Enumerator

by using it we define data type.

```
enum month
{
    jan, feb, mar, apr, may, jun,
    jul, aug, sep, oct, nov, dec
};
main()
{
    enum month m1;
    .....
    if (m1 == apr)
    {
    }
}
```

m1 have integer value.

by using enumerator we can easily compare by writing name.

here jan = 0 and feb = 2, mar = 3 .....

if we want it 1 so we can initialize jan = 1 and other will be automatically +1.

jan = 1, feb = 1, ...

if you want you can assign in between aug = 2 so sep = 28, oct = 29.....

```
enum weekdays
{
    mon, tues, Wed, Thu, Sat, Sun
};
main()
{ enum weekdays w1;
  if (w1 == mon)
}
```

len is a synonyme of int

typedef is a keyword. it can rename any datatype it gives neco / alternative name to data type

```
typedef int len;
main()
{
  len x=4, y=5;
  printf("%d %d", x, y);
  getch();
}
```

datatype at place of int

```
struct Book
{
  int bookid;
  char title[20];
  float price;
};
main()
{
  struct Book b1;
}
```

```
typedef int struct
{
  int bookid;
  char title[20];
  float price;
} Book;
main()
{
  Book b1;
}
```

→ at place of len

Storage classes
- Automatic
- Register
- Static
- External

basically हम जो variable बनाते हैं तो इनमे से किसी एक catagry को belong करता है

Storage classes

	Keyword	Storage	default	Scope	life
Automatic	auto	RAM	Garbage	Limited to the block in which it is declared	Till the end execution of the block in which it is declared
Register	register	CPU registers	Garbage	same -11-	same -11-
Static	static	RAM	0	same -11-	Till the execution of program
External	extern	RAM	0	whole program	same -11-

```
main()
{
  int x;
}
```
→ means
```
main()
{
  auto int x;
}
```

```
Void f1()
{
  auto int x=5;
  printf("%d", x);
  { auto int y=10;
    print("%d %d", x, y);
  }
  printf("%d %d", x, y);
  getch();          // error
}
main()
{ f1();
  getch();
```

```
Void f1()
{ auto int x=5;
  printf("%d", x);
  { auto int y=10;
    printf("%d %d", x, y);
  }
  printf("%d", x);
  getch();          // scope and life tim
}
main()
{ f1();
  getch();
}
```

```
void f1()
{
auto int x=5;
printf("%d", x);
{
 auto int x =10;
 printf("\n%d", x);
}
 printf("\n %d", x);
}
main()
{ f1();
 getch();
}
```

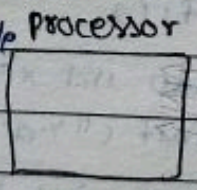
in this situation printf gives more preferance to more local variable.

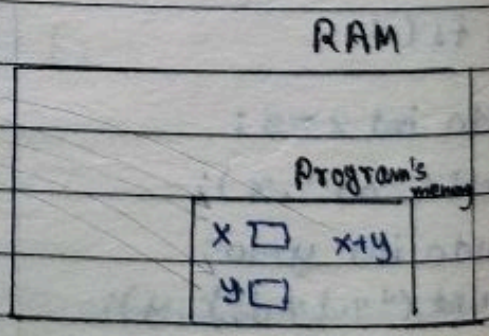
for e.g. you are in india and in mp too.
but firstly you are in MP.

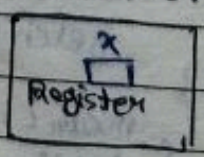
6.

Register

2. if we have give memory to x in RAM then processor will take long time for accessing variable that is in RAM.
for increasing speed if we are using any veriable oftenly the we will give memory to that variable in processor register



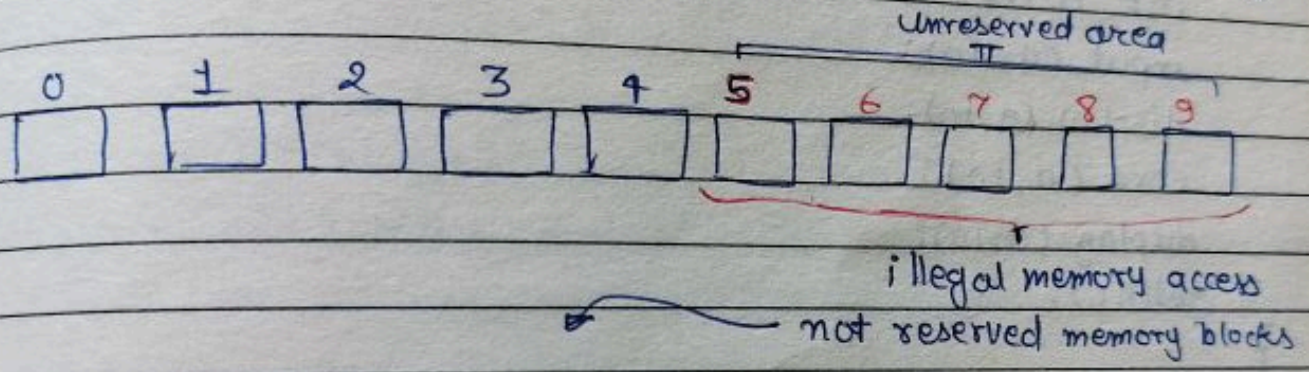




8. int a[5], i;

```
for (i=0; i<=5; i++)
scanf("%d", &a[i]);
```

// no Error - because in C and C++ there is no cheaking of array's uperbound



It may caus of - crash of program
you shadd Never ever made this mistake as a C programmer

9. int a[] = {10, 20, 30, 40, 50}

Array and Functions

1 if we want to pass the array as an argument in any function called.

2
```
main()
{
    int a[10];
    input(a,10);
    display(a,10);
    sort(a,10);
3   display(a,10);
    getch();
}

4 void input(int b[], int size)
{   int i;
    printf("Enter %d numbers", size);
5   for(i=0; i<=size-1; i++)
    scanf("%d", &b[i]);
6 }

void display(int b[], int size)
{ int i;
7 for(i=0; i<=size-1; i++)
  printf("%d", b[i]);
}
```

```
void sort (int b[], int size)
{  int  i, r, t;

for( r=1 ; r<=size-1 ; r++)

{ for ( i=0; i<=size-r-1 ; i++)
  if ( b[i] > b[i+1])
  { t = b[i];
    b[i] = b[i+1];
    b[i+1] = t;
  }
}
}
```

Array and Functions direction to mesure is dimension

if we want to pass the array as an argument in any function called.

Two dimensional array

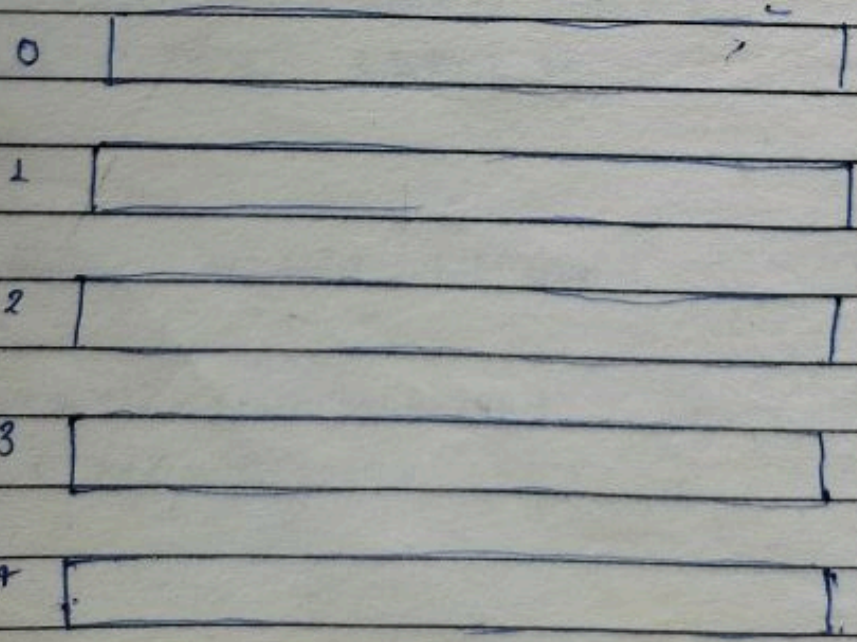
Imagin we have to store data of 5 classes each class have 10 students and we want to stor their marks

if their is one class and 10 in student in that class then we will store the data in one dimentional array a[7]

but in given condition if we store data in one dimentional array it will get dificult to find the required index number

int a[50] (X)

