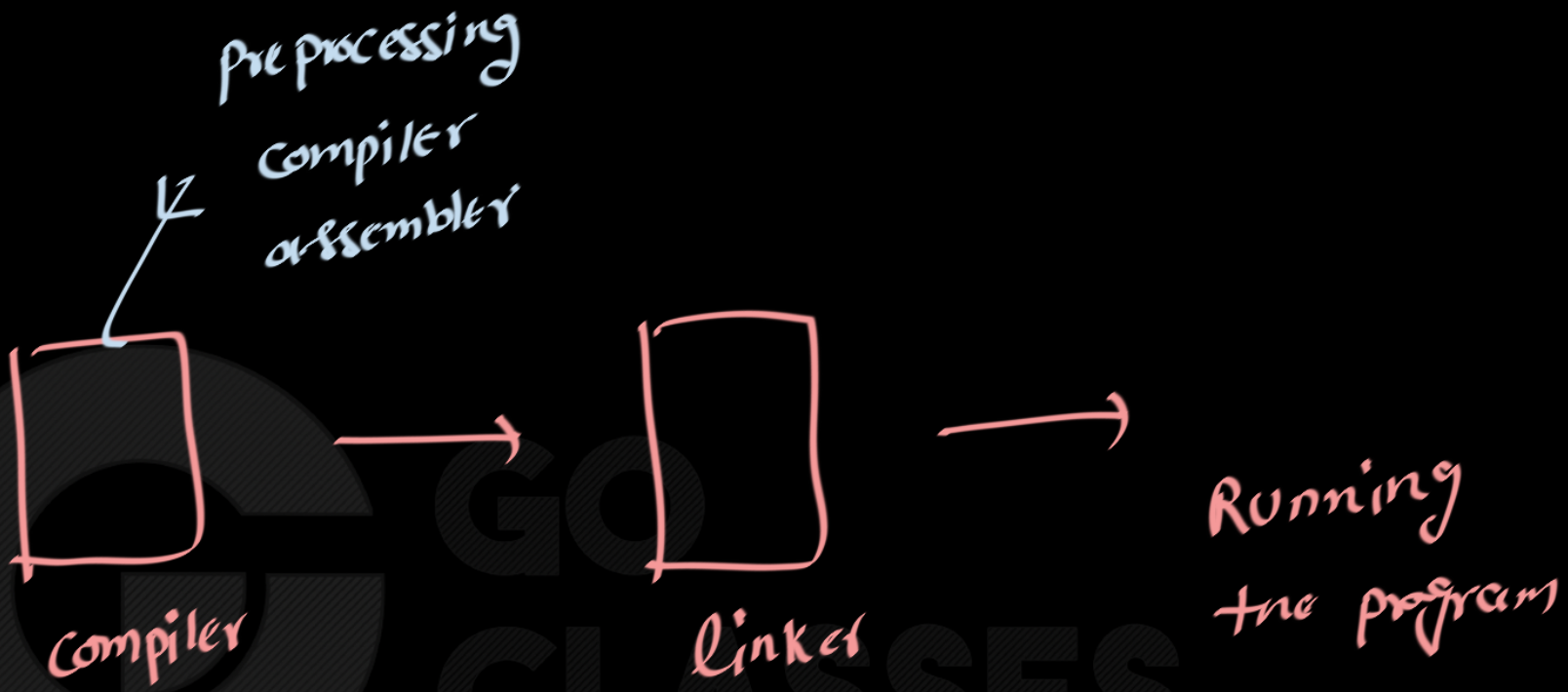
Extern Storage Class example -

Example 2

this line is telling to the compiler that - Please don't worry, there is a global variable max which is either in this file or in another file

```c
#include <stdio.h>

int main() {
    int len;
    extern int max;
    printf("%d", max);
    max = 5;
}

int max;
```

0 output


Preprocessing
Compiler
assembler
Compiler
Linker
Running
the program

Extern Storage Class example -

```c
#include <stdio.h>

int max;

int main() {
    int len;
    extern int max;
    printf("%d", max);
    max = 5;
}
```

this is useless

```c
#include <stdio.h>

int main() {
    int len;
    extern int max;
    printf("%d", max);
    max = 5;
}

int max;
```

this is useful

Extern Storage Class example -

```c
#include <stdio.h>

int max;

int main() {
    int len;
    extern int max;
    printf("%d", max);
    max = 5;
}
```

this is useless

```c
#include <stdio.h>

int main() {
    int len;
    extern int max;
    printf("%d", max);
    max = 5;
}

int max;
```

this is useful (but can be omitted)

why not shift the int max to top and remove extern?

