**Pointer**

1. **Array name:** pointer (Done)
2. **Pointer to int array: e.g.**

#include <stdio.h>

int main(void) {

int a[5]={123, 456};

int \*p;

p=a;

printf("\*p+1: %d\n", \*p+1);

printf("\*(p+1): %d\n",\*(p+1)); **//\*p+1 is NOT \*(p+1)**

printf("\*(p+5): %d\n",\*(p+5));

printf("\*p++: %d\n",\*p++);

printf("\*(p++): %d\n",\*(p++)); **//\*p++ is the same as \*(p++)**

**// address + 1 and then get value**

printf("++\*p: %d\n",++\*p); **// get value and the value increases 1**

printf("\*(++p): %d\n",\*(++p)); **//address + 1 and then get value**

return 0;

}

1. **Pointer to char array: e.g.**

#include <stdio.h>

int main(void) {

char a[10]="ABCDEF";

char \*p;

p=a;

printf("\*p: %c\n", \*p); **//Ans: ? A**

printf("\*a: %c\n", \*a); **//Ans: ? A**

printf("a: %c\n", a); **//Ans:** **?**

printf("a: %s\n", a);

printf("a++: %s\n", a++); **//Ans: ?**

printf("\*p++: %c\n", \*p++); **//Ans: ?**

printf("\*p: %c\n", \*p); **//Ans: ?**

printf("p: %s\n", p); **//Ans: ?**

return 0;

}

#include <stdio.h>

int main(void) {

char a[10]="guru";

//a="geek"; //Error

printf("a: %s\n",a);

a[0] = 'j';

printf("a[0]: %c\n", a[0]);

char \*p="geek";

p="nerd";

printf("p: %s\n",p);

p[0] = 'j'; //Invalid

printf("p[0]: %c\n", p[0]);

return 0;

}

1. **Const pointer: e.g.**

#include <stdio.h>

int main(void) {

int x=1234;

int \*const p1 = &x;      **//const pointer must be initialed**

**//otherwise, error like**

**//int \*const p0; p0 = &x;**

printf("\*p1: %d\n",\*p1);

int y=4567;

**//p1=&y; //Error: const pointer can't change location in memory**

const int z=1111;

const int \*p2 = &z;

printf("\*p2: %d\n",\*p2);

p2=&x; **//p2 is regular pointer, which may point to const**

printf("\*p2: %d\n",\*p2);

const int w=78910;

const int \*const p3 = &w; **//const point will point to const var**

**//w=78910; //Error**

printf("\*p3: %d\n",\*p3);

return 0;

}

1. **Void pointer:** A void pointer is a pointer that has no associated data type with it. A void pointer can hold address of any type and can be typcasted to any type.

**e.g.**

#include <stdio.h>

int main(void) {

int a = 10;

char b = 'A';

void \*p = &a; // void pointer holds ONLY address of int 'a'

printf("p:%p\n", p);

printf("\*(int\*)p:%d\n", \*(int\*)p);

p = &b;

printf("p:%p\n", p);

//printf("\*p:%d\n", \*p); //Error: no concrete DType

printf("\*(char\*)p:%c\n", \*(char\*)p);

return 0;

}

#include <stdio.h>

#include <stdlib.h>

int main(void)

{

int n=1;

// Note that malloc() returns **void \*** which can be

// typecasted to any type like **int \*, char \*, ..**

int \*x = malloc(sizeof(int) \* n);

char \*y = malloc(sizeof(char) \* n);

}

#include<stdio.h>

int main()

{

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

void \*ptr = &a; //Point to 1st adr in a

ptr = ptr + sizeof(int); //Move pointer to next elem in array

printf("\*(int \*)ptr: %d\n", \*(int \*)ptr); //casting

return 0;

}

#include<stdlib.h>

#include<stdio.h>

#include<time.h>

void \*randomNum(void \*a, int length)

{

void\* result;

result = a + rand()%(length-1);

return result;

}

int main(void){

int i;

srand(13);

int array[9]={1,5,6,85,132,65463,1354,5863,134};

for (i=0;i<9; i++){

printf("%d\n",\*(int\*)randomNum(array,9));

}

return 0;

}

1. **Pointer of array & array of pointer:**

**e.g.**

#include<stdio.h>

int main()

{

int arr[] = { 3, 5, 6, 7, 9 };

int \*p = arr; **//point of array or pointer to array**

int (\*ptr)[5] = &arr; **//array of pointer**

printf("p = %p, ptr = %p\n", p, ptr);

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

printf("sizeof(p) = %lu, sizeof(\*p) = %lu\n", sizeof(p), sizeof(\*p));

printf("sizeof(ptr) = %lu, sizeof(\*ptr) = %lu\n", sizeof(ptr), sizeof(\*ptr));

printf("\*(\*ptr): %d\n",\*(\*ptr)); **//Why is this result?**

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

printf("\*\*ptr+1: %d\n",\*\*ptr+1);

return 0;

