DATA Structures

Dr. Asmaa Gad El-Kareem

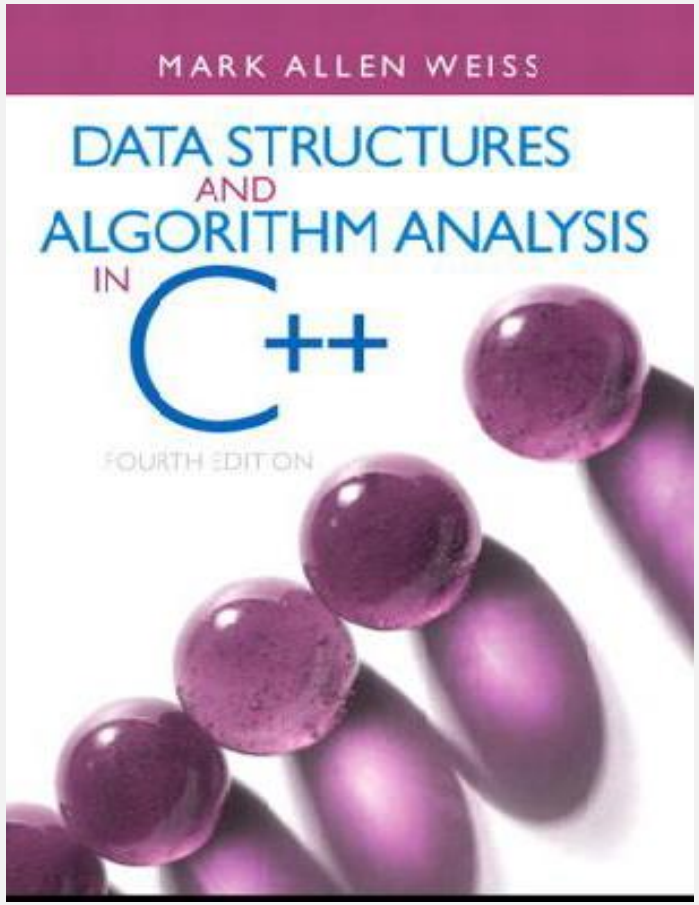
Lecture #1

Get in Touch with Instructor

- **Class Meeting Times & Locations: Sunday 12:00 pm – 2:00 pm**
- **Instructor Name: Dr. Asmaa Gad El-Kareem**
- **Instructor WhatsApp: 01000470736**
- **Instructor E-Mail Address: asmaa@Aru.edu.eg**
- **Instructor Office Hours: Sunday 2:00 pm - 4:00 pm**
- **Instructor Office Location : Staff Member Room -CS Dept.**

Subject Information

- **SUBJECT CODE & NAME:** CS 214- DataStructures
- **CREDIT HRS** : 3
- **CONTACT HRS :** 2 per week
- **TEXT BOOK REFERANCE:**
  - Data Structures and Algorithm Analysis in C++, Fourth Edition
    - by Mark Allen Weiss
  - Lecture material primarily based on my notes



Prerequisites

- Basics of Programming Language (C++)
- Basics Knowledge of Classes and Objects

Topics Covered

- Introduction (Chapter 1+ Lectures 1&2)
- Lists (Chapter 3)
- Stacks (Chapter 3)
- Queues (Chapter 3)
- Trees (Chapter 4)
- Graphs and Graph Algorithms (Chapter 9)
- Hash Tables (Chapter 5)

Grading Policy

Final exam	60 %
Midterm	15%
Oral & Lab	10%
Sheet Excercises	15 %
Total	**100 %**

Conduct in Class

- Don't distract the students. I don't insist that you pay attention, but you must allow others to participate. This means:
  - Do not distract others with conversation.
  - Do not distract others with your phone.
  - Do not distract others by using your laptop in front of the class.
  - If you snore, I will wake you up.
  - Do not attend the class late; you have only 10 minutes to attend it. Otherwise, you will be marked absent on that day.
  - There are no makeup quizzes (if you miss a quiz, you will get zero for that quiz).

Lecture 1: Introduction

- Data and data structures
- Algorithm
- Classification of data structures
- Data Structure Philosophy
- Objectives of this Course
- C++ Revision:
  - Variables & Data types
  - Operators
  - Conditional Processing
  - Looping

Data and Data Structures

- **What is data?**

  - It is a collection of **unprocessed facts** that include numbers, texts, or symbols which can be processed by computers to obtain useful information.

- **Data structure** is a particular way of **storing** and **organizing** a collection of data in a memory so that it can be retrieved and computer performs operations on these data efficiently.

What is Data Structure?

