atc
1 Oct 2017

"Programming"

"Scope of a variable" :-

1. Static Scoping — Compile time (Static)
2. Dynamic Scoping — Run time (Dynamic)

:- Dynamic Type checking will take more time than static type checking.
As for Recursion, it will check again-again

:- If At Runtime variable type keep on changing there only dynamic type checking required.

:- Static type (Compile time) checking will give all types of errors. ap It scan the whole program once.

main m/y
- Static Area (global variables)
- Stack Area
- Heap Area

Ex:- Static Scoping —
Dynamic Scoping —

Ex :-

```
int a=5;
main()
{
    int a=10;
    B();
}
```

```
B()
{
    int a=15;
    C();
}
```

```
C()
{
    int a=20;
    D();
}
```

```
D()
{
    int a=25;
    Pf(a);
}
```

★ global variable m/y will allocate in the Static Area at the compile time itself

declaration:- purpose:- creating m/y.

★ Local variable m/y will allocate in the Stack Area at run time.

Scope problem:- without declaring the variable using it.

```
D()
{
    /* No variable declared as 'a' */
    Pf(a); // free variable
}
```

Solutions:-

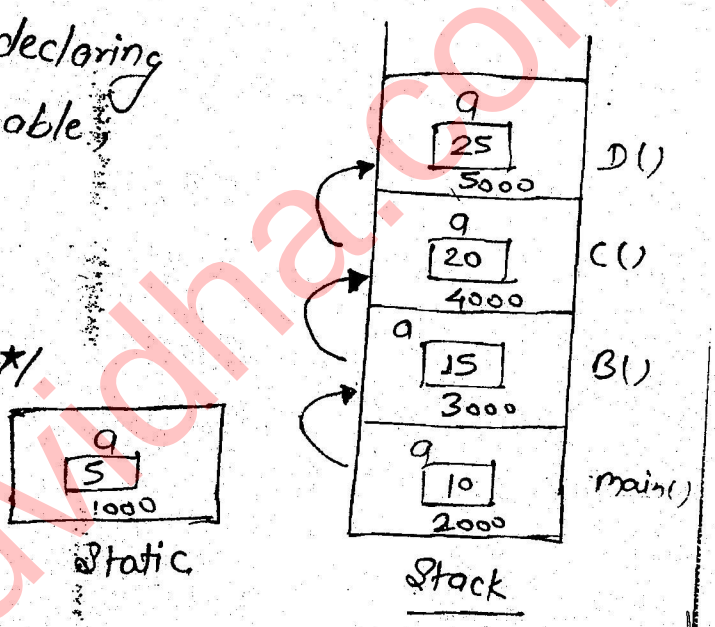
In Static Scoping, (compiler) only global variables are present at compile time so global variable printed. so o/p = 5' (global variable)

In Dynamic Scoping:- (Run time), the previous value of 'a' will be printed

a=20 → a=15 → a=10 → a=5

C B main Global

so o/p:-20' (1st previous value). (Pop op^n have to be performed)



★ free variable:-

using the variable but not declared locally i.e don't know, from where it come.

:- Static Scoping is Preferred.

By default :- Every language follows Static Scoping as Dynamic Scoping having unnecessory pop operations.

Ex : ② :-

```
int a=5, b=10;
main()
{
    int a=11, b=21;
    C();
    a=7, b=9;
    E(a);
    Pf(a,b);
}
```

```
C()
{
    Pf(a,b) // 5, 10   (Static)
    a=77, b=91;
    D();
    a=2, b=3;
}
```

```
D()
{
    int b=72;
    Pf(a); // 5
    a=4, b=3
    E(a);
}
```

```
E(int b)   b=4
{
    int a=3;
    Pf(a,b); // 3, 4
    a=4, b=7;
}
```

