Coding Bootcamp

Karan Taneja

Fourth-year Dual Degree Student

Electrical Engineering Department

Topics

- OOP with C++ Programming Language
- Understanding Pointers
- First Program in Java
- Getting Started with Programming Problems
- Resources for Placement Preparation

Humble Request

Raise you hand. Higher!

Topics

- OOP with C++ Programming Language
- Understanding Pointers
- First Program in Java
- Getting Started with Programming Problems
- Resources for Placement Preparation

Assumptions

- Using any compiler for C++ on Unix or any IDE on windows, etc.
- You can read this code and understand it **completely**:

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

int main() {
  cout << "Hello World";
  return 0;
}
```

Assumptions

- ‘for’ and ‘while’ loop
- ‘if else-if else’ statements
- Operators: ==, !=, &, &&, ? etc.
- Creating variables, types (and their sizes in memory)
- Functions
- Classes: public and private - variables and functions/methods
- Local and global variables

Some More Syntax

#define Preprocessor

```
#define LENGTH 10
#undef LENGTH
```

‘typedef’ Declarations:

```
typedef type newname;
typedef int feet
feet distance;
```

Enumerated Types

```
enum color_t { red, green, blue };
color_t c = blue;
enum color_t { red, green = 5, blue };
```

Some More Syntax

```
typedef struct {
        double *ptr;
} A;
void func(const A *a){
        a->ptr[0] = 1.0;
}
```

```
int main(){
        A *a;
        a = new A;
        a->ptr = new double[10];
        a->ptr[0]=10.0;
        func(a);
        cout<<a->ptr[0]<<endl;
        return 0;
}
```

Some work you may need to do…

- How to use cmath functions
- How to generate random numbers
- About arrays – multi-dimensional, etc.
- About strings – character arrays and string class
- I/O libraries – cout, cin, read/write files, etc.

Functions

- Call by value
- Call by pointer
- Call by reference

```
int sum(int a, int b = 20) {
   int result;
   result = a + b;
   return result;
}
```

Functions

- Call by value
- Call by pointer
- Call by reference

```
void swap(int *x, int *y) {
  int temp;
  temp = *x;
  *x = *y;
  *y = temp;
  return;
}
```

Functions

- Call by value
- Call by pointer
- Call by reference

```
void swap(int &x, int &y) {
  int temp;
  temp = x;
  x = y;
  y = temp;
  return;
}
```

C++ Data Structures

```
#include <iostream>
#include <cstring>

struct Books {
   char  title[50];
   char  author[50];
   char  subject[100];
   int   book_id;
};
```

```
int main() {
   struct Books book1;
   strcpy(book1.title, "Learn C++”);
   strcpy(book1.author, "Chand Miyan");
   strcpy( book1.subject, "C++");
   book1.book_id = 6495407;

   cout << “Title : " << book1.title <<endl;
   cout << “Author : " << book1.author <<endl;
   cout << “Subject : " << book1.subject <<endl;
   cout << “ID : " << book1.book_id <<endl;
   return 0;
}
```

C++ Data Structures

```
#include <iostream>
#include <cstring>

typedef struct {
   char  title[50];
   char  author[50];
   char  subject[100];
   int   book_id;
} Books;
```

```
int main() {
   Books book1;
   strcpy(book1.title, "Learn C++”);
   strcpy(book1.author, "Chand Miyan");
   strcpy( book1.subject, "C++");
   book1.book_id = 6495407;

   cout << “Title : " << book1.title <<endl;
   cout << “Author : " << book1.author <<endl;
   cout << “Subject : " << book1.subject <<endl;
   cout << “ID : " << book1.book_id <<endl;
   return 0;
}
```

Classes

```
class Box {
   public:
      double length;   // Length of a box
      double breadth;  // Breadth of a box
      double height;   // Height of a box
};
```

Classes – private and public

```
class Box {
   double width;

   public:
      double length;
      void setWidth( double wid );
      double getWidth( void );
};
```