So logical view 0 1 2 3 4 5 6 7 8 9



2)

```
struct Book ;
{  int bookid ;
   char title[20] ;
   float price ;
};
void input ( struct Book * );
void display ( struct Book );
main()
{
   struct Book b1 = { 1 , "Drilling C" , 450.0 };
   struct Book b2 , b3 ;
   struct Book
   b2. bookid = 2
   Strcpy ( b2. title , "Java");
   b2. price = 300.0 ;
  input ( &b3 );
  display (b3);
  display (b2);
  display ( b1 );
} getch();
  void input ( struct Book * p )
{ printf ( "Enter bookid , title and price );
  scanf ( " %d" , & p -> bookid );
  fflush ( stdin );
  gets ( p -> title );
} scanf ( " %f " , & p -> price );
Void display ( struct Book b )
{
  printf ( " %d %s %f ", b.bookid , b.title , b.price );
}
```

Date: / / Page No.:

SMA	DMA
Static memory Allocation	Dynamic memory Allocation
int x; float y; int *p; char str[10]; struct Book b1; अगर हमने declaration statment लिखकर variable बनाया है, तो वो SMA वाला variable कहलाएगा।	malloc() → predefined functions calloc() इन functions को कॉल करने से variable बनता है। तो इन्हे call करने से जो variable बनेगा वो DMA वाला variable कहलाएगा।
SMA वाले variables का कुछ नाम होता है।	DMA वाले variable का कोई नाम नही होता। नाम नही है पर adress तो है, adress को हम किसी pointer मे रख लेंगे और इस pointer की मदद से उस variable को access कर लेंगे।
SMA वाले variable के बारे मे कई सारे characterstrics compile time पर ही decide हो जाते है। जैसे - कितनी memory मिलेगी कब मिलेगी, lifetime क्या होगा, कहा-2 से आप access कर पाएंगे	DMA वाले variable को बनाते वक्त हम ये नही बताते है कि variable किस type का बनाना है। हम सिर्फ size बताते हैं। कितने bytes
During execution of the program इसमे कोई बदलाव नहीं हो सकते time से पहले इसकी memory	DMA वाले variables की characteristics compile time पर decide नही होती।

SMA वाला variable बनना train के लिए पहले से reservation कराने जैसा है। reservation के time सारी choices भी बताई जाती हैं, जैसे characteristics पहले decide हो जाती है	क्योंकि compiler सिर्फ declaration statements को समझ सकता है, action statements को नहीं, इसलिए malloc, calloc क्या कर रहे हैं, ये compiler को पता नहीं चलता। सब कुछ run time पर decide होगा - • malloc चलेगा नी चलेगा • कितनी मैमोरी चाहिए • lifetime, scope
DMA वाला variable बनाना बिना reservation के train में बैठना है, जब ट्रेन आएगा तो खाली सीट से एक book कर लेंगे	
जब पहले से पता है कितनी memory की जरूरत है, कब जरूरत है, तो SMA वाले variable बनाएंगे।	जब हमें memory में flexibility चाहिए तब हम DMA variable का use करेंगे।

Use of malloc → इसे call करते वक्त इसके अंदर एक integer value pass करते हैं, जो कि है size of variable in bytes.

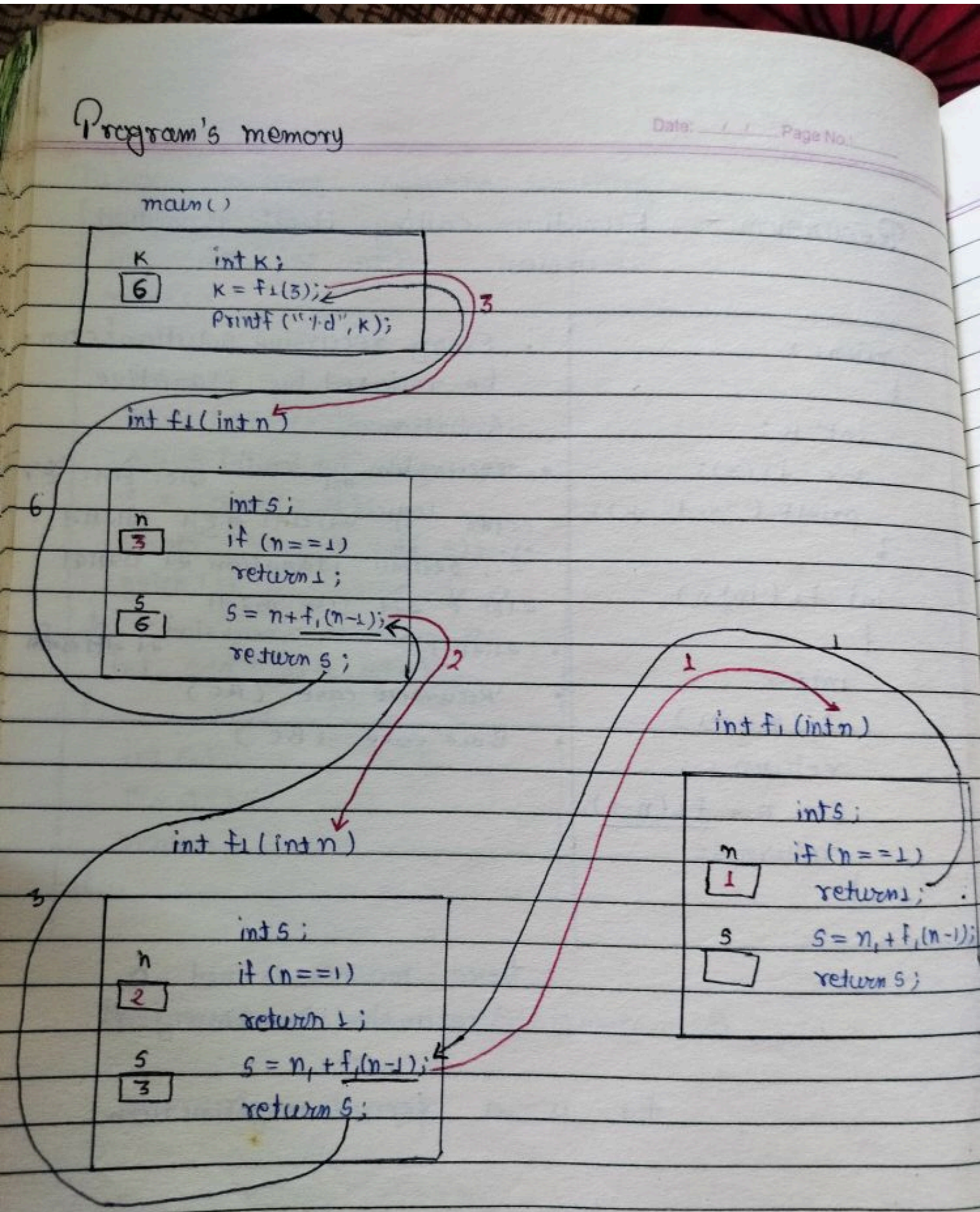
```
malloc(4);
```

malloc को नहीं पता की किस type का variable है, malloc को जानना भी नहीं है malloc तो सिर्फ उसकी पहली byte का address return करता है।

size से type का पता नहीं चलता है। यहाँ 4 लिखा है तो ये int, float, char, struct... या कुछ भी हो सकता है।

