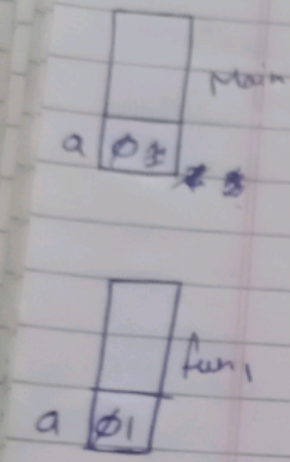
* Static variable :-

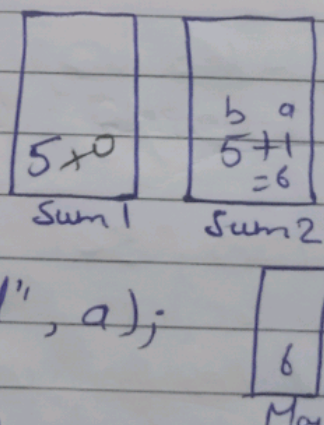
⇒
```
int fun()
{
static int a = 0;
a++;
return (a);
}
void main()
{
int a;
a = fun();
a = fun();
a = fun();
printf("%d", a);
}
```

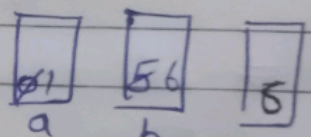


Output : 3

⇒
```
int fun()
{
int a;
a = 0;
a++;
printf("%d", a);
}
```

⇒
```
int sum()
{
static int a = 0;
int b = 5
int s;
s = a+b; a++; b++; return s;
}
```





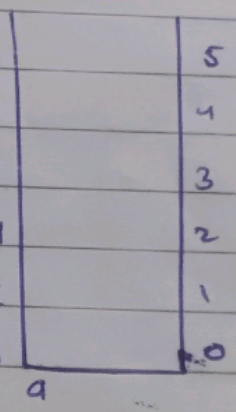

```
void main()
{ int a;
a=6 fun();
Main=6
a=6 fun();
a=sum();
a=sum();
printf("%d",a);
}
```

Output : 116

⇒
```
void fun()
{
 int a;
 static int b;
 printf("a=%d\n", a);
 printf("static b=%d", b);
}
void main()
{
 fun();
}
```

Array

Array is used to store the same type of data. It provide us memory by same name.



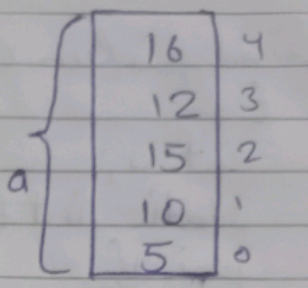
How to declare an array

DATA_TYPE NAME of ARRAY [SIZE];

e.g int a[10];
float a[20];

initialization ? int a[10] = {1,2,3,4,5,6,7,8,9,10}
int a[] = {1,2,3,4,5}; ✓
int a[];
a= 1,2,3,4,5;] ✗

```
a[0] = 5;
a[1] = 10;
a[2] = 15;
a[4] = 16;
a[3] = 12;
```



1-D Array

⇒
```
int a[5];
int i;
for (i=0; i<5; i++)
{
  printf ("enter the value of %d", i);
          array index number");
  scanf ("%d", &a[i]);
}
for (i=0; i<5; i++)
{
  printf ("%d %d", i, a[i]);
}
```

2-D Array

```
        R  C
int a[3][3];
```

⇒
```
# include <stdio.h>
int main ()
{
int j, i, a;
a[j][i];
for (j=0; j<3; j++)
```

```
{
 for (i=0; i<3 ; i++)
  {
   printf (" enter the element");
   scanf ("%d", & a[i][j]);
  }
 }
```

⇒ linear eqn

55	13 a(0)	25 a(1)	4 a(2)	5 a(3)	11 a(4)	12 a(5)

```
# include <stdio.h>
 void main()
 {
  int a[5];
  for (i=0 , i<5; i++)
   {
    printf (" enter the array");
    scanf ("%d", &a[i]);
   }
   printf (" enter the item that you
            want to search");
   scanf ("%d", & item);
    for (i=0, i<5 , i++)
     {
      if (a[i] = item)
      {
       printf (" item found");
```

```
printf ("position %d", i+1)
    break;
}
if (i>=5)
{
printf ("i is not found");
}
```

★ string :- set of characters.

How to declare a string :-

To implement a string we have to use character type of array.

char a[50];

Datatype — Name of string array.

declaration + Initialization

char a[] = "Hello";

'\0' - Null char. size of array = 6.

⇒ char a[50] = "Hello";

★ strlen
Size of (a)

★ strcpy (a, b) ⇒ string copy

★ strcmp ⇒ string compare

```
int c;
c = strcmp(a, b);
```

★ strcat ⇒ string concatinate

```
c = strcat(a, b);
    Destination  source
```

⇒ **structure** ⇒ when full memory.

We can store diff. types of data.

Syntax:-

```
struct structureName {
        int ---
        float ---
        char ---
};
```

```
struct student
{
4byte — int age;
10byte — char name[10];
} s1, s2, s3;
```

name age name age name age

```
# Void main ()
  {
    int a;
struct student s1, s2, s3;
```

struct student Name Variable Name;

⇒ store data in structure variable.