}

**Pointer to array** vs **2-D array:**

#include<stdio.h>

int main()

{

int arr[3][4] = {

{10, 11, 12, 13},

{20, 21, 22, 23},

{30, 31, 32, 33}

};

int (\*ptr)[4]; **//Error: if int \*ptr[4]**

ptr=arr;

printf("%p %p %p\n", ptr, ptr + 1, ptr + 2);

printf("%p %p %p\n", \*ptr, \*(ptr + 1), \*(ptr + 2));

printf("%d %d %d\n", \*\*ptr, \*(\*(ptr + 1) + 2), \*(\*(ptr + 2) + 3));

printf("%d %d %d\n", ptr[0][0], ptr[1][2], ptr[2][3]);

return 0;

}

1. **Pointer to pointer:**

#include<stdio.h>

int main()

{

int arr[3][4] = {

{10, 11, 12, 13},

{20, 21, 22, 23},

{30, 31, 32, 33}

};

for(int i=0; i<12; i++){

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

}

for(int i=0; i<12; i++){

printf("\*\*arr+%d: %d\n", i,\*\*arr+i); **//Pointer to pointer**

}

return 0;

}

1. **Pointer argument in function: e.g.**

#include<stdio.h>

void swap(int\*, int\*);

int main()

{

int m = 10, n = 20;

printf("m = %d\n", m);

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

swap(&m, &n); //passing address of m and n to the swap function

printf("After Swapping:\n\n");

printf("m = %d\n", m);

printf("n = %d", n);

return 0;

}

/\* pointer 'a' and 'b' holds and

points to the address of 'm' and 'n'

\*/

void swap(int \*a, int \*b)

{

int temp;

temp = \*a;

\*a = \*b;

\*b = temp;

}

1. **Function pointer: e.g.**

#include <stdio.h>

void fun(int a) {

printf("Value of a is %d\n", a);

}

int main(void) {

void (\*fun\_ptr)(int) = &fun; **//Function name is like array name**

//void (\*fun\_ptr)(int) = fun;

fun\_ptr(10);

return 0;

}

#include <stdio.h>

void add(int a, int b) {

printf("Addition is %d\n", a+b);

}

void subtract(int a, int b) {

printf("Subtraction is %d\n", a-b);

}

void multiply(int a, int b) {

printf("Multiplication is %d\n", a\*b);

}

int main(void) {

// fun\_ptr\_arr is an array of function pointers

void (\*fun\_ptr\_arr[])(int, int) = {add, subtract, multiply};

unsigned int ch, a = 15, b = 10;

printf("Enter Choice: 0 for add, 1 for subtract and 2 for multiply\n");

scanf("%d", &ch);

if (ch > 2)

return 0;

(\*fun\_ptr\_arr[ch])(a, b);

return 0;

}

#include <stdio.h>

void fun1() { printf("Fun1\n"); }

void fun2() { printf("Fun2\n"); }

**// Function pointer as argument in function**

void wrapper(void (\*fun)()) {

fun();

}

int main(void) {

wrapper(fun1);

wrapper(fun2);

return 0;

}

#include <stdio.h>

void foo(char \*x,char \*y){

printf("\*x: %c, \*y: %c\n", \*x, \*y);

}

void swap(char \*x,char \*y){

char temp;

temp=\*x;

\*x=\*y;

\*y=temp;

}

void swap\_print(char i, void (\*pfn)(char\*, char\*)){

char a='K', b='O';

pfn(&a, &b);

printf("%c%c %c\n",a, b, i);

}

int main(void){

char a='!';

swap\_print(a, swap);

swap\_print(a, foo);

}

1. **Return address from local variable to function : e.g.**

C language does not advocate to return the address of a local variable to outside of the function, so you would have to define the local variable as static variable.

#include <stdio.h>

int \*foo(){

static int a[10]={123};

//int a[10]={123}; //Will create compiling warning

return a;

}

int main(void){

int \*x;

x=foo();

for(int i=0; i<5; i++){

x[i]=i+1;

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

}

}

1. **Programmer can be allowed to set pointer address, but can’t be allowed to assign value to this pointer at the location: e.g.**

#include <stdio.h>

#include <stdint.h> //define data type

int main(void) {

int \*p=NULL;

int a=123, b=456;

int \*p1=&a, \*p2=&b;

p=(int\*)((uintptr\_t)(p1)+(uintptr\_t)(p2));

printf("p1: %p, p2:%p, p:%p\n", p1, p2, p);

\*p=123; //Segment fault

printf("p1: %p, p2:%p, p:%p\n", p1, p2, p);

return 0;

}