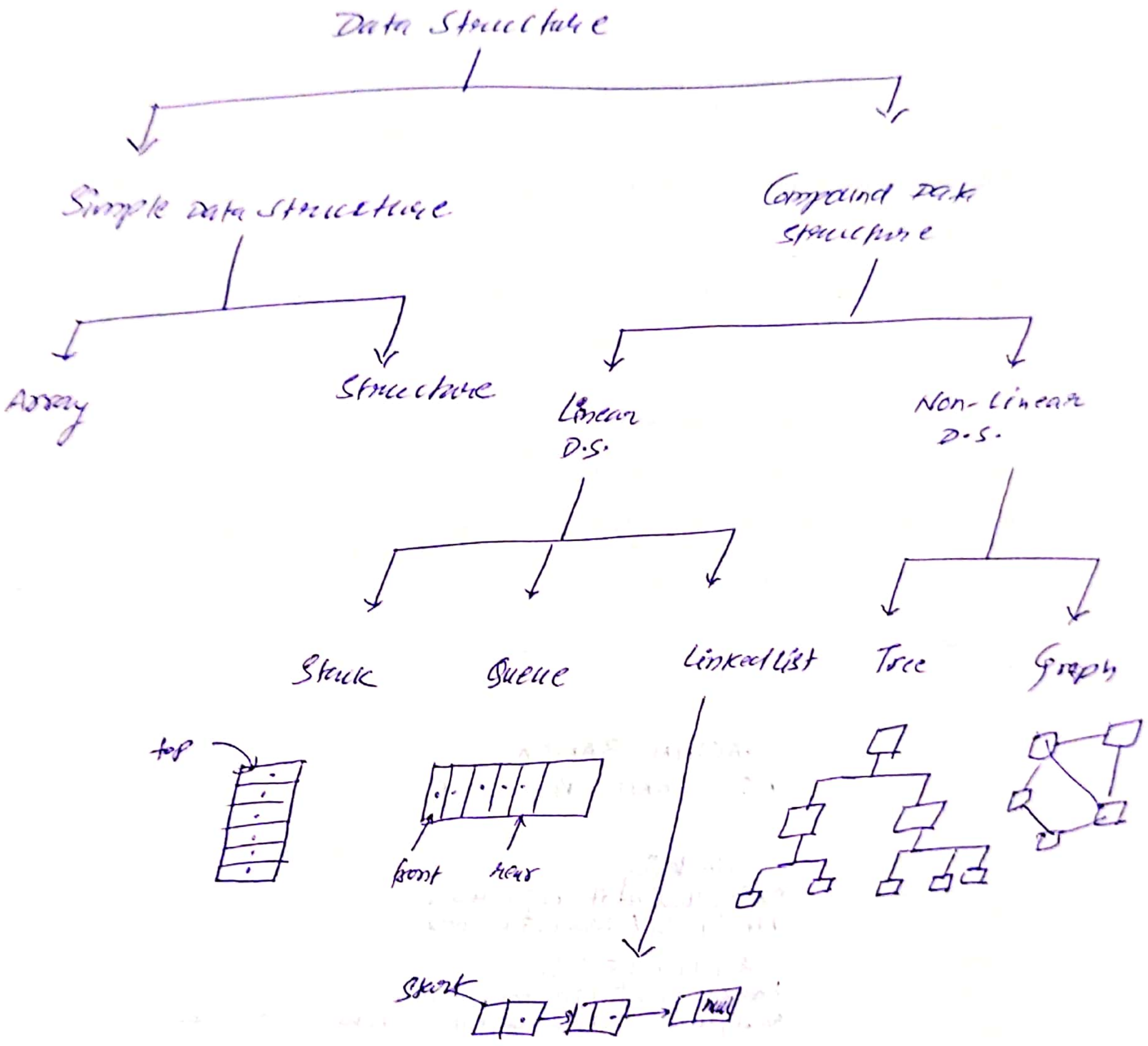

Data Structure
Simple data structure
Compound Data structure
Array
Structure
Linear D.S.
Non-Linear D.S.
Stack
Queue
Linked list
Tree
Graph
top
front
rear
start
null

