Data Structures and Algorithms

Pointers
Arrays
Recursive functions

Pointers

- For a C++ program, the memory of a computer is like a succession of memory cells, each one byte in size, and each with a unique address.
- These single-byte memory cells are ordered in a way that allows data representations larger than one byte to occupy memory cells that have consecutive addresses.
- This way, each cell can be easily located in the memory by means of its unique address.
- When a variable is declared, the memory needed to store its value is assigned a specific location in memory (its memory address).
- Generally, C++ programs do not actively decide the exact memory addresses where its variables are stored. Fortunately, that task is left to the environme where the program is run - generally, an operating system that decides the particular memory locations on runtime.
- However, it may be useful for a program to be able to obtain the address c a variable during runtime in order to access data cells that are at a certain position relative to it.

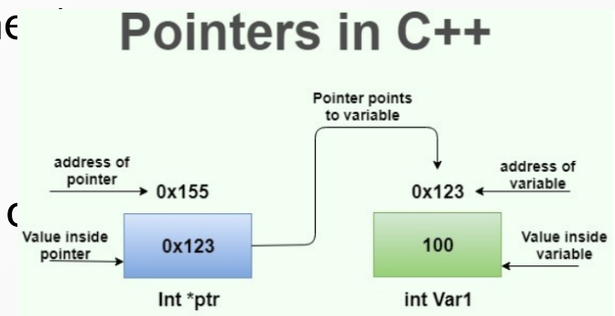
işgal etmek

ardışık

Memory Address	Variable in memory
1000	1003
1001	
1002	
1003	5
1004	

*p

i



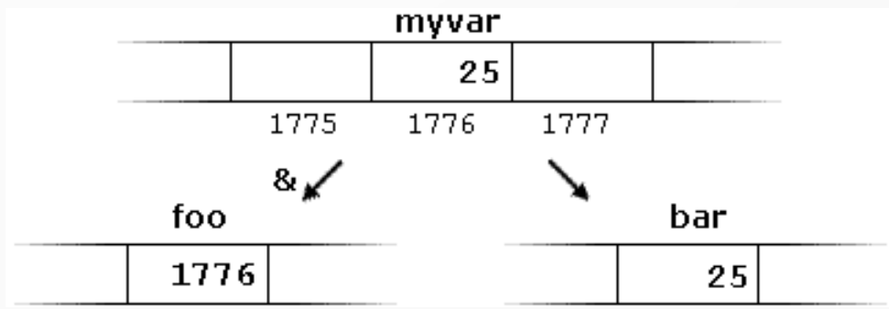
https://www.cplusplus.com/doc/tutorial/pointers/

Pointers: Address-of operator (&)

önceki

- The address of a variable can be obtained by preceding the name of a variable with an ampersand sign (&), known as address-of operator. For example:
- foo = **&myvar**;
- This would assign the address of variable myvar to foo.
- The actual address of a variable in memory cannot be known before runtime, but let's assume, in order to help clarify some concepts, that myvar is placed during runtime in the memory address 1776.
- In this case, consider the following code fragment:

  - char myvar = 25;
  - char * foo = &myvar;
  - char bar = myvar;



Pointers: Address-of operator (&)

```
#include <iostream>
using namespace std;
int main()
{
    // declare variables
    int var1 = 3;
    int var2 = 24;
    int var3 = 17;

    // print address of var1
    cout << "Address of var1: "<< &var1 << endl;

    // print address of var2
    cout << "Address of var2: " << &var2 << endl;

    // print address of var3
    cout << "Address of var3: " << &var3 << endl;
    return 0;
}
```

```
Address of var1: 0x62ff1c
Address of var2: 0x62ff18
Address of var3: 0x62ff14
```

The 0x before the address indicates the address is in hexadecimal form.

Dereference operator: *

- **Pointers in c/c++** are **variables** that points to a **specific address** in the **memory** occupied by another variable.
- Pointers are used to store addresses rather than values.
- Here is how we can declare pointers.

```
int* pointVar;
int var = 5;
// assign address of var to pointVar
pointer
pointVar = &var;
```

- Here, 5 is assigned to the variable var. And, the address of var is assigned to the pointVar pointer with the code pointVar = &var

https://simplesnippets.tech/cpp-pointers-concept-with-example/
https://www.programiz.com/cpp-programming/pointers

Example

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

int main () {
int var = 5;

// declare pointer variable
int *pointVar;

// store address of var
pointVar = &var;

// print value of var
cout << "var = " << var << endl;

