DAY --- 10

**Pointers:**

* It will hold the address or it can point to a address.

**Syntax:**

dt \* ptrName

* pointer doesn’t belong to any type of datatype
* int \*ptr ----- ptr is a special variable pointing to int dt.
* Pointers can be dynamic.

**Null Pointer ---** int \*ptr = NULL

**Void pointer** --- generic pointer -- void \*ptr

**Wild pointer**--- without any initialization

Danging Pointer

#include <stdio.h>

int main(){

int a =10;

float b = 20.2;

void \*ptr = NULL;

printf("\nAddress of a=%u and its value %d",&a,a);

printf("\naddrress of b =%u and its value %f",&b,b);

printf("\naddress of ptr=%u and its value =%u",&ptr,ptr);

printf("\nsize of a=%d",sizeof(a));

printf("\nsize of b =%d",sizeof(b));

printf("\nsize of ptr =%d",(ptr));

}

\*d ----- de reference

Invalid use of void expression

#include <stdio.h>

int main(){

int a =10;

float b = 20.2;

int \*ptr1 = NULL;

void \*ptr = NULL;

printf("\nAddress of a=%u and its value %d",&a,a);

printf("\naddrress of b =%u and its value %f",&b,b);

printf("\naddress of ptr=%u and its value =%u",&ptr,ptr);

printf("\nsize of a=%d",sizeof(a));

printf("\nsize of b =%d",sizeof(b));

printf("\nsize of ptr =%d",(ptr));

ptr = &a;

ptr1 =&a;

printf("\nptr value = %u",ptr);

printf("\nvalue pointed by ptr=%d ", \*(int \*)ptr);

printf("\nvalue pointed by ptr1=%d", \*ptr1);

}

THUMB RULES:

1.address of operation and \* --- nullify each other----&\*

op[] -- \*op array to pointer

\*op --- op[]

E.g.:

#include <stdio.h>

int main(){

int a =10;

int b[3]={10,14,15};

int i;

int \*ptr = NULL;

ptr = b;

for(i =0;i<3;i++){

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

}

\*(ptr+1)=27;

for(i =0;i<3;i++){

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

}

Eg:

#include <stdio.h>

#include <stdlib.h>

int main() {

int a =10;

int \*ptr = NULL;

// ptr = &a;//

//printf("\nvalue stored at ptr = %d",\*ptr);//

\*ptr = 101;

printf("\nvalue stored at ptr = %d",\*ptr);

printf("\nvalue of a =%d",a);

printf("\n\n");

return 0;

}

This code gives an error **segmentation fault** because here the pointer is not pointing to any address.

#include <stdio.h>

#include <stdlib.h>

int main() {

int a =10;

int \*ptr = NULL;

ptr = &a;

printf("\nvalue stored at ptr = %d",\*ptr);

\*ptr = 101;

printf("\nvalue stored at ptr = %d",\*ptr);

printf("\nvalue of a =%d",a);

printf("\n\n");

return 0;

}

output: value stored at ptr = 10

value stored at ptr = 101

value of a =10

**Malloc:**

The function allocates the size bytes and returns a pointer to the allocated memory.

Void \*malloc(size\_t\_size)

Malloc will fail only if it does not allocate the memory it will return NULL.

If malloc not fails it will return BASE ADDRESS.

e.g.:

#include <stdio.h>

#include <stdlib.h>

int main() {

int a =10;

int \*ptr = NULL;

// ptr = &a;

// printf("\nvalue stored at ptr = %d",\*ptr);

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

printf("\naddress of the pointing ptr = %u",ptr);

\*ptr = 101;

printf("\nvalue stored at ptr = %d",\*ptr);

printf("\nvalue of a =%d",a);

printf("\n\n");

return 0;

}

#include <stdio.h>

#include <stdlib.h>

int main() {

int a =10;

int i;

int \*ptr = NULL;

// ptr = &a;

// printf("\nvalue stored at ptr = %d",\*ptr);

ptr = (int \*)malloc(10000000000000000);

if(ptr == NULL){

perror("malloc: ");

exit(0);

}

printf("\naddress of the pointing ptr = %u",ptr);

ptr[0] = 101;

ptr[1] =102;

ptr[2] = 103;

printf("\nvalue stored at ptr = %d",\*ptr);

printf("\nvalue of a =%d",a);

printf("\narray elements are\n");

for(i=0;i<3;i++)

printf("\n%d\n",ptr[i]);

printf("\n\n");

return 0;

}

#include <stdio.h>

#include <stdlib.h>

int main() {

int a =10;

int i;

int \*ptr = NULL;

// ptr = &a;

// printf("\nvalue stored at ptr = %d",\*ptr);

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

if(ptr == NULL){

perror("malloc: ");

exit(0);

}

printf("\naddress of the pointing ptr = %u",ptr);

/\* \*(ptr+0) = 101;

\*(ptr+1) =102;

\*(ptr+2) = 103;\*/

for (i=0;i<3;i++)

printf("\n%d element address = %u",i,&ptr[i]);

\*ptr=101;

printf("\n%d stored at = %u",\*ptr,ptr);

ptr++;

\*ptr=102;

printf("\n%d stored at =%u",&ptr,ptr);

ptr++;

\*ptr=103;

printf("\n%d stored at =%u",&ptr,ptr);

ptr++;

/\*printf("\narray elements are\n");

for(i=0;i<3;i++)

printf("\n%d\n",\*(ptr+i));\*/

printf("\n\n");

return 0;

}

Output:

address of the pointing ptr = 26907296

0 element address = 26907296

1 element address = 26907300

2 element address = 26907304

101 stored at = 26907296

102 stored at =26907300

103 stored at =26907304

**Dangling pointer** ---- Pointer is pointing to the address /reference where is the address/reference id destroyed.

Can overcome this problem by storage classes.

Storage class in C:

**Static**

int func();

int main(){

int ret =0;

ret = func();

printf("\nRet=%d\n",ret);

ret = func();

printf("\nRet = %d",ret);

printf("\n\n");

return 0;

}

int func(){

static int a = 10;

a = a+2;

return a;

}

**Extern** ---gives the reference of the value that may be present in the another file.

Used in mutable array

Auto

**Register**

Accessing the value in register is more faster compared to the static class.

int main(){

register int ret =0;

int i;

for(i =0;i<10;i++)

printf("%d",i);

printf("\n\n");

return 0;

}

POINTER FUNCTIONS:

E.g.:

#include <stdio.h>

int wel(int,int);

int main() {

wel('\n',1);

wel('=',50);

wel('\n',1);

wel('=',21);

printf("%s","Welcome");

wel('=',22);

wel('\n',1);

wel('=',50);

wel('\n',1);

return 0;

}

int wel(int ch ,int n){

int i;

for(i=0;i<n;i++)

putchar(ch);

}

==================================================

=====================Welcome======================

==================================================