ACHARYA

Dept of ISE

ACHARYA

BENGALURU, INDIA

WHERE THE WORLD COMES TO LEARN

MODULE 4 Pointers

ACHARYA

Dept of ISE

Pointers

- The basic data types in C language are int float char, double and void.
- Pointer is a special data type which is derived from these data types.
- So pointer is called derived data type.
- The pointers are always associated with the following three concept.

➢ Pointer constants

➢ Pointer values

➢ Pointer variables

- The computer memory is divided into a number of locations called **storage cells.**
- Each location can hold one byte of information and each location is associated with address.
- These memory address are called **pointer constants**.
- These memory location assigned to variable by the system are called **pointer values.**
- A variable which holds the address of another variable is called a **pointer variable**.

ACHARYA

Dept of ISE

Example: If i is a variable and address of i is stored in a variable P

p=&i;

Then the variable p is called a **Pointer variable.**

Accessing variables through pointers

Steps to be followed while using pointers

Declare a data variable

Ex: int a;

Declare a pointer variable

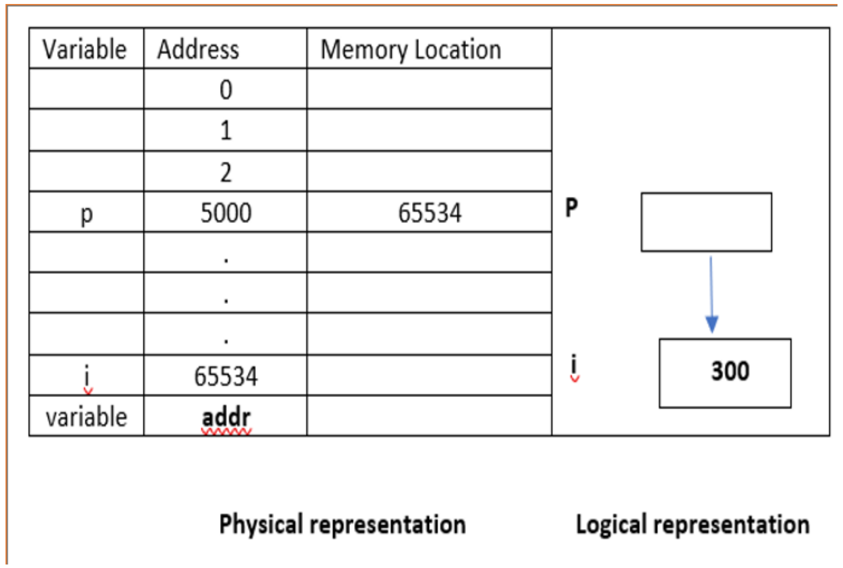
Ex: int *p;

Initialize a pointer variable

Ex: p=&a;

Access data using pointer variable

Ex: printf("%d",*P);



Variable	Address	Memory Location
	0	
	1	
	2	
p	5000	65534
	.	
	.	
	.	
i	65534	
variable	addr	

Physical representation

Logical representation

ACHARYA

Dept of ISE

Pointers declaration and definition:

- Pointer variables should be declared before they are used.

  Syntax: type * identifier;

Example 1: int *p;
double *pd;

In the declaration, the position of * is immaterial.

Example 2: int *pa;
int * pa;
int* pa;

Example 3: int* pa,pb,pc;

```
    int *pa;          //pa is the pointer variable
int pb;    //pb and pc are ordinary variable
int pc;
```

ACHARYA

Dept of ISE

Dangling Pointer
Example:

```
        int *p;

    p              Garbage value
```

A pointer variable should contain a **valid address**.
A pointer variable which does not contain a valid address is called **dangling pointer.**

ACHARYA

Dept of ISE

Initializing a pointer variable

Initializing of a pointer variable is the process of assigning the address of a variable to a pointer variable.

Declare a data variable

Ex: int x;

Declare a pointer variable

Ex: int *px;

Assign address of a data variable to pointer variable using & operator and assignment operator.

Ex: px=&x;

OR

int x,*px=&x;

Example: int p,*ip;
float d,f;
ip = &p; **/* Correct */** ip= &d; **/* wrong*/**

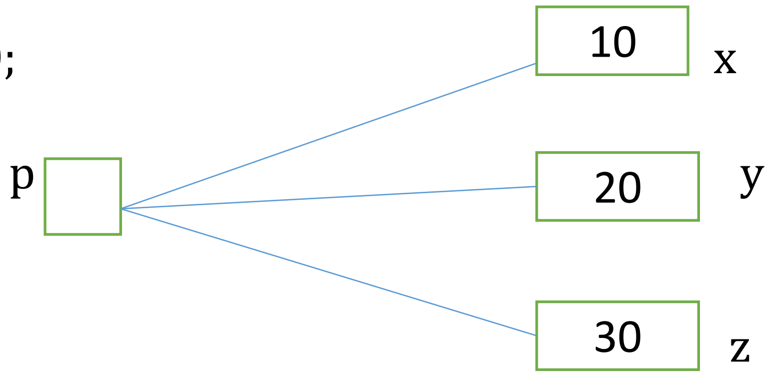
ACHARYA

Dept of ISE

Pointers are flexible: i.e pointer can point to different data Variables by storing the address of appropriate variables.

