Pointers:

- Derived Data type
- Used to store address of memory location
- It has added power and flexibility to the language
- Powerful tools

Benefits of Pointers:

- More efficient in handling arrays
- Used to return multiple values from functions
- Facilitate pass by reference mechanism
- Supports Dynamic memory management
- Reduce the length of the program
- Increase the speed of execution.

Understanding pointes:

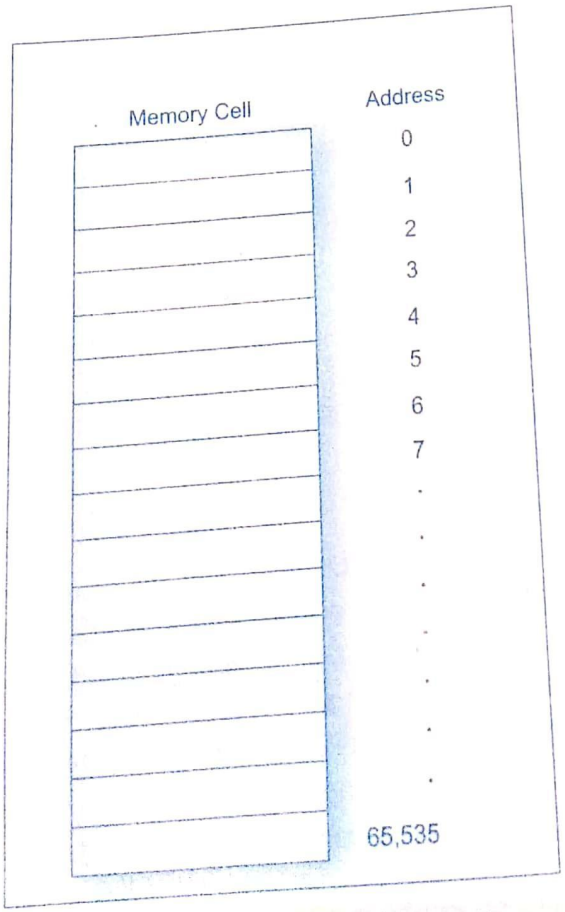
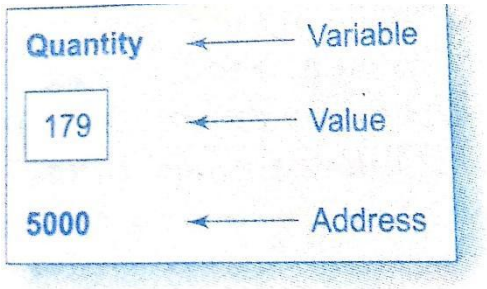
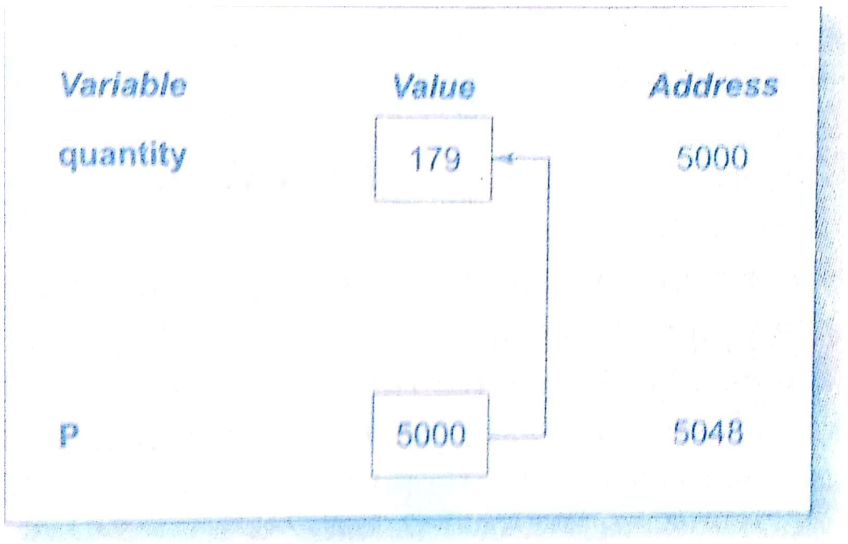


Fig. 11.1 Memory organisation

int quantity = 179; scanf("%d", &quantity);

