INTRODUCTION POINTERS

- A pointer is a derived data type in c.
- Pointers contains memory addresses as their values.
- A **pointer** is a variable whose value is the address of another variable, i.e., direct address of the memory location.
- Like any variable or constant, you must declare a pointer before using it to store any variable address.
- Pointers can be used to access and manipulate data stored in the memory.

Advantages

(i) Pointers make the programs simple and reduce their length.

(ii) Pointers are helpful in allocation and de- allocation of memory during the execution of the program.

Thus, pointers are the instruments dynamic memory management.

(iii) Pointers enhance the execution speed of a program.

(iv) Pointers are helpful in traversing through arrays and character strings. The strings are also arrays of characters terminated by the null character ('\0').

(v) Pointers also act as references to different types of objects such as variables, arrays, functions, structures, etc. In C, we use pointer as a reference.

(vi) Storage of strings through pointers savesmemory space.

(vii)Pointers may be used to pass on arrays, strings, functions, and variables asarguments of a function.

(viii) Passing on arrays by pointers saves lot of memory because we are passing on onlythe address of array instead of all the elements of an array, which would mean passing on copies of all the elements and thus taking lot of memory space.

(ix) Pointers are used to construct different data structures such as linked lists, queues, stacks, etc.

ACCESSING THE ADDRESS VARIABLE

- The operator & immediately preceding a variable returns the address of the variable associated with it.

Example

p=&quantity;

- Would assign 5000 (the location of quantity) to the variable p.
- The & operator is a address operator.
- The & operator can be used only with asimple variable or an array element.

&125 □ pointing at constants

int x[10]; illegal

&x □ pointing at array names &(x+y) □

pointing at expressions

- If x is an array then expression such as &x[0] is valid.

```
#include <stdio.h>
#include <conio.h>void
main()
{
int x=125;
float p=20.345;char
a='a'; clrscr();
printf("%d is stored at addr %u\n",x,&x); printf("%f is stored at addr %u\n",p,&p); printf("%c is stored at addr %u\n",a,&a); getch();
}
```

DECLARING POINTER VARIABLES

Syntax

data_type *pt_name;

1. The * tells that the variable pt_name is aname of the pointer variable.
2. Pt_name needs a memory location.
3. Pt_name points to a variable of type data_type.

Example

int *p;

- Declares the variable p as a pointer variable that points to an integer data type.
- The declarations cause the compiler to alocate memory locations for the pointer variable p.

INITIALIZATION OF POINTER VARIABLES

- The process of assigning the address of a variable to a pointer variable is known as initialization.
- All uninitialized pointers will have some unknown values that will be interpreted as memory addresses.
- They may not be valid addresses or they may point to some values that are wrong.
- Once a pointer variable has been declared we can use the assignment operator to initialize the variable.

Example

1. *int q;*
   **p;*
   p=&q;

2. *int q;*
   *int *p=&q*

3. *int x,*p=&xint*

Illegal statement *int *p=&x, x;*

- We can also define a pointer variable with an initial value to NULL or 0.

*int *p=null: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;
……..
```

- With the exception of NULL and 0, no other constant value can be assigned to a pointer variable.

ACCESSING A VARIABLE THROUGH ITS POINTERS

- **We can access the value of another variable using the pointer variable.**

Steps:

- Declare a normal variable, assign the value.
- Declare a pointer variable with the same type as the normal variable.
- Initialize the pointer variable with the address of normal variable.
- Access the value of the variable by using asterisk (*****) - it is known as **dereference operator (indirection operators).**

```
#include <stdio.h>
 int main(void)
 {
 //normal variable
 int num = 100;
//pointer variable
 int *ptr;
 //pointer initializationptr =
 &num;
//pritning the value
printf("value of num = %d\n", *ptr);
 return 0;
}
```

EXAMPLE

```
#include <stdio.h>
void main()
{
int x,y; int
*ptr;x=10;
ptr=&x;
y=*ptr;
printf("Value of x is %d\n",x);
```

