

# General Announcements

## CS10001: Programming & Data Structures



**Pallab Dasgupta**

Professor, Dept. of Computer Sc. & Engg.,  
Indian Institute of Technology Kharagpur

# Items

---

- **Load and Credit Distribution**
- **Availability of Course Material**
- **Attendance matters**
- **References**

# About the Course

- **L-T-P rating of 3-1-0.**
- **There is a separate laboratory of 0-0-3.**
  - **Grading will be separate.**
- **Tutorial classes (one hour per week) will be conducted on a “per section” basis.**
- **Evaluation in the theory course:**
  - **Mid-semester** **30%**
  - **End-semester** **50%**
  - **Two class tests and attendance** **20%**

**5% of all components for regularity & performance**

# Course Materials

---

- The slides for the lectures will be made available on the web (in PDF form).  
**<http://144.16.192.60/~pds>**
- All important announcements will be put up on the web page.
- A copy of the slides will be kept at the Ramakrishna Xerox Centre. You may choose to bring the handouts to the class and take notes on them.

# Attendance REALLY matters

---

- **Students having poor attendance will loose credits**
- **Any student with less than 75% attendance may be debarred from appearing in the examinations**
- **Leave due to medical reasons must be certified by the B.C. Roy Technology Hospital**

# Text / Reference Books

**Classroom lectures are not substitutes for books. You must use reference books to clarify your doubts and enhance your knowledge.**

- 1. Programming with C (Second Edition)**  
**B.S. Gottfried, Schaum's Outline Series, Tata McGraw-Hill, 2006.**
- 2. Programming in ANSI C (Second Edition)**  
**E. Balagurusamy, Tata McGraw-Hill, New Delhi, 1992.**
- 3. Data structures**  
**S. Lipschutz, Schaum's Outline Series, Tata McGraw-Hill, 2006.**
- 4. Data structures using C and C++ (Second Edition)**  
**Y. Langsam, M.J. Augenstein, A.M. Tanenbaum, Prentice-Hall of India.**

# Acknowledgements

---

Several slides used in my presentations are adapted from the slides used by Prof. Indranil Sengupta when he took this course a few semesters ago.

# Introduction

## CS10001: Programming & Data Structures

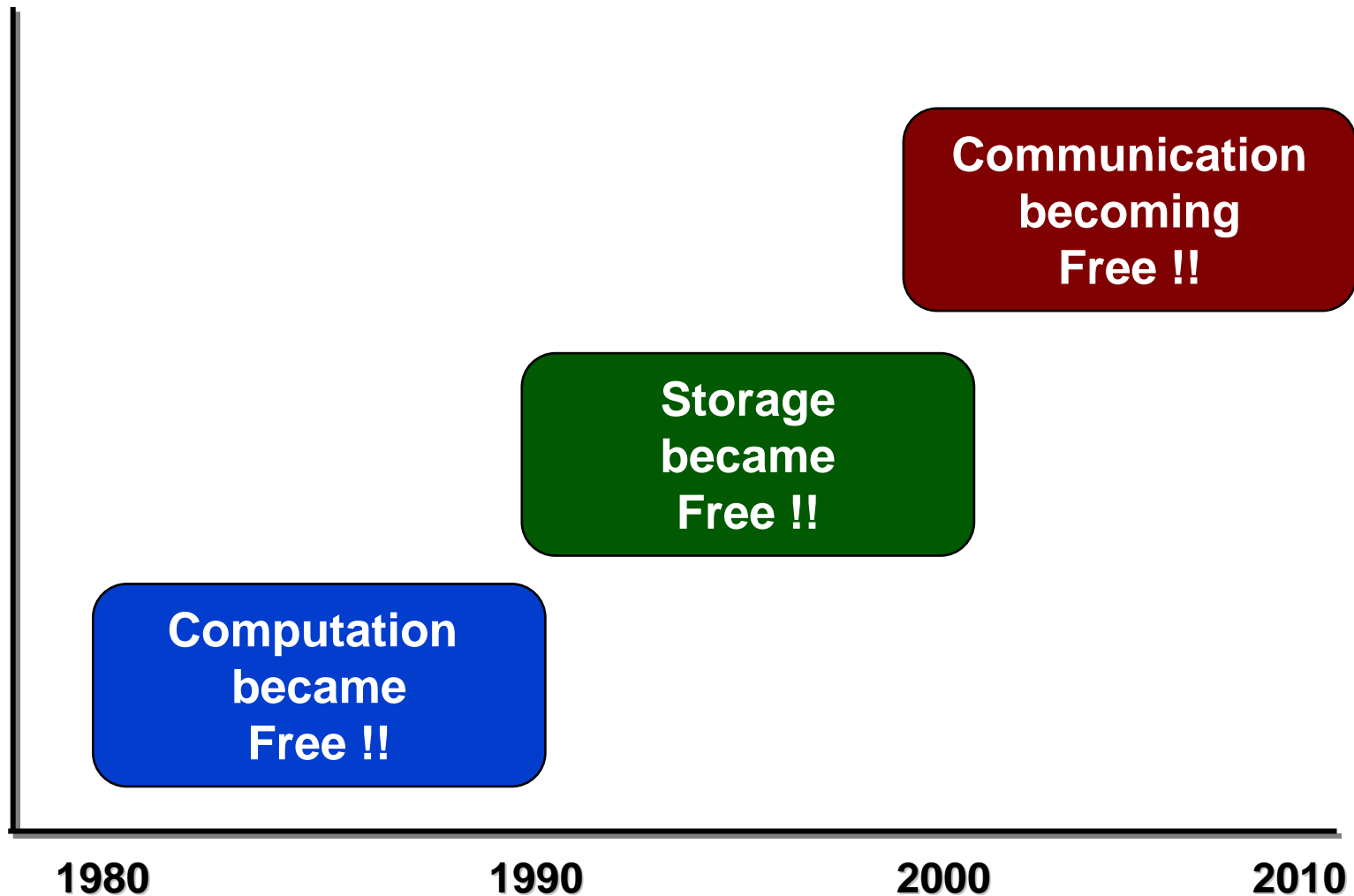


**Pallab Dasgupta**

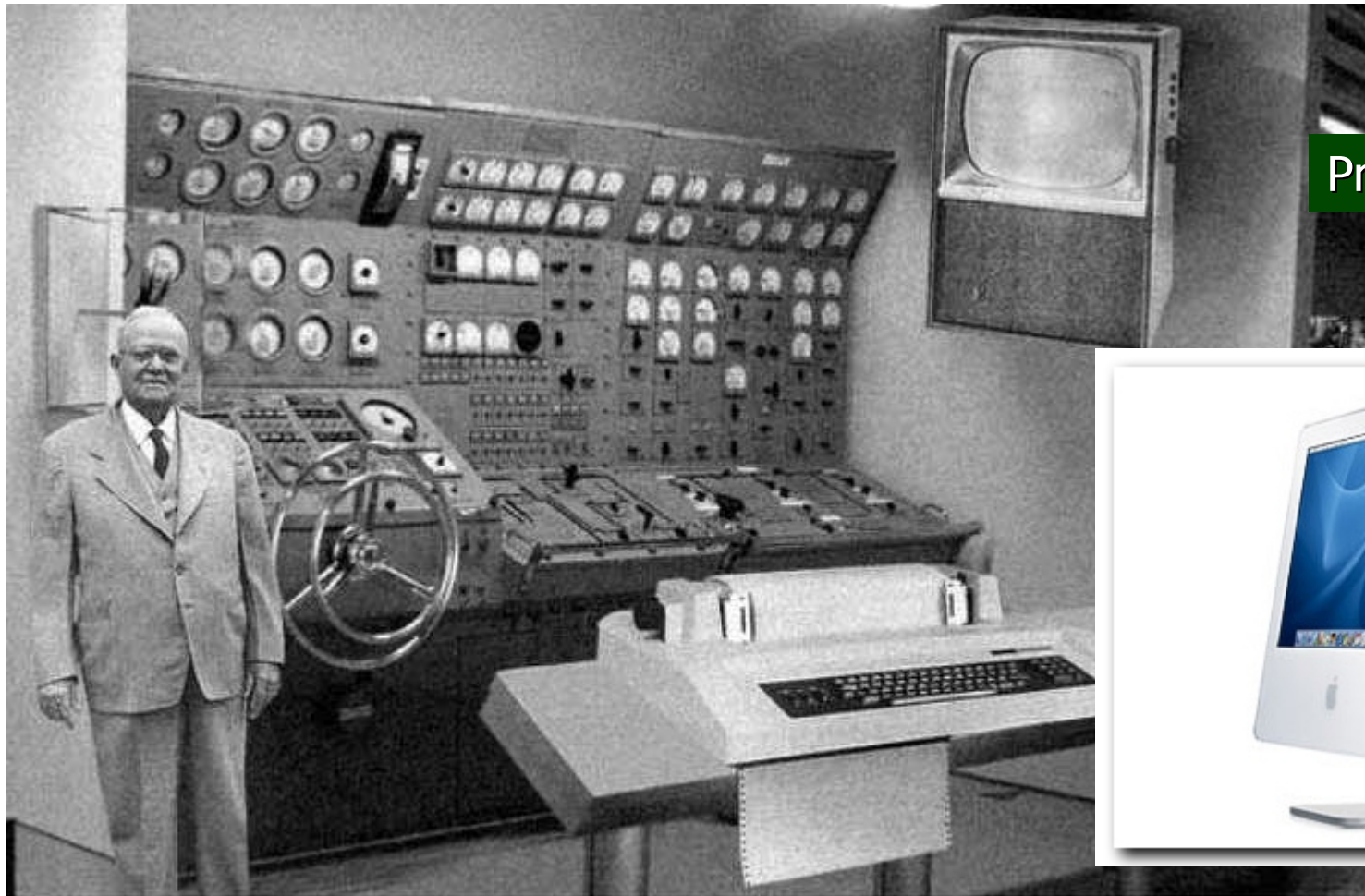
Professor, Dept. of Computer Sc. & Engg.,  
Indian Institute of Technology Kharagpur



# The Evolution of Electronic Computing



# Home Computer @ 2004: Predicted versus Real



Predicted in 1954



Reality

*Scientists from the RAND Corporation have created this model to illustrate how a "home computer" could look like in the year 2004. However the needed technology will not be economically feasible for the average home. Also the scientists readily admit that the computer will require not yet invented technology to actually work, but 50 years from now scientific progress is expected to solve these problems. With teletype interface and the Fortran language, the computer will be easy to use.*

# Storage has become free!!

**Magnetic Tape Drive**



**5 MB Hard Disk [1956]**

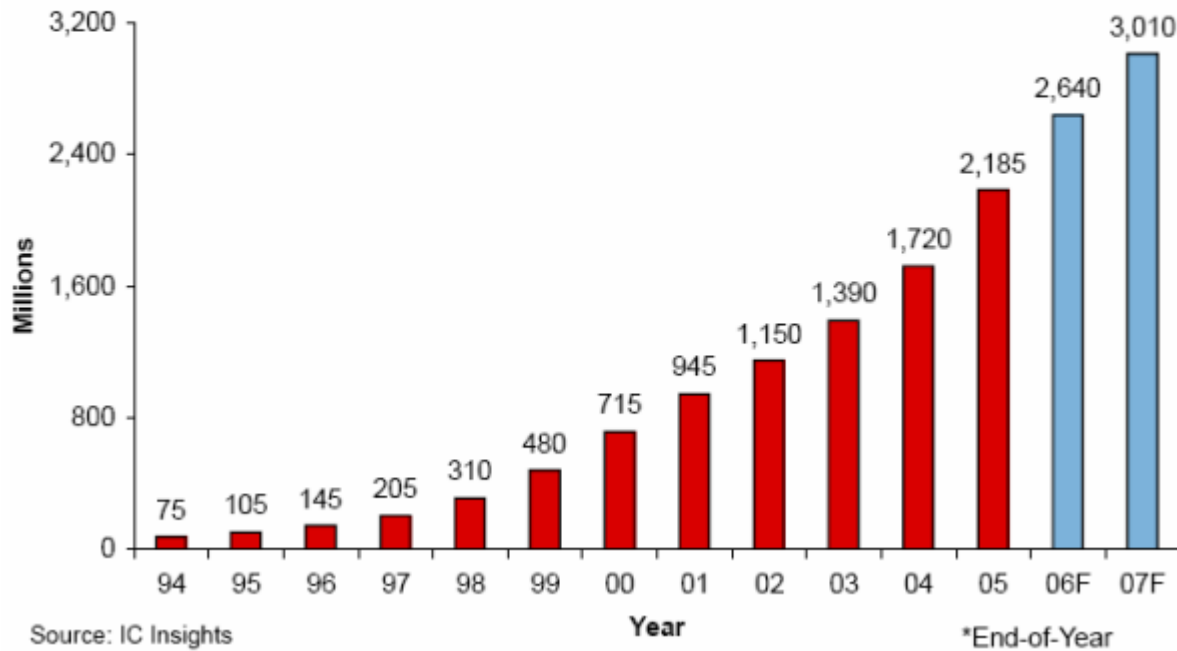


**4 GB Pen Drive**

# Communication??



Total Worldwide Cellular/PCS Subscriber Base\*  
(1994-2007)



