BINUS
UNIVERSITY
INTERNATIONAL

Data Structures and Algorithms

Introduction to Data Structures

Design and Analysis of Algorithms

Week 1

Hello World ☺

Maria Seraphina Astriani
(Sera)

D3697

seraphina@binus.ac.id

WA Group?

https://1drv.ms/u/s!AkfRgh1gcQzbmyPhuGw5IoOJtA23?e=Q3scy9

dsa

IDE:

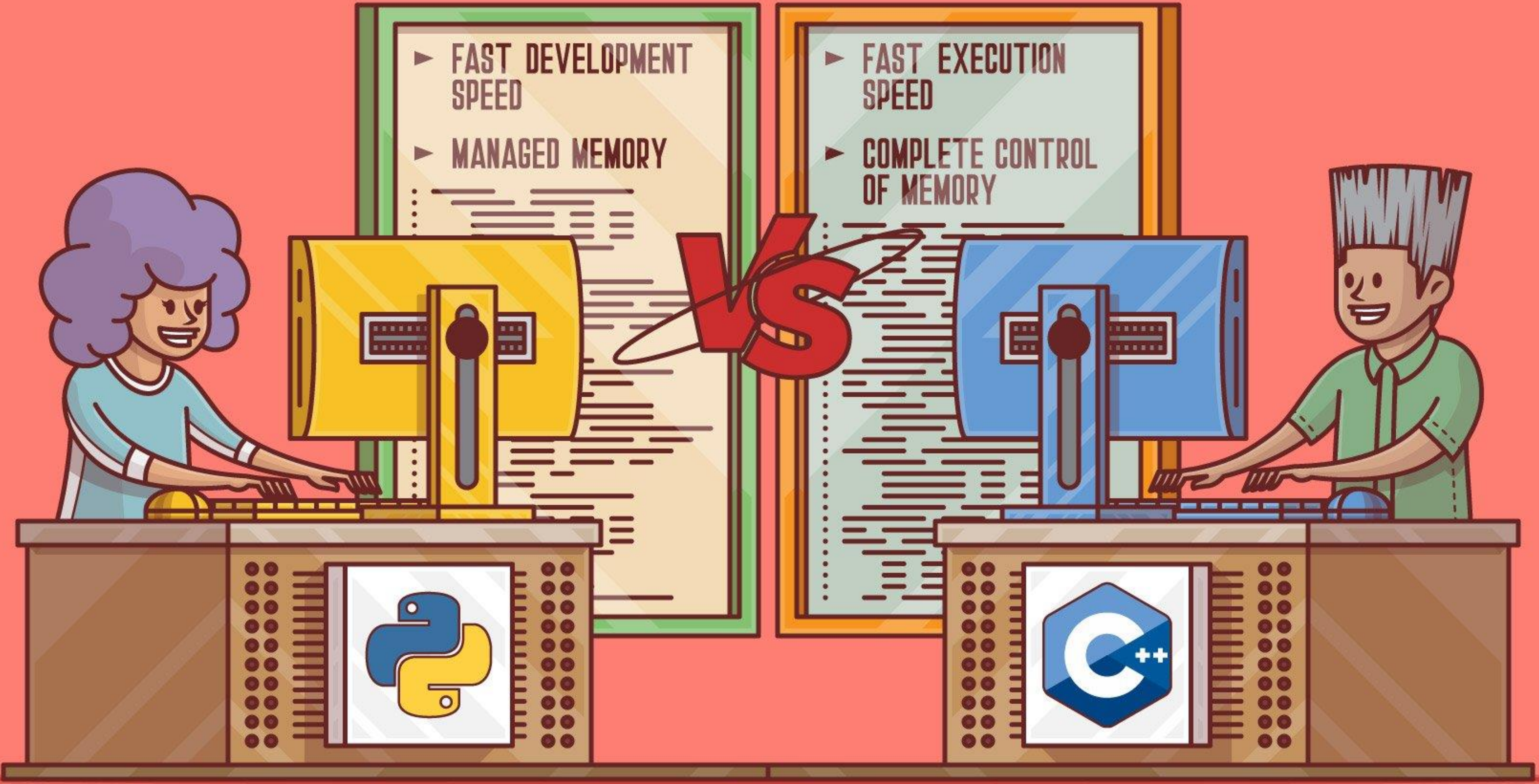
Dev-C++ (download: https://sourceforge.net/projects/orwelldevcpp/)

NetBeans

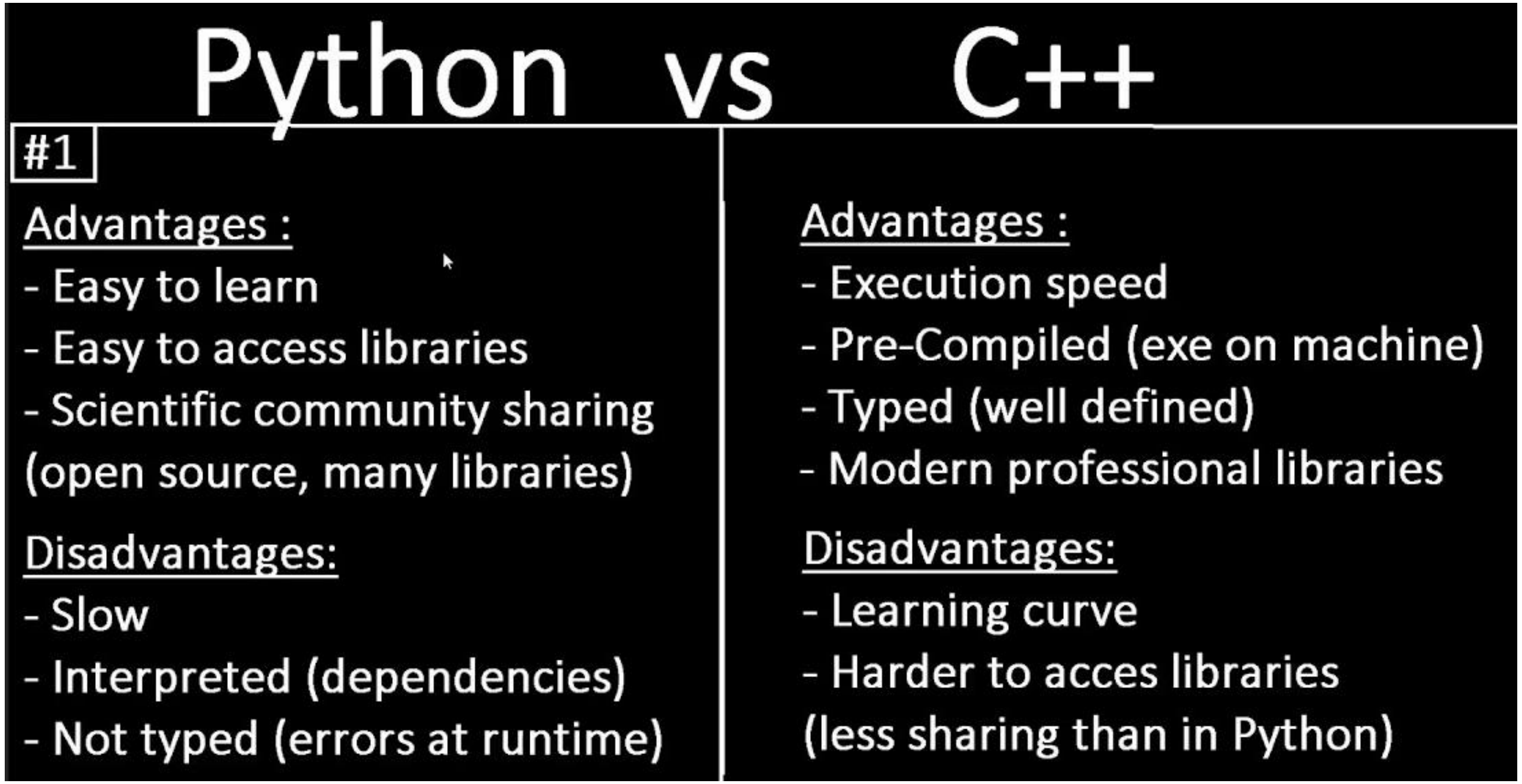
Visual Studio (Express)

Eclipse

Etc…


► FAST DEVELOPMENT SPEED
► MANAGED MEMORY
VS
► FAST EXECUTION SPEED
► COMPLETE CONTROL OF MEMORY
C++

Real Python



Python vs C++

#1

Python

Advantages :

- Easy to learn
- Easy to access libraries
- Scientific community sharing (open source, many libraries)

Disadvantages:

- Slow
- Interpreted (dependencies)
- Not typed (errors at runtime)

C++

Advantages :

- Execution speed
- Pre-Compiled (exe on machine)
- Typed (well defined)
- Modern professional libraries

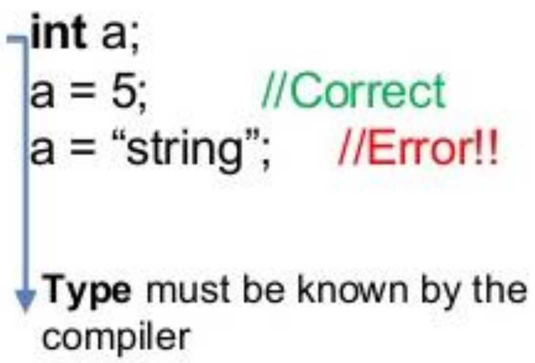
Disadvantages:

- Learning curve
- Harder to acces libraries (less sharing than in Python)

Syntax

C++ (CPP)

Strong Typing



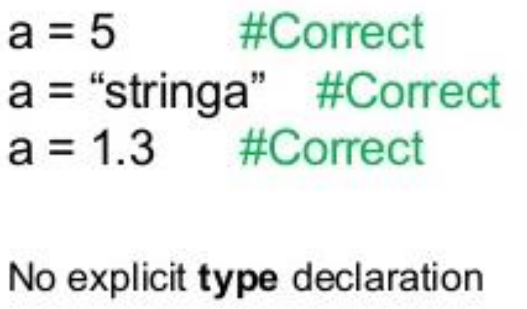
```
int a;
a = 5;         //Correct
a = “string”;      //Error!!
```

Type must be known by the compiler

Python

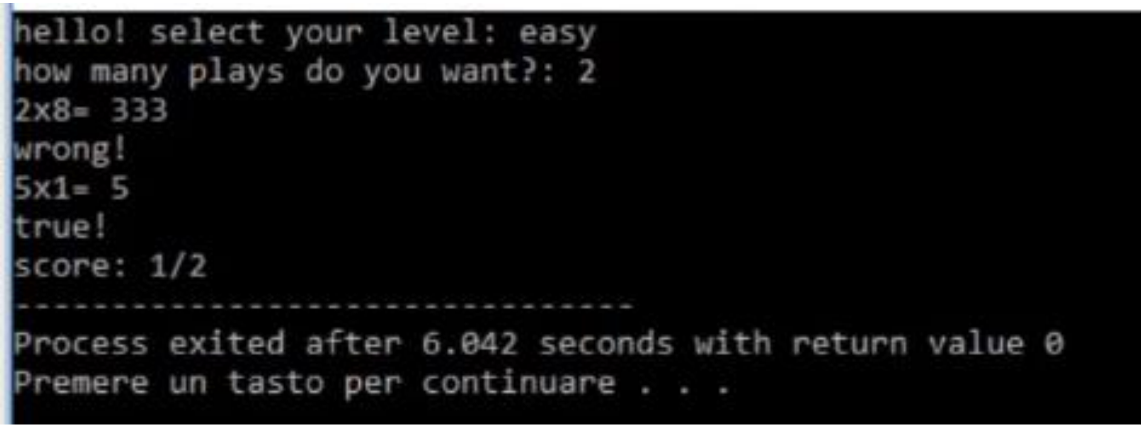
You need to pay attention to the indentation

Weak typing



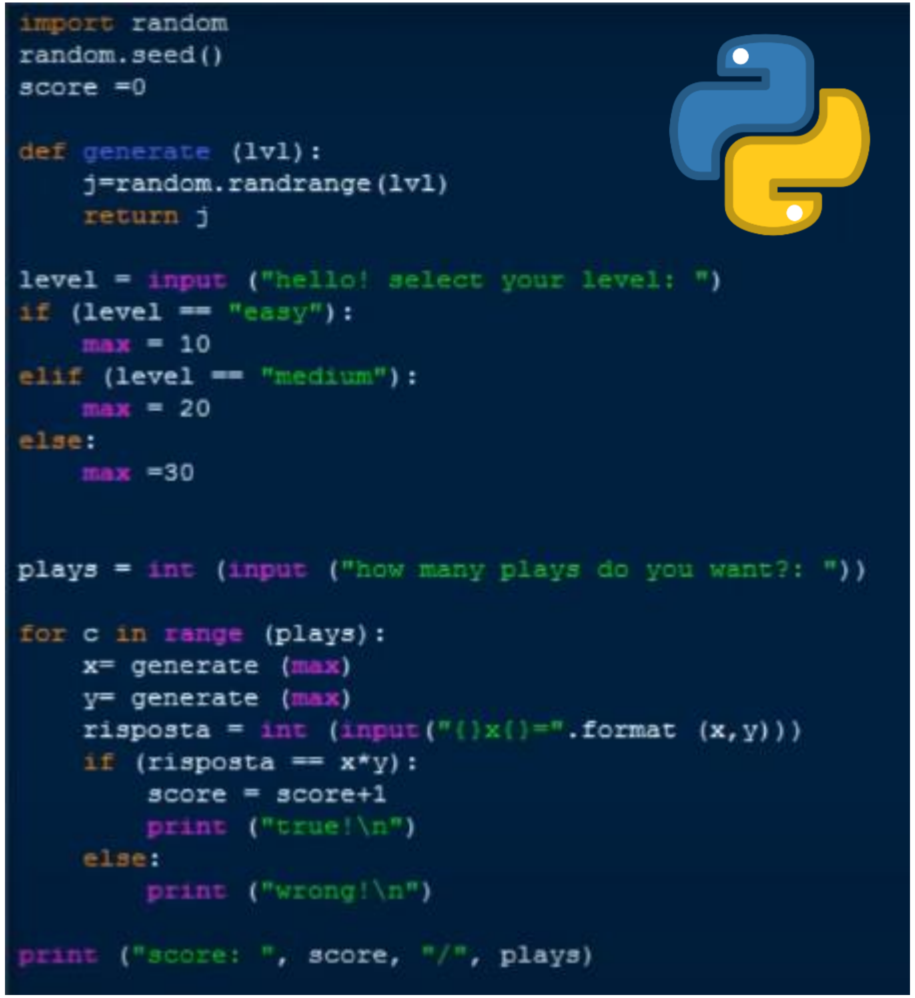
```
a = 5          #Correct
a = “stringa”    #Correct
a = 1.3       #Correct
```

