LEARN DSA WITH C++

WEEK :: 05

LEARN DSA WITH C++

PDF

COURSE INSTRUCTOR BY
ROHIT NEGI

MADE BY-
PRADUM SINGHA

Check Profile

My profile

LEARN DSA WITH C++

WEEK :: 05 **DAY: 01** **DATE: 15-05-2023**

STRING (KMP ALGORITHM)

#Prefix and Suffix:-

a	b	a	b

Prefix : a, ab, aba;
Suffix : b, ab, bab; Output = 2;

a	a	a	a

Prefix : a, aa, aaa;
Suffix : a, aa, aaa; Output : 3

#Solve

a	b	a	b

Prefix pointer : a [Prefix always start from first element of array]
Suffix pointer : a [Suffix start from second a because Prefix start a]

Then one Increment
Prefix pointer : b
Suffix pointer : b loop break [no Element]

Output : 2

#Example :: 02:-

a	b	c	a	b	d	a	b	c	a

Pointer

Prefix	Suffix	
a	a	Prefix start first element; and Suffix start second a.
b	b	increment
c	d	Not same [wrong]
Agine		
a	a	Now Suffix start Third a
b	b	
c	c	
d	d	loop break Output: 4 Time Complexity : O(N^2) N=size of array

Optimization Code :: -

a	**b**	**c**	**d**	**e**	**a**	**b**

Prefix =a **So 0 0 0 0 0 1 1** **0 = Not same**

Suffix=a OutPut = 2

Exp: 2.1

a b c d a b c e h a b c d a b

Prefix = a **0 0 0 0 1 1 1 0 0 1 1 1 1 1 1**

Suffix = a OutPut = 6

Exp : 2.2

ind	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18
str	a	b	c	d	a	b	c	e	a	b	c	d	a	b	c	d	a	b
LPA	0	0	0	0	1	2	3	0	1	2	3	4	5	6	7	4	5	6

Index start = 1; LPA = write index, when first and second pointer same str.

Pointer

First	Second
1	5
2	6
3	7
Not same	
1	9
2	10
3	11
4	12
5	13
6	14
7	15
Not Same	
4	16
5	17
6	18

#Longest Prefix Suffix >> GeeksforGeek <<

```
class Solution{
  public:
        int lps(string s) {
          // Your code goes here
          int index[s.size()+1];
          char str[s.size()+1];

          for(int i=0; i<s.size(); i++)
          {
              str[i+1]+s[i];
              index[i]=0;
          };

          index[s.size()]=0;

          int first =0, second =2;
          while(second<=s.size())
          {
              if(str[first+1]==str[second])
              {
                  index[second]= first+1;
                  first++, second++;
              }
              else
              {
                  if(first==0)
                  second++;
                  else
                  first = index[first];
              }
          }
          return index[s.size()];
        }
```

#Check if string is rotated by two places >> GeeksforGeek <<

str1 : a m a z o n

str2 : o n a m a z

ind	1	2	3	4	5	6	7	8	9	10	11	12	13
str	a	m	a	z	o	n	$	o	n	a	m	a	z
LPA	0	0	1	0	0	0	0	0	0	1	2	3	4

LPA = 4; Rotation = size of str (6) - LPA (4) = 2

#Convert palindrome ::

Str : a a c e c a a a ; First pointer = first element , second pointer = last element;

First	second
a	a
a	a
c	a

Not match , So, add first from last element

Str : a a a c e c a a a; first pointer = second elem, second pointer = second last elem

first and last elem match

First	second
a	a
a	a
c	c
e	e

Its palindrome

#Exp : 2:: a b c

Str : a b c

First second

a c Not match

Str: c a b c add first position from last elem

a b Not match

Str : c b a b c add second position from second last elm

:: a b c biggest Palindrome=a ; add other elm in front b c = c b a b c = 3 - 1 = 2

:: r o o r t b biggest Palindrome = r o o r ; add other elm in front t b = b t r o o r b t = 6 - 4 = 2

#Minimum characters to be added at front to make string palindrome >> GeeksforGeek <<

Str : r o o r t b

ind	1	2	3	4	5	6	7	8	9	10	11	12	13
str	r	o	o	r	t	b	$	b	t	r	o	o	r
LPA	0	0	0	1	0	0	0	0	0	1	2	3	4