Containers

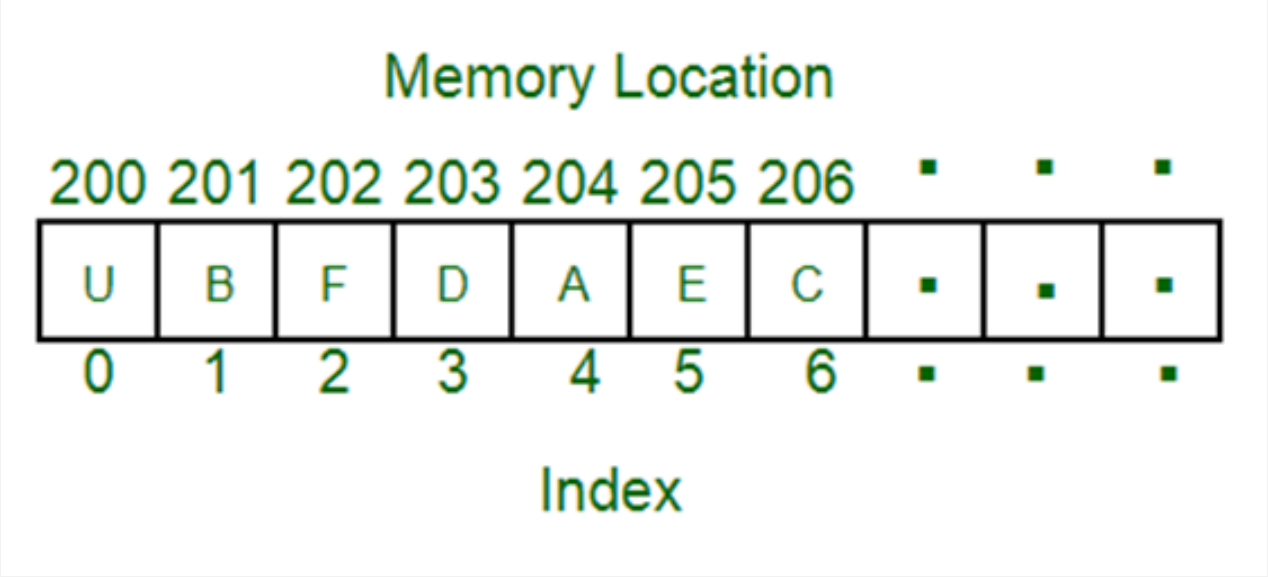
- Contain other data and allow it to be manipulated with one name
- Data structures are like a basket

Organizers

- Organize data for efficient access
- Characterized by the supported operations: insert, search, remove, visit

Data Strutures

For example, we can store a list of items having the same data-type using the array data structure.



Data Structures

contain and organize data

- Data structure considers not only the elements stored but also their relationship to each other.
- No single data structure works well for all purposes, and so it is important to know the strengths and limitations of several of them

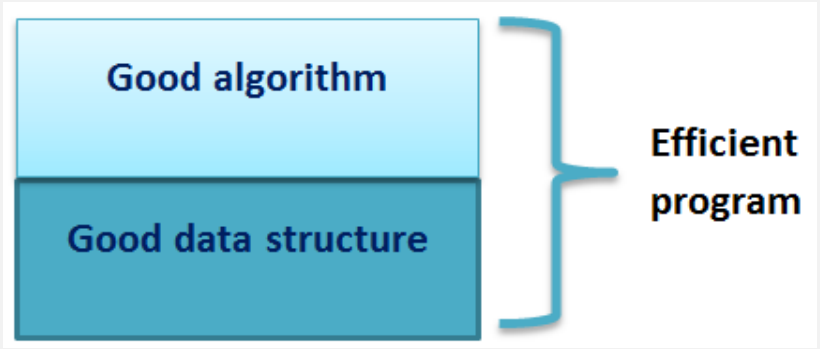
Why do we need to study data structures?

In order to write efficient programs

What is the efficient program?

It is the one that has less cost than other alternatives in terms of space (memory) and time.

- To write an efficient program, you should select good **algorithm**, and select the most appropriate **data structure**.



Algorithm

- An **algorithm** is a clearly specified set of simple instructions to be followed to solve a problem(transform the input into the output).
- It can be expressed either as an **informal high level description** as **pseudocode** or using a **flowchart**.
- An algorithm is not the complete code or program, it is just the **core logic** (solution) of a problem.
- Once an algorithm is given for a problem and decided (somehow) to be correct, an important step is to determine how much in the way of **resources** (time or space), the algorithm will require.
- The process of determining the resources used by the algorithm is called **algorithm analysis**.

Algorithm

- Programs are an implementation of the algorithm in a particular programming language (i.e. a **program** = Data structures + Algorithms
- An algorithm is said to be **efficient** if it solves the problem within its **resource constraints**.
- The ***cost*** of an algorithm is the amount of resources that the algorithm consumes.
- The choice of data structure and algorithm can make the difference between a program running in a few seconds or many days.

Why should we care about program's speed and space when current computers have high speed and large memories?