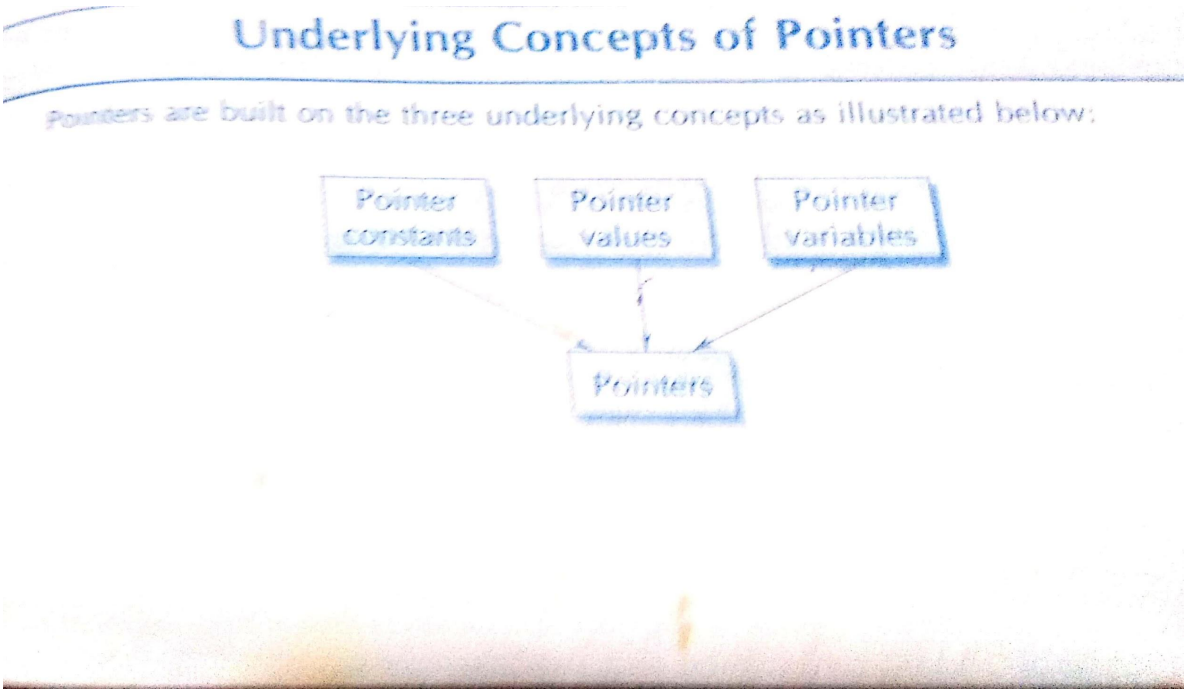


11.2 Representation of a variable



=

```
int quantity = 179;
 int    *P ;
       P = &quantity;      printf("%d", quantity);
       Printf("address of quantity = %u", &quantity)// 5000
         Printf("P = %u", P); //  5000
         Printf("P  address is %u", &P); // 5048
```



Declaring Pointers:

Datatype* pointername; // int* ptr; float* ptr;

Datatype *pointername;// int *ptr; float *ptr;

Datatype * pointername; // int * ptr; float * ptr;

Initialization of pointers: Can be initialized only to address, using & operator. // int x ,*p1; p1 = &x;

Initial value of NULL or 0 can be given. // int *p = NULL;

P= NULL;

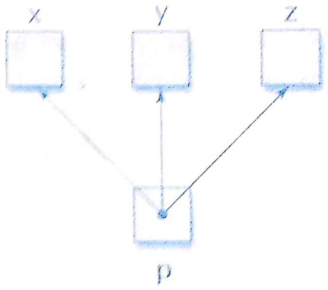
P= 0;

int *p = 0;

Pointer Flexibility

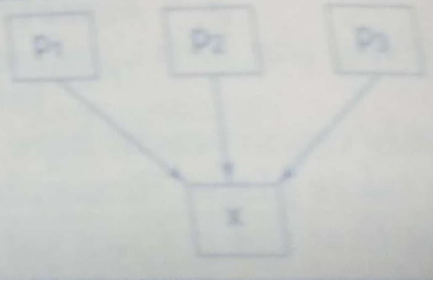
Pointers are flexible. We can make the same pointer to point to different data variables in different statements. Example;

```
int x, y, z, *p;
.....
p = &x;
.....
p = &y;
.....
p = &z;
.....
```



