SNS COLLEGE OF ENGINEERING

Kurumbapalayam (Po), Coimbatore – 641 107

An Autonomous Institution

Accredited by NBA – AICTE and Accredited by NAAC – UGC with 'A' Grade
Approved by AICTE, New Delhi & Affiliated to Anna University, Chennai

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

COURSE NAME : 23ITT101 PROBLEM SOLVING AND C PROGRAMMING

I YEAR /II SEMESTER

Unit 4- FUNCTIONS AND POINTERS

Topic 3: Pointers - Definition – Initialization

Brain Storming

1. How to access memory location?

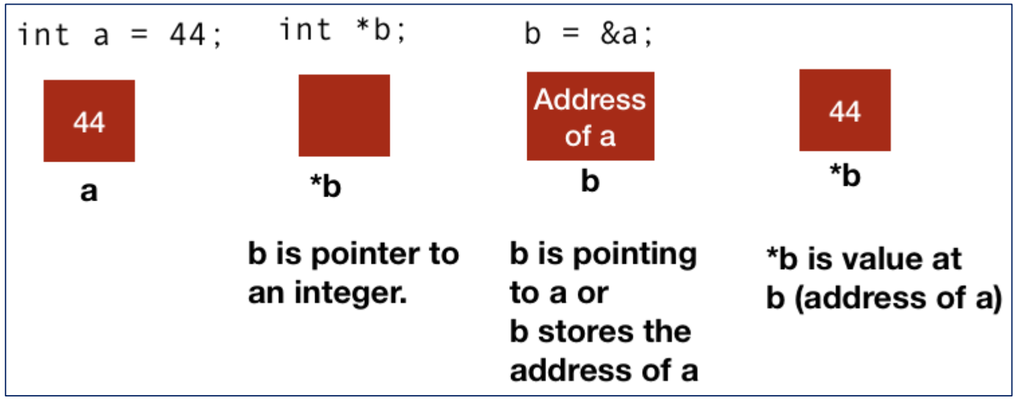
- Hint: int a=5;
- Single storage location is alloted for 5 in a variable “a”.
- How to access memory location?

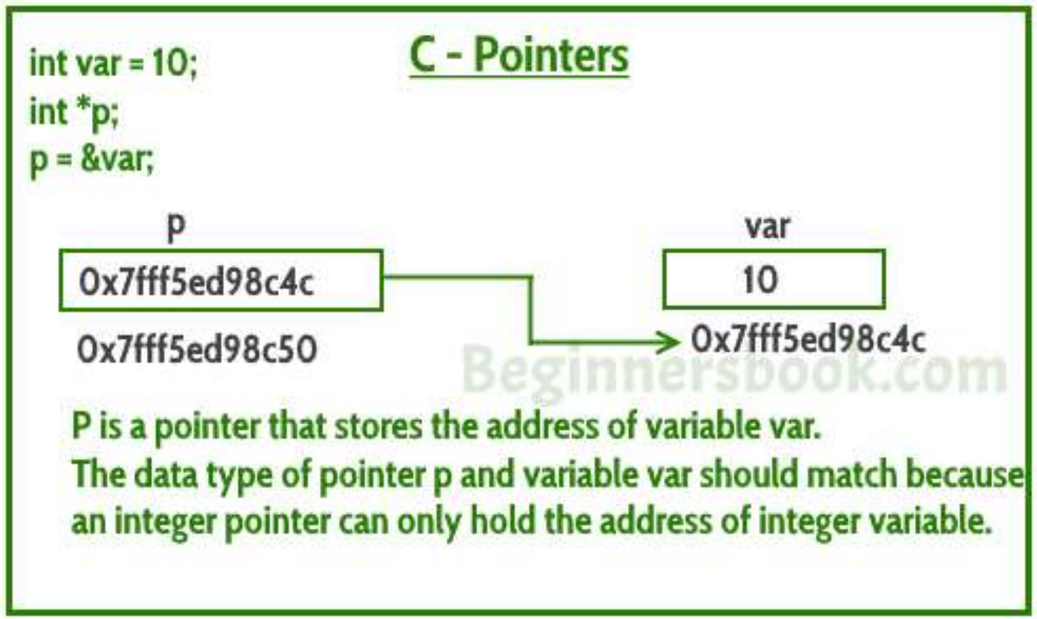
Pointer

- The pointer in C language is a variable which stores the address of another variable.
- This variable can be of type int, char, array, function, or any other pointer.
- The size of the pointer depends on the architecture.
- **However, in 32-bit architecture the size of a pointer is 2 byte.**