- Because data also has grown rapidly, they need to be stored, processed and retrieved efficiently
- Example of huge data:

1. Facebook: 3.07 billion active users (2024)

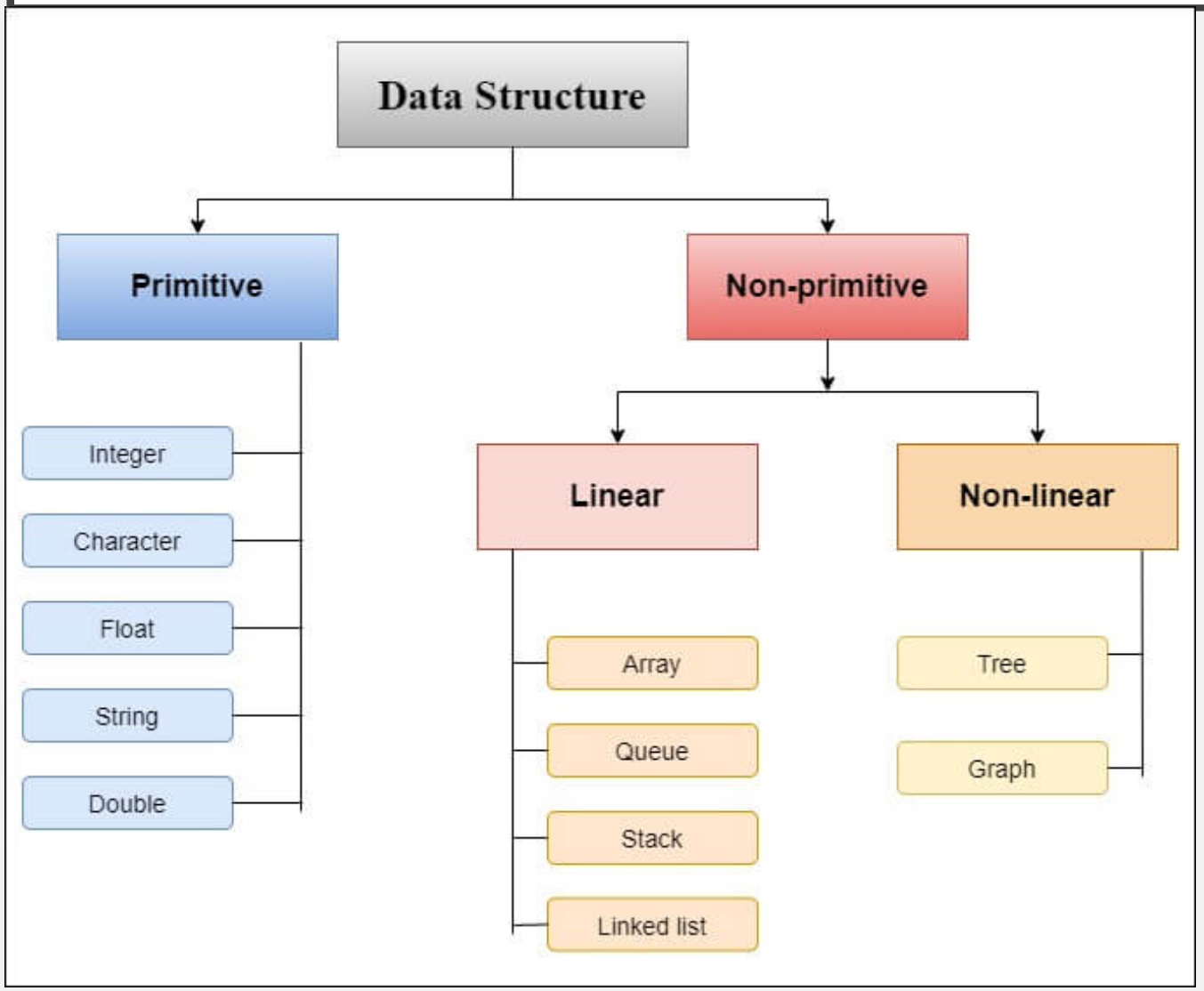
Every day, half million new users are created. Every minute, 510,000 comments are posted.

2. Google receives 63,000 search request/ second, means 3.8 million searches per minute

Data Strutures

- Data Structures are widely used in almost every aspect of Computer Science i.e. Operating System, Compiler Design, Artificial intelligence, Graphics and many more.
- Data Structures are the main part of many computer science algorithms as they enable the programmers to handle the data in an efficient way.
- It plays a vital role in enhancing the performance of a software or a program as the main function of the software is to store and retrieve the user's data as fast as possible.

Classification/ Types of Data Structures (DS)



The DS are divided into two types:

1) Primitive
2) Non primitive

Non primitive divided into two type

1) Linear DS
2) Non linear DS

DATA TYPES

A particular kind of data item, as defined by the values it can take, the Programming language used, or the operations that can be performed on it.