```
printf("%d is stored at address %u\n",x,&x);
printf("%d is stored at address %u\n",*&x, &x);printf("%d is stored at address %u\n",*ptr,ptr); printf("%d is stored at address %u\n",ptr,&ptr); printf("%d is stored at address %u\n",y,&y);
*ptr=100;
printf("\nNew value of x =%d\n",x);
}
```

Output

Value of x is 10

10 is stored at address 2996846848

10 is stored at address 2996846848

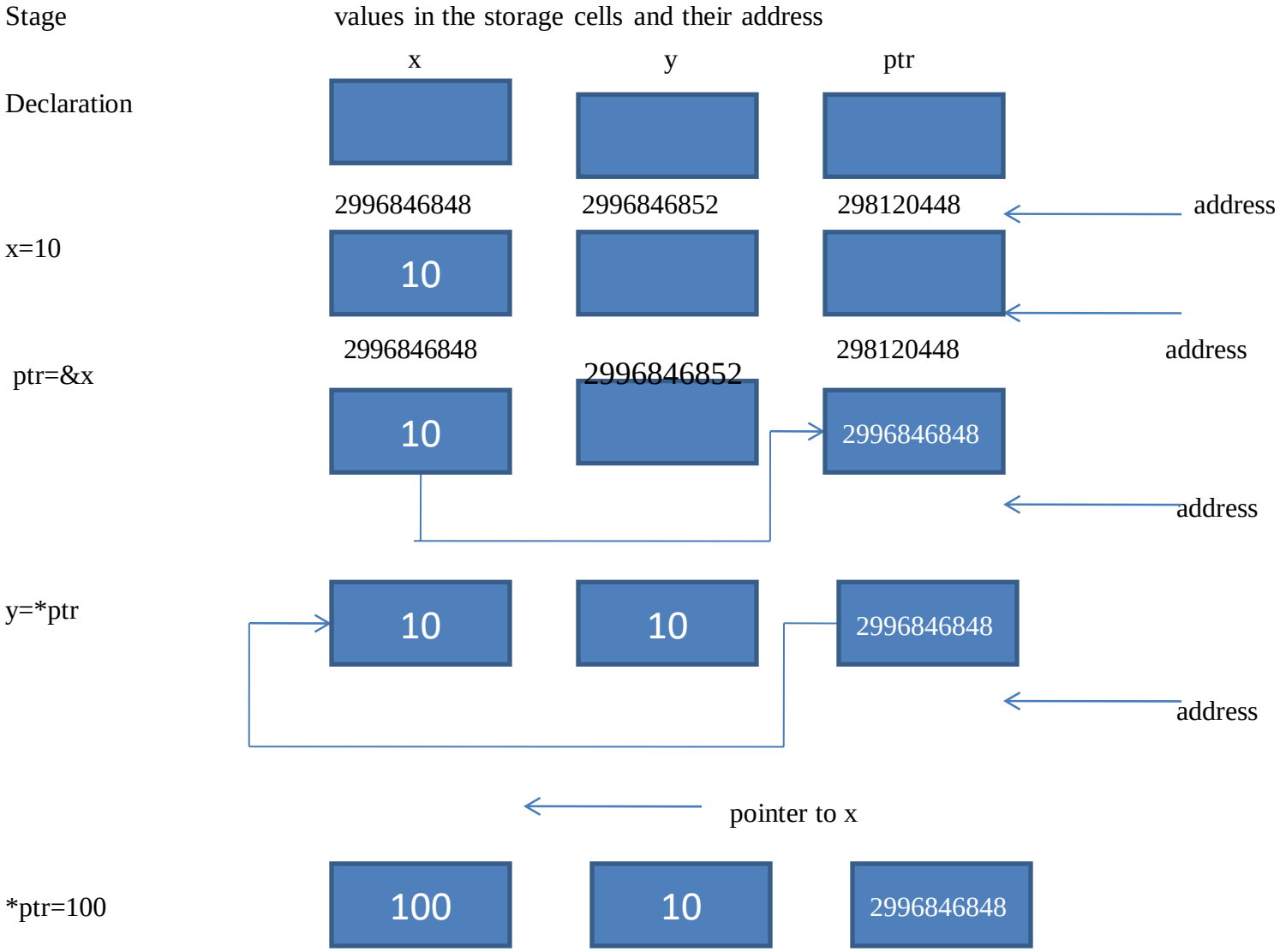
10 is stored at address 2996846848

298120448 is stored at address 2996846856

10 is stored at address 2996846852

New value of x =100

ILLUSTRATION OF POINTER EXPRESSION



CHAIN OF POINTER

- Pointer to point to another pointer, thus creating a chain of pointer.

p2 p1 variable

- The pointer variable p2 contains the address of the pointer variable p1, which points to the location that contains the desired value.
- This is known as multiple indirections.
- A variable that is pointer to a pointer must be declared using additional indirection operator symbol in front of the name.

  int **p2;

- The declaration tells the compiler that p2 is a pointer to a pointer of int type.

- The pointer p2 is not a pointer to an integer, but rather a pointer to an integer pointer.
- We can access the target value indirectlypointed to by pointer to a pointer by applying the indirection operator twice.

