getchar() : Reading a Character

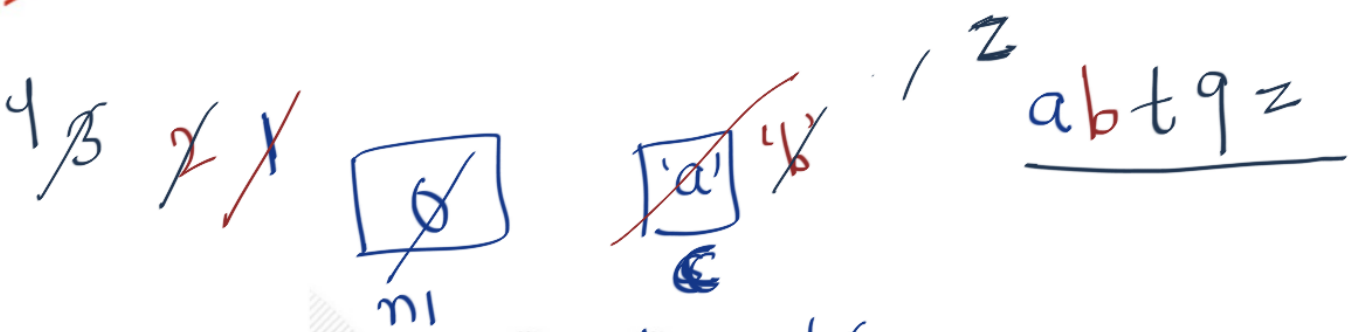
```
c = getchar() is equivalent to scanf("%c", &c)
```

getchar() : Reading a Character

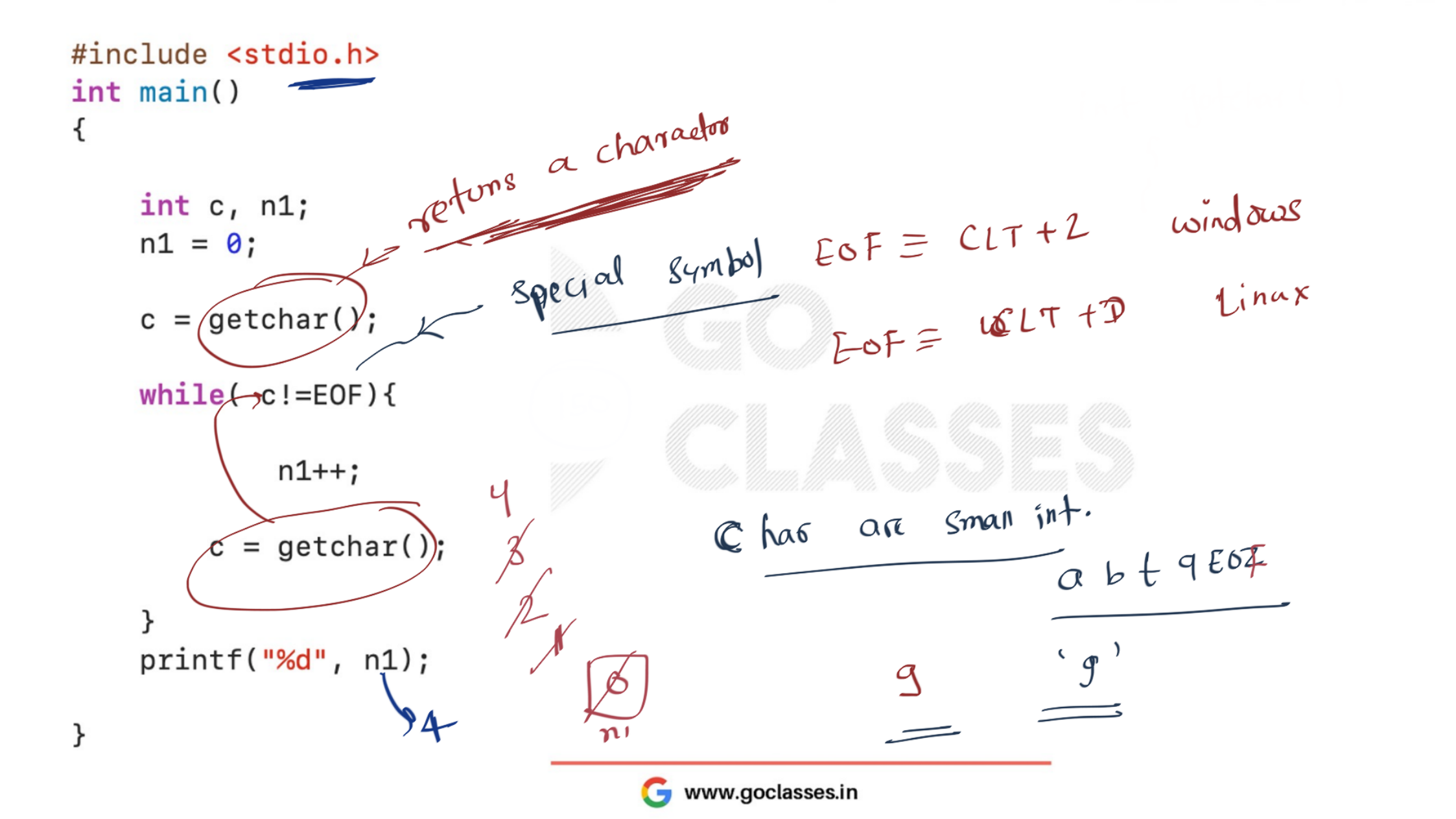
scanf is overkill just to read one character

```
c = getchar() is equivalent to scanf("%c", &c)
```

```c
#include <stdio.h>
int main()
{
    int c, n1;
    n1 = 0;

    c = getchar();

    while( c!='z'){

            n1++;
        c = getchar();

    }
    printf("%d", n1);

}
```