// print address of var
cout << "Address of var (&var) = " << &var << endl << endl;

// print pointer pointVar
cout << "pointVar = " << pointVar << endl;

// print the content of the address pointVar points to
cout << "Content of the address pointed to by pointVar
(*pointVar) = "
<< *pointVar << endl;

return 0;
}
```

```
var = 5
Address of var (&var) = 0x62ff18

pointVar = 0x62ff18
Content of the address pointed to by pointVar
(*pointVar) = 5
```

inside the point var there is a address so that it goes to that address and find the value inside

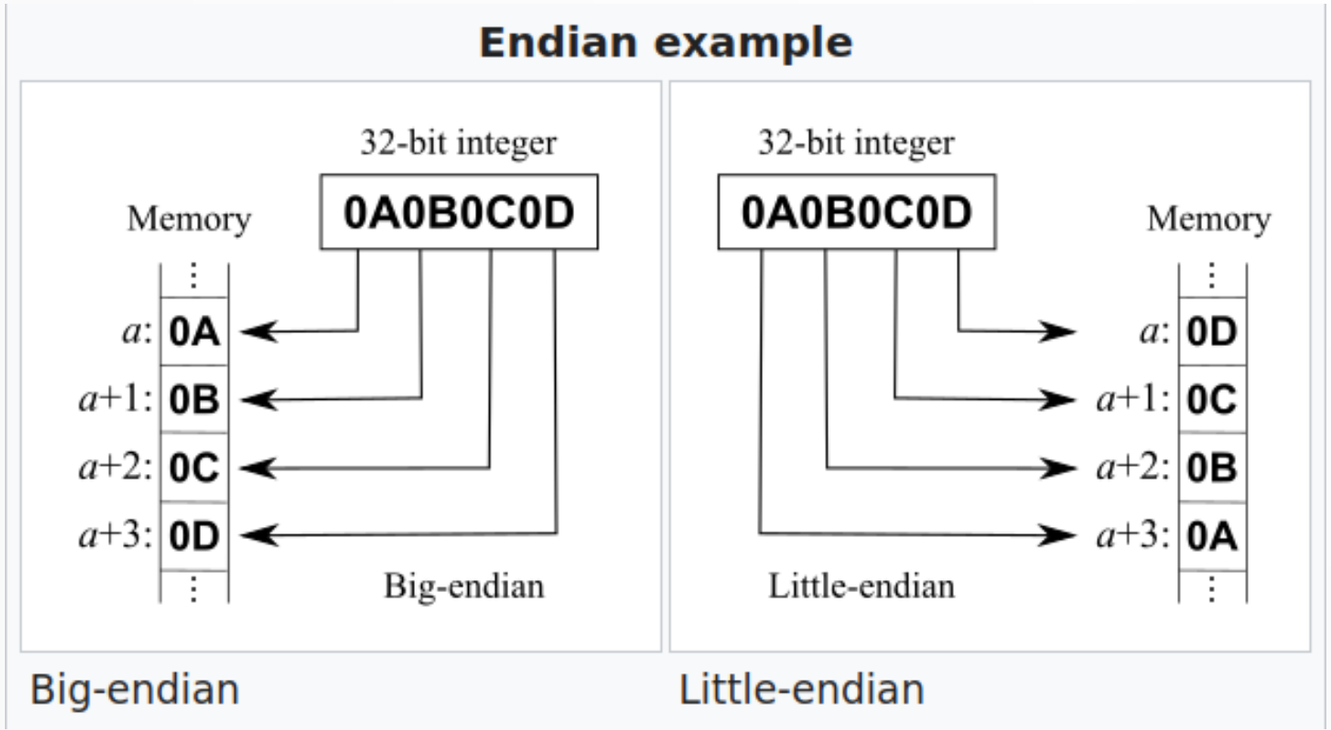
Example

```
int var = 5;
int* pointVar;
// store address of var
pointVar = &var;

// print var
cout << "var = " << var << endl;

// print *pointVar
cout << "*pointVar = " << *pointVar << endl
     << endl;

cout << "Changing value of var to 7:" << endl;

// change value of var to 7
var = 7;

// print var
cout << "var = " << var << endl;

// print *pointVar
cout << "*pointVar = " << *pointVar << endl
     << endl;

cout << "Changing value of *pointVar to 16:" << endl;

// change value of var to 16
*pointVar = 16;

// print var
cout << "var = " << var << endl;
// print *pointVar
cout << "*pointVar = " << *pointVar << endl;
```

