||Pointer||

1) what is pointer ? - Appl of pointer

Pointer is a variable which refers address .

```
int k=90 ;
int *p=&k ;              // here 'p' is a pointer
Printf("%d", p);         // p gives address of k
Printf ("%d", *p);       // p gives ~~address~~ values of k
```

* pointer in GCC complier to access 4 GB memory.

*Interchange two number

```
# include "stdio.h"
void swap (int *, int *);
main()
{

     int a=5, b=8 ;
              Printf ("%d\n", a,b);
     Swap (&a, &b);
              Printf ("%d\n", a,b);
}
void swap (int *p, int *q)
{

     int c = *p;
      *p = *q;
      *q = c;
}
```

```
// p gives address of a
//*p  "    values  "  a
//*q  "     "      "  b
//q   "    address "  b
```

```
* # include "stdio.h"
main()
{
      int k=90;
      int *p= &k;
              Printf(" %d ", k); //90
      k=100;
             Printf(" %d ",k); //100
      *p=130;
             Printf(" %d", k); //130
```

```
* # include "stdio.h"
 void increment (int, int*);
 main()
 {
      int a=3, b=3;
             Printf(" %d\n", a, b); // 3   3
      increment (a, &b);
             Printf(" %d %d\n", a, b); //3  5
 }
 void increment (int p, int *q)
 {
      p=p+2;
      *q = *q+2;
 }

 // *q means value of b
```

* Every func allocates memory in stack segment.

```
* #include"stdio.h"
main()
{
        malloc (20);
        calloc (2,10);
        calloc (10,2);
        calloc (4,5);
        calloc (5,4);
}
```

```
command → valgrind --tool=memcheck ./a.out
```

2) Diff bet^n malloc & calloc ?

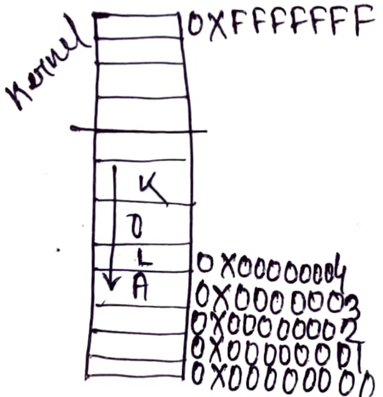
malloc memory allocation is 1D array equivalent

calloc " " " 2D " "

* If a pointer refers starting address or base address in memory is call null pointer.

* Doing read or write oper^n using pointer the pointer is called "dereference".

void *p → generic pointer



```
main()
{
    char *p= (char *)0X00000003;
    *(P+0) = 'A';     }
    *(P+1) = 'L';     }  write
    *(P+2) = 'O';     }
    *(P+3) = 'K';     }

    Printf("%c", *(P+3));   }
    Printf ("%c", *(P+2));  }  read
    Printf ("%c", *(P+1));  }
    Printf ("%c", *(P+0));  }
```

* If a pointer referes unauthorized address in memory called 'dangling' pointer.

* Deforence to a dangling pointer becomes segmetation fault.

* If a pointer doesn't referes any address of memory that Pointer is called wild pointer. It also segmentⁿ fault.
runtime error

* Pointer size is always 4 byte irrespective of any datatypes

```
main()
{
    char *p;
    short int *q;
    int *r;
    float *s;
    double *t;
    Printf("%d %d %d %d", sizeof(P), sizeof(q), sizeof(r),
        sizeof(s), sizeof(t);
```

Pointer Arithmatic :-

(Address+1) is equal to next address a/c to datatypes.
(Address-1) " " " Previous " " " "

```
main()
{
    double *p = (double *) 500;
    Printf ("%d", p+1); // 508;
    Printf ("%d", p-1); // 492
}
```

There are the pointer arithmatic allowed in c;

address + number = address;
address - number = address;
address - address = number

```
main()
{
    int *p = (int *) 500;
    int *q = (int *) 520;
    Printf ("%d", q-p); //5
```

Chain of pointer :-

```
main()
{
        int k=90 ;
        int *p = &k ;
        int **q = &p ;
        int **r = &q ;
}
```

* A pointer having 'n' dir^n can refer address of a variable having 'n-1' indirec^n.