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

    int c, n1;
    n1 = 0;

    c = getchar();

    while(c!=EOF){

        n1++;

        c = getchar();

    }
    printf("%d", n1);

}
```



char c = 9 ✓

char d = '9'

c == d NO

Characters are small integers

putchar():

putchar () function writes a character to screen.

```
putchar(c) is equivalent to printf("%c", c);
```

```c
int main()
{

    char ch = 'a';

    putchar(ch);

}
```

a

```c
#include <stdio.h>

int main()
{

    char ch;

    ch = getchar();

    putchar(ch);

}
```

'x'

x

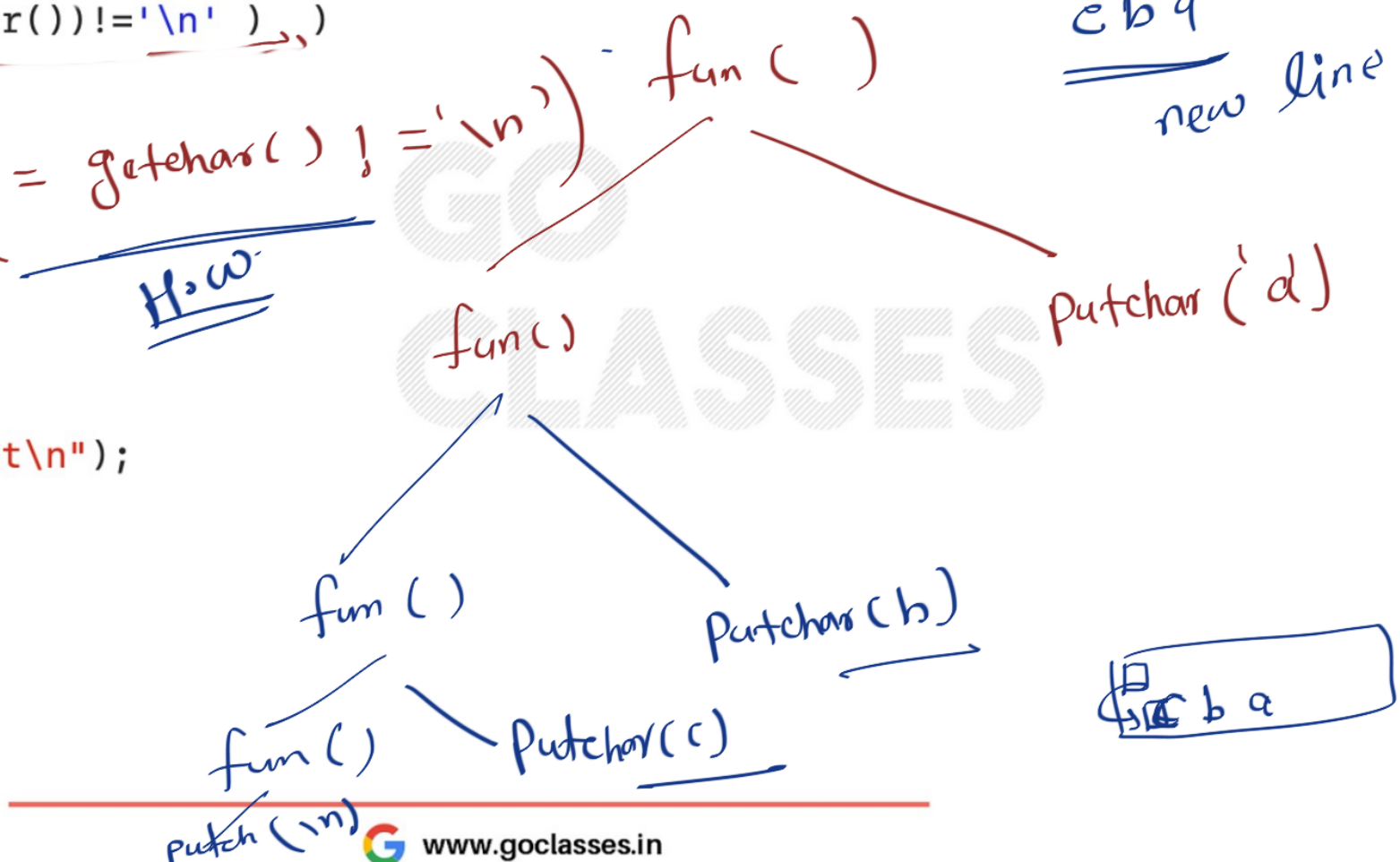
```c
void fun(void)
{
    int c;
    if(  (  (c=getchar())!='\n' ))
        fun();

