**DAY10**

\*Nesting of functions is not recursive function

\*recursive function means calling itself

\*recursive function may leads stack overflow(memory outrage)

//Demo on recursive function

#include <stdio.h>

int f(int);

int main() {

int res=f(5);

printf("\nRes=%d\n\n",res);

return 0;

}

int f(int v)

{

if(v==0)

return 1;

v--;

f(v);

printf("\nv value in func:%d",v);

return v;

}

output:

v value in func:0

v value in func:1

v value in func:2

v value in func:3

v value in func:4

Res=4

\*In functions all values have different addresses,here the addresses are being destroyed.

\*But in recursive function we have only one adress for different values.here values are being destroyed.

//Demo on recursive function for fibonacci number

#include <stdio.h>

int f(int);

int main() {

int res=f(5);

printf("\nRes=%d\n\n",res);

return 0;

}

int f(int v)

{

int ret;

printf("\nv value in func:%d\n",v);

if(v==1)

return 1;

ret=v\*f(--v);

printf("\nv value in func:%d\n",v);

return ret;

}

POINTERS:\*pointer is a special variable

\*It will point to the address not values.

pointer is declared (dt \*ptrname);

\*pointers does not belongs to any kind of datatype.

\*The size of all pointers are same(4 or 8)it can be any datatype.

ex:int \*ptrname:ptr is a special variable pointing to integer datatype

There are 4 types of pointers:

1.Null:int \*ptr=NULL; it is pointing to nothing

2.void:it is also called as genric pointer.it can hold the address of any datatype

3.wlid:float \*ptr;here we are declaring a pointer without intiliztion so it is called as wlid pointer.

4.danging pointer:poniter is pointing to an reference where an reference is destroyed

\*to overcome danging will use core storage classes static

#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("\nAddress 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",sizeof(ptr));

ptr=&a; =>here we are assingning address of a to ptr by &

printf("\naddress of ptr=%u",ptr);=>here in printf we used only ptr because we are getting only address not the value

printf("\nvalue of ptr=%d",\*(int\*)ptr);=>here we are printing the value of a into ptr

\*here we used type casting

}

output:

Address of a=2327754268 and its value=10

Address of b=2327754264 and its value=20.200001

Address of ptr=2327754256 and its value=0

size of a=4

size of b=4

size of ptr=8

value of ptr=1098943212

=>/\*

thumb rules:

1)&\*=nullfiy each other(no symbols)

2)op[] =>\*op

\*op =>op[]

\*/

#include <stdio.h>

int main()

{

int a=10;

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

int \*ptr=NULL;

//ptr=&b[0]; ptr is pointing to the BA of b=BA

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

//ptr=b+0

ptr=b;

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

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

//b[1]=100;

\*(ptr+1)=100;

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

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

return 0;

}

=>#include <stdio.h>

#include<stdio.h>

int main() {

int a=10;

int \*ptr=NULL;

/\*ptr=&a;

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

\*/=>here we are directly giving value without pointing address

\*ptr=101;

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

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

return 0;

}

output:segmention fault

Dynamic pointer(unnamed address):\*if we want to give value without pointing address we should allocate memory

\*using malloc, calloc,realoc

=>void \*malloc(size):it just allocates base address.

=>\*If the malloc failes it returns null or unique value that can be later be succesffuly

passed to free().

=>calloc:malloc(nmemb\*size);

#include <stdio.h>

#include<stdio.h>

int main() {

int a=10;

int \*ptr=NULL;

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

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

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

ptr[0]=101;

ptr[1]=102;

ptr[2]=103;

\*ptr=101;

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

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

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

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

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

return 0;

}

output:

Address of ptr pointing to =33624768

value stored at ptr=101

value stored at a=10

Array elements are

101

102

103

=>#include <stdio.h>

#include<stdlib.h>

int main() {

int a=10;

int \*ptr=NULL;

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

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

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

ptr[0]=101;

ptr[1]=102;

ptr[2]=103;

\*ptr=101;

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

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

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

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

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

return 0;

}

Address of ptr pointing to =13542080

value stored at ptr=101

value stored at a=10

Array elements are

101

102

103=>here 102 and 103 are not garunty values it can be change

=>#include <stdio.h>

#include<stdlib.h>

int main() {

int a=10;

int i;

int \*ptr=NULL;

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

if(ptr=NULL)

{

perror("maloc:");

exit(0);

}

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

ptr[0]=101;

ptr[1]=102;

ptr[2]=103;

\*ptr=101;

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

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

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

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

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

return 0;

}

output:Segmentation fault

=>#include <stdio.h>

#include<stdlib.h>

int main() {

int a=10;

int i;

int \*ptr=NULL;

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

if(ptr=NULL)

{

perror("maloc:");

exit(0);

}

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

/\*

\*(ptr+0)=101;

\*(ptr+1)=102;

\*(ptr+2)=103;

\*/

\*ptr=101;

ptr++;=>here address of pointer is incrementing

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

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

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

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

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

return 0;

}

=>#include <stdio.h>

#include<stdlib.h>

int main() {

int a=10;

int i;

int \*ptr=NULL;

int \*temp=NULL;

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

if(ptr=NULL)

{

perror("maloc:");

exit(0);

}

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

/\*

\*(ptr+0)=101;

\*(ptr+1)=102;

\*(ptr+2)=103;

\*/

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

printf("\n elemnt 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);

/\*ptr--;

ptr--;

\*/

ptr=temp;

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

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

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

\*/

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

{

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

ptr++;

}

return 0;

}

Assignment:write a program to find a value present in array using dyamic arrays

Dagning:

#include <stdio.h>

int \*allocMem();

int main() {

int a=10;

int \*ptr=NULL;

//ptr=&a;

ptr=allocMem();

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

return 0;

return 0;

}

int \*allocMem()

{

static int a=10;

return &a;

}