Program's memory

Date: / / Page No.:



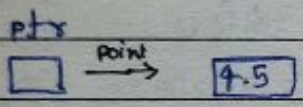
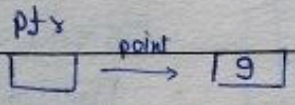
Date: / / Page No:

अब हमे तो पता है हमे किस type का variable चाहिए।

malloc जो first byte का adress return करता है, उसे हम assign करवा सकते हैं। कहाँ - हम जिस pointer variable में चाहेंगे।

malloc function की मदद से बनने वाले variable मे bydefault garbage होता है।

calloc —||— bydefault 0 होता है

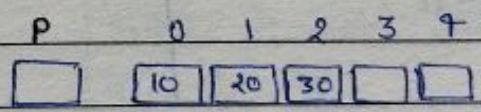
float *ptr	int * ptr
ptr = malloc (4)	ptr = malloc(4)
	
*ptr = 4.5;	*ptr = 9

calloc (number of variable, size of each variable);

calloc (5, 4);

calloc (a, b); // a, b की value हम user से ले सकते हैं।

```
int *p;
p = calloc(5, 4);
```




```
*(p+0) = 10;
*(p+1) = 20;
*(p+2) = 30;
```

```
main()
{ int k;
  k = f1(3);
  printf("%d", k);
}
int f1(int n)
{ int s;

  if (n==1)
  return 1;          } BC
  s = n + f1(n-1);
  return s;          } RC
}
```

f1(3)
3 + f1(2)
2 + f1(1)
1

f1(10)
10 + f1(9)
9 + f1(8)

Sum(n)
n + sum(n-1)

Base case - वो situation जब function खुद को call करना बंद कर देता।

in a recursive function, make sure की वो कहीं पर खत्म हो रहा है, और जहाँ खत्म हो रहा है वही Base case है, you have to always think about this base case

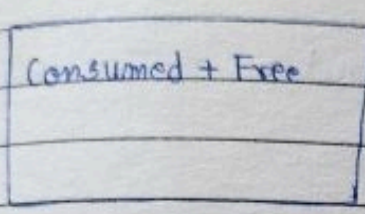
Recursive case - this is our main formula to solve the problem.

recursion से किसी problem को solve करना है तो उसमे एक ऐसी लाइन लिखनी पड़ेगी जिसमे तुम उसी function को दोबारा call कर रहे होंगे, लेकिन अब जब call कर रहे हो तो original problem से थोड़ा simpler version के लिए call करो। जैसे 10 number का निकालने से तो simple है 9 number का sum निकालना, और 9 का कैसे निकालना है वो recursion पर छोड़ दो।

most important interview

memory leak concept

Program's memory — The ratio of consumed and free memory




```
F1 ()
{
  int * p;
  p = malloc (4);
  ....
}
```

when F1 function will end

- p will destroy // it's a local variable and SMA का बना variable
- malloc से बना variable
  ये destroy नहीं होगा • क्योंकि it is not considered as local variable.
  क्योंकि DMA variables का life time decide हुआ है नहीं compile time पर
  जब तक program end नहीं हो रहा तब तक ये variable memory में रहेगा।

After ending of the function p destroy हो चुका है; पर malloc से बना variable तो है।

हमे इस variable का नाम नहीं पता पर memory block पता है पर p destroy हो जाएगा तो हम इस memory block को access नहीं कर पाएंगे।
उसपर दोबारा malloc कॉल करेंगे तो नया memory block बन जाएगा। उस पहले memory block को अब हम access नहीं कर सकते।

malloc की मदद से बना ये variable consumed में ही गिना जाएगा पर हम इसे use नहीं कर सकते, this concept is called memory leak.

This memory is consumed but we cannot use it

Date: / / Page No.:

So the

Arrays

- Array is a linear collection of similar elements

If we want to write a program to calculate average of 100 numbers. then we have to declare 100 variables

At the time of creating this number is not an index but it is total number of variables in array

```
int a[100];
```

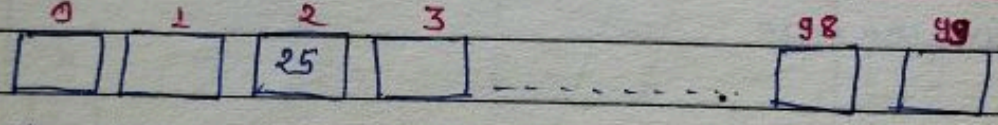
- Array is a group of variables.
- Array is also known as subscript variable

The value writen in squar bracket is subscript [100]

Subscript a_1 a_2 a_3 a_4

Array's name → a

All variables - individualy have no name - but have possition number



also called index or subscript value

this thing is important when you want to access any variable of this group.

`a[2] = 25;` it is not the name of that variable but it is a way of representing that variable. here

[] subscript operator

a and 2 are operands

Prevention from memory leak.

```
f(1)
{
int * p;
p = malloc (4);
----
free (p);
}
```

⊙ if we have no need of malloc's variable after ending of f1 function. then you can releas it's memory before ending of f1 function. for this there is a predefined function free (adress)

with the help of free function we can only releise the memory of DMA's variables.

⊙ if we want to use malloc's variable after ending of F1 function

```
int * ptr ;
ptr = f1();
Int* f1()
  {
    int * p ;
    p = malloc (4);
    ----
    return p ;
  }
```

return the value of p. after ending of f1 function p will destroy but the adress that was stored in p will be copied in ptr . and now pointer ptr will point that malloc's variable.

Same mathod is used in calloc function.

```
main()
{
  int i, j ;

 for ( i=1 ; i<=4 ; i++)
{
  for ( j=1 ; j<=5 , j++ )
  {
    if()
     printf ("*");
    else
      printf ("$");
  }
    printf ("\n");
}
   getch();
}
```

i | j | i=2 | i=3

1<=4 2<=4 3<=4

1<=5.... 1<=4.... 1<=4....

if (j<=1) if(1<=2) if(1<=3)

```
*
* *
* * *
* * * *
```

j | i<=4 | 1<=4 | if(1<=4)

Star pattern

j colomn

i rows	1	2	3	4
1	*			
2	*	*		
3	*	*	*	
4	*	*	*	*

i	j
1	1
2	1 2
3	1 2 3
4	1 2 3 4

i	j
1	j<=1
2	j<=2
3	j<=3
4	j<=4

j <= i

So the program

```
main()
{
int i, a[100], s=0;
float avg;
printf("Enter 100 numbers");

for (i=0; i<=99; i++)
 scanf("%d", &a[i]);

for (i=0; i<=99; i++)
  s = s + a[i];

avg = s/100.0;
printf("Average is %f", avg);

}
```

Assignment -

in Ass copy

Date: / / Page No.:

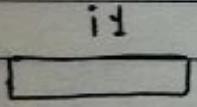
Union

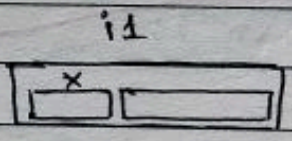
it is a user defined data type. It is same as structure but memory allocation is different.

memory = data type of highest size (in bytes)

```
Union Item
{
  int x;
  double y;
};
main()
{
  Union Item i1;
  i1.x = 10;
  i1.y = 3.5;
}
```

```
struct Item
{
  int x;
  double y;
};
main()
{
  struct Item i1;
  i1.x = 10;
  i1.y = 3.5;
}
```





in union at one time we can use only one variable.

for eg. it is of 8 byte

its 4 byte will be treated as x

and whole 8 byte will be treated as y.