```
class Box {
   private:
      double width;

   public:
      double length;
      void setWidth( double wid );
      double getWidth( void );
};
```

Classes – constructor and destructor

```
#include ...
class Line {
  public:
    void setLength( double len );
    double getLength( void );
    Line();   // constructor
    ~Line();  // destructor
  private:
    double length;
};
```

```
Line::Line(void) {
  cout << "Object is being created" << endl;
}
Line::~Line(void) {
  cout << "Object is being deleted" << endl;
}
int main() {
  Line line;
  line.setLength(6.0);
  cout << "Length of line : " << line.getLength() <<endl;
  delete line;
  return 0;
}
```

Classes – member initialization list

```
#include <iostream>

class Foo {
   public:
      int bar;
      Foo(int num): bar(num) {};
};

int main(void) {
   std::cout << Foo(42).bar << std::endl;
   return 0;
}
```

Classes – this Pointer

```
double Volume() {
        return length * breadth * height;
}
int compare(Box box) {
        return this->Volume() > box.Volume();
}
```

Creating Objects

Correct declaration(s)?

1. Dog myDog;
2. Dog myDog = new Dog();
3. Dog myDog = Dog()
4. Dog *myDog = new Dog();
5. Dog *myDog = Dog()

Classes – static variables and methods

```
class Box {
  public:
    static int objectCount;
    Box(double l = 2.0, double b = 2.0,
          double h = 2.0) {
      cout <<"Constructor called." << endl;
      length = l;
      breadth = b;
      height = h;
      objectCount++;
    }
    double Volume() {
      return length * breadth * height;
    }
    static int getCount() {
      return objectCount;
    }

  private:
    double length, breadth, height;
};
```

Inheritance – base class and derived class

```
class Shape {
  public:
    void setWidth(int w) {width = w;}
    void setHeight(int h) {height = h;}
  protected:
    int width, height;
};
class Rectangle: public Shape {
  public:
    int getArea() {
      return (width * height);
    }
};
```

```
int main(void) {
  Rectangle Rect;
  Rect.setWidth(5);
  Rect.setHeight(7);

  cout << "Total area: " << Rect.getArea()
       << endl;
  return 0;
}
```

Inheritance – accessing private and protected variables

```
class Base {
  private:
    int MyPrivateInt;
  protected:
    int MyProtectedInt;
  public:
    int MyPublicInt;
}
```

```
class Derived : Base {
public:
  int foo1()  { return MyPrivateInt;}
  int foo2()  { return MyProtectedInt;}
  int foo3()  { return MyPublicInt;}
};
```

Inheritance – accessing private and protected variables

```
class Base {
  private:
    int MyPrivateInt;
  protected:
    int MyProtectedInt;
  public:
    int MyPublicInt;
}
```

```
class Unrelated {
  private:
    Base B;
  public:
    int foo1()  { return B.MyPrivateInt;}
    int foo2()  { return B.MyProtectedInt;}
    int foo3()  { return B.MyPublicInt;}
};
```

Types of inheritance

class Rectangle: public/protected/private Shape

When the component is declared as:	When the class is inherited as:	The resulting access inside the subclass is:
public	public	Public
protected		protected
private		none
public	protected	protected
protected		protected
private		none
public	private	private
protected		private
private		none

class derived-class: access baseA, access baseB...

Overloading

```
class printData {
  public:
    void print(int i) {
      cout << "Printing int: " << i << endl;
    }
    void print(double  f) {
      cout << "Printing float: " << f << endl;
    }
    void print(char* c) {
      cout << "Printing character: " << c << endl;
    }
};
```

```
class Box {
  public:
    Box operator+(const Box& b) {
      Box box;
      box.length = this->length + b.length;
      box.breadth = this->breadth + b.breadth;
      box.height = this->height + b.height;
      return box;
    }
  private:
    double length, breadth, height;
};
```

Polymorphism – static versus dynamic linkage