    putchar(c);
}

main()
{
    printf("Enter text\n");

    fun();

}
```

What will be output for input abc<ENTER> ?

c b a in new line

if(c = getchar() != '\n')

H.W.



GATE CSE 2008

Choose the correct option to fill $?1$ and $?2$ so that the program below prints an input string in reverse order. Assume that the input string is terminated by a new line character.

```
void reverse(void)
{
    int c;
    if(?1) reverse();
    ?2
}

main()
{
    printf("Enter text");
    printf("\n");

    reverse();

    printf("\n");
}
```

A. $?1$ is $(getchar() != '\n')$

$?2$ is $getchar(c);$

B. $?1$ is $((c = getchar()) != '\n')$

$?2$ is $getchar(c);$

C. $?1$ is $(c != '\n')$

$?2$ is $putchar(c);$

D. $?1$ is $((c = getchar()) != '\n')$

$?2$ is $putchar(c);$

Macros

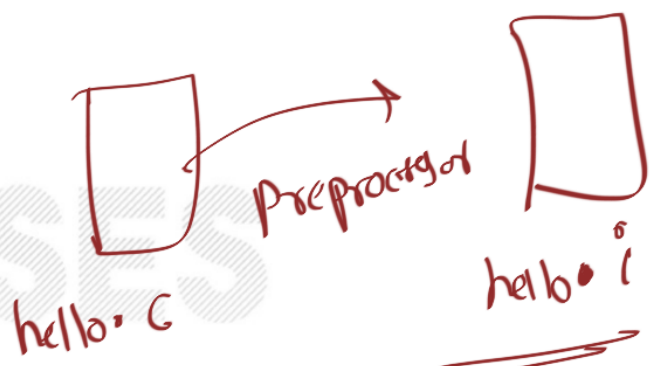
include <stdio.h>

- all lines that start with # are processed by preprocessor

```
#define MAXLINE 120

char buf[MAXLINE+1];

becomes

char buf[120+1];
```

120

after preprocessor



define fun(x) x*x

2*2

int x = fun(2)

int x = 2*2

define Plusone(x) (x+1)

int i = 3 * Plusone(2)

3 * (2+1)

int j = 3 * Plusone(5+4)

= 3 x 3

3 * 5 * 4 + 1 = 9

60 + 1 = 61

```
#define plusone(x)  x+1

i = 3*plusone(2);
```

becomes

```
i = 3*2+1
```

= 7

```c
#include<stdio.h>

#define max 100
int main()
{
    printf("max is %d", max);
    return 0;
}
```

max is 100