:- Functions :-

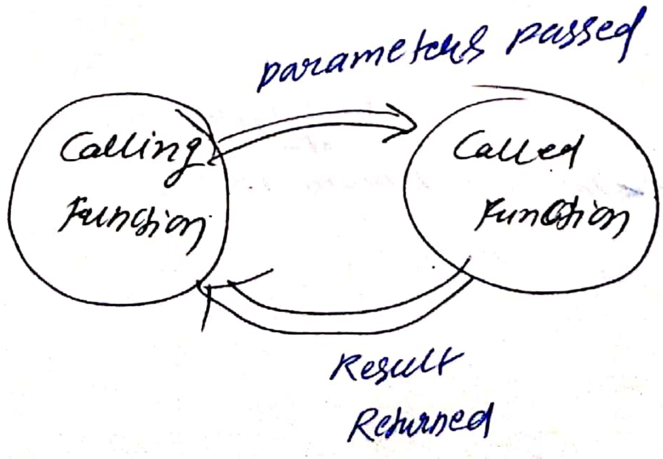
Function :-

- → set of Statements designed to perform a spicified task.
- → Types.
  - → User defined
  - → System defined (Library)
- → Three parts
  - → prototyping declaration [it is required when ~~function~~ ~~calling~~ function defined ~~Before~~ After Calling]
  - → Function defination
  - → function Calling.

Syntax :-

```
led  returntype   function name (datatype var, datatype var)
tion {
         ≡
      return (var);
     }                     both should be same    // Function Defination

     →void xyz()
ling   {
tion    ≡
        z = function(x,y);   // function Call
       }
```



On the Basis of return value & parameter passing, we can Categorize user defined functions in four categories

Types of user defined functions:-

① No parameters passed and no return value

e.g.

```
void sum ( )        // defination of function
  {
    =
  }
main ( )
 {
   =
   sum ( );          // calling function
   =
 }
```

② No-parameters passed but a return value

③ Parameters passed but no-return value

④ parameters passed and a return value

$\Rightarrow$ By default Return $\hookrightarrow \boxed{int}$

→ you can write multiple "return" in a called Function

↳ But only 1 return is executed

→
```
int sum(int a, int b)
{
  int c;
  c=a+b;
  return c;
}
void main()
{
  int x,y;
  scanf("%d%d", &x, &y);
  printf("%d", sum(x,y));
}
```
`int a, int b` → Formal parameters.

`c` ↔ `x,y` → Actual parameters

`sum(x,y)` → Function

To write "Function" after the block in which it is called

↳ [we need to tell specific details about function to compiler before using function.

Come into the picture ← → Function prototype:-

returntype funcname (parameters);

No & type of

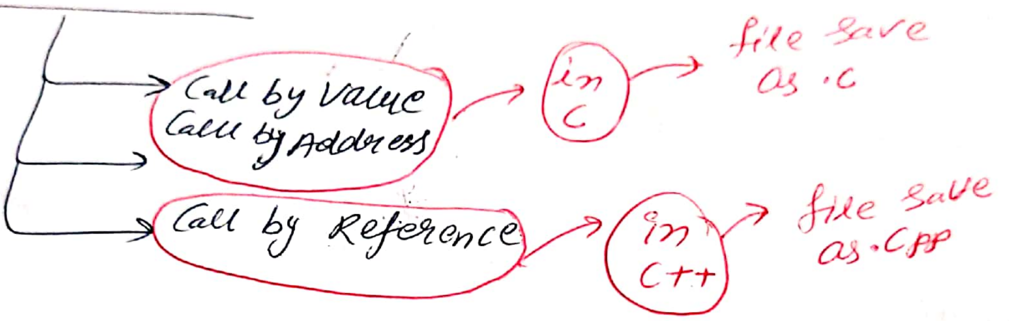
Q; what error will in the following function give on compilation -

```
int f( int a, int b)
{
    int a;
    a = 20;
    return a;
}
```

Compile time error.