Example:

```
int x=10,y=20,z=30;
Int *p;
P=&x;
P=&y;
P=&z;
```



NULL pointer:-

A NULL pointer is defined as a special pointer value that point to nowhere in the memory.

Example: #include<stdio.h>

```
          int *p=NULL
      if(p==NULL)     //Error condition
      printf("P does not point to any memory\n");
      else
       printf("Access the value of P\n");
```

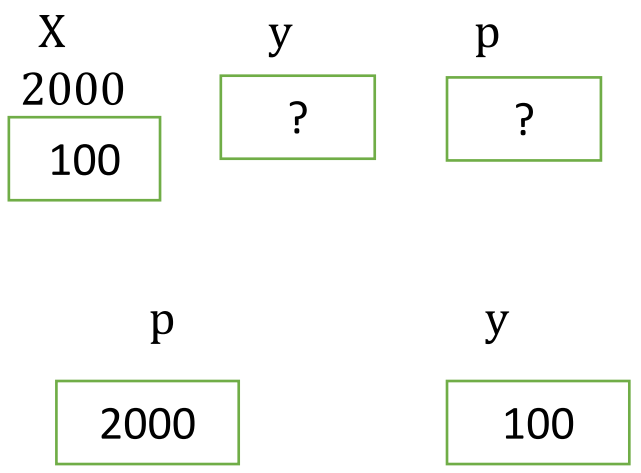
ACHARYA

Dept of ISE

Accessing variables through pointers

The value of a variable can be accessed using pointer variable using unary operator *(called asterisk)

```
Example:  int x=100,y;
           int *p;
          p=&x;
         y=*p;
```



ACHARYA

Dept of ISE

/* C program to add two number using pointers */

```
#include <stdio.h>
 int main()
{
 int num1, num2, sum;
 int *ptr1, *ptr2;
ptr1 = &num1; // ptr1 stores the address of num1
ptr2 = &num2; // ptr2 stores the address of num2
printf("Enter any two numbers: ");
scanf("%d%d", ptr1, ptr2);

sum = *ptr1 + *ptr2;

printf("Sum = %d", sum);
 return 0;
}
```

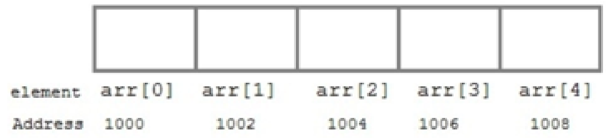
ACHARYA

Dept of ISE

Arrays and pointers:

int arr[5] = { 10, 20, 30, 40, 50};

arr=1000



10 20 30 40

50

arr=&arr[0]=1000

*p;

p=x; is equal to p=&x[0];

Now we can access every value of x using p++ to move from one element to another.

p=&x[0](=1000)

p+1=&x[1](=1002)

p+2=&x[2](=1004)