```c
#include <stdio.h>

#define MULTIPLY(a, b) a*b
int main()
{
    printf("%d", MULTIPLY(2+3, 3+5));
}
```

$2+3*3+5$

$= \quad 2+9+5$

$= \quad 16$

```c
#include <stdio.h>

#define MULTIPLY(a, b) (a)*(b)

int main()
{
    printf("%d", MULTIPLY(2+3, 3+5));
    return 0;
}
```

40

(2+3) * (3+5)

5 * 8

= 40

```c
#include <stdio.h>

#define MULTIPLY(a, b) (a)*(b)

int main()
{
    printf("%d", MULTIPLY(2+3, 3+5));
    return 0;
}
```

parenthesize macro parameters in definition

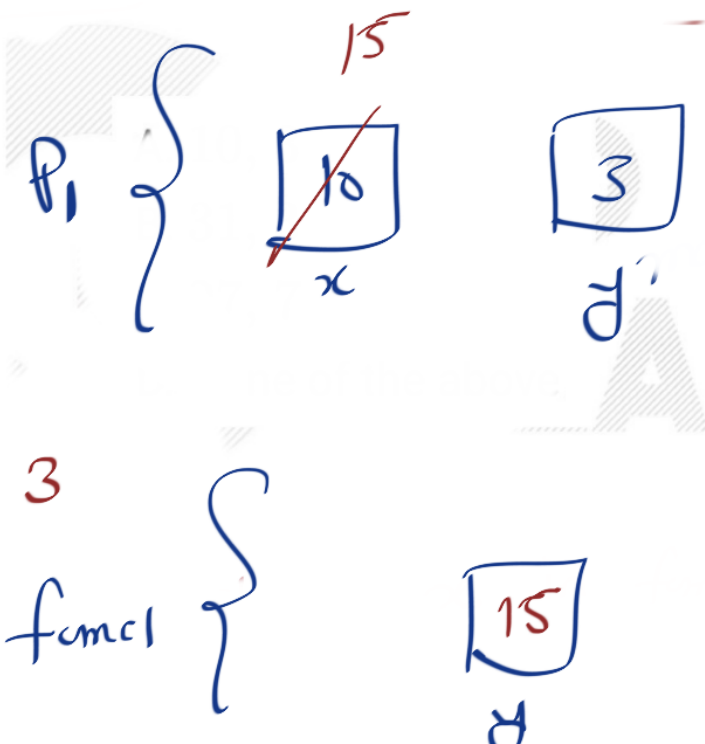
Call by reference
(NOT REQUIRED FOR GATE)

(Removed from Syllabus, Now only C programming is in syllabus)

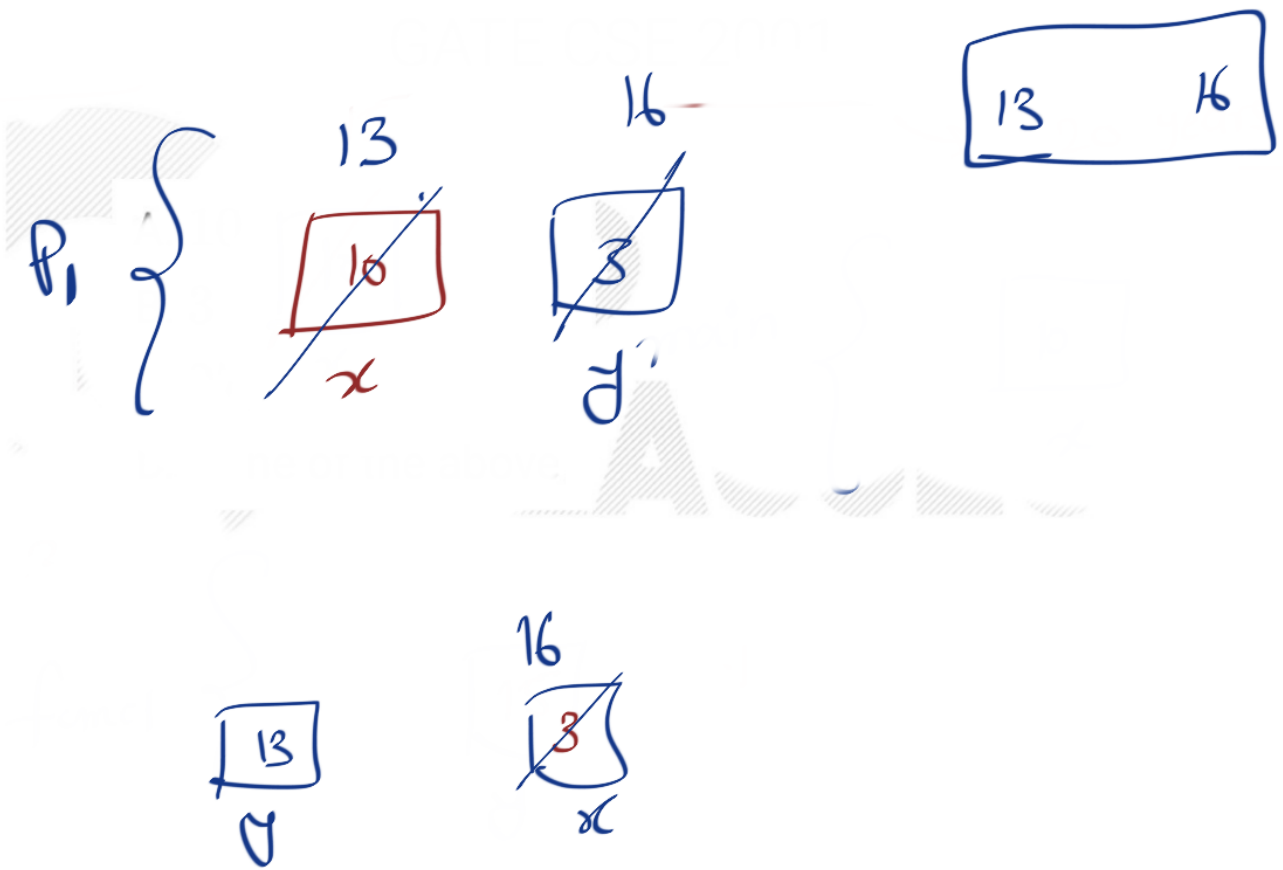
20 years back

Call by value