Redeclaration of a

passing parameters :-



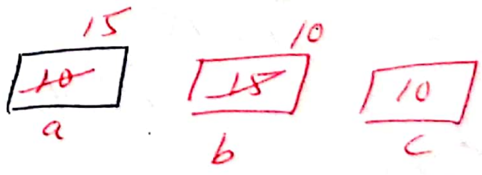
Call by Value :-

- Only one value be returned
- passes the value from one block to another
- In Call by Value we ~~can~~ Can pass more then one parameters But can return only one value.
  - Call by Address

Come into picture

eg

```
void swap(int a, int b)
{
    int c;
    c = a;
    a = b;
    b = c;
    Printf("In swap a=%d , b=%d", a, b);
}
```



```
void main()
{
  int x=10, y=15;
   printf("Before swap x=%d ,y= %d", x,y);
   swap(x,y);
   prinf("After swap x= %d ,y= %d", x,y);
 }
```

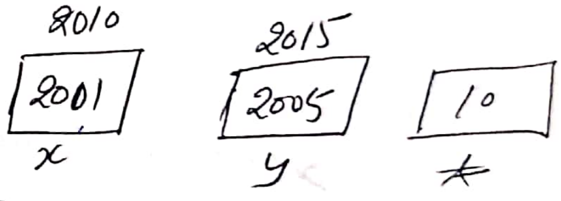
O/P

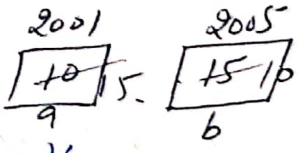
```
Before swap x=10 ,y=15
In Swap a=15, b=10
After swap x=10 ,y=15
```

Call by Address :-

```
void Swap( int *x , int *y)
 {
  int t ;
   t = *x; → *(&a) → *(2001) = 10,
   *x = *y;
   *y = t;
  printf("%d %d", *x, *y);
 }
void main()
  {
   int a= 10 ,b=15;
   printf ("Before swap %d %d", a,b);
   swap( &a ,&b);
   printf("After swap a=%d ,b=%d", a,b);
   getch();
  }
```

value at address





O/P

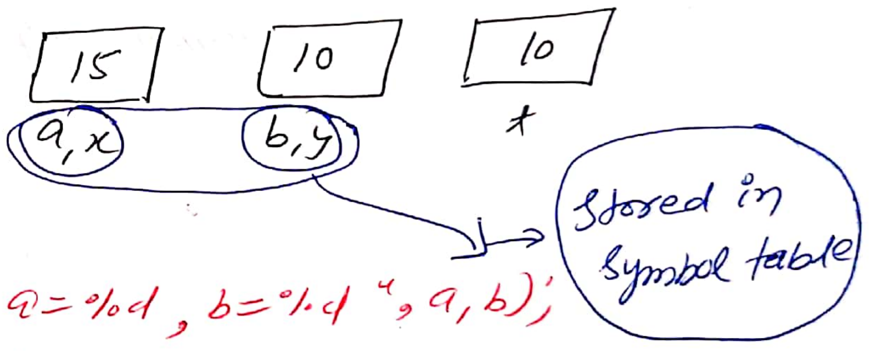
```
a=10 ,b=15 (Before Swaping)
10 ,15
a=15, b=10 (After swaping)
```

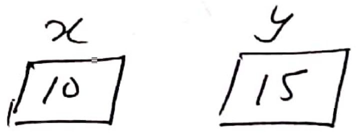
Call by Reference:-

- → we pass Reference from Actual to formal parameters.
- → It is more Benifial then ~~add~~ call by Address
  - → Reason → Swaping is Done without Creating any extra space (pointer)
- → Call by Reference has same power as call by Address but it is more efficient.

e.g

```
void swap(int &a, int &b)
 {
   int t;
   t=a;
   a=b;
   b=t;
   printf("in swap a=%d, b=%d", a, b);
 }
void main()
{
 int x=10, y=15;
 printf("Before swap x=%d, y=%d", x, y);
 swap(x,y);
 printf("After swap x=%d, y=%d", x, y);
}
```





output:-

```
Before Swaping x=10, y=15
in Swap a=15, b=10;
After swap x=15, y=10
```