Primitive data structures

- Primitive Data Structure are basic structure and directly operated upon by machine instructions.
- These data types are available in most programming languages as built in type.
- **Integers**, **floats**, **character** and **pointers** are examples of primitive data structures.
  Integer: It is a data type which allows all values without fraction part. We can used it for whole numbers.

  Float: It is a data type which is use for storing fraction numbers.
  Character: It is a data type which is used for character values.

  Pointer: A variable that hold memory address of another variable are called pointer.

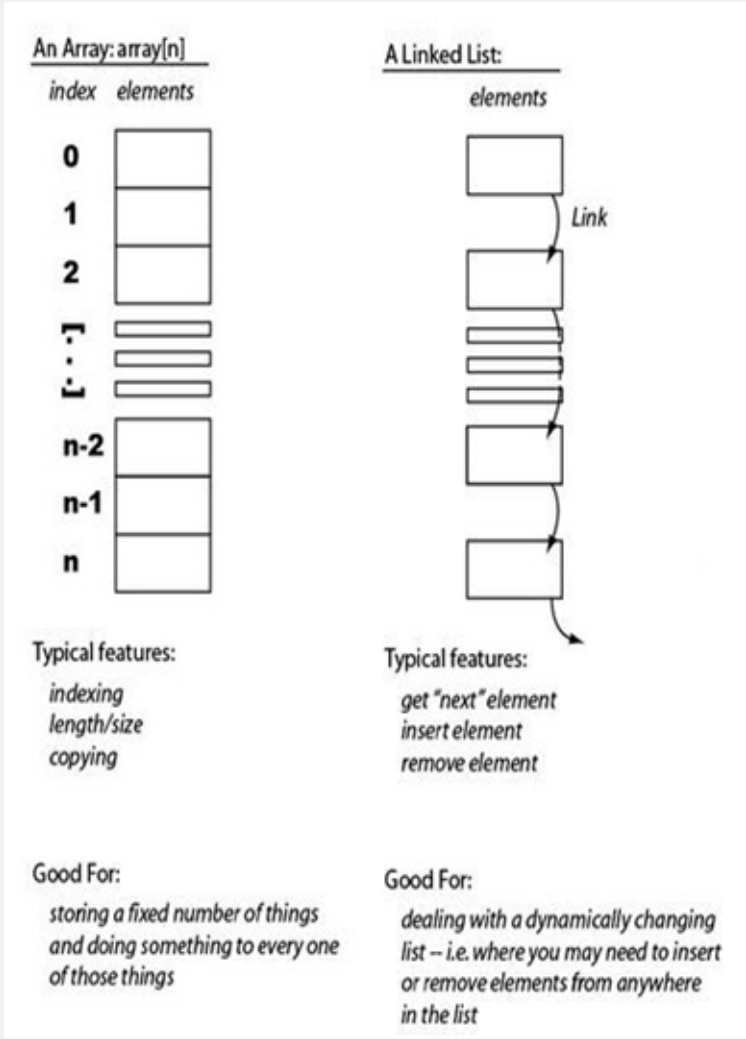
Non-primitive data structure

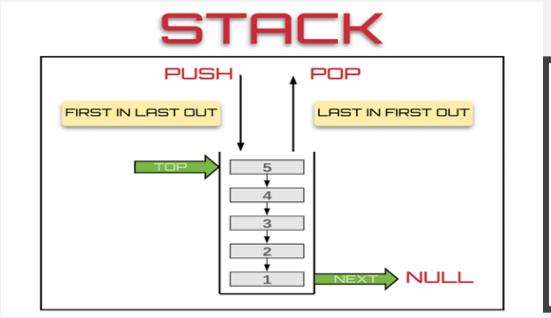
- These are more sophisticated data structures.
- These are derived from primitive data structure.
- The non-primitive data structures emphasize on structuring a group of homogeneous (same type) or heterogeneous (different type) data items.
- A non – primitive data type is further divided into **Linear** and **non – Linear** data structure.
- Examples of non – primitive data types are Array, Stack, and Queue etc.
  Array: is a fixed size sequenced collection of elements of the same data type.
  Stack : is a data structure in which insertion and deletion operations are performed at one end only.
  Queue: is a data structure which permits the insertion at one and deletion at another end

Classification of Data Structures

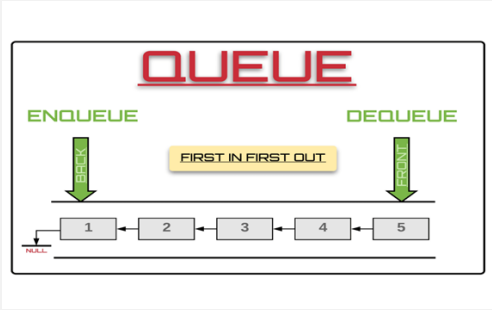
Linear data structure

- Data structure in which data elements are arranged sequentially or linearly, where each element is attached to its previous and next adjacent elements, e.g, **array**, **stack**, **queue**, **linked list**, etc.
- Example is the **array** of characters it represented by one character after another.
- The possible operations on the linear data structure are:
  1) Traversing 2) Insertion 3) Deletion 4) Searching
  5) Sorting 6) Merging