No explicit **type** declaration

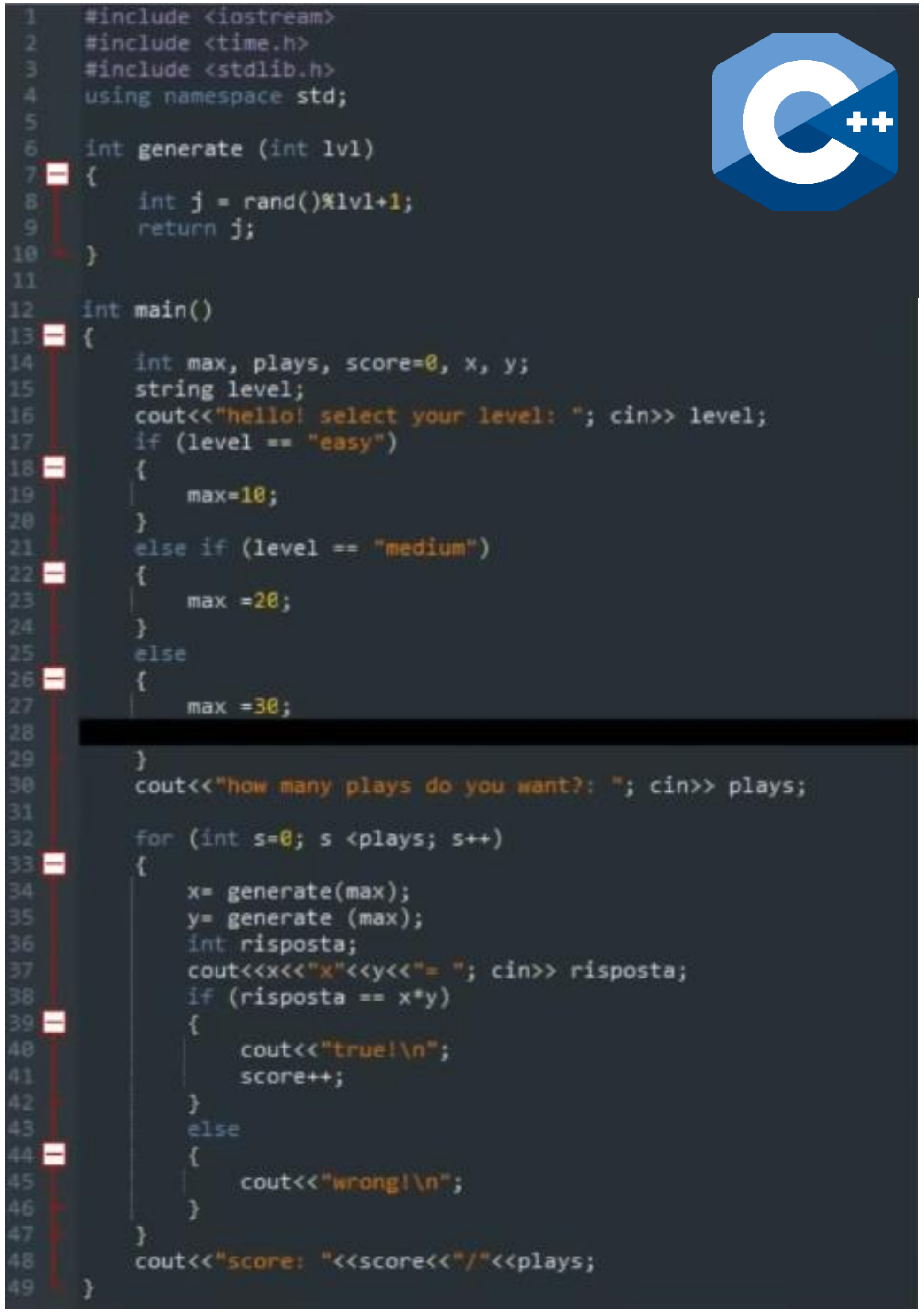


```
hello! select your level: easy
how many plays do you want?: 2
2x8= 333
wrong!
5x1= 5
true!
score: 1/2
--------------------------------
Process exited after 6.042 seconds with return value 0
Premere un tasto per continuare . . .
```

Differences?



```python
import random
random.seed()
score =0

def generate (lvl):
    j=random.randrange(lvl)
    return j

level = input ("hello! select your level: ")
if (level == "easy"):
    max = 10
elif (level == "medium"):
    max = 20
else:
    max =30


plays = int (input ("how many plays do you want?: "))

for c in range (plays):
    x= generate (max)
    y= generate (max)
    risposta = int (input("{}x{}=".format (x,y)))
    if (risposta == x*y):
        score = score+1
        print ("true!\n")
    else:
        print ("wrong!\n")

print ("score: ", score, "/", plays)
```



```cpp
#include <iostream>
#include <time.h>
#include <stdlib.h>
using namespace std;

int generate (int lvl)
{
    int j = rand()%lvl+1;
    return j;
}

int main()
{
    int max, plays, score=0, x, y;
    string level;
    cout<<"hello! select your level: "; cin>> level;
    if (level == "easy")
    {
        max=10;
    }
    else if (level == "medium")
    {
        max =20;
    }
    else
    {
        max =30;

    }
    cout<<"how many plays do you want?: "; cin>> plays;

    for (int s=0; s <plays; s++)
    {
        x= generate(max);
        y= generate (max);
        int risposta;
        cout<<x<<"x"<<y<<"= "; cin>> risposta;
        if (risposta == x*y)
        {
            cout<<"true!\n";
            score++;
        }
        else
        {
            cout<<"wrong!\n";
        }
    }
    cout<<"score: "<<score<<"/"<<plays;
}
```

Please prepare your IDE

(we will use it later....)

BINUS
UNIVERSITY
INTERNATIONAL

Course Description

- COMP6571 – Data Structures and Algorithms
- SCU 4+2 credits
- Lecture session
- Lab session

Syllabus

Course Description

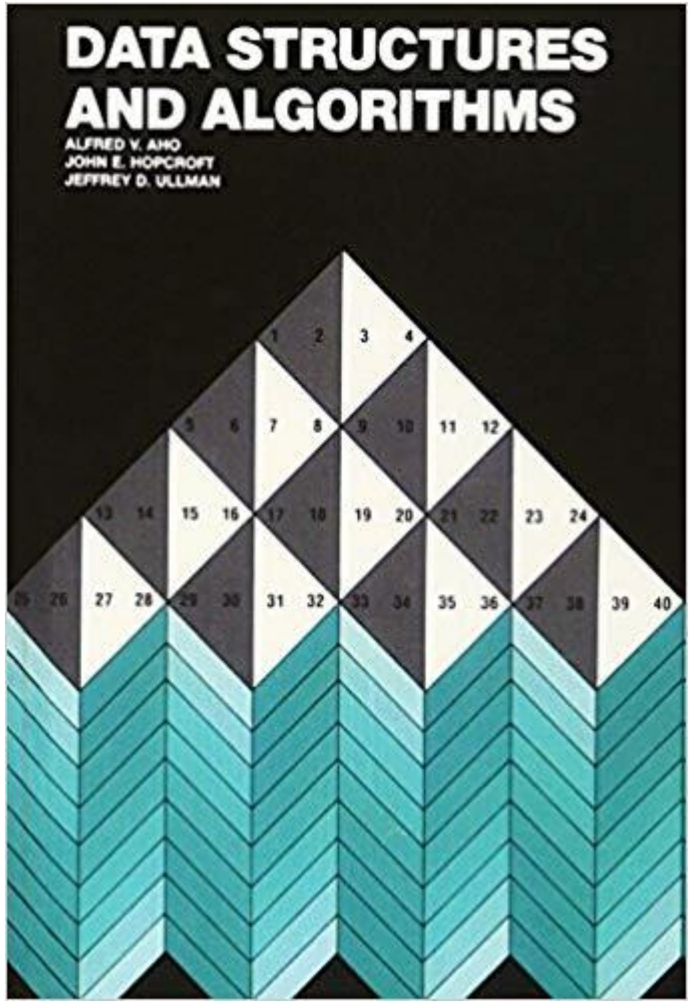
- This course serves as a one of the foundation courses in Computer Science.
- It provides students with an understanding of the principles of data structures and algorithms in the design and development of computer software.
- Students will learn basic data structures and its use in different algorithms that are commonly used in making structured and efficient software programs

Session Learning Outcomes

Upon completion of this session, students are expected to be able to

1. Describe the use of various data structures
2. Apply appropriate operations for maintaining common data structures.
3. Apply appropriate data structures and simple algorithms for solving computing problems.
4. Design computer programs by applying different data structures and related algorithms
5. Explain the efficiency of some basic algorithms
6. Design efficient software solutions that are appropriate for specific problems

Textbooks



- **Aho, A., Hopcroft, J, Ullman, J. (*1987*). Data Structures and Algorithms. Stanford, California: Addison-Wesley Publishing Company. ISBN:0-201-0023-7**
- Malik, D. S. (2010). Data Structures Using C++ (2nd Ed.). Cambridge, Massachusetts: Course Technology. ISBN-13: 978-0-324-78201-1
  - Cormen, T.H., Leiserson, C.E., Rivest, R.L., Stein, C. (2009). Introduction to Algorithms, 3rd Edition, Cambridge, Massachusetts: The MIT Press
  - Drozdek, A. (2000). Data Structures and Algorithms in C++, Second Edition 2nd Edition, Cambridge, Massachusetts: Course Technology
  - Weiss, M. A. (2007). Data structures and algorithm analysis in C++, 3/E, (3rd Ed.). Boston, Massachusetts: Addison Wesley
  - Gilberg, R. F., & Behrouz A. F. (2005). Data structures - A pseudocode approach with C (2nd Ed.). Boston, Massachusetts: Thomson Course Technology
  - Sedgewick, R. (1998). Algorithms in C++ (3rd Ed.). Boston, Massachusetts: Addison-Wesley.

SCHEDULE

Week	Topics
1.	- Introduction to Data Structures Design and Analysis of Algorithms
2.	- Basic Data Types
3.	- Stacks - **Final Project: Idea - consultation**
4.	- Queues
5.	- Trees
6.	- Sets - Quiz
7.	- Advanced Set Representation - **Final Project: Proposal - presentation**
8.	- Directed Graphs
9.	- Undirected Graphs
10.	- Sorting and Searching
11.	- Algorithm Analysis Techniques
12.	- Algorithm Design Techniques - Quiz
13.	- **Final Project Presentation**

BINUS UNIVERSITY INTERNATIONAL

Assessment

No.	Components	Percentage	Course Intended Learning Outcomes
1.	Quizzes	20 %	LO 1, LO 2, LO 3
2.	Assignments	10 %	LO 1, LO 2
3.	Laboratory	20 %	LO 2, LO 3, LO 4
4.	Project	30 %	LO 2, LO 3, LO 4, LO 6
5.	Final Examination	20 %	LO 1, LO 2, LO 3, LO 5

Project (Final Project)

- To be announced (TBA)

Questions?

Form The Groups

- 2-3 students (max. 3)
- Group:
  - Final Project
  - Group assignment(s)
  - Discussions

Weekly Group Presentation

- https://docs.google.com/spreadsheets/d/16CMSU7_3qU8NjYr-1wBx3KMNojSjOtwphqARWofclvM/edit?usp=sharing

Topics

- Introduction to Data Structures
- Introduction to Analysis of Algorithm

Introduction

- **Without software a computer is of no use**
- It is the software that enables you to do things that were, perhaps, fiction a few years ago

- However, **software is not created overnight**.
- From the time a software program is conceived until it is delivered, it goes through several phases.
- There is a branch of computer science, called software engineering, which specializes in this area

Software Life Cycle

- A program goes through many phases from the time it is first conceived until the time it is retired, called the ***life cycle*** of the program.
- The three fundamental stages through which a program goes are:
  - **development,**
  - **use,**
  - and **maintenance**.
    - the program is modified to fix the (identified) problems and/or to enhance it

When a program is considered too expensive to maintain, the developer might decide to retire the program and no new version of the program will be released.

Software Development Phase

- The software development phase is the first and perhaps most important phase of the software life cycle.
- A program that is well developed will be **easy and less expensive to maintain**.
- Software engineers typically break the software development process into the following four phases:
  - Analysis
  - Design
  - Implementation
  - Testing and debugging

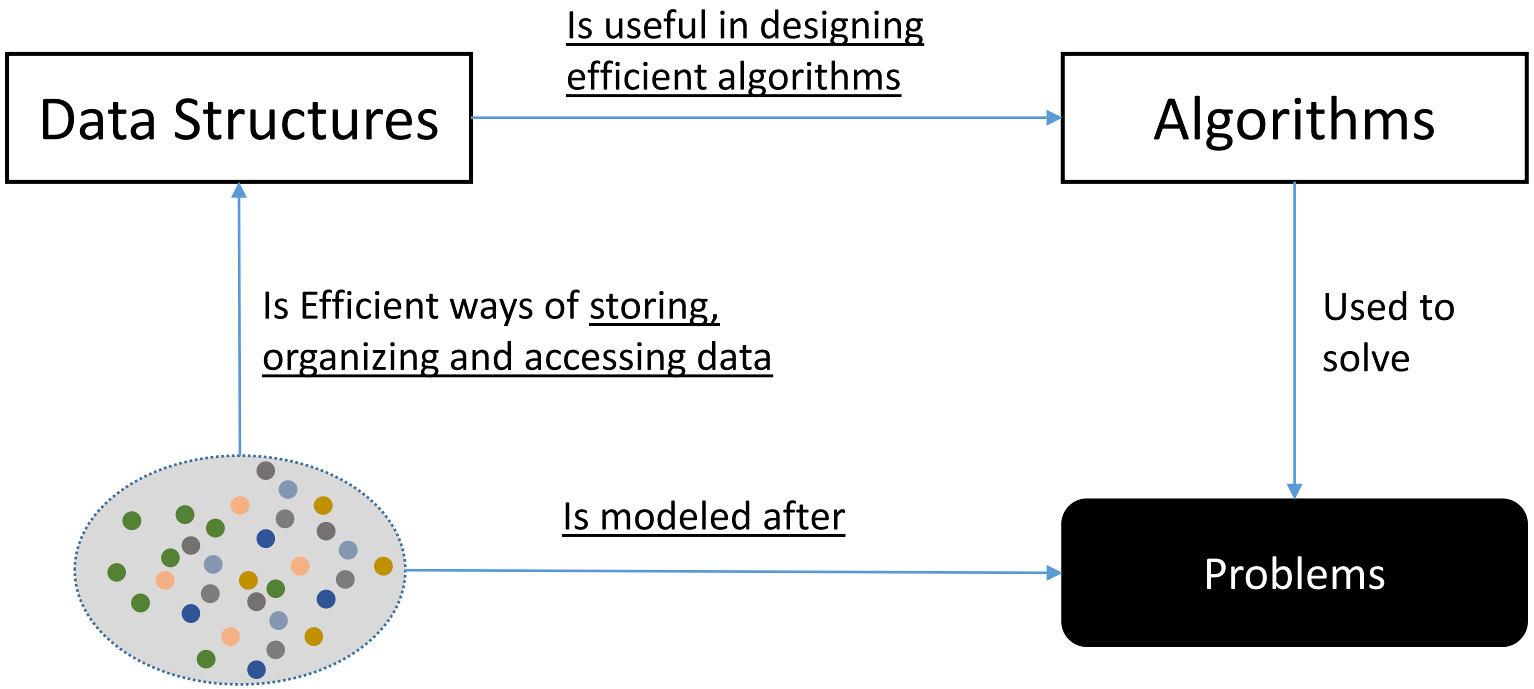
Discussion – 10 minutes

- What do you think of these phases (what you need to do?)
  - Analysis
  - Design
  - Implementation
  - Testing and debugging