V. Imp.

Recursion :-

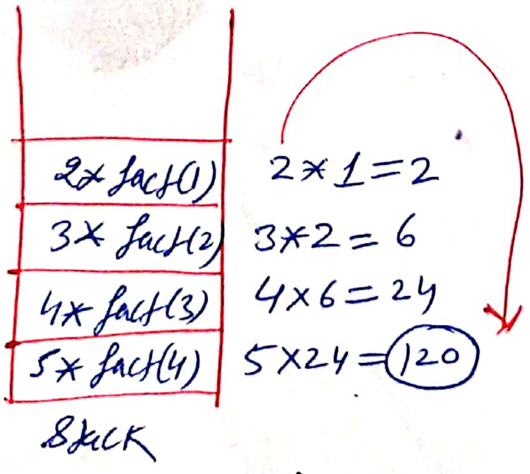
- → Function calling itself
- → Necessary Condition
  - → ⓐ function should call itself
  - → ⓑ there should be a terminating condition.
- → main() can also be recursive.

e.g ✓

```
int factorial (int x)
{
  if (x == 1)  // terminating condition
      return 1;
  else
    return x * factorial (x-1);   → calling itself.
}
```

calling fact(5) $\xrightarrow{O/P}$ 120

Method 1



Note : in Recursion
↳ use Stack
↓
All statements after function re-called are
⇓
pushed into stack in order
↓
so that it is executed in same order

These are utilized by compiler on terminating of Recursion ethod 2

factorial (5); or fact(5) = ?

fact(1) → 1

fact(2) → 2 × fact(1) → 2 × 1 = 2.

fact(3) → 3 × fact(2) → 3 × 2 = 6

fact(4) → 4 × fact(3) → 4 × 6 = 24

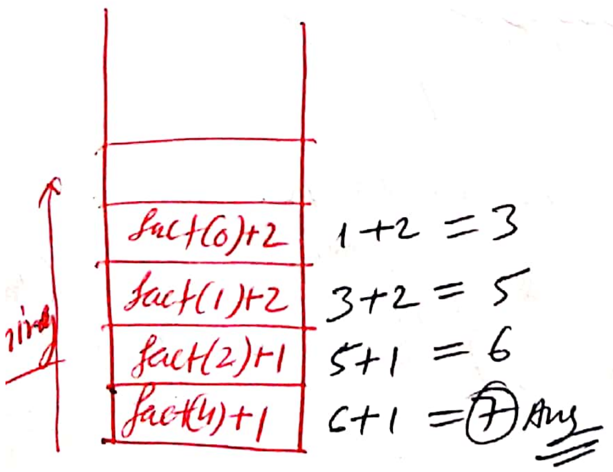
fact(5) → 5 × fact(4) → 5 × 24 = 120 Ans

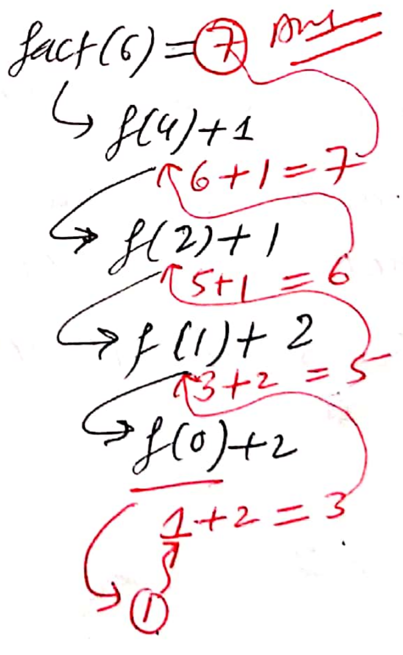
✓ → give to solve Student.

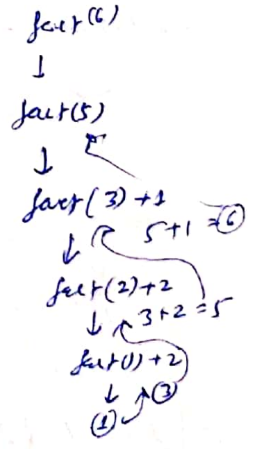
Q:
```
int fact(int n)
{
    if (n<=0)
        return 1;
    if (n>3)
        return fact(n-2)+1;
    else
        return fact(n-1)+2;
}
```
→ O/P → $\boxed{7}$