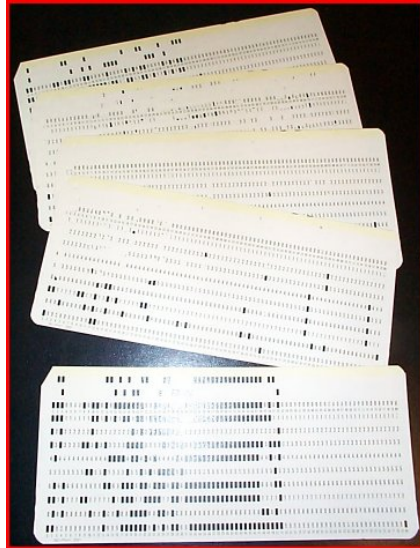
Source: IC Insights

\*End-of-Year

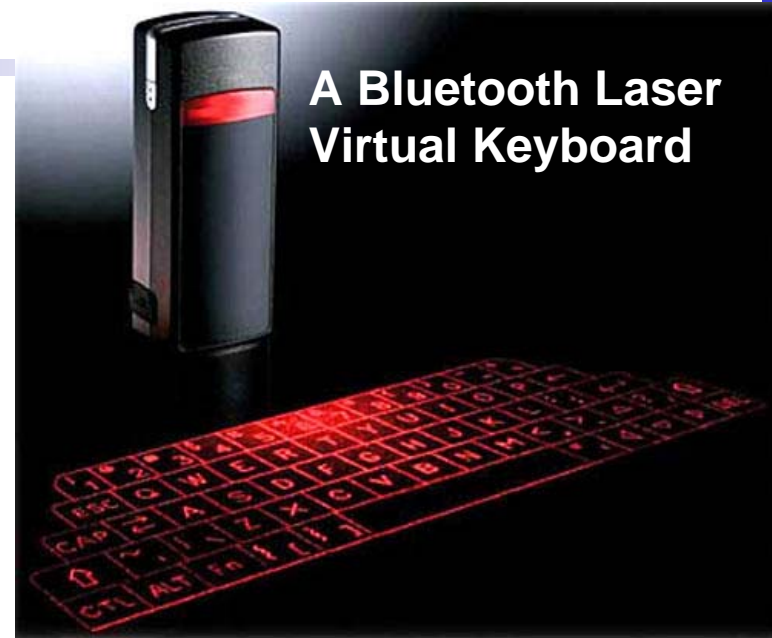
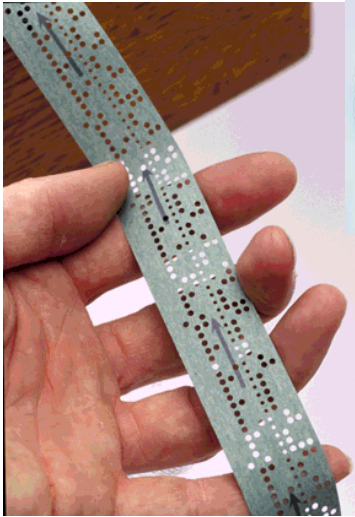




# Interfaces



From punch cards  
to paper tapes  
to keyboards

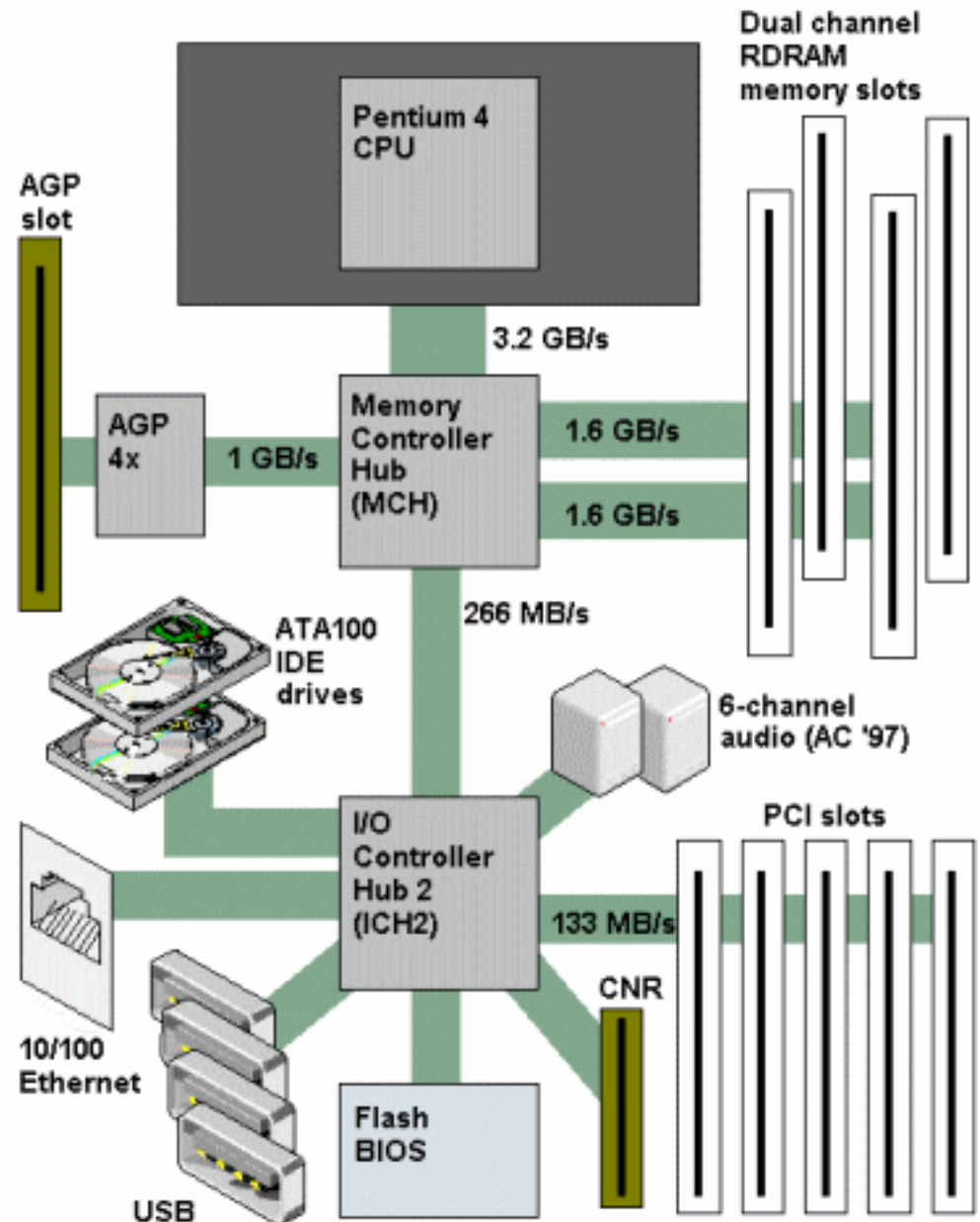


A Bluetooth Laser  
Virtual Keyboard



# Architecture

Typical system architecture  
for a desktop PC



# CPU (Central Processing Unit)

---

- All computations take place here in order for the computer to perform a designated task.
- It has a large number of registers which temporarily store data and programs (instructions).
- It has functional units (circuitry) to carry out arithmetic and logic operations
- It retrieves instructions from the memory, interprets (decodes) them, and performs the requested operation
- Fetch → Decode → Execute cycle
- CPU is also referred to as the processor
- Computers may have multiple processors
- Modern processors are multi-core (multiple processors in one chip)

# Main Memory

---

- **Uses semiconductor technology**
  - **Allows direct access**
- **Memory sizes in the range of 256 MegaBytes to 8 GigaBytes are typical today.**
- **Some measures to be remembered**
  - **1 K =  $2^{10}$  (= 1024)**
  - **1 M =  $2^{20}$  (= one million approx.)**
  - **1 G =  $2^{30}$  (= one billion approx.)**



# I/O and Peripherals

---

- **Input Device**
  - **Keyboard, Mouse, Scanner, Digital Camera**
- **Output Device**
  - **Monitor, Printer**
- **Storage Peripherals**
  - **Magnetic Disks: hard disk, floppy disk**
    - **Allows direct (semi-random) access**
  - **Optical Disks: CDROM, CD-RW, DVD**
    - **Allows direct (semi-random) access**
  - **Flash Memory: pen drives**
    - **Allows direct access**
  - **Magnetic Tape: DAT**
    - **Only sequential access**

# Typical Configuration of a PC

---

- **CPU:** Pentium IV, 3 GHz
- **Main Memory:** 1 GB
- **Hard Disk:** 160 GB
- **Floppy Disk:** Not present
- **CDROM:** DVD combo-drive
- **Input Device:** Keyboard, Mouse
- **Output Device:** 17" color monitor
- **Ports:** USB, Firewire, Infrared

# How does a computer work?

---

- **Stored program concept.**
  - **Main difference from a calculator.**
- **What is a program?**
  - **Set of instructions for carrying out a specific task.**
- **Where are programs stored?**
  - **In secondary memory, when first created.**
  - **Brought into main memory, during execution.**

# Number System – *The Basics*

- We are accustomed to using the so-called *decimal number system*.
  - Ten digits :: 0,1,2,3,4,5,6,7,8,9
  - Every digit position has a weight which is a power of 10.

- Examples:

$$234 = 2 \times 10^2 + 3 \times 10^1 + 4 \times 10^0$$

$$250.67 = 2 \times 10^2 + 5 \times 10^1 + 0 \times 10^0 + 6 \times 10^{-1} + 7 \times 10^{-2}$$

# A computer works on the binary number system

- **Binary number system:**

- Two digits :: 0,1
- Every digit position has a weight which is a power of 2.

- **Examples:**

$$101 \text{ (Binary)} = 1 \times 2^2 + 0 \times 2^1 + 1 \times 2^0 = 5 \text{ (Decimal)}$$

$$11001 \text{ (Binary)}$$

$$= 1 \times 2^4 + 1 \times 2^3 + 0 \times 2^2 + 0 \times 2^1 + 1 \times 2^0$$

$$= 25 \text{ (Decimal)}$$

# Bits and Bytes

---

- **Bit**
  - A single binary digit (0 or 1).
- **Nibble**
  - A collection of four bits (say, 0110).
- **Byte**
  - A collection of eight bits (say, 01000111).
- **Word**
  - Depends on the computer.
  - Typically 4 or 8 bytes (that is, 32 or 64 bits).

# Contd.

---

- **An k-bit decimal number**
  - Can express unsigned integers in the range  $0$  to  $10^k - 1$ 
    - For  $k=3$ , from  $0$  to  $999$ .
- **An k-bit binary number**
  - Can express unsigned integers in the range  $0$  to  $2^k - 1$ 
    - For  $k=8$ , from  $0$  to  $255$ .
    - For  $k=10$ , from  $0$  to  $1023$ .

# Classification of Software

---

- **Two categories:**

- 1. Application Software**

- Used to solve a particular problem.
- Editor, financial accounting, weather forecasting, etc.

- 2. System Software**

- Helps in running other programs.
- Compiler, operating system, etc.



# Computer Languages

---

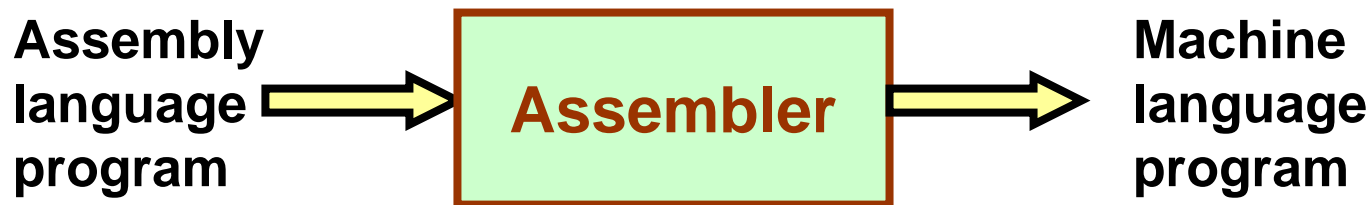
- **Machine Language**

- Expressed in binary.
- Directly understood by the computer.
- Not portable; varies from one machine type to another.
  - Program written for one type of machine will not run on another type of machine.
- Difficult to use in writing programs.

# Contd.

- **Assembly Language**

- Mnemonic form of machine language.
- Easier to use as compared to machine language.
  - For example, use “ADD” instead of “10110100”.
- Not portable (like machine language).
- Requires a translator program called *assembler*.