```
#include <stdio.h>void
main()
{
int x, *p1,**p2;x=100;
p1=&x; p2=&p1;
printf("pointer to pointer value %d",**p2);
}
```

Output

pointer to pointer value 100

POINTER EXPRESSIONS

- Pointer variables can be used in expressions

Example

- If p1 and p2 are properly declared andinitialized pointers then the following statements are valid.

*y= *p1 * *p2; y=(*p1) * (*p2)sum= sum + *p1;*

z=5 - *p1/ *p2 (5* (-(*p1)))/(*p2);*

- There is blank space between / and *p2

**p2= *p2 + 10;p1+4;*
p2-2;
p1-p2;
p1++;
-p2;
*sum += *p2;*

- In addition to arithmetic operations , the pointer can also be compared using the relational operators.

p1>p2 p1==p2
p1 != p2

- We may not use pointers in division or multiplications.

*p1/p2 p1 **
p2p1/3

Example

```
#include <stdio.h>
int main()
{
  int a, b,*p1, *p2,x,y,z;a=10;
  b= 5;
  p1=&a;
  p2=&b;
  x= *p1 * *p2;y=
  *p1 + *p2;
  printf("Address of a = %u\n",a);
  printf("Address of b = %u\n",b);
  printf("a= %d\tb=%d\n",a,b); printf("x=
  %d\ty=%d\n",x,y);
  *p2= *p2 +5;
  *p1= *p1-5;
   z= *p1 * *p2 -7;
   printf("a= %d\tb=%d\n",a,b);
printf("*p1 = %d\n",*p1);
   printf("*p2 = %d\n",*p2);
   printf("z= %d\n",z); return 0;
}
```

Output

```
Address of a = 71870892 Address
of b = 71870896 a= 10 b=5
x= 50      y=15
a= 5       b=10
*p1 = 5
*p2 = 10
z= 43
```

POINTER INCREMENT & SCALE FACTOR

p1++;

- The pointer p1 to point to the next value of its type.
- If p1 is an integer pointer with an initial value, say 4020, then the operation p1++, the value of p1 will be 4022.
- Ie, the value increased by the length of the data type that it points to.

char	1 byte
int	2 bytes
float	4 bytes
long int	4 bytes
double	8 bytes

POINTERS AND ARRAYS

- The address of &x[0] and x is the same. It's because the variable name x points to the first element of the array.
- &x[0] is equivalent to x. And, x[0] is equivalent to *x.
- Similarly, &x[1] is equivalent to x+1 and x[1] is equivalent to *(x+1).
- &x[2] is equivalent to x+2 and x[2] is equivalentto *(x+2).
- Basically, &x[i] is equivalent to x+i and x[i] is equivalent to *(x+i).

Example 1: Pointers and Arrays

```
#include <stdio.h>int
 main()
{
 int i, x[20], sum = 0,n;
printf("Enter the value of n: ");
scanf("%d",&n);
printf("Enter number one by one\n");
for(i = 0; i < n; ++i)
{
/* Equivalent to  scanf("%d", &x[i]); */
scanf("%d", x+i);
 // Equivalent to sum += x[i]
 sum += *(x+i);
 }
printf("Sum = %d", sum);
return 0;
}
```

Output