Diff bet^n near & far pointer :-

→ If a pointer referes address in same segments memory called near pointer.

→ If a pointer referes address in other " " called far pointer.

What is memory leak ?

At any moment a pointer lost reference of a memory block in heap segment called memory leak.

Question:-

1) what is pointer ? Appl of pointer ?
2) what is null pointer ? Appl
3) what is generic pointer ? Appl
4) Diff bet^n wild & dangling pointer ?
5) " " near & far pointer ?
6) " " pointer & reference ?

8) what are the concept is called 'deference' ?

9) what is chain of pointer in c ?

10) Diff betn malloc () & calloc ()

11) " " static & dynamic memorey allocation ?

12) what is memory leak ?

```
#include"stdio.h"
main()
{
    int K=90;
    int *p=&K;
```

→ address of

Indirn

Pointer

→ Pointer is a varieable which refers address.

→ Address is a location of memorey, which is 32 bite or 4 byte in GCC.

→ There are 3 types of memorey (i) logical (ii) virtual (iii) physical

1 TB = 1024 GB
1 GB = 1024 MB
1 MB = 1024 KB
1 KB = 1024 byte
1 byte = 8 bits

0 - decimal
00 - octal
0X0 - hexadecimal

→ Bus is a set of wires connected from one device to another device is called 'Bus'.

Intel

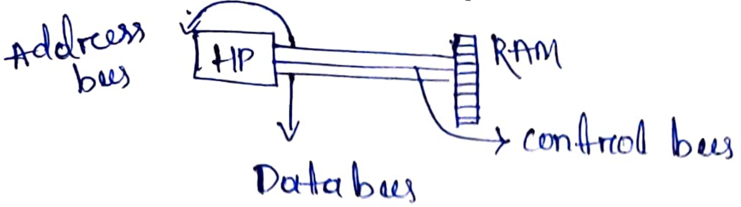
4004 - 4 bits
8085 - 8 bits
8086 - 16 bits
80386 - 32 bits

} list of intel micro processor

i3
i5
i7

} 64 bits

→ 3 dedicated bus are connected microprocessor & RAM.

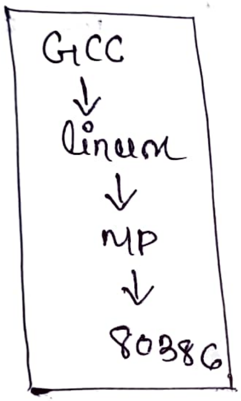


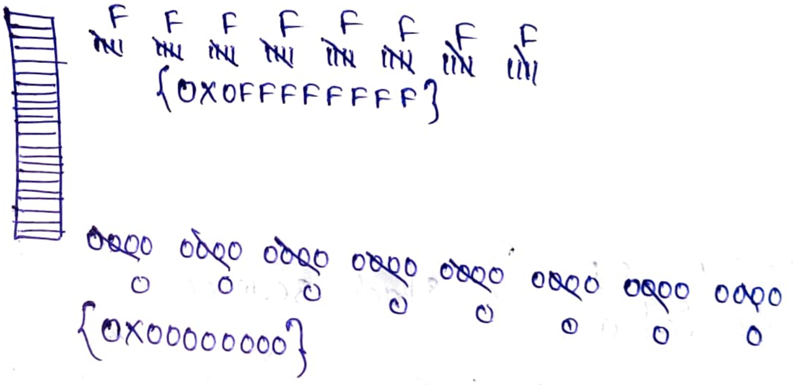
GCC - Gnu's Compailer Collection

→ Pointer in GCC compiler permits to access 4 GB memory.