Extern Storage Class example -

```c
#include <stdio.h>

int max;

int main() {
    int len;

    extern int max;

    printf("%d", max);

    max = 5;

}
```

```c
#include <stdio.h>

int main() {
    int len;

    extern int max;

    printf("%d", max);

    max = 5;

}

int max;
```

Extern Storage Class example -

Common practice is to place definitions of all external variables at the beginning of source file and then omit all the extern declarations.

GO CLASSES

```
extern int a;

main ( )
{
}

int a
```

Global use of extern

We can NOT refer local variable using extern

```
main ( )
{
  int a;
  extern int a;
}
```

X

here extern is reffering to local variable

```
main ( )
{
  extern int a;
}
int a;
```

use local at extern

Note -

- Extern always refer global variables but it can be used with local or global variable.

GO CLASSES

Optional

Real use of extern keyword

(with 2 files or more)

```c
/* main.c */
#include <stdio.h>

extern int gInt;

void change_extern(void);

int main()
{
    printf("main1 gInt %d\n", gInt);
    change_extern();
    gInt = 5;
    change_extern();
    printf("main2 gInt %d\n", gInt);
    return 0;
}
```

Will this program get compiled? ⇒ YES.

```c
/* main.c */
#include <stdio.h>

extern int gInt;

void change_extern(void);

int main()
{
    printf("main1 gInt %d\n", gInt);
    change_extern();
    gInt = 5;
    change_extern();
    printf("main2 gInt %d\n", gInt);
    return 0;
}
```

what kind of error it will show

⇒ linker error

```c
/* main.c */
#include <stdio.h>

extern int gInt;

void change_extern(void);

int main()
{
    printf("main1 gInt %d\n", gInt);
    change_extern();
    gInt = 5;
    change_extern();
    printf("main2 gInt %d\n", gInt);
    return 0;
}
```

```c
/* main.c */
#include <stdio.h>

extern int gInt;

void change_extern(void);

int main()
{
    printf("main1 gInt %d\n", gInt);
    change_extern();
    gInt = 5;
    change_extern();
    printf("main2 gInt %d\n", gInt);
    return 0;
}
```

```c
/* extern.c */
#include <stdio.h>

int gInt;

void change_extern(void)
{
    printf("change_extern1 gInt %d\n", gInt);
    gInt = 10;
    printf("change_extern2 gInt %d\n", gInt);
}
```

GO CLASSES

How does it look like after Linking?

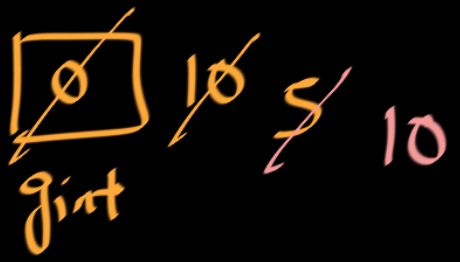
One file

```c
extern int gInt;

void change_extern(void);

int main()
{
    printf("main1 gInt %d\n", gInt);
    change_extern();
    gInt = 5;
    change_extern();
    printf("main2 gInt %d\n", gInt);
    return 0;
}
```

```c
int gInt;

void change_extern(void)
{
    printf("change_extern1 gInt %d\n", gInt);
    gInt = 10;
    printf("change_extern2 gInt %d\n", gInt);
}
```

```c
int gInt;

void change_extern(void)
{
    printf("change_extern1 gInt %d\n", gInt);
    gInt = 10;
    printf("change_extern2 gInt %d\n", gInt);
}

int main()
{
    printf("main1 gInt %d\n", gInt);
    change_extern();
    gInt = 5;
    change_extern();
    printf("main2 gInt %d\n", gInt);
    return 0;
}
```