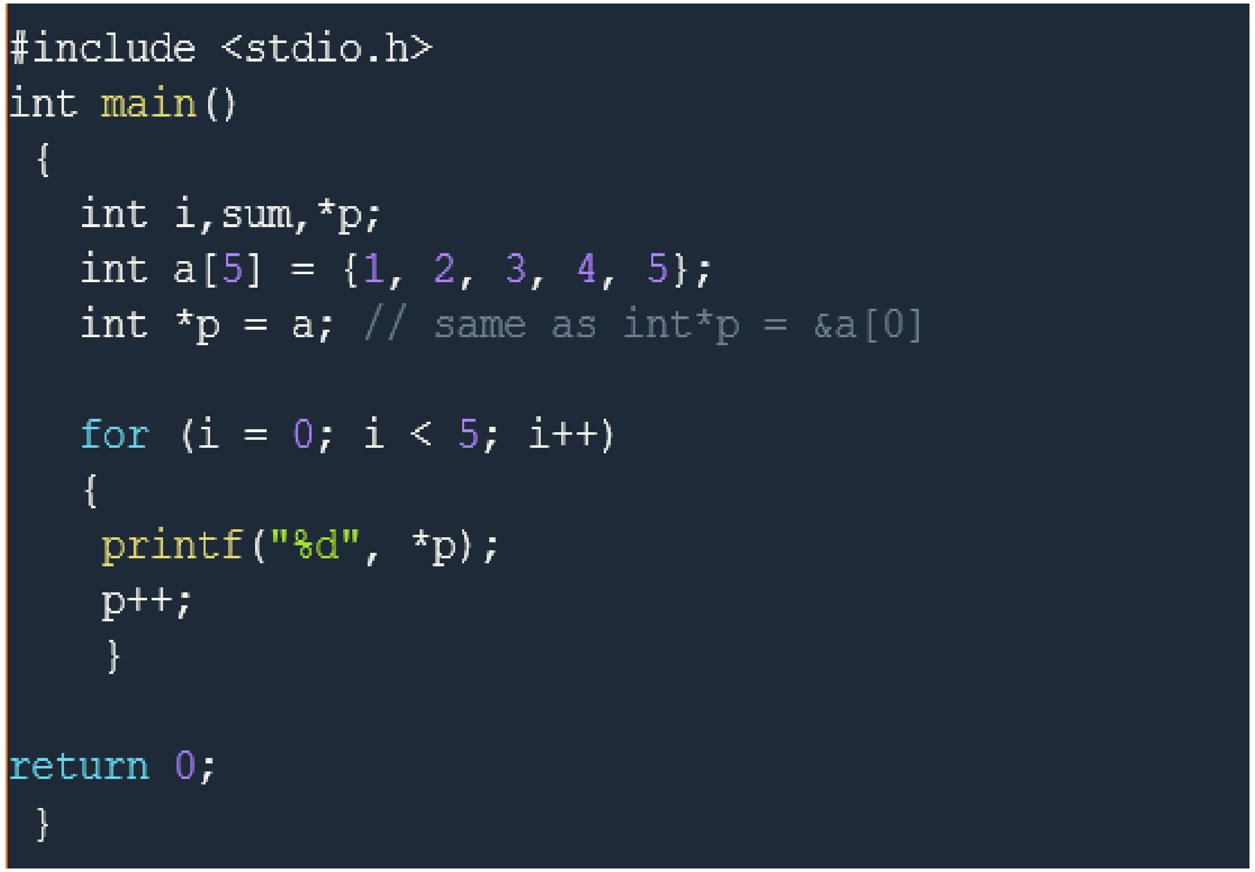
p+3=&x[3](=1006)

p+4=&x[4](=1008)

ACHARYA

Dept of ISE

Traversing an array by using pointers



```
#include <stdio.h>
int main()
 {
   int i,sum,*p;
   int a[5] = {1, 2, 3, 4, 5};
   int *p = a; // same as int*p = &a[0]

   for (i = 0; i < 5; i++)
   {
    printf("%d", *p);
    p++;
    }

return 0;
 }
```

OUTPUT:

```
1
2
3
4
5
```

ACHARYA

Dept of ISE

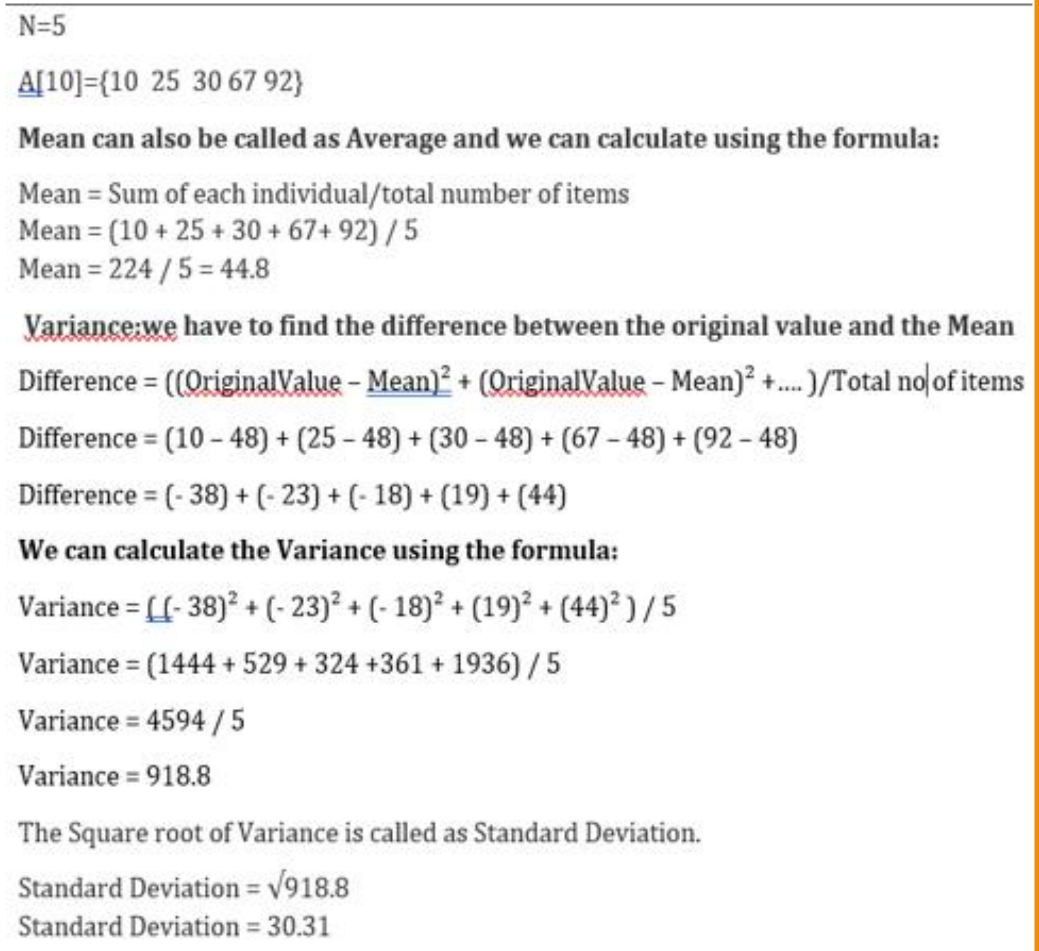
Algorithm 14: To compute the sum, mean and standard deviation of all elements stored in an array of n real numbers.

