Lecture 1: Introduction

Date: 08/22/2023

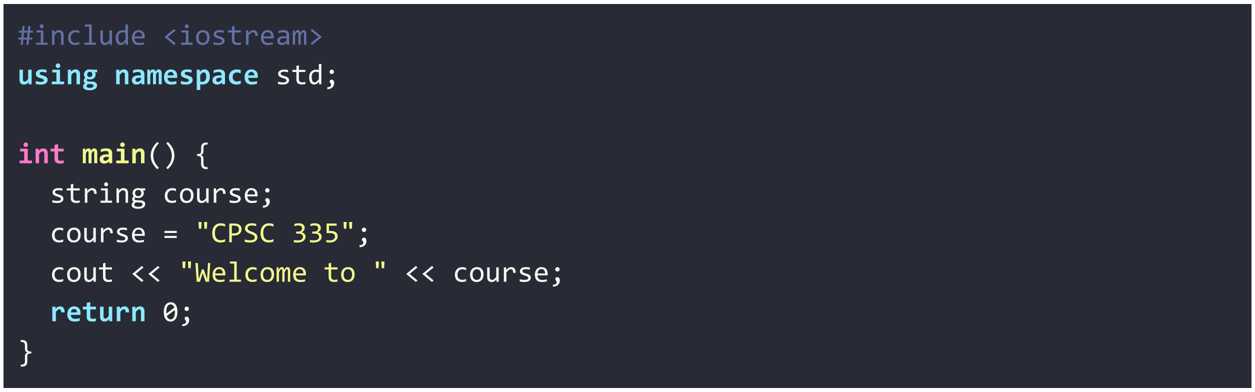
Brief Introduction to Algorithms

What is an Algorithm?

- “An algorithm is a set of instructions for accomplishing a task.” – **Grokking Algorithms** (page 1)
- “An algorithm is any well-defined computational procedure that takes some value, or set of values, as input and produces some value, or set of values, as output in a finite amount of time” – **Introduction to Algorithms** (page 28)
- “An algorithm is a process which, given an instance of a specific problem, produces A solution for that instance, and
    1. may be described clearly enough to be implemented (clarity),
    2. always produces a correct solution (correctness), and
    3. takes a finite amount of time (termination)” – **ADITA** (page 21)