linked file



main1 gInt 0

change_extern1 gInt 0

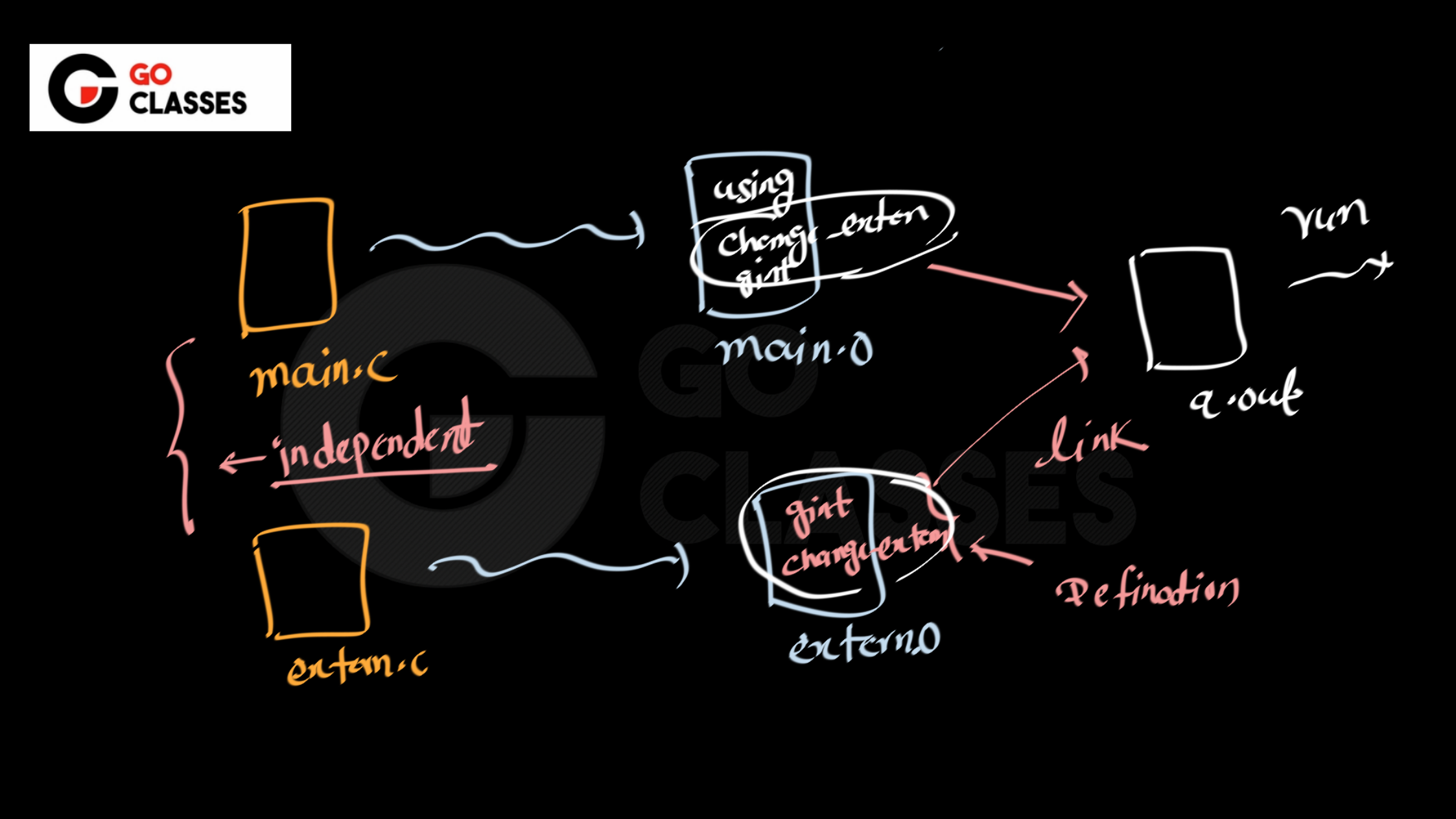
change_extern2 gInt 10

change_extern1 gInt 5

change_extern2 gInt 10

main2 gInt 10

```
main1 gInt 0
change_extern1 gInt 0
change_extern2 gInt 10
change_extern1 gInt 5
change_extern2 gInt 10
main2 gInt 10
```


GO CLASSES
main.c
'independent
extern.c
using change_extern
first
main.o
extern.o
first changeextern
Definition
link
a.out
run

