SNS COLLEGE OF TECHNOLOGY

Coimbatore – 35

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 INFORMATION TECHNOLOGY

PROGRAMMING IN C AND DATA STRUCTURES

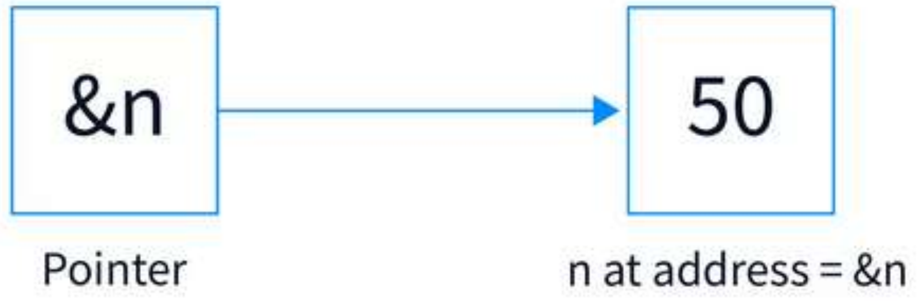
I YEAR - II SEM

UNIT III - ARRAYS AND INTRODUCTION TO DATA STRUCTURES

TOPIC – POINTERS

Introduction

- Pointers in C are used to store the address of variables or a memory location. This variable can be of any data type i.e, int, char, function, array, or any other pointer. The pointer of type void is called **Void pointer** or **Generic pointer**. Void pointer can hold address of any type of variable. The size of the pointer depends on the computer architecture like 16-bit, 32-bit, and 64-bit.
- A pointer is a variable whose value is the address of another variable of the same type. The variable's value that the pointer points to is accessed by dereferencing using the * operator.
- We can access the address in our C program using the & operator.



Syntax of pointers

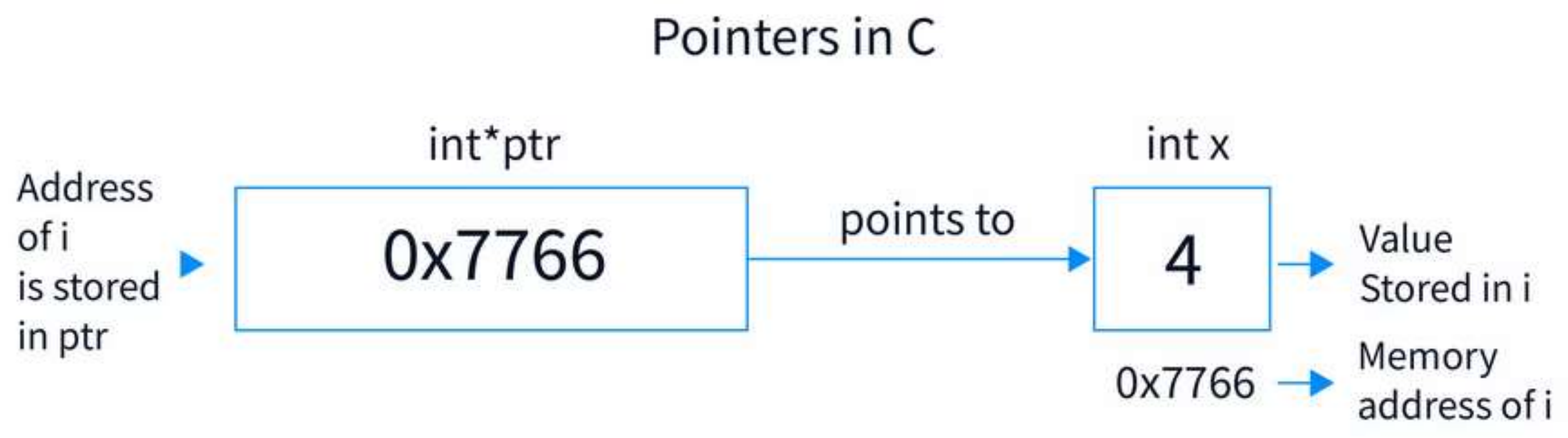
data_type * pointer_variable_name;

Example:

```
int    *ptr_in;        // pointer to an integer
char   *ptr_ch ;       // pointer to a character
double *ptr_dbl;       // pointer to a double
float  *ptr_fl;        // pointer to a float
```

How to Use Pointers in C?