```
class Shape {
  protected:
    int width, height;
  public:
    Shape(int a = 0, int b = 0) {
      width = a;
      height = b;
    }
    // virtual function
    virtual int area() {
      cout << "Parent class area :" <<endl;
      return 0;
    }
    // pure virtual function
    // virtual int area() = 0;
};
```

```
class Rectangle: public Shape {
  public:
    Rectangle( int a = 0, int b = 0):Shape(a, b) { }
    int area () {
      cout << "Rectangle class area :" <<endl;
      return (width * height);
    }
};
class Triangle: public Shape {
  public:
    Triangle( int a = 0, int b = 0):Shape(a, b) { }
    int area () {
      cout << "Triangle class area :" <<endl;
      return (width * height / 2);
    }
};
```

Some More Technical Terms

(as if PAW-LEE-MAUR-FI-ZUM was not enough)

- Abstraction
- Encapsulation
- Interfaces

```
class Box {
  public:
    // pure virtual function
    virtual double getVolume() = 0;

  private:
    double length;      // Length of a box
    double breadth;     // Breadth of a box
    double height;      // Height of a box
};
```

Topics

- OOP with C++ Programming Language
- Understanding Pointers
- First Program in Java
- Getting Started with Programming Problems
- Resources for Placement Preparation

Ampersand (&)

```
#include <iostream>

using namespace std;
int main () {
  int  var1;
  char var2[10];
  cout << &var1 << endl;
  cout << &var2 << endl;
  return 0;
}
```

Pointer (*)

```
#include <iostream>
using namespace std;
int main () {
   int  var = 20;.
   int  *ip;
   ip = &var;

   cout << var << endl;
   cout << ip << endl;
   cout << *ip << endl;
   return 0;
}
```

NULL Pointer

```
#include <iostream>
using namespace std;
int main () {
   int  *ptr = NULL;
   cout << "The value of ptr is " << ptr << endl ;
   return 0;
}
```

```
if(ptr)     // succeeds if p is not null
if(!ptr)    // succeeds if p is null
```

Pointer Arithmetic

```
#include <iostream>
using namespace std;
const int MAX = 3;

int main () {
  int  var[MAX] = {10, 100, 200};
  int  *ptr;
  ptr = var;
  …
```

```
for (int i = 0; i < MAX; i++) {
    cout << "Address of var[" << i << "] = ";
    cout << ptr << endl;

    cout << "Value of var[" << i << "] = ";
    cout << *ptr << endl;
    ptr++;
  }
  return 0;
}
```

Arrays and Pointers

```
#include <iostream>
using namespace std;
int main ()  {
  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] << ", ";
  return 0;
}
```

Note: * has higher precedence than ++ or --.

const keyword with Pointer

(well, you're gonna hate me)

```
int x;
int y = 10;
const int * p = &y;
x = *p;         // ok: reading p
*p = x;         // ok or not-ok?
```

const keyword with Pointer

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

void increment_all (int* start, int* stop) {
  int * current = start;
  while (current != stop) {
    ++(*current);
    ++current;
  }
}
```

```
void print_all (const int* start, const int* stop) {
  const int * current = start;
  while (current != stop) {
    cout << *current << '\n';
    ++current;
  }
}
int main () {
  int numbers[] = {10,20,30};
  increment_all (numbers,numbers+3);
  print_all (numbers,numbers+3);
  return 0;
}
```