## Contd.

- Assembly language is also difficult to use in writing programs.
  - Requires many instructions to solve a problem.
- Example: Find the average of three numbers.

```
MOV    A,X    ; A = X
ADD    A,Y    ; A = A + Y
ADD    A,Z    ; A = A + Z
DIV    A,3    ; A = A / 3
MOV    RES,A  ; RES = A
```

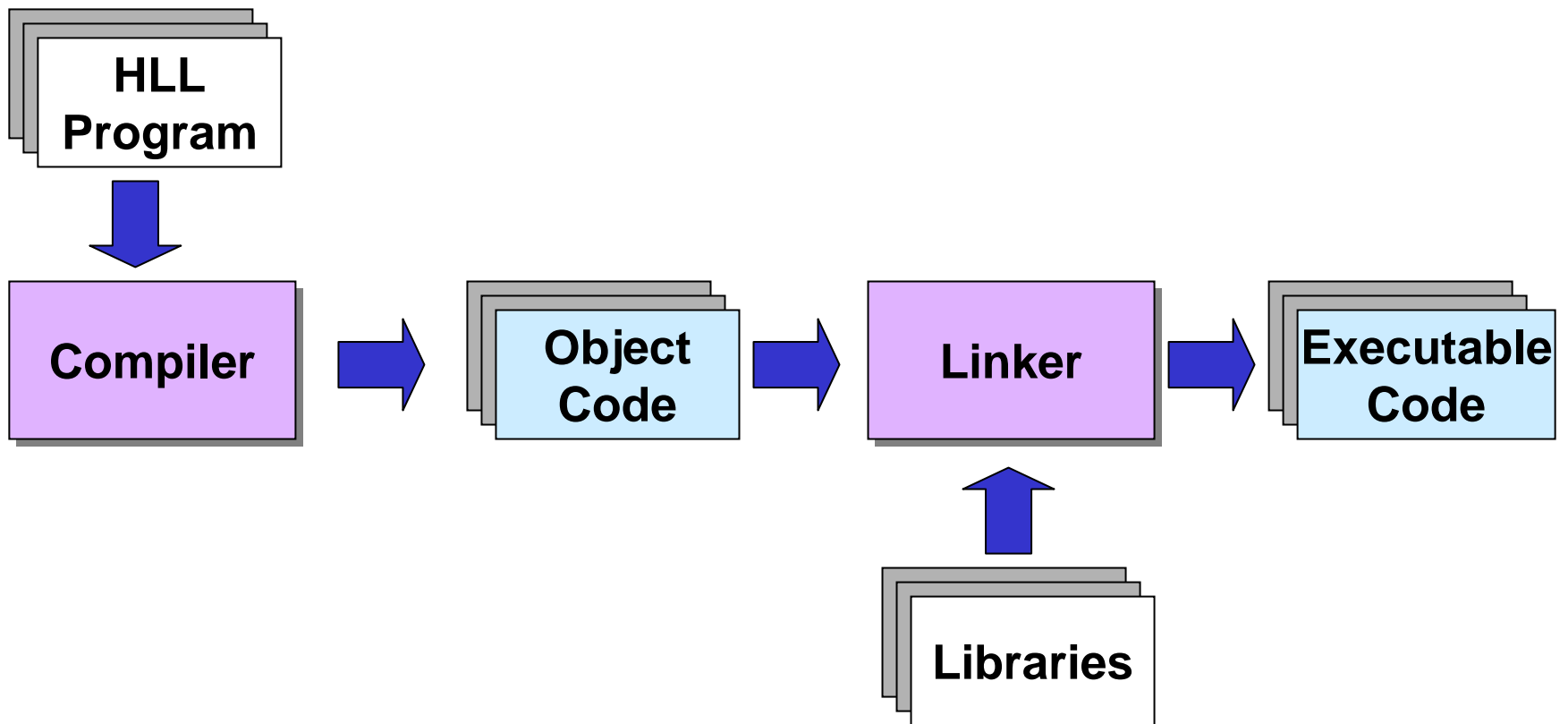
In C,

```
RES = (X + Y + Z) / 3
```

# High-Level Language

- Machine language and assembly language are called low-level languages.
  - They are closer to the machine.
  - Difficult to use.
- High-level languages are easier to use.
  - They are closer to the programmer.
  - Examples:
    - Fortran, Cobol, C, C++, Java.
  - Requires an elaborate process of translation.
    - Using a software called *compiler*.
  - They are portable across platforms.

# From HLL to executable



# Operating Systems

---

- **Makes the computer easy to use.**
  - **Basically the computer is very difficult to use.**
  - **Understands only machine language.**
- **Operating systems make computers easy to use.**
- **Categories of operating systems:**
  - **Single user**
  - **Multi user**
    - **Time sharing**
    - **Multitasking**
    - **Real time**

# Contd.

---

- **Popular operating systems:**
  - **Windows 2000/XP:**      **single-user multitasking**
  - **Unix:**                      **multi-user**
  - **Linux:**                      **a free version of Unix**
- **The laboratory class will be based on Linux.**
- **Question:**
  - **How multiple users can work on the same computer?**

# Contd.

---

- **Computers connected in a network.**
- **Many users may work on a computer.**
  - **Over the network.**
  - **At the same time.**
  - **CPU and other resources are shared among the different programs.**
    - **Called time sharing.**
    - **One program executes at a time.**



# Basic Programming Concepts

## CS10001: Programming & Data Structures



**Pallab Dasgupta**

Professor, Dept. of Computer Sc. & Engg.,  
Indian Institute of Technology Kharagpur

# Some Terminologies

---

- **Algorithm / Flowchart**

- A step-by-step procedure for solving a particular problem.
- Independent of the programming language.

- **Program**

- A translation of the algorithm/flowchart into a form that can be processed by a computer.
- Typically written in a high-level language like C, C++, Java, etc.

# Variables and Constants

---

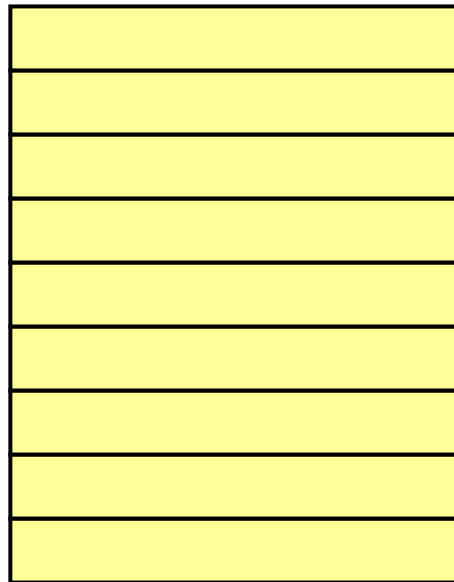
- **Most important concept for problem solving using computers**
- **All temporary results are stored in terms of variables**
  - **The value of a variable can be changed.**
  - **The value of a constant do not change.**
- **Where are they stored?**
  - **In main memory.**

# Contd.

---

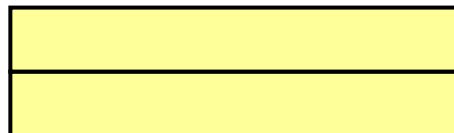
- How does memory look like (logically)?
  - As a list of storage locations, each having a unique address.
  - Variables and constants are stored in these storage locations.
  - A variable is like a *bin*
    - The contents of the *bin* is the *value* of the variable
    - The variable name is used to refer to the value of the variable
    - A variable is mapped to a *location* of the memory, called its *address*

# Memory map



Address 0  
Address 1  
Address 2  
Address 3  
Address 4  
Address 5  
Address 6

**Every variable is mapped to a particular memory address**



Address N-1

# Variables in Memory

Instruction executed

Variable X

Time  
↓

$X = 10$



10

$X = 20$



20

$X = X + 1$



21

$X = X * 5$



105

# Variables in Memory (contd.)

Instruction executed

Time  
↓

$$X = 20$$



$$Y = 15$$



$$X = Y + 3$$



$$Y = X / 6$$



Variable

X

Y

20

?

20

15

18

15

18

3

# Data Types

---

- **Three common data types used:**
  - **Integer :: can store only whole numbers**
    - **Examples: 25, -56, 1, 0**
  - **Floating-point :: can store numbers with fractional values.**
    - **Examples: 3.14159, 5.0, -12345.345**
  - **Character :: can store a character**
    - **Examples: 'A', 'a', '\*', '3', ' ', '+'**



# Data Types (contd.)

---

- How are they stored in memory?
  - **Integer ::**
    - 16 bits
    - 32 bits
  - **Float ::**
    - 32 bits
    - 64 bits
  - **Char ::**
    - 8 bits (ASCII code)
    - 16 bits (UNICODE, used in Java)

Actual number of bits vary from one computer to another

# Problem solving

---

- **Step 1:**
  - Clearly specify the problem to be solved.
- **Step 2:**
  - Draw flowchart or write algorithm.
- **Step 3:**
  - Convert flowchart (algorithm) into program code.
- **Step 4:**
  - Compile the program into object code.
- **Step 5:**
  - Execute the program.