Review of Programming Fundamentals

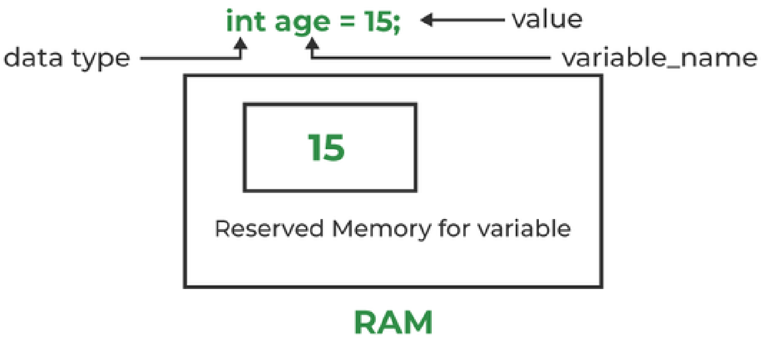
1_welcome.cpp



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

int main() {
  string course;
  course = "CPSC 335";
  cout << "Welcome to " << course;
  return 0;
}
```

⇒ Variables:

containers for storing data values, a memory location for a data type



Reference: geeksforgeeks.org

Primitive Data Types:

built-in or predefined data types,

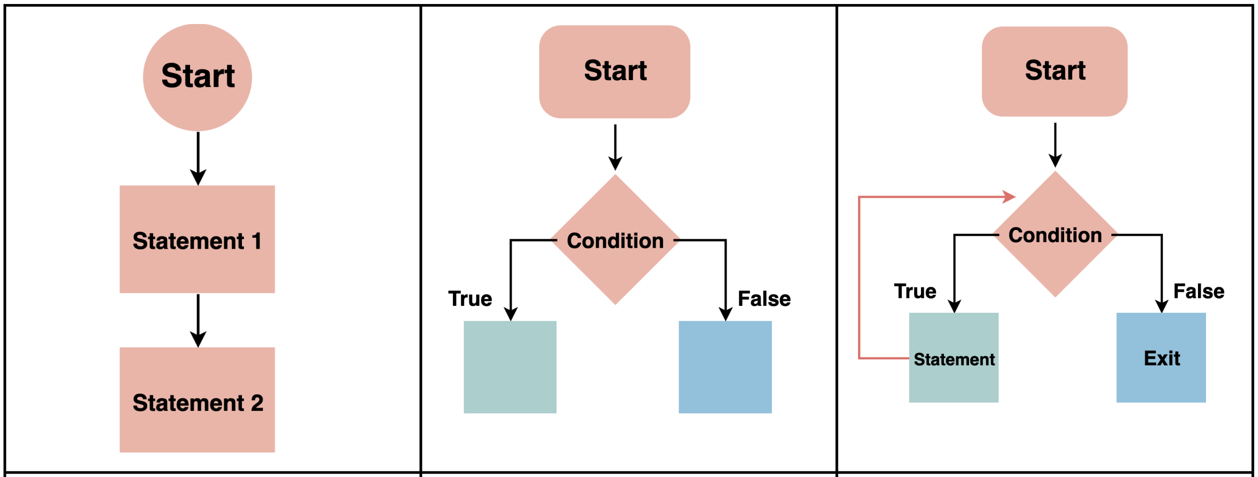
- `int`: Integer
- `char`: Character
- `bool`: Boolean
- `float`: Floating Point
- `double`: Double Floating Point

Derived Data Types:

derived from primitive or built-in data types,

- Function
- Array
- Pointer
- Reference

⇒ Flow Control Structures:



Sequential | Selection | Iteration

Reference: educative.io

⇒ Pointers:

a variable that stores the memory address as its value.

2_pointers-1.cpp

```
int main(){
   // declaring variables
   int var1 = 10;
   int var2 = 10;

   // printing address of variables
   cout << "Address of var1: "<< &var1 << endl;
   cout << "Address of var2: " << &var2 << endl;

   return 0;
}
```

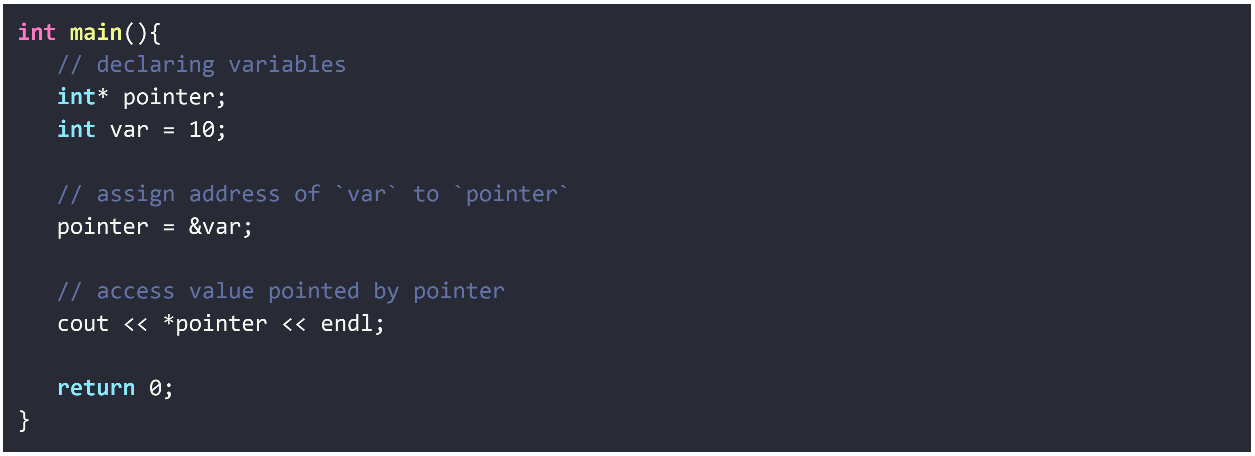
Output:

```
Address of var1: 0x16b86b018
Address of var2: 0x16b86b014
```

Variables when created, are stored in RAM and once the execution of a program finishes, variables are disposed. So if you run the above program a second time, output will change.
Output:

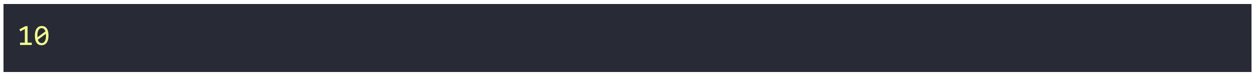
```
Address of var1: 0x16f643018
Address of var2: 0x16f643014
```

3_pointer-2.cpp



```cpp
int main(){
   // declaring variables
   int* pointer;
   int var = 10;