Data Structures vs Algorithms

- **Data Structures**
  - Efficient ways of storing, organizing and accessing data
  - “how to store, organize and access data”
- **Algorithms**
  - Step by step method to solve a problem
  - “how to solve problem using step by step method”

- So, eventhough Data Structures often comes together with Algorithms, such as Data Structures and Algorithms, but
  - BY ABSTRACTION, Data Structures is different than Algorithms
  - Data structures are useful in designing efficient algorithms



Data Structures

Is useful in designing efficient algorithms

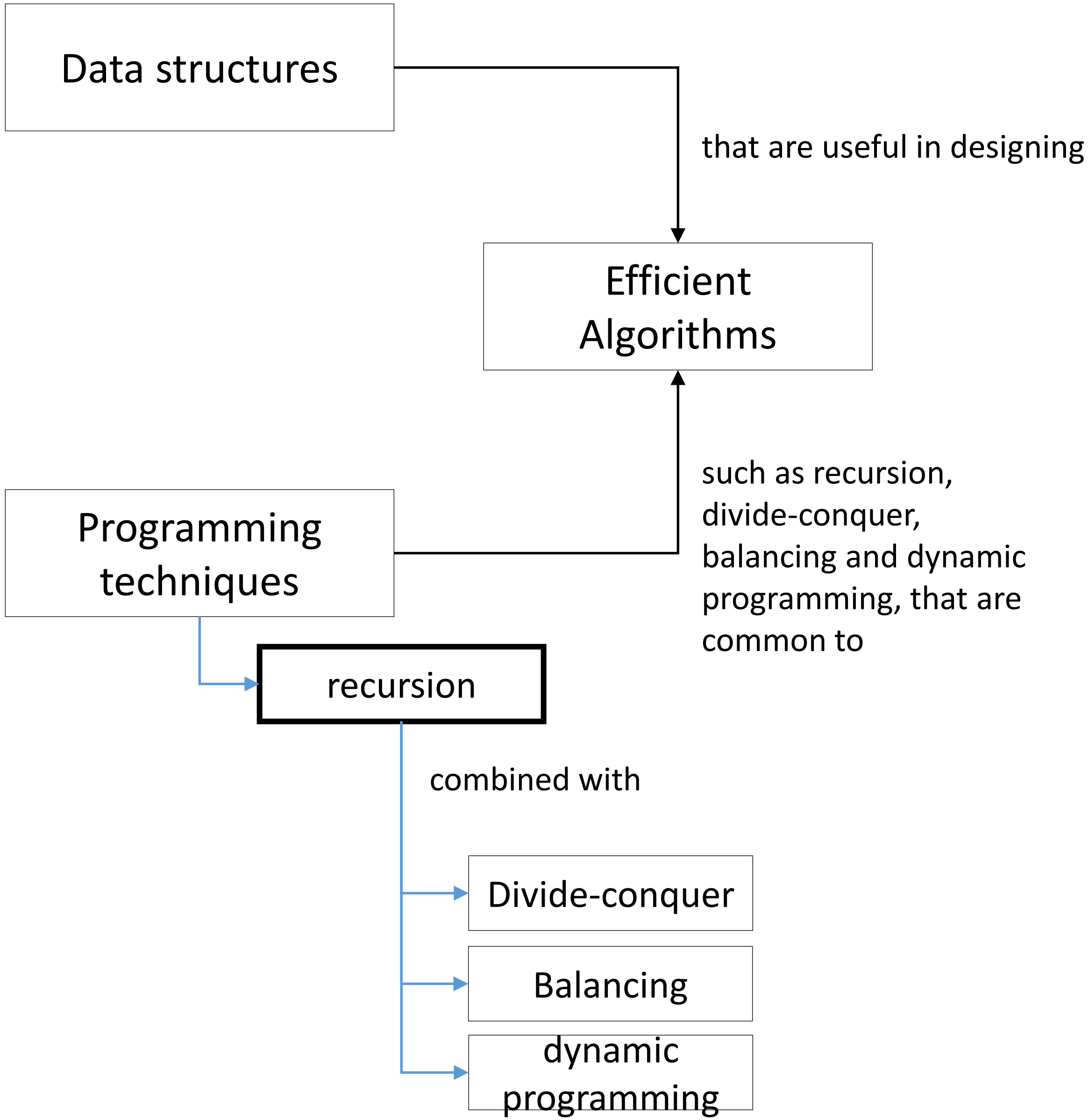
Algorithms

Used to solve

Is Efficient ways of storing, organizing and accessing data

Is modeled after

Problems



Data structures

that are useful in designing

Efficient Algorithms

Programming techniques

such as recursion, divide-conquer, balancing and dynamic programming, that are common to

recursion

combined with

Divide-conquer

Balancing

dynamic programming

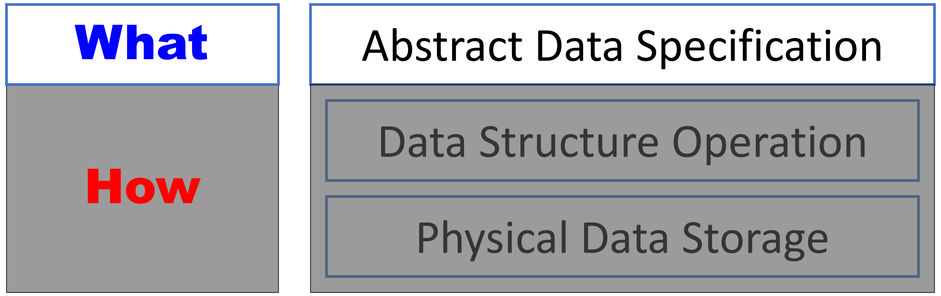
From Problem to Program

- 6 steps that you should follow in order to solve a problem:
  1. Understand the Problem
  2. Formulate a Model
  3. Develop an Algorithm
  4. Write the Program
  5. Test the Program
  6. Evaluate the Solution

Abstract Data Types

- Definitions
  - An **abstract data type (ADT) is** a collection of related data items together with basic relations between them and operations to be performed on them.
- Why "abstract?" Data, operations, and relations are studied **independently of how they are going to be implemented**.

What not How



What

How

Abstract Data Specification

Data Structure Operation

Physical Data Storage

Example of ADT: List

- **List**: A collection of elements of the same type.
- The **<u>length</u>** of a list is the number of elements in the list.
- **<u>Operations</u>** on a list:
  - Create the list. The list is initialized to an empty state.
  - Determine whether the list is empty.
  - Determine whether the list is full.
  - Find the size of the list.
  - Destroy, or clear, the list.
  - Determine whether an item is the same as a given list element.
  - Insert an item in the list at the specified location.
  - Remove an item from the list at the specified location.
  - Replace an item at the specified location with another item.
  - Retrieve an item from the list from the specified location.
  - Search the list for a given item.

An implementation of an ADT

- An implementation of an ADT consists of storage structures (commonly called data structures) to store the data items and algorithms for the basic operations and relations.
- ADT is a way of thinking about Data Abstraction: Separating the definition of a data type from its implementation.
    - An important concept in software design.
    - Usually the storage structures / data structures used in implementation are those provided in a language or built from them.

Good reference:
https://www.cs.odu.edu/~zeil/cs330/latest/Public/implementingADTS/index.html

ADT and Data Structures

- The terms abstract data type and data structure are often used interchangeably.
- However, we will **use abstract data type when data is studied at a logical or conceptual level**, independent of any programming language or machine considerations.
- The term **data structure refers to a construct in some programming language** that can be used to store data.

Header Files and Template

- Traditionally, definitions of member functions have been put in an implementation file ClassName.cpp corresponding to the class' header file.
- This is done to enforce **data abstraction — separating the interface of the ADT from its implementation details.**
  - (Unfortunately, the class data members, which store data and are therefore part of the implementation, must be in the .h file.)
- With the increasing use of **templates, however, this practice is becoming less common** because current compiler technology doesn't permit this split for templates — everything has to be in the same file.
  - Thus the reason for dropping the ".h" from standard class libraries.
  - They're really class-template libraries, and there are therefore no corresponding ".cpp" files.

Steps of Working with Data Structures

- Step 1:
  - Understand the problem and think about the suitable data structure
- Step 2:
  - Define the ADT, that is the specification of the data structure (valid value and operation)
- Step 3:
  - Select the low-level storage: Arrays or Linked List
- Step 4:
  - Implement the ADT (the structured data and operation on the data)
- Step 5:
  - Analyze the algorithm complexity of the operations (worst case analysis)

Algorithm Analysis

- Just as a problem is analyzed before writing the algorithm and the computer program, after **an algorithm is designed it should also be analyzed**.
  - Usually, there are various ways to design a particular algorithm.
    - **Speed**: certain algorithms take very little computer time to execute, whereas others take a considerable amount of time.
    - **Space**: certain algorithms take more memory to store the data while executing the program, whereas others take a less memory.

Algorithm Analysis: The Big-O Notation

- Just as a problem is analyzed before writing the algorithm and the computer program, after an algorithm is designed it should also be analyzed.
- Usually, there are various ways to design a particular algorithm.
- Certain algorithms take very little computer time to execute, whereas others take a considerable amount of time.

What is Big O Notation.mp4

Algorithm Analysis: The Big-O Notation

- 50 packages are to be delivered to 50 different houses.
- The shop, while making the route, finds that the 50 houses are one mile apart and are in the same area.

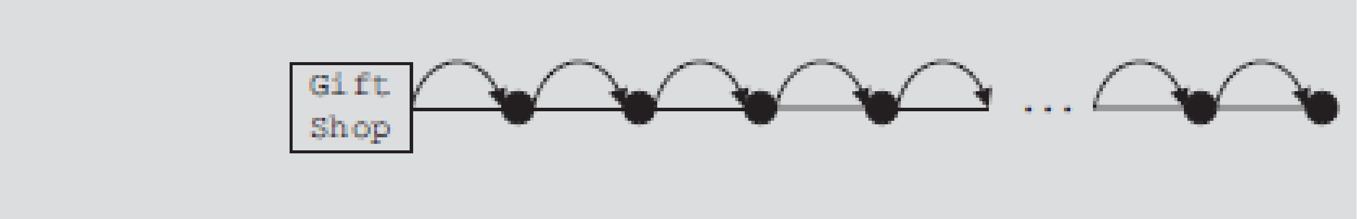


FIGURE 1-2 Package delivering scheme

Therefore, the total distance traveled by the driver to deliver the packages and then getting back to the shop is:
50 + 50 = 100 miles

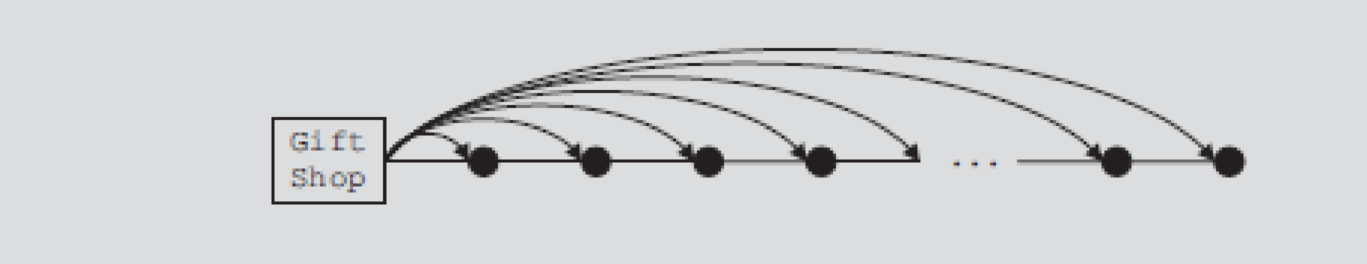


FIGURE 1-3 Another package delivery scheme

2 . (1+2+3+…+50) = 2550 miles

Algorithm Analysis: The Big-O Notation

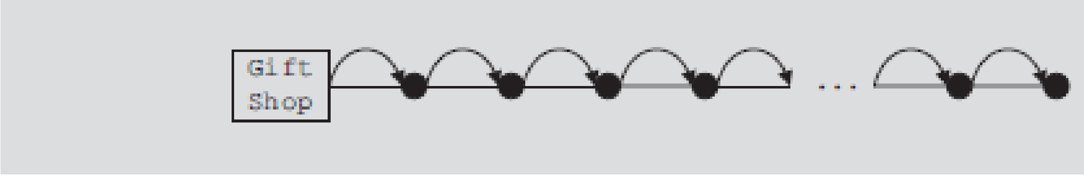


FIGURE 1-2 Package delivering scheme

Therefore, the total distance traveled by the driver to deliver the packages and then getting back to the shop is:
50 + 50 = 100 miles

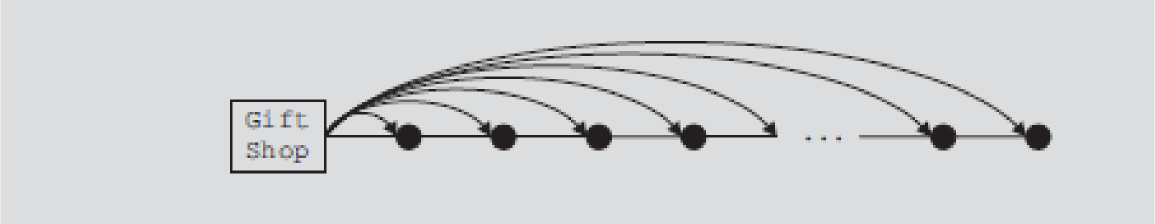
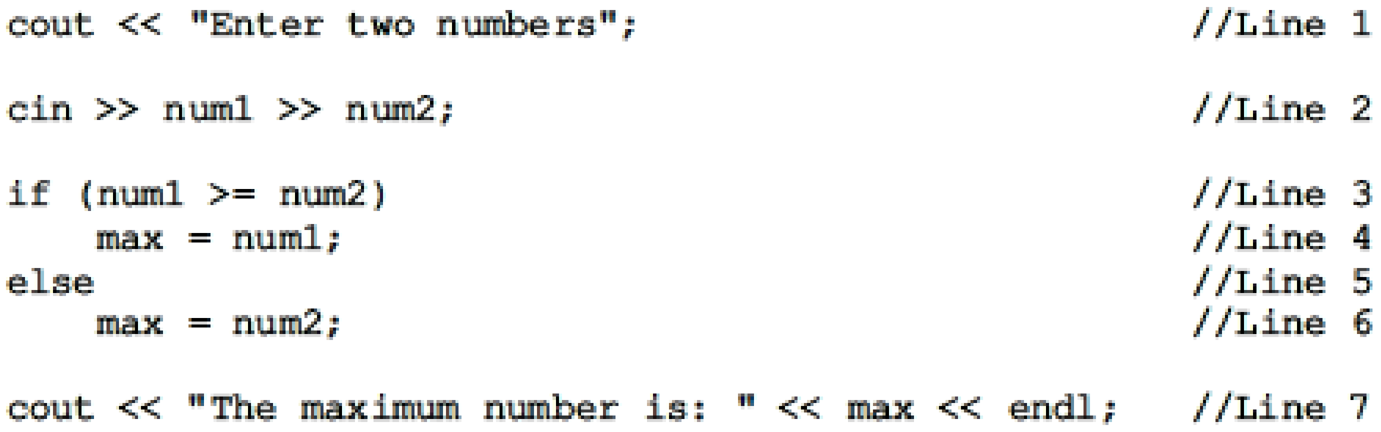