Example

- int *a;//pointer to int
- char *c;//pointer to char





Pointer Operator

Operator	Operator Name	Purpose
*	Value at Operator	Gives Value stored at Particular address
&	Address Operator	Gives Address of Variable

Example program

```
#include<stdio.h>
int main()
{
int number=50;
int *p;
p=&number; // or int *p=&number
printf("Address of p variable is %x \n",p);
printf("Value of p variable is %d \n",*p);
return 0;
}
```

OUTPUT:

Address of p variable is fff4

Value of p variable is 50

```
/* Sum of two integers using pointers*/
  #include <stdio.h>
 void main()
{
int first, second, *p, *q, sum;
printf("Enter two integers to add\n");
scanf("%d%d", &first, &second);
p = &first;
q = &second;
sum = *p + *q;
printf("Sum of entered numbers =
%d\n",sum);

 }
```

Pointer Flexibility

Pointers are flexible. We can make the same pointer to point to different data variables in different statements.

Example;

```
int x, y, z, *p;
. . . . .
p = &x;
. . . . .
p = &y;
. . . . .
p = &z;
. . . . .
```

We can also use different pointers to point to the same data variable. Example;

```
int x;
int *p1 = &x;
int *p2 = &x;
int *p3 = &x;
. . . . .
. . . . .
```

With the exception of NULL and 0, no other constant value can be assigned to a pointer variable. For example, the following is wrong:

```
 int *p = 5360; / *absolute address */
```

NULL Pointer

- A pointer that is not assigned any value but NULL is known as the NULL pointer.
- If you don't have any address to be specified in the pointer at the time of declaration, you can assign NULL value. int *p=NULL;
- With the exception of NULL and 0, no other constant value can be assigned to a pointer variable.

ACCESSING A VARIABLE THROUGH ITS POINTER

```
void main() {
int x, y;
 int *ptr;
 x = 10;
 ptr = &x;
 y = *ptr;
 printf("Value of x is %d\n\n",x);
 printf("%d is stored at addr %u\n", x, &x);
 printf("%d is stored at addr %u\n", *&x, &x);
 printf("%d is stored at addr %u\n", *ptr, ptr);
 printf("%d is stored at addr %u\n", ptr, &ptr);
 printf("%d is stored at addr %u\n", y, &y);
 *ptr = 25;
 printf("\nNow x = %d\n",x);
}
```

Output:

```
Value of x is 10
 10 is stored at addr 4104
 10 is stored at addr 4104
 10 is stored at addr 4104
 4104 is stored at addr 4106
 10 is stored at addr 4108
 Now x = 25
```

Pointer Arithmetic

- Following arithmetic operations are possible on the pointer in C language:
- Increment
- Decrement
- Addition
- Subtraction
- Comparison

Incrementing Pointer in C

- If we increment a pointer by 1, the pointer will start pointing to the immediate next location.
- This is somewhat different from the general arithmetic since the value of the pointer will get increased by the size of the data type to which the pointer is pointing.
- The Rule to increment the pointer is given below:
- **new_address= current_address + i * size_of(data type)**

Conti...

Where i is the number by which the pointer get increased.

32-bit:

For 32-bit architecture, it will be incremented by 2 bytes.

64-bit:

For 64-bit architecture, it will be incremented by 4 bytes.

Let's see the example of incrementing pointer variable on 64-bit architecture.

```
#include<stdio.h>
int main(){
int number=50;
int *p;//pointer to int
p=&number;//stores the address of number variable
printf("Address of p variable is %u \n",p);
p=p+1;
printf("After increment: Address of p variable is %u \n",p); // in our case,
p will get incremented by 4 bytes.
return 0;
}
```

Output

- Address of p variable is 3214864300
- After increment: Address of p variable is 3214864304

- **This is similar for Decrementing Pointer**
- Address of p variable is 3214864300
- After Decrement: Address of p variable is 3214864296

```
// C Program to illustrare pointer comparision
#include <stdio.h>

void main()
{
    // declaring array
    int arr[5];

    // declaring pointer to array name
    int* ptr1 = &arr;
    // declaring pointer to first element
    int* ptr2 = &arr[0];

    if (ptr1 == ptr2) {
        printf("Pointer to Array Name and First Element are Equal.");
                          }
    else {
        printf("Pointer to Array Name and First Element are not Equal.");
          }
}
```

Output:

Pointer to Array Name and First Element are Equal.