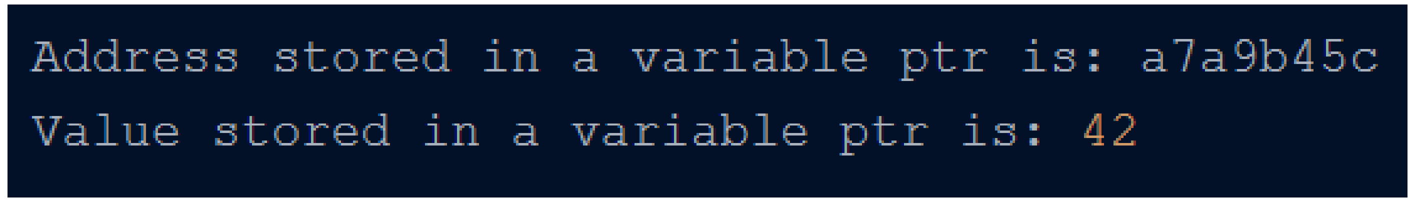
1. Declare a pointer variable.
2. A variable's address is assigned to a pointer using the **&** operator.
3. Use the address in the pointer variable to get the value by using the *(asterisk) operator, which returns the variable's value at the address indicated by its argument.



In the given example, a variable int i = 4 is declared, the address of variable i is 0x7766. A pointer variable int *ptr=&i is declared. It contains the address of variable int i. The value of *ptr will be value at address 0x7766; that value would be 4.

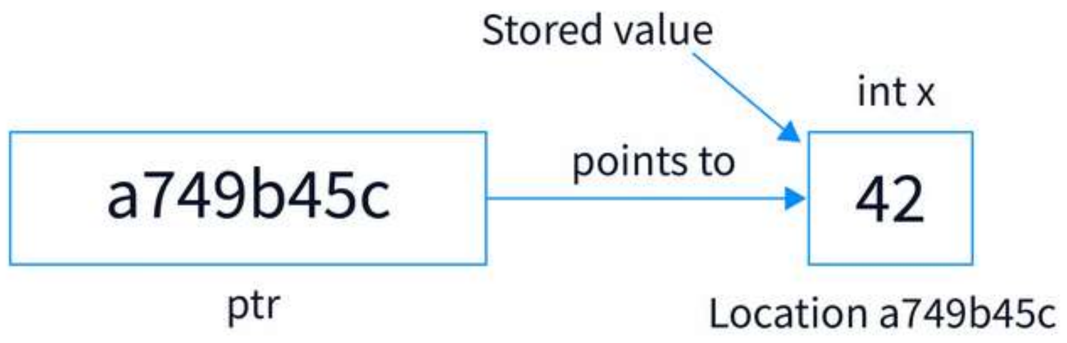
Example

```
#include <stdio.h>
int main()
{
int x = 42;    //variable declaration
int *ptr;     //pointer variable declaration
ptr = &x;    //store address of variable x in pointer
ptr
//printing the address
printf("Address stored in a variable ptr is: %x \n",
ptr);
//printing the value
printf("Value stored in a variable ptr is: %d \n",
*ptr);
return 0;
}
```



```
Address stored in a variable ptr is: a7a9b45c
Value stored in a variable ptr is: 42
```

Explanation



1. In the given example, an int variable x is declared first.
2. The memory location where variable x is declared is a7a9b45c. The value stored in x is 42.
3. The pointer variable ptr is declared using ***(asterisk)** symbol, as mentioned that the data type of the pointer will also be the same as the variable it will point to.
4. In this, ptr = &x, by using & operator, the address of x variable is stored in ptr.
5. The value stored in x is accessed using * operator, *ptr will give the value at location **a7a9b45c**, i.e., 42.

Types of Recursion

There are two types of recursion in the C language.

