**POINTERS**

It will hold the address or it can point to the particular address

Pointer does not belong to any type of datatype.

The size of the pointer should be same for any datatype.

EX:

int \*ptr-----ptr is special variable pointing to interger datatype.

int a=10;

a-home , contents of the home---10

2000----base address

**Types of pointers:**

1.Null pointer

2.Void pointer ----generic pointer(it can hold the address of any datatype, It can points to any type of datatype)

3.Wild pointer

4.Danging pointer

int \*ptr=NULL;----null ponter

float \*ptr; ----- wild pointer

**EX:**

#include <stdio.h>

int main() {

int a=10;

float b=20.2;

long long int \*ptr = NULL;

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

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

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

printf("\n\n");

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

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

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

printf("\n");

return 0;

}

EX:

#include <stdio.h>

int main() {

int a=10;

float b=20.2;

void \*ptr = NULL;

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

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

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

printf("\n\n");

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

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

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

printf("\n");

ptr = &a;

printf("\n ptr value=%u",ptr); ------address of the a will store in ptr

printf(“%d”, \*ptr)----contents of the “a” will display

printf("\n");

return 0;

ERROR: dereferencing 'void \*' pointer

printf("\n value pointed by ptr=%d", \*ptr);----it is unable to convert it to integer

Generic pointers will not convert the types.

**printf("\n value pointed by ptr=%d", \*(int \*)ptr); ------type convertion**

**If we using the void the we need to use the type convertion.**

**When we use specific data type then type convertion is not required.**

**EX: int \*ptr = NULL;**

**EX**: #include <stdio.h>

int main() {

int a=10;

int b[3]={11,12,13};

int \*ptr=NULL;

printf("\nAddress of ptr=%u and stored address in ptr=%u",&ptr,ptr);

ptr=&a;

printf("\nAddress of ptr=%u and stored address in ptr=%u",&ptr,ptr);

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

printf("\nBase Address of b=%u\n",&b[0]);

ptr=&b[0];

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

//array notation

printf("Array notation");

printf("\nb[0]=%d", ptr[0]);

printf("\nb[1]=%d", ptr[1]);

printf("\nb[2]=%d", ptr[2]);

//pointer notation

printf("\n pointer notatin");

printf("\nb[0]=%d", \*(ptr+0));

printf("\nb[1]=%d", \*(ptr+1));

printf("\nb[2]=%d", \*(ptr+2));

return 0;

}

**TWO THUMB RULES OF PONTERS:**

1.&\*---------nullify each other

2.[]to a pointer notation------op[]—(convert [] to \*)

\*op------op[]

**EX: UPDATION ON A VALUE**

#include <stdio.h>

int main() {

int a=10;

int b[3]={11,12,13};

int i;

int \*ptr=NULL;

//ptr=&b[0];

//ptr=&\*(b+0);

//ptr=(b+0);

ptr=b;

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

{

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

}

printf("\nUpdate Value\n");

//b[1]=100;

\*(ptr+1)=100;

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

{

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

}

printf("\n");

return 0;

}

**EX:**

#include <stdio.h>

#include<stdlib.h>

int main() {

int a=10;

int \*ptr=NULL;

/\*

ptr=&a;

printf("\n Value stord at ptr =%d", \*ptr);

\*/

\*ptr=101;

printf("\n Value stord at ptr =%d", \*ptr);

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

printf("\n\n");

return 0;

}

ERROR: segmentation fault----because pointer is not pointing to any location.

**DYNAMIC MEMORY ALLOCATION: malloc, calloc,realloc**

**Malloc:**

#inlcude<stdlib.h>

Void \*malloc(size\_t\_size)----size will be created and we can it anywhere.

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

If malloc not fails it will return **BASE ADDRESS.**

**malloc(nmemb\*size);**

**EX:** #include <stdio.h>

#include<stdlib.h>

int main() {

int a=10;

int \*ptr=NULL;

/\*

ptr=&a;

printf("\n Value stord at ptr =%d", \*ptr);

\*/

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

printf("\nAddress of ptr pointing to =%u",&ptr);

\*ptr=101;

printf("\n Value stord at ptr =%d", \*ptr);

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

printf("\n\n");

return 0;

}

**EX:**

#include <stdio.h>

#include<stdlib.h>

int main() {

int a=10;

int i;

int \*ptr=NULL;

/\*

ptr=&a;

printf("\n Value stord at ptr =%d", \*ptr);

\*/

ptr= (int \*)malloc(3\*sizeof(int));

if(ptr == NULL)

{

perror("malloc: ");//error captured by the perror

exit(0);