$\boxed{2^{32} = 4GB}$

→ The address is generated by the microprocessor is known as logical address.





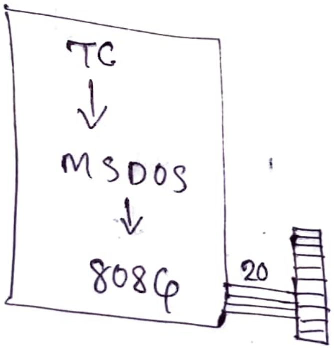
→ The running program of a system is known as process.

→ Program doesnot allocate memory but a process allocate memory.

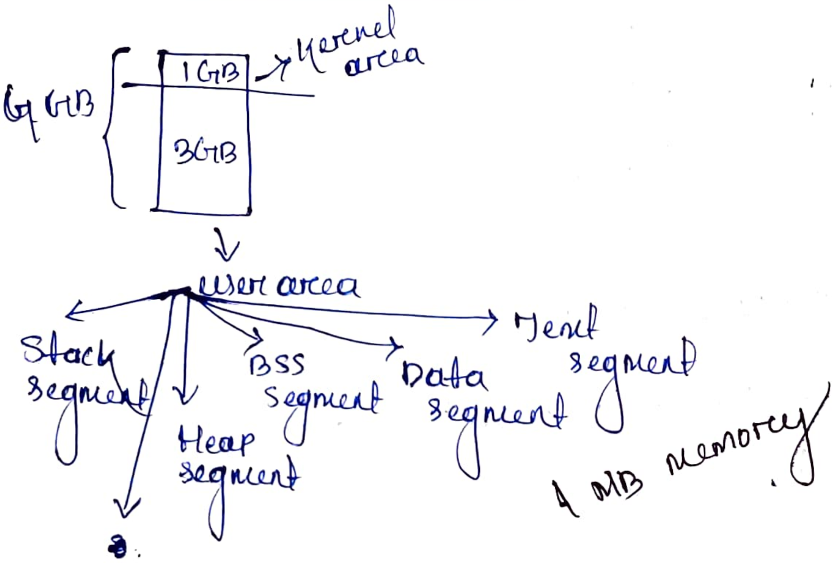
→ A program is an executable file which is store in a disk.

→ Every process allocates virtual memory.

→ Turboc compiler is permits to access 1 MB memory.



$2^{20} = 1MB$



→ All func^n allocates memory in stack segment.

→ Malloc(), calloc(), realloc() are allocates memory in heap segment

→ Pointer in GCC compiler permits to access 4GB memory.

Ex :-

```
main()
{
 auto int i;      }
 auto int a;      } stack segment
 static int c;    } BSS segment (uninitialized)
 static int d=0;  } data segment
```

→ No variable func^n allocates memory in text segment

→ Command to know the header file?
man3

→ man3 display the manual page of library func^n

→ malloc() & calloc() are for dynamic memory alloc^n.

what is dynamic memory alloc^n ?

→ Heap segment is suitable for dynamic memory alloc^n.

→ malloc(20) is equal to writing calloc(2,10);
calloc(10,2);
calloc(5,4);
calloc(4,5);

what is diffⁿ betⁿ malloc & calloc funcⁿ ?

Malloc memorial locⁿ is 1D array equivalent but calloc memorial locⁿ is 2D array equivalent.

Ex :-

```
main()
{
    calloc(3,7);
    calloc(2,10);
    malloc(15);
    malloc(6);
```

/ Here 62 bytes memory is allocated in 4 blocks.

→ Number of bracket represents number of dimensional to an array.

Ex :-

```
main()
{
    char x [];
    char x [][];
    char x [][][];
    char x [][][][];
}
```

Ex :-

```
main()
{
    malloc(20);
        char x[20];
    calloc (2,10)
        char x [2] [10];
}
```

*memory allocⁿ using malloc() & calloc() is always same.

Appl of Generic pointer

Generic pointer is suitable to refer the address allocated by malloc & calloc func^n.

Ex :-

```
main()
{
    void* p = malloc(20);
    void *q = calloc(2,10);
}
```

Ex :-

```
main()
{
    int k=100;
    int *p= &k;
        Printf("%d",k) // 100
        Printf("%d",*p) // 100
```

Ex :-

```
main()
{
    int k=100; // directly
    int *p = &k; // indirectly
        Printf("%d\n",k);
    *p = 200; // indirectly
        Printf("%d\n",k);
    k=300; // directly change
        Printf("%d\n",k);
    *p=5;
        Printf("%d\n",k);
}
```

→ If formal parameter is a pointer type is changing the formal parameter it will change the value of actual parameter.

→ If formal parameter is a value type changing the formal Parameter it doesnot change the value of actual parameter.

Ex:-

```
void increment (int, int*);
main()
{
    int a=5, b=5;
        Printf("%d %d\n", a,b); // 5 5
    increment (a, &b);
        Printf(" %d %d\n", a,b); // 5 25
}
void increment (int p, int *q)
{
    p= p*p;
    *q =(*q) * (*q);
}
```

→ Actual parameter allocates memory but formal parameter doesnot allocates memory.

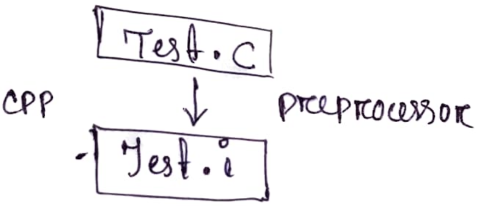
→ CPP reads the source file & generate intermediate source File which is called preprocessing.

Ex:-

```
#define max 10+2
main()
{
    int k = max/2;
    Printf("%d",k);
}
```

Ex:-

```
main()
{
    int k = 10+2/2;
    Printf("%d",k);
}
```



→ Command For, generating intermediate sourcefile

```
cpp test.c -o test.i
```

→ Variables names case sensitive meaning

```
main ()
{
        int a = 90;
        int A = 10;
        Printf("%d", a); // 90
        Printf("%d", A); //10
}
```

→ Variables names & func names are known as identifiers.

→ Identifiers begins with alphabet & underscore char

which is valid & not valid

```
main()
{
      int a1 = 90;
      int 1a = 100;
       int a-1 = 200;
       int a-a = 70;
        int _ = 8;
        int First number = 99;
         int First & second = 100;
}
```

→ Identifier can't be a keyword

```
main ()
{
      int assima = 100;
      int auto = 200;
       int sima = 90;
        int break = 100;
}
```

→ AS many types of variables that much types of constrate in c.

```
main()
{
    int x = 90;
    char y = 'a';
     float z = 4.5;
      double s = 4.5;
       char [] = 'india';
}
```

→ Variable is an object which allocates memory. Variable value can be changed in a program. But constant is a value which can't be changed in a program.

→ All the constants are known as literals.

→ All const, variable, func name operators are called Tokens.

```
main()
{
    int x = 90;
    char y = 'a';
    Printf ("Hello");
    for (i=1; i<=5; i++)
          Printf ('on');
}
```

Pointer Arithmatic :-

(Address+1) is equal to next address a/c to data types.

```
main()
{
    int *p = (int *) 500;
    Printf ("%d", p+1); // 504
    Printf ("%d", p-1); // 496
}
```

These are the pointer arithmatic allowed in c;

address + number = address
address - number = address
address - address = number

Ex :-

```
main()
{
    int *p = (int *) 500;
    int *q = (int *) 520;
    Printf("%d", q,p);
```

Diff appearance of pointer in c :-

```
int *p;         // p is a pointer
int *sum();     // Sum is a funcn return pointer
int(*q)();      // q is a funcn pointer
int *r[5];      // r is array of pointer
int(*t)[];      // t is a pointer to array
```

Ex :-

```
int sum(int, int);
main()
{
    int x;
    x = sum(5,6);
    Printf("%d", x);
}
int sum(int a, int b)
{
    int c = a+b;
    return c;
}
```

Ex :-

```
int *sum(int, int);
main()
{
    int *x;
    x = Sum(5,6);
    Printf("%d", *x);
}
int *sum(int a, int b)
{
    int c = a+b;
    return c;
}
```

→ Func^n name is a pointer which refer the starting address of the func^n.

→ A func^n can be call directly or indirectly.

Ex :-

Func^n directly call

```
main()
{
  int a,b,c;
  Printf("Enter first no");
   Scanf("%d", &a);
   Printf("Enter second no");
    Scanf("%d", &b);
    c=a+b;
     Printf("%d", c);
```

Ex :-

Func^n indirectly call

```
main()
{
  int a,b,c;
  int (*p)(), (*q)();
  P = Printf;
  q = Scanf;
 (*p)("Enter first no");
 (*q) ("%d", &a);
  (*p) ("Enter second no");
   (*q) ("%d", &b);
    c=a+b;
  (*p) ("%d", c);
```

Appl of pointer :-

→ It is suitable for dynamic memory alloc^n.

→ " " " " hardware programming.

→ It is suitable to implement an array.

→ " " " " call a func^n indirectly.

Ex :-

```
main()
{
    int i;
    char str[] = "hellosir";
    Printf("old string %s\n", str);
    for (i=0; i<strlen(str); i++)
    {
        str[i] = str[i]-32;
    }

    Printf("New string = %s", str);
// Printf("%d\n", sizeof(str)); // 9
  Printf("%d\n", strlen(str)); // 8
```

Toggle case :-

```
main()
{
    int i;
    char str[] = "HEllosir";
    Printf("oldstring is %s\n", str);
    for (i=0; i<strlen(str); i++)
    {
      if(str[i]=65 && str[i] <=90)
          str[i] = str[i]+32;
      else
          str[i] = str[i]-32;
    }
    Printf("New string = %s", str);
}
```

Diff bet^n static & dynamic memory alloc^n ?

Every program allocate memory during load time & run time. But how much memory will be allocated if the size is decided during run time called "dynamic memory" alloc^n. If the size is decided during compile time called "static memory" alloc^n.

→ Library is of two types (i) static
(ii) dynamic

Library	linux	windows
static	.a	.lib
dynamic	.so	.dll

.a → archeiving file
.so → share object file
.dll → dynamic linking library
.lib → library

→ func^n can be defined below or above the main func^n.

→ If func^n are defined below or above main func^n that func^n can't be utilized other 'c' program.

→ To overcome this problem library is used.

→ Header file contains only func^n decl^n

```
// ravana.h
void powerof2 (int, int);
int fact (int);
int sumdigits (int);
```

```
// lata.c
int sumdigits (int n)
{
    int s=0;
    while (n>0)
    {
        s=s+n%10;
        n=n/10;
    }
    return s;
}
int fact (int n)
{
    int f=1;
    while (n>0)
    {
        f=f*n;
    f = f*n;
    n--;
  }
  return f;
}
void powerof2 (int i, int j)
{
    while (i<=j)
    { if (i&i-1)==0)
                  printf("%d", i);
      i++;
    }
}

// test.c
 main()
 {
    powerof2 (1,100);
 }
```

To create a static library

① First compile

② then create a library or

```
gcc -c lata.c -w → compile
ar crs mylib.a lata.o → Create static library
 gcc test.c mylib.a → link program with static library
 ./a.out → run
```

Create a dynamic

gcc-c- FPIc data-c-w → compile

gcc-shared-wl -soname, libdemo.so-o
libdemo.so sata.o → Create a dynamic lib

gcc test.c libdemo.so - link program with dynamic lib

export LD_LIBRARY_PATH = 'pwd'.

Set lib path