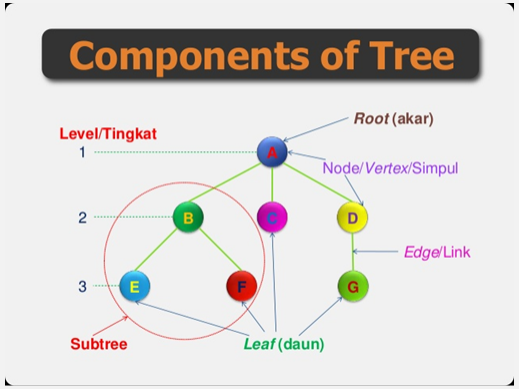




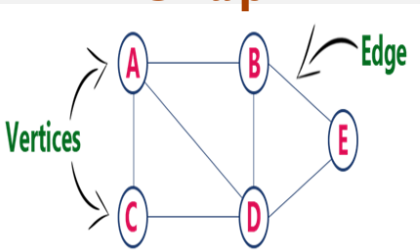
Classification of Data Structures



- There are two types of linear data structure for representing data in memory:
    1. **Static data structure**: it has a **fixed** memory size. It is easier to access the elements in a static data structure, e.g, **array**.
    2. **Dynamic data structure**: in it, the size is **not fixed**. It can be randomly updated during the runtime which may be considered efficient concerning the memory (space) complexity of the code, e.g., **queue**, **stack**, etc.



Classification of Data Structures



Non-linear data structure

- Data structures where data elements are not placed sequentially or linearly.
- Elements are arranged in one-many, many-one and many-many dimensions.
- In a non-linear data structure, we can't traverse all the elements in a single run only.
    - Examples are **trees** and **graphs**.

Difference Between Linear and Non-Linear Data Structure

Linear Data Structure

- Every item is related to its previous and next item.
- Data is arranged in linear sequence.
- Data items can be traversed in a single run
- E.g. Array, Stacks, Linked list, Queue
- Implementation is easy.

Non – Linear Data Structure

- Every item is attached with many other items.
- Data is not arranged in sequence.
- Data cannot be traversed in a single run.
- E.g. Tree, Graph
- Implementation is difficult.

Operation on Data Structures

Design of efficient data structure must take operations to be performed on the DS into account. The most commonly used operations on DS are broadly categorized into following types

1. **Create:** This operation results in **reserving memory** for program elements. This can be done by declaration statement. Creation of DS may take place either during compile-time or run-time.
2. **Destroy:** This operation **destroy memory space** allocated for specified data structure .
3. **Selection:** This operation deals with **accessing** a particular data within a data structure.
4. **Updation:** It **updates or modifies** the data in the data structure.
5. **Searching:** It finds the **presence** of desired data item in the list of data items, it may also find locations of all elements that satisfy certain conditions.
6. **Sorting:** This is a process of **arranging** all data items in a DS in particular order, for example either ascending order or in descending order.
7. **Splitting:** It is a process of **partitioning** single list to multiple list.
8. **Merging:** It is a process of **combining** data items of two different sorted list into single sorted list.
9. **Traversing:** It is a process of **visiting** each and every node of a list in systematic manner.

Abstract Data Type (ADT)

- **Abstract data type (ADT)** s a conceptual model that defines a set of operations and behaviors for a data structure, without specifying how these operations are implemented or how data is organized in memory.
- The set of **objects** such as lists, sets, and graphs together with a set of **operations**, can be viewed as ADTs, just as integers, reals, and booleans are data types.
- The definition of ADT only mentions what operations are to be performed but **not** how these operations will be implemented. It does not specify how data will be organized in memory and what algorithms will be used for implementing the operations.
- It is called "**abstract**" because it provides an implementation-independent view.
- The process of providing only the essentials and hiding the details is known as **abstraction**.

Selecting A Data Structure

Select a data structure as follows:

1. Analyze the problem to determine the resource constraints (space and time) a solution must meet.
2. Determine basic operations that must be supported. Quantify resource constraints for each operation.
3. Select the data structure that best meets these requirements.

Data Structure Philosophy

- Each data structure has costs and benefits.
- Rarely is one data structure better than another in all situations.
- A data structure requires:
  1. space for each data item it stores,
  2. time to perform each basic operation,
  3. programming effort.
- Each problem has constraints on available time and space.
- Only after a careful analysis of problem characteristics can we know the best data structure for the task.