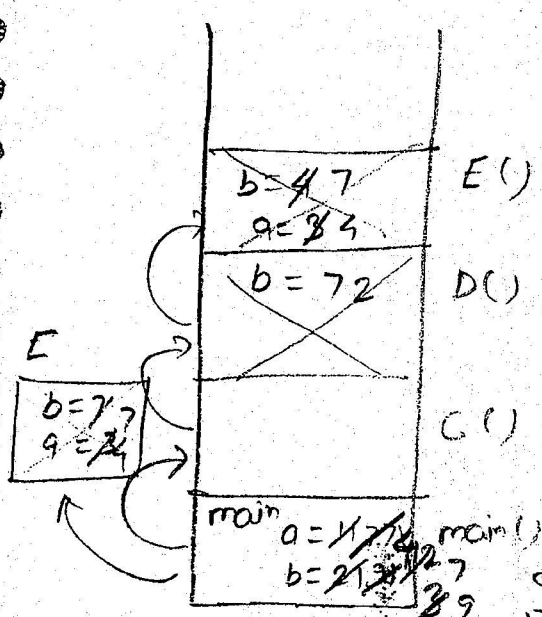
Static :- 5, 10, 77, 3, 4, 3, 7, 7, 9

5, 10, ~~5~~ 77, 3, ~~4~~ 4, 3, ~~7~~ 7, 7, 9

Dynamic :-

11, 21, 77, ~~2~~, 3, ~~4~~ 4, 3, ~~9~~ 7, 9

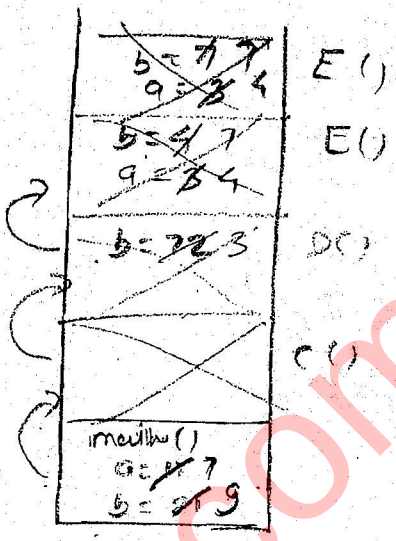
11, 21, 77, 3, 4, 3, 7, 7, 9



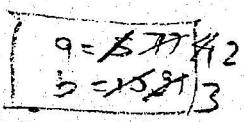
Dynamic

Static

a = 5
b = 10



Static scoping



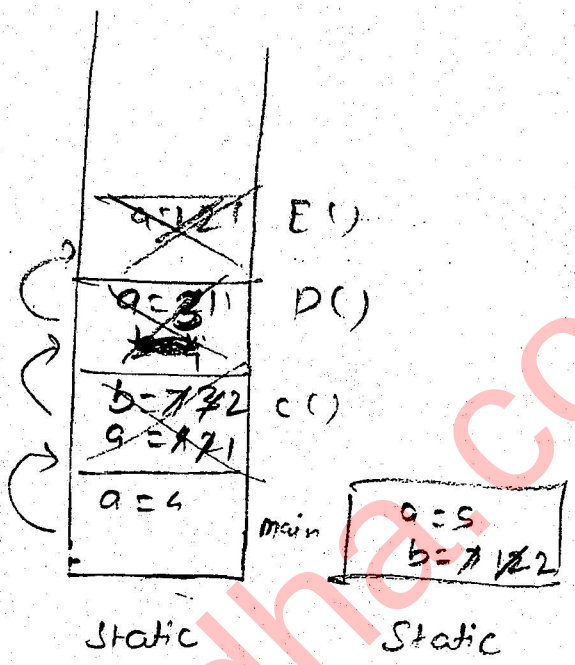
Static

Static :- 5, 10, 3, 4, 3, 7, 7, 9

Ex:- int a =

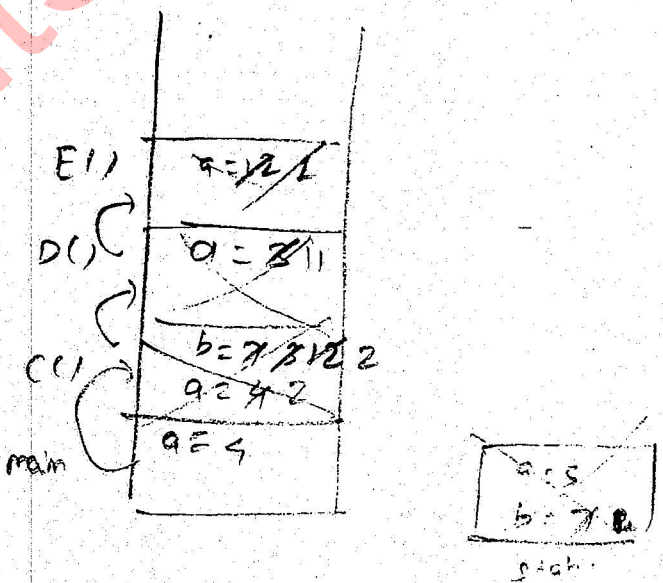
Ex: int a=5, b=7

```
main()
{
    Pf(a,b)
    a=4
    C(b,a)
    Pf(a,b)
}

C (int a, int b)
{
    a=2, b=3
    Pf(a,b);
    D(b);
    a=1, b=2;
}

D (int a)
{
    Pf(a,b)
    a=11, b=12
    E();
}

E()
{
    int a=12
    Pf(a,b)
    a=1, b=2
}
```



Static :- 5, 7, 2, 3, 3, 7, 12, 12, 4, 2

Dynamic :- 5, 7, 2, 3, 3, 3, 12, 12, 4, 7



★:- In static area, All variable values "by default 0"

```
E()
{ int a;
  Pf(a)
  a=1, b=2
}
```

/* O/P:- garbage value for both static scoping & dynamic scoping */

★★★

"Static variable":-

Ex:-

```
main()
{
  Static int var=5;
  Pf("%d", var--);
  If(var)
    main();
}
```

★ Conditional statement is true for Every Non-zero. -ve or +ve

True

Without static:-

```
main()
{
  1. var
     [5]
     1000
  2. Pf(5);
  3. If(4)
  4.   main()
       {
         1. var
            [5]