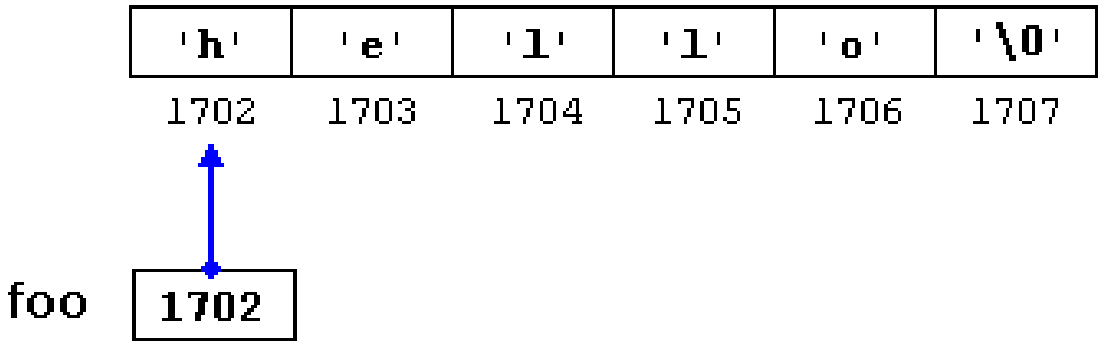
const keyword with Pointer

```
int x;
int *p1 = &x;                  // non-const pointer to non-const int
const int *p2 = &x;            // non-const pointer to const int
int * const p3 = &x;           // const pointer to non-const int
const int * const p4 = &x;     // const pointer to const int

const int *p2a = &x;           // non-const pointer to const int
int const *p2b = &x;           // also non-const pointer to const int
```

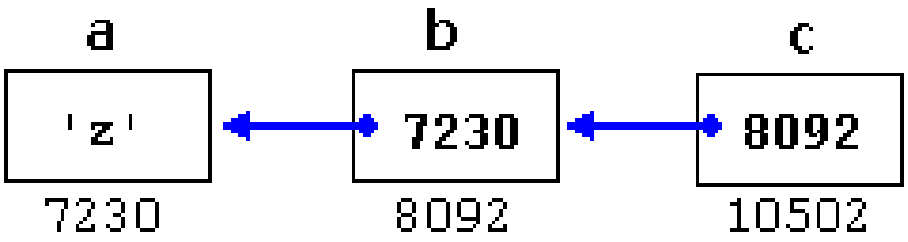
Strings and Pointers

const char * foo = "hello";



Pointers to Pointers

```
char a;
char * b;
char ** c;
a = 'z';
b = &a;
c = &b;
```



void Pointer

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

void increase (void* data, int psize) {
  if ( psize == sizeof(char) )
  { char* pchar; pchar=(char*)data;
++(*pchar); }
  else if (psize == sizeof(int) )
  { int* pint; pint=(int*)data; ++(*pint); }
}
```

```
int main () {
  char a = 'x';
  int b = 1602;
  increase (&a,sizeof(a));
  increase (&b,sizeof(b));
  cout << a << ", " << b << '\n';
  return 0;
}
```

Invalid Pointers and NULL Pointers

```
int * p;
```

```
int myarray[10];
int * q = myarray+20;
```

```
int * p = 0;
int * q = nullptr;
```

Pointers to Functions

```
include <iostream>
using namespace std;
int addition (int a, int b)
  { return (a+b); }
int subtraction (int a, int b)
 { return (a-b); }
int operation (int x, int y, int
(*functocall)(int,int)) {
 int g;
 g = (*functocall)(x,y);
 return (g);
}
```

```
int main (){
 int m,n;
 int (*minus)(int,int) = subtraction;

 m = operation (7, 5, addition);
 n = operation (20, m, minus);
 cout << n;
 return 0;
}
```

Topics

- OOP with C++ Programming Language
- Understanding Pointers
- First Program in Java
- Getting Started with Programming Problems
- Resources for Placement Preparation

First Java Program

