void pointer :

```
int a = 10;
char b = 'x';
void *p = &a; // void pointer holds address of int 'a'
printf(“%d”, *(int *)p);
*p=30;
printf(“%d”, *(int *)p);
p = &b; // void pointer holds address of char 'b'
printf(“%d”,*(char *)p);
*p=’y’;
printf(“%d”, *(char *)p);
```

Null pointer:

```
int main()
{
 int *i, *j;
 int *ii = NULL, *jj = NULL;
 if(i == j)
 {
  printf("This might get printed if both i and j are same by chance.");
 }
 if(ii == jj)
```

```
 {
  printf("This is always printed coz ii and jj are same.");
 }
 return 0;
}
```

o/p:

This is always printed coz ii and jj are same.

Constant pointers:

Pointer to variable:

```
int main(void)
{
   int i = 10;
   int j = 20;
   int *ptr = &i;
   /* pointer to integer */
   printf("*ptr: %d\n", *ptr);

   /* pointer is pointing to another variable */
   ptr = &j;
   printf("*ptr: %d\n", *ptr);

   /* we can change value stored by pointer */
   *ptr = 100;
   printf("*ptr: %d\n", *ptr);

   return 0;
}
```

Output:

```
*ptr: 10
*ptr: 20
*ptr: 100
```

Pointer to constant.

const int *ptr;

or

int const *ptr;

```
#include <stdio.h>
int main(void)
{
        int i = 10;
        int j = 20;
        /* ptr is pointer to constant */
        const int *ptr = &i;

        printf("ptr: %d\n", *ptr);
        /* error: object pointed cannot be modified
        using the pointer ptr */
        *ptr = 100;
```

```
        ptr = &j;               /* valid */
        printf("ptr: %d\n", *ptr);

        return 0;
}
```

Constant pointer to variable:

```
int *const ptr;

#include <stdio.h>

int main(void)
{
int i = 10;
int j = 20;
/* constant pointer to integer */
int *const ptr = &i;

printf("ptr: %d\n", *ptr);

*ptr = 100; /* valid */
printf("ptr: %d\n", *ptr);
```

```
ptr = &j;       /* error */

return 0;

}
```

Difference between const char *p, char * const p and const char * const p:

const char *ptr : This is a pointer to a constant character. **You cannot change the value pointed by ptr, but you can change the pointer itself**.

```
// C program to illustrate
// char const *p
#include<stdio.h>
#include<stdlib.h>

int main()
{
   char a ='A', b ='B';
   const char *ptr = &a;

   //*ptr = b; illegal statement (assignment of read-only location *ptr)

   // ptr can be changed
   printf( "value pointed to by ptr: %c\n", *ptr);
   ptr = &b;
   printf( "value pointed to by ptr: %c\n", *ptr);
}
```

NOTE: There is no difference between **const char *p and char const *p** as both are pointer to a const char and position of '*'(asterik) is also same.

char *const ptr : This is a constant pointer to non-constant character. **You cannot change the pointer p, but can change the value pointed by ptr.**

```
int main()
{
```

```
    char a ='A', b ='B';
    char *const ptr = &a;
    printf( "Value pointed to by ptr: %c\n", *ptr);
    printf( "Address ptr is pointing to: %d\n\n", ptr);

    //ptr = &b; illegal statement (assignment of read-only variable ptr)

    // changing the value at the address ptr is pointing to
    *ptr = b;
    printf( "Value pointed to by ptr: %c\n", *ptr);
    printf( "Address ptr is pointing to: %d\n", ptr);
}
```

const char * const ptr : This is a constant pointer to constant character. **You can neither change the value pointed by ptr nor the pointer ptr.**

```
int main()
{
    char a ='A', b ='B';
    const char *const ptr = &a;

    printf( "Value pointed to by ptr: %c\n", *ptr);
    printf( "Address ptr is pointing to: %d\n\n", ptr);

    // ptr = &b; illegal statement (assignment of read-only variable ptr)
    // *ptr = b; illegal statement (assignment of read-only location *ptr)

}
```

dangling pointers:

Variable goes out of scope

```
void main()
{
   int *ptr;
   .....
   .....
   {
      int ch;
```

```
    ptr = &ch;
  }
  .....
  Printf(“%c”,*ptr);
  // Here ptr is dangling pointer
}
```

Check:

```
printf("%d",NULL);

printf("%lu",sizeof(NULL));

 printf("%d",*ptr);
printf("%lu",sizeof(void));

 printf("%lu",sizeof(void *));
```