   // assign address of `var` to `pointer`
   pointer = &var;

   // access value pointed by pointer
   cout << *pointer << endl;

   return 0;
}
```

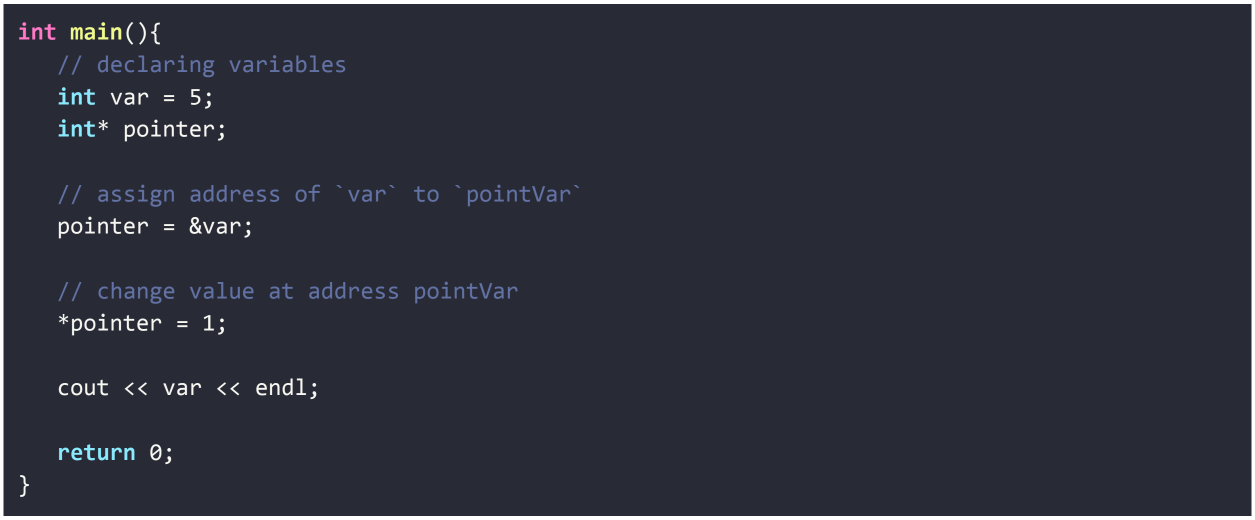
Output:



```
10
```

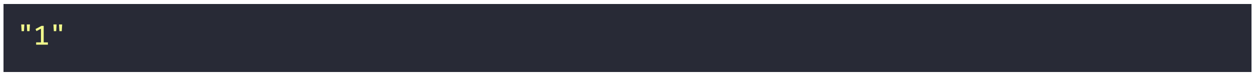
Here, “*” operator works as a dereference operator. It will go to the address pointer is referring to and get the value from that specific address.

4_pointer-3.cpp



```cpp
int main(){
   // declaring variables
   int var = 5;
   int* pointer;

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

   // change value at address pointVar
   *pointer = 1;

   cout << var << endl;

   return 0;
}
```

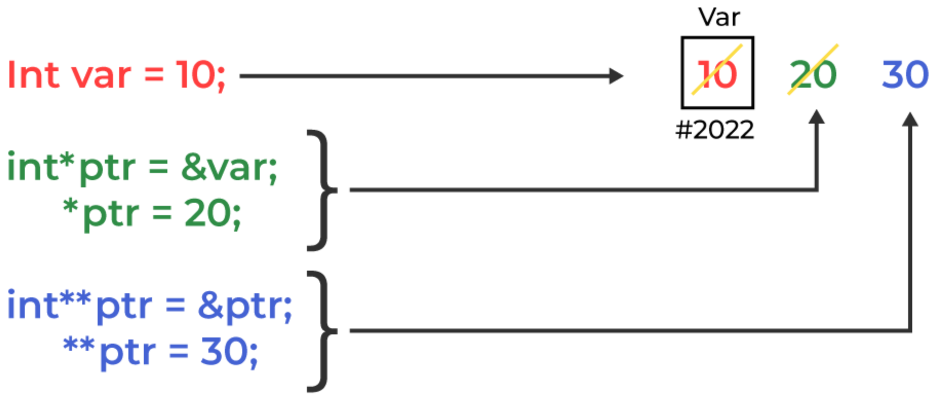
Output:



```
"1"
```

We can also use “*” operator to change the value at the address, current pointer is pointing to.

Another example,