```
var = 5
*pointVar = 5

Changing value of var to 7:
var = 7
*pointVar = 7

Changing value of *pointVar to 16:
var = 16
*pointVar = 16
```

https://www.programiz.com/cpp-programming/pointers

Representation of integers in memory



- In computing, endianness is the order or sequence of bytes of a word of digital data in computer memory.
- Endianness is primarily expressed as big-endian (BE) or little-endian (LE). A big-endian system stores the most significant byte of a word at the smallest memory address and the least significant byte at the largest.
- A little-endian system, in contrast, stores the least-significant byte at the smallest address

https://en.wikipedia.org/wiki/Endianness

Arrays

- An array is a variable that can store multiple values of the same type.

C++ Array Declaration on the stack memory region:

```
dataType arrayName[arraySize];
```

- **For example:**

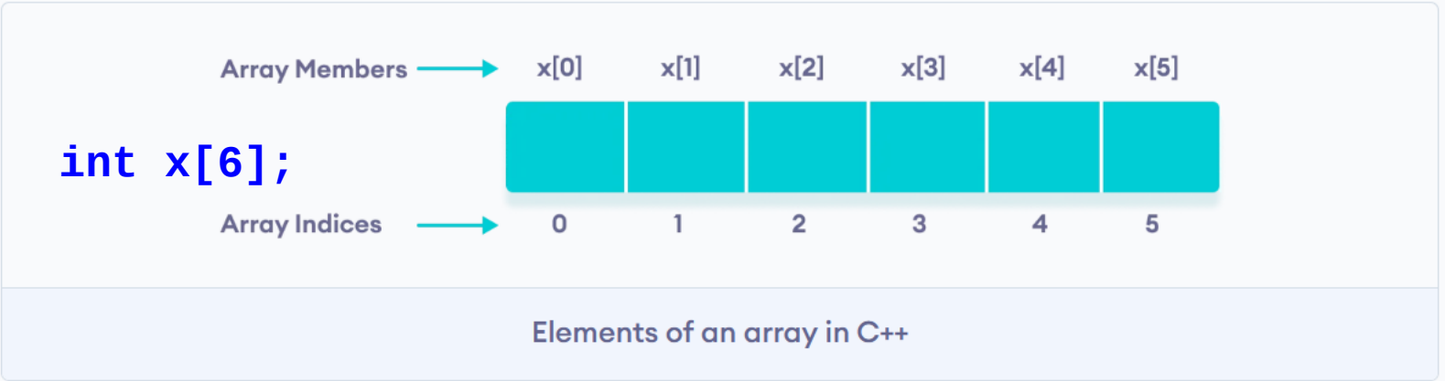
```
int x[6];
```

- **Here,**

```
    int - type of element to be stored
    x - name of the array
    6 - size of the array
char Y[11];
double Z[11];
```

https://www.programiz.com/cpp-programming/arrays

Arrays



Elements of an array in C++

Few Things to Remember:

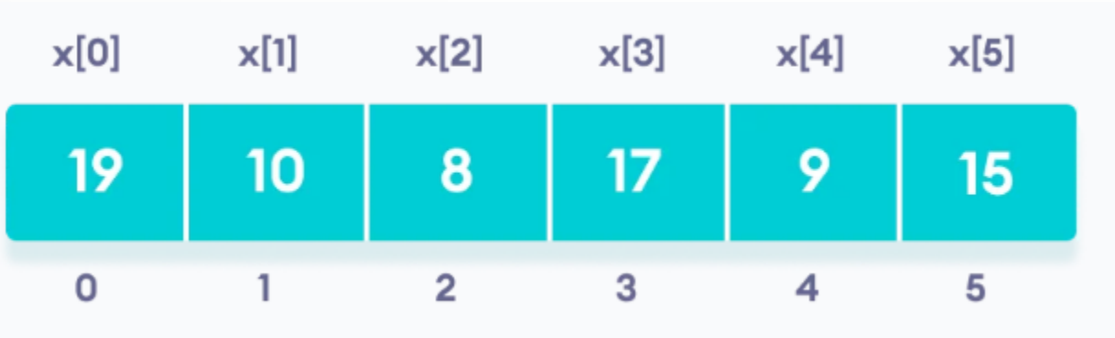
- The array indices start with 0. Meaning x[0] is the first element stored at index 0.
- If the size of an array is n, the last element is stored at index (n-1). In this example, x[5] is the last element.
- Elements of an array have consecutive addresses. For example, suppose the starting address of x[0] is 2120d. Then, the address of the next element x[1] will be 2124d, the address of x[2] will be 2128d and so on.
- Here, the size of each element is increased by 4. This is because the size of int is 4 bytes.

ardışık

https://www.programiz.com/cpp-programming/arrays

Arrays