Biggest palindrome = 4;

Size of String = 6 Minimum character add = 6-4 = 2

LEARN DSA WITH C++

WEEK :: 05 **DAY: 02** **DATE: 16-05-2023**

POINTER IN C++

#Memory Store ::

Int num = 5;

0	0	0	32 bit, 4 Byte	0	0	0	1	0	1

Read only 32 bit because data int type

Char name = ‘ a ‘;

a convert binary = 1 1 0 0 0 0 1   ascii value = 97

0	1	1	0	0	0	0	1

Read only 8 bit because data char type

#Address :

Int *p;

P is a pointer which is pointed int type value.
Every data defines an address in memory.

Address symbol table ::

Variable name	address
Num	200
Name	100

Int *p;
Int num = 5;
P = & num;

P is define address of num variable ; & use to define address ;
P directly changes the address of any value.

Int *p;
Int num = 5;
P = & num;

p	101	102	103	104	105
value			Int num = 5;		

*p = 20 change value address 103

p	101	102	103	104	105
value			Int num = 20;		

#Code ::

Change value

```
#include<iostream>
using namespace std;

int main()
{
    int num = 30;
    cout<<num<<endl;
    int *p;
    p = &num;

    *p = 20;
    cout<<num;

return 0;
};
```

Char *p :: p is a pointer which is define char type value;

4 GB Ram :: 2 ^32 byte

4 Gb ram define address only 32 byte

#Address of Array ::

Int arr[5] = { 10, 20, 30, 40, 50};

	10	20	30	40	50	
Array address =	500	504	508	512	516	every value 4 byte

Char arr[5] = ‘ a, b, c, d, e’;

	a	b	c	d	e	
Array address =	517	518	519	520	521	every char 1 byte

#Address define ::

Arr = Base address (500)

(Arr + 0) = 500 + 0*4

(Arr + 1) = 500 + 1*4

(Arr + 2) = 500 + 2*4

If i enter the inside of array so use pointer :: *(arr +3) = 500 + 3*4

Array already carry address so, don't need to add & when : p = arr; don’t p = & arr;

```
#include<iostream>
using namespace std;

int main()
{
    int arr[5] = {10, 20, 30, 40, 50};
    int *p;
    p=arr;
    cout<<*(p+0)<<endl;
    cout<<*(p+1)<<endl;
    cout<<*(p+2)<<endl;
    cout<<*(p+3)<<endl;
    cout<<*(p+4)<<endl;
return 0;
};
```

```
#include<iostream>
using namespace std;

int main()
{
    int arr[5] = {10, 20, 30, 40, 50};
    int *p;
    p=arr;
    for(int i=0; i<5; i++)
    {
        cout<<*p<<endl;    //*arr not allow
        p++;               //arr++
    }
return 0;
};
```

P++	Arr++
It is acceptable, p is pointed to the address of value. P is allowed to increase and decrease but don't allow it to be out of the range. P jump on value depends on data type of value.	It is not acceptable but it is also a pointed address of value. Arr can’t allow to change

#address Change ::

Int num1 = 5, num2 = 10;
Int *p , *q;
P = & num1;
q = & num2;

Pointer	p	q
Address	100	200
Value	5	10

Now p = q;

Pointer	p	q
Address	200	200
Value	10	10

#Call by pointer (Address) ::

Call by Pointer (Address)

```
#include<iostream>
using namespace std;

void fun(int *c, int *d)
{
    *c = (*c)*2;
    *d = (*d)*2;
}
int main()
{
    int a=12, b= 6;
    fun(&a, &b);     //call by address(pointer)
    cout<<a<<" "<<b;

return 0;
};
```

Call By Reference

```
#include<iostream>
using namespace std;

void fun(int &c, int &d)
{
    c = c*2;
    d = d*2;
}
int main()
{
    int a = 500, b = 1000;
    fun(a, b);      // call by reference
    cout<<a<<" "<<b;
return 0;
};
```

Value SWAP

```
#include<iostream>
using namespace std;

void swap(int &c, int &d)
{
    int temp = c;
    c = d;
    d = temp;
}
int main()
{
    int a=5 , b=8;
    swap(a,b);
    cout<<a<<" "<<b;

return 0;
};
```