We can also use different pointers to point to the same data variable. Example.

```
int x;
int *p1 = &x;
int *p2 = &x;
int *p3 = &x;
.....
.....
```



Accessing a variable though a pointer:

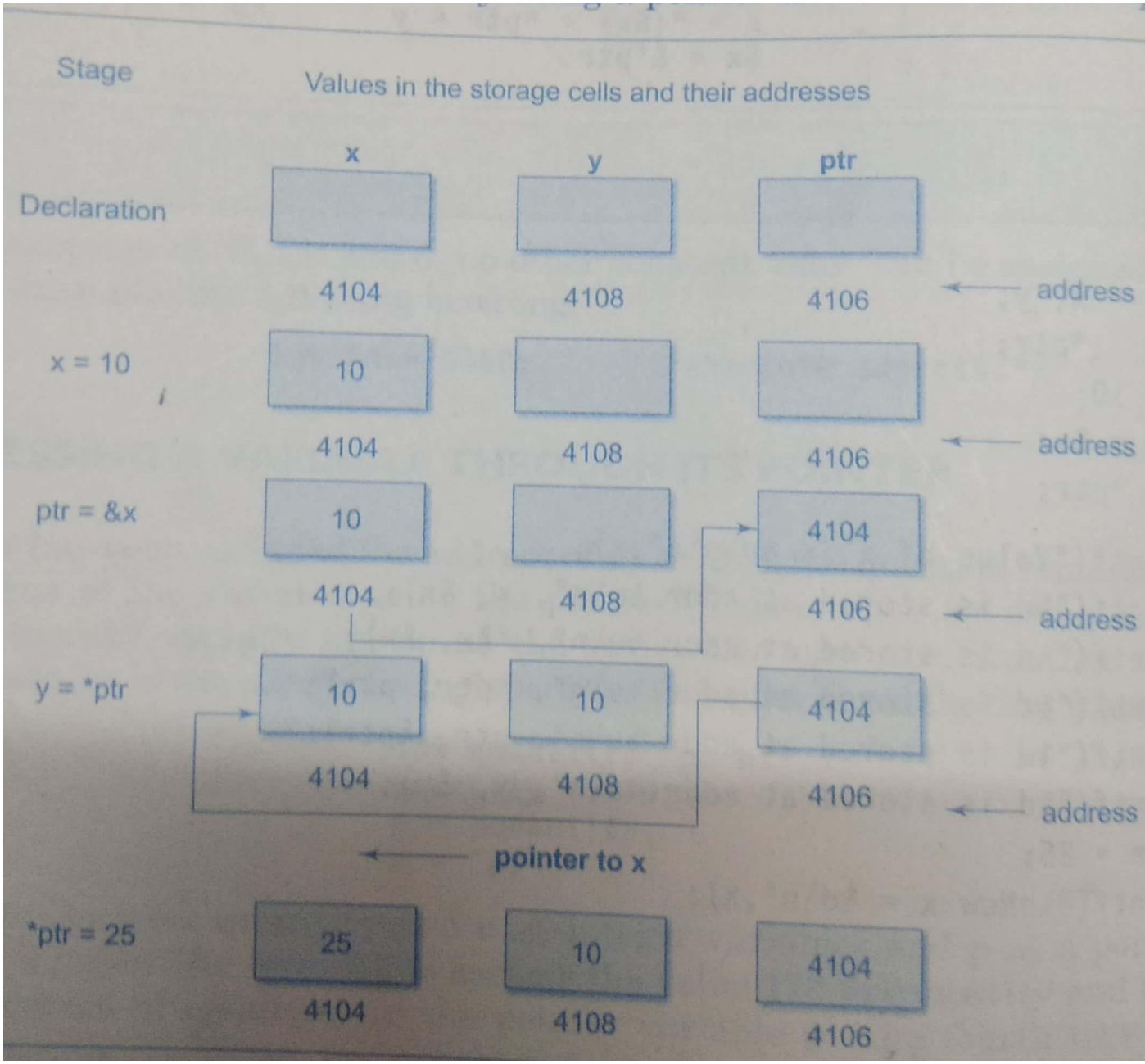
Using indirection operator(*)

int X, *p1, *p2, *p3;