```
// declare and initialize and array
int x[6] = {19, 10, 8, 17, 9, 15};
```



Another method to initialize array during declaration:

```
// declare and initialize an
array
int x[] = {19, 10, 8, 17, 9,
15};
```

Here, we have not mentioned the size of the array. In such cases, the compiler automatically computes the size.

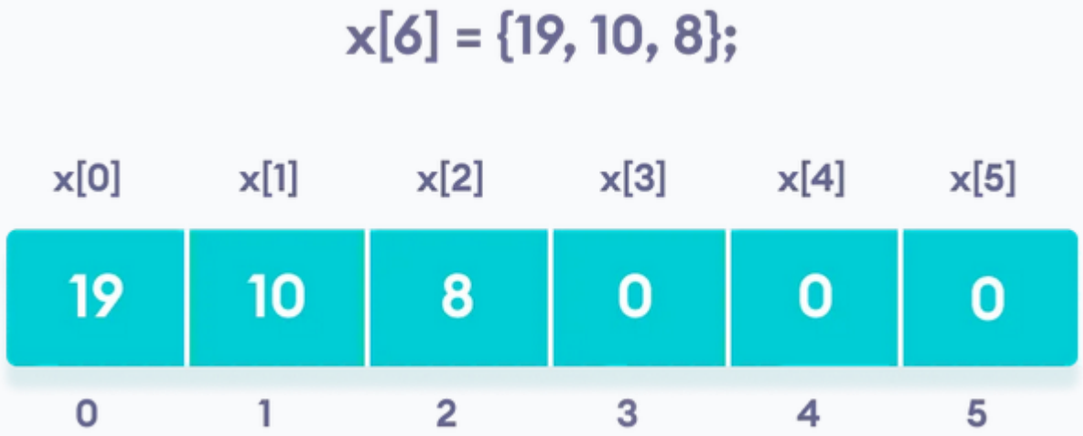
https://www.programiz.com/cpp-programming/arrays

Arrays

```
// store only 3 elements in the array
int x[6] = {19, 10, 8};
```

- Here, the array x has a size of 6. However, we have initialized it with only 3 elements.

- In such cases, there may be random values in the remaining places. Oftentimes, this random value is simply 0.



https://www.programiz.com/cpp-programming/arrays

Arrays

- How to assign and print array elements?

```
int mark[5] = {19, 10, 8, 17, 9};

// change 4th element to 9
mark[3] = 9;

// take input from the user
// store the value at third position
cin >> mark[2];



// take input from the user
// assign at ith position
int i=3;
cin >> mark[i-1];

// print first element of the array
cout << mark[0];

// print ith element of the array
cout << mark[i-1];
```

Example: Write a program that take Inputs from User and Store Them in an Array

```
int numbers[5];

cout << "Enter 5 numbers: " <<
endl;

//  store input from user to array
for (int i = 0; i < 5; ++i) {
    cin >> numbers[i];
}

cout << "The numbers are: ";

//  print array elements
for (int n = 0; n < 5; ++n) {
    cout << numbers[n] << "  ";
}
```

- Once again, we have used a for loop to iterate from i = 0 to i = 4. In each iteration, we took an input from the user and stored it in numbers[i].

- Then, we used another for loop to print all the array elements.

https://www.programiz.com/cpp-programming/arrays

Example: Display sum and average of array elements using for loop

```
int numbers[5] = { 7, 5, 6, 12, 35 };

cout << "The numbers are: ";

//  Printing array elements
// using range based for loop
for (const int &n : numbers) {
cout << n << "  ";
}

cout << "The numbers are: ";
// using range based for loop
for (int n : numbers) {
cout << n << "  ";
}

cout << "\nThe numbers are: ";

//  Printing array elements
// using traditional for loop
for (int i = 0; i < 5; ++i) {
cout << numbers[i] << "  ";
}
```

- In our range based loop, we have used the code const int &n instead of int n as the range declaration. However, the const int &n is more preferred because:
- Using int n simply copies the array elements to the variable n during each iteration. This is not memory-efficient.
- &n, however, uses the memory address of the array elements to access their data without copying them to a new variable. This is memory-efficient.
- We are simply printing the array elements, not modifying them. Therefore, we use const so as not to accidentally change the values of the array.

https://www.programiz.com/cpp-programming/arrays

Pointers and arrays

```
int numbers[5];
int *p;
p = numbers;
*p = 10;
p++;
*p = 20;
p = &numbers[2];
*p = 30;
p = numbers + 3;
*p = 40;
p = numbers;
*(p + 4) = 50;

for (int n = 0; n < 5; n++)
  cout << numbers[n] << ", ";
```

- Brackets ([]) were explained as specifying the index of an element of the array.
- In fact these brackets are a dereferencing operator known as offset operator.
- They dereference the variable they follow just as * does, but they also add the number between brackets to the address being dereferenced.