$$\text{mean} = \frac{\sum_{i=0}^{n-1} a[i]}{n}$$

$$\text{variance} = \frac{\sum_{i=0}^{n-1} (a[i] - mean)^2}{n}$$

$$\text{standard deviation} = \sqrt{\text{variance}}$$

Lab Program-14. Develop a program using pointers to compute the sum, mean and standard deviation of all elements stored in an array of 'n' real numbers.



N=5

A[10]={10 25 30 67 92}

Mean can also be called as Average and we can calculate using the formula:

Mean = Sum of each individual/total number of items
Mean = (10 + 25 + 30 + 67+ 92) / 5
Mean = 224 / 5 = 44.8

Variance:we have to find the difference between the original value and the Mean

Difference = ((OriginalValue – Mean)^2 + (OriginalValue – Mean)^2 +....)/Total no of items

Difference = (10 – 48) + (25 – 48) + (30 – 48) + (67 – 48) + (92 – 48)

Difference = (- 38) + (- 23) + (- 18) + (19) + (44)

We can calculate the Variance using the formula:

Variance = ((- 38)^2 + (- 23)^2 + (- 18)^2 + (19)^2 + (44)^2) / 5

Variance = (1444 + 529 + 324 +361 + 1936) / 5

Variance = 4594 / 5

Variance = 918.8

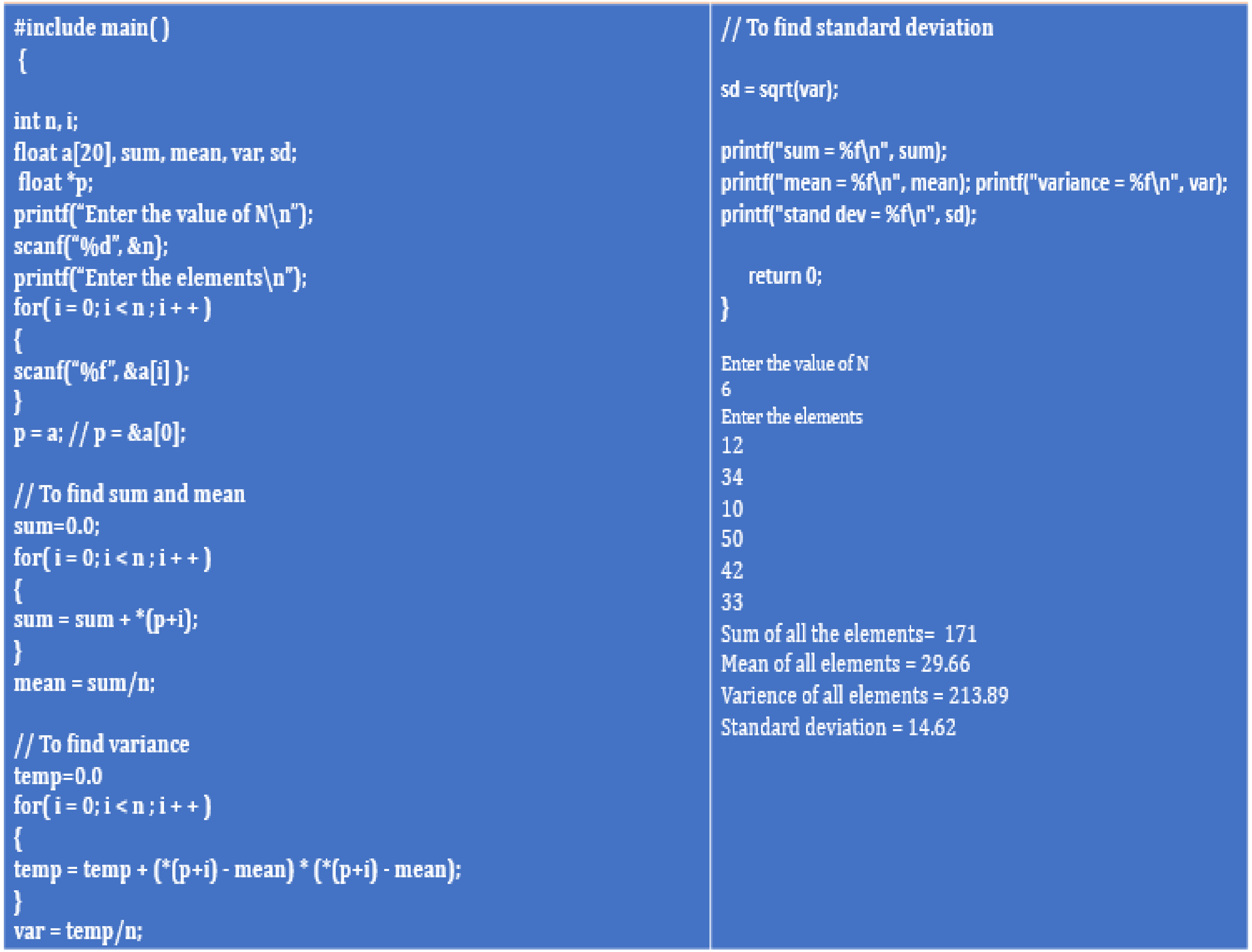
The Square root of Variance is called as Standard Deviation.