Direct Recursion

Indirect Recursion

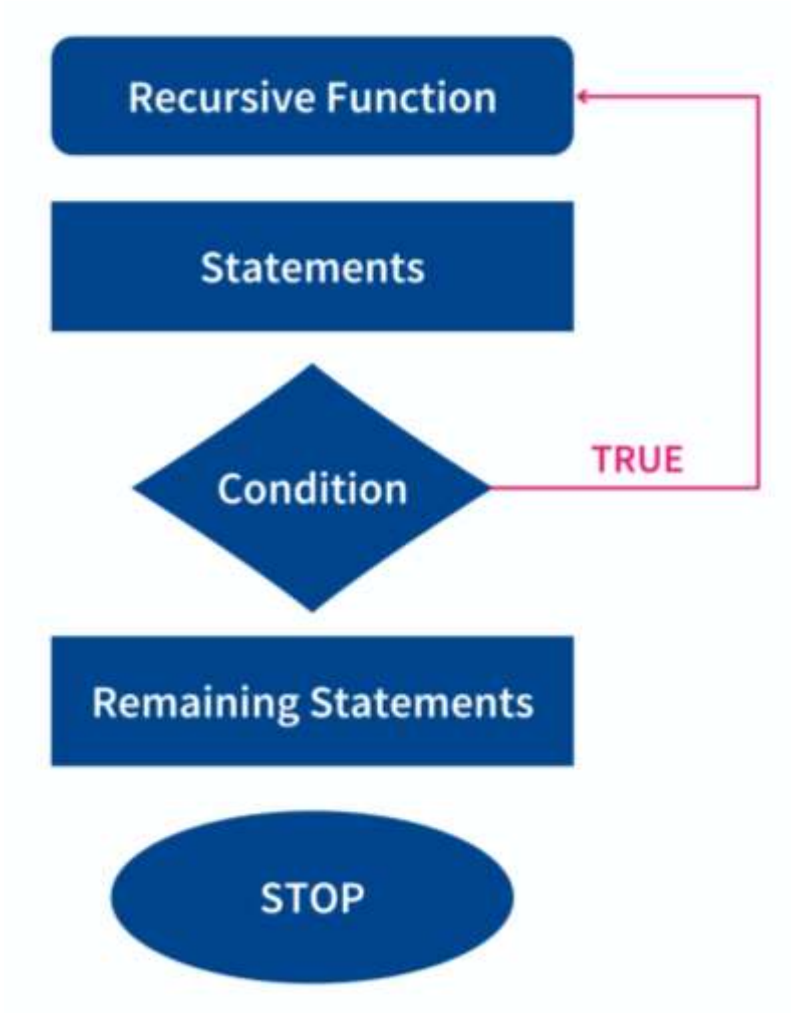
1. Direct Recursion in C

Direct recursion in C occurs when a function calls itself directly from inside. Such functions are also called direct recursive functions.

Following is the structure of direct recursion.

```
function_01( )
{
 //some code
 function_01( );
 //some code
}
```

Flowchart of Recursion



In the following image, there is a recursive function inside which there is a recursive call that calls the recursive function until the condition of the problem is true. If the condition gets satisfied, then the condition is false, and the program control goes for the remaining statements and stops the program.

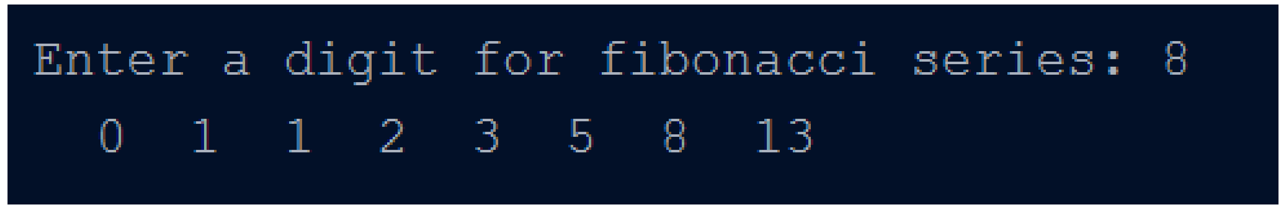
How does Recursion Work?

- ✓ The recursive function or method has two main parts in its body, i.e., the base case and the recursive case.
- ✓ While the recursive method is executed, first, the base case is checked by the program.
- ✓ If it turns out true, the function returns and quits; otherwise, the recursive case is executed.
- ✓ Inside the recursive case, we have a recursive call that calls the function inside which it is present.

Example – Fibonacci Series

```c
#include<stdio.h>
int fibonacci_01(int i)
{
 if (i == 0)
  {
   return 0;
  }
 if (i == 1)
  {
   return 1;
  }
 return fibonacci_01(i - 1) + fibonacci_01(i - 2);
}
int main()
{
  int i, n;
  printf("Enter a digit for fibonacci series: ");
  scanf("%d", & n);
  for (i = 0; i < n; i++)
   {
    printf(" %d ", fibonacci_01(i));
   }
  return 0;
}
```

1. In the given C program, we have declared a function named fibonacci_01().
2. It takes an integer i as input and returns the ith element of the Fibonacci series.
3. At first, the main() function will be executed where we have taken two variables i and n.
4. We will take input from the user that will be stored in n, and the for loop will execute till n iteration where with each iteration, it will pass the parameter to fibonacci_01() function where the logic for the Fibonacci series is written.
5. Now inside fibonacci_01() function, we have nested if-else.
6. If input = 0, it will return 0, and if the input = 1, it will return 1. These are the base cases for the Fibonacci function.
7. If the value of i is greater than 1, then fibonacci(i) will return fibonacci_01 (i - 1) + fibonacci_01 (i -2) recursively, and this recursion will be computed till the base condition.



```
Enter a digit for fibonacci series: 8
  0  1  1  2  3  5  8  13
```