FIGURE 1-3 Another package delivery scheme

2 . (1+2+3+…+50) = 2550 miles

- First method = 1 + 1 + … + 1 + n = 2n
- Second method = 2 . (1+2+3+…+n) = 2 . (n(n+1)/2) = n^2+n

Algorithm Analysis: The Big-O Notation - Example



```
cout << "Enter two numbers";                          //Line 1

cin >> num1 >> num2;                                  //Line 2

if (num1 >= num2)                                     //Line 3
    max = num1;                                       //Line 4
else                                                  //Line 5
    max = num2;                                       //Line 6

cout << "The maximum number is: " << max << endl;     //Line 7
```

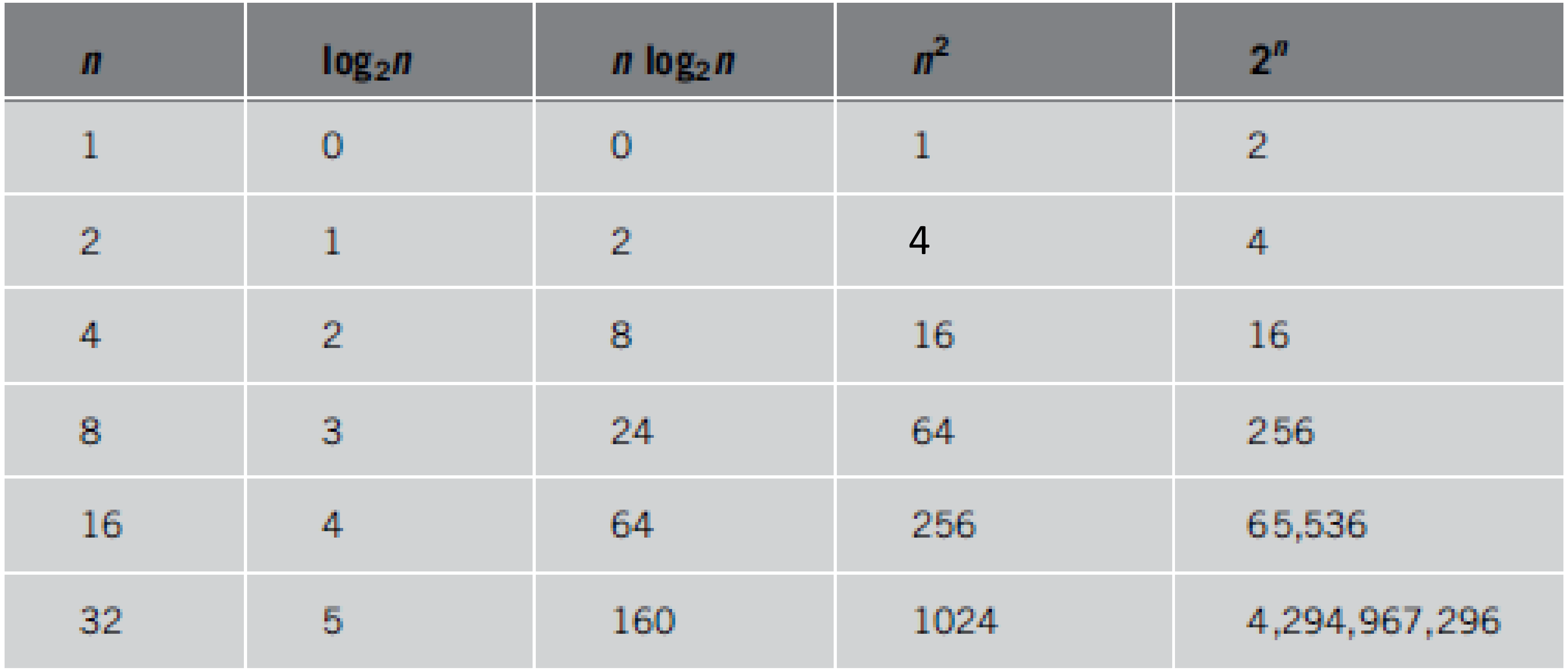
Assume that all variables are properly declared

- Line 1 has one operation, <<;
- Line 2 has two operations;
- Line 3 has one operation, >=;
- Line 4 has one operation, =;
- Line 6 has one operation;
- and Line 7 has three operations.
- Either Line 4 or Line 6 executes.
- Therefore, the total number of operations executed in the preceding code is 1 + 2 + 1 + 1 + 3 = 8. In this algorithm, the number of operations executed is fixed.

Algorithm Analysis: The Big-O Notation

f(n)

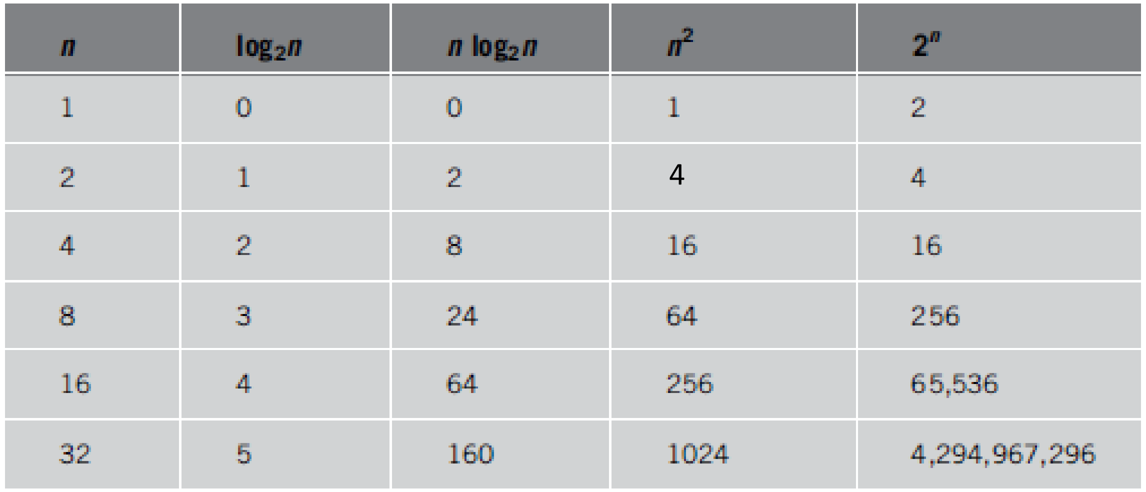
TABLE 1-2 Growth rates of various functions



n	$\log_2 n$	$n \log_2 n$	n^2	2^n
1	0	0	1	2
2	1	2	4	4
4	2	8	16	16
8	3	24	64	256
16	4	64	256	65,536
32	5	160	1024	4,294,967,296

Algorithm Analysis: The Big-O Notation

TABLE 1-2 Growth rates of various functions



n	$\log_2 n$	$n \log_2 n$	n^2	2^n
1	0	0	1	2
2	1	2	4	4
4	2	8	16	16
8	3	24	64	256
16	4	64	256	65,536
32	5	160	1024	4,294,967,296

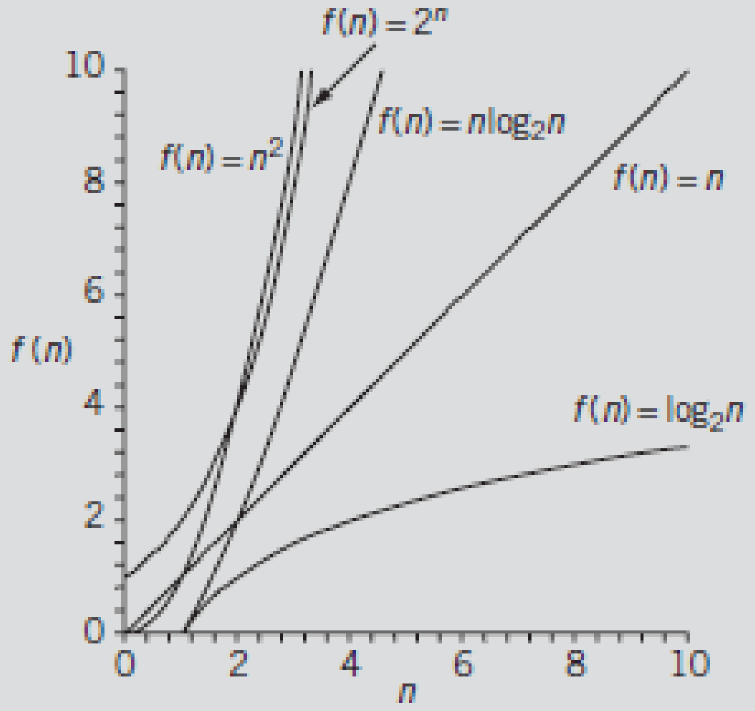


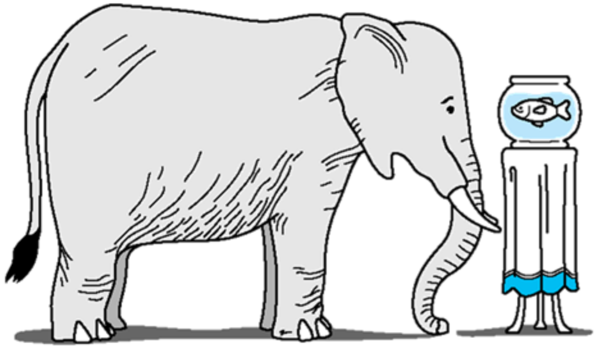
FIGURE 1-4 Growth rate of various functions

The Big-O Notation

- **Big-O notation** is a notation that **expresses computing time (complexity)** as the term in a **function** that increases most rapidly relative to the size of a problem.

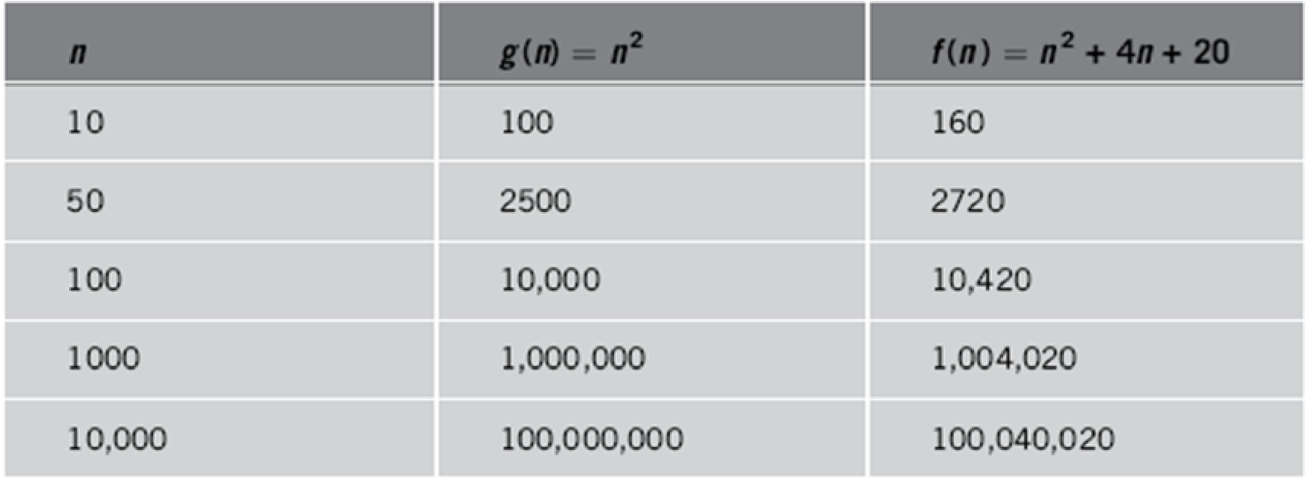
  $f(n) = n^4 + 100n^2 + 10n + 50$

- f (n) is of order n^4—or, in **Big-O notation**, O(n^4).
  - When we compare algorithms using Big-O, we are concerned with what happens when ***n* is “large” (asymptotic)**



Dominant Operations

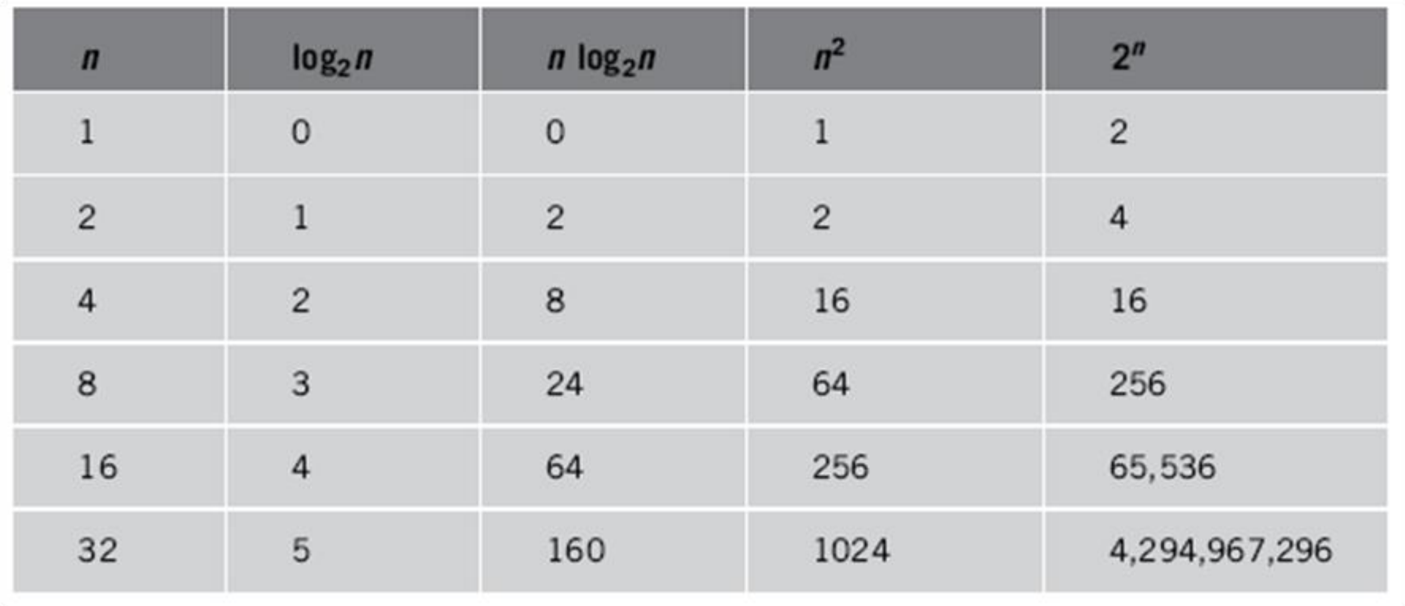
- While analyzing a particular algorithm, we usually count the number of operations performed by the algorithm.
- Usually, in an algorithm, **certain operations are dominant**.
- Consider the functions $g(n) = n^2$ and $f(n) = n^2 + 4n + 20$.
  - Clearly, the function g does not contain any linear term, that is, the coefficient of n in g is zero
  - As n becomes larger and larger the term $4n + 20$ in $f(n)$ becomes insignificant, and the term n^2 becomes the dominant term.