Standard Deviation = √918.8
Standard Deviation = 30.31

ACHARYA

Dept of ISE

```
Step 1: [Start]
Step 2: [Input the elements]
            Read n, a[i]
Step 3:     [Initialize]
            sum=0, temp=0
Step 4: [Compute sum, mean, sta dev]
        Ptr=a
        For i=0 to n
 steps of 1 sum=sum+*ptr
             Ptr++
             Mean=sum/n
             Ptr=a
             For i=0 to n in steps of 1
temp=temp+pow((*ptr-mean),2)
            Ptr++
                 var=temp/n
            std= sqrt(sumstd/n)
Step 5: [Output]
Print sum, mean, standard deviation
Step 6:     [Stop]
```



```
#include main( )
 {

int n, i;
float a[20], sum, mean, var, sd;
 float *p;
printf("Enter the value of N\n");
scanf("%d", &n);
printf("Enter the elements\n");
for( i = 0; i < n ; i + + )
{
scanf("%f", &a[i] );
}
p = a; // p = &a[0];

// To find sum and mean
sum=0.0;
for( i = 0; i < n ; i + + )
{
sum = sum + *(p+i);
}
mean = sum/n;

// To find variance
temp=0.0
for( i = 0; i < n ; i + + )
{
temp = temp + (*(p+i) - mean) * (*(p+i) - mean);
}
var = temp/n;
```

```
// To find standard deviation

sd = sqrt(var);

printf("sum = %f\n", sum);
printf("mean = %f\n", mean); printf("variance = %f\n", var);
printf("stand dev = %f\n", sd);

    return 0;
}
```

```
Enter the value of N
6
Enter the elements
12
34
10
50
42
33
Sum of all the elements= 171
Mean of all elements = 29.66
Varience of all elements = 213.89
Standard deviation = 14.62
```

ACHARYA

Dept of ISE

ACHARYA

Dept of ISE

/* Pointers and functions */

There are two ways of passing parameters to the functions

- Pass by value (also called call by value)
- Pass by reference (also called call by reference)

Pass by value (call by value)

- When a function is called with **actual parameters**, the values of actual parameters are copied into formal parameters.

- If the values of the **formal parameters** changes in the functions, the values of the actual parameters are not changed.