Show what happen on Calling fact(6); → Solve by Both methods ↓ next

ethod 1







method 2

$fact(0) = 1$

$fact(1) = fact(0) + 2 = 1+2 = 3$

$fact(2) = fact(1) + 2 = 3+2 = 5$

$fact(3) = fact(2) + 2 = 7$

$fact(4) = \underline{fact(2)} + 1 = 5+1 = \textcircled{6}$

$fact(5) = fact(3) + 1 = 7+\cancel{1} = 8$

$fact(6) = fact(4) + 1 = 6+1 = \textcircled{7}$ Ans

Gate Q.

```
int f(int n, int sum)
{
  int K=0, j=0;
   if (n==0)
      return;   or→ return(0)  (Same things)
   K=n%10;
   j=n/10;
   sum = sum+K;
   f(J,sum);
   printf("%d",K);
 }
int main().
 {
  int a=2048, sum=0;
  f(a,sum);
  printf("%d\n",sum);
 }
```

(a) 8 4 0 2 14

(b) 8 4 0 2 0

(c) 2 0 4 8 14

[✓ (d)] 2 0 4 8 0

Gate
58.

```
int f(int n)
{
  static int r=0
   if (n<=0)
     return 1;
    if (n>3)
    {
     r=n;
     return (f(n-2)+2);
    }
  else return (f(n-1)+r);
}
```

→ return (r + f(n-1));

f(5) — 18
r = ~~0~~ 5
↓
f(3)+2 — 18
↓
5 + f(2) — 16
↓
5+f(1) = 11
↓
5+f(0) = 6
1

↓
18

what is the value of f(5)

(a) 5 (c) 9
(b) 7 [✓ (d) 18]

Sol^n

f(5) → 18
r=5
↳ f(3)+2 = 18
↓
f(2)+5 = 16
↓
f(1)+5 = 11
↓
f(0)+5 = 6
↓
1

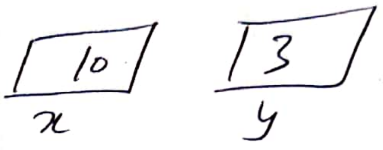
$\boxed{\cancel{0}\ 5}$
r.

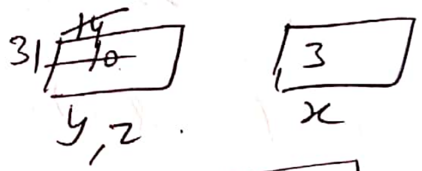
f(5)
$\boxed{r=5}$
f(3) + 2   13
= 5 + f(1) +   11
↓
5 + f(0)   6
1

Call by Reference example -

Q: What is var printed by the print statement in program P1 assuming call by Reference parameter passing.

```
program P1()
{
    x=10;
    y=3
    func1(y,x,x);
      printf x;
      printf y;
}
func1 (x,y,z)
 {
   y=y+4;
   z= x+y+z;
 }
```

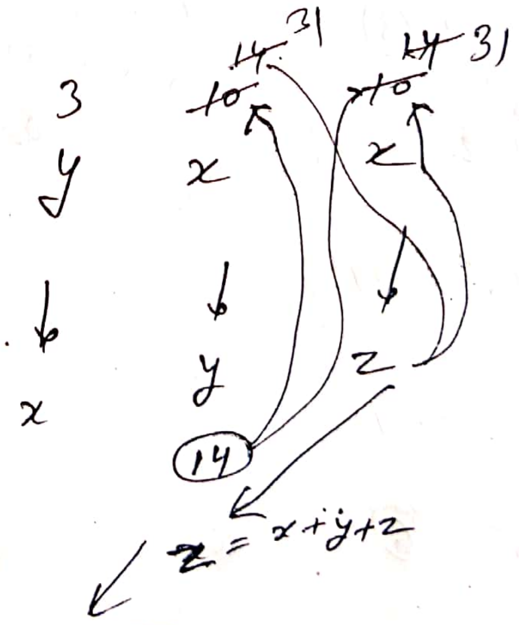




$\boxed{x=31, y=3}$

(a) 10,3 (b) 31,3

(c) 27,7 (d) None of these



opp

31, 3