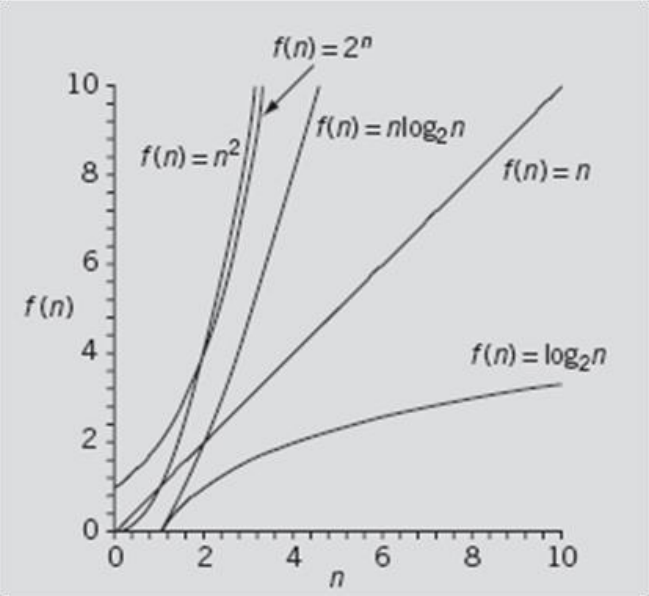
n	$g(n) = n^2$	$f(n) = n^2 + 4n + 20$
10	100	160
50	2500	2720
100	10,000	10,420
1000	1,000,000	1,004,020
10,000	100,000,000	100,040,020

Asymptotic Growth Rate

- For large values of n, we can predict the behavior of *f*(*n*) by looking at the behavior of *g*(*n*).
- In algorithm analysis, if the complexity of a function can be described by the complexity of a quadratic function without the linear term, we say that the function is of **O(n^2)**, called Big-O of n^2.

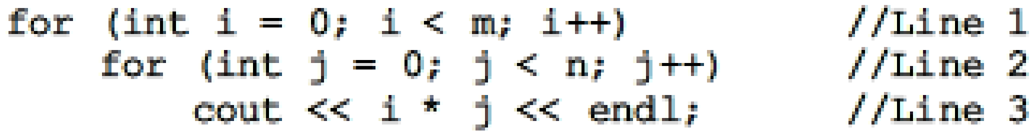


n	$\log_2 n$	$n \log_2 n$	n^2	2^n
1	0	0	1	2
2	1	2	2	4
4	2	8	16	16
8	3	24	64	256
16	4	64	256	65,536
32	5	160	1024	4,294,967,296



Algorithm Analysis: The Big-O Notation

- Exercise – 15 minutes
  - Consider the following code, where m and n are int variables and their values are nonnegative:



```
for (int i = 0; i < m; i++)           //Line 1
    for (int j = 0; j < n; j++)       //Line 2
        cout << i * j << endl;        //Line 3
```

  - Big-O?

Algorithm Analysis: The Big-O Notation

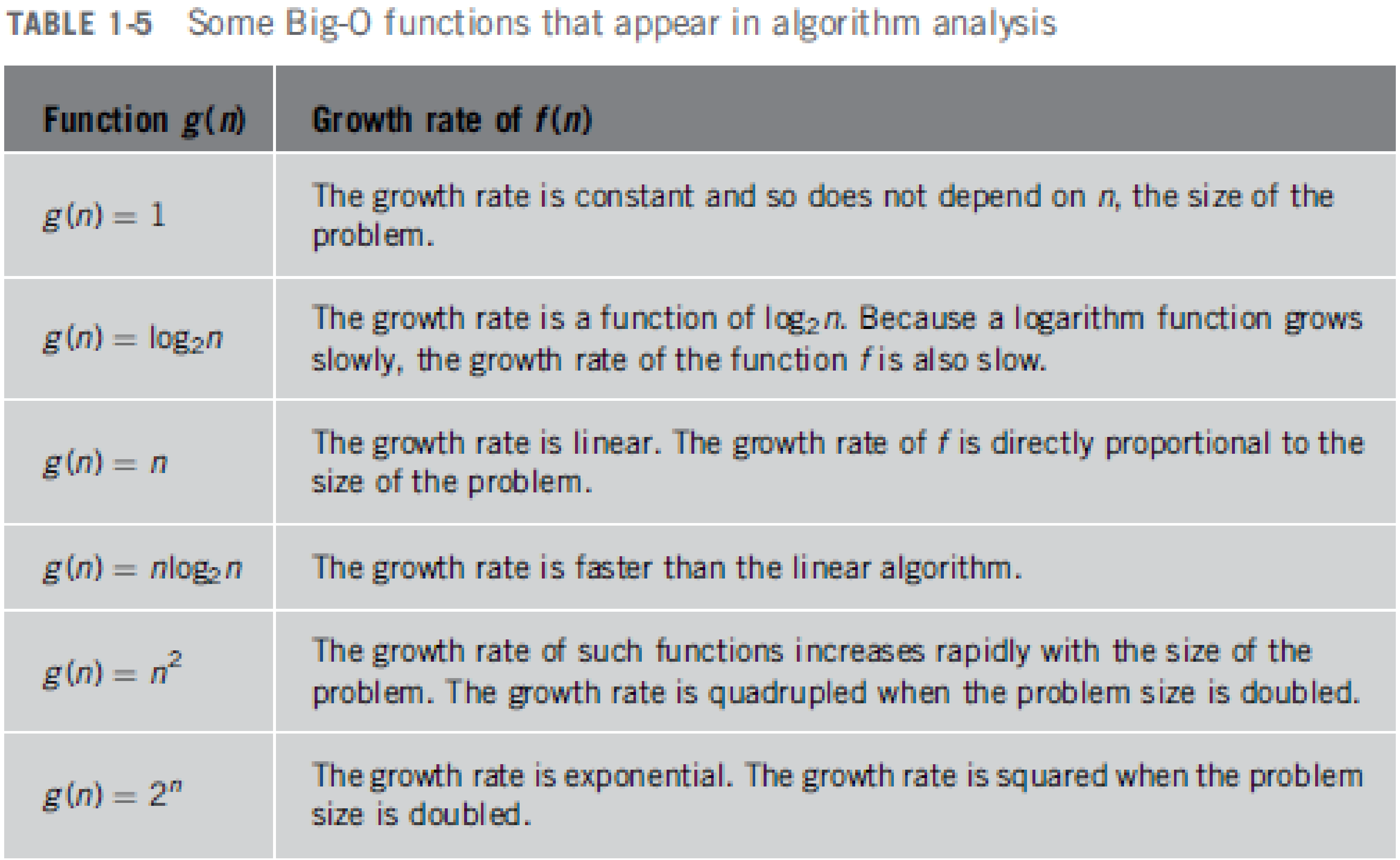


TABLE 1-5 Some Big-O functions that appear in algorithm analysis

Function $g(n)$	Growth rate of $f(n)$
$g(n) = 1$	The growth rate is constant and so does not depend on n, the size of the problem.
$g(n) = \log_2 n$	The growth rate is a function of $\log_2 n$. Because a logarithm function grows slowly, the growth rate of the function f is also slow.
$g(n) = n$	The growth rate is linear. The growth rate of f is directly proportional to the size of the problem.
$g(n) = n\log_2 n$	The growth rate is faster than the linear algorithm.
$g(n) = n^2$	The growth rate of such functions increases rapidly with the size of the problem. The growth rate is quadrupled when the problem size is doubled.
$g(n) = 2^n$	The growth rate is exponential. The growth rate is squared when the problem size is doubled.

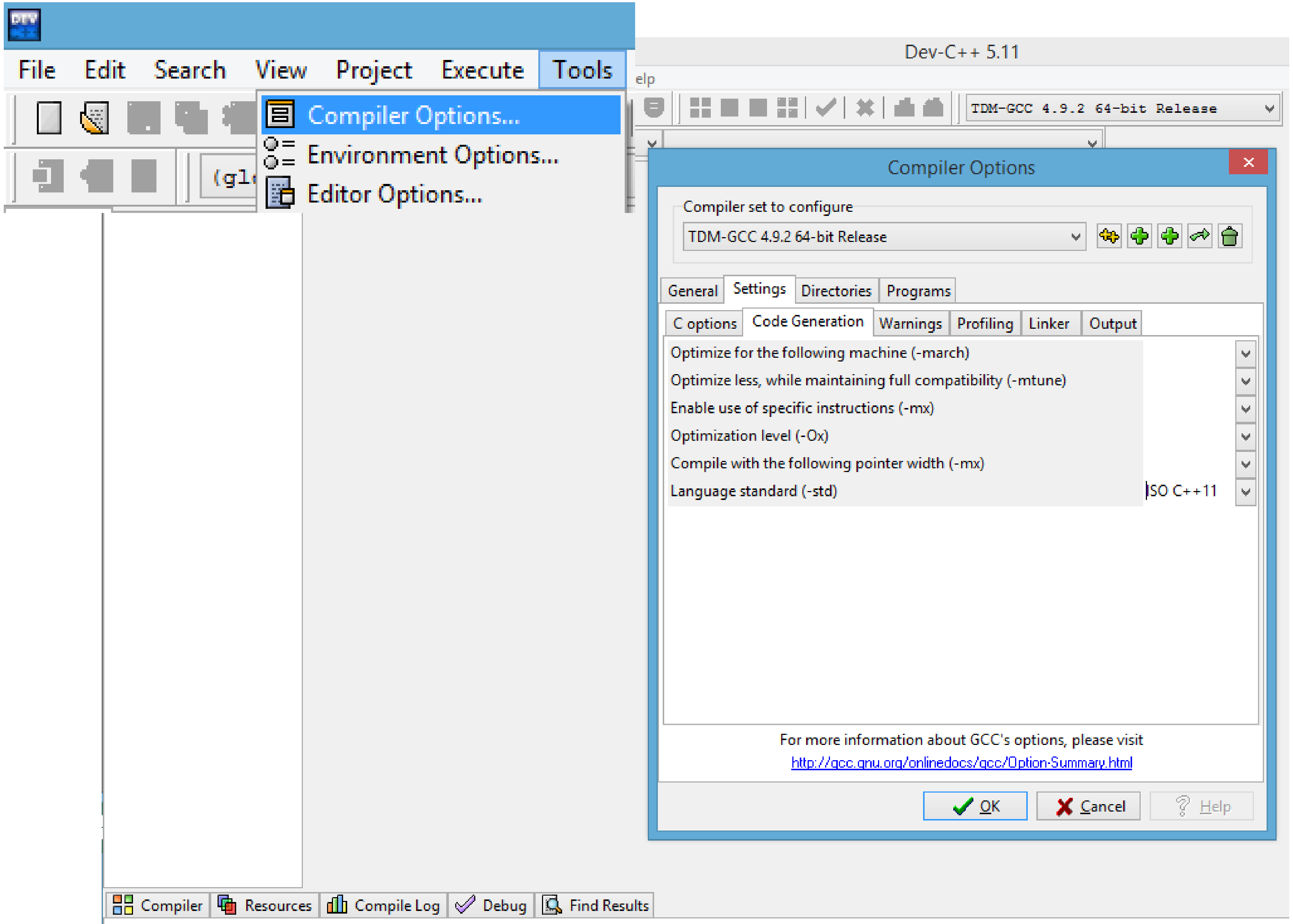
BINUS UNIVERSITY INTERNATIONAL

CPP / C++

IDE

- Dev-C++
  (download: https://sourceforge.net/projects/orwelldevcpp/)
- NetBeans
- Visual Studio (Express)
- Eclipse
- Xcode
- Etc…

Dev-C++ (choose language standard)



Compiler set to configure: TDM-GCC 4.9.2 64-bit Release

Optimize for the following machine (-march)

Optimize less, while maintaining full compatibility (-mtune)

Enable use of specific instructions (-mx)

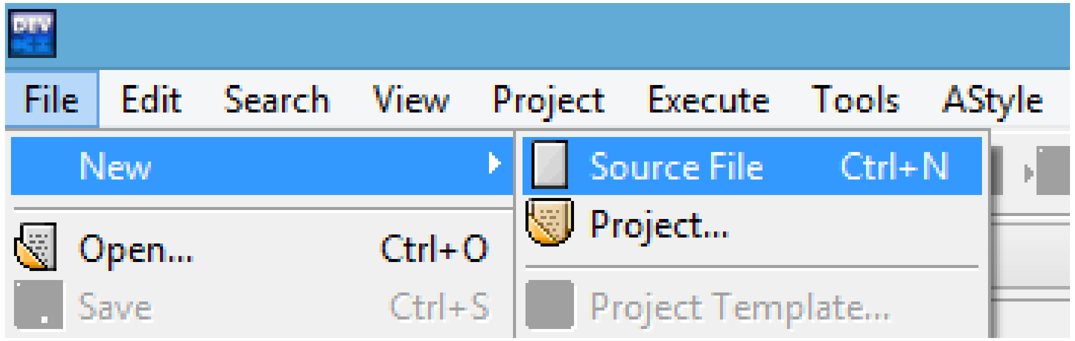
Optimization level (-Ox)

Compile with the following pointer width (-mx)

Language standard (-std) ISO C++11

For more information about GCC's options, please visit
http://gcc.gnu.org/onlinedocs/gcc/Option-Summary.html

Develop your own C++



stream	description
cin	standard input stream
cout	standard output stream
cerr	standard error (output) stream
clog	standard logging (output) stream

```
cout << "Hello"; // prints Hello
cout << Hello; // prints the content of variable Hello
```

```
int age;
cin >> age;
```

http://www.cplusplus.com/doc/tutorial/basic_io/

Develop your own C++

Case sensitive

- Program Structure

```
// my first C++
#include <iostream>
int main()
{
   ...........
}
```