Objectives of this Course

- Understand the basic concepts such as Abstract Data Types, Linear and Non Linear Data structures.
- Understand the behavior of data structures such as stacks, queues, trees, hash tables, search trees, Graphs and their representations.
- Choose an appropriate data structure for a specified application.
- Reinforce the concept that there are costs and benefits for every data structure.
- Learn to implement ADTs such as lists, stacks, queues, trees, graphs, search trees in C++ to solve problems.

C++ Revision

You will be using the C++ programming language in this course

Main part in C++ program

```
#include<iostream.h>
int main()
{

Any code will be
written here

Return 0;
}
```

: indicates a pre-processor directive

include : pre-processor directive include

iostream: input / output stream

.h : header file

main() : entry point

{ } : curly braces encloses the body of the function

Return 0 : terminate the execution of main(), and return the value 0 to the host environment.

; : all program statements are terminated with ';'

Escape sequences in C++ program

- ‘\n’ New line: equivalent to carriage return/line feed
- ‘\t’ Tab
- ‘\?’ Question mark
- ‘\\’ ‘\’ character
- ‘\”’ Double quote

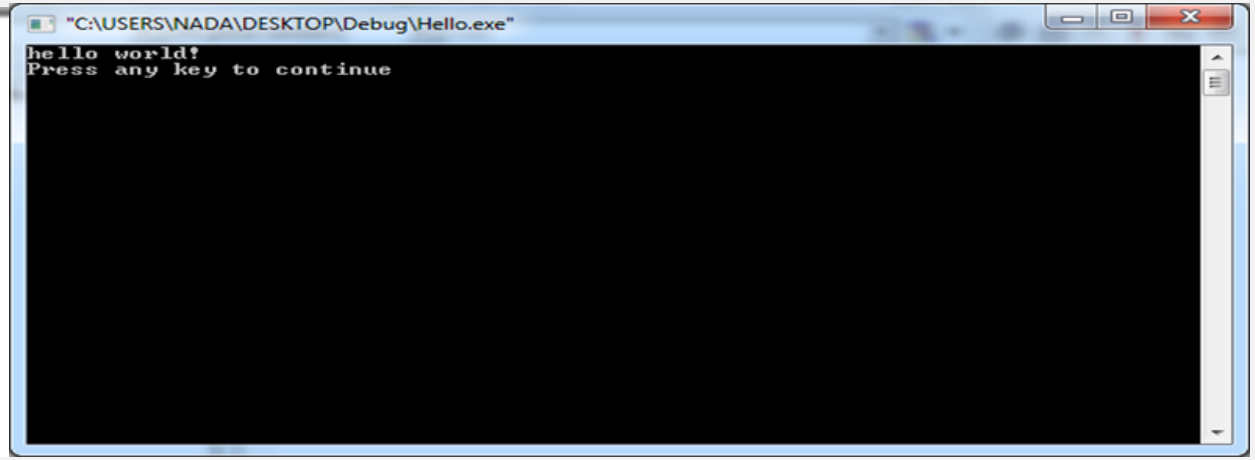
Using comments in C++ program

- Used to **describe the code** in English or provide non-code information
- // symbols indicate a line comment - apply to just **the rest of the line**.
- Block comments start with /* and end with */ - apply to as **many lines** as you like
- **E.g.** to include the name of the program or the author's name

cout stream in C++ program

- cout<< : write to standard output terminal

```
#include<iostream.h>
void main()
{
    cout<<"hello world!"<<"\n";
}
```



Variables & Data types

Identifiers

- Names for **data** and **objects** to be **manipulated** by the program.
- Must begin with a **letter or underscore** (not recommended).
- Consist **only** of letters, digits, and underscores
- **Upper** and **lower** case significant
- Cannot be **reserved word**

Variables & Data types

Reserved Words	Meaning
`const`	Constant; indicates a data element whose value cannot change
`float`	Floating point; indicates that a data item is a real number
`include`	Preprocessor directive; used to insert a library file
`int`	Integer; indicates that the main function returns an integer value

Variables & Data types

Data types

- Integers (short, int, long)
- Positive and negative whole numbers,
  - **E.g.** : 5, -52, 343
- Floating point (real) (float, double, long double)
  - Number has two parts, integral and fractional
  - **E.g.** : 3.6, -.034, 5.0

Variables & Data types

Data types

- Boolean
  - values: true and false
- Characters
  - Represent individual character values
  - **E.g.**: ’A’ , ’a’ ,’2’ ,’*’
  - Stored in 1 byte of memory

Data type	Size	Example
Char	1 byte	-128 to 127
unsigned char	1 byte	0 to 255
int	2 bytes	-32,768 to 32,767
unsigned int	2 bytes	0 to 65,535
long or (long int)	4 bytes	-2,147,483,648 to 2,147,483,647
unsigned long	4 bytes	0 to 4,294,967,295
float	4 bytes	3.4 x 10^-38 to 3.4 x 10^38
double	8 bytes	1.7 x 10^-308 to 1.7 x 10^308
long double	10 bytes	3.4 x 10^-4932 to 3.4 x 10^4932