- This way of passing parameters is called **pass by value or call by value**

ACHARYA

Dept of ISE

C program swapping of two numbers

Calling function

```
#include<stdio.h>
Void exchange(int *m, int *n);

void main()
{
     int a,b;
     a=10, b=20;

    printf("Before exchange");
    printf("a=%d and b=%d\n",a,b);

    exchange(&a,&b);

    printf("After exchange");
    printf("a=%d and b=%d\n",a,b);
}
```

OUTPUT

```
Before exchange
  a=10 and b=20
After exchange
 a=20 and b=10
```

Called function

```
Void exchange(int *m,int *n)
{
int temp;
printf("Before exchange\n");
printf("m=%d and n=%d\n",m,n);

temp=m;
 m=n;
 n=temp;

printf("After exchange\n");
printf("m=%d and n=%d\n",m,n);

}
```

```
Before exchange
  m=10 and n=20
After exchange
 m=20 and n=10
```

ACHARYA

Dept of ISE

Pass by reference (call by reference)

- In pass by reference, a function is called with addresses of actual parameters.
- In the function header, the format parameters receive the addresses of actual parameters.
- Now, the formal parameters do not contain values, instead they contain addresses.
- Any variable if it contains an address, it is called a **pointer variable.**
- Using pointer variables, the values of the actual parameters can be changed.
- This way of passing parameters is called **Pass by reference or call by reference.**

ACHARYA

Dept of ISE

C program swapping of two numbers

Calling function

```
#include<stdio.h>
Void exchange(int *m, int *n);

void main()
{
     int a,b;
     a=10, b=20;

    printf("Before exchange");
    printf("a=%d and b=%d\n",a,b);

    exchange(&a,&b);

    printf("After exchange");
    printf("a=%d and b=%d\n",a,b);
}
```

OUTPUT

```
Before exchange
  a=10 and b=20
After exchange
 a=20 and b=10
```

Called function

```
Void exchange(int *m,int *n)
{
int temp;
printf("Before exchange\n");
printf("m=%d and n=%d\n",m,n);

temp=m;
 m=n;
 n=temp;

printf("After exchange\n");
printf("m=%d and n=%d\n",m,n);

}
```

```
Before exchange
  m=10 and n=20
After exchange
 m=20 and n=10
```

ACHARYA

Dept of ISE

Functions returning pointers

- A function can return a single value using return statement or multiple values using pointers in parameters.
- Since pointer is also a data type in C, we can return a pointer from a function.

Example:-Program to return pointer to larger of two numbers.

```
void main()
{
int x,y,*big;
printf("Enter the value of x & y\n");
scanf("%d%d",&x,&y);
big=largest(&x,&y);
printf("Maximum(%d,%d)=%d\n",x,y,*big);
}
int *largest(int *a, int *b)
{
if (*a>*b)
      return a;
 else
       return b;
}
```

ACHARYA

Dept of ISE

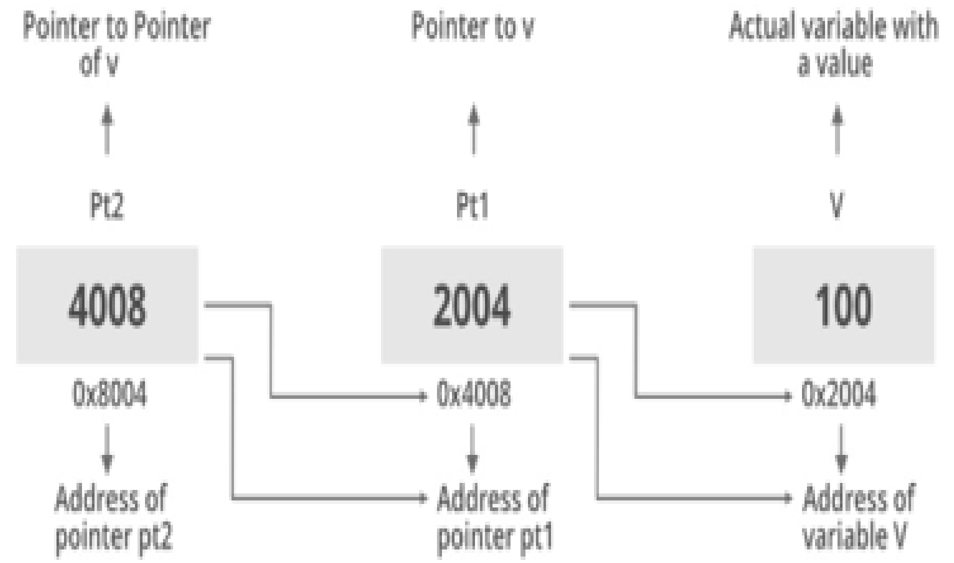
/* Pointer to pointer */

It is possible to make a pointer to point to another pointer variable.

A variable which contains address of a pointer variable is called pointer to pointer.