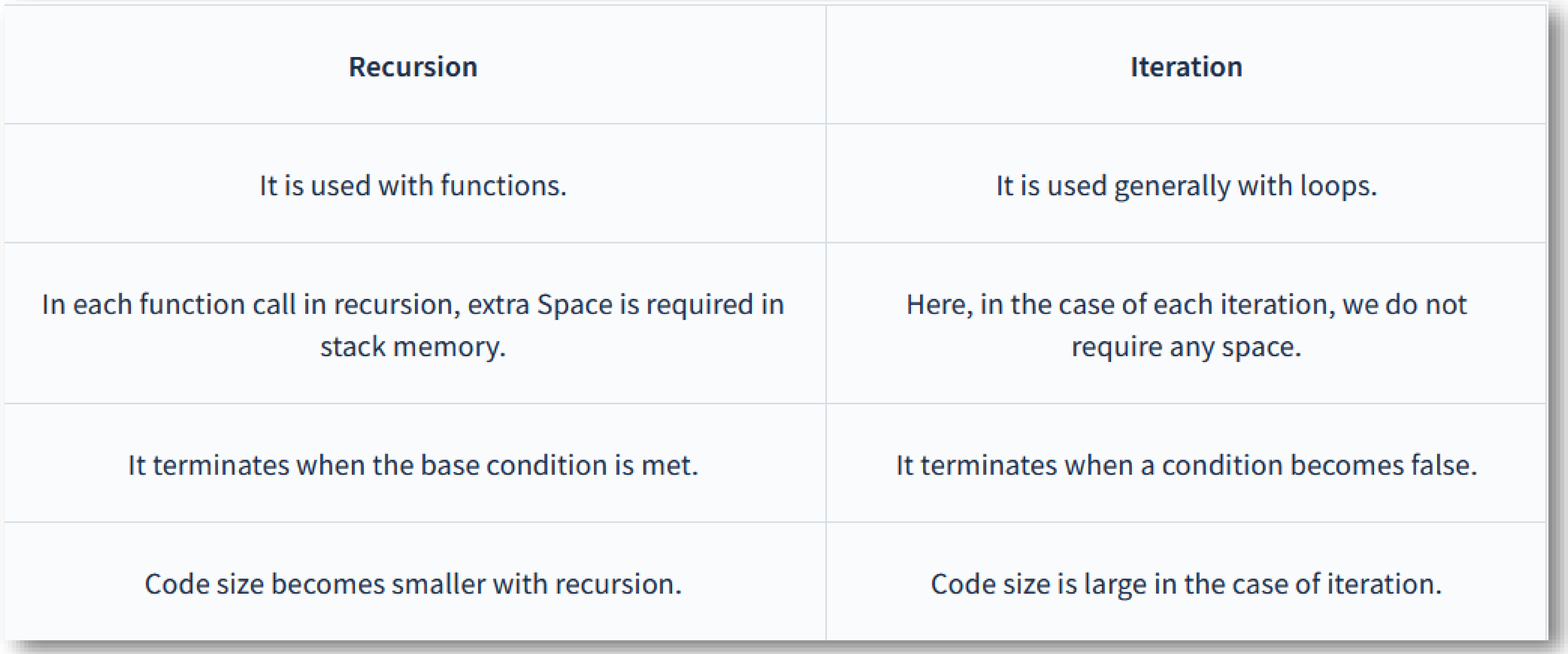
Indirect Recursion

Indirect recursion in C occurs when a function calls another function and if this function calls the first function again. Such functions are also called indirect recursive functions.
Following is the structure of indirect recursion.

```
function_01()
{
    //some code
    function_02();
}
function_02()
{
    //some code
    function_01();
}
```

In the indirect recursion structure the function_01() executes and calls function_02(). After calling now, function_02 executes where inside it there is a call for function_01, which is the first calling function.

Difference between Recursion & Iteration



Recursion	Iteration
It is used with functions.	It is used generally with loops.
In each function call in recursion, extra Space is required in stack memory.	Here, in the case of each iteration, we do not require any space.
It terminates when the base condition is met.	It terminates when a condition becomes false.
Code size becomes smaller with recursion.	Code size is large in the case of iteration.

Advantages of Recursion

- ✓ The code becomes shorter and reduces the unnecessary calling to functions.
- ✓ Useful for solving formula-based problems and complex algorithms.
- ✓ Useful in Graph and Tree traversal as they are inherently recursive.
- ✓ Recursion helps to divide the problem into sub-problems and then solve them, essentially divide and conquer.

Disadvantages of Recursion

- ❑ The code becomes hard to understand and analyze.
- ❑ A lot of memory is used to hold the copies of recursive functions in the memory.
- ❑ Time and Space complexity is increased.
- ❑ Recursion is generally slower than iteration.

Conclusion

- There are two types of recursion in the C language.
- The first is Direct recursion and Indirect recursion.
- The Direct recursion in C occurs when a function calls itself directly from inside.
- Indirect recursion occurs when a function calls another function, and then that function calls the first function again.
- The function call to itself is a recursive call, and the function will become a recursive function.
- The stack is maintained in the memory to store the recursive calls and all the variables with the value passed in them.