Single line comment

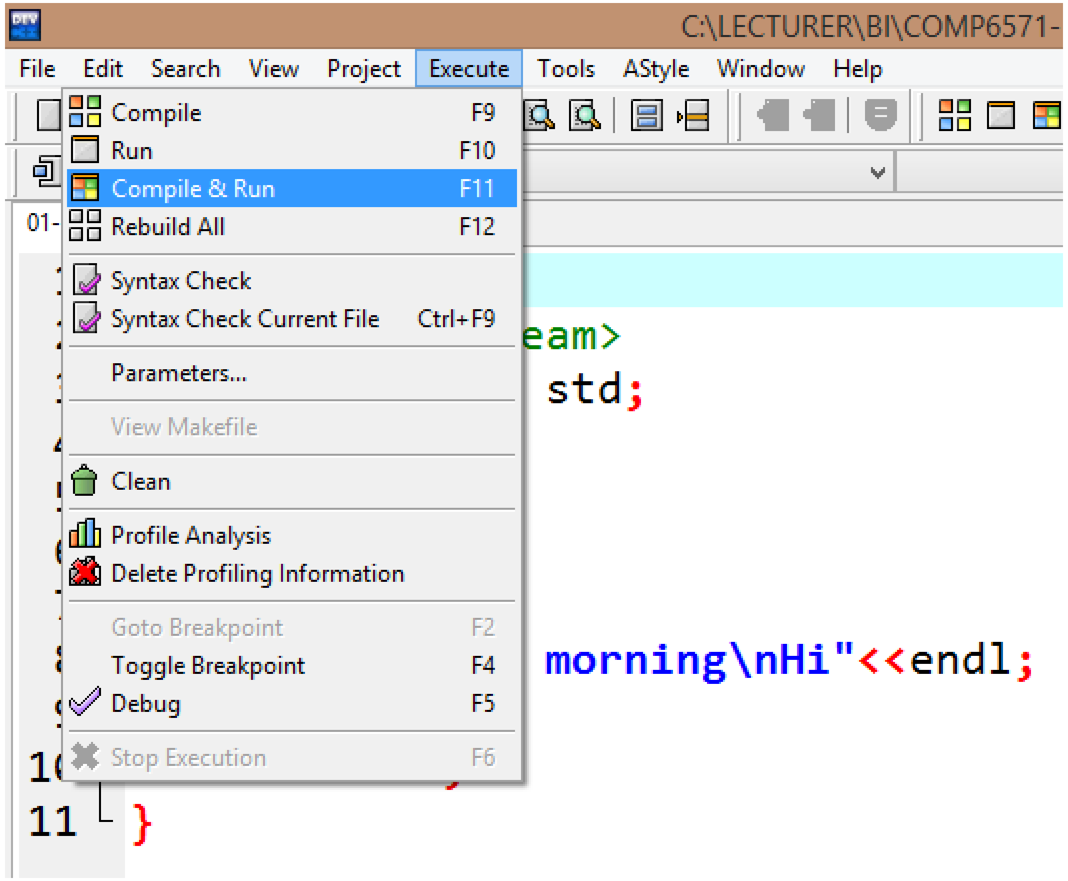
http://www.cplusplus.com/doc/tutorial/program_structure/

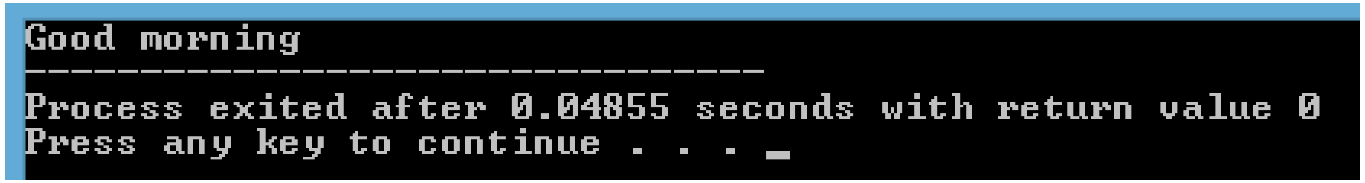
Example 1

#include <iostream>

Lines beginning with a hash sign (#) are directives read and interpreted by what is known as the *preprocessor*. They are special lines interpreted before the compilation of the program itself begins. In this case, the directive #include <iostream>, instructs the preprocessor to include a section of standard C++ code, known as *header iostream*, that allows to perform standard input and output operations

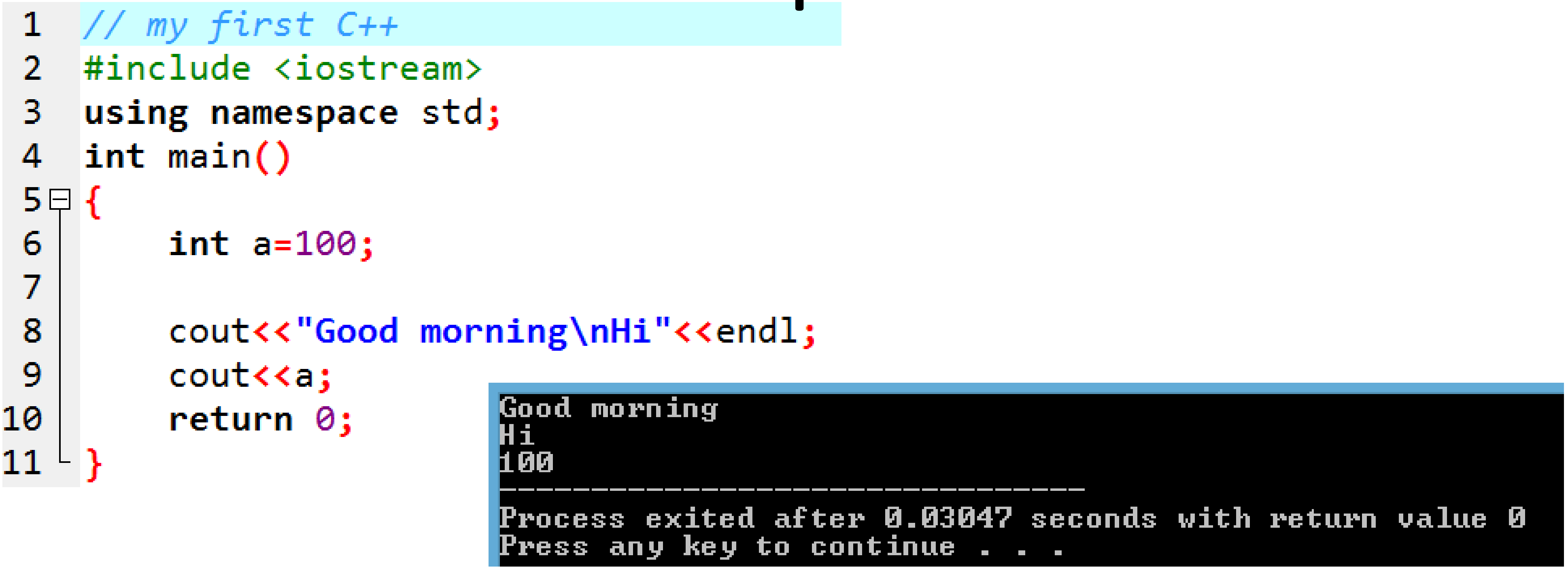
```
// my first C++
#include <iostream>
int main()
{
  std::cout<<"Good morning";
}
```





```
Good morning
--------------------------------
Process exited after 0.04855 seconds with return value 0
Press any key to continue . . . _
```

Example 2



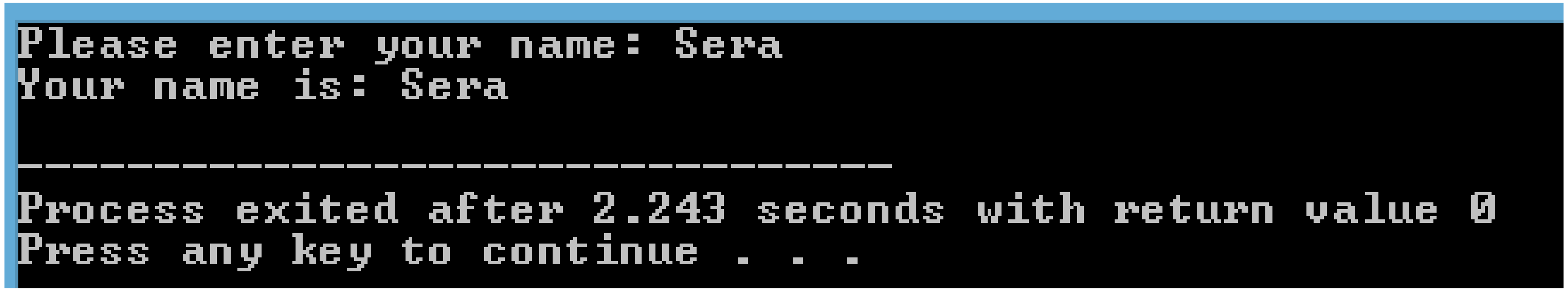
```cpp
// my first C++
#include <iostream>
using namespace std;
int main()
{
    int a=100;

    cout<<"Good morning\nHi"<<endl;
    cout<<a;
    return 0;
}
```

```
Good morning
Hi
100
--------------------------------
Process exited after 0.03047 seconds with return value 0
Press any key to continue . . .
```

Why??? What it means?

- Using namespace std;
- \n
- endl
- return 0;

Example 3

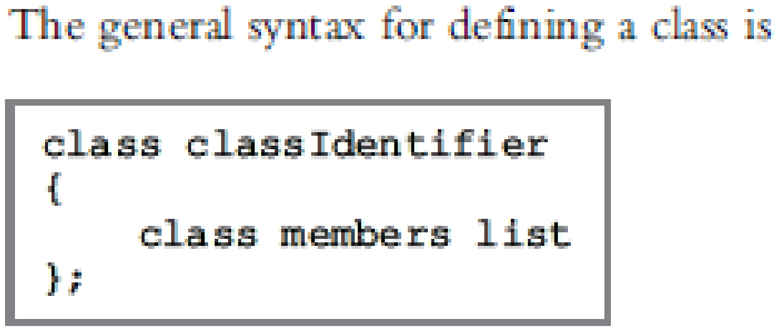


```
Please enter your name: Sera
Your name is: Sera

--------------------------------
Process exited after 2.243 seconds with return value 0
Press any key to continue . . .
```

Classes

Classes

- OOD (OO Design), the first step is to identify the components called objects; an object combines data and the operations on that data in a single unit, called encapsulation.
- In C++, the mechanism that allows you to combine data and the operations on that data in a single unit is called a class.
- A class is a collection of a fixed number of components.
- The components of a class are called the members of the class.

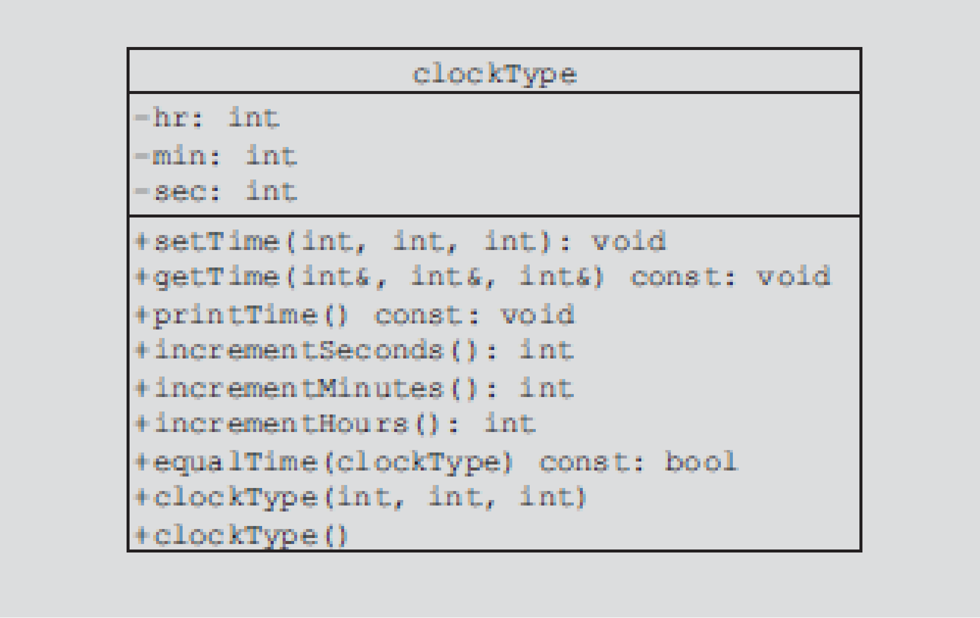


The general syntax for defining a class is

```
class classIdentifier
{
    class members list
};
```

Unified Modelling Language Diagrams

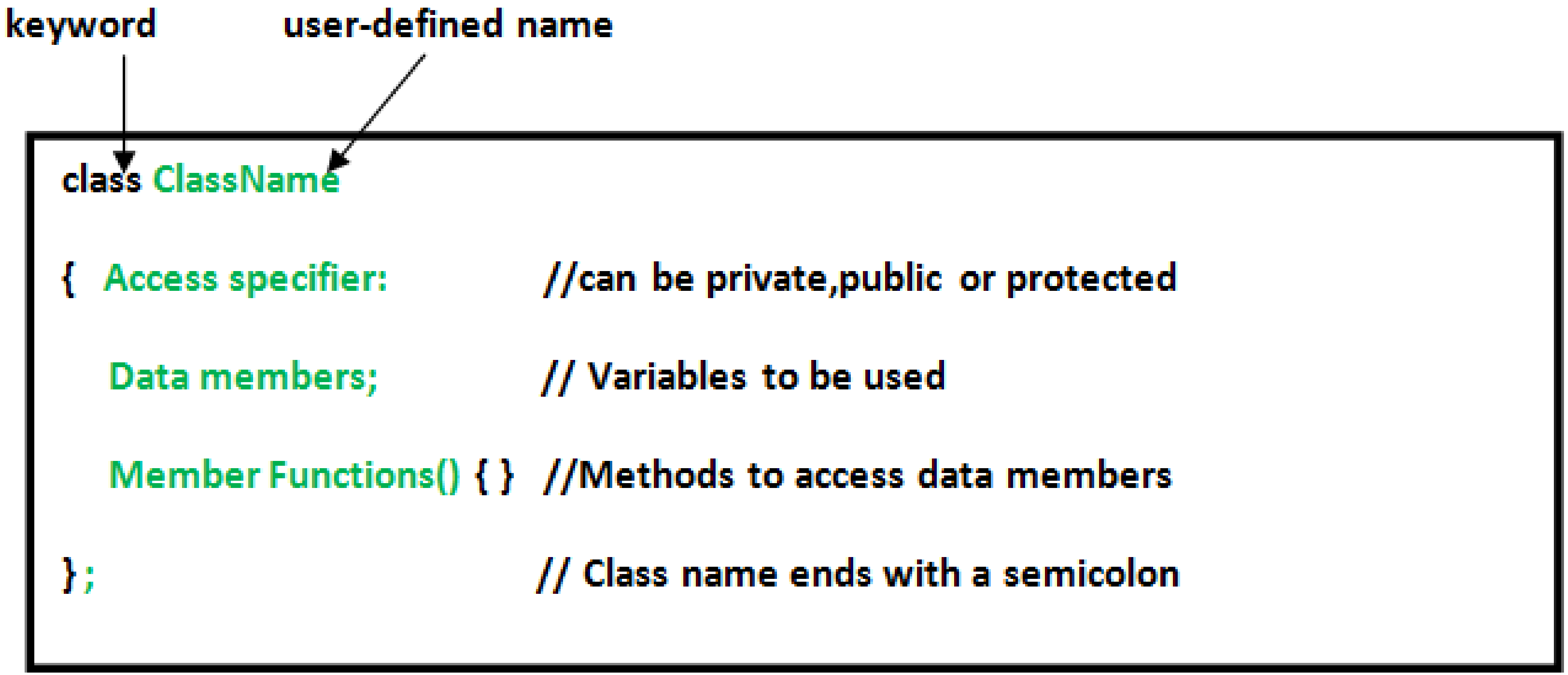
- A class and its members can be described graphically using a notation known as Unified Modeling Language (UML) notation.
- For example, Figure 1-5 shows the UML class diagram of the class clockType.



```
clockType
-hr: int
-min: int
-sec: int
+setTime(int, int, int): void
+getTime(int&, int&, int&) const: void
+printTime() const: void
+incrementSeconds(): int
+incrementMinutes(): int
+incrementHours(): int
+equalTime(clockType) const: bool
+clockType(int, int, int)
+clockType()
```