// C program to illustrate Subtraction of two pointers

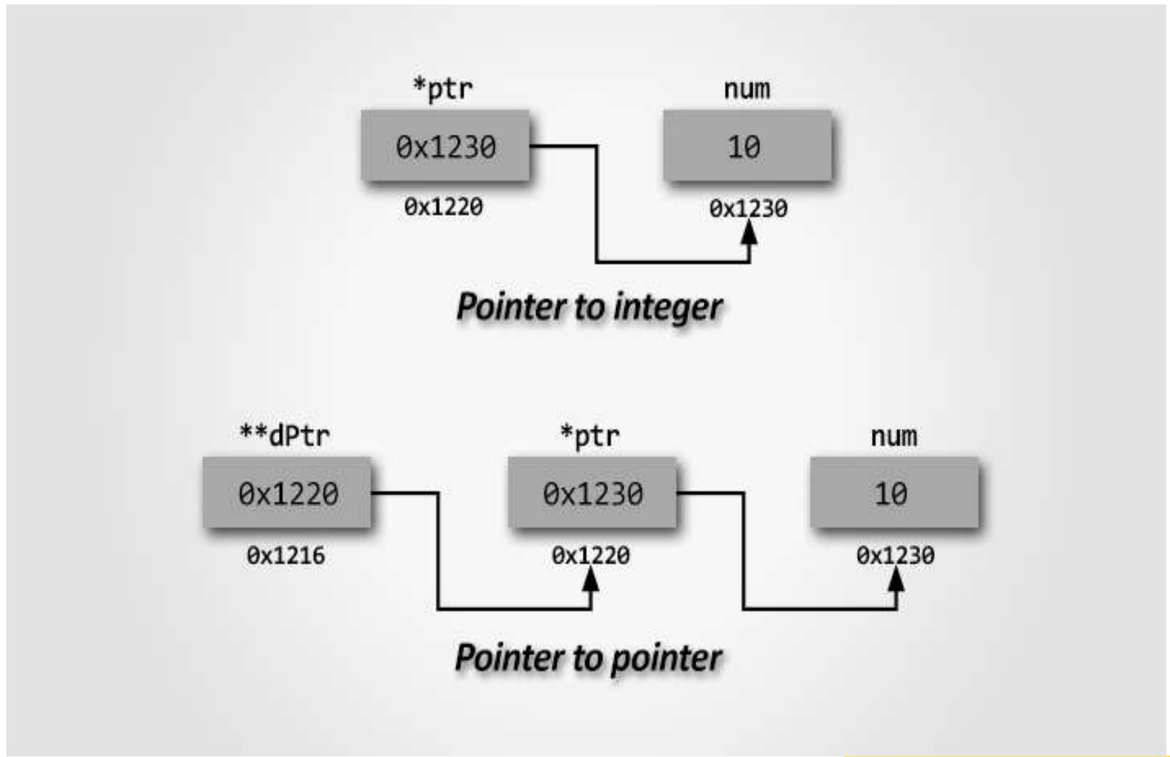
```
#include <stdio.h>

void main()
{
    int x = 6; // Integer variable declaration
    int y = 4;

    // Pointer declaration
    int *ptr1, *ptr2;

    ptr1 = &y; // stores address of y
    ptr2 = &x; // stores address of x

    printf(" ptr1 = %u, ptr2 = %u\n", ptr1, ptr2);    // %p gives an hexa-decimal value,
    // We convert it into an unsigned int value by using %u

    // Subtraction of ptr2 and ptr1
    x = ptr1 - ptr2;

printf("Subtraction of ptr2 from ptr1 is %d\n",  x);
}
```