```
1 a[5] = 0;       // a [offset of 5] = 0
2 *(a+5) = 0;     // pointed to by (a+5) = 0
```

https://www.cplusplus.com/doc/tutorial/pointers/

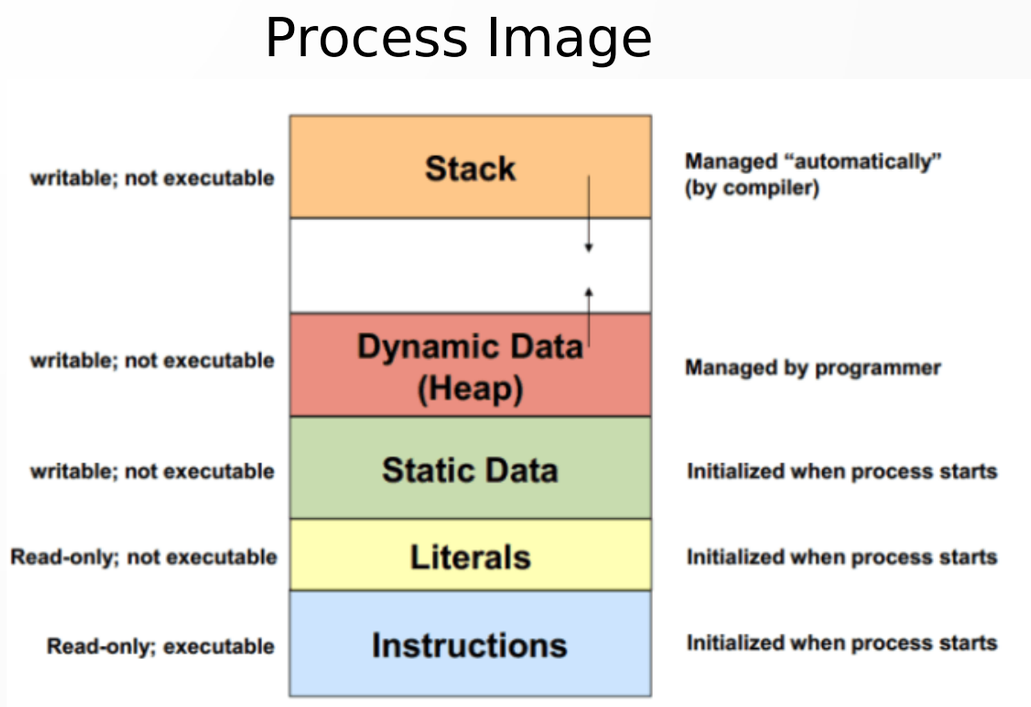
Allocating on array on the heap region

An array on the stack:

```
int A[N];
```

An array on the heap:

```
int *B=new int[N];
```



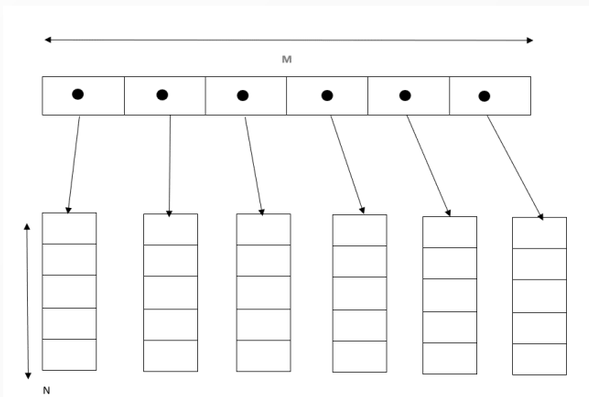
Two dimensional arrays

On the stack:

```
int B[N][M];
```

On the heap:

```
int** a = new int*[rowCount];
for(int i = 0; i < rowCount; ++i)
    a[i] = new int[colCount];
```

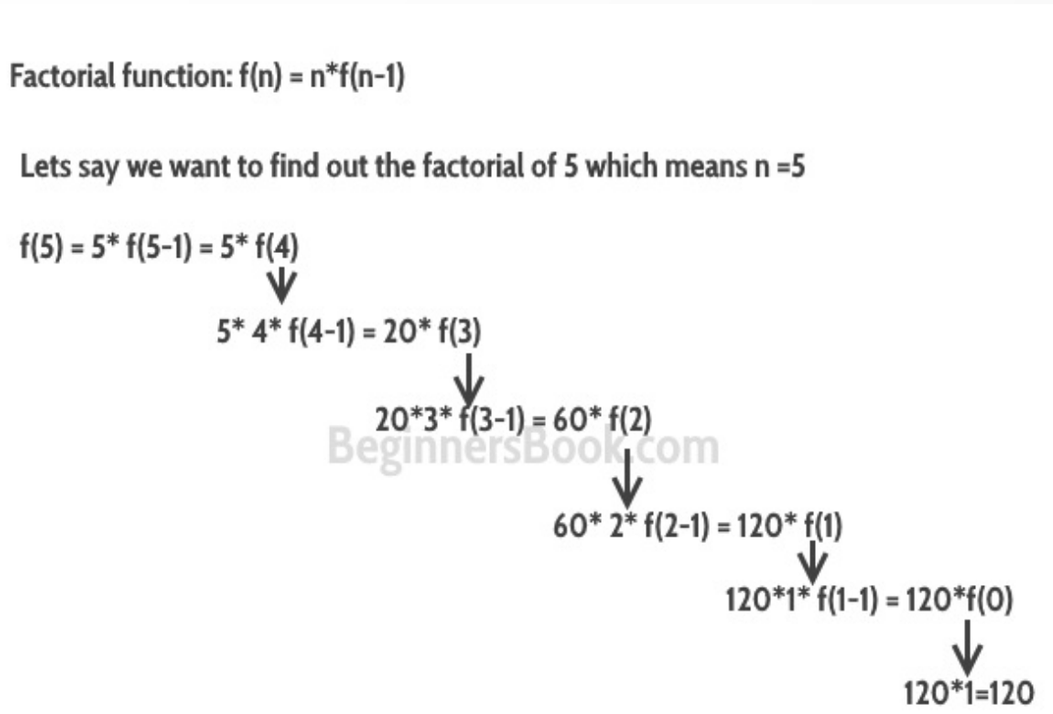


Recursive Functions

- The process in which a function calls itself is known as recursion and the corresponding function is called the recursive function. The popular example to understand the recursion factorial function.

- Factorial function: f(n) = n*f(n-1), base condition: if n<=1 then f(n) = 1. Don't worry wil discuss what is base condition and why it important.

- In the following diagram. I have shown that how the factorial function is calling itself until the function reaches to the base condition.



https://beginnersbook.com/2017/08/cpp-recursion/

Recursive Functions: Factorial