```
Enter the value of n: 5
Enter number one by one5
10
15
20
25
Sum = 75
```

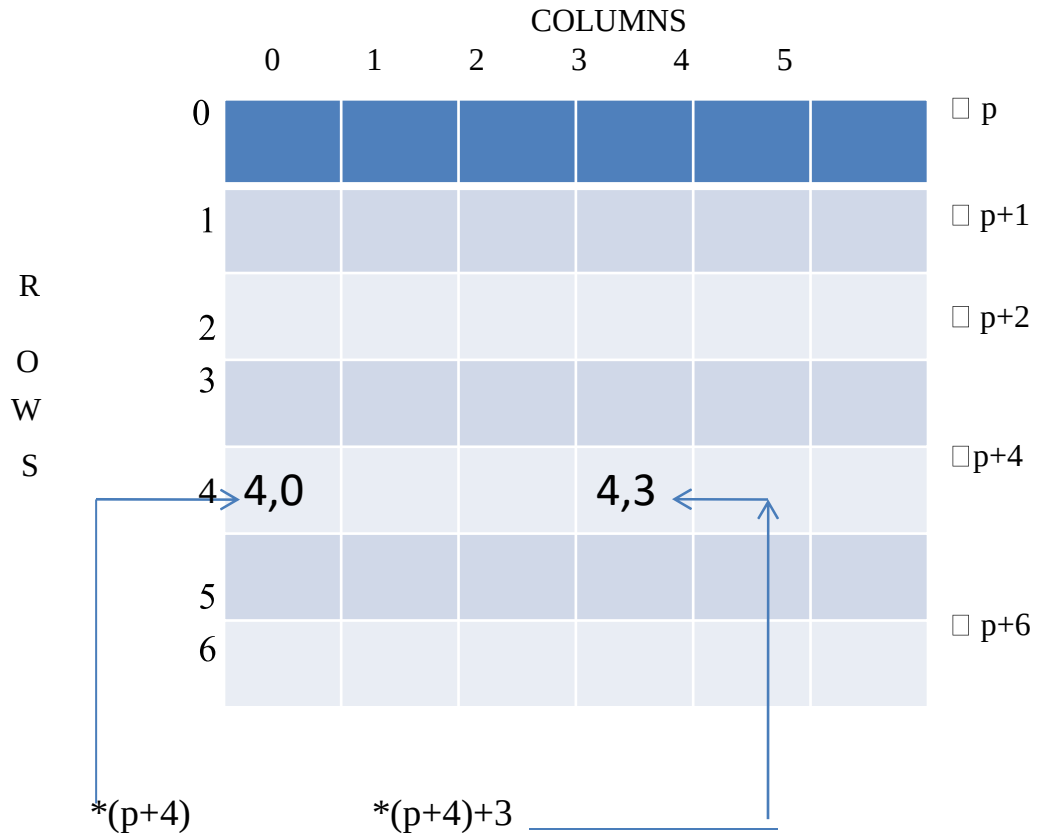
Example :2

```
#include <stdio.h>
int main()
 {
int *p,sum,i;
 int n,x[10];
printf ("Enter the value of n\n");
 scanf("%d",&n);
printf("Enter the array elements one by one\n");for
(i=0;i<n;i++)
   scanf("%d",&x[i]);
 p=x;
printf("Elements\t Value\t Address\n");for
(i=0;i<n;i++)
{
 printf("x[%d] %d %u\n",i,*p,p);sum
 +=*p;
 p++;
}
printf("\n Sum = %d",sum);
printf("\n address of first element (&x[0]) =
       %u",&x[0]);
printf("\n p = %u",p);
return 0;
}
```

output

```
Enter the value of n

5
Enter the array elements one by one

1
2
3
4
5
 Elements      Value  Address

x[0]
        1             2316341728

x[1]        2             2316341732

x[2]        3             2316341736

x[3]        4             2316341740

x[4]        5             2316341744

 Sum = 16
 address of first element (&x[0]) = 2316341728

 p = 2316341748
```

- Pointers can be used to manipulate two-dimensional arrays also.
- An two-dimensional array can be represented by the pointer expression as follows
- *(*(a+i)+j) or *(*(p+1)+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 ithrow

((p+i)+j) valu stored in the ith row and jth columns.

Example

```
#include<stdio.h>int
main()
{
   int arr[3][4] = {{11,22,33,44},
      {55,66,77,88},{11,66,77,44}};
   int i, j;
for(i = 0; i < 3; i++)
  {
printf("Address of %d th array %u \n",i , *(arr + i));for(j =
 0; j < 4; j++)
    {
 printf("arr[%d][%d]=%d\n", i, j, *( *(arr + i) + j) );
      }
printf("\n\n");
  }
  // signal to operating system program ran finereturn 0;
}
```

Output

```
arr[2][3]=44Address of 0 th array 2692284448
arr[0][0]=11
arr[0][1]=22
arr[0][2]=33
arr[0][3]=44
Address of 1 th array 2692284464
arr[1][0]=55
arr[1][1]=66
arr[1][3]=88
Address of 2 th array 2692284480arr[2][0]=11
arr[2][1]=66
arr[2][2]=77
arr[2][3]=44
```

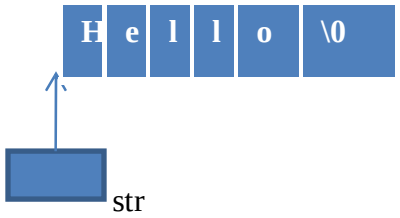
POINTERS AND CHARACTER STRINGS

- C supports an alternate method to create strings ***using pointer variables of type char.***

Example

char *str= “Hello”;

- This creates a string for the literal and then stores its address in the pointer variable str.
- The pointer str now points to the first character of the string “Hello” as



We can also use runtime assignment for giving values to a string pointer.

char *str;
str= “hello”;