# Flowchart: basic symbols

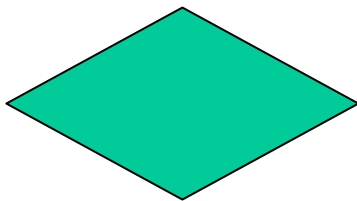
---



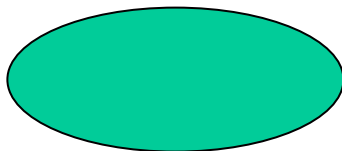
**Computation**



**Input / Output**



**Decision Box**



**Start / Stop**

# Contd.

---

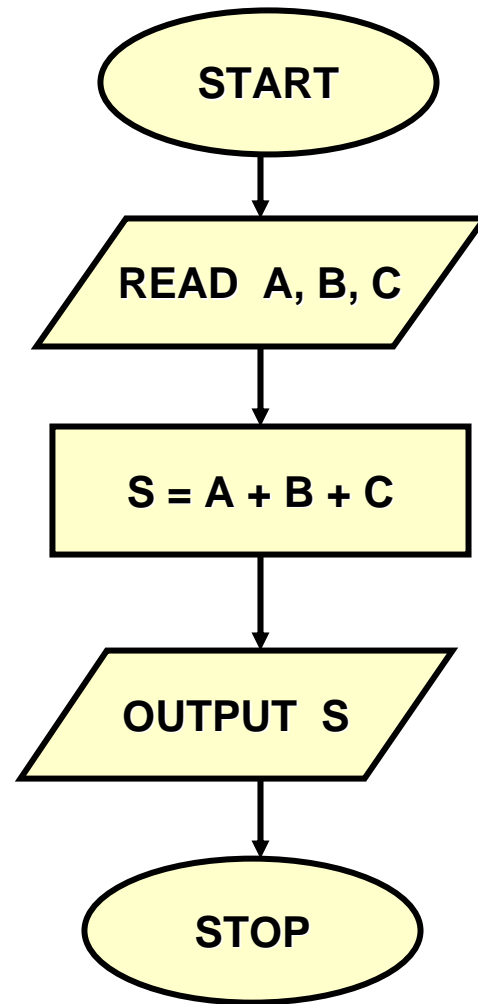


**Flow of  
control**

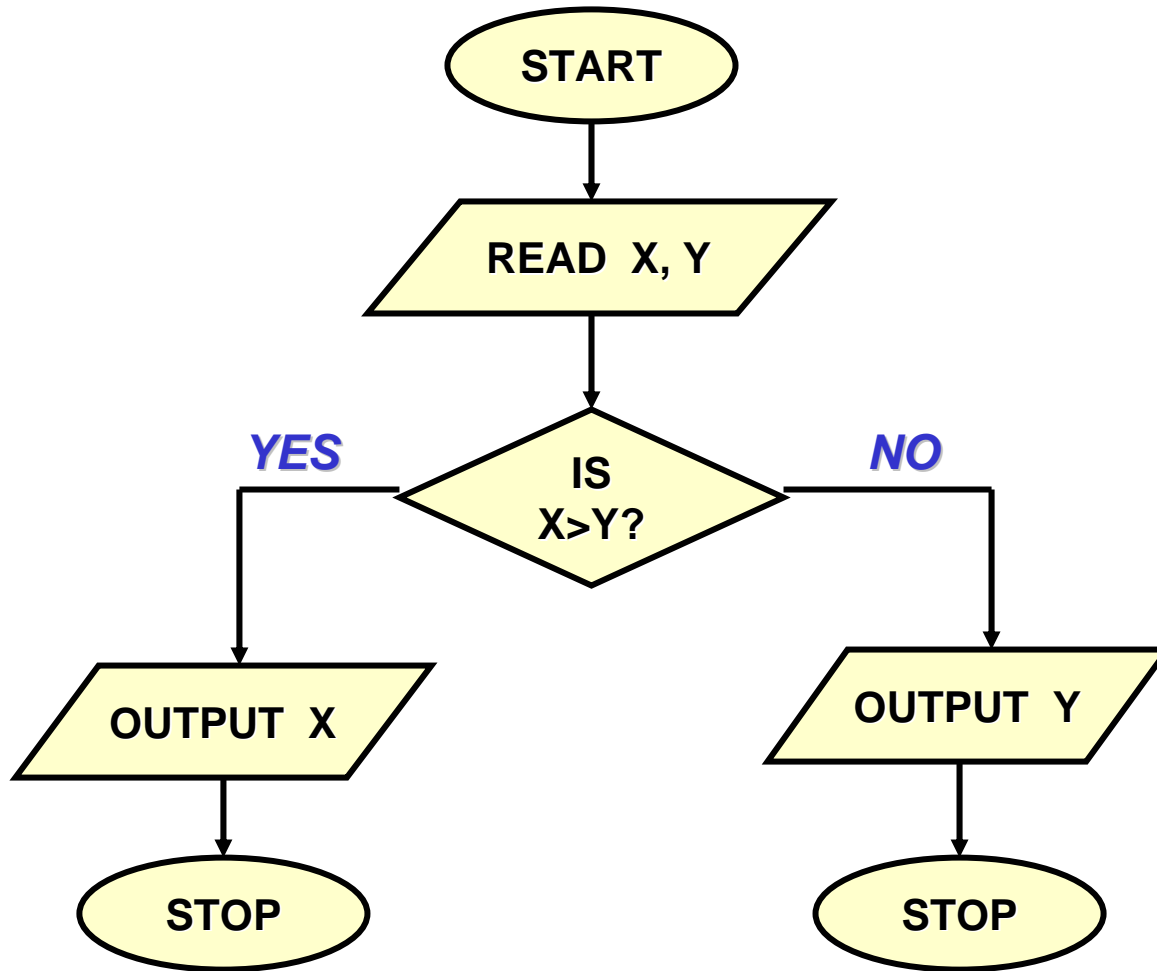


**Connector**

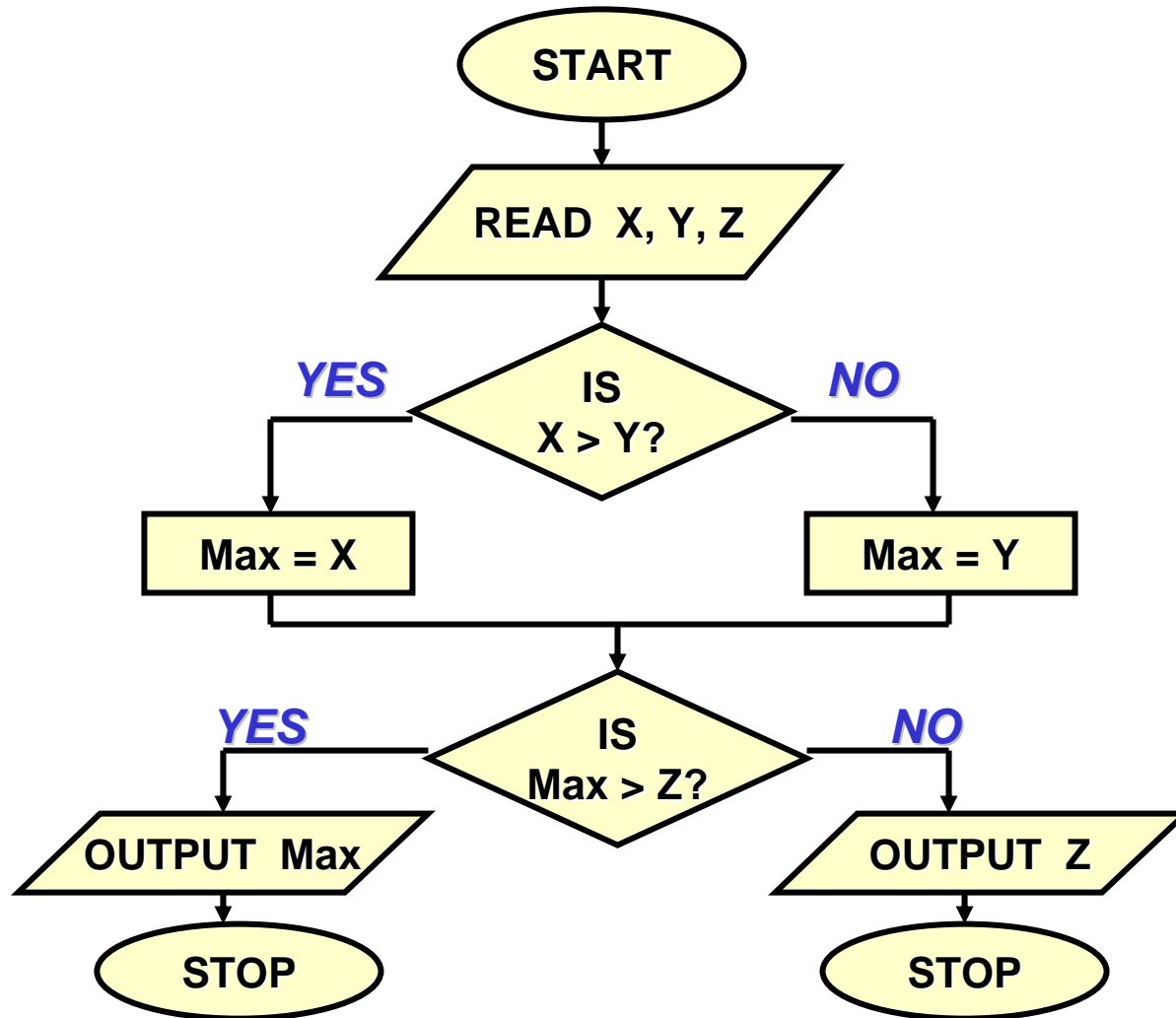
## Example 1: Adding three numbers



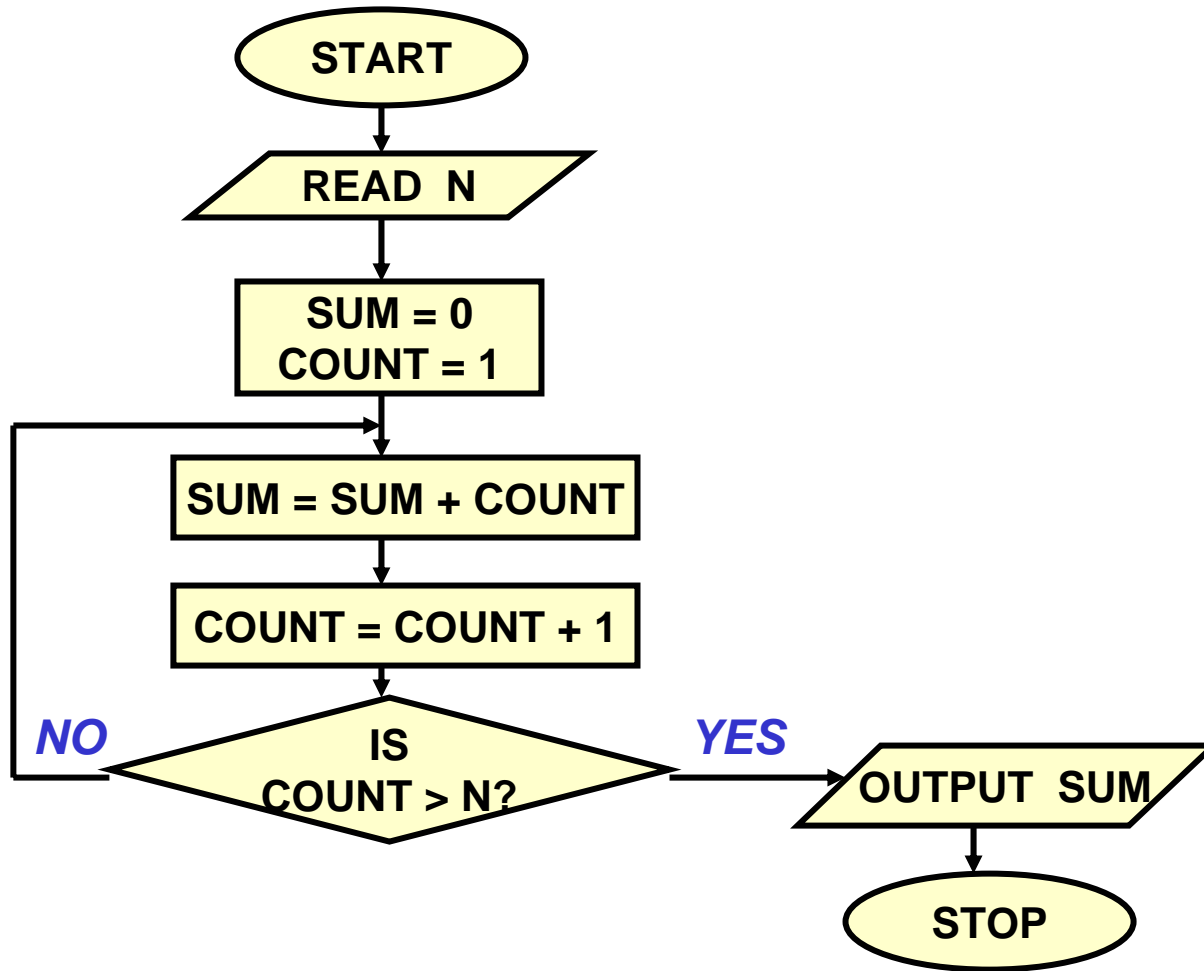