Variables & Data types

String Class

- Strings not built-in, but come from library
- String literal enclosed in double quotes
- **E.g.**: “Enter speed: “ “ABC” “B” “true” “1234”
- **#include <string.h>**
- for using string identifiers, but **not needed** for literals

Variables & Data types

Declarations

- Set aside memory with a specific name for the data and define its values and operations
- The value can change during execution, **E.g.**: float x, y;

Initialization

- This refers to when you first store something useful in a variable's memory location, **E.g.**: int C=2;

Constant Declarations

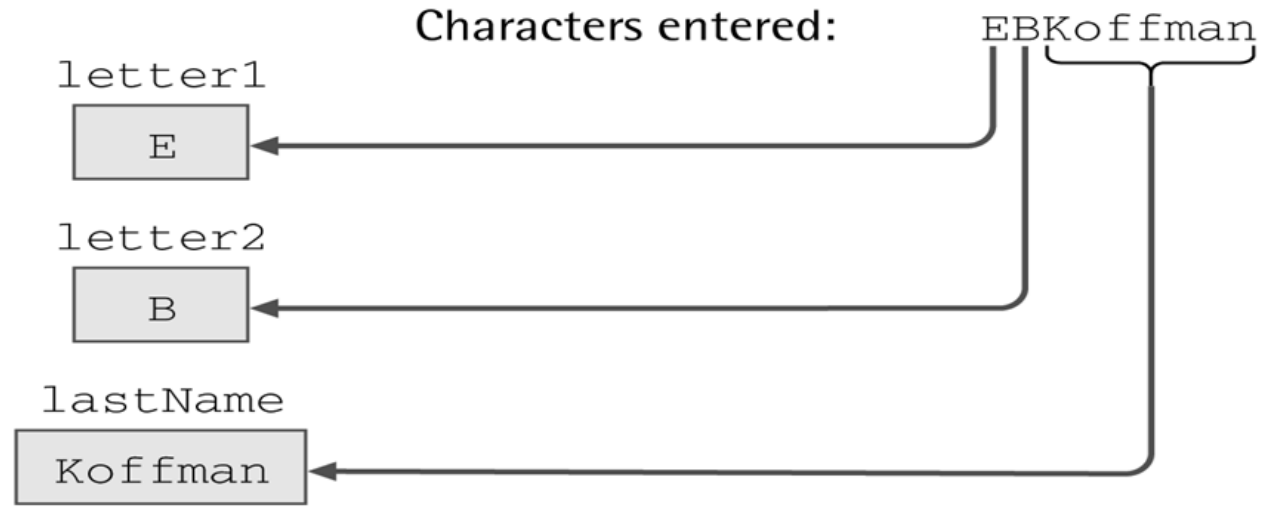
- Memory cells whose values cannot change once set
- const type constant-identifier = value; **E.g.**: const float K = 1.6;

Input Statements

- Obtain data for **program to use** - different each time program executes
- Standard stream **library iostream**
- cin - name of stream associated with standard **input device** (keyboard by default)
- Extraction operator (>>)
- **E.g.**: cin >> miles;

Input Statements

- char letter1,letter2;
- string lastName;
- cin >> letter1 >> letter2 >> lastName;



Operators

Types of operators:

- Assignment
- Arithmetic
- Relational
- Boolean

Using the Assignment Operator

- An assignment operator is an **expression** and can be used whenever an expression is permitted.
- The expression on **the right** is assigned to the variable **on the left**:
- The expression on the **right** is always **evaluated** before the assignment.

```
int var1 = 0, var2 = 0;
var1 = 50;          // var1 now equals 50
var2 = var1 + 10; // var2 now equals 60
```

Using the Assignment Operator

- Assignments can be strung together:

```
var1 = var2 = var3 = 50;
```

Working with Arithmetic Operators

- Perform basic **arithmetic** operations
- Work on **numeric variables** and **literals**

```
int a, b, c, d;
a = 2 + 2;     // addition
b = a * 3;     // multiplication
c = b - 2;     // subtraction
d = b / 2;     // division
e = b % 2;     // returns the remainder of division
```

Working with Arithmetic Operators

- The ++/-- operator
- increments/decrements by 1:

```
int var1 = 3;
var1++;          // var1 now equals 4
```

- The ++/-- operators can be used in two ways:

```
int var1 = 3, var2 = 0;
var2 = ++var1;    // Prefix:  Increment var1 first,
                  //           then assign to var2.
var2 = var1++;    // Postfix: Assign to var2 first,
                  //           then increment var1.
```

Exploring Comparisons

- Relational and equality operators:

>	greater than
>=	greater than or equal to
<	less than
<=	less than or equal to
==	equal to
!=	not equal to

```
int var1 = 7, var2 = 13;
boolean res = true;
res = (var1 == var2);      // res now equals false
res = (var2 > var1);       // res now equals true
```

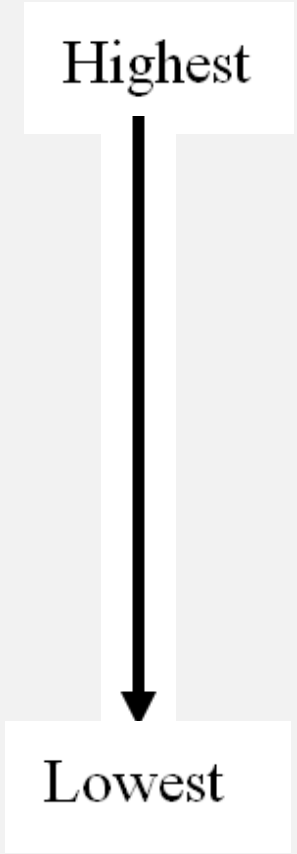
Using Logical Operators

- Results of Boolean expressions can be

&& & || |	and (with / without short-circuit evaluation) or (with / without short-circuit evaluation)
!	not

```
int var0 = 0, var1 = 1, var2 = 2;
boolean res = true;
res = (var2 > var1) & (var0 == 3);   // now false
res = !res;                          // now true
```

Order of Operator Precedence



( ) Power
2 ^3
++, --
*, /, %
+, -

Conditional Processing

- If statement
- Switch case

Using the if Statement

- General Form

```
if ( boolean_expr )
    statement1;
```

- It is better style to use **curly brackets** { }, to demarcate clearly the body of the if statement .
-The style of writing C++ programs that includes **indentation** to indicate the structure of a program..
- Example:

-

```
if (i % 2 == 0) {
        cout << i;
        cout << " is even";
}
```

Using the if Statement

- Using else

```
if ( boolean_expr )
    statement1;
[else
    statement2];
```

- Example:

```
if (i % 2 == 0)
   cout << "Even";
else
   cout << "Odd";
…
```

Nesting if Statements

```
if (speed >= 25)
    if (speed > 65)
            cout << "Speed over 65";
    else
            cout << "Speed greater than or equal to
                    25 but less than or equal to 65";
else
            cout << "Speed under 25";
```

Nesting if Statements

```
if (speed > 65)
        cout << "Speed over 65";
else if (speed >= 25)
        cout << "Speed greater… to 65";
    else
        cout << "Speed under 25";
```

Defining the switch Statement

```
switch ( integer_expr ) {

    case constant_expr1:
        statement1;
        break;
    case constant_expr2:
        statement2;
        break;
    [default:
        statement3;]
}
```

Defining the switch Statement

- Case labels must be constants.
- Use break to jump out of a switch.
- It is recommended to always provide a **default**.

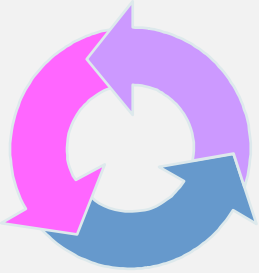
```
switch (choice) {
  case 37:
     cout << "Coffee?";
     break;

  case 45:
     cout << "Tea?";
     break;

  default:
     cout << "???";
}
```

Looping

Looping in any programming language is **repeatedly** executing a **block of code** until a **specified condition** is met.

There are three types of loops :

- while
- do…while
- for



Looping

All loops have at least two parts:

- Iteration condition
- Body

May have:

- Initialization
- Termination

Using the while Loop

While is the simplest loop statement and contains the following general form:

```
while ( boolean_expr )
    statement;
```

Example:

```
int i = 0;
while (i < 10) {
    cout << "i = " << i;
    i++;
}
```

Using the do…while Loop

do…while loops place the test at the end:

```
do
    statement;
while ( termination );
```

Example:

```
int i = 0;
do {
  cout << "i = " << i;
  i++;
} while (i < 10);
```

Using the for Loop

for loops are the most common loops:

```
for ( initialization; termination; iteration )
    statement;
```

Example:

```
for (i = 0; i < 10; i++)
    cout << i;
```

More on the for Loop

Variables can be declared in the initialization part of a for loop:

```
for (int i = 0; i < 10; i++)
    cout << "i = " << i;
```

Initialization and iteration can consist of a list of comma-separated expressions:

```
for (int i = 0, j = 10; i < j; i++, j--) {
    cout "i = " << i;
    cout "j = " << j;
}
```

Implementing the break Statement

Breaks out of a loop or switch statement:

- **Transfers** control to the first statement after the loop body or switch statement

```
…
while (age <= 65) {
    balance = (balance+payment) * (1 + interest);
    if (balance >= 250000)
        break;
    age++;
}
…
```

End of Lecture