```
Program P1()
{
      x = 10;
      y = 3;
      fun (x)
      print (x, y)
}

func1(y)
{
    y = 15
}
```



```
Program P1()
{
      x = 10;
      y = 3;
      fun ( x, y)
       print (x,y)
}

func1( y, x )
{
      y = y + x
      x = x + y
}
```



What is printed by the print statements in the program P1 assuming call by reference parameter passing?

GATE CSE 2001

Hypothetical

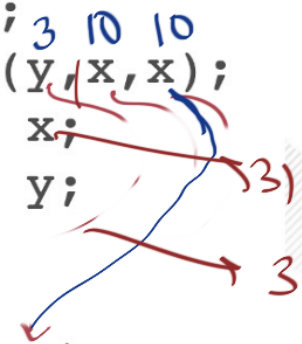
```
Program P1()
{
     x = 10;
     y = 3;
     func1(y,x,x);
     print x;
     print y;
}

func1(x,y,z)
{
     y = y + 4;
     z = x + y + z
}
```

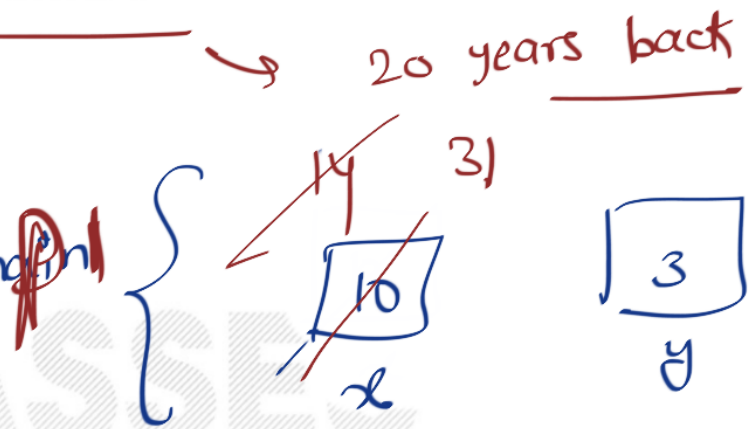
A. 10, 3

B. 31, 3

C. 27, 7

D. None of the above



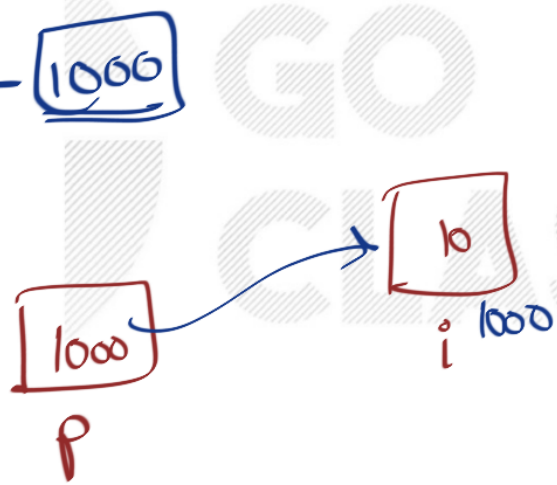


10

3 + 14 + 14 = 31