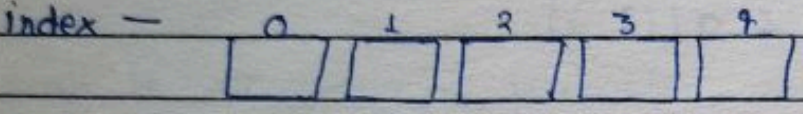
Both values can not exist at same time. we can use only one member variable in one time. x को use कर रहे हो तब तक y को use मत करो और जब y को use करो तो x को मत करो।

दोनो member को at a time जरूरत पड़ेगी तो हम structure use करेंगे

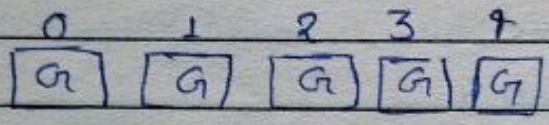
Array

1. `int a[];` // Error

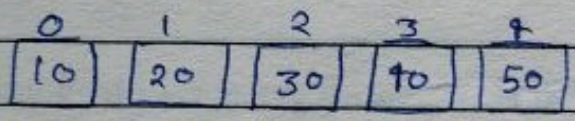
2. `int a[5];` - not an index
   - total number of variables
   - natural number



3. `int a[5];`

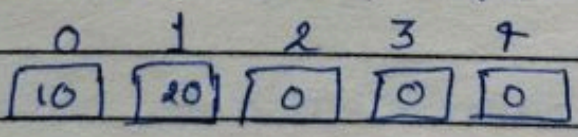


4. `int a[5] = { 10, 20, 30, 40, 50 };`

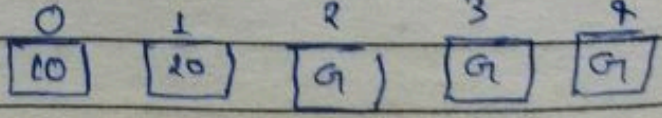


5. `int a[5] = { 10, 20, 30, 40, 50, 60, 70, 80 };` // Error

6. `int a[5] = { 10, 20 };`



7. `int a[5]`
   `a[0] = 10;`
   `a[1] = 20;`



Date: / / Page No.:

1. Function returning structure

We want to make a function which will return a structure variable.

```
struct Book
{ int bookid ;
  char title [20] ;
  float price ;
};
struct Book input ();
main()

{ struct Book b1 = { 1 , "Drilling C" , 450.0 } ;
  struct Book b2 , b3 ;
  b2.bookid = 2;
  strcpy ( b2. title , "Java") ;
  b2. price = 300.0 ;
b3 = input () ;
printf ("\n %d %s %f " , b3. bookid , b3. title , b3. price ) ;
getch()
}
  Struct Book input ()
{ struct Book b ;
printf ( "Enter bookid , title and price" ) ;
scanf ( " %d " , & b.bookid ) ;
fflush (stdin) ;
gets ( b3. title ) ;
Scanf ( " %f " , & b3. price ) ;
return b ;
}
```

2. Function call by passing structure

We want to display / print by calling a function display.

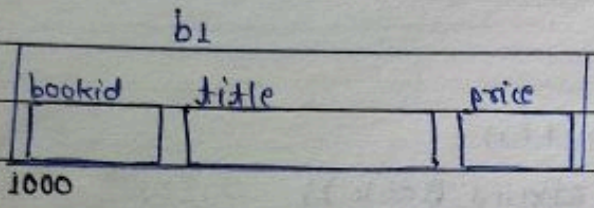
```
    —11—
{   —11—
};
    Struct Book input();
    void display ( struct Book );
main()
{   —11—
    —11—
    b3 = input();
    display(b3);   // we can call display again
    display ( b2);
    display ( b1);
    getch();
}
Struct Book input()
{   —11—
    —11—
}
void display ( struct Book b )
{
printf ("\n %d %s %f", b.bookid , b.title, b.price );
}
```

We can make structure array → struct Book b[5];
in this structure variable b[0], b[1], b[2], b[3], b[4]
each variable is of 28 bytes.

Date: / / Page No.:

Structure Pointer

structure variable → struct Book b1 ;



pointer → struct Book *p ;
p = &b1 ;

p
1000

// p point कर रहा है b1 को

Now if we want to access the member variable. we cannot directly access it (bookid = 10) it is not posible.
There are two ways for accessing member variable.

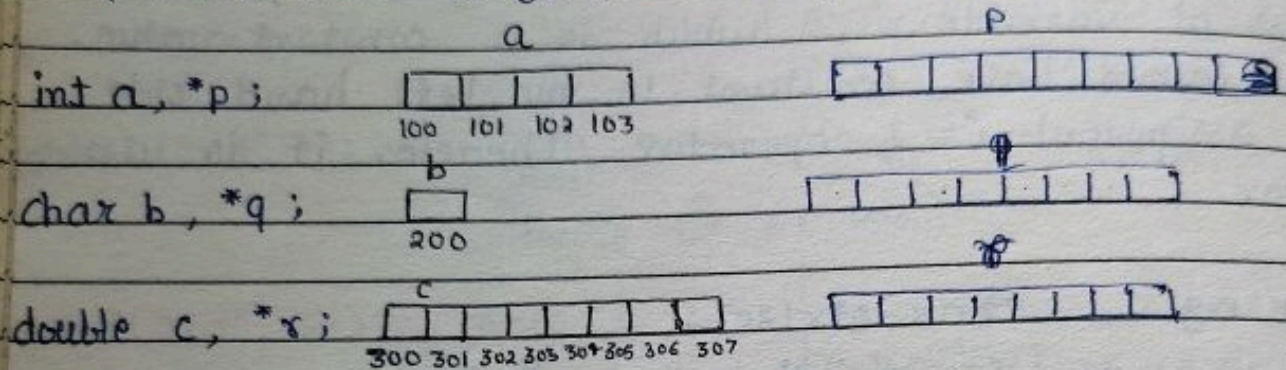
1. b1.bookid = 10 ; member variable जिस structure variable के अंदर है, वही structure variable dot लगाकर member variable लिखेंगे।

2. through pointer - वो pointer जो इस structure variable को point कर रहा है, जिसमे ये member variable है।

p -> bookid = 10 ; जो b1 को point कर रहा है उस pointer की मदद से आप b1 के member को access कर सकते हो।

So we can write (*p).bookid = 10
but we do not prefere it

pointer variable have to store adress.
the size of pointer variable does not depend on its type. it is always of fix size that is 8 bytes (64 bit)
(64 bit software - 8 byte, 32 - 4, 16 - 2)



pointer is always of same bytes (8). so what is the meaning of data type.

p always stores Base adress (Adress of the first byte)
e.g. 100 in p

*p लिखा होने पर, derifrencing operator को ये कैसे समझ मे आएगा की 100 के बाद 101, 102, 103 को भी consider करना है तब जाकर तो पुरा a variable represent होगा।
*p = a

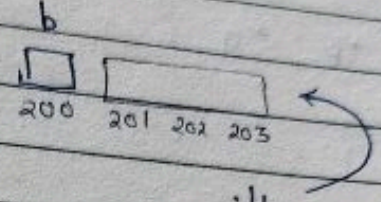
same for *q, *r that is why data type is meaningful.

p का type int है इसका मतलब ये नहीं है कि p में ordinary integer रखा जाएगा, बल्कि it means p में int variable का adress रखा जाएगा।

Pointer का जो type होता है, उसी type के variable का वो adress contain करेगा है।

if by mistak you have writen

```
p = &b;
*p
```



illigal memory excess
can caus of crass of program.