```
Int var = 10;
int*ptr = &var;
   *ptr = 20;
int**ptr = &ptr;
   **ptr = 30;
```

Reference: geeksforgeeks.org

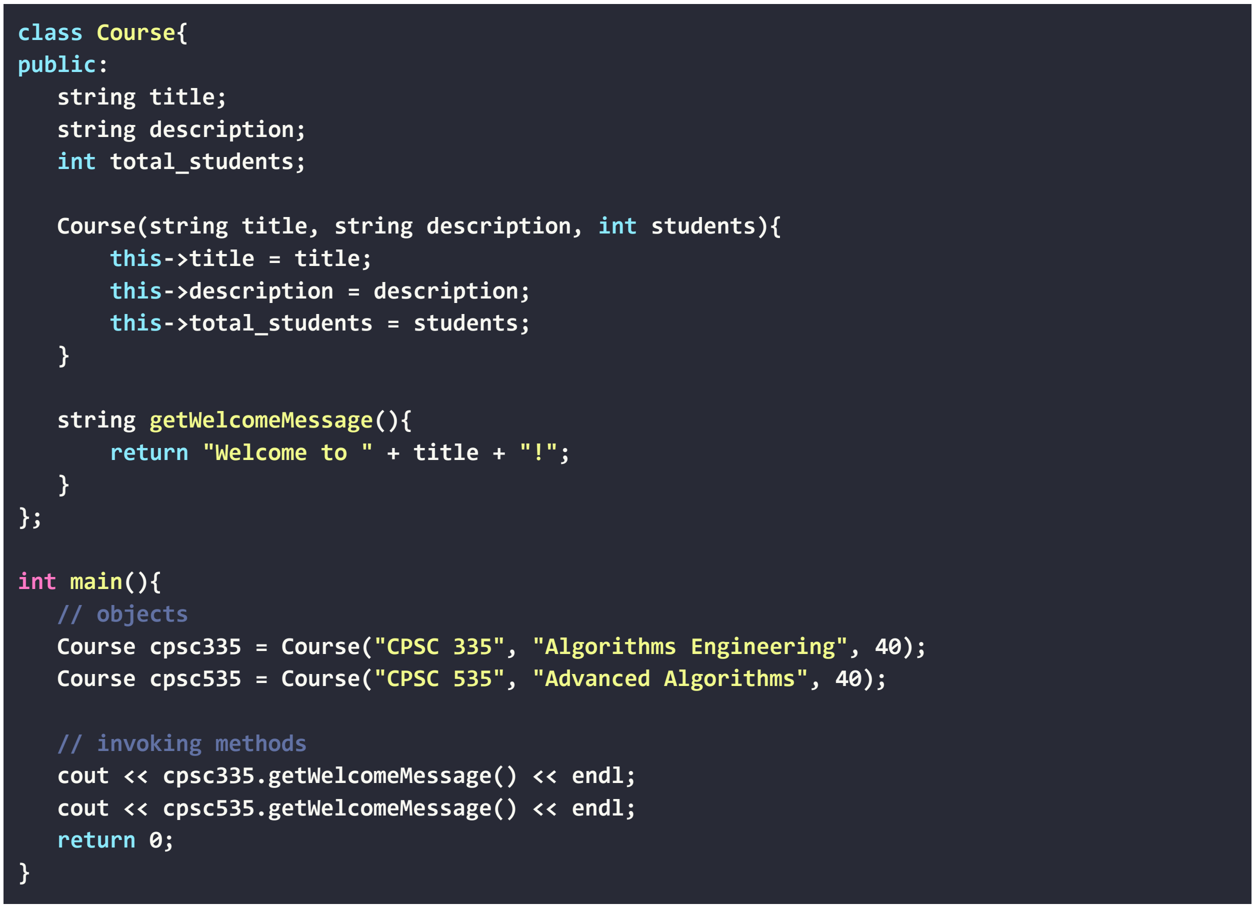
⇒ Class and Objects

Class	Object
A class is a blueprint for declaring and creating objects.	An object is a class instance that allows programmers to use variables and methods from inside the class.
Memory is not allocated to classes. Classes have no physical existence.	When objects are created, memory is allocated to them in the heap memory.
You can declare a class only once.	A class can be used to create many objects.
Class is a logical entity.	An object is a physical entity.
We cannot manipulate class as it is not available in memory.	Objects can be manipulated.
Class is created using the class keyword like class Dog{}	Objects are created through new keywords like Dog d = new Dog();. We can also create an object using the newInstance() method, clone() method, factory method and using deserialization.

Example: Mobile is a class.	If Mobile is the class then iphone, redmi, blackberry, samsung are its objects which have different properties and behaviors.

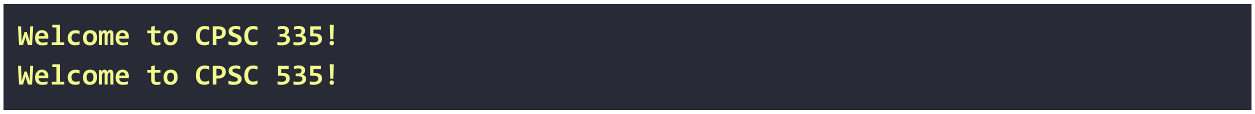
Reference: scaler.com

5_class-objects.cpp



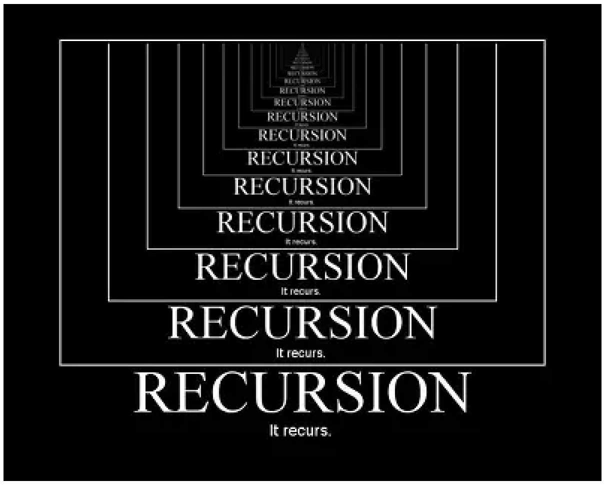
```cpp
class Course{
public:
   string title;
   string description;
   int total_students;