X=10; // add x = 5000

```
   p1 = &X;   // p1?   Add p1 5048
   printf("X = %d", X);
   printf("X=%d",  *p1);   //   *(5000)
   *p1 =  *p1 +15;  //   X = X+ 15;
  Printf("x = %d", X) // 25
  printf("X=%d",  *p1); // 25
   p2 = &X;   // p2 = 5000
   p3 = &X;   // p3= 5000
     *p2= *p2 + 2 ; //  27
 * p3 = * p3 + 1; // 28
Printf(" X = %d", *p2); //  28
Printf("x = %d", *p1);// 28
 Printf(" X = %d", *p3); // 28


Int   x, y,  *P;
X= 10, Y=20;
P=  &X;
*P = *P +1;
Printf("%d", *P) // 11
P = &Y;
```

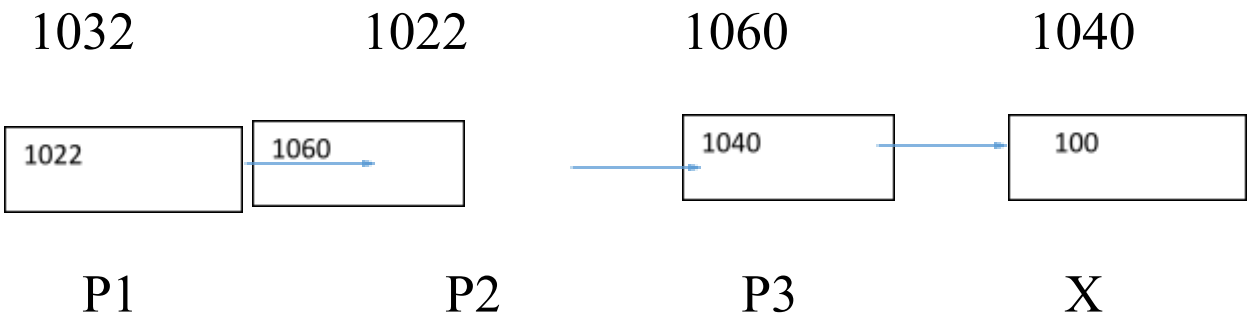
```
        *P = *P +1;
Printf(“%d”, *P);  //  21
```



Ptr = &y; // ptr = 4108

*ptr = 30; // y = 30

Chain of pointers:



Int X = 100, *P3, **P2 , ***P1 ;

P3 = &X;

P2 = &P3;

P1 = &P2;

Witing:

Printf(“ X = %d”, X);

printf(“X = %d”, *P3) // *(1040)

printf(“X = %d”, **P2) // * (*(1060)) = *1040 =100

printf(“X = %d”, ***P1) // *(*(*1022)) * *1060 = *1040=100

Reading:

scanf(“%d”, &x) // 1040

scanf(“%d”, P3) // 1040

scanf(“%d”, *P2) // *1060 =1040

scanf(“%d”, **P1) // * *1022 =* 1060=1040

scanf(“%d”, &X); // scanf(“%d”, **P1);

scanf(“%d”, P3); // scanf(“%d”, *P2);

Pointer Expressions:

X= *p1 + *p2;

Y= *p1 *   *p2; //   p1 * p2

Z= 10 -*p1/ *p2

P1 = p1-1 // P1 = 1020   // int x, *p1;   p1 =&x; P1 = P1+1

We can add or subtract integers

We can subtract one pointer from another(provided the two pointers are pointing to the same array

Short hand operators can be applied:

++p1, p1--,   (Incrementing and decrementing pointers make sense only when a pointer is pointing to a collection)