Example:

```
 int v;
 int  *pt1;
 int  **pt2;
void main()
{
int a;
int *p1;
int **p2;
a=10;
p1=&a
p2=&p1;
printf("a=%d",a);                    //a=10
printf("*p1=%d\n",*p1);          //*p1=10
Printf("**p2=%d\n",**p2);       //**p2=10
}
```

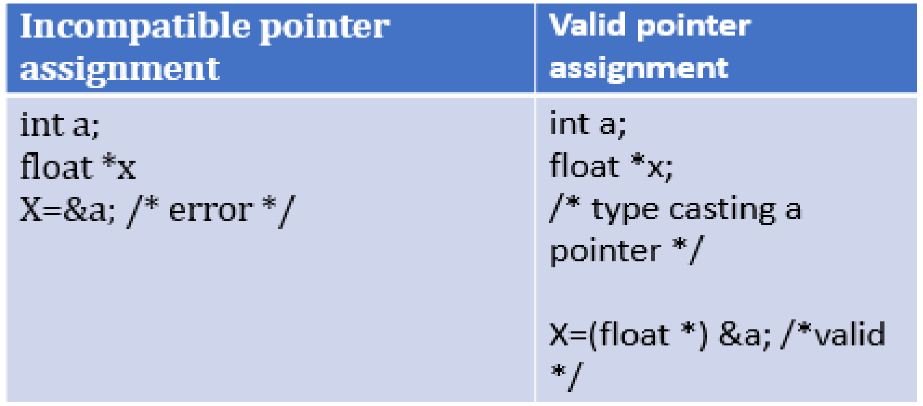


ACHARYA

Dept of ISE

Compatibility and void pointer

- As we convert the type of one data variable into type of another data variable using type casting.
- We can make an assignment between incompatible pointer types using explicit type casting.



Incompatible pointer assignment	Valid pointer assignment

int a;
float *x
X=&a; /* error */
	int a;
	float *x;
	/* type casting a pointer */
	
	X=(float *) &a; /*valid */

Void pointer:

- A void pointer is a special type of pointer.
- It points to any data type.
- Void pointer is also called universal pointer or generic pointer.

Example:

```
void *p;
```

ACHARYA

Dept of ISE

```
void main()
{
        int x=10;
        double y=3.14156;
        void *p;
        p=&x;
        printf("x=%d\n",*((int *)p));
}
```

An object that occurs on the left hand side of assignment operator is called **L-value.**

Anything that occur on the right hand side of assignment statement is called **R-value.**

ACHARYA

Dept of ISE

Pointer Expressions:-

Like other variables, pointer variables can be used in expressions.

Example:

- **declared and initialized pointer.**

```
y=*p1  *  *p2;
sum=sum + *p1;
z=5* - * p2/*p1;
*p2 = *p2 + 10;
```

- **Add integer to or subtract integer from pointer.**

```
p1+4
 p2-2
 p1-p2
```

- **Short –hand operator with the pointers.**

```
 p1++;
 --p2 ;
sum+=*p2;
```

- **Pointers can also be compared using the relational operator.**

  p1>p2,p1==p2 & p1!=p2

- **We may not use pointers in division or multiplication.**

  p1/p2 or p1*p2 or p1/3 are not allowed.

- **Two pointers cannot be added.**

  i.e p1+p2 is illegal.

ACHARYA

Dept of ISE

```
#include<stdio.h>

int main()
{
int n1=50,n2=20,res;
int *p1,*p2;//pointer to int

p1=&n1;//stores the address of  variable
P2=&n2;
printf("Address of n1 and n2 is %u\t%u",p1,p2);

res = *p1 *  *p2-6;
printf("Result=%d\n",res);
return 0;
}
```

```
#include<stdio.h>
void main ()
{
   int i = 100;
   int *p = &i;
   int *temp;
   temp = p;
   p = p + 3;
   printf("Pointer Subtraction:
             %d - %d = %d",p, temp, p-temp);
}
```

OUTPUT:

```
Pointer Subtraction:
1030585080 - 1030585068 = 3
```

```
#include<stdio.h>

int main()
{
int number=50;
int *p;//pointer to int

p=&number;//stores the address of  variable

printf("Address of p variable is %u \n",p);

p=p+3;   //adding 3 to pointer variable

printf("After adding 3: Address of p is %u \n",p
);
return 0;
}
```

OUTPUT:

```
Address of p variable is 3214864300
 After adding 3:
Address of p variable is 3214864312
```

```
#include<stdio.h>

int main()
{
int number=50;
int *p;//pointer to int

p=&number;//stores the address of number variable

printf("Address of p variable is %u \n",p);
p=p-3; //subtracting 3 from pointer variable
printf("After subtracting 3: Address of p is %u \n",p);

return 0;
}
```

OUTPUT:

```
Address of p variable is 3214864300 After
subtracting 3: Address of p variable is 321486428
```

ACHARYA

Dept of ISE

```
#include<stdio.h>

int main()
{
int n1=50,n2=20,res;
int *p1,*p2;//pointer to int

p1=&n1;//stores the address of  variable
P2=&n2;
printf("Address of n1 and n2 is %u\t%u",p1,p2);

res = *p1 *  *p2-6;
printf("Result=%d\n",res);
return 0;  }
```

```
#include<stdio.h>
void main ()
{
   int i = 100;
   int *p = &i;
   int *temp;
   temp = p;
   p = p + 3;
   printf("Pointer Subtraction:
             %d - %d = %d",p, temp, p-temp);
}
```

