Pointer

Introduction

In programming languages, normal variable stores a value, but pointer stores the address of another variable of the same data type. int, float, char, double, short, etc. belongs to the pointer variable. The size of pointer depends on the architecture. The 32-bit architecture the size of pointer is 2 byte.

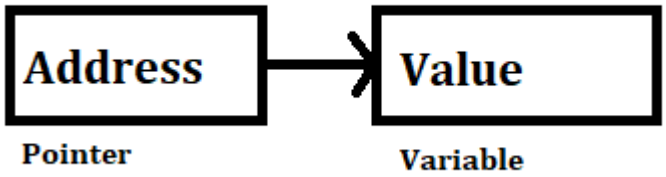
Example:

```
#include<stdio.h>
int main()
{
        int addr1;
        char addr2[10];
        printf("Address of addr1: %u", &addr1);
        printf("Address of addr2: %u", &addr2);
        return 0;
}
```

Define: A Pointer is a variable that stores or pointer address of another variable and the value of a pointer variable gets in another memory location.

Steps of Pointer variable:

1. Define the pointer variable.
2. Assign the address of a variable to a pointer.
3. Access the value at the address variable.



Declaring Pointer Variable:

A pointer is declared with data type of the variable to which it points to and name of the variable.

Syntax:

data_type *variable_name

Where

data_type: It is the data type of the variable to which pointer points to.

variable_name: It is the name of the variable to which pointer points.

Example:

int *pointer

Initializing Pointer Variable:

Pointer initializing is the process of assigning address of a variable to a pointer variable. To initialize the pointer variable, use the following steps:

- Declare a pointer variable and note down the data type.
- Declare another variable with the same data type as that of pointer variable.
- Initialize ordinary variable and assign some value to it.
- Initialize pointer by assigning the address of ordinary variable to pointer variable.

Example:

```
#include <stdio.h>
int main()
{
   int a;
   a = 10;
   int *pointer = &a;
   printf("%d\n", *pointer);
   printf("%d\n", *&a);
   printf("%u\n", &a);
   printf("%u\n", pointer);
   printf("%u\n", &pointer);
   return 0;
}
```

Pointer and Array

Array are closely related to pointer in C programming Difference is that a pointer variable takes different addresses as value whereas, in case of array it is fixed.

Example:

```
#include<stdio.h>
int main()
{
        int i;
        int arr[5] = {2, 4, 6, 8, 10};
        int *ptr = arr;
        for(i=2; i<=6; i++)
        {
                printf("The Value is %d\n", *ptr);
                ptr++;
        }
        return 0;
}
```

Output:

The Value is 2

The Value is 4

The Value is 6

The Value is 8

The Value is 10

Pointer and Function

It is possible to declare a pointer pointing to a function which can then be used as an argument in another function.

Syntax: type (*ptrname)(parameters);

Example: int (*sum) ()

A function pointer can point to a specific when it is assigned the name of that function.

int sum(int, int);

int(*s)(int int);

s = sum;

Here, s is a pointer to a function sum. Now sum can be called using function pointer s along with providing the required argument values.

s(1, 2);

Example:

```
#include<stdio.h>
int sum(int x, int y)
{
        return x + y;
}
int main()
{
        int (*fp)(int, int);
        fp = sum;
        int s=fp(10, 15);
```

```
        printf("Sum is %d", s);
        return 0;
}
```

Output:

sum is 25

Pointer Arithmetic

It is possible to perform arithmetic operations on pointer like:

- Increment (++)
- Decrement (--)
- Addition and subtraction of integer to a pointer
- Pointer arithmetic between two pointer
- Pointer comparison

1. Increment (++)

Incrementing a pointer to an integer data will cause its value to be incremented by two. Pointer in increment depends on the data type to which it points to.

Example:

```
#include <stdio.h>
int main()
{
    int number = 10;
    int *pointer;
    pointer = &number;
    printf("Pointer " "before Increment: ");
    printf("%p \n", pointer);
    pointer++;
    printf("Pointer after" " Increment: ");
    printf("%p \n\n", pointer);
    return 0;
}
```

Output:

Pointer before Increment: 0x7ffd0a92614c

Pointer after Increment: 0x7ffd0a926150

2. Decrement (--):

Decrementing a pointer to an integer data will cause its value to be decremented by size two.

Example:

```
#include <stdio.h>
int main()
{
    int number = 10;
    int *pointer;
    pointer = &number;
    printf("Pointer " "before Decrement: ");
    printf("%p \n", pointer1);
    pointer--;
    printf("Pointer after" " Decrement: ");
    printf("%p \n\n", pointer1);
    return 0;
}
```

Output:

Pointer before Decrement: 0x7ffef36697ac

Pointer after Decrement: 0x7ffef36697a8

3. Addition and Subtraction of integer to a pointer:

It is possible to add integer to add integer to pointer variable or to subtract integer from a pointer variable. An integer may be added to a pointer (+ or +=). In C we can add any integer number to Pointer.

4. Pointer Arithmetic between two pointer:

If we have two pointer p1 and p2 of base type pointer to int with addresses 1000 and 1016 respectively, then p2-p1 will give 4, since the size of int type is 4 bytes.

5. Pointer Comparison:

Relational pointers can be used to compare two pointers. Pointers cannot be multiplied or divided.

Example:

```
#include <stdio.h>
int main()
{
  int *p2;
  int *p1;
  p2 = (int *)1000;
  p1 = (int *)1002;
  if(p1 > p2)
    {
          printf("P1 is greater than p2");
    }
```

```
    else
    {
            printf("P2 is greater than p1");
    }
    return(0);
}
```

Output:

P1 is greater than p2

Dynamic Memory Allocation

The dynamic allocation of memory during the program execution is achieved through the following built in function.

They are 4 functions of stdlib.h header file.

1. malloc
2. calloc
3. realloc
4. free

1. malloc

malloc method is used to dynamically allocate a single large block of memory with the specified size. It returns a pointer of type void which can be cast into a pointer of array form.

Syntax:

ptr = (cast-type*) malloc(byte-size);

Example:

ptr = (int*) malloc(100 * sizeof(int));

2. calloc

calloc allocates multiple blocks of storage, each of same size. It takes the number of elements and the size of each element, initializes each element to zero and then returns a void pointer to the memory.

Syntax:

void *calloc(n, element-size);

Example:

ptr = (float*) calloc(20, sizeof(float));

3. realloc

realloc function modifies the allocated memory size by malloc and calloc function to new size.

realloc should only be used for dynamically allocated memory.

Syntax:

void *realloc(ptr, new_size);

Here, ptr is realloc with size of new_size.

4. free

This function frees the allocated memory by **malloc, calloc, realloc function** and returns the memory to the system.

Syntax:

void free(ptr);