   Course(string title, string description, int students){
       this->title = title;
       this->description = description;
       this->total_students = students;
   }

   string getWelcomeMessage(){
       return "Welcome to " + title + "!";
   }
};

int main(){
   // objects
   Course cpsc335 = Course("CPSC 335", "Algorithms Engineering", 40);
   Course cpsc535 = Course("CPSC 535", "Advanced Algorithms", 40);

   // invoking methods
   cout << cpsc335.getWelcomeMessage() << endl;
   cout << cpsc535.getWelcomeMessage() << endl;
   return 0;
}
```

Output:



```
Welcome to CPSC 335!
Welcome to CPSC 535!
```

⇒ Recursion



Reference: compucademy.net

6_recursion.cpp

```cpp
void print(int n){
   // base case
   if(n < 0){
       return;
   }

   // body
   cout << n << " ";

   // recursion
   print(n-1);
}

int main(){
   int n = 5;
   print(n);
}
```

Output:

```
5 4 3 2 1
```

Review of Data Structures

⇒ What is a Data Structure?

- “Data structure is a specialized format for organizing, processing, retrieving and storing data.” – Article
- “Data structure is a storage that is used to store and organize data. It is a way of arranging data on computer so that it can be accessed and updated efficiently.” – Article

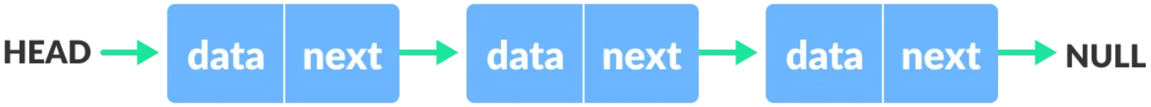
⇒ Why do we need “structure”?

- Data structures make it easy for users to access and work with the data they need in appropriate ways.