}

printf("\nAddress of ptr pointing to =%u",&ptr);

//ptr[0]=101;

//ptr[1]=102;

//ptr[2]=103;

/\*

\*(ptr+0)=101;

\*(ptr+1)=102;

\*(ptr+2)=103;

\*/

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

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

\*ptr=101;

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

ptr++;

\*ptr=102;

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

ptr++;

\*ptr=103;

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

/\* printf("\n Value stord at ptr =%d", \*ptr);

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

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

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

{

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

}\*/

printf("\n\n");

return 0;

}

**EX:**

#include <stdio.h>

#include<stdlib.h>

int main() {

int a=10;

int i;

int \*ptr=NULL;

/\*

ptr=&a;

printf("\n Value stord at ptr =%d", \*ptr);

\*/

ptr= (int \*)malloc(3\*sizeof(int));

if(ptr == NULL)

{

perror("malloc: ");//error captured by the perror

exit(0);

}

printf("\nAddress of ptr pointing to =%u",&ptr);

//ptr[0]=101;

//ptr[1]=102;

//ptr[2]=103;

/\*

\*(ptr+0)=101;

\*(ptr+1)=102;

\*(ptr+2)=103;

\*/

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

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

\*ptr=101;

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

ptr++;

\*ptr=102;

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

ptr++;

\*ptr=103;

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

/\* printf("\n Value stord at ptr =%d", \*ptr);

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

\*/

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

ptr--;

ptr--;

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

{

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

ptr++;

}

printf("\n\n");

return 0;

}

**Write a program to find the value present in the list.**

**DANGLING POINTERS:**

Pointer is pointing to the address/reference where in the address/reference is destroyed.

Storage classes in C----static,auto,register,extern

**EX:**

#include <stdio.h>

int \*allocMem();

int main()

{

int a=10;

int \*ptr=NULL;

//ptr=&a;

ptr=allocMem();

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

return 0;

}

int \*allocMem()

{

static int a=10;

return &a;

}

STATIC:

#include <stdio.h>

int func();

int main()

{

int ret=0;

ret=func();

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

ret=func();

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

return 0;

}

int func()

{

static int a=10;//14 or int a=10 -12

a=a+2;

return a;

}

EX: #include <stdio.h>

int func();

int main()

{

int ret=0;

ret=func();

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

ret=func();

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

return 0;

}

int func()

{

static int a=10;//14 or int a=10 -12 ---if we static it become global variable

a=a+2;

return a;

}

**REGISTER:**

Accessing the value in register is more faster when compared to the value stored in stack.

**EX:** #include <stdio.h>

int main()

{

register int ret=0;

int i;

for(i=0;i<100000;i++)

printf("%d\n",i);

return 0;

}

**EXTERN:** Give the reference of the value that may be present in the another file.

**EX:**

#include <stdio.h>

extern CAP=5;

int main()

{

int arr[CAP];

for(int i=0;i<CAP;i++)

{ printf("\nEnter the values:");

scanf("%d",&arr[i]);

}

printf("\nLIST IS:\n");

for(int j=0;j<CAP;j++)

printf("%d\n",arr[j]);

return 0; }

**POINTER FUNCTION:**

**EX:**

#include <stdio.h>

#include<string.h>

int wel(int,int);

int main() {

char msg[]="Welcome";

int i;

// wel('\n',1);

wel('=',50);

wel('\n',1);

wel('=',21);

// printf("welcome");

for(i=0;i<strlen(msg);i++)

wel(msg[i],1);

wel('=',22);

wel('\n',1);

wel('=',50);

return 0;

}

int wel(int ch,int n)

{

int i;

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

putchar(ch);

}

**OUTPUT:**

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

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

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

#include<string.h>

int wel(int,int);

int (\*func)(int,int)=wel;

int main() {

// int (\*func)(int,int)=wel

//char msg[]="Welcome";

//int i;

// wel('\n',1);

wel('=',50);

wel('\n',1);

wel('=',21);

printf("welcome");

//for(i=0;i<strlen(msg);i++)

//wel(msg[i],1);

wel('=',22);

wel('\n',1);

wel('=',50);

return 0;

}

int wel(int ch,int n)

{

int i;

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

putchar(ch);

}

**ERROR: free(): double free detected in tcache 2**

**Aborted(core dump)**

#include <stdio.h>

#include<stdlib.h>

int main() {

int \*ptr=NULL;

ptr=(int \*)malloc(3\*sizeof(int));

free(ptr);

// free(ptr);

printf("\n\n");

return 0;

}