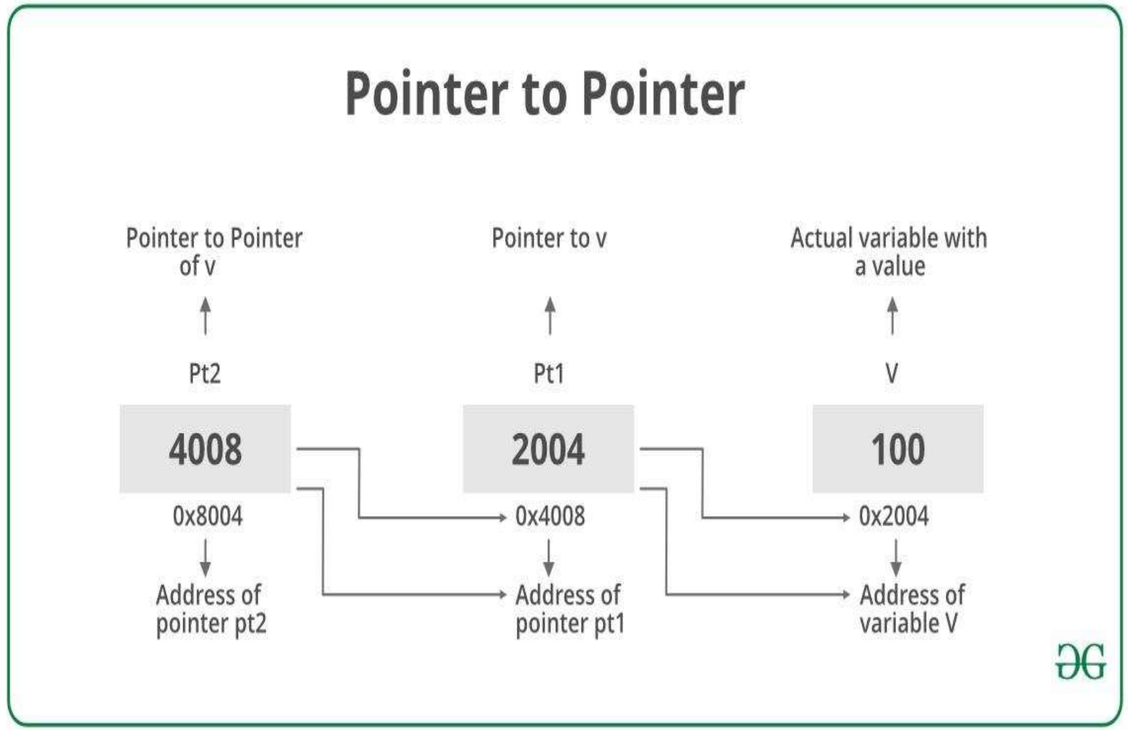
Output:

```
 ptr1 = 2715594428, ptr2 = 2715594424
Subtraction of ptr2 from ptr1 is 1
```

Pointer to Pointer / Double Pointer

- A pointer to a pointer is a form of multiple indirection, or a chain of pointers.
- Normally, a pointer contains the address of a variable.
- When we define a pointer to a pointer, the first pointer contains the address of the second pointer, which points to the location that contains the actual value as shown below.





Pointers and arrays

Example 1: Pointers and Arrays

```c
#include <stdio.h>
int main() {
  int i, x[6], sum = 0;
  printf("Enter 6 numbers: ");
  for(i = 0; i < 6; ++i) {
  // Equivalent to scanf("%d", &x[i]);
      scanf("%d", x+i);

  // Equivalent to sum += x[i]
      sum += *(x+i);
  }
  printf("Sum = %d", sum);
  return 0;
}
```

```
Enter 6 numbers:  2
 3
 4
 4
 12
 4
Sum = 29
```

Traversing/Accessing array using pointers

```
int myNumbers[4] = {25, 50, 75, 100};
int *ptr = myNumbers;
int i;

for (i = 0; i < 4; i++)
{
  printf("%d\n", *(ptr + i));
}
```

Result:

```
25
50
75
100
```

```
int myNumbers[4] = {25, 50, 75, 100};

// Change the value of the first element to 13
*myNumbers = 13;

// Change the value of the second element to 17
*(myNumbers +1) = 17;

// Get the value of the first element
printf("%d\n", *myNumbers);

// Get the value of the second element
printf("%d\n", *(myNumbers + 1));
```

Result:

```
13
17
```

```
#include <stdio.h>
// Function to sort the numbers using pointers
void sort(int n, int* ptr)
{
    int i, j, t;
    // Sort the numbers using pointers
    for (i = 0; i < n; i++) {

        for (j = i + 1; j < n; j++) {

            if (*(ptr + j) < *(ptr + i)) {

                t = *(ptr + i);
                *(ptr + i) = *(ptr + j);
                *(ptr + j) = t;
            }
        }
    }
    // print the numbers
    for (i = 0; i < n; i++)
        printf("%d ", *(ptr + i));
}
```

```
int main()
{
    int n = 5;
    int arr[] = { 0, 23, 14, 12, 9 };

    sort(n, arr);

    return 0;
}
```

OUTPUT:

0 9 12 14 23

References

1. Reema Thareja, “Programming in C”, Oxford University Press, Second Edition, 2016

Thank You