## Example 2: Larger of two numbers



## Example 3: Largest of three numbers

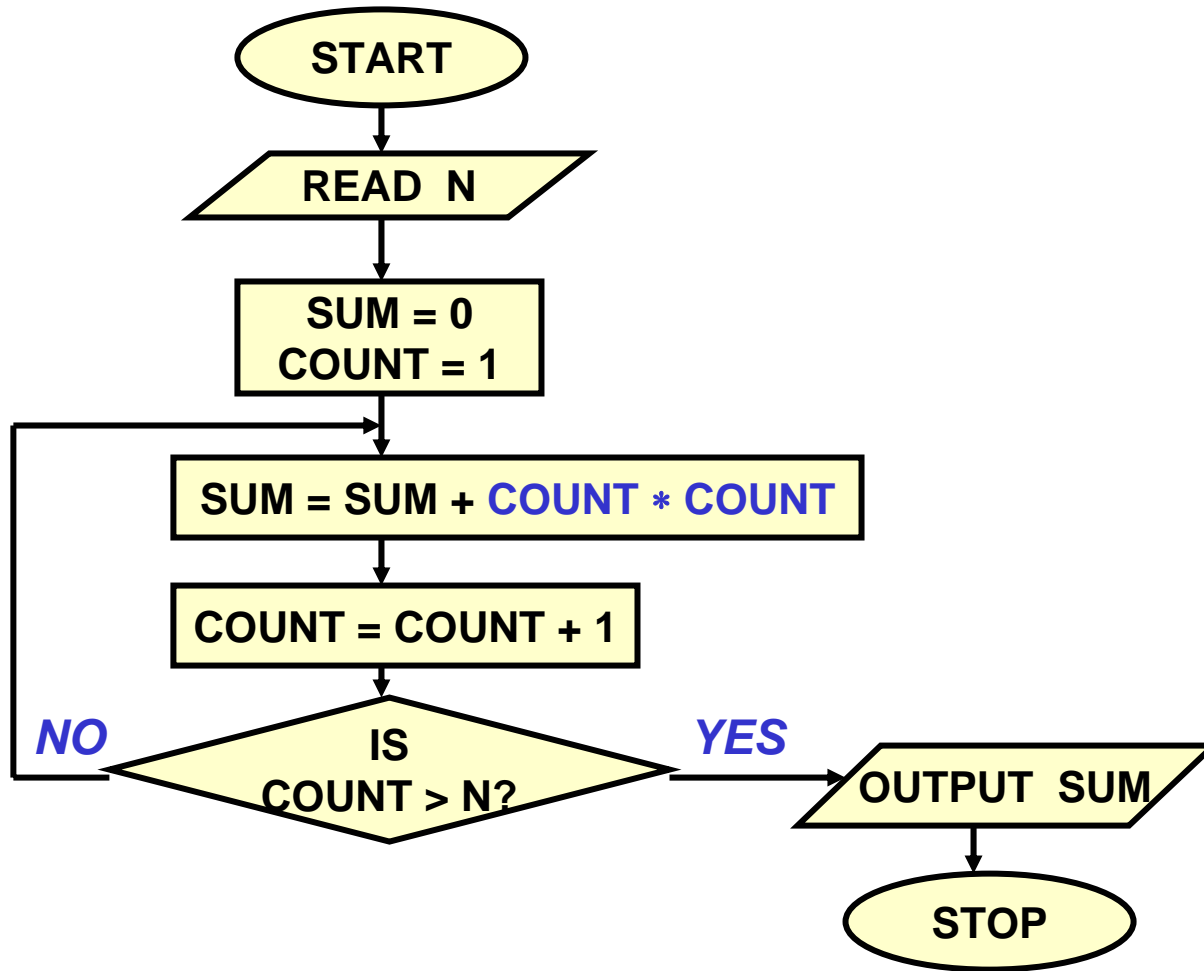


## Example 4: Sum of first $N$ natural numbers

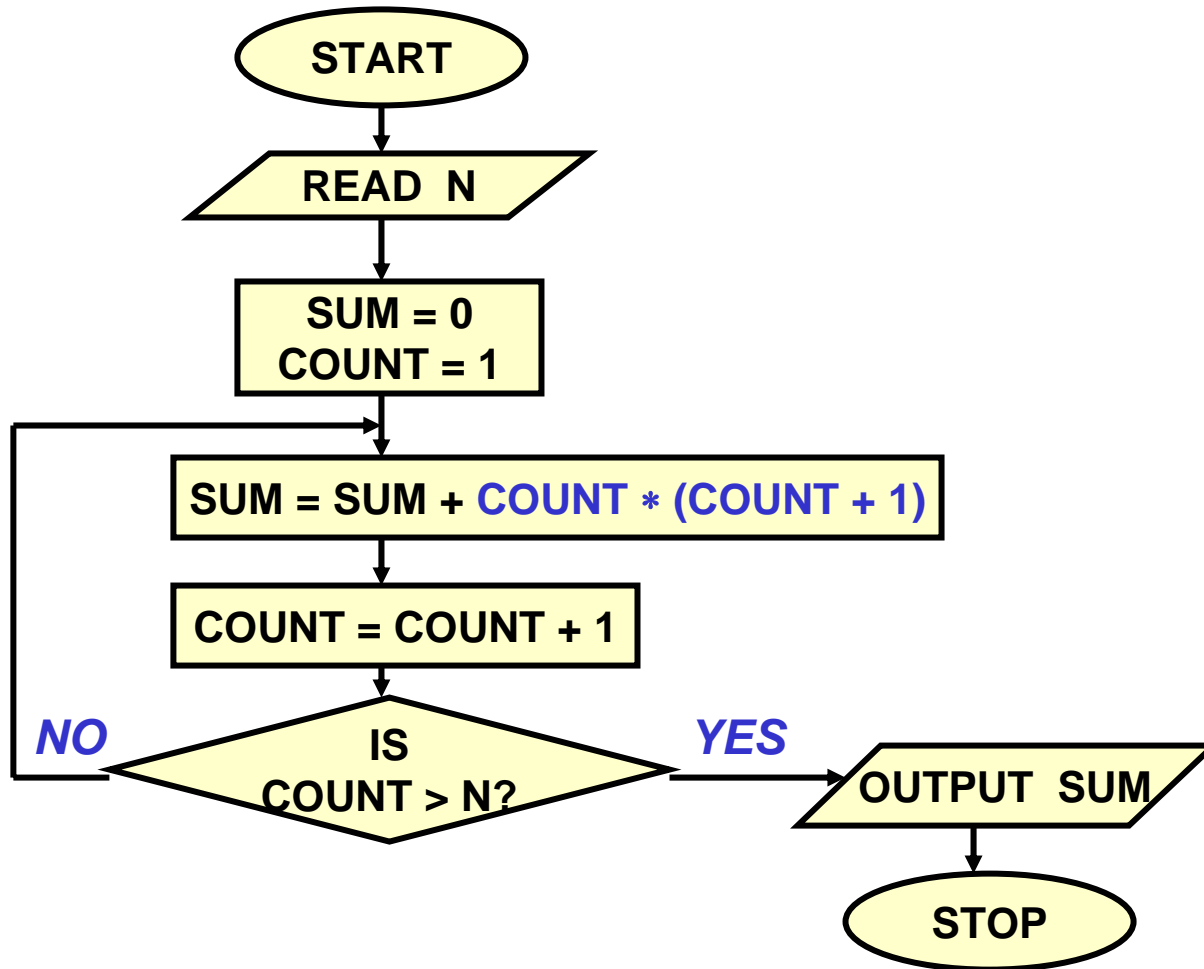




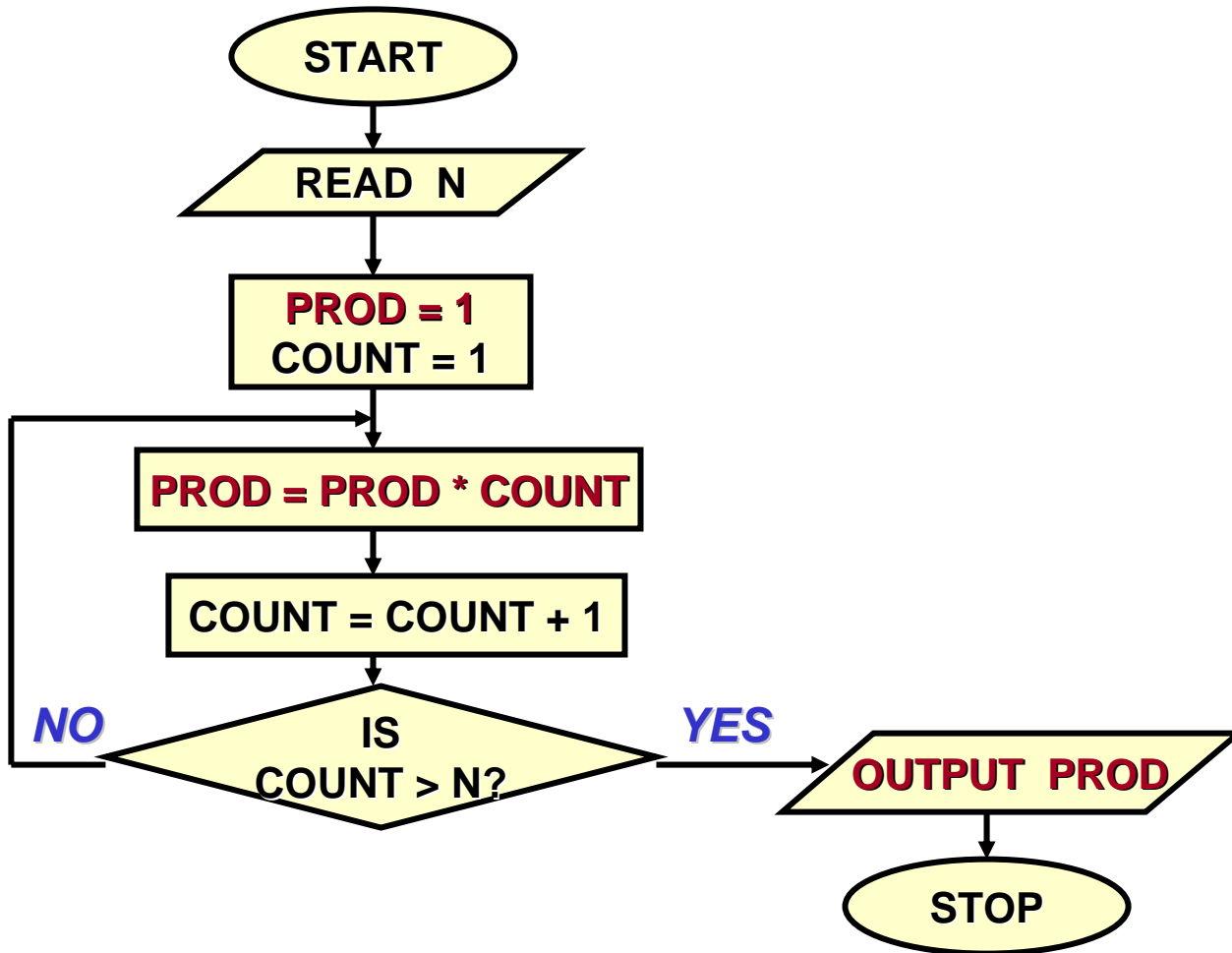
## Example 5: $SUM = 1^2 + 2^2 + 3^2 + N^2$



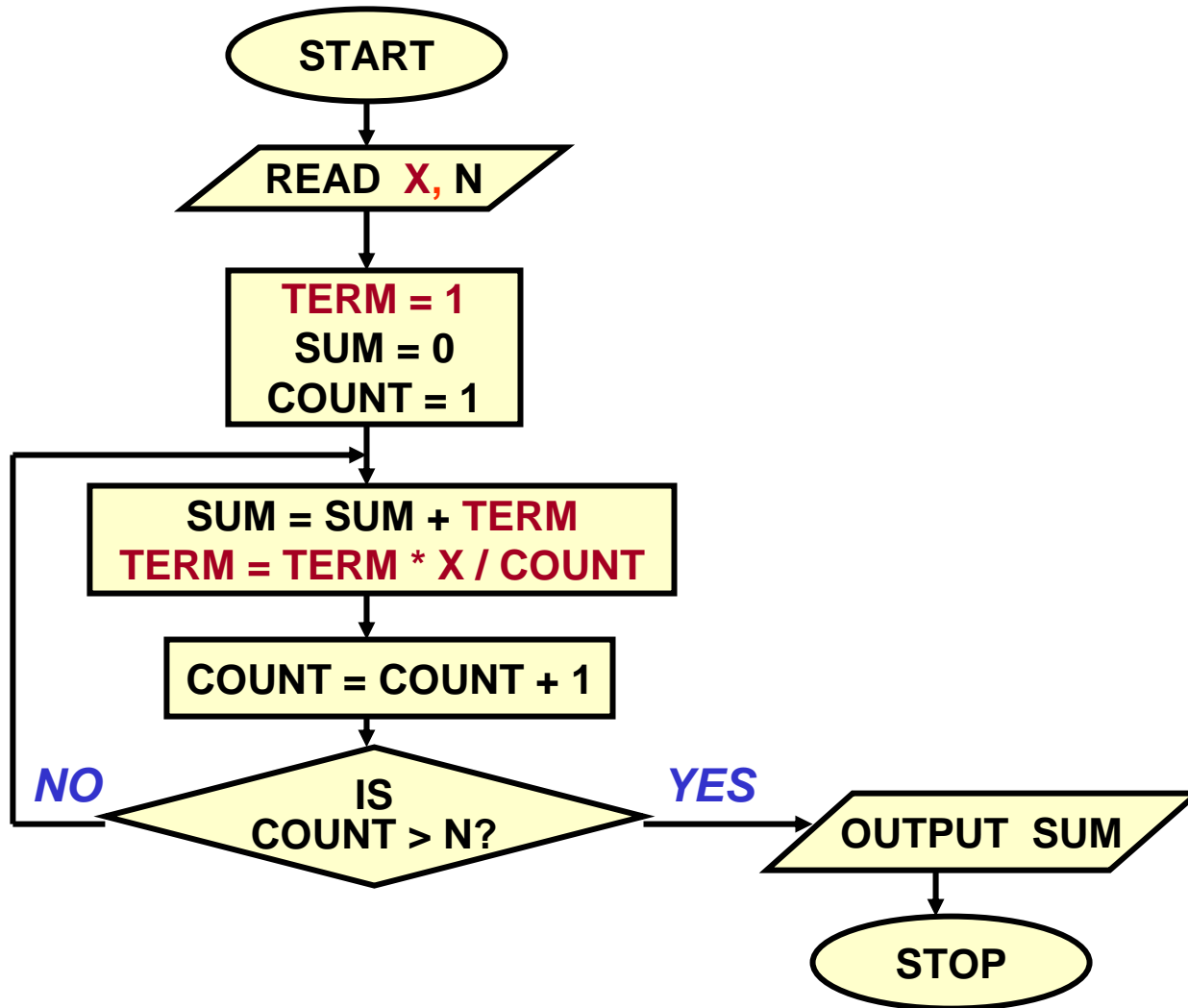
## Example 6: $SUM = 1.2 + 2.3 + 3.4 + \dots$ to $N$ terms