Array Reverse using Pointer

```
#include<iostream>
using namespace std;

void reverse(int *arr, int size)
{
    int start = 0, end = size -1;
    while(start<end)
    {
        int temp = arr[start];
        arr[start] = arr[end];
        arr[end] = temp;
        start++, end--;
    }
}
int main()
{
    int a[5] = {10, 20, 30, 40, 50};
    reverse(a,5);
    for(int i=0; i<5; i++)
    cout<<a[i]<<" ";
return 0;
};
```

Address Using pointer

```cpp
#include<iostream>
using namespace std;

int main()
{
    int a = 100;
    int *p = &a;

    cout<<a<<endl;   // a value
    cout<<&a<<endl;  // a address
    cout<<p<<endl;   // a address
    cout<<*p<<endl;  // a value
    cout<<&p<<endl;  // p address

return 0;
};
```

WEEK :: 05 **DAY: 03** **DATE: 17-05-2023**

POINTER IN ADVANCE

#Reference Variable ::

Int i = 20;
Int & j = i;
i and j pointed to the same value 20.

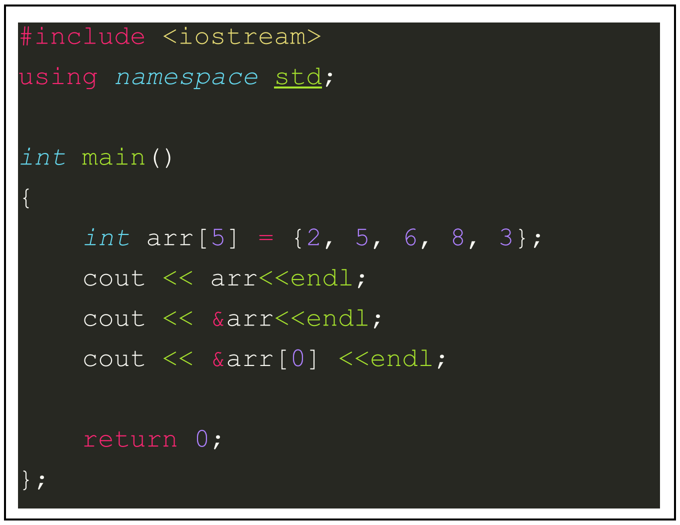
Array value Double using pointer :::

```
#include <iostream>
using namespace std;

// a[] and *a both are pointer
void Double_value(int a[], int size)
{
    for (int i = 0; i < size; i++)
    {
        *a = (*a) * 2;
        a++;
    }
}
int main()
{
    int arr[5] = {2, 5, 6, 8, 3};
    Double_value(arr, 5);
    for (int i = 0; i < 5; i++)
        cout << arr[i] << " ";

    return 0;
};
```

a[ ] = * a is same
p[0] = *(p +0)

arr start address =	100
cout<<arr;	100
cout<<&arr;	100
cout<<&arr[0];	100

#cout<<arr :: address of first element in arr.
#cout<<&arr :: arr where it is stored in memory location.
#cout<<&arr[0] :: address of 0 index position element



```
#include <iostream>
using namespace std;

int main()
{
    int arr[5] = {2, 5, 6, 8, 3};
    cout << arr<<endl;
    cout << &arr<<endl;
    cout << &arr[0] <<endl;

    return 0;
};
```

# 2 D ARRAY ::

Arr :: store address of first row
Arr + 1 :: store address of second row
Arr + 2 :: store address of third row
Arr + n :: store address of n row
&arr :: what is the address of array in memory location
***arr** :: address of the first element of the first row.
****arr** :: value of the first element of the first row
arr[0] = *(arr + 0) :: address of first element in first row
&(*(arr+0)) = *(&(arr+0)) :: Address of first row

HOME ADDRESS

100	104	112
116	120	124
128	132	136

arr = 100;
*(arr) = 100;
&arr = 100;
((*arr) + 1) = 104;
(arr + 1) = 116;
*(arr + 1) = 116;
*((arr + 1) + 2)= 124

## Arr[i][j] = *(*(Arr + i) + j)
[Define i and j]

# Address & Pointer ::

Int num = 10;	There is one num, it's type int and store value 10.
Int *P = &num;	There is one P, it is a pointer which is pointed to int type value.
Int ** X = &p	There is X, it is a pointer which is pointed to an int type pointer.
Int ***T = &X	T is a pointer which is pointed to another pointer, it also pointed to another pointer and finally it is pointed to int value.