- Data structure **enhances efficiency, reusability, and abstraction.** – Article

Example, let’s say you have a book where you write all your friends/family’s phone numbers. If it is unstructured then you will have to go through all pages to find any one number. But, if you sort them by name then retrieving the phone number will be way more easier.

⇒ Data Structures

- Linked List:

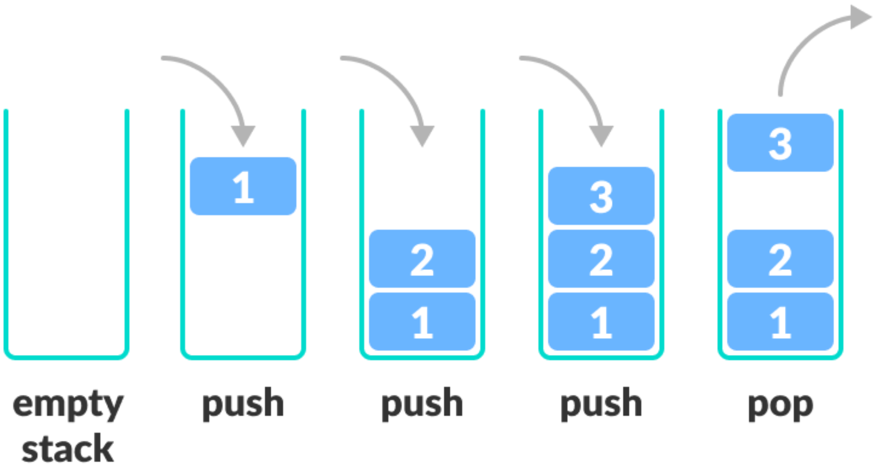


Reference: Programiz

Use Cases:

- Dynamic memory allocation
- Implemented in Stack and Queue

- Stack:

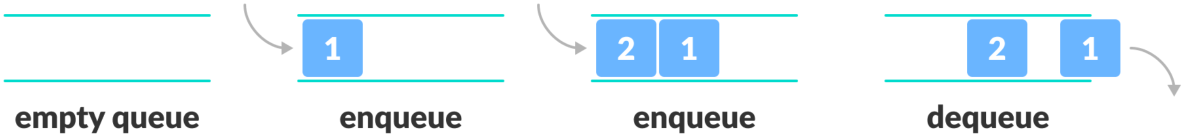


Reference: [Programiz]

Use Cases:

- To reverse a word
- Calculating prefix/postfix of an expression

- Queue



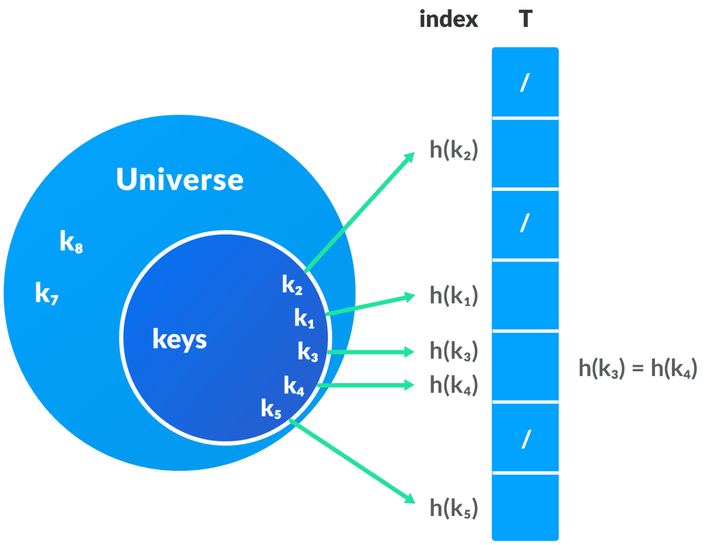
Reference: [Programiz]

Use Cases:

- CPU scheduling, Disk Scheduling
- Call Center phone systems use Queues to hold people calling them in order

---

- Hash Table:

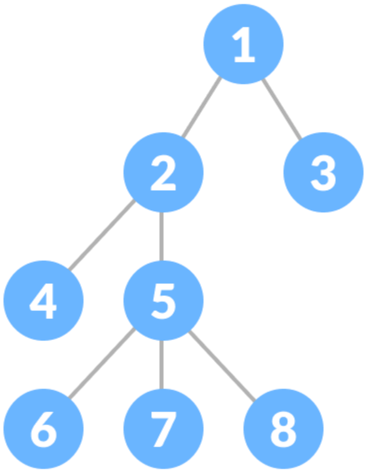


Reference: Programiz

Use Cases:

- Constant time lookup and insertion is required
- Cryptographic applications

- Tree:

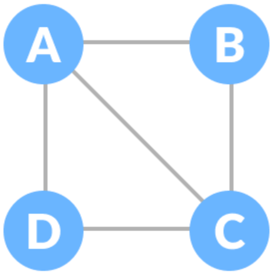


Reference: Programiz

Use Cases:

- Binary Search Trees(BSTs) are used to quickly check whether an element is present in a set or not.
- Compilers use a syntax tree to validate the syntax of every program you write.

- Graph:



Reference: Programiz

Use Cases:

- Finding the path from one vertex to another