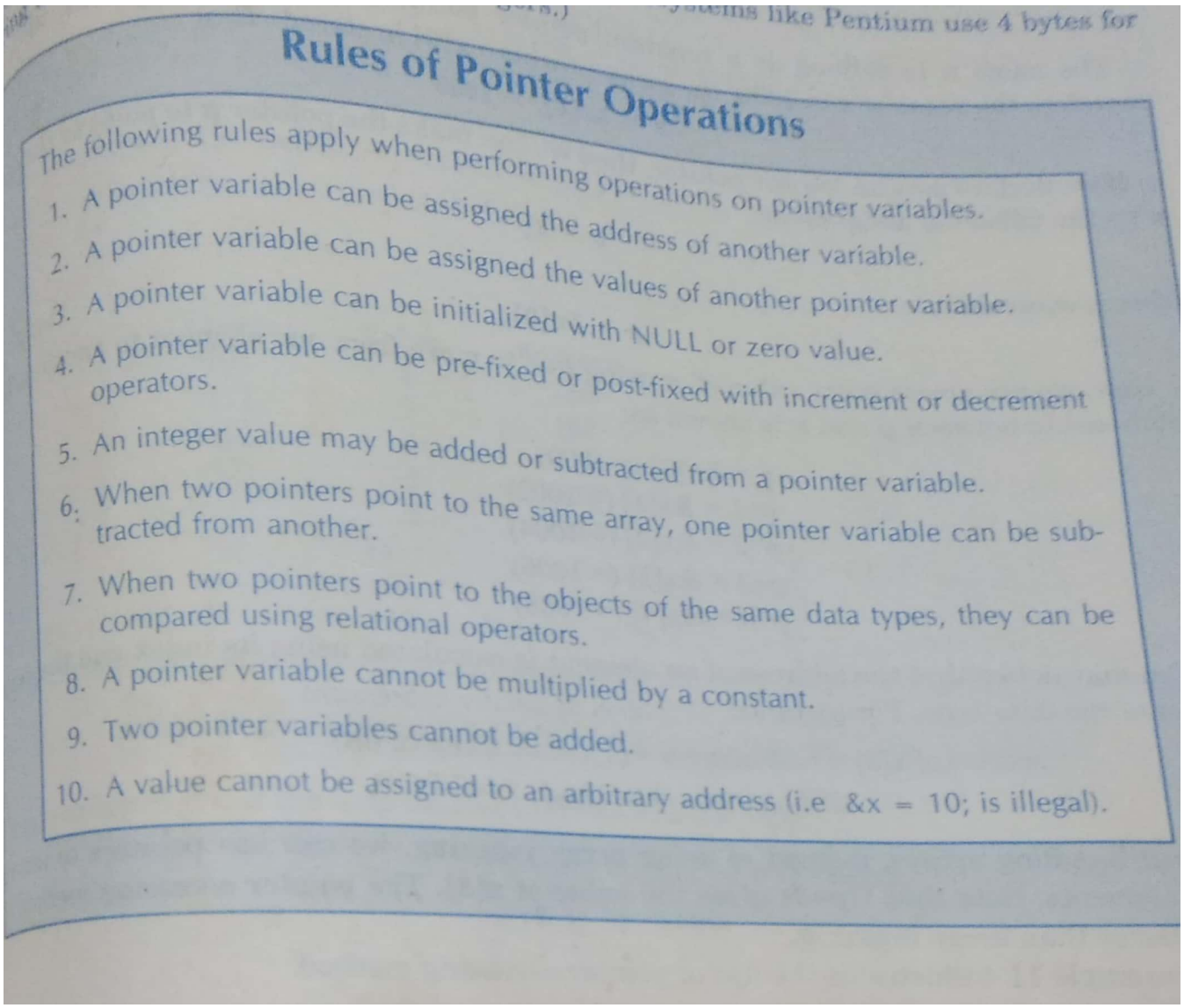
sum += *p1

Pointers can be compared using relational operators.

P1 > P2 ; (valid only p1 and p2 are pointing to same collection)

P1 == P2

P1 != P2



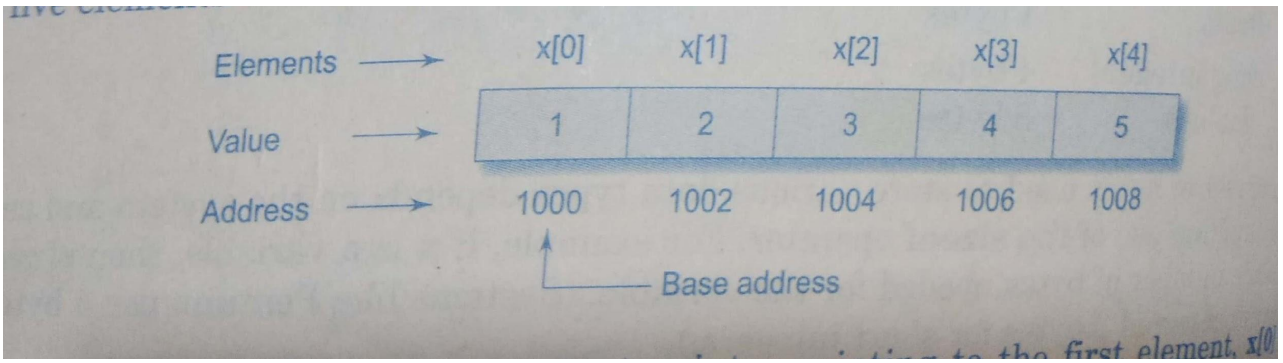
...stems like Pentium use 4 bytes for

Rules of Pointer Operations

The following rules apply when performing operations on pointer variables.