Call by value or reference ?

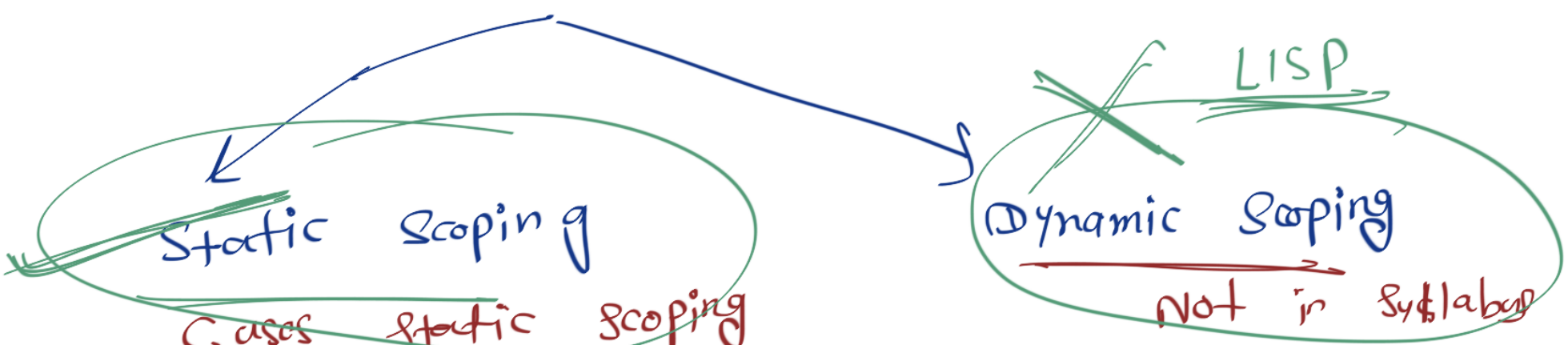
```c
#include <stdio.h>

void addOne(int *ptr) {
    *ptr = *ptr+1;
}

int main()
{
  int* p, i = 10;
  p = &i;
  addOne(p);
  printf("%d", *p); // 11
  return 0;
}
```




```
addone (int i)
{
    i = i+1
}
main ( )
{
    i = 10
    addone (i)
    Pf ( i )
}
```

Call by ref ⟶ forget

Call by value ⟶ C uses call by value only.



compile time

Static Scoping

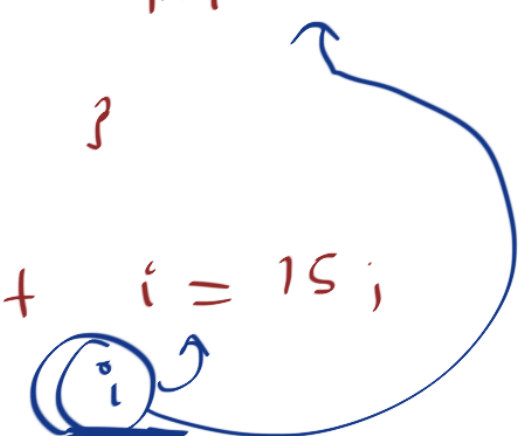
(C uses Static Scoping only)

```
int b = 5;
int foo()
{
    int a = b + 5;
    return a;
}

int bar()
{
    int b = 2;
    return foo();
}

int main()
{
    foo(); // returns 10
    bar(); // returns 10
    return 0;
}
```

Block Structure Programming