Extended Concept of pointers

```
int x=6, *p, **q
p = &x;
// pointe variable का adress रखे आगर q में
q = &p;
```

So declaration

x	p	q
5	1000	2000
1000	2000	3000

// अगर q का adress रखना है तो
dedaration में `***r;`

```
r = &q;
```

x	p	q	r
5	1000	2000	3000
1000	2000	3000	4000

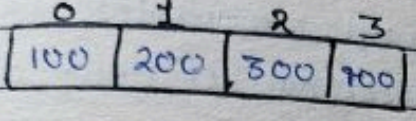
here x is 0 level
p is 1 level pointer
q is 2 level pointer
r is 3 level pointer

```
printf("%d %d %d %d", x, *p, &q, x);
// output 3000, 5, 3000, 5
printf("%d %d %d %d", **r, &p, *q, &x);
// output 1000, 2000, 1000, 4000
printf("%d %d %d %d", ***r, **q, p, *r);
// output 5, 5, 1000, 2000
```

Array of pointers

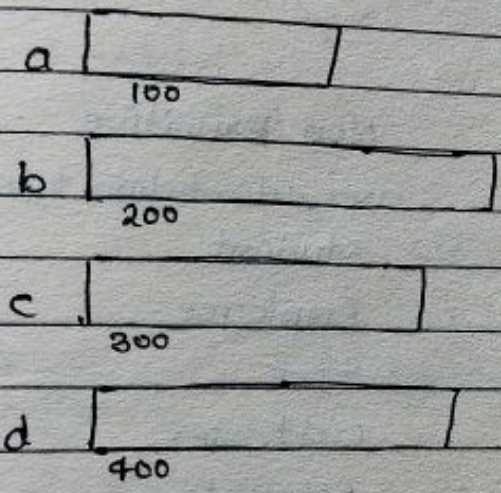
in this we make an array in which all variables are pointers

```
int *p[4];
```




```
int a[5], b[8], c[6], d[7];
```

```
p[0] = a;
p[1] = b;
p[2] = c;
p[3] = d;
```



p[0][j]
p[1][j]
p[2][j]
p[3][j]

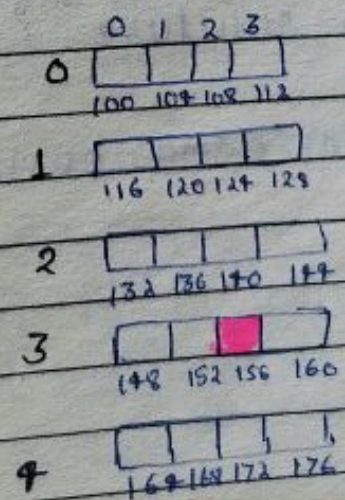
p[i][j]
*(p[i]+j)
((p+i)+j)

Pointer to Array

a single pointer which we made for pointing an array

```
int (*p)[4];
```

```
int a[5][4];
```




```
p = &a[0][0];
```

p → 100

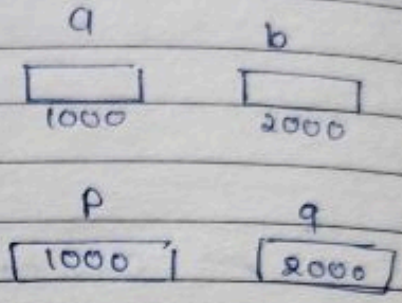
p+1 = 116
p+2 = 132

((p+3)+2)

p[3][2]

Pointer's Airthmetic

```
int a, b, *p, *q;
p = &a;
q = &b;
```



invailid operations

- p+q
- p*q
- p/q
- p*5
- p/4

valid operations

- p+2
- q-5
- q-p

p+1 it depends on type of p. p is a int type variable. one variable of int type is of 4 byte if p = 1000

So

p+1	1004
p+2	1008
p+3	1012
p+5	1020
q-1	1996
p-2	992

two adress can be substract, provided both are of same type

q-p	250
p-q	-250

show direction only

Date: / / Page No.

Structure

1. Structure can be a collection of dissimilar elements
2. structure is a way to group variables
3. defining structure means creating a new custom data type

Data type

Primitive	Non Primitive
pre defined data type	user defined data type
int	Student
char	Employee
float	Book
double	customer
	Account

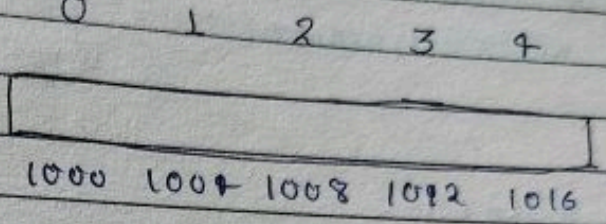
Non primitive → custum data type → User define data type → Secondary data type

How to define a structure ?

Structure define करना मतलब custom data type बनाना।

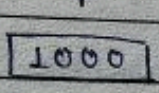
this concept will be helpful when we handle array

int a[5];



int *p

p = &a[0]



p+0	1000
p+1	1004
p+2	1008
p+3	1012
p+4	1016

*(p+0)	a[0]
*(p+1)	a[1]
*(p+2)	a[2]
*(p+i)	a[i]
p+i	&a[i]

Struct - it is a keyword
Book - user defined data type

between curly bracket we have to declare variable which are creating a data type by grouping

```
struct Book
{
    int bookid;
    char title[20];
    float price;
};
```

here we define a data type book.

it is just a part of structure defination

We have define this data type we have not use it yet.
So the cunsumed memory is 0 byte.

memory data type को नही मिलती variable को मिलती है।

जैसे int premitive data type है mean it is a predefined data type पर इसे मैमोरी तब तक नहीं मिलती जब तक हम variable declare ना कर दें, मतलब जब तक int x; ना लिखे।

उसी तरह से book भी define data type (user define) है पर उसे मेमोरी तब तक नहीं मिलेगी जब तक इसका use नहीं होगा। book को memory नहीं मिलेगी क्योंकि book तो data type है। क्योंकि अभी data type को use ही नहीं किया है तो consumed memory 0 byte है।

if we want to swap two variables by calling a function in main function

```
main()
{
    int a, b;
    printf("Enter two numbers");
    scanf("%d%d", &a, &b);
    printf("a = %d  b = %d", a, b);
    swap(a, b);
    printf("\n a = %d  b = %d", a, b);
    getch()
}

    void swap(int x, int y)
{
    int t;
    t = x;
    x = y;
    y = t;
}
```

if we have given the values of variables of main function, then this function can use the values but can not make any change in main function variable. variable में changes करने के लिए जरूरी है variable को Axis कर लें।

but it will not do the work that we want. because this is a wrong way the data of a and b will be copied in x and y, then swaping in x and y, but there is no change in a and b. changes in variable x and y does not make any difference in a and b.

value तो return करा नी सकते।

We can define data type globally as well as locally

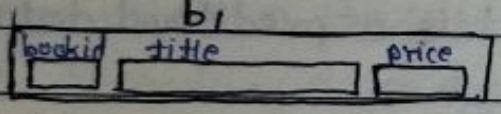
How to create variables of structure ?

```
struct Book
{
   int bookid;
   char title[20];
   float price;
} b1 , b2 ;
```

// globlly created variables

// locally created variables

now variable b_1 will consume 28 byte memory.



b_1 → structure variable

bookid
title
price
} member variables

```
Struct Book
{
   int bookid;
   char title[20];
   float price;
};

main()
{
   struct Book b1;
}
```

Pronounciation: * a = astric a
dereferencing operator

The right way is to pass the adress of a and b.
for reciving the adress we need pointer variables

```
void swap ( int * , int * );
main ()
{  int a, b ;
   printf ("Enter two numbers");
   scanf ("%d %d", &a, &b);
   printf ("a = %d  b = %d", a, b);
   swap (&a, &b);   // call by reference or by adress
   printf ("\n a = %d  b = %d", a, b);
   getch();
}
   void swap (int *x, int *y)
{   int t;
    t = *x;
    *x = *y;
    *y = t;
}
```

if we want from any called function to make changes in our variable, then we have to pass the adress of our variable.

in scanf we use & because ↑ we want to make chang in variable

How to initialize structure variable during declaration?

```
struct Book
{
 int bookid;
 char title[20];
 float price;
};

main()
{
 struct Book b1 = {1, "Drilling C", 450.0};
}
```

How to initialize structure member variables after declaration of structure variable ?