2 D ARRAY Using POINTER

```
#include <iostream>
using namespace std;

void fun(int a[][4],int row, int col)
{
    int count=0;
    for(int i=0; i<row; i++)
    {
        for(int j=0; j<col; j++)
        a[i][j] = count++;
    }
}
int main()
{
    int arr[3][4];
    fun(arr,3,4);
    for(int i=0; i<3; i++)
    {
        for(int j=0; j<4; j++)
        cout<<arr[i][j]<<" ";
        cout<<endl;
    }
    return 0;
};
```

MEMORY

HEAP
Stack
Global/Local variable
code

#Stack :: Stack memory is a memory usage mechanism that allows the system memory to be used as temporary data storage that behaves as a first-in-last-out buffer.

#HEAP :: A memory heap is a location in memory where memory may be allocated at random access.

HEAP Memory >> STACK Memory

#How to take Memory from HEAP ::

int *P = new int ; ⇒ It is return Address and store in Pointer || int *P => store in stack
Int * P = new int [10]; ⇒ It is return Address and store in pointer || new int => store in heap

```
#include<iostream>
using namespace std;

int main()
{
    /*int *p = new int;
    *p = 10;
    cout<<*p<<endl;
    cout<<"address: "<<p;*/

     // for array
    int *p = new int[10];
    for(int i=0; i<10; i++)
    p[i]= i*2;

    for(int i=0; i<10; i++)
    cout<<p[i]<<" ";

    return 0;

return 0;
};
```

# DELETE Memory ::

Stack are automatically deleted or Heap memory delete manually

Int *p = new int; ⇒ allocate memory in heap delete address using delete P.
Int *p = p[ ] ⇒ delete address using delete p[ ].

LEARN DSA WITH C++

WEEK :: 05 **DAY: 04** **DATE: 18-05-2023**

2D Array with POINTER + Recursion

#2D Array :

Define : const int col = 5;

```
Void Fun (int arr[ ] [col], Row)
 {
              }
int arr[m]
int arr[row][col]
```

m, row & col = always constant.