FIGURE 1-5 UML class diagram of the **class** clockType

Class

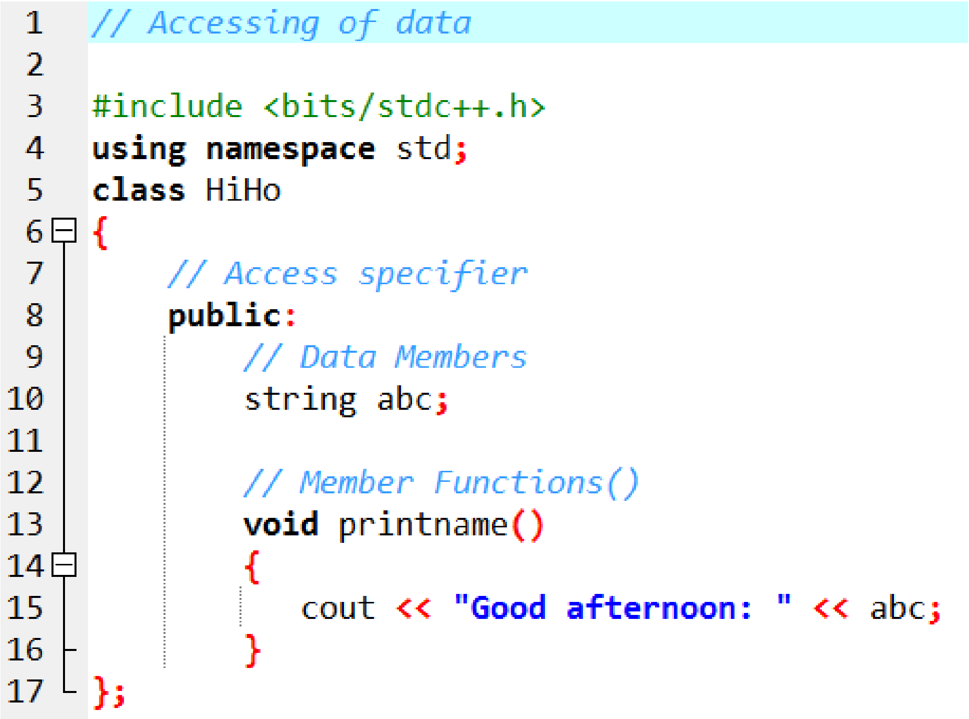


```
class ClassName

{   Access specifier:           //can be private,public or protected

    Data members;               // Variables to be used

    Member Functions() { }     //Methods to access data members

};                              // Class name ends with a semicolon
```

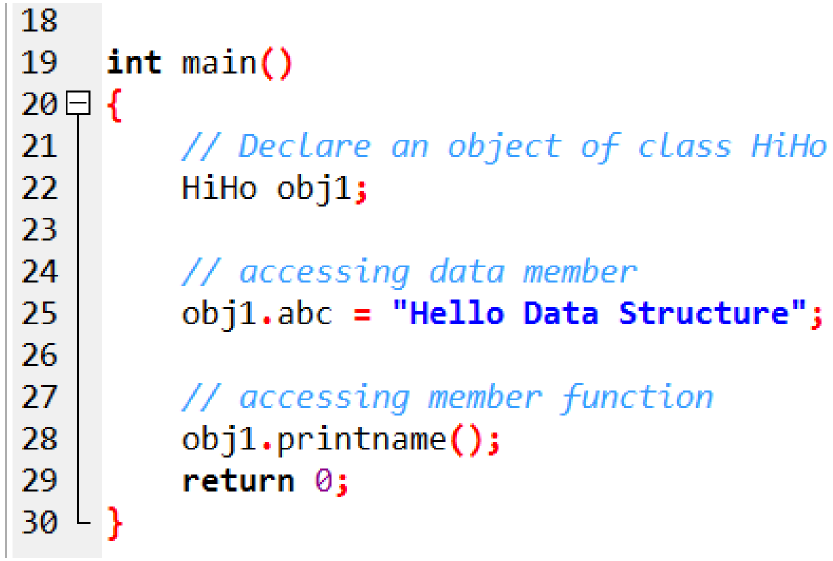
https://www.geeksforgeeks.org/c-classes-and-objects/

https://www.geeksforgeeks.org/bitsstdc-h-c/

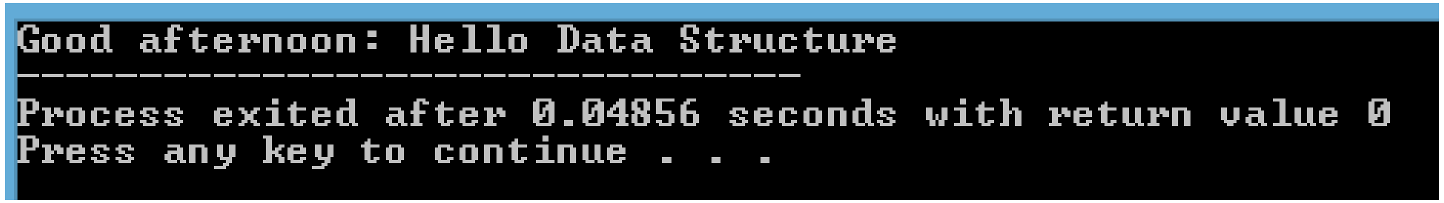
Example 4



```cpp
// Accessing of data

#include <bits/stdc++.h>
using namespace std;
class HiHo
{
    // Access specifier
    public:
        // Data Members
        string abc;

        // Member Functions()
        void printname()
        {
            cout << "Good afternoon: " << abc;
        }
};
```



```cpp

int main()
{
    // Declare an object of class HiHo
    HiHo obj1;

    // accessing data member
    obj1.abc = "Hello Data Structure";

    // accessing member function
    obj1.printname();
    return 0;
}
```



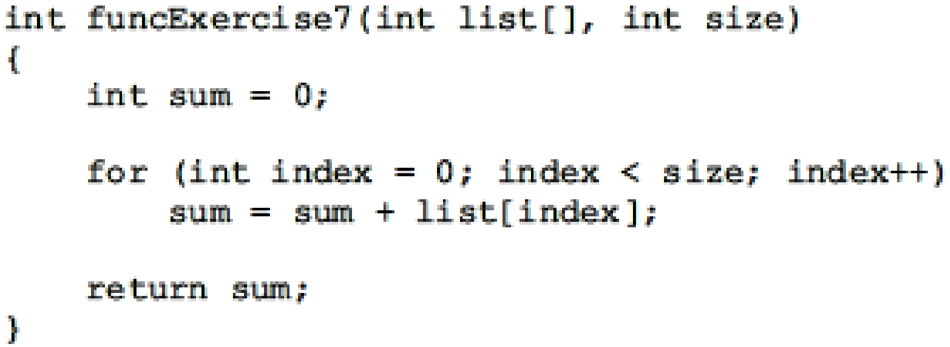
```
Good afternoon: Hello Data Structure
--------------------------------
Process exited after 0.04856 seconds with return value 0
Press any key to continue . . .
```

Further readings …….

- OO
  - Private
  - Protected
  - Public
- Constructors
- Structs
- Destructors
- Data abstraction
  - ADT (abstract data type)
    - domain
    - set of operations
- Identifying Classes, Objects, and Operations

Exercise

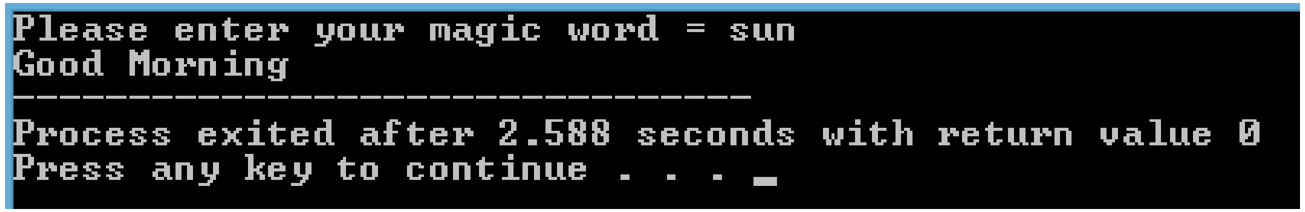
1. Consider the following function



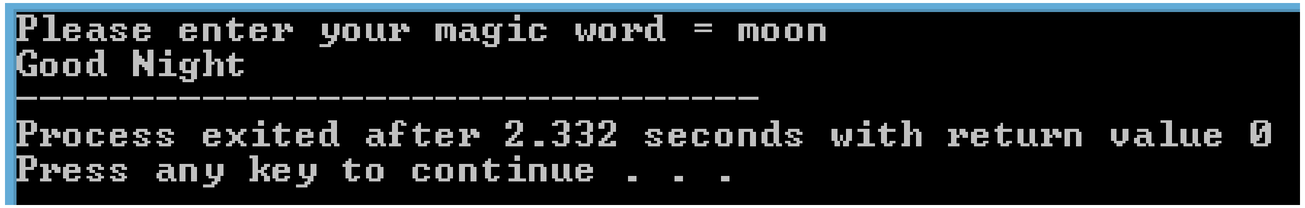
```
int funcExercise7(int list[], int size)
{
    int sum = 0;

    for (int index = 0; index < size; index++)
        sum = sum + list[index];

    return sum;
}
```

Find the number of operations executed by the function funcExercise7 if the value of size is 10!

2. Please create C++ code using Class to create this output
   - Enter “sun”, output = “Good Morning”
   - Enter “moon”, output = “Good Night”



```
Please enter your magic word = sun
Good Morning
--------------------------------
Process exited after 2.588 seconds with return value 0
Press any key to continue . . .
```



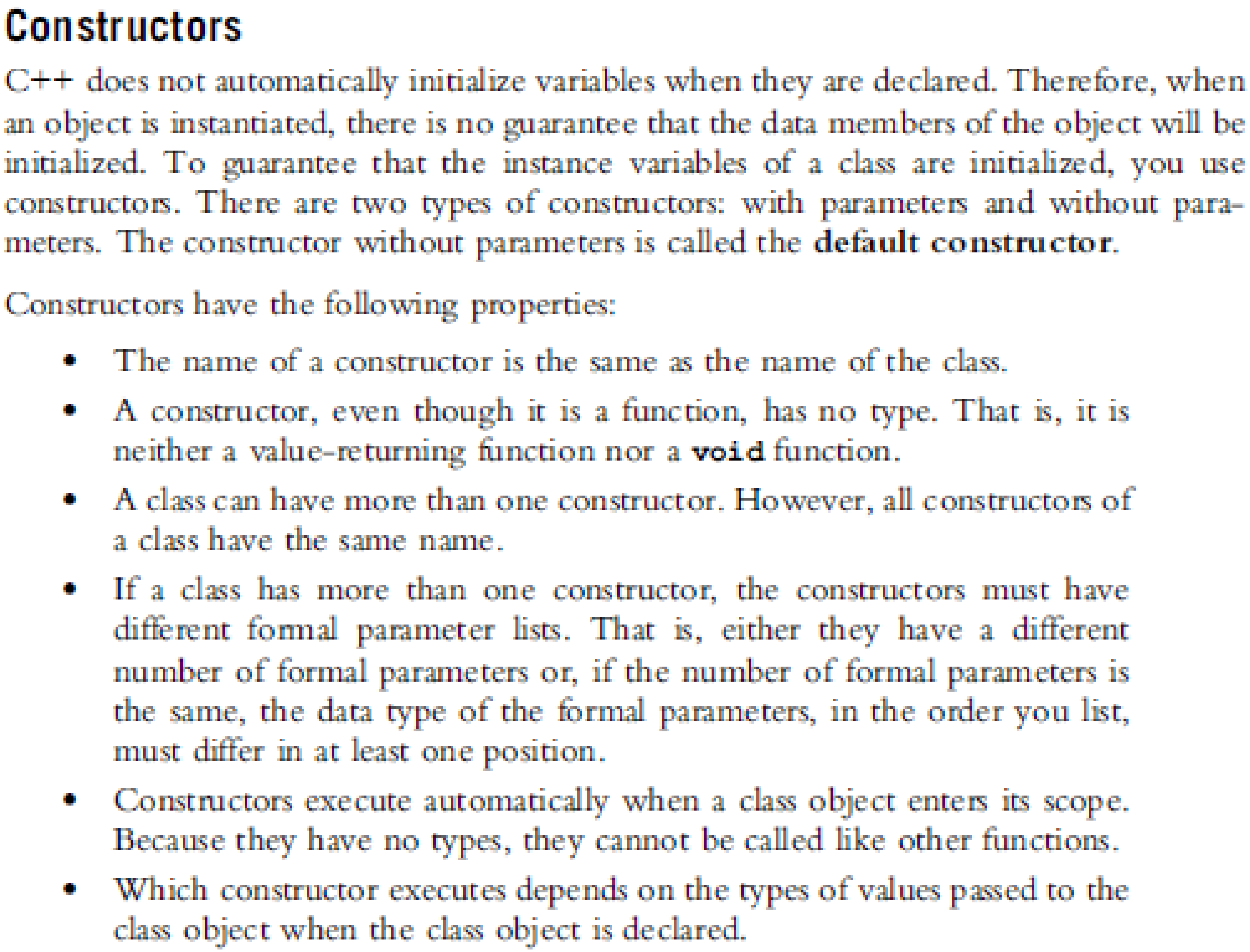
```
Please enter your magic word = moon
Good Night
--------------------------------
Process exited after 2.332 seconds with return value 0
Press any key to continue . . .
```