GO CLASSES

Few more Questions

```c
#include<stdio.h>
main()
{
   extern int num;
   num = 20;
   printf("%d", num);
}
```

there is a global variable num either in this file or any other file (don't worry)

No compilation error

Linker error

```c
#include<stdio.h>
main()
{
   extern int num;
   num = 20;
   printf("%d", num);
}
```

there is a global variable num either in this file or any other file (don't worry)

linker error

```c
#include<stdio.h>
main()
{

   num = 20;
   printf("%d", num);
}
```

then is a compiler error.

```c
#include <stdio.h>
extern int i; //extern variable
int main(){
    i=5;
    printf("%d",i);
    return 0;
}
```

```c
#include <stdio.h>
extern int i; //extern variable
int main(){
    i=5;
    printf("%d",i);
    return 0;
}
```

No compiler error
linker error

```c
#include <stdio.h>
extern int i;  //extern variable
int main(){
    int i=5;
    printf("%d",i);
    return 0;
}
```

Not using this anywhere (No linking required)

```c
#include <stdio.h>
extern int i;  //extern variable
int main(){
    int i=5;
    printf("%d",i);
    return 0;
}
```

[No Compiler error
No linker error]

```c
#include <stdio.h>
extern int i;     //extern variable
int main(){
     printf("Hello");
     return 0;
}
```

```c
#include <stdio.h>
extern int i;     //extern variable
int main(){
    printf("Hello");
    return 0;
}
```

i am not using this i anywhere

No Compiler error

```c
#include <stdio.h>

void fun(void);

int main(){
    printf("Hello");
    return 0;
}
```

```c
#include <stdio.h>

void fun(void);

int main(){
    printf("Hello");
    return 0;
}
```

No compiler error

not using/calling this f^n anywhere

No linker error also.

```c
#include <stdio.h>

void fun(void);

int main(){
    printf("Hello");
    fun();
    return 0;
}
```

GO CLASSES

```c
#include <stdio.h>

void fun(void);

int main(){
    printf("Hello");
    fun();
    return 0;
}
```

No compiler error

Linker error

we are using/calling

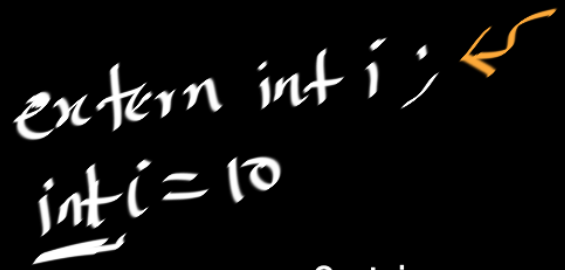
GO CLASSES

```c
#include <stdio.h>

int main(){
    printf("Hello");
    fun();
    return 0;
}
```

```c
#include <stdio.h>

int main(){
    printf("Hello");
    fun();
    return 0;
}
```

compiler will produce error here.

```c
#include<stdio.h>
int main() {

    extern int i = 10;
    printf("%d", i);
    return 0;
}
```



Options:

A. Garbage value

B. 10

C. Compiler Error

D. Linker Error

No one use '=' with extern.

```
#include<stdio.h>
int main() {

    extern int i = 10;
    printf("%d", i);
    return 0;
}
```

extern int i;
int i = 10;

Options:

A. Garbage value

B. 10

C. Compiler Error

D. Linker Error

extern int i;
int i = 10;

```c
#include<stdio.h>
extern int i = 10;
int main() {

    extern int i;
    printf("%d", i);
    return 0;
}
```

Options:

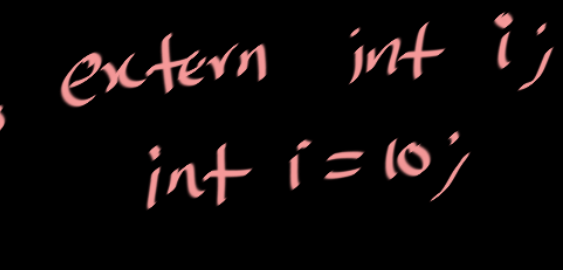
A. Garbage value

B. 10

C. Compiler Error

D. Linker Error

```c
#include<stdio.h>
extern int i = 10;
int main() {

    extern int i;
    printf("%d", i);
    return 0;
}
```



Options:

A. Garbage value

B. 10

C. Compiler Error

D. Linker Error