            1000
         2. Pf(5);
         3. If(4)
         4.   main()
              until "stack overflow"
```

x = 10

```
y = (x--)      y = (--x)
Pf(x,y)        Pf(x,y)
(9,10)         (9,9)
Post dec       Pre-dec
```

```
If(0.0 -> /o -> Null  (then only 0 false)
Pf("hi");
else
Pf("bye");
```

O/P:- 5,5,5 ....

with 'static'

:- because of "static" keyword, m/y for var will be created in compile time in static Area.

★ by default Local variable m/y allocation done in Stack at run time, but 'static' keyword force, to compiles to create m/y at compile time in static Area. (so single copy only)
due to "static" Processor won't create m/y.

var [5] 4 3 2 1 0
1000

```
main()
{
1.    // static, Processor skip it
2.  5
3.  if(4)   // Post decreament
4.
}
```

```
main()
{
1.
2.  4
3.  if(3)
4.
}
```

```
main()
{
1.
2.  3
3.  if(2)
4
}
```

```
main()
{
1.
2.  2
3.  if(1)
4.
}
```

```
main()
{
1.
2.  1
3.  if(0)
}
```

exit

O/P:- 5, 4, 3, 2, 1

★ Note :-

★ i) for static variable m/y will create only once because compilation also once only.

★ ii) for static variable m/y will be created in compile time in static Area.

★(iii) for static variable initialization will be done only once.

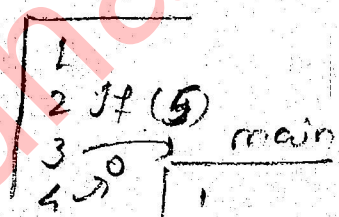
★(iv) By default static variable will be initialized = 0 (∵ static area).