Thank you

BINUS UNIVERSITY INTERNATIONAL

FYI

Constructors



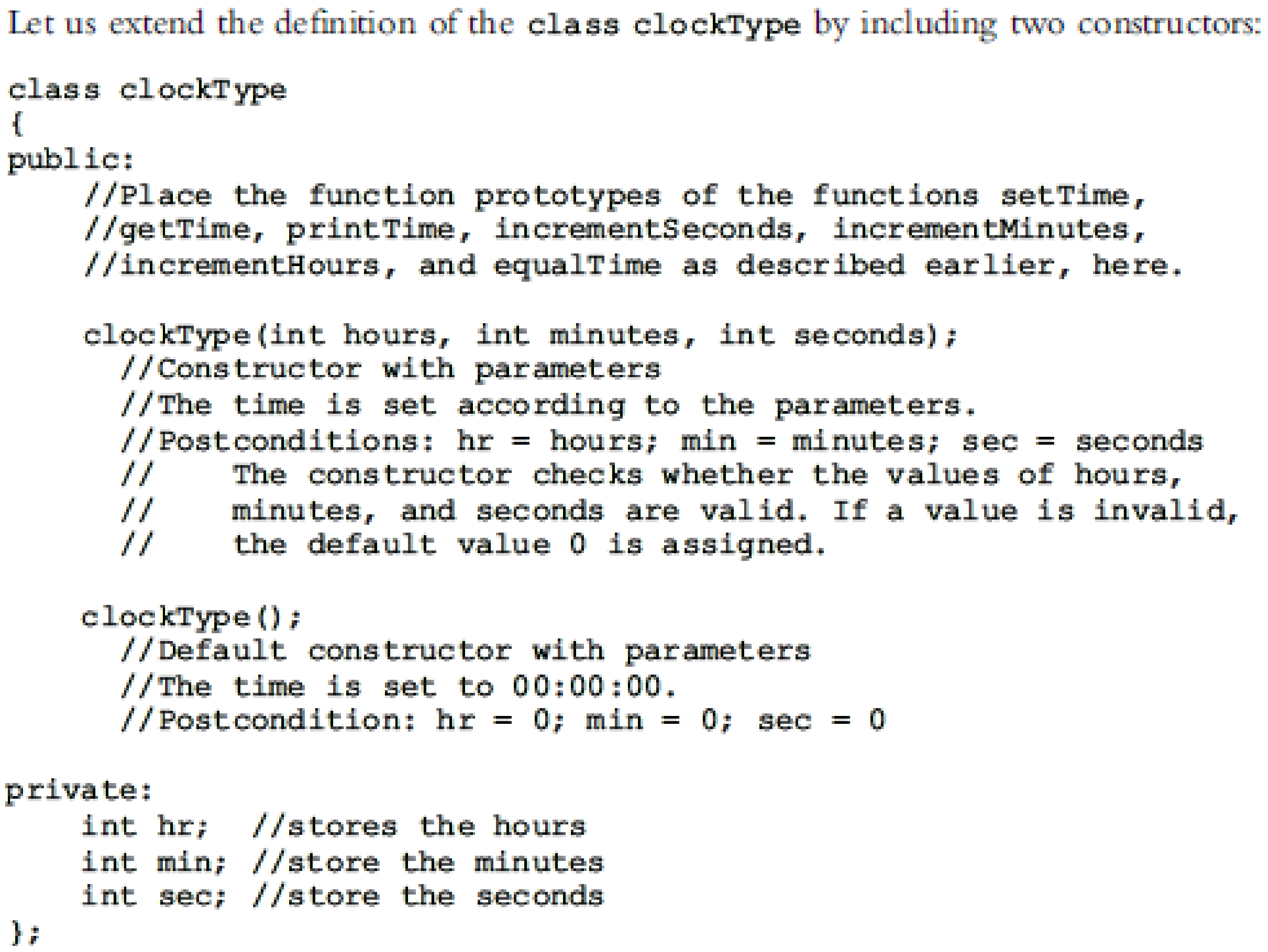
Constructors

C++ does not automatically initialize variables when they are declared. Therefore, when an object is instantiated, there is no guarantee that the data members of the object will be initialized. To guarantee that the instance variables of a class are initialized, you use constructors. There are two types of constructors: with parameters and without parameters. The constructor without parameters is called the **default constructor**.

Constructors have the following properties:

- The name of a constructor is the same as the name of the class.
- A constructor, even though it is a function, has no type. That is, it is neither a value-returning function nor a **`void`** function.
- A class can have more than one constructor. However, all constructors of a class have the same name.
- If a class has more than one constructor, the constructors must have different formal parameter lists. That is, either they have a different number of formal parameters or, if the number of formal parameters is the same, the data type of the formal parameters, in the order you list, must differ in at least one position.
- Constructors execute automatically when a class object enters its scope. Because they have no types, they cannot be called like other functions.
- Which constructor executes depends on the types of values passed to the class object when the class object is declared.

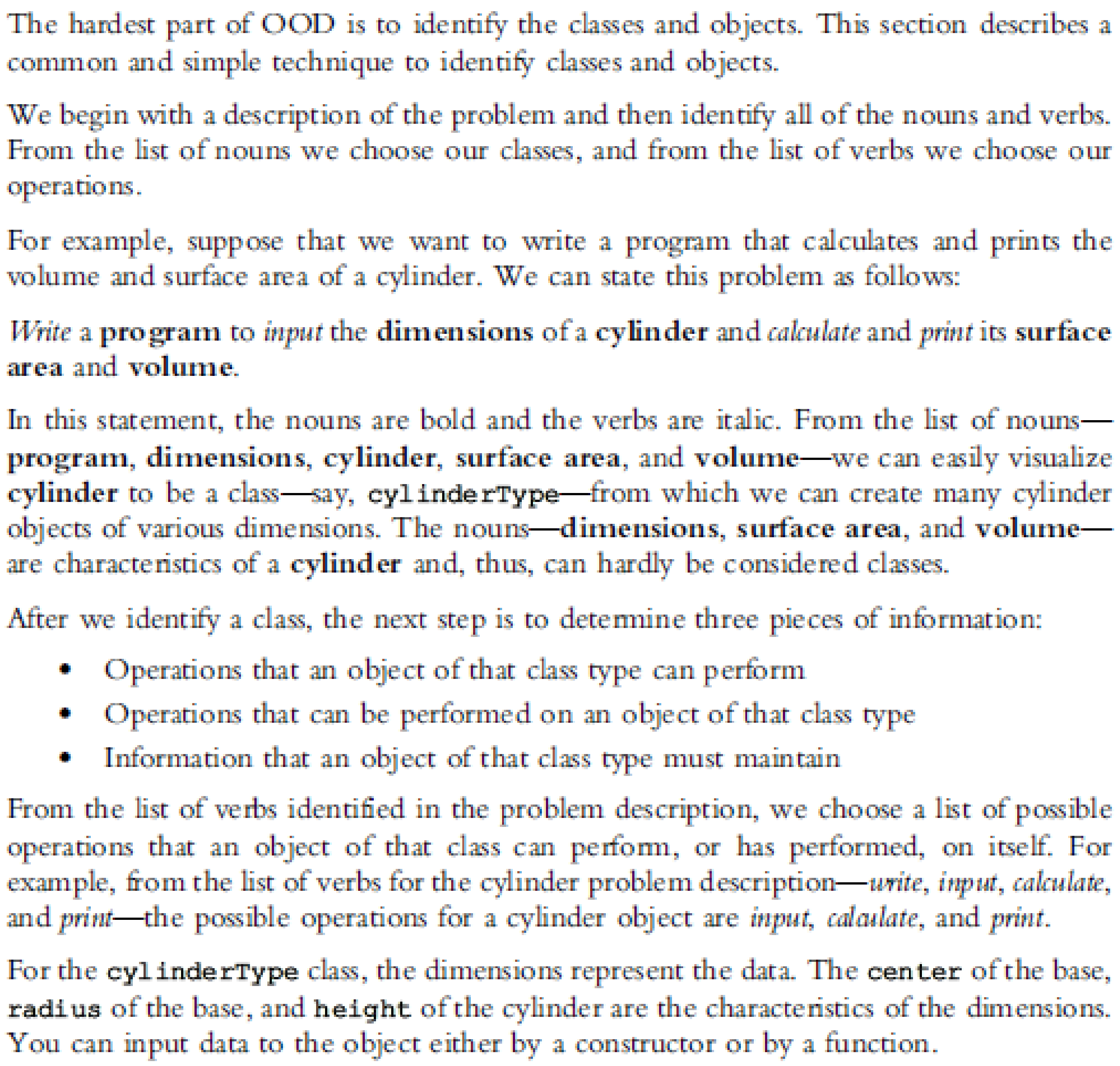
Constructors



Let us extend the definition of the `class clockType` by including two constructors:

```
class clockType
{
public:
    //Place the function prototypes of the functions setTime,
    //getTime, printTime, incrementSeconds, incrementMinutes,
    //incrementHours, and equalTime as described earlier, here.

    clockType(int hours, int minutes, int seconds);
      //Constructor with parameters
      //The time is set according to the parameters.
      //Postconditions: hr = hours; min = minutes; sec = seconds
      //    The constructor checks whether the values of hours,
      //    minutes, and seconds are valid. If a value is invalid,
      //    the default value 0 is assigned.

    clockType();
      //Default constructor with parameters
      //The time is set to 00:00:00.
      //Postcondition: hr = 0; min = 0; sec = 0

private:
    int hr;  //stores the hours
    int min; //store the minutes
    int sec; //store the seconds
};
```

Identifying Classes, Objects, and Operations



The hardest part of OOD is to identify the classes and objects. This section describes a common and simple technique to identify classes and objects.

We begin with a description of the problem and then identify all of the nouns and verbs. From the list of nouns we choose our classes, and from the list of verbs we choose our operations.

For example, suppose that we want to write a program that calculates and prints the volume and surface area of a cylinder. We can state this problem as follows:

Write a **program** to *input* the **dimensions** of a **cylinder** and *calculate* and *print* its **surface area** and **volume**.

In this statement, the nouns are bold and the verbs are italic. From the list of nouns—**program**, **dimensions**, **cylinder**, **surface area**, and **volume**—we can easily visualize **cylinder** to be a class—say, `cylinderType`—from which we can create many cylinder objects of various dimensions. The nouns—**dimensions**, **surface area**, and **volume**—are characteristics of a **cylinder** and, thus, can hardly be considered classes.

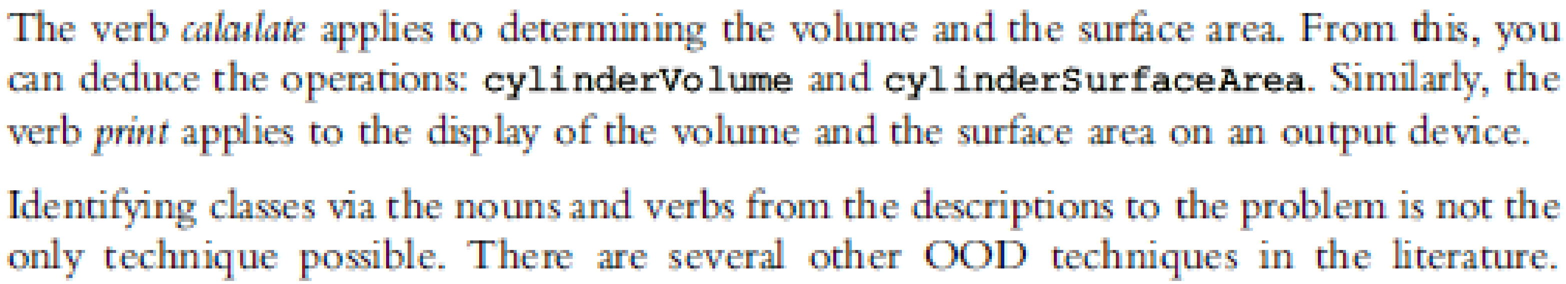
After we identify a class, the next step is to determine three pieces of information:

- Operations that an object of that class type can perform
- Operations that can be performed on an object of that class type
- Information that an object of that class type must maintain

From the list of verbs identified in the problem description, we choose a list of possible operations that an object of that class can perform, or has performed, on itself. For example, from the list of verbs for the cylinder problem description—*write*, *input*, *calculate*, and *print*—the possible operations for a cylinder object are *input*, *calaulate*, and *print*.

For the `cylinderType` class, the dimensions represent the data. The `center` of the base, `radius` of the base, and `height` of the cylinder are the characteristics of the dimensions. You can input data to the object either by a constructor or by a function.

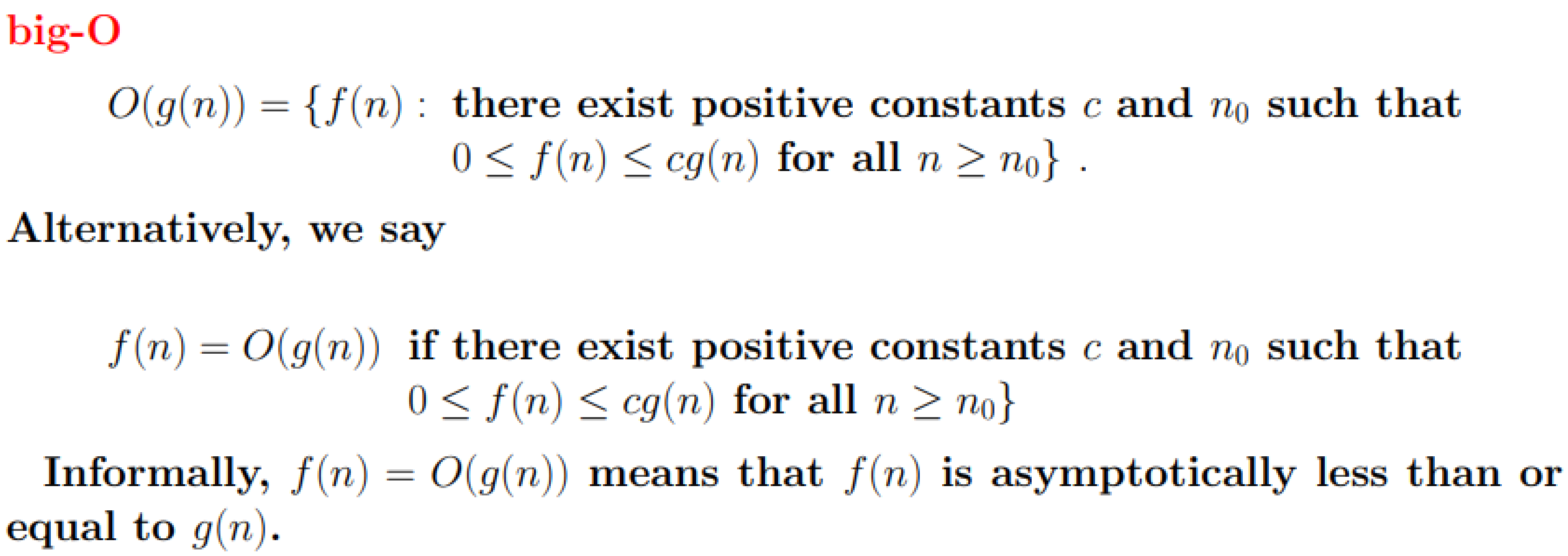
Identifying Classes, Objects, and Operations



The verb *calaulate* applies to determining the volume and the surface area. From this, you can deduce the operations: `cylinderVolume` and `cylinderSurfaceArea`. Similarly, the verb *print* applies to the display of the volume and the surface area on an output device.

Identifying classes via the nouns and verbs from the descriptions to the problem is not the only technique possible. There are several other OOD techniques in the literature.

Example



big-O

$$O(g(n)) = \{f(n) : \text{ there exist positive constants } c \text{ and } n_0 \text{ such that } 0 \le f(n) \le cg(n) \text{ for all } n \ge n_0\} \; .$$

Alternatively, we say

$$f(n) = O(g(n)) \text{ if there exist positive constants } c \text{ and } n_0 \text{ such that } 0 \le f(n) \le cg(n) \text{ for all } n \ge n_0\}$$

Informally, $f(n) = O(g(n))$ means that $f(n)$ is asymptotically less than or equal to $g(n)$.

- Each of the following expressions represents the number of operations for certain algorithms. What is the order of each of these Big-O expressions?
  - $n^2 + 3n + 5$

- Answer
  - Let $f(n) = n^2+3n+5$ and $g(n) = n^2$
  - Now, $n^2+3n+5 <= n^2 + n^2 <= 2 n^2$. Hence $f(n) = O(n^2)$

Exercises: Algorithm Analysis

- Each of the following expressions represents the number of operations for certain algorithms. What is the order of each of these Big-O expressions?

  a. 1

  a. $6n + 4$

  b. $5n^2 + 2n + 8$

  c. $(n^2 + 1)(3^n + 5)$

  d. $5(6^n + 4)$

  d. Log $(n + 4)$

  d. nLog(n)

- Write a function that uses a loop to find the sum of the squares of all integers between 1 and n. What is the order of your function?