1. A pointer variable can be assigned the address of another variable.
2. A pointer variable can be assigned the values of another pointer variable.
3. A pointer variable can be initialized with NULL or zero value.
4. A pointer variable can be pre-fixed or post-fixed with increment or decrement operators.
5. An integer value may be added or subtracted from a pointer variable.
6. When two pointers point to the same array, one pointer variable can be subtracted from another.
7. When two pointers point to the objects of the same data types, they can be compared using relational operators.
8. A pointer variable cannot be multiplied by a constant.
9. Two pointer variables cannot be added.
10. A value cannot be assigned to an arbitrary address (i.e &x = 10; is illegal).

Pointers and Arrays:




```
int  x[10];          X = 1000 = &X[0]
int  *P;
P = X       // &X[0]
For(i=0; i<N; i++)
    //  Scanf("%d",  &X[i]);
        Scanf("%d",  P+i  ); // i=1   1000+1
    Printf("%d ",   X[i]);
    Printf("%d  ", *(P+i))
```

Address of any ith element can be obtained by

Add of x[i] = Base add + i * size;

X[0]= 1000+0*2= 1000

X[4] = 1000 + 4*2 = 1008

Accessing ith element using p : *(p+i)

Note: Pointer accessing method is much faster than indexing

Pointers to 2D Arrays:

Int A[10][10]; // R = 3 C = 2

Address of A[i][j] = Base add + (i * no_of_ columns + j) * size

While accessing the element using pointer:

A[i][j] = (p + i * no_of_col + j)

A[2][1]	1010
A[2][0]	1008
A[1][1]	1006
A[1][0]	1004
A[0][1]	1002
A[0][0]	1000

A[0][0] = 1000 + 0 = 1000

A[0][1] = 1000 + (0*2+1)* 2 = 1002

A[1][1] = 1000 + (1 *2 +1)*2 = 1006

A[2][1] = 1000 + (2 * 2 + 1) * 2 = 1010

A[2][1] = 1000 + (2 *2 +1)*2 = 1010

```
Int *P;
P = &A[0][0];   P = A;
```

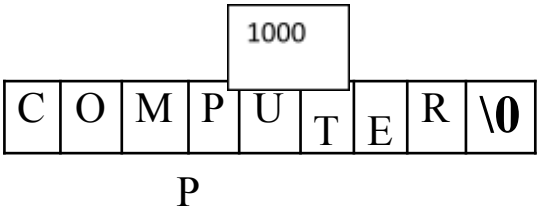
```
For( i = 0; i<R; i++)
    For(j=0; j<C; j++)
      Scanf("%d", &A[i][j]);
  // scanf("%d", (P+i*C+j));

For( i = 0; i<R; i++)
    For(j=0; j<C; j++)
      printf("%d  ", A[i][j]);
      printf("%d  ",    *(P+i*C+j));
```

Pointers to Character Arrays:

Char S[10]; // char S[ ]="COMPUTER"; int x= 10

Char S[ ]= {'C', 'O'……,'\0'}

S= "COMPUTER" scanf("%s", S) // strcpy("S, "COMPUTER");



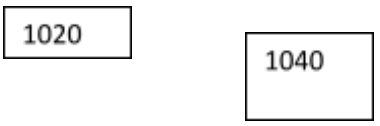

```
char  S[10], *P;
 P = S;
P= "COMPUTER";     scanf("%s", P);
```

```
Printf("%s", P);
```