Ex: ② :-

If (--var) then O/P :- 0000

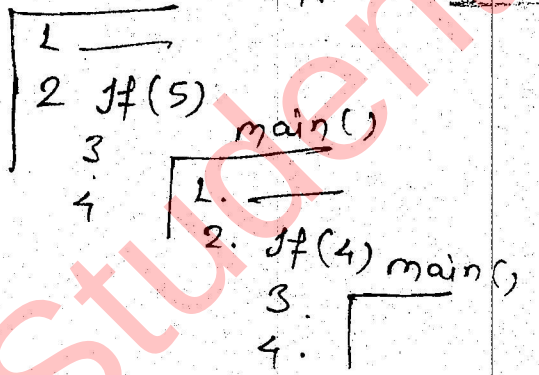
```
main()
{
  static int var=5;
  If (--var)
  {
    main();
    Pf(var);
  }
}
```

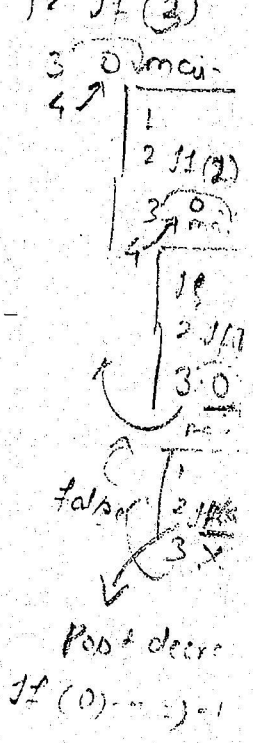
var [5] 4 3 2 1 0 -1



O/P :- ~~0000~~

O/P :- -1 -1 -1 -1 -1





Ex : (3) :-

```
void main()
{
    int x=5, y=10;
    for(i=1; i<=2; i++)
    {
        y += f(x) + g(x);   // Assignment (=) Right Associative
        pf(y);
    }
}   // No return value so, need void

f(int x)   // By default function having return type -> int
{
    int y;
    y = g(x);
    return (x+y);
}

g(int x)
{
    static int y=15;
    y++;
    return(x+y);
}
```

$a *= b + c \Rightarrow a = a * (b+c)$ { = Right associative

{ = (Assignment) Right side first

→ 26 + 22 = 48 + 10 = 58, 110

21 + 5 = 26

static: 16

11 + 5 = 21

```
main()
{
   Static int

f(int n)
{
   Static int r=5;
   If (n<=1)
   {
      r=15;
      return (1);
   } else
   If (n>3)
      return (r+ f(n-2))
   else
      return (r- f(n-1))
}
```

static [5] 15
r

I/p :- f(7)

O/p :- 31

r + f(5)

r + f(3)

r - f(2)

r - f(1)

= 31

r + f(5)

16

r + f(3)

r - f(2)

15+1

15-14 = 1

r - f(1)

r = 15

(1)

15-1 = 14

O/p : 31

a = 5

→ Pf(a++, a++, a++)

→ 5, 6, 7

← 7, 6, 5

} o/p decided by execution side (side effect)

a=5

Pf (a, a+1, a+2)

5, 6, 7

→ 5, 6, 7 ←

} side effect free

{Not Needed} for exam

:- Consider the following C Program :-

```
main ()
{
int i=-1, J=-1, k=-1, l=2, m;
m = ((i++ && J++ && k++) || (l++))
Pf( i, J, k, l, m);
}
```

0 0 0 3 1 ✗

0 0 0 2 1 ✓

Boolean Exp (0 or 1)

(or)

1 || — = 1 won't check

Short circuit

If k=0

m= ((i++ && J++ && k++) || (l++))