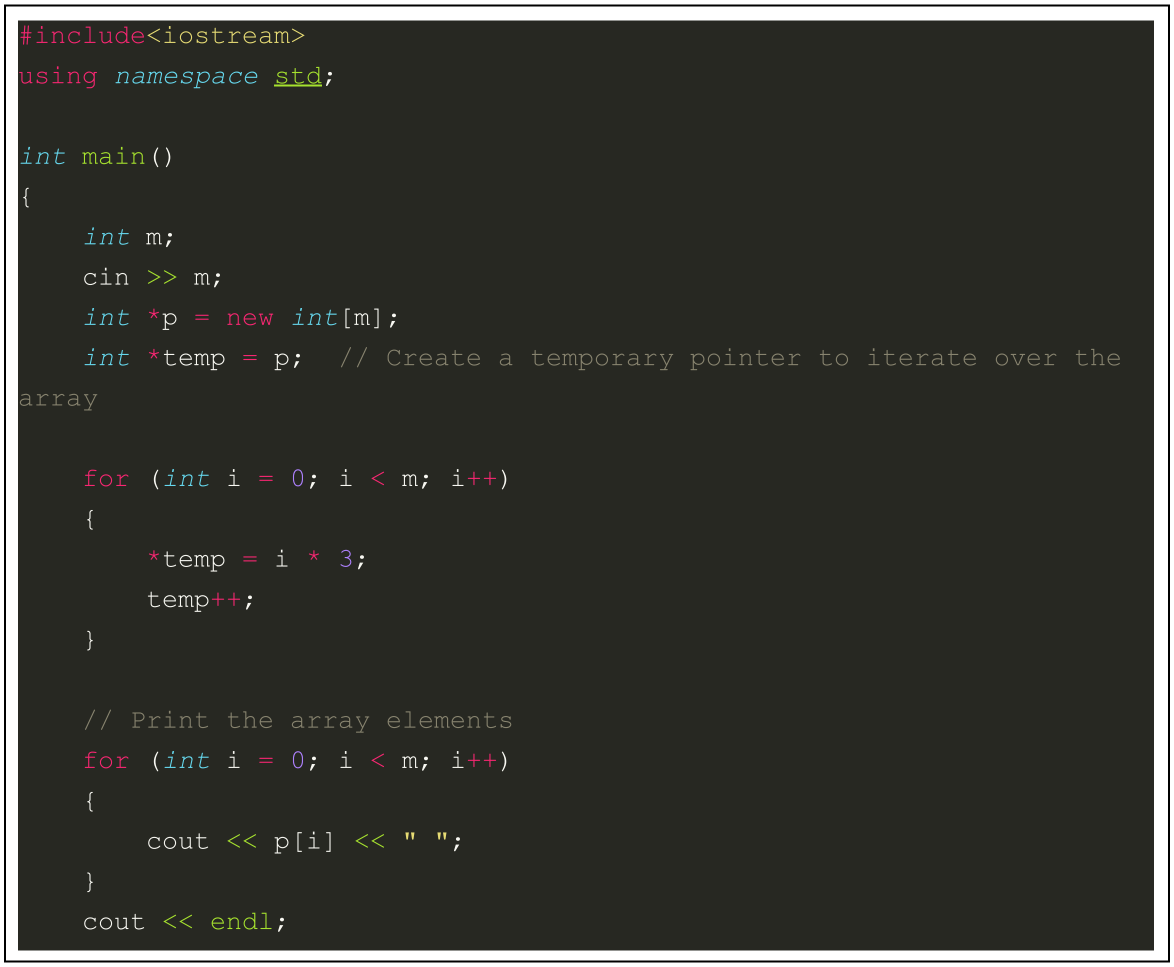
#Take Input ::

vector<int>v; store in heap automatic delete.

int *p = new int[m] variable(m) can take

Take input 2D Array

```
int **p = new int *[10]
for(int i=0; i<10; i++)
P[i] = new int [5];
```



```
#include<iostream>
using namespace std;

int main()
{
    int m;
    cin >> m;
    int *p = new int[m];
    int *temp = p;   // Create a temporary pointer to iterate over the array

    for (int i = 0; i < m; i++)
    {
        *temp = i * 3;
        temp++;
    }

    // Print the array elements
    for (int i = 0; i < m; i++)
    {
        cout << p[i] << " ";
    }
    cout << endl;
```

```cpp
    delete[] p;   // Free the dynamically allocated memory

    return 0;
}
```

1 D ARRAY USING FUNCTION

```cpp
#include <iostream>
using namespace std;

void fun(int* a, int size)
{
    for (int i = 0; i < size; i++)
        a[i] = i;
}

int main()
{
    int m;
    cin >> m;
    int* p = new int[m];
    fun(p, m);

    for(int i=0; i<m; i++)
    cout<<i<<" ";
    delete[] p; // Deallocate memory

    return 0;
}
```

2D ARRAY USING FUNCTION

```cpp
#include<iostream>
using namespace std;

void fun(int **x, int row, int col)
{
    for(int i=0; i<row; i++)
    for(int j=0; j<col; j++)
    x[i][j] = i+j;
}
int main()
{
    int n, m;
    cin>>n>>m;
    int **p = new int *[n];
    for(int i=0; i<n; i++)
    p[i] = new int [m];

    fun(p, n, m);

    for(int i=0; i<n; i++)
    {
    for(int j=0; j<m; j++)
    cout<<p[i][j]<<" ";
    cout<<endl;
    }
return 0;
};
```

RECURSION

Recursion is the technique of making a function call itself.