```
public class MyFirstJavaProgram {
   public static void main(String []args) {
      System.out.println("Hello World");
   }
}
```

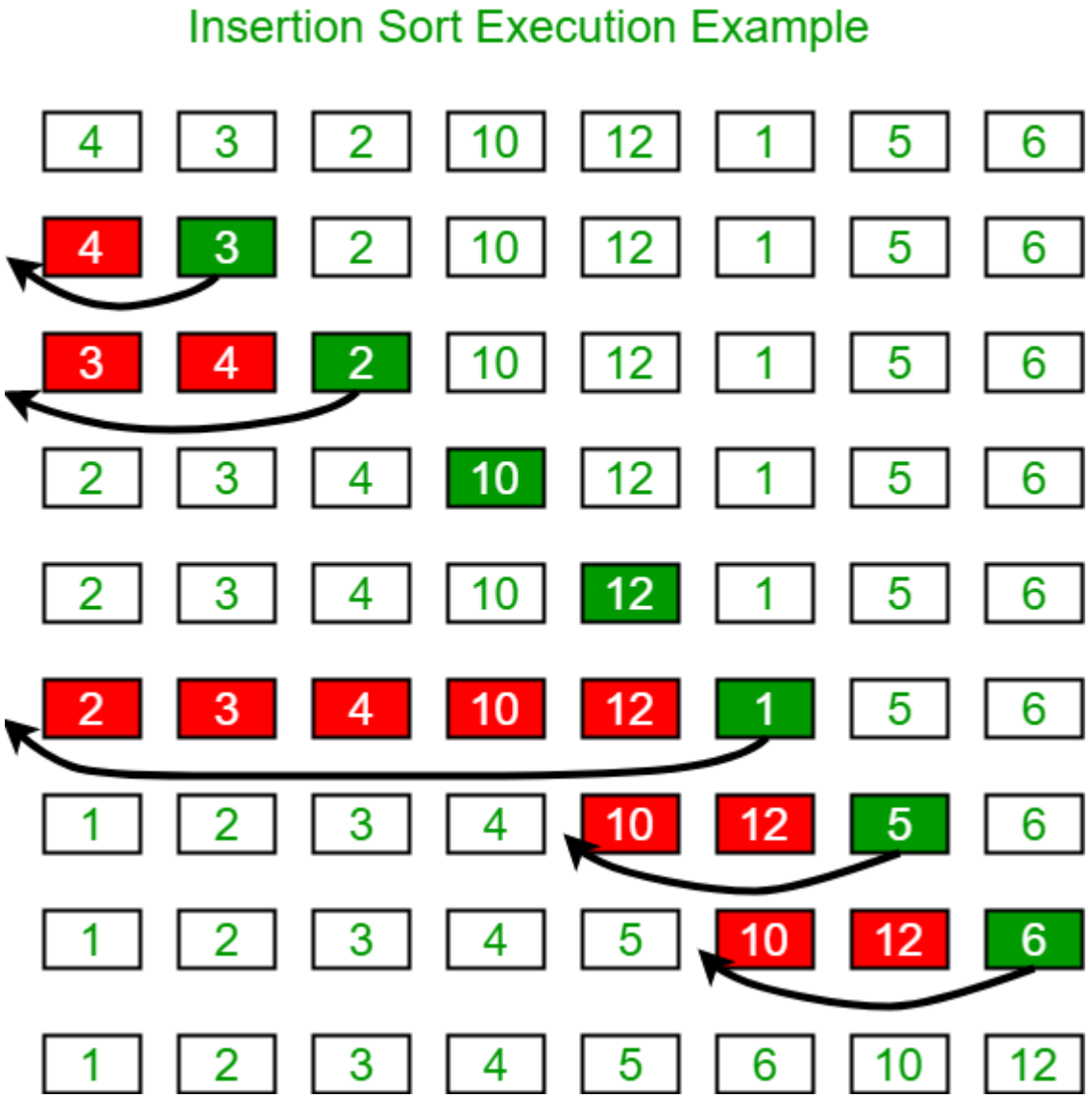
Java Classes