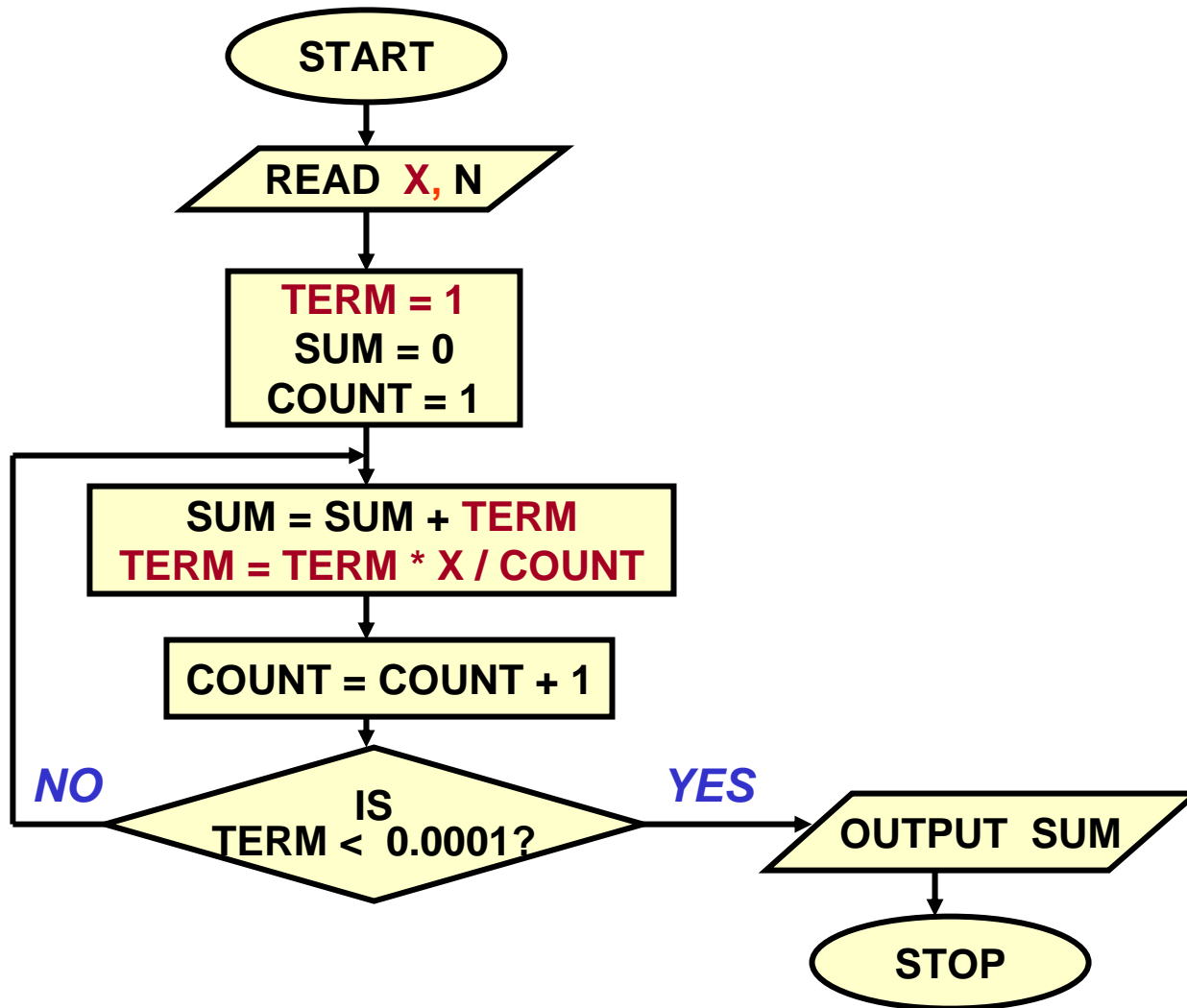
## Example 7: Computing Factorial



## Example 8: Computing $e^x$ series up to $N$ terms



## Example 8: Computing $e^x$ series up to 4 decimal places



## Example 10: *Roots of a quadratic equation*

---

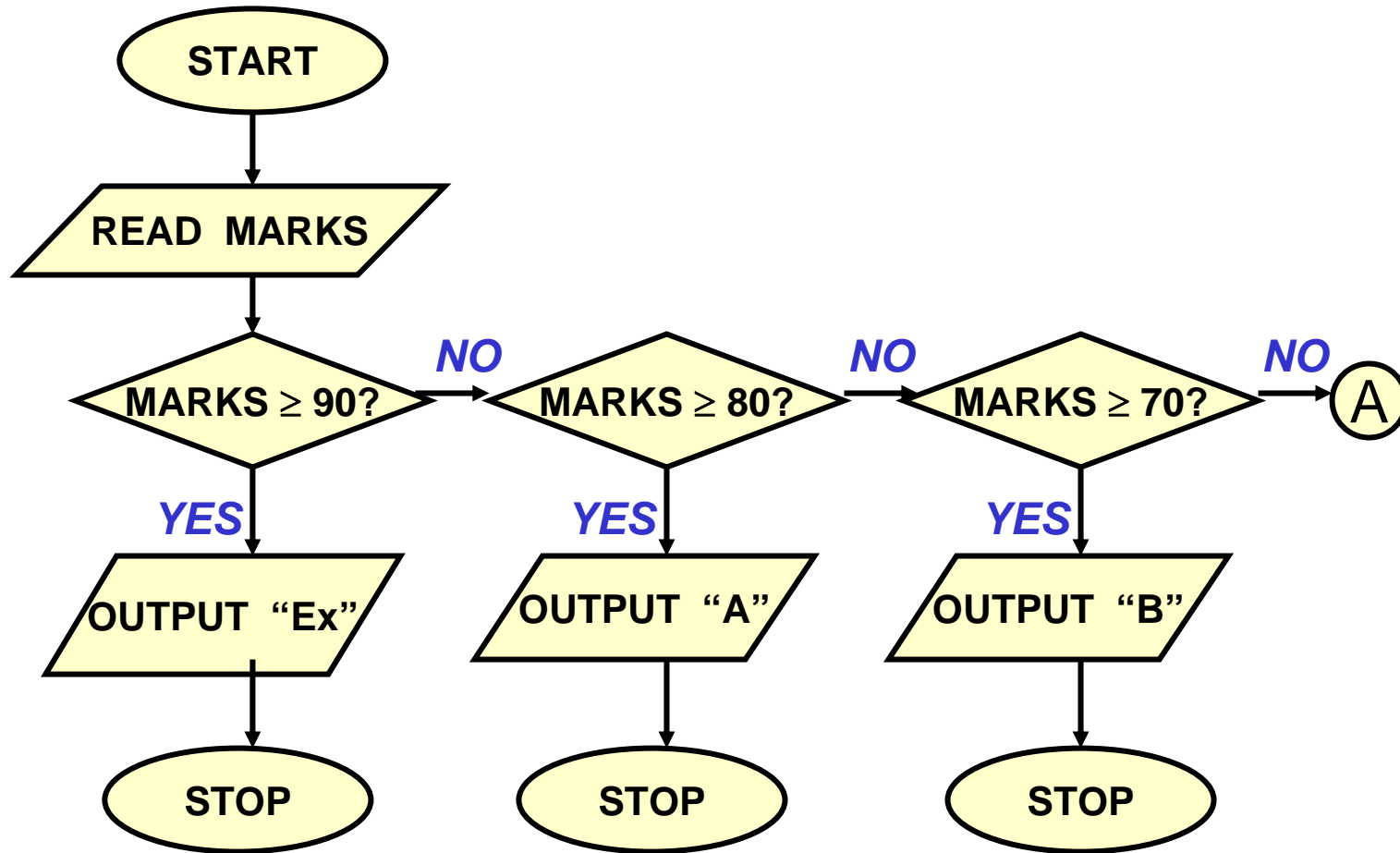
$$ax^2 + bx + c = 0$$

***TRY YOURSELF***

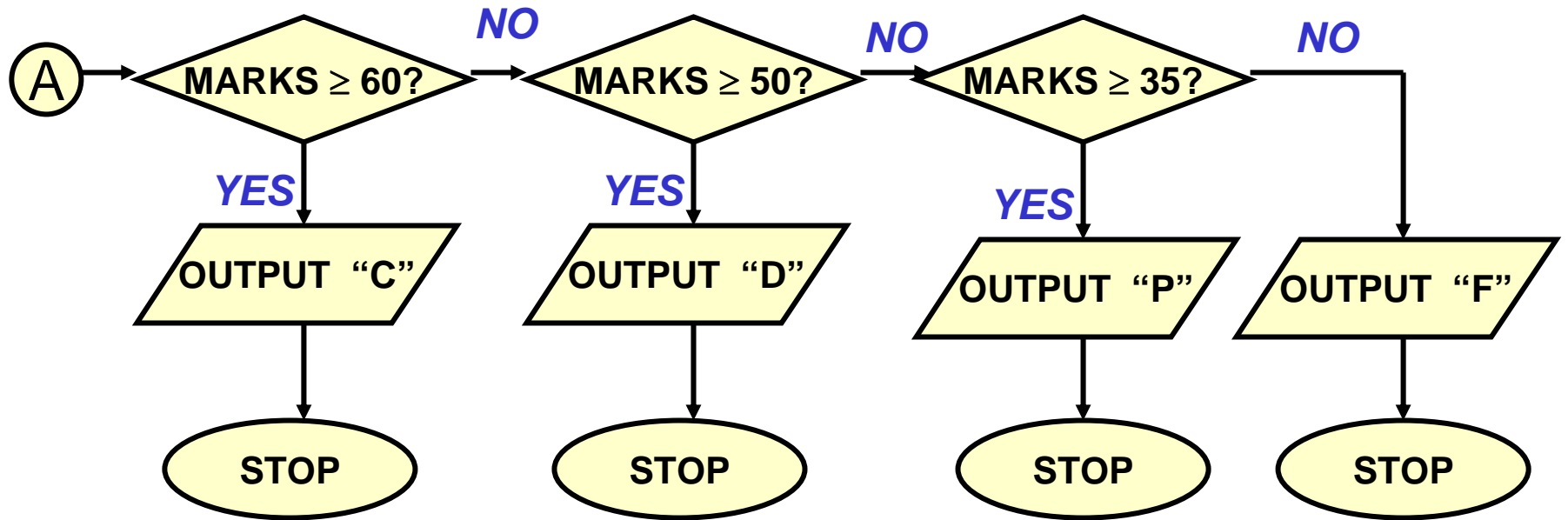
## Example 11: *Grade computation*

<b>MARKS <math>\geq</math> 90</b>	<b>→ Ex</b>
<b>89 <math>\geq</math> MARKS <math>\geq</math> 80</b>	<b>→ A</b>
<b>79 <math>\geq</math> MARKS <math>\geq</math> 70</b>	<b>→ B</b>
<b>69 <math>\geq</math> MARKS <math>\geq</math> 60</b>	<b>→ C</b>
<b>59 <math>\geq</math> MARKS <math>\geq</math> 50</b>	<b>→ D</b>
<b>49 <math>\geq</math> MARKS <math>\geq</math> 35</b>	<b>→ P</b>
<b>34 <math>\geq</math> MARKS</b>	<b>→ F</b>

# Grade Computation (contd.)







# Programming in C: *Basics*

**CS10001: Programming & Data Structures**



**Pallab Dasgupta**

Professor, Dept. of Computer Sc. & Engg.,  
Indian Institute of Technology Kharagpur

# History of C

---

- Originally developed in the 1970's by Dennis Ritchie at AT&T Bell Laboratories.
  - Outgrowth of two earlier languages BCPL and B.
- Popularity became widespread by the mid 1980's, with the availability of compilers for various platforms.
- Standardization has been carried out to make the various C implementations compatible.
  - American National Standards Institute (ANSI)
  - GNU

# Why teach C?

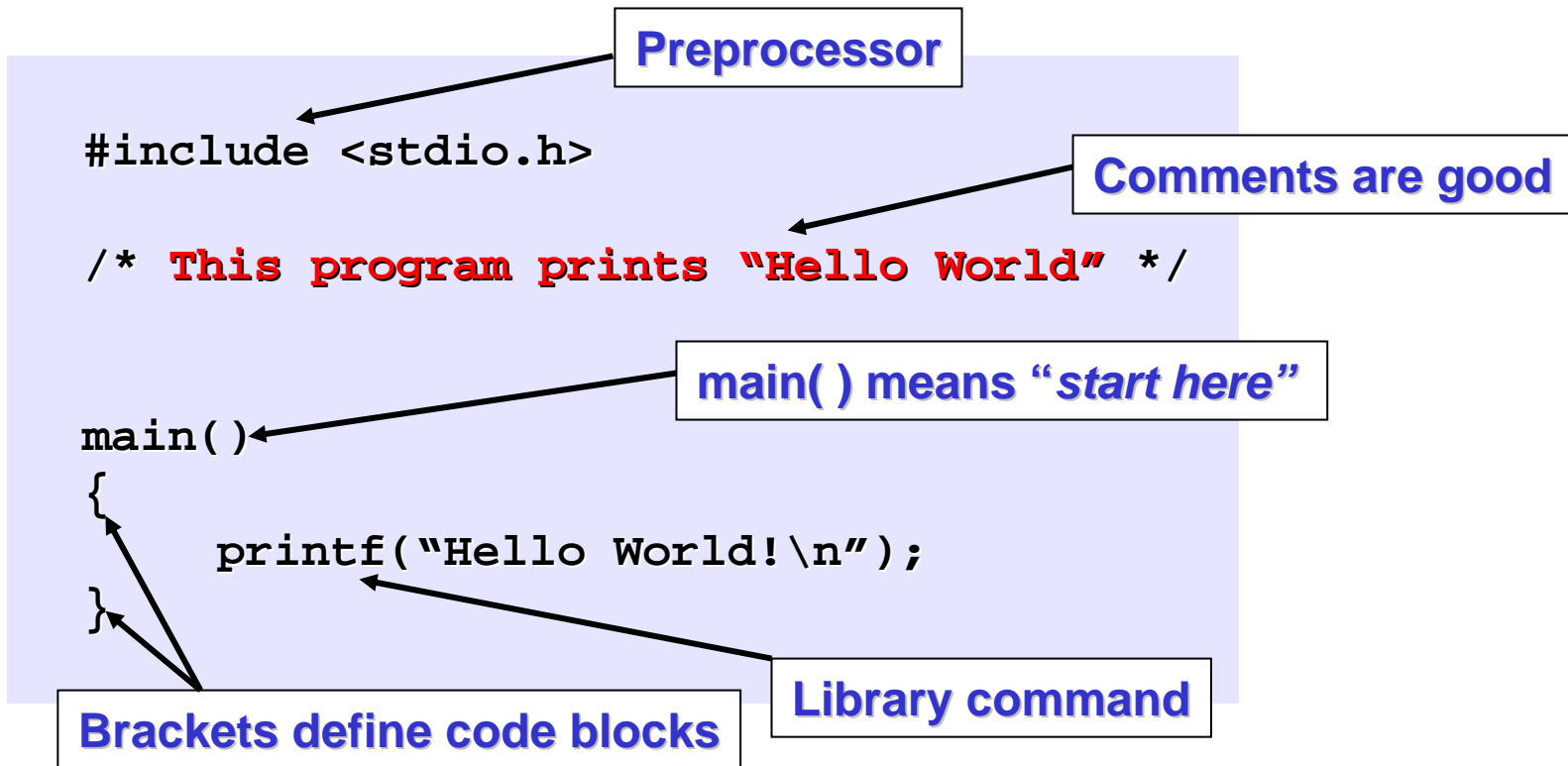
---

- C is *small* (only 32 keywords).
- C is *common* (lots of C code about).
- C is *stable* (the language doesn't change much).
- C is *quick running*.
- C is the *basis for many other languages* (Java, C++, awk, Perl).
- It may not feel like it but C is one of the easiest languages to learn.