# Recursion importance part :

1. Breaking Condition
2. Calling itself

Exp: 01 :: Factorial

Calculate Factorial Using Recursion ::

5!	5 * 4!	120
4!	4 * 3!	24
3!	3 * 2!	6
2!	2 * 1!	2
1!	1	1

## Code ::

```
Int Factorial (int n)
        {
            if(n == 1)
                Return 1;
            Int num = Factorial(n-1)
                Num = n * num;
                Return  num;
        }

        Int main ( )
```

```
#include<iostream>
using namespace std;


int factorial(int n)
{
    if(n==1)
    return 1;
```

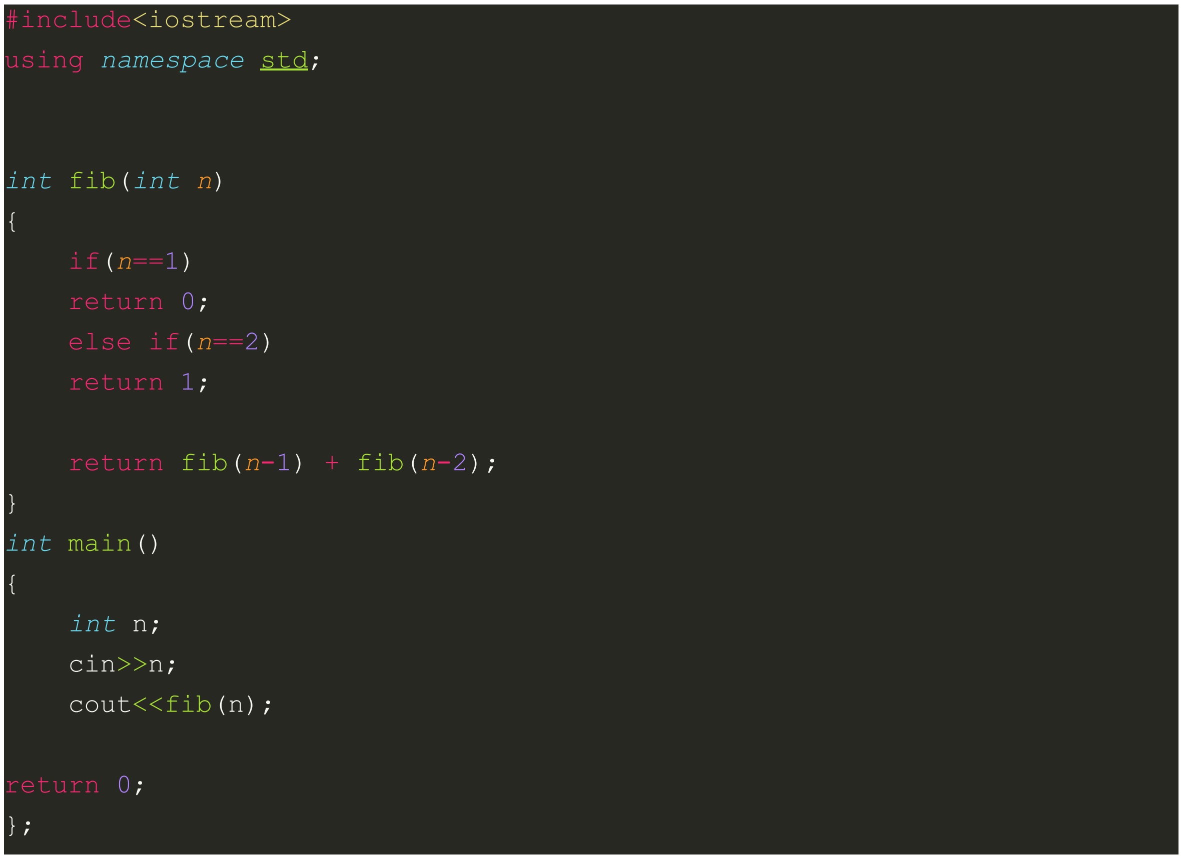
{
Int n;
cin>> n;
cout<<Factorial(n);
Return 0;
	`    int ans = factorial(n-1);`
	`    ans = ans* n;`
	`    return ans;`
	`}`
	`int main()`
	`{`
	`    int n;`
	`    cin>>n;`
	`    cout<<factorial(n);`
	
	`return 0;`
	`};`

#Power ::   3^n

f(n) = 3* f(n-1)
f(n-1) = 3* f(n-2)
	3^4 = 3* 3^3 = 81
	3^3 =3* 3^2 = 27
	3^2 =3* 3^1 =9
	3^1 = 3
	3^0 = 1