```
#include <iostream>
using namespace std;
//Factorial function
int f(int n){
   if (n <= 1) // base condition
        return 1;
   else
       return n*f(n-1);
}
int main(){
   int num;
   cout<<"Enter a number: ";
   cin>>num;
   cout<<"Factorial of entered number: "<<f(num);
   return 0;
}
```

- **Base condition**
- In the above program, you can see that I have provided a base condition in the recursive function. The condition is:

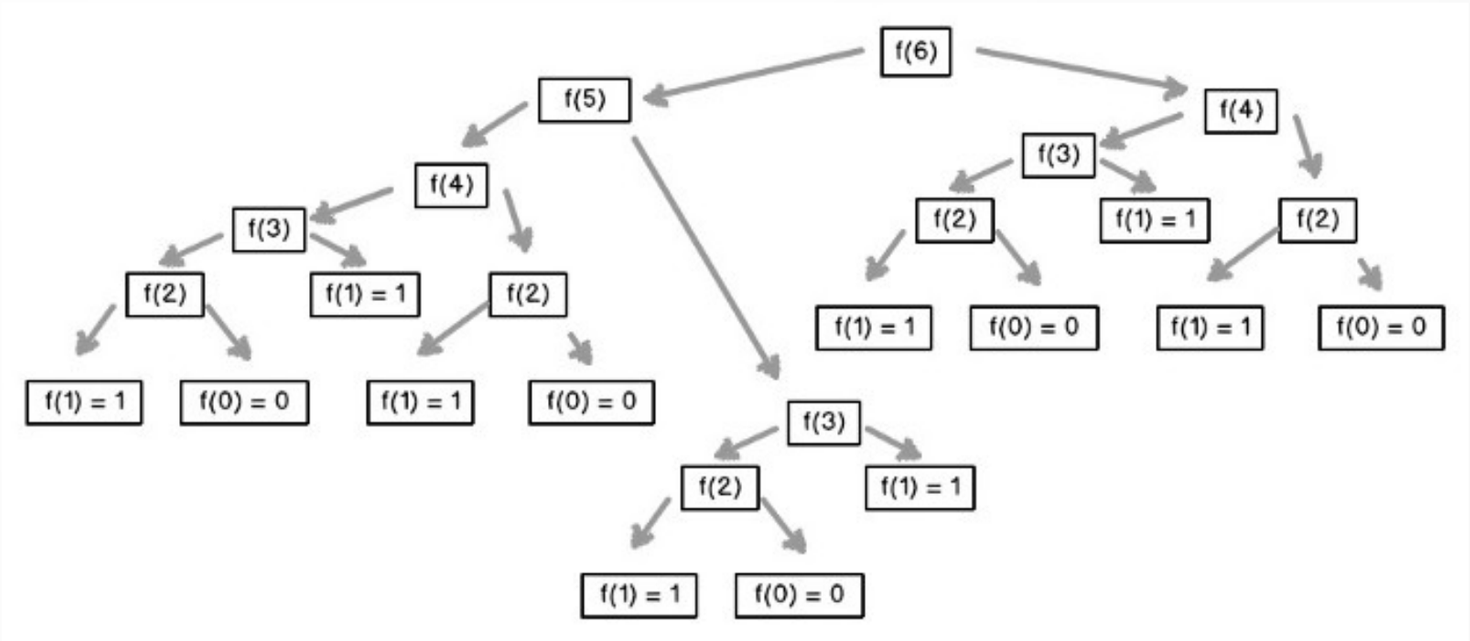
- if (n <= 1)

  return 1;

- The purpose of recursion is to divide the problem into smaller problems till the base condition is reached. For example in the above factorial program I am solving the factorial function f(n) by calling a smaller factorial function f(n-1), this happens repeatedly until the n value reaches base condition(f(1)=1). If you do not define the base condition in the recursive function then you will get stack overflow error.

Recursive Functions: Fibonacci

```
#include<iostream>
using namespace std;
int fibonacci(int);
int main(void)
{
    int num;
    cout<<"n=";cin>>num;
    cout<<fibonacci(num)<<endl;
    return 0;
}
int fibonacci(int num)
{
    //base condition
    if(num == 0 || num == 1)
    {
        return num;
    }
    else
    {
        // recursive call
        return fibonacci(num-1) + fibonacci(num-2);
    }
}
```



https://medium.com/launch-school/recursive-fibonnaci-method-explained-d82215c5498e

Recursive Functions

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

int fa(int n){
   if(n<=1)
      return 1;
   else
      return n*fb(n-1);
}

int fb(int n){
   if(n<=1)
      return 1;
   else
      return n*fa(n-1);
}

int main(){
   int num=5;
   cout<<fa(num);
   return 0;
}
```

- **Direct recursion vs indirect recursion**

- **Direct recursion:** When function calls itself, it is called direct recursion, the example we have seen above is a direct recursion example.

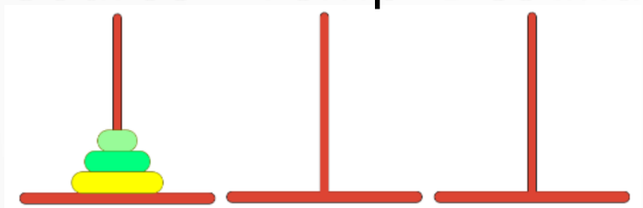
- **Indirect recursion:** When function calls another function and that function calls the calling function, then this is called indirect recursion. For example: function A calls function B and Function B calls function A.