```
if ( )
{
}
```

```
main
{
    {
        int i = 10
    }
    int i = 15;
}
```

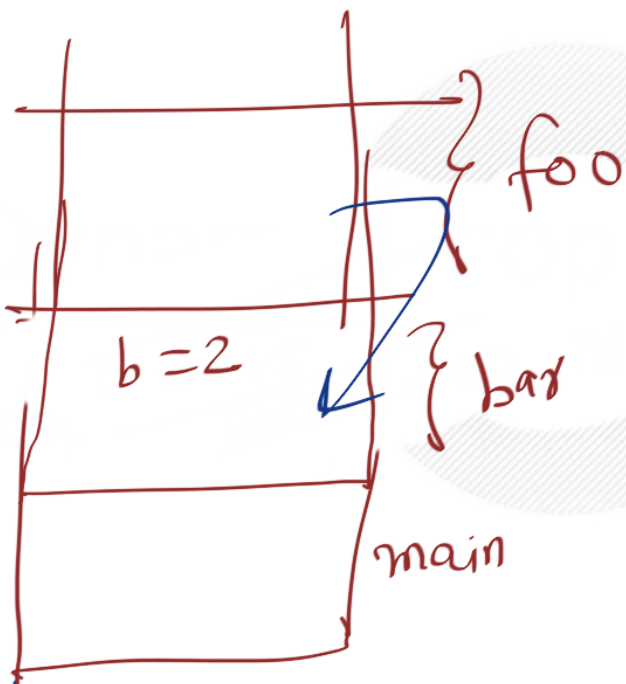


Dynamic Scoping

(C does not use dynamic Scoping only)

```
int b = 5;
int foo()
{
    int a = b + 5;
    return a;
}

int bar()
{
    int b = 2;
    return foo();
}

int main()
{
    foo(); // returns 10
    bar(); // returns 7
    return 0;
}
```

Dynamic Scoping




```
int b = 5;
int foo()
{
    int a = b + 5;
    return a;
}

int bar()
{
    int b = 2;
    return foo();
}

int main()
{
    foo(); // returns 10
    bar(); // returns 7
    return 0;
}
```

```c
#include<stdio.h>
#define a (x+1)
int x = 2;
void c()
{
    printf("%d", a);
}

void b()
{
    int x =1;
    c();
    printf("%d", a);
}


main()
{
    b();
    c();
}
```

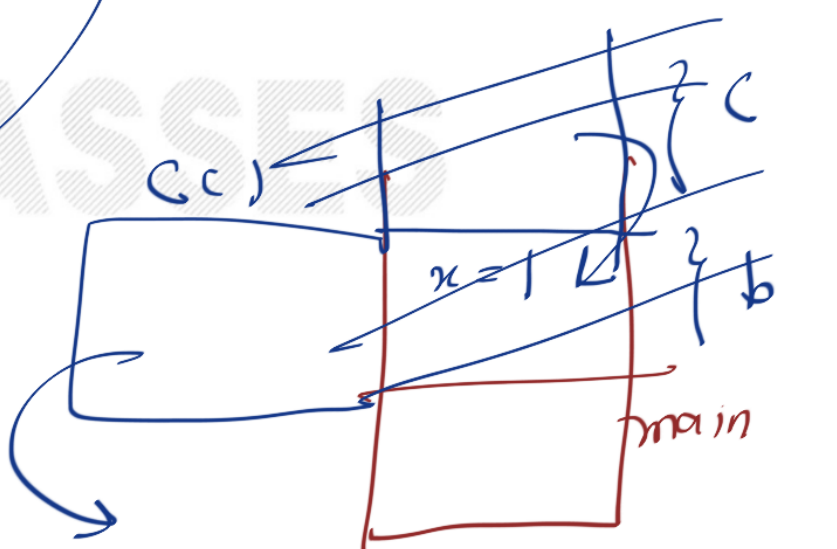
C Program

Static

3, 2

2, 2, 3

dynamic



```
int X = 0;
int Y;

void fun () {
    X++;
}

voi foo (){
    X++;
    fun();
}

main()
{
    read(y) // in C program scanf("%d", Y);
    if(Y>0){
        int X = 5;
        foo();
    }
    else foo();
    write(X) // in C program printf("%d", X);

}
```

H.W.

Static

Dynamic

Thank You