`#include<iostream>`
`using namespace std;`
 	
`int power(int n)`
`{`
`    if(n==1)`
`    return 3;`
 	
`    return 3*power(n-1);`
`}`
`int main()`
`{`
`    int n;`
`    cin>>n;`
`    cout<<power(n);`
 	
`return 0;`
`};`	

|| FIBONACCI ||



```
#include<iostream>
using namespace std;


int fib(int n)
{
    if(n==1)
    return 0;
    else if(n==2)
    return 1;

    return fib(n-1) + fib(n-2);
}
int main()
{
    int n;
    cin>>n;
    cout<<fib(n);

return 0;
};
```

LEARN DSA WITH C++

WEEK :: 05 **DAY: 05** **DATE: 19-05-2023**

RECURSION

Print n Natural Number Using RECURSION

```cpp
#include<iostream>
using namespace std;


void print_num(int start, int end)
{
    if(start>end)
    return;

    cout<<start<<" ";
    print_num(start+1, end);
}
int main()
{
    int n = 10;
    print_num(1, 10);

return 0;
};
```

Reach Top of the bottom take step at a time 1 and 2

Consider = You reach 5 th position of the top.
Take step = 1 + 1 + 1 + 1 + 1= 5
= 1 + 2 + 1 + 2 =6
= 1 + 2 + 1 + 2 =6 when we can't reach 5 **return 0;**
= 1 + 1 + 1 + 2 = 5 when we can reach 5 **return 1;**
And last
Return (jump (step1, n) + jump(step2, n))

## Print Sum of the Array Using Recursion ::

3	1	2	5	8
3	4	6	11	19

```
#include<iostream>
using namespace std;

void print_sum(int *a, int sum, int size)
{
    if(!size)
    return;
    sum += a[0];
    cout<<sum<<" ";
    print_sum(a+1, sum, size-1);
}
int main()
{
    int arr[5] = { 3, 1, 2, 5, 8};
    int sum = 0;
    print_sum(arr, sum, 6);

return 0;
};
```

Linear Search Using Recursion

```
#include<iostream>
using namespace std;

int search(int *a, int size, int key)
{
    if(size==0)
    return 0;

    if(a[0] == key)
    return 1;

    return search(a+1, size-1, key);
}
int main()
{
    int arr[6] = { 3, 1, 2, 5, 8};
    int key = 5;
    cout<<search(arr, 6, key);

return 0;
};
```

Print Double of Array value

```
#include<iostream>
using namespace std;

void Double_value(int *a, int size)
{
    if(!size)
    return;
    a[0] *= 2;
    Double_value(a+1, size-1);
}
int main()
{
    int arr[6] = { 3, 1, 2, 5, 8};
    Double_value(arr, 5);
    for(int i=0; i<5; i++)
    cout<<arr[i]<<" ";

return 0;
};
```

Binary Search Using Recursion

```
#include<iostream>
using namespace std;

int binary_search(int *a, int start, int end, int key)
{
    if(end<start)
    return 0;

    int mid = end+(start-end)/2;

    if(a[mid]== key)
    return 1;
    else if(a[mid]>key)
    return binary_search(a, start, mid-1, key);
    else
    return binary_search(a, mid+1, end, key);
}
int main()
{
    int arr[10] = { 2, 3, 4, 6, 7, 9, 10, 15, 16, 18};
    int key = 16;
    cout<<binary_search(arr, 0, 9, key);

return 0;
};
```

Print Index :: Binary Search Using Recursion

```
#include<iostream>
using namespace std;

void binary_search(int *a, int start, int end, int key, int &index)
{
    if(end<start)
    return;

    int mid = end+(start-end)/2;

    if(a[mid]== key)
    {
        index = mid;
        return ;
    }
    else if(a[mid]>key)
    return binary_search(a, start, mid-1, key, index);
    else
    return binary_search(a, mid+1, end, key, index);
}
int main()
{
    int arr[10] = { 2, 3, 4, 6, 7, 9, 10, 15, 16, 18};
    int key = 16;
    int index = -1;
    binary_search(arr, 0, 9, key, index);
    cout<<index;

return 0;
};
```