StructreVariable . member variable

→ dot (.) is a variable to access member variable

```
-11-
main()
{
 Struct Book b1 = {1, "DrillingC", 450.0};
 Struct Book b2;
 b2.bookid = 2;
 strcpy(b2.title, "Java");
 b2.price = 300.0;
}
```

str = Bhopal // Error

title = Java // Error

strcpy (b2.title, "Java");

Pointers and Arrays

```
main()
{
int a[10];
input (a, 10);
}
void input (int b[], int size)
{
int i ;
printf ("Enter %d numbers", size);

for (i=0; i<size; i++)
scanf ("%d", &b[i]);
```

```
void input (int *, int);
void display (int *, int);
main ()
{
 int a[10];
 input (a, 10);
}display (a, 10);
 void input (int* P, int size)
 {
  int i ;
  printf (" Enter %d numbers", size);
   for(i=0; i<size; i++)
  scanf ("%d", P+i);
 }
```

```
void display (int *P, int size)
{ int i ;
for (i=0; i<size ; i++)
printf ("%d", *(P+i));
}
```

a 1000 / a is a constant a++ ; error

P 1000 / P is a variable P++ ; Correct

a[3] internaly → *(a+3) can write → *(3+a) → 3[a]

*(P+3) → P[3] → 3[P] → *(3+P)

How to take input from user ?

—II—

```
main()
{
struct Book b1 = { 1, "Drilling C", 450.0 };
struct Book b2, b3;
b2.bookid = 2;
strcpy(b2.title, "Java");
b2.price = 300.0;

printf("Enter bookid, title and price");
scanf("%d", &b3.bookid);
gets(b2.title);
scanf("%f", &b3.price);

printf("\n%d %s %f", b3.bookid, b3.title, b3.price);
getch();
```

it's output will be wrong because ⇒ scanf and gets function directly data keybord से नहीं लेते हैं, बल्कि एक input buffer से लेते हैं (assume it a midater (a huge char array)). keybord से data पहले buffer में जाता है buffer से scanf या gets data उठाते हैं और हमारे variable में रखते हैं।

जब scanf चला उस समय buffer खाली था तो scanf buffer में से डाटा नहीं निकाल पाया इसलिए fresh input होता है, कि buffer खाली है।

जब आप keybord से कुछ enter करते हो तो वो data buffer में जाता है। scanf उस data को उठाता है और bookid वेरिएबल में रख देता है लेकिन उ क (bookid) enter करने के बाद आपने जो enter key press की थी scanf ने उ तो उठा लिया था पर enter को उसी में रहने दिया था, इस वजह से buffer अभी खाली नहीं है और अब जब गेट्स function चलता है तो वो ये देखता है कि buffer तो अभी खाली नहीं है तो वो fresh input को allow नहीं करता। अब इसके बाद जब scanf चला तो buffer खाली हो चुका है क्योंकि gets ने खाली कर दिया था अब ये scanf के कारण screen रुकी हुई है, तो ये actual में price के लिए चल रहा है। पर title के लिए तो रुका ही नहीं।

for removing this problem → fflush(stdin);

और इसे लिखने के लिए #include <stdio.h>

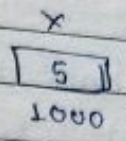
stdin is input buffer

```
#include <stdio.h>
struct Book
{ —||— 
};
main()
{
 —||—
 —||—
printf("Enter bookid, title and price");
scanf("%d", &b3.bookid);
fflush(stdin);
gets(b3.title);
scanf("%f", &b3.price);
 —||—
}
```

fflush is a predefined function जो buffer को खाली करता है।

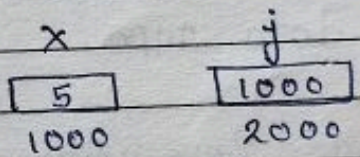
ये problem ज्यादातर तब आएगी जब character या string input करवा रहे है और इसके पहले input operation perform हो चुका है।

```
int x = 5 ;
&x = 7 ;  //Error
```



&x is not a variable but it is a way to represent adress of variable x, which is a constant value. Ie cannot have constant in the left hand side of assignment (=) operator. Therefor it is an error.

other e.g.
```
char str[20];
str = "Bhopal";  // Error
```




```
int x = 5 ;
int *j ;
 j = &x ;
```

we have declared j variable as *j so that compiler can understand that is not a normal variable like x but it is a special variable, that stores only value that is adress.

```
printf ("%d %d %d", &x, j, *&x);
//output- 1000 1000 5

printf ("%d %d %d", *j, x, &j);
//output- 5 5 2000
```

j is a pointer variable which contains adress of variable x.

pointers are variables which can store adress of another variable.

*j == x

for printing three strings

```
int i ;
for ( i=0 ; i<=2 ; i++ )
printf ( "%s\n" , str[i] );
```

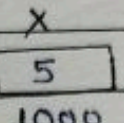
for user input

```
for ( i=0 ; i<=2 ; i++ )
  gets ( str[i] );
```

Pointer.

What is pointer ?

```
int x = 5;
```



adress number

```
printf("%d", x);  5
```

किसी भी variable का adress क्या हो ये हम decide नहीं कर सकते ।

```
printf("%d", &x);  1000
```

- adress number
- also called reference number
- always whole number

```
printf("%d", *&x);  5
```

```
*&x == x
```

&

- Address of operator
- Referencing Operator
- Unary Operator
- `&__` ← operand → always variable
- result of this operator is adress number.

*

- indirection operator
- dereferencing operator
- unary operator
- `*__` ← operand → always adress number
- result of this operator is variable.

Date: / / Page No:

without strlen function

```
main()
{
   char str[20];
   int i;
   printf("Enter the string");
   gets(str);
  for(i=0; str[i]; i++);
   printf("Length is %d", i);
   getch();
}
```

There is no statement in for loop we can also write it as
for(i=0; str[i]; i++)
;

char* strupr(char*) it is a predefined function it changes the string into upper case

at the time of calling in arguments we will pass the name of char array or string constant.
~~but never ever char value~~ ⊗ never

char* लिखने का मतलब है कि ये pointer variable है।

return type → string का adress return होगा। receive किसी pointer में करेंगे।

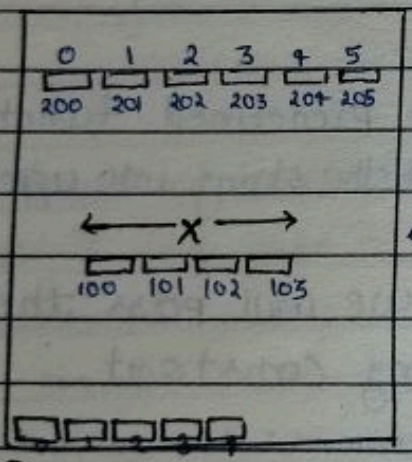
char* strlwr (char*);

in this, actual argument is name of the array or string constant.

it will return adress

for runing a program - allocate memory

- allocate memory for runing a program does not depend on the size of program.
- it may varies on different operating system
- this memory is lackes of bites. (1 bite = 8 bites)



it we have use int variable in our program so it is of 4 bytes

it is of 4 bytes (variable x)

it may be in anywhere in this allocate memory.