```
public class Puppy {
  int puppyAge;
  public Puppy(String name) {
     System.out.println("Name chosen is :" +
name );
  }
  public void setAge( int age ) {
    puppyAge = age;
  }
  public int getAge( ) {
    System.out.println("Puppy's age is :" +
puppyAge );
    return puppyAge;
  }
 public static void main(String []args) {
    Puppy myPuppy = new Puppy( "tommy" );
    myPuppy.setAge( 2 );
    myPuppy.getAge( );
    System.out.println("Variable Value :" +
myPuppy.puppyAge );
   }
}
```

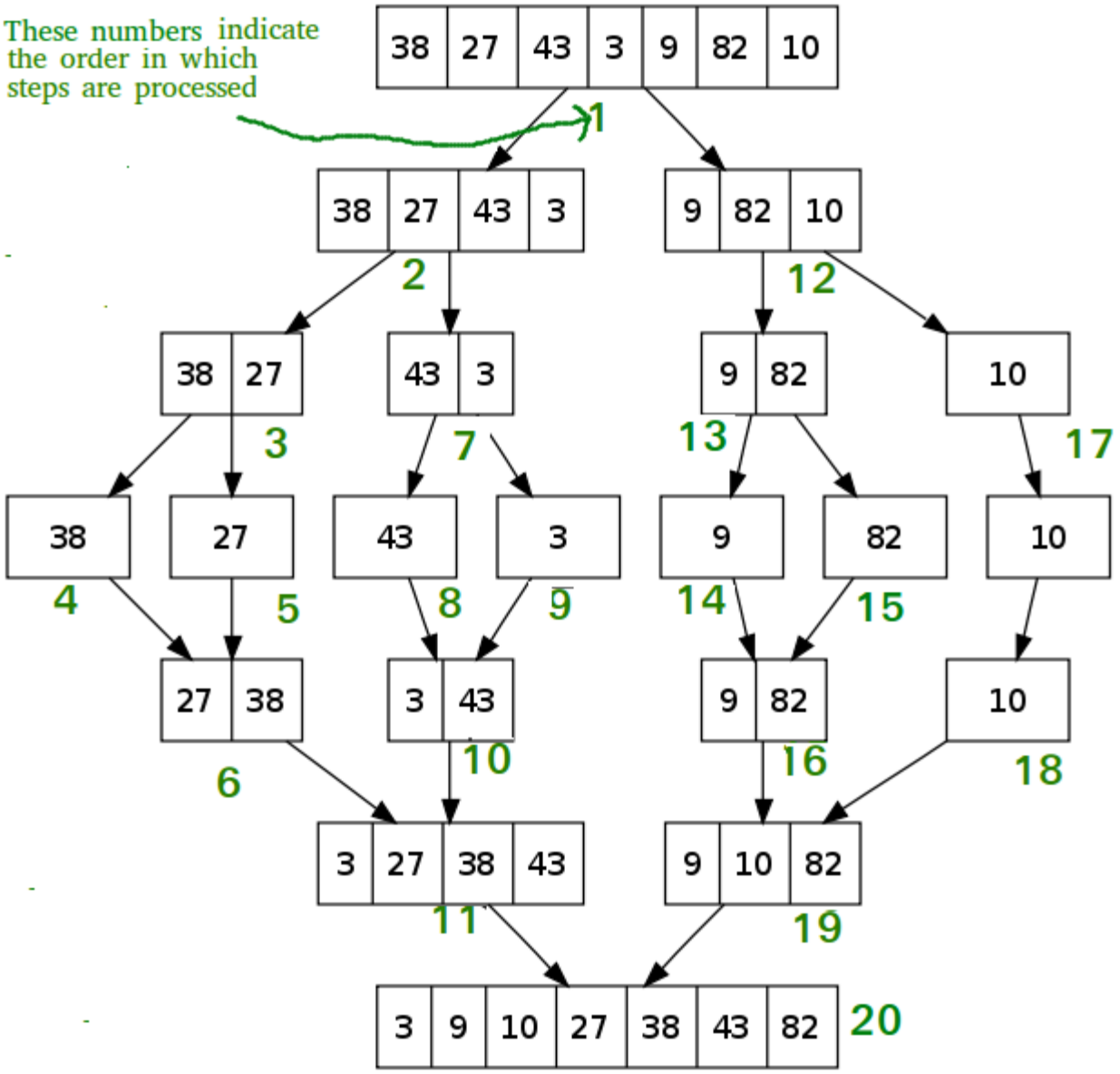
Topics

- OOP with C++ Programming Language
- Understanding Pointers
- First Program in Java
- Getting Started with Programming Problems
- Resources for Placement Preparation

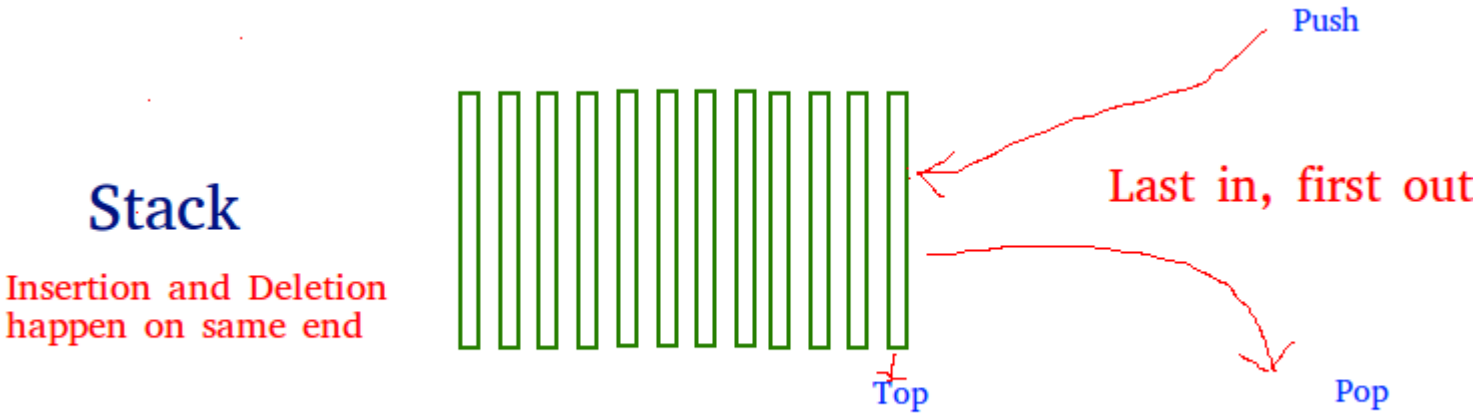
Insertion Sort



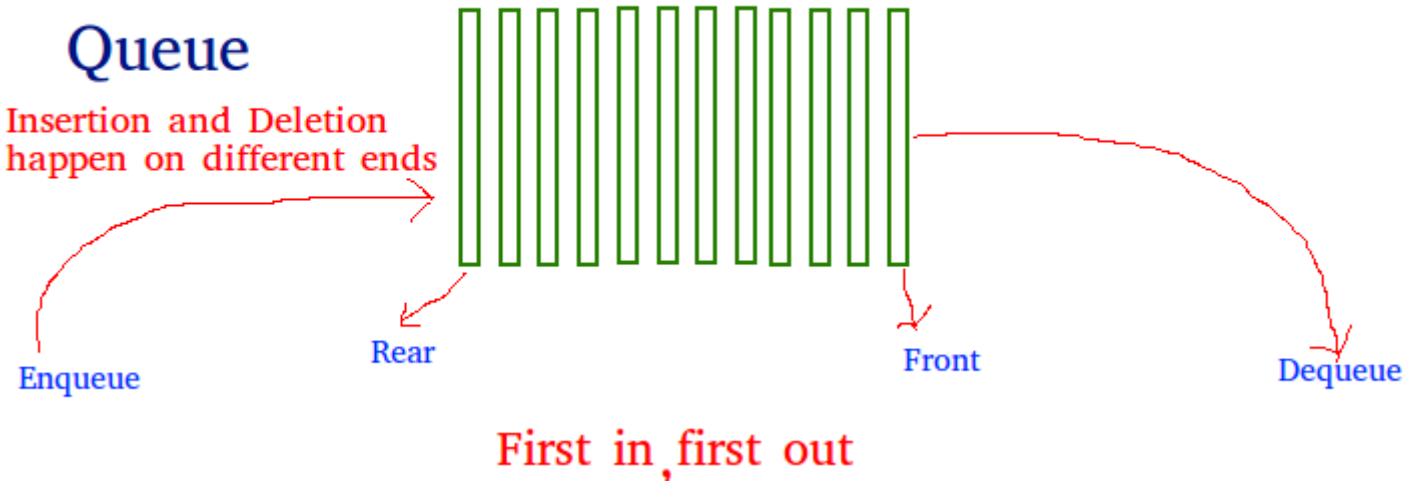
Merge Sort


These numbers indicate the order in which steps are processed
38 27 43 3 9 82 10
1
38 27 43 3
9 82 10
2
12
38 27
43 3
9 82
10
3
7
13
17
38
27
43
3
9
82
10
4
5
8
9
14
15
27 38
3 43
9 82
10
6
10
16
18
3 27 38 43
9 10 82
11
19
3 9 10 27 38 43 82
20

Stacks


Stack
Insertion and Deletion happen on same end
Push
Last in, first out
Top
Pop

Queues


Queue
Insertion and Deletion