Variable Name . Member name

```
S1.age = 18;
printf ("%d", S1.age);
#include <string.h>
struct student {
    int age;
    char name[10];
} S1;
void main()
{ struct student S2;
  S1.age = 18;
  scanf ("%d", &S2.age);
  strcpy (S1.Name, "Xyz");
  printf ("enter the Name for S2");
    gets (S2.name);
  printf ("age of S1 %d", S1.age);
  printf ("Name = ");
   puts (S1.Name);
   printf ("age of S2 %d", S2.age);
   printf ("Name = ");
   puts (S2.name);
}
```

* struct → Keyword; student → structure name; S1 → declaration of structure stored as Student variable or Student type variable

⇒ structure is a user defined datatype. Structure is a collection of different type of variable.

for e.g struct student {
int age; 2 byte
char name [20]; 10 byte
};
12 bytes

Total memory ~~of~~ for above defined structure is 12 byte.

UNION → when less mem.

UNION Example :- {
int a; [2/4 byte]
char b; [1 byte]
} e1, e2;

Union is a user defined datatype which can accept diff. kind of data for e.g. int, float, char.

Declaration Syntax for union is :-

* Union Name_of_union {
member 1;
member 2;
:
} variable_1, variable_2;

e.g. above

⇒
```
# include <stdio.h>
  void main()
  union example {
      int age;
      char b;
      } ;
  void main()
  {
    union example e1, e2, e3;
    e1.age = 10;
    e1.b = 'z';
    printf ("%d\n", e1.age);
    printf ("%d", e1.b);
```

* Pointer

*→ D referencing pointer
Data_type* Name of the variable;

How to initialize → p = &a;

```
# include <stdio.h>
  void main()
  {
  int a;
  int *ptr;
  ptr = &a;
  a = 10;
  *ptr = 20;
  printf ("%d\n", &a); printf ("%d", ptr); }
```

Print the values of & adress of two variable

Pointer

```
# include <stdio.h>
void main ()
{
int a, b; int *p = &b;
a = 13;
b = 14;
*p = 50;
printf ("%d", a);
printf ("%d", &b);
printf ("%d", &a);
printf ("%d", p);
} printf ("%d %d %d", a, *p, b);
```

a = 1024
b = 1028
p = 1032

13 1028 1024 1028 13 50 50

Write a program → Pointer arithmetic

```
int a[4];
ptr = &a;
printf ("%d", &a);
for (i=0; i<4; i++)
  {
    printf ("%d", *ptr);
      ptr++;
  }
```

```
# include <stdio.h>
void swap ()
{
int a, b, c;
printf ("enter the value of a, b);
scanf ("%d %d", &a, &b);
```

(Call by reference)

```
c=a ; a=b ; b=c;
printf("%d\n %d\n", a, b);
}
  void main ()
{
  swap ();
}
```

⇒ Call by reference

```
void main()
{
  int a, b;
  a=10; b=20;
  swap (&a, &b);
  printf ("a =%d b =%d", a, b);
}
  void swap (int*a, int*b)
{
  int c;
  c=*a;
  *a = *b;
  *b =c;
}
```

Recursion

A function calls itself in recursion

e.g.
```
void sum()
{
sum();
}
```

Termination condition is mandatory for recursion.

Stack :- LIFO (Last in first out)
FILO (First in last out)

Stack data structure is used by recursion function call.

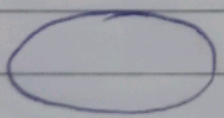
Recursion :-
```
#includ <stdio.h>
void fact (int n)
{
 if (n>=1)
{
 return n * fact(n-1);
}
 else
 {
 return 1;
 }
 }
void main()
{
int n;
```

```
printf ("enter the value of n");
scanf ("%d", &n);
printf ("%d", sum(n));
```

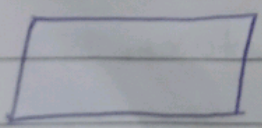
Flowchart :-

① A flowchart provides a graphical representation of a program.
② It represents the set of seq. steps that a program must follow to obtain the desired o/p.
③ It helps programmer and the non-pro. to understand the control flow of a program.

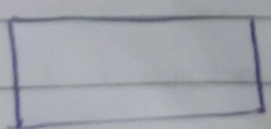
The symbols which are used to design the flowchart are given below:-



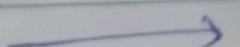
Start | End



I/P & O/P



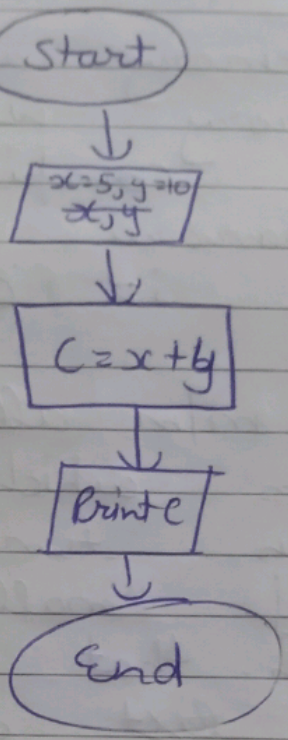
Process



Direction of flow

Decision or condition

Flowchart for add 2 variable are given below :-



★ Dynamic Memory Allocation

- Static Memory allocation is a type of memory which is allocated at the compile time for e.g. memory allocation to array in C.

- Dynamic m/m allocation is a type of m/m which is allocated at run time with the help of 'malloc' and 'Calloc' function. that is required for creating list data structure M/m is allocated in heap
Data str.

⇒ Malloc :- It is used to allocate a single block of memory at run time. Syntax :-

```
void * malloc (size of (Data type));
int * p = (int*) (malloc (size of (int)))
```

⇒ Calloc :- is used to assign multiple blocks of memory at run time.

Syntax :

```
void* Calloc (n, size of (Data type))
              ↓
           no. of blocks
```