```
#include      <stdio.h>
#include  <string.h> int
main ()
 {
 char name[25];
char *ptr;
 strcpy(name,"gaccbe");
 ptr=name;
while(*ptr !='\0')
    {
       printf("\n %c is stored at address %u",*ptr,ptr);ptr++;
     }
  return 0;
}
```

Output

g is stored at address 3432464000

a is stored at address 3432464001

c is stored at address 3432464002

c is stored at address 3432464003

b is stored at address 3432464004

e is stored at address 3432464005

ARRAY OF POINTERS

Example

char name[4][25];

- The name is a table containing four names, each with maximum of 25 characters.
- The total storage requirements is 75 bytes.
- The individual strings will of equal lengths.

Example

```
char *names[4] = {
                    "Anu",
                    "Banu", "Chandru",
                    "Deepak"
                };
```

- Declares name to be an array of four pointers to characters, each pointer pointing to a particular name.

```
#include <stdio.h> const
 int MAX = 4;int main ()
 {
char *names[] = { "Anu", "Banu", "Chandru",
       "Deepak" };
 int i = 0;
 for ( i = 0; i < MAX; i++)
  {
printf("Value of names[%d] = %s\n", i, names[i] );
}
 return 0;
}
```

Output

Value of names[0] = Anu Value of
names[1] = Banu Value of names[2]
= ChandruValue of names[3]
=Deepak

POINTERS AS FUNCTION ARGUMENTS

- Pointer as a function parameter is used to hold addresses of arguments passed duringfunction call.
- This is also known as **call by reference**.
- When a function is called by reference any change made to the reference variable will effect the original variable.

EXAMPLE

```
#include <stdio.h>
void exchange(int *a, int *b);int
 main()
 {
 int m = 10, n = 20; printf("m
= %d\n", m); printf("n =
%d\n\n", n);swap(&m, &n);
printf("After Swapping:\n\n");
printf("m = %d\n", m);
 printf("n = %d", n);
 return 0;
 }
void exchange (int *a, int *b)
{
 int temp;
temp = *a;
 *a = *b;
 *b = temp;
 }
```

Output

m = 10

n = 20

After Swapping:m =
20

n = 10

FUNCTIONS RETURNING POINTERS

- A function can return a single value byits name or return multiple values through pointer parameters.
- A function can also return a pointer to the calling function.
- Local variables of function doesn't live outside the function.
- They have scope only inside thefunction.
- Hence if you return a pointer connectedto a local variable, that pointer will be pointing to nothing when the function ends.

```
#include <stdio.h> int*
larger(int*, int*);void
main()
{
 int a = 10;int
b = 20;int *p;
p = larger(&a, &b); printf("%d
is larger",*p);
}
 int* larger(int *x, int *y)
 {
 if(*x > *y)
        return x;
Else
         return y;
 }
```

Output

20 is larger

POINTERS TO FUNCTIONS

- It is possible to declare a pointer pointing toa function which can then be used as an argument in another function.
- A pointer to a function is declared asfollows,

  *type (*pointer-name)(parameter)*

Example

int (*sum)(); □legal declaration of pointer to function

int *sum(); □This is not a declaration ofpointer to function.

- A function pointer can point to a specific function when it is assigned the name of that function.

  *int sum(int, int);int (*s)(int, int); s = sum;*

- s is a pointer to a function sum.
- sum can be called using function pointer s along with providing the required argument values.

  s (10, 20);

Example

```
#include <stdio.h> int
sum(int x, int y)
 {
return x+y;
 }
int main( )
 {
 int (*fp)(int, int);fp =
sum;
 int s = fp(10, 15);
printf("Sum is %d", s);return
0;
 }
```

Output25

POINTERS AND STRUCTURES

- We know that the name of an array stands for the address of its zero-th element.
- Also true for the names of arrays ofstructure variables.

Example