happen on different ends
Enqueue
Rear
Front
Dequeue
First in, first out

Recursions

- Davis has a number of staircases in his house and he likes to climb each staircase 1, 2, or 3 steps at a time. Being a very precocious child, he wonders how many ways there are to reach the top of the staircase.
- Given the respective heights for each of the staircases in his house, find and print the number of ways he can climb each staircase, module 10^9 + 7 on a new line.

Some General Tips for Solving Problems

- Read input and output instructions very carefully.
- See the problem constraints carefully.
- *Segmentation Faults*: Most common error is accessing a memory location whose access is not permitted.
- *Don't Repeat Calculations*: Caching the outputs that have to be calculated again and again.
- *How Big is the Problem*: Take care of the size of outputs and use the types accordingly.
- *Macros*: Create macros from problem instructions.

Topics

- OOP with C++ Programming Language
- Understanding Pointers
- First Program in Java
- Getting Started with Programming Problems
- Resources for Placement Preparation

Resources: C++, Java

- C++ Official Tutorial: http://www.cplusplus.com/doc/tutorial/
- Java Official Tutorial: https://docs.oracle.com/javase/tutorial/
- Tutorials Point: http://www.tutorialspoint.com/

Resources: Algorithms

(Problems and Other Help)

- HackerRank Interview Preparation Kit: https://www.hackerrank.com/interview/interview-preparation-kit
- Geeks for Geeks: https://www.geeksforgeeks.org/
- CodeChef: https://www.codechef.com/
- SPOJ: https://www.spoj.com/

Resources: C++

- Other Data Types in C++: http://www.cplusplus.com/doc/tutorial/other_data_types/
- Preprocessor Directives in C++: http://www.cplusplus.com/doc/tutorial/preprocessor/
- Operators in C++: http://www.cplusplus.com/doc/tutorial/operators/
- CMATH Library in C++: http://www.cplusplus.com/reference/cmath/
- Pointers in C++: http://www.cplusplus.com/doc/tutorial/pointers/

Resources: Getting Better at Solving Problems

- Important Shortcuts: https://www.geeksforgeeks.org/important-shortcuts-competitive-programming/
- Common Beginner Mistakes to Avoid: https://www.geeksforgeeks.org/common-mistakes-avoided-competitive-programming-c-beginners/

Getting this presentation…

- Search on Google “Karan Taneja GitHub”
  - https://krntneja.github.io/resources/placements.html

THANK YOU

ALL THE BEST!