Program's memory in RAM

the position of first byte of x variable can be any position lets assume it is 100.

position number of first byte is known as adress number. // reasone in pointers chapter

if we have use char array.
one block of array is of one byte

if the size of array is 20, so there will be 20 block in memory.

1 char variable = 1 byte

if we write str it means the adress of its first block

str == &str[0] == 200 (in eg.)

```
int i ;
for (i=0 ; str[i] ; i++)
printf ("%c", str[i]);
```

or

```
printf ("%s", str);
```

1 st situation →
(when main function is calling the function printf, printf को str[i] की value दी जाती है।
means firstly the value of str[0] will be given to printf and printf will print that character.

#So printf recive the charactor and print it for printing second charactor we need to call printf again

means str[0] की value देने से सिर्फ एक character print हो पाएगा। इससे पुरी string input हो पाना impossible है। इसके लिए printf को उतनी बार call करना पड़ेगा जितनी बार character

2nd situation

```
printf ("%s", str);
```

यहाँ जब function को हमने call किया है तब printf को str दिया है। लिखा है। it means हमने (&str[0]) str[0] का adress दिया है।

according to eg. हमने 200 दिया है

* pointer की मदद से हम उस memory location को access कर सकते है जिसका adress हमारे पास है।

eg. और printf 200 को recive कर रहा है तो वो 200 adress पर रखी value को access कर पाएगा, तो पहला character print हो जाएगा।

और simple mathematical calculation करके next bite का adress भी creat कर लेगा। means 201 अब इस adress पर रखी value को भी access कर पाएगा और दुसरा character भी print हो जाएगा।

ऐसी ही सारे characters को access कर पाएंगे जब तक null character ना आ जाए। इस वजह से string properly print हो रही है।

the similar process happenes in gets function too.

gets (str); यहाँ हम array का नाम लिख रहे है मतलब adress दे रहे है।

और इसी वजह से हमारे पुरे array को वो access कर पाता है। और जो keybord से character आ रहे है वो हमारे array के अंदर एक-एक जगह पर store करवा पाएगा क्योंकि वो पुरे array को access कर सकता है

int strlen (char*)
int strlen (str)

we have pass adress in strlen function. it means strlen which is a predefined function have contain pointer.

and with the help of pointer we will be able to access the complete (whole) array.

same happenes in functions char* strupr (char*)
char* strlwr (char*)

the return type of this both function is char* it means it will be returning adress.

we can use it in program as - convert the string into upper case and then calculate the length of the string.

```
main()
{
char str[20];
int l;
printf("Enter your name");
gets(str);
l = strlen(strupr(str));
printf("Length is %d", l);
getch();
}
```

```
main()
{
 char str[20];
 int l;
printf("Enter your name");
gets(str);
printf("String in upper is %s", strupr(str));
getch();
```

char* strrev(char*); // it reverse the string

char* strcpy(char*, char*); // it copies the string

Why two arguments ? →

in first argument → name of the array in which we will store and copy.
in second argument → source string's name

char* strcat(char*, char*);
// it is used for concatination / joint / append

int strcmp(char*, char*); // it compares two strings
// it compares two strings. it's result → -1 dictionary order
0 same
1 opposite of dictionary order

```
main()
{
  char str1[20] = "ABC", str2[20] = "ADF";
  int l;
  l = strcmp(str1, str2);
  printf("%d", l);
  getch();
}
```

Output ⇒ -1 dictionary order

```
main()
{
  char str1[20] = "BBC", str2[20] = "ADF";
  int l;
  l = strcmp(str1, str2);
  printf("%d", l);
  getch();
}
```

Output ⇒ 1 opposite of dictionary order

```
main()
{
  char str1[20] = "BBC", str2[20] = "BBC";
  int l;
  l = strcmp(str1, str2);
  printf("%d", l);
  getch();
}
```

Output ⇒ 0 same

Date: / / Page No.:

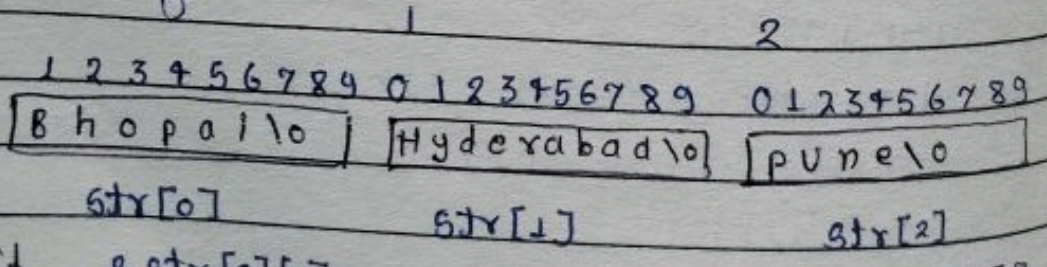
String Functions

```
int strlen (char* );
char* strupr (char* );
char* strlwr (char*);
char* strrev (char*);
char* strcpy (char*, char* );
char* strcat (char* , char* );
char* strcmp (char* , char*);
```

Handling multiple string

Two dimentional char array

```
char str[3][10] = { "Bhopal", "Hydrabad", "Pune" };
```



So we will use the function gets.
gets function is created only for inputing string that's why formate specifier is not used here.

```
main()
{
  char str[50];
  int i;
  printf("Enter your name");
  gets(str);
  printf("Hi, %s", str);
  getch();
}
```

String functions

String.h ⇒ it is a header file in C language.
it contains some declared function.

// predefine functions are defined in library file.
in header file there is only declaration.

? if you know the prototype of a function or you know declaration then can you assume what is the work of this function.

for eg. int strlen (char*);

its return type is int so it simply means it calculates the lengh of the string and show.

it is take something return something so what does it take ⇒ string

int strlen(char*); it is a predefine function so we will call it whenever we need.

at the time of calling - what will write in arguments

int strlen (char); what does it mean ?

Answer ⇒ it means at the time of calling we would be passing one character (Actual argument)

char variable (formal argument)

// formal argument ⇒ function define करते वक्त paranthesis में क्या बना होगा।

formal argument should be ⇒ char type variable

char type का variable एक character constant store कर सकता है। तो फिर call करते वक्त pass किया जाने वाला एक character.

if int strlen (char*);

it means formal argument में बनेगा pointer variable

at the time of calling the function, we will not enter a charater instead of it we will enter the name of char array or we will enter string

e.g.
```
main()
{
char str[20];
printf ("Enter your name");
gets (str);
strlen (str);   // or   strlen ("BHOPAL");
}
```

char लिखने का मतलब ये कि ये जो str है ये char type के array का नाम है int type के array का नही।

```
main()
{ char str[20];
int l;
printf ("Enter your name");
gets (str);
l = strlen (str);
printf ("Length is %d", l);
getch();
}
```

```
void f1()
{ register int x=5;
  x++;
  y=x+4;
}
main()
{ f1();
  getch();
}
```

जरूरी नही है register मे memory मिल ही जाएगी। हमने request की है मिल गई तो ठीक वरना वैसे ही चलेगा जैसे पहले था। RAM मे ही रहेगी मिलि हुई memory। ये बहुत पुराना concept है; पहले speed की ~~problems~~ ज्यादा होती थी।

Static → Static variable's life will end after ending of program.