# Some programmer jargon

- **Some words that will be used a lot:**
  - **Source code**: The stuff you type into the computer. The program you are writing.
  - **Compile (build)**: Taking source code and making a program that the computer can understand.
  - **Executable**: The compiled program that the computer can run.
  - **Language**: The core part of C central to writing C code.
  - **Library**: Added functions for C programming which are bolted on to do certain tasks.
  - **Header file**: Files ending in .h which are included at the start of source code.

# Our First C Program: *Hello World*



# C doesn't care much about spaces

```
#include <stdio.h> /* This program prints "Hello World" */
int main( ) {printf("Hello World!\n");}
```

```
#include <stdio.h>
/* This program
prints "Hello
World"
*/
int
main( )
{
printf("Hello
World!
\n")
;
}
```

**Both of these programs are exactly the same as the original as far as your compiler is concerned.**

# Keywords of C

- Flow control (6) – `if`, `else`, `return`, `switch`, `case`, `default`
- Loops (5) – `for`, `do`, `while`, `break`, `continue`
- Common *types* (5) – `int`, `float`, `double`, `char`, `void`
- *structures* (3) – `struct`, `typedef`, `union`
- Counting and sizing things (2) – `enum`, `sizeof`
- Rare but still useful *types* (7) – `extern`, `signed`, `unsigned`, `long`, `short`, `static`, `const`
- Evil keywords which we avoid (1) – `goto`
- Wierdies (3) – `auto`, `register`, `volatile`



# The C Character Set

- **The C language alphabet:**
  - Uppercase letters 'A' to 'Z'
  - Lowercase letters 'a' to 'z'
  - Digits '0' to '9'
  - Certain special characters:

!	#	%	^	&	*	(	)
-	_	+	=	~	[	]	\
	;	:	'	"	{	}	,
.	<	>	/	?	blank		

# Some simple operations for variables

- In addition to +, -, \* and / we can also use

**+=, -=, \*=, /=, -- and % (modulo)**

**n++**            *increment n*

**n--**            *decrement n*

**a+=5**            *is equivalent to*            **a = a+5;**

**a-=5**            *is equivalent to*            **a = a-5;**

**a\*=5**            *is equivalent to*            **a = a\*5;**

**a/=5**            *is equivalent to*            **a = a/5;**

**(x % y)** gives the remainder when **x** is divided by **y**

# Types of variable

---

- We must *declare* the *type* of every variable we use in C.
- Every variable has a *type* (e.g. `int`) and a *name*.
- This prevents some bugs caused by spelling errors (misspelling variable names).
- Declarations of types should always be together at the top of main or a function (see later).
- Other types are `char`, `signed`, `unsigned`, `long`, `short` and `const`.

# Identifiers and Keywords

---

- **Identifiers**

- Names given to various program elements (variables, constants, functions, etc.)
- May consist of *letters*, *digits* and the *underscore* ('\_') character, with no space between.
- First character must be a letter or underscore.
- An identifier can be arbitrary long.
  - Some C compilers recognize only the first few characters of the name (16 or 31).
- **Case sensitive**
  - 'area', 'AREA' and 'Area' are all different.

# Valid and Invalid Identifiers

- **Valid identifiers**

X

abc

simple\_interest

a123

LIST

stud\_name

Empl\_1

Empl\_2

avg\_empl\_salary

- **Invalid identifiers**

10abc

my-name

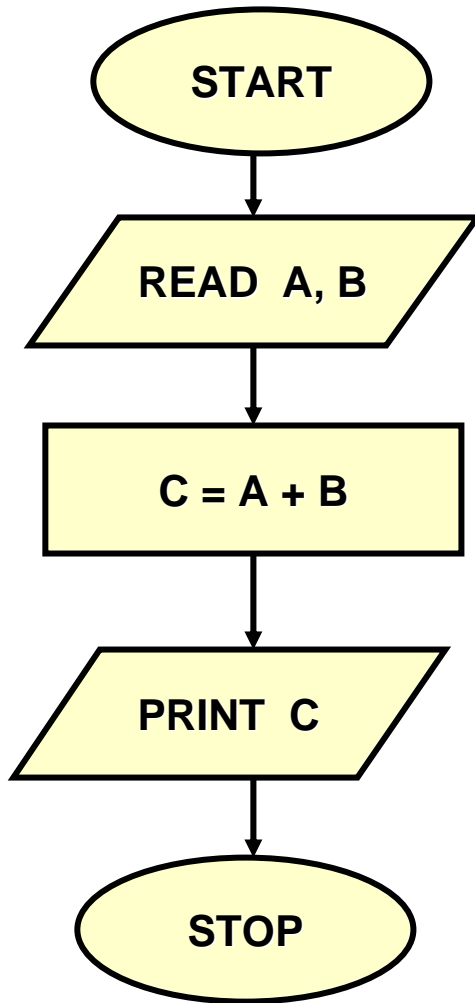
“hello”

simple interest

(area)

%rate

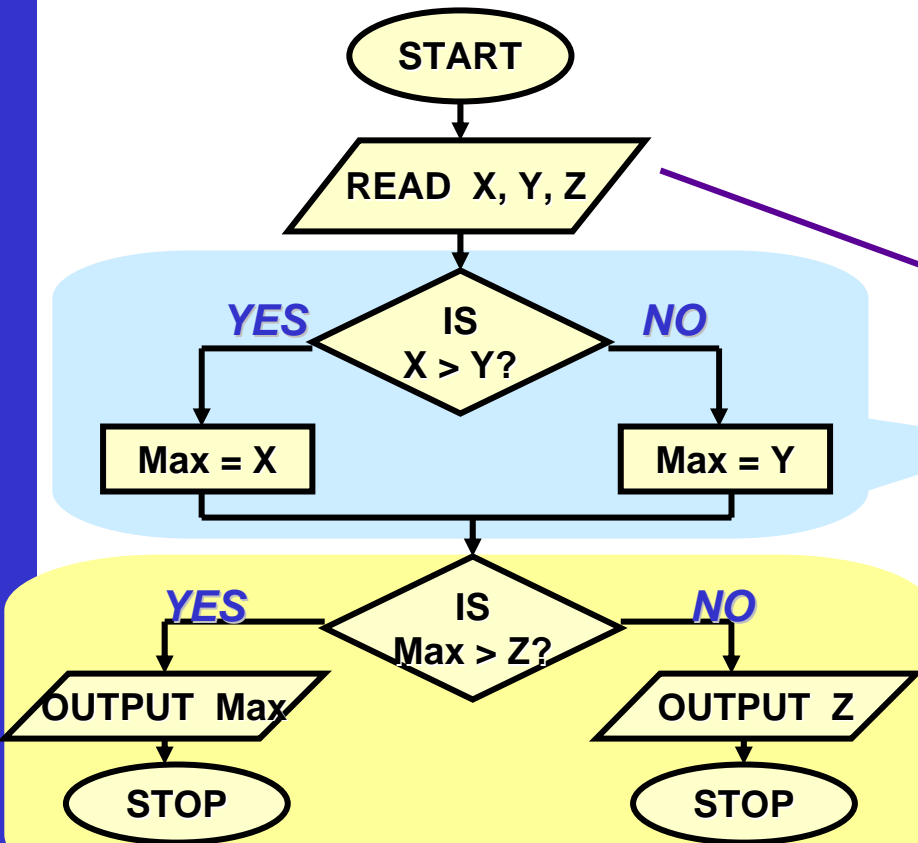
# Another Example: Adding two numbers



```
#include <stdio.h>
main()
{
    int a, b, c;
    scanf("%d%d", &a, &b);
    c = a + b;
    printf("%d", c);
}
```

Variable Declaration

# Example: Largest of three numbers



```
#include <stdio.h>
/* FIND THE LARGEST OF THREE NUMBERS */
main()
{
    int a, b, c, max;
    scanf ("%d %d %d", &x, &y, &z);

    if (x>y)
        max = x;
    else max = y;

    if (max > z)
        printf("Largest is %d", max);
    else printf("Largest is %d", z);
}
```

# Largest of three numbers: *Another way*

```
#include <stdio.h>

/* FIND THE LARGEST OF THREE NUMBERS */

main()
{
    int  a, b, c;
    scanf ("%d %d %d", &a, &b, &c);
    if ((a>b) && (a>c))  /* Composite condition check */
        printf ("\n Largest is %d", a);
    else
        if (b>c)        /* Simple condition check */
            printf ("\n Largest is %d", b);
        else
            printf ("\n Largest is %d", c);
}
```



# Use of functions: *Area of a circle*

```
#include <stdio.h>
```

```
#define PI 3.1415926
```

**Macro definition**

**Function definition**

```
/* Function to compute the area of a circle */
```

```
float myfunc (float r)
```

```
{ float a;
```

```
  a = PI * r * r;
```

```
  return (a); /* return result */
```

```
}
```

**Function argument**

```
main()
```

```
{ float radius, area;
```

```
  float myfunc (float radius);
```

**Function declaration**

**(return value defines the type)**

```
  scanf ("%f", &radius);
```

```
  area = myfunc (radius);
```

**Function call**

```
  printf ("\n Area is %f \n", area);
```

```
}
```

# Structure of a C program

---

- Every C program consists of one or more functions.
  - One of the functions must be called *main*.
  - The program will always begin by executing the main function.
- Each function must contain:
  - A function *heading*, which consists of the function *name*, followed by an optional list of *arguments* enclosed in parentheses.
  - A list of argument *declarations*.
  - A *compound statement*, which comprises the remainder of the function.

# Desirable Programming Style

- **Clarity**
  - The program should be clearly written.
  - It should be easy to follow the program logic.
- **Meaningful variable names**
  - Make variable/constant names meaningful to enhance program clarity.
    - 'area' instead of 'a'
    - 'radius' instead of 'r'
- **Program documentation**
  - Insert comments in the program to make it easy to understand.
  - Never use too many comments.
- **Program indentation**
  - Use proper indentation.
  - Structure of the program should be immediately visible.

# Indentation Example: *Good Style*

```
#include <stdio.h>

/* FIND THE LARGEST OF THREE NUMBERS */

main()
{
    int  a, b, c;

    scanf("%d%d%d", &a, &b, &c);

    if ((a>b) && (a>c))
        printf("\n Largest is %d", a);
    else
        if (b>c)
            printf("\n Largest is %d", b);
        else
            printf("\n Largest is %d", c);
}
```

# Indentation Example: *Bad Style*

---

```
#include <stdio.h>

/* FIND THE LARGEST OF THREE NUMBERS */
main()
{
int  a, b, c;
scanf("%d%d%d", &a, &b, &c);
if ((a>b) && (a>c))
printf("\n Largest is %d", a);
    else
if (b>c)
    printf("\n Largest is %d", b);
else
printf("\n Largest is %d", c);
}
```

# Data Types in C

---

**int** :: integer quantity

Typically occupies 4 bytes (32 bits) in memory.

**char** :: single character

Typically occupies 1 byte (8 bits) in memory.

**float** :: floating-point number (a number with a decimal point)

Typically occupies 4 bytes (32 bits) in memory.