Pointers as Function arguments:

```
Void   swap(int* , int*);
void main()
{
   Int x=10, y=20;
```


```
// &x = 1020   &y = 1040
Printf("  Before exchange:  X = %d y = %d\n", x, y); // 10 20
Swap(&x, &y);
 Printf("  After exchange:  X = %d y = %d\n", x, y);    //20  10
}
 Void swap(int *x, int *y)
```




```
  {
    /// // &x = 1050  &y = 106    //  int *x = &x = 1020

      Int temp;
       Temp =  *x;
       *x =  *y;
      *y= temp;
  Printf("  After exchange in swap :  X = %d y = %d\n", *x, *y); // 20 10

}
```

Function returning pointers:

```
Int *  Max(int *, int *y);
Void main()
{
   Int x, y,  *P;        x = 30   y = 40    &x = 1020   &y = 1030
   Scnaf('%d %d", &x, &y);
  P =Max(&x, &y);   // P = 1030
 Printf("Large = %d", *P);
}
 Int * Max(int *A, int *B)    // A = 1020   B = 1030
{
    If(*A > *B)  return A ;
   Return   B;
}
```

Pointers to functions:

```
    Int  a;   &a = 1012
   Int *p;  P = &a;
```

Int fun(int x); &fun =1090

Declaration: /// struct student (*ptr)();

Datatype (*ptr) ( ) ; // int (*ptr)( ); float (*ptr1)(); // int * (*p3)()

Int * (*ptr2)();

Datatype * ptr(); // int * ptr();// wrong

Assigning pointer to function:

Ptr = function name;

ptr = fun;

p3= Max;

fun(10);

(*ptr1)(10); // (*ptr1)(a);

(*p3)(&x, &y);

Pointers to structures:

```
Struct student
{
    Int regno;
    Char name[10];
    Int totalmarks;
};
```

Struct student S1; S.regno, S.name S.totalmarks

Int X; int *p;

P = &X; scanf("%d", P) printf("%d", *P);

Struct student *sp;

sp = &S1

sp->regno sp->name sp->totalmarks

87	1012
Aaa	1002
111	1000

scanf("%d%s%d", &sp->regno, sp->name , &sp->totalmarks)

printf("%d %s %d", sp->regno, sp->name, sp->totalmarks)

sp->regno = (*sp).regno

Struct student S[10]; S[i].regno, S[i].name S[i].totalmarks

Int X[10], *p;

P = X ?; // &X[0]

Struct student *sp;

sp = S // S = &S[0]

67	
www	
333	1028
89	
xxx	
222	1014
78	
aaa	
111	1000

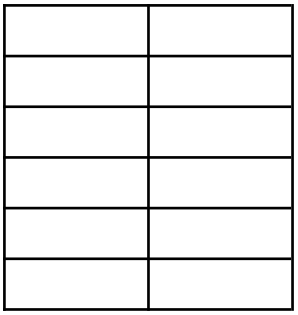
```
For(i=0; i<3; i++)
  // Scanf("%d%s%s", &S[i].regno, S[i].name, &S[i].totalmarks);
   Scanf("%d%s%s", &(sp+i)->regno, (sp+i)->name, &(sp+i)->totalmarks);
```

Sp+1 = 1000+1 = 1014

Sp = sp+2= 1000 + 2(14) = 1028

```
For(sp= S ; sp<=sp+2 ; sp++)
 Scanf("%d%s%s", &sp->regno, sp->name, &sp->totalmarks);
```

```
For(sp= S ; sp<=sp+2 ; sp++)
Printf("%d %s  %d\n",  sp->regno, sp->name, sp->totalmarks);
```

Void pointers -Generic pointer)

Void * ptr;(can be used for pointing to any type of variable)

Ex: int x; float y ; char c struct student S;

Ptr = &x;

Ptr = &y;

Ptr = &c;

Ptr = &S;

Note: While accessing the vaue, we have to type cast to the required type.

Ex: void *P;

Int x = 10; float y = 20.5; char c = 'A';

P = &x;

Printf("%d", *(int*)P) ;

P = &y;

Printf("%f", *(float*)P);

Scanf("%f", (float*)P); // reading y

Troubles with Pointers:

Assuming values to uninitialized pointers

Ex: int *p, x= 10;

*p = x

Assigning value to a pointer variable

Ex: int *p, x= 10;

p =x

Not dereferencing a pointer when required:

Ex: int *p, x= 10;

p = &x; printf("%d", p);

Comparing pointers that point to different objects:

Int x, y, *p1, *p2;

P1 = &x; p2= &y;

If(P1>P2)