# Programming and Data Structure

Palash Dey
Department of Computer Science & Engg.
Indian Institute of Technology
Kharagpur

Slides credit: Prof. Indranil Sen Gupta

## Some General Announcements

## About the Course

- Will be conducted with a L-T-P rating of 3-0-0.
- Laboratory with a L-T-P of 0-1-3.
- Evaluation in the theory course:

| | |
|---|---|
| – Mid-semester | 30% |
| – End-semester | 50% |
| – Two class tests and attendance | 20% |

## Course Materials

- The slides for the lectures will be made available on the web (in PDF form).

  http://cse.iitkgp.ac.in/~pds/current/

- All important announcements will be put up on the web page.

## ATTENDANCE IN THE CLASSES IS MANDATORY

Students having poor attendance will be penalized in terms of the final grade.

## Text/Reference Books & Notes

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.

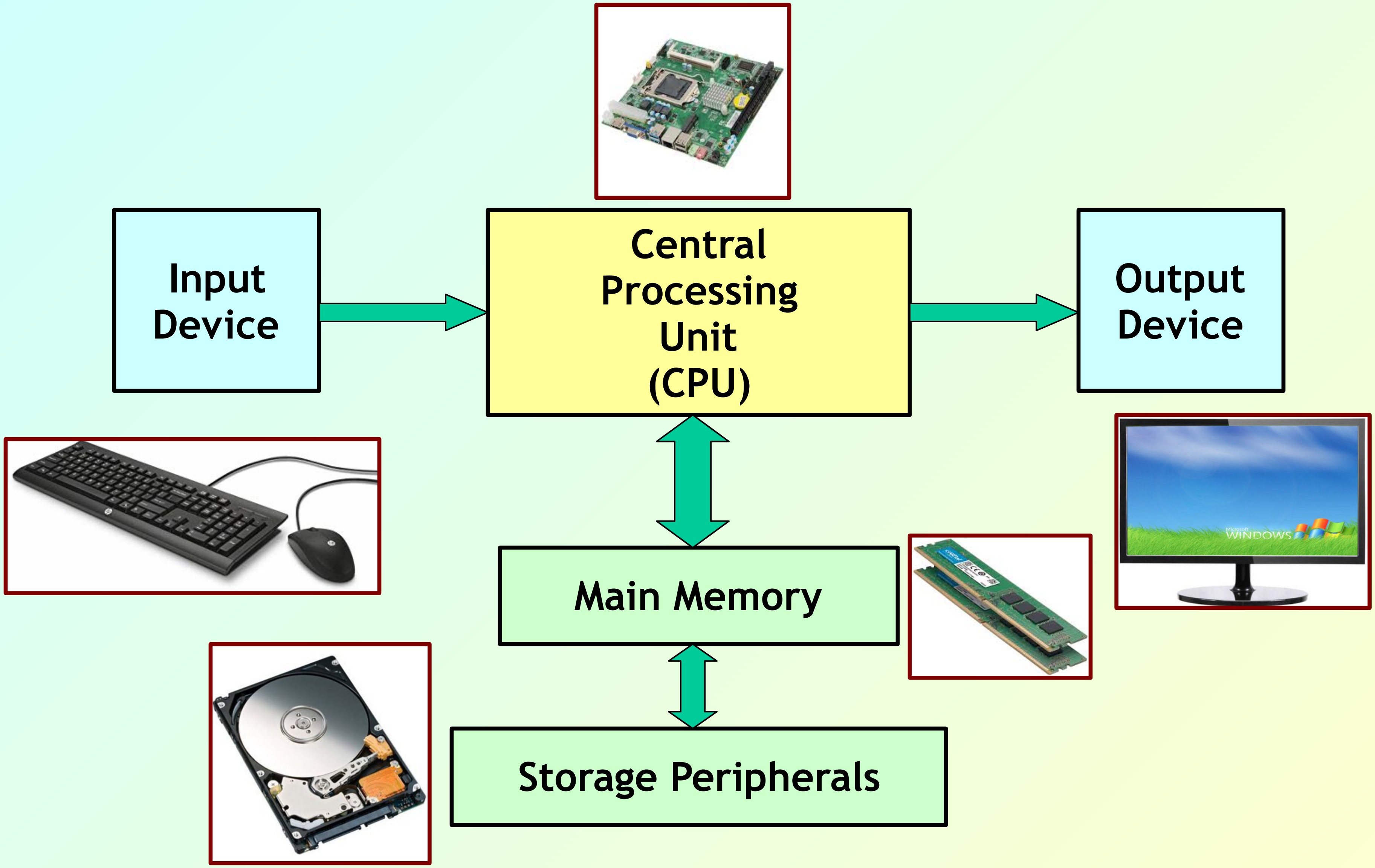
4. Data structures using C and C++ (Second Edition)

   Y. Langsam, M.J. Augenstein, A.M. Tanenbaum, Prentice-Hall of India.

5. `http://cse.iitkgp.ac.in/~pds/notes/`

## Introduction

## What is a Computer?



- CPU
  - All computations take place here in order for the computer to perform a designated task.
  - It has a number of registers which temporarily store data and programs (instructions).
  - It has circuitry to carry out arithmetic and logic operations, take decisions, etc.
  - It retrieves instructions from the memory (fetch), interprets (decode) them, and performs the requested operation (execute).

- Main Memory
  - Uses semiconductor technology.
  - Memory sizes in the range of 4 to 16 Gbytes are typical today.
  - Some measures to be remembered

| | |
|---|---|
| 1 K (kilo) | = $2^{10}$ (= 1024) |
| 1 M (mega) | = $2^{20}$ (= one million approx.) |
| 1 G (giga) | = $2^{30}$ (= one billion approx.) |
| 1 T (tera) | = $2^{40}$ (= one trillion approx.) |
| 1 P (peta) | = $2^{50}$ |

- Input Device
  - Keyboard, Mouse, Scanner
- Output Device
  - Monitor, Printer
- Storage Peripherals
  - Magnetic Disks: hard disk, floppy disk
    - Allows direct (semi-random) access
  - Optical Disks: CDROM, CD-RW, DVD, BlueRay
    - Allows direct (semi-random) access
  - Flash Memory and Solid State Drive
    - Allows direct access
  - Magnetic Tape: DAT
    - Only sequential access

## 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.

- Example:

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

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

## Contd.

- A digital computer is built out of tiny electronic switches.
  - From the viewpoint of ease of manufacturing and reliability, such switches can be in one of two states, ON and OFF.
  - A switch can represent a digit in the so-called *binary number system*, 0 and 1.
- A computer works based on the binary number system.

- Binary number system
  - Two digits :: 0 and 1
  - Every digit position has a weight which is a power of 2.

- <u>Example</u>:

$$1110 = 1 \times 2^3 + 1 \times 2^2 + 1 \times 2^1 + 0 \times 2^0$$
$$= 14 \text{ (in decimal)}$$

# Concept of 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-digit 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