0(F) or 2(T)

i=0, J=0, k=1

m=1

l=3

1

0&& — =0

If i=0; m= ((i++ && J++ && k++) || (l++))

won't check

F(0) T(1)

i=1, k=-1, J=-1

```
main()
{
    int i;
    for(i=1; i<=30; i++)
    {
        switch(i)
        {
        case 1: i+=3;
        case 2: i+=5;
        case 3: i+=4;
        default: i+=3
                 break;
        }
        Pf(i);
    }
}
```

i: 1000 — 1 4 9 16 17

1+3 = 4

4+5 = 9

13

16, break

16, 19, 22, 25, 28, 31

16 17, 20, 21, 24, 25, 28, 29, 32

i++ ↓ i++ i++ i++

16, 20, 24, 29, 32

★ Note :- In switch case, After Every case break; is required, otherwise it will execute remaining statement also, eg: After default No need of break because any how it is last statement.

Date: 05/10/2017

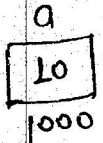
:- "Extern variable" :-

Ex: ① :-

```
int a=10;
main()
{
  extern int a;
  Pf(a);
}
```

O/P := 10

In this Program two variable
Local 'a' (extern)
global 'a'
- Pointing to same m/y of global 'a'.



/* If extern int a, this line not present then Pf(a) is a free variable so for both static scoping & dynamic scoping a=10; */

/* If extern not there then a = garbage value */

(Local variable won't take m/y).

Ex: ② :-

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

O/P :- Error

/* due to extern, no m/y created locally, check outside for variable 'a'. which is not found */

So: O/P :- Error :- undefined symbol.

Ex: ③ :-

```
main()
{
  extern int a;
  Pf("hi");
}
```

O/P :- hi

Ex: ④ :-

```
int a
main ()
{
 extern int a;
 Pf (a);
 a = 10
 Pf (a);
}
```

a
[Ø] 10

o/p:- 0 → (static area no value initialized)
10 → local variable, sharing same m/y.

Ex: ⑤ :-

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

/* No one creating m/y */

o/p:- Error : undefined symbol.

* Note :-

If extern int a (global variable) initialized, then m/y will allocate and it will be same as int a.

```
extern int a = 10;
main ()
{
   extern int a;
    Pf (a);
}
```

a
[10] (global variable)
static area.

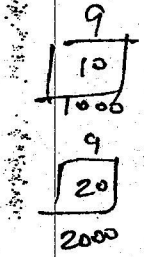
o/p:- 10

Note :- extern variable which is local, we can't initialize, so my will not create.

```
extern int a;
main()
{
  extern int a=10;
    Pf(a);
}
```

O/P:- Error : undefined symbol.

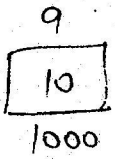
```
main()
{ int a=10;
  int a=20;
  Pf(a);
}
```



O/P:- Error : multiple declaration of 'a'

Ex: (5) :- i)

```
int a=10;
main()
{
  extern int a;
  extern int a;
  extern int a;
  extern int a;
  Pf(a);
}
```



O/P = 10

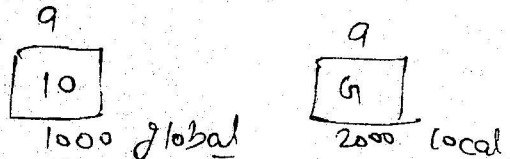
/* because only one my referenced by all local variables */

5 variables :- 4 local, 1 global

ii)

```
int a=10;
main()
{
  int a;
  extern int a;
  extern int a;
```

local ← int a;
local ← extern int a;



O/P:- Error : multiple declaration as one local variable (extern) a=10 and other

* Using 'extern' we can declare same variable many times in the same function.

Ex:

```
int a=5, b=10;
main()
{
  auto int a=15;
  static int b=7;
  C()
  Pf(a,b);
  E();
}
C()
{
  extern int a;
  Pf(a,b);
  a=3, b=4;
  D();
}
D()
{
  a=b+a;
}
static int a=27;
Pf(a,b);
E();
a=5, b=7;
}
E()
{
  extern int a;
  static int b;
  Pf(a,b);
  a=77, b=88;
}
```