Scope → block, जिस block में बने है उसी में access होगा

```
void f1()
{
  int x;
  static int y;
  x++;
  y++;
  printf("\n x=%d y=%d", x, y);
}
main()
{ f1();
  f1();
  getch();
}
```

in y by default value 0

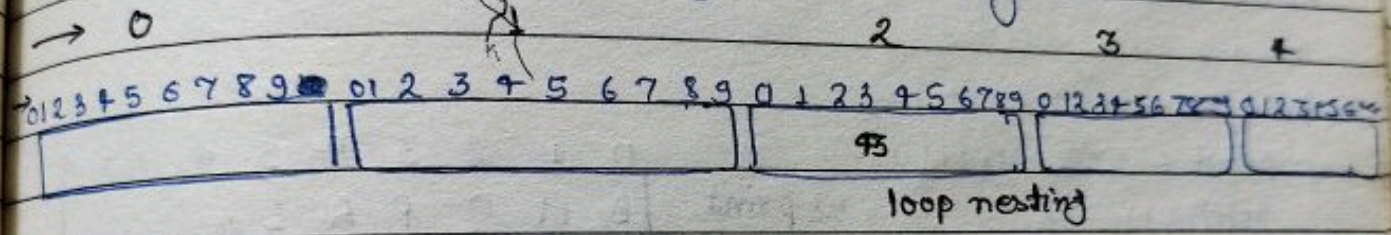
Static variable को program के शुरुवा होते ही memory मिल जाएगी

output - x=1 y=1
x=1 y=2

2 way of declaring 2 d array int a[5][10];

Actual view - memory alocation

memory consumption is same as 1 d array.
the difference is only of indexing



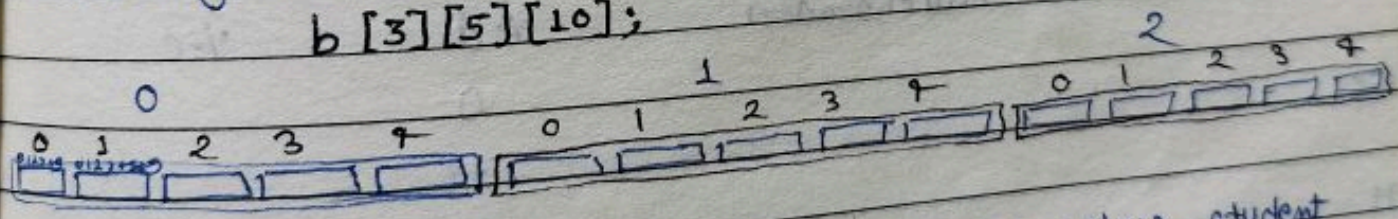
a[2][3] = 93

```
i = 0 to 4
j = 0 to 9
a[i][j]
```

```
for (i=0; i<=4; i++)
for(j=0; j<=9; j++)
printf("%d", a[i][j];
```

3d array ⇒ dimentions - institude, class, student

b[3][5][10];



4 d array ⇒ dimentions - zone, institute, class, student

C[4][3][5][10];

5 d array ⇒ region, zone, institute, class, student

Date: / / Page No.:

External storage class → global variable
by default global variables value is 0.

accessable in both ~~sti~~ blocks.

```
int x;
void f1()
{
  int x=0;
printf ("\n f1x = %d", x);
}
main()
{
printf ("main : x = %d", x);
f1();
getch();
}
```

output f1 x = 0 main : x = 0

if you have declared x variable globaly but somwhere in different position as in e.g. then **extern** keyword is neccessary

if you declare any local variable in ~~x~~ f1 function, so print local वाला होगा, local को ज्यादा preference दी जाती है।

```
Void f1()
{
  extern int x;
  printf ("\n f1 : x = %d", x);
}
  int x;
  main()
{ printf ("\n main : x = %d", x);
  f1();
  getch();
}
```

String

Sequence of characters terminated at null character is a known as a string

```
main()
{
  char str[10] = {'B','H','O','P','A','L'};
  int i;
  for(i=0; i<=9; i++)
    printf("%c", str[i]);
  getch();
}
```

	0	1	2	3	4	5	6	7	8	9
%c print	B	H	O	P	A	L				
store	66	72	79	80	65	76	0	0	0	0
%d print	66	72	79	80	65	76	0	0	0	0
in program	'B'	'H'	'O'	'P'	'A'	'L'	'\0'	'\0'	'\0'	'\0'

ASCII

codes	characters
0	'\0' (null character)
32	' '
48	'0'
65	'A'
97	'a'
255	

%d	%c
0	
32	
48	0
65	A
97	a

A programer can cheak wether a perticular variable contains space or null charact

space and null character are not same although at ⅈC the output of both is same.

String is a sequence of characters terminated at null character.

- space can be part of a string. space-character
- null charactor cannot be a part of string. null has a special meaning in strings.

So the final output of our program ⇒ BHOPAL followed by four spaces.

B H O P A L _ _ _ _

or in real BHOPAL ↖ Curser

here we are trying to print null charactor which is useless their is no point to print null charactor

here the size of array is 10 but string do not have 10 charact and we want to print string.

So for(i=0; i<=9; i++) is a blunder

✓ for(i=0; i<=5; i++)

```
main()
{
char str[10] = {'B', 'H', 'O', 'P', 'A', 'L'};
int i;

for (i=0; i<=5; i++)
printf ("%c", str[i]);
getch();
}
```

But if string is not declared at the time of initialization may be we are scaning it from user or it is coming from somewhere else.

then how will we apply the condition. ~~str[i] != '\0'~~

may be it is not declared

```
main()
{
char str[10] = {'B', 'H', 'O', 'P', 'A', 'L'};
int i;
for (i=0; ~~str[i]~~ str[i]!='\0'; i++)
printf ("%c", str[i]);
getch();
}
```

str[i] = '\0' = 0 ⇒ false

So we can write it as

```
main()
{
char str[10] = {'B','H','O','P','A','L'};
int i;
for (i=0; str[i]; i++)
printf ("%c", str[i]);
getch();
}
```

We will use this loop when we want reapeat any operation on each character of string. or we want to examine each character

for printing a string the easiest way

```
main()
{
char str[10] = {'B','H','O','P','A','L'};
printf ("%s", str);
getch();
}
```

formate spacifier %s

%c is used when you want to print only one character and %s -11- print whole string. with %c we write str[i]
with %s → str (this is the name of the array)

we can write

```
main()
{
char str[10] = {'B', 'H', 'O', 'P', 'A', 'L'};
printf ("%s", str);
getch
```

```
main()
{
char str[10] = "BHOPAL"
printf ("%s", str);
getch();
```

How to take input from user

```
main()
{
  char str[50];
  int i;
  printf ("Enter your name");
  for (i=0; i<=49; i++)
    scanf ("%c", &str[i]);
    getch();
}
```

This is a wrong way because user have to enter 50 characters although user's name may have less then 50 charac

This is also a wrong way because the value of str[i] == 0 when character is '\0' null character there value will not become 0 because here string is not initalized at declaration and user is not going to use enter character '\0' null character

```
main()
{
  char str[50];
  int i;
  printf ("Enter your name");
  for (i=0; str[i]; i++)
    scanf ("%c", &str[i]);
    getch();
}
```

Now

Now

```
main()
{
char str[50];
int i;
printf("Enter your name");
scanf("%s", str);
printf("Hi, %s", str);
getch();
}
```

after running this program if we enter Nidhi yadav then the output will be Hi, Nidhi

It happens becaus of scanf → scanf have three delimiters

- space
- tabe
- newline cha. / enter

if the data entered by the user includes any delimiter than it will be considered that data have end there.

and that's why the space after Nidhi will consider as delimeter or ending mark and scanf consider the input Nidhi only. Scanf is not capable to input multiword string