**double** :: double-precision floating-point number

# Contd.

---

- **Some of the basic data types can be augmented by using certain data type qualifiers:**
  - **short**
  - **long**
  - **signed**
  - **unsigned**
- **Typical examples:**
  - **short int**
  - **long int**
  - **unsigned int**

# Some Examples of Data Types

- **int**

0, 25, -156, 12345, -99820

- **char**

'a', 'A', '\*', '/', ''

- **float**

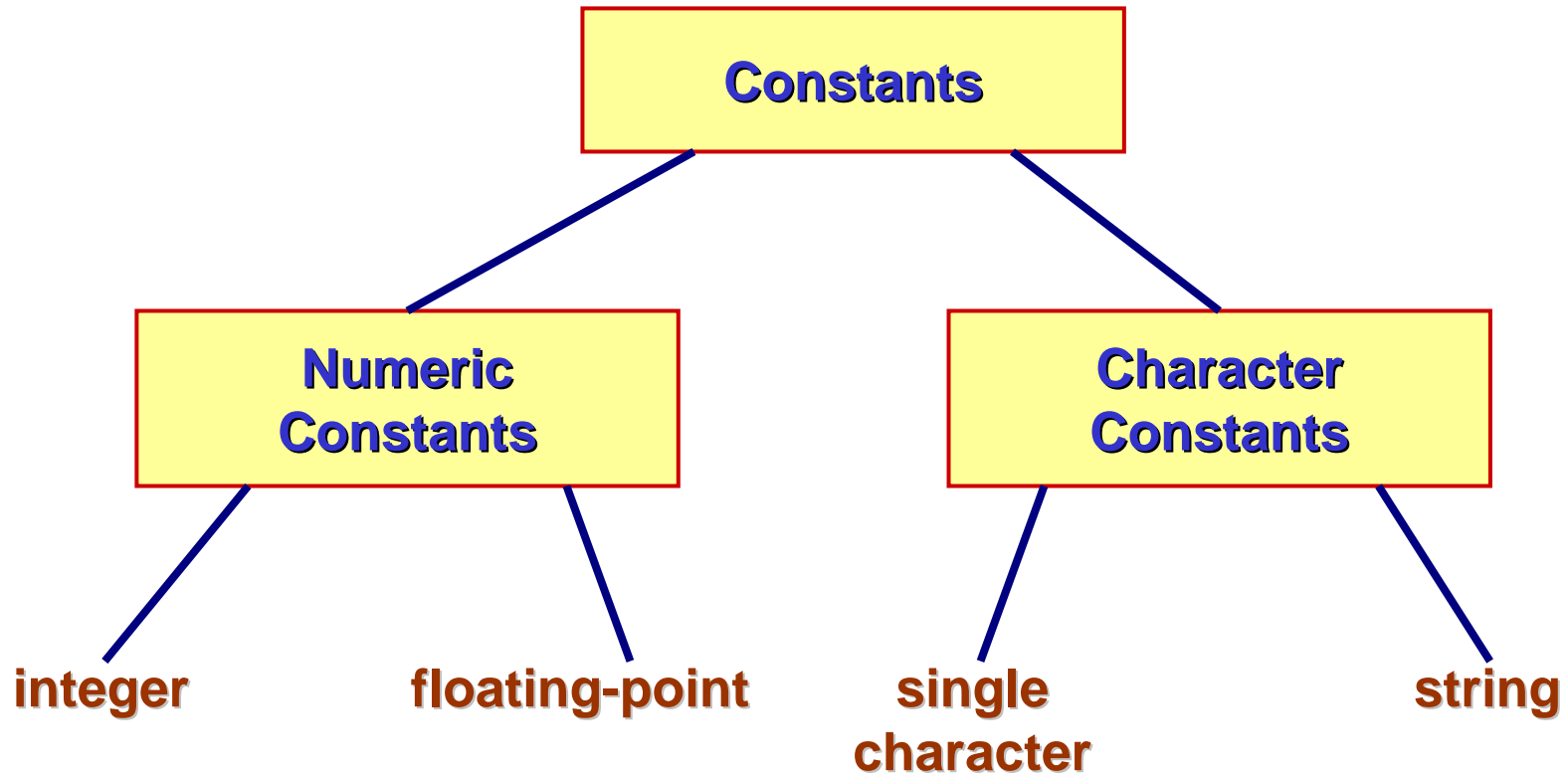
23.54, -0.00345, 25.0

2.5E12, 1.234e-5

E or e means "10 to the power of"



# Constants



# Integer Constants

---

- **Consists of a sequence of digits, with possibly a plus or a minus sign before it.**
  - **Embedded spaces, commas and non-digit characters are not permitted between digits.**
- **Maximum and minimum values (for 32-bit representations)**
  - Maximum :: 2147483647**
  - Minimum :: -2147483648**

# Floating-point Constants

- Can contain fractional parts.
- Very large or very small numbers can be represented.  
**23000000 can be represented as 2.3e7**
- Two different notations:
  1. **Decimal notation**  
**25.0, 0.0034, .84, -2.234**
  2. **Exponential (scientific) notation**  
**3.45e23, 0.123e-12, 123E2**

e means “10 to the power of”

# Single Character Constants

- Contains a single character enclosed within a pair of single quote marks.
  - Examples :: '2', '+', 'Z'
- Some special backslash characters
  - '\n'      new line
  - '\t'      horizontal tab
  - '\''      single quote
  - '\"'      double quote
  - '\\'      backslash
  - '\0'      null

# String Constants

- **Sequence of characters enclosed in double quotes.**
  - **The characters may be letters, numbers, special characters and blank spaces.**
- **Examples:**
  - **“nice”, “Good Morning”, “3+6”, “3”, “C”**
- **Differences from character constants:**
  - **‘C’ and “C” are not equivalent.**
  - **‘C’ has an equivalent integer value while “C” does not.**

# Declaration of Variables

- **There are two purposes:**
  1. It tells the compiler what the variable name is.
  2. It specifies what type of data the variable will hold.
- **General syntax:**

**data-type variable-list;**
- **Examples:**

```
int velocity, distance;  
int a, b, c, d;  
float temp;  
char flag, option;
```

# A First Look at Pointers

- A variable is assigned a specific memory location.
  - For example, a variable **speed** is assigned memory location **1350**.
  - Also assume that the memory location contains the data value **100**.
  - When we use the name **speed** in an expression, it refers to the value **100** stored in the memory location.

`distance = speed * time;`

- Thus every variable has an **address** (in memory), and its **contents**.

# Contd.

---

- In C terminology, in an expression  
**speed** refers to the **contents** of the memory location.  
**&speed** refers to the **address** of the memory location.
- Examples:  

```
printf ("%f %f %f", speed, time, distance);  
scanf ("%f %f", &speed, &time);
```



# An Example

---

```
#include <stdio.h>
main()
{
    float speed, time, distance;

    scanf ("%f %f", &speed, &time);
    distance = speed * time;
    printf ("\n The distance traversed is: \n", distance);
}
```

# Assignment Statement

- Used to assign values to variables, using the assignment operator (=).

- General syntax:

`variable_name = expression;`

- Examples:

`velocity = 20;`

`b = 15; temp = 12.5;`

`A = A + 10;`

`v = u + f * t;`

`s = u * t + 0.5 * f * t * t;`

# Contd.

---

- A value can be assigned to a variable at the time the variable is declared.

```
int speed = 30;
```

```
char flag = 'y';
```

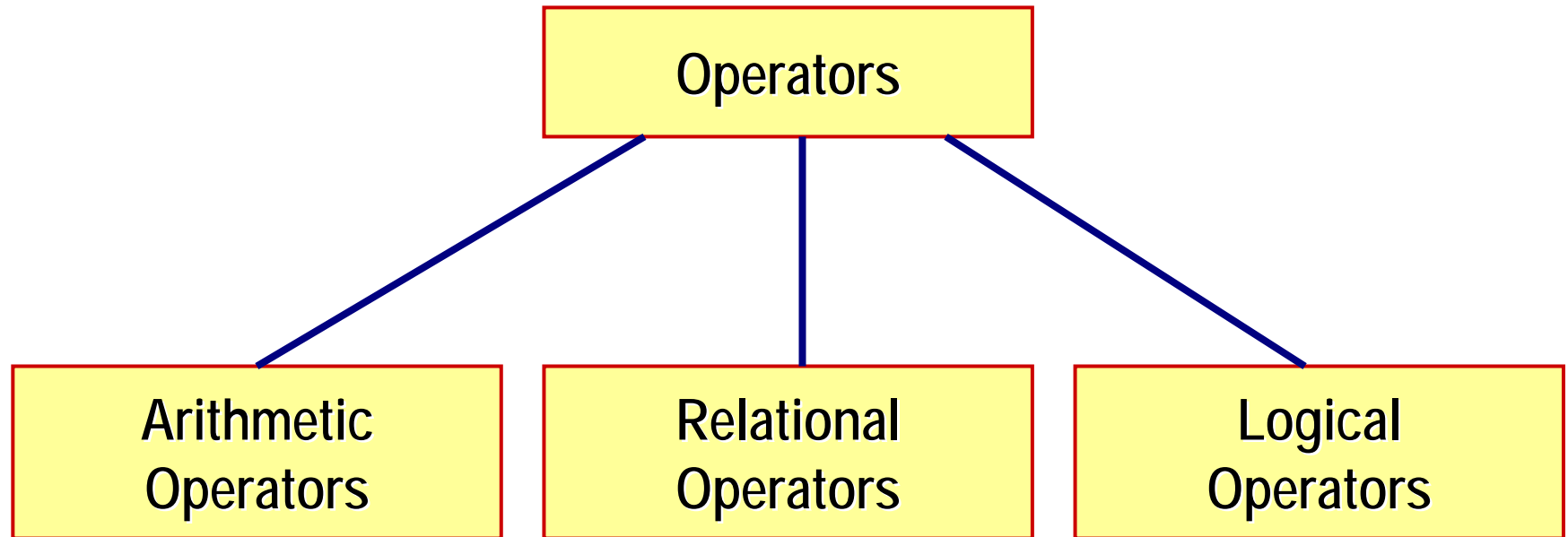
- Several variables can be assigned the same value using multiple assignment operators.

```
a = b = c = 5;
```

```
flag1 = flag2 = 'y';
```

```
speed = flow = 0.0;
```

# Operators in Expressions



# Arithmetic Operators

- Addition :: +
- Subtraction :: -
- Division :: /
- Multiplication :: \*
- Modulus :: %

## Examples:

**distance = rate \* time ;**

**netIncome = income - tax ;**

**speed = distance / time ;**

**area = PI \* radius \* radius;**

**y = a \* x \* x + b\*x + c;**

**quotient = dividend / divisor;**

**remain =dividend % divisor;**

## Contd.

- Suppose  $x$  and  $y$  are two integer variables, whose values are 13 and 5 respectively.

$x + y$	18
$x - y$	8
$x * y$	65
$x / y$	2
$x \% y$	3

# Operator Precedence

- In decreasing order of priority
  1. Parentheses :: ( )
  2. Unary minus :: -5
  3. Multiplication, Division, and Modulus
  4. Addition and Subtraction
- For operators of the *same priority*, evaluation is from *left to right* as they appear.
- Parenthesis may be used to change the precedence of operator evaluation.

## Examples: Arithmetic expressions

$$a + b * c - d / e \quad \rightarrow \quad a + (b * c) - (d / e)$$

$$a * -b + d \% e - f \quad \rightarrow \quad a * (-b) + (d \% e) - f$$

$$a - b + c + d \quad \rightarrow \quad (((a - b) + c) + d)$$

$$x * y * z \quad \rightarrow \quad ((x * y) * z)$$

$$a + b + c * d * e \quad \rightarrow \quad (a + b) + ((c * d) * e)$$



# Integer Arithmetic

---

- When the operands in an arithmetic expression are integers, the expression is called *integer expression*, and the operation is called *integer arithmetic*.
- Integer arithmetic always yields integer values.

# Real Arithmetic

---

- Arithmetic operations involving only real or floating-point operands.
- Since floating-point values are rounded to the number of significant digits permissible, the final value is an approximation of the final result.  
 **$1.0 / 3.0 * 3.0$  will have the value  $0.99999$  and not  $1.0$**
- The modulus operator cannot be used with real operands.

# Mixed-mode Arithmetic

---

- When one of the operands is integer and the other is real, the expression is called a *mixed-mode* arithmetic expression.
- If either operand is of the real type, then only real arithmetic is performed, and the result is a real number.

$$25 / 10 \rightarrow 2$$

$$25 / 10.0 \rightarrow 2.5$$

- Some more issues will be considered later.

# Type Casting

---

```
int a=10, b=4, c;
```

```
float x, y;
```

```
c = a / b;
```

```
x = a / b;
```

```
y = (float) a / b;
```

*The value of c will be 2*

*The value of x will be 2.0*

*The value of y will be 2.5*

# Relational Operators

---

- Used to compare two quantities.

**<** is less than

**>** is greater than

**<=** is less than or equal to

**>=** is greater than or equal to

**==** is equal to

**!=** is not equal to

# Examples

---

**10 > 20** is false

**25 < 35.5** is true

**12 > (7 + 5)** is false

- When arithmetic expressions are used on either side of a relational operator, the arithmetic expressions will be evaluated first and then the results compared.

**$a + b > c - d$  is the same as  $(a+b) > (c+d)$**

# Examples

---

- Sample code segment in C

```
if (x > y)
    printf ("%d is larger\n", x);
else
    printf ("%d is larger\n", y);
```

# Logical Operators

---

- There are two logical operators in C (also called logical connectives).
  - && → Logical AND**
  - || → Logical OR**
- What they do?
  - They act upon operands that are themselves logical expressions.
  - The individual logical expressions get combined into more complex conditions that are true or false.



# Logical Operators

## – Logical AND

- Result is true if both the operands are true.

## – Logical OR

- Result is true if at least one of the operands are true.

X	Y	X && Y	X    Y
FALSE	FALSE	FALSE	FALSE
FALSE	TRUE	FALSE	TRUE
TRUE	FALSE	FALSE	TRUE
TRUE	TRUE	TRUE	TRUE

# Input / Output

- **printf**

- **Performs output to the standard output device (typically defined to be the screen).**
- **It requires a format string in which we can specify:**
  - **The text to be printed out.**
  - **Specifications on how to print the values.**  
`printf ("The number is %d.\n", num) ;`
  - **The format specification %d causes the value listed after the format string to be embedded in the output as a decimal number in place of %d.**
  - **Output will appear as: The number is 125.**

# Input / Output

- **scanf**

- Performs input from the standard input device, which is the keyboard by default.
- It requires a format string and a list of variables into which the value received from the input device will be stored.
- It is required to put an ampersand (&) before the names of the variables.

```
scanf ("%d", &size) ;
```

```
scanf ("%c", &nextchar) ;
```

```
scanf ("%f", &length) ;
```

```
scanf ("%d %d", &a, &b);
```