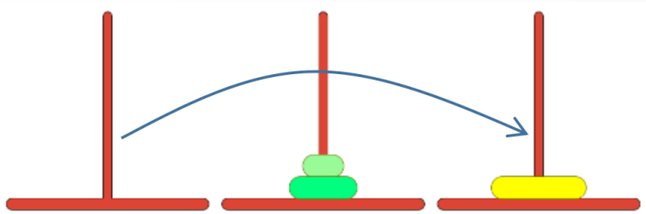
Recursive Functions: Binary Search

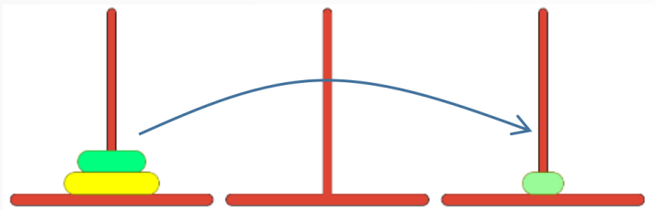
```
#include <iostream>
#include <cstdlib>
using namespace std;
int binarySearch(int A[],int first,int last,int x){
    cout<<"binarySearch(first="<<first<<",last="<<last<<")"<<endl;
if (first>last) return -1;
    int middle=(first+last)/2;
    if (x==A[middle]) //Element found
          return middle;//Index of the element
    else if (x <A[middle]){ //
        return binarySearch(A,first,middle-1,x);//smaller than middle
    }
    else{//(x >A[middle])
        return binarySearch(A,middle+1,last,x);//bigger than middle
    }
}

int main()
{
    int A[]={1,3,5,6,8,11,23,45,67,89,99,111,122,134};
    int N=sizeof(A)/sizeof(int);
int x=1;// seek for x in array
    int find=binarySearch(A,0,N-1,x);
if (find==-1){
  cout<<"Element couldn't be found."<<endl;
}
else{
  cout<<"Element found at "<<find<<endl;
  cout<<"A["<<find<<"]="<<A[find]<<endl;
}
    return 0;
}
```

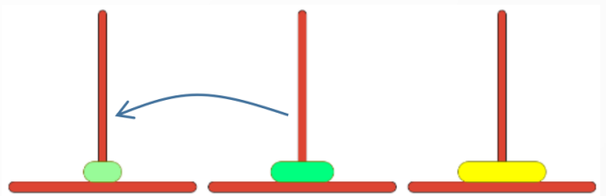
Recursive Functions: Hanoi towers

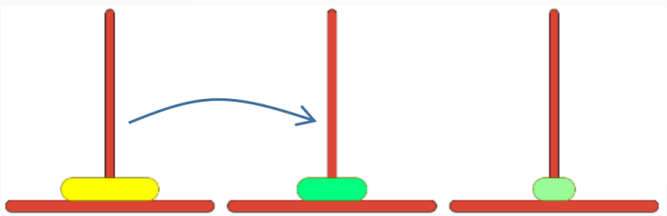
Source Temp Destination

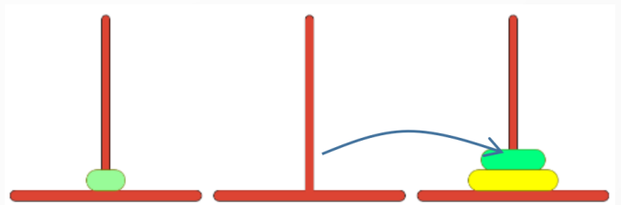


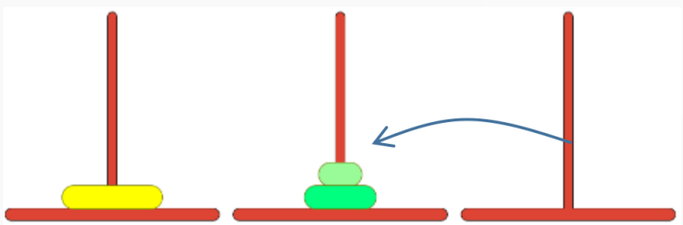


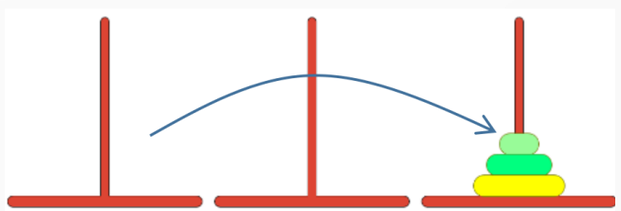












Recursive Functions: Hanoi towers

```
5  void hanoi(int nDisk,string Src,string Dst,string Tmp){
6          if (nDisk==1){
7                  cout<<"move "<<Src<<"->"<<Dst<<endl;
8          }
9          else{//nDisk>1
10 (1)            hanoi(nDisk-1, Src,Tmp,Dst);
11 (2)            hanoi(1, Src,Dst,Tmp);
12 (3)            hanoi(nDisk-1, Tmp,Dst,Src);
13         }
14 }
```

```
18 int main(){
19         hanoi(3, "Src","Dst","Tmp");
20         return 0;
21 }
```