OUTPUT:

```
Pointer Subtraction:
1030585080 - 1030585068 = 3
```

```
#include<stdio.h>

int main()
{
int number=50;
int *p;//pointer to int

p=&number;//stores the address of  variable
printf("Address of p variable is %u \n",p);

p=p+3;   //adding 3 to pointer variable

printf("After adding 3: Address of p is %u \n",p);
return 0;
}
```

OUTPUT:

```
Address of p variable is 3214864300
 After adding 3:
Address of p variable is 3214864312
```

```
#include<stdio.h>

int main()
{
int number=50;
int *p;//pointer to int

p=&number;//stores the address of number variable
printf("Address of p variable is %u \n",p);
p=p-3; //subtracting 3 from pointer variable
printf("After subtracting 3: Address of p is %u \n",p);
return 0;
}
```

OUTPUT:

```
Address of p variable is 3214864300 After subtracting 3: Address of p variable is
321486428
```

ACHARYA

Dept of ISE

Pointers can be used to manipulate two – dimensional arrays as well .

One dimensional array x, the expression.

*** (x+i) or *(p+i)   = x[i]**

Two dimensional array can be represented by the pointer expression as follows:

((a+i)+j) or *(*(p+i)j)

((p+i)+j)→value stored in the cell (i,j)

p→ pointer to first row
p+i→ pointer to ith row
*(p+i)→ pointer to first element in the ith row

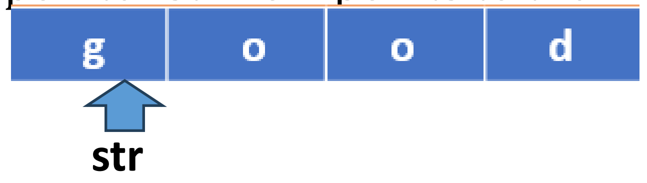
*(p+i)+j→ pointer to jth element in the ith row

Pointers and Character Strings

C supports an alternative method to create strings using pointer variables of type char.

char *str="good";

The pointer str now points to the first character of the string "good: as



printf("%s", str); //puts(str);

Write a program using pointer to determine the length of a character string.

```
main()
{
  char *name;
   int length;
   char *cptr = name;
    name = DELHI";
   printf("%s\n",name);
   while(*ctr!='\0')
   {
   printf("%c is stored at address %u\n",*cptr,cptr);
  cptr++;
   }
   length = cptr-name;
   printf("\n length of the string = %d\n",length);
}
```

ACHARYA

Dept of ISE

ACHARYA

Dept of ISE

Pointer to functions

A function, like a variable, has a physical location in the memory..

Like assigning address of a variable to pointer, it is possible to assign address of a function to a pointer.

Once a pointer points to a function, the function can be invoked using this pointer

Using function pointer, it is possible to pass functions as parameters .

Syntax: **type (*fp)();**

type : type of value returns from the function

*fp is pointer to a function

ACHARYA

Dept of ISE

```
#include<stdio.h>
int addition ();
int main ()
{
   int result;
   int (*ptr)();
   ptr = &addition;
   result = (*ptr)();
   printf("The sum is %d",result);
}
int addition()
{
   int a, b;
   printf("Enter two numbers?");
   scanf("%d %d",&a,&b);
   return a+b;
}
```

```
Enter two numbers?

10 and 15

The sum is 25
```

ACHARYA

Dept of ISE

Pointers and Structures

The name of an array stands for the address of its zeroth element.

The same thing is true of the name of arrays of structure variables.

Suppose product is an array variable of struct type.

The name product represents the address of its zeroth element.

Consider the following declaration:

```
struct inventory
{
char name[30];
int number[30];
float price;
}product[2],*ptr;
ptr=product;
```

Its members can be accessed using the following

```
ptr→name
ptr→number
ptr→price
```

ACHARYA

Dept of ISE

/*Write a program to illustrate the use of structure pointer */

```
struct invent
        {
        char name[30];
        int number[30];
        float price;
        };

main()
{
struct invent product[3],*ptr;
printf("INPUT \n\n");

for(ptr=product;ptr<product+3;ptr++)
scanf("%s%d%f",ptr→name,&ptr→number,&ptr→price);

printf("\n OUTPUT \n\n");
ptr=product;
while(ptr<product+3)
{
printf("%-20s%5d%10.2f\n",ptr→name, ptr→number, ptr→price);
ptr++;
}
}
```

ACHARYA

Dept of ISE

Thank you!