```
struct    inventory
 {
      int    no;
      char name[30];float
       price;
}   product[5], *ptr ;
```

- The name product represents the addressof the zero-th element of the structure array.
- ptr is a pointer to data objects of the typestruct inventory.
- The assignment

  ptr = product ;

  will assign the address of product [0] to ptr.

- Its member can be access

  ptr –>name ;ptr –> no ; ptr –> price;

  The symbol "–>" is called the arrowoperator or member selection operator.
- When the pointer ptr is incremented by one (ptr++) :The value of ptr is actually increased by sizeof(inventory).
- It is made to point to the next record.
- We can also use the notation

  *(*ptr).no;*
- When using structure pointers, we shouldtake care of operator precedence.
- Member operator "." has higher precedence than "*".
- ptr –> no and (*ptr).no mean the same thing.
- ptr.no will lead to error.

- The operator "–>" enjoys the highest priority among operators
- ++ptr –> no will increment roll,not ptr.
- (++ptr) –> no will do the intendedthing.

Example

```
void main ()
{
struct book
{
char name[25]; char
author[25];int edn;
};
struct book b1 = { "Programming in C","E
    Balagurusamy", 2 } ;
 struct book *ptr ;ptr =
 &b1 ;
printf ( "\n%s %s edition %d ", b1.name,
    b1.author, b1.edn ) ;
printf ( "\n%s %s edition %d", ptr-
    >name,  ptr->author, ptr->edn ) ;
}
```

Output

Programming in C E Balagurusamy edition 2
Programming in C E Balagurusamy edition 2

Types of Pointers in C

There are various types of pointer in C, to put a number on it, we have 8 different types of pointers in C. They are as follows

1) Void pointer	**2) Null pointer**	**3) Wild pointer**	**4) Dangling pointer**
5) Complex pointer	**6) Huge pointer**	**7) Near pointer**	**8) Far pointer**

1) Void pointer

A pointer is said to be void when a pointer has no associated data type with it. In other words, it can point to any data type and also, it can be typecasted to any type.

Since a void pointer does not have any standard type(data type) it is also referred to as a generic pointer sometimes.

A pointer can be declared as a void pointer by simply using the keyword void.

Declaration: **void** *ptr = NULL;

Example:

```
int main()

{

int num = 5;

void *ptr = &num;

printf("%d", *ptr);

return 0;

}
```

Use of Void pointer

The built functions like malloc and calloc functions (which are used for allocation of memory) returns a void pointer. Due to this reason, they can allocate a memory block for any type of data.

2) Null pointer

Another type of pointer under this classification is Null pointer. It is a special type of pointer that does not point to any memory location. In other words, If we assign a NULL value to a pointer, then the pointer is considered a Null pointer.

Note: In case of null pointer, it always contains 0 or NULL as its value.

Example:

```
#include<stdio.h>

int main()

{

int *ptr = NULL;

return 0;

}
```

Use of Null pointer

1. Used when to initialize a pointer when that pointer is not assigned any memory address for that instance of time.
2. It is useful for handling errors when using in-built functions like malloc function.

3) Wild pointer

A Wild pointer is a type of pointer which is used to point to some arbitrary memory location, that's why it is referred to as Uninitialized pointers.The wild pointer is uninitialized and due to this, they are not very efficient. The inefficiency occurs because they may point to some unknown memory location which may cause a program to perform unintended actions or may even lead to program crash..

Example:

```
int main()

{

int *ptr;  //This is a wild pointer

*ptr = 5;

return 0;

}
```

4) Dangling pointer

A dangling pointer is another type of pointer and it is a pointer which points to some non-existing memory location.

This situation arises when a pointer is pointing at the memory address of a variable but after some time that variable is deleted from that memory location while the pointer is still pointing to it.

Example:

```
int main()

{

int *ptr = (int *)malloc (sizeOf(int));

…

…

…

free(ptr);  //memory is released

return 0;

}
```

In the above code, when the free() function is executed, the memory assigned to ptr is released but the pointer is still pointing to the deallocated memory and hence it is called a Dangling pointer.

6) Huge pointer

A huge pointer can access memory outside the current segment and is typically of size 32 bit.

A Huge pointer is not fixed and hence the part within which a huge pointer is located can be changed .

7) Near pointer

A Near pointer is a pointer that is utilized to bit-address up to 16 bits within a given section of that computer memory which is 16 bit enabled.

Example:

```
#include<stdio.h>

int main()

{

   int x = 25;
```

```
  int near* ptr;
  ptr= &x;
  printf("%d",sizeof ptr);
  return 0;
}
```

8) Far pointer

A far pointer is typically 32 bit which can access memory outside that current segment. It is similar to a huge pointer but a far pointer is fixed and hence that part of that sector within which they are located cannot be changed.

Example:

```
#include<stdio.h>
int main()
{
  int x= 10;
  int far *ptr;
  ptr=&x;
  print("%d", sizeof ptr);
  return 0;
}
```