**DAY - 10  
POINTERS**

Datatype \*ptrName;

Pointer does not belong to any kind of datatype. It is just a special variable that holds the address

Int 🡨\* ptr (read from right to left)

* The ptr special variable is pointing to integer datatype. Pointers should always be initialized.
* The size of the pointers is 4bytes/8bytes. The size of the pointers is same for all data types.
* The value of the pointer is always positive so we use %u

**4types of Pointer**

1. Null pointer
2. Void pointer
3. Wild pointer
4. Dangling pointer (situation)

Void \*ptr 🡪void pointer/generic pointer. It can point to any kind of datatype

Int \*ptr = NULL; 🡪null pointer. It points to nothing

Int \*ptr 🡪wild pointer. As it is not initialized. It can point to any kind of data .

ptr 🡪 gives the address of the variable pointing to

\*ptr 🡪gives the value(dereferencing using \*)

&ptr 🡪gives the address of pointer

Error! : Invalid use of void expression

* When we use void ptr and it is holding to an int datatype and if we want to print the value, it gives this error
* We need to do type casting as the pointer do not know to which bytes of data it should point.

Basically, if we are using int(any)pointer for int(any) datatype no need of type casting

Int a=10;

Void \*ptr=NULL;

Ptr= &a;

Printf (“%d value of a is:”, \*(int \*) ptr); // First \* is for dereferencing int \* is to convert into int

**Arrays and Pointers:** Arrays are static pointers can be dynamic.

int a[3] ={1,2,3};

int \*ptr= NULL;

ptr= &a[0];

printf (“%d”, ptr[0]); // array notation

printf (“%d”, ptr[1]);

printf (“%d”, ptr[2]);

printf (“%d”, \*(ptr+0)); // pointer notation

printf (“%d”, \*(ptr+1));

printf (“%d”, \*(ptr+2));

&\* 🡪nullify each other

2. Op[ ] 🡪 \*op

\*op 🡪op[ ]

We can directly give the name of array to pointer

ptr = &b[0] ==== ptr =b;

**Dynamic Pointers**

To point to an unnamed address

Read man malloc

Malloc 🡪allocates size bytes and returns a pointer to allocated memory the base address

* If size is 0, the malloc return Null or as unique pointer that can later successfully passed to free

void \*malloc(size\_t,size)

We can do free only when malloc is successful. The static memory cannot be freed by us but dynamic memory has to be free else there will be a memory leak in heap

int \*ptr =NULL;

ptr =(int \*)malloc(4); //generally malloc return void data type, so we need to do type casting

ptr= (int \*)malloc(1\*sizeof(int));

\*ptr=1001;

**## Array dynamically**

ptr = (int \*) malloc(4\*sizeof(int));

ptr [0] =10; or \*(ptr+0)=10;

ptr [1] =20; or \*(ptr+1)=20;

ptr [2] = 30; or \*(ptr+2)=30;

ptr [3] = 40; or \*(ptr+3)=40;

perror(“”); 🡪capture the error and prints the error

\*ptr= 101;

Printf (“Address: %u,ptr);

Ptr++;

\*ptr= 102;

Printf (“Address: %u,ptr);

Ptr++;

\*ptr= 103;

Printf (“Address: %u,ptr);

Ptr++;

Ptr--; // to get back to base address we can also use ptr=temp

Ptr--;

For (int i=0; i<4;i++)

{

Printf (“%d”, \*(ptr+i));

Ptr++;

}

ptr++ it points to the next address

we need to store the ptr value in temp

**WAP to search a value in list using dynamic arrays**

**// Online C compiler to run C program online**

#include <stdio.h>

#include<stdlib.h>

int search(int \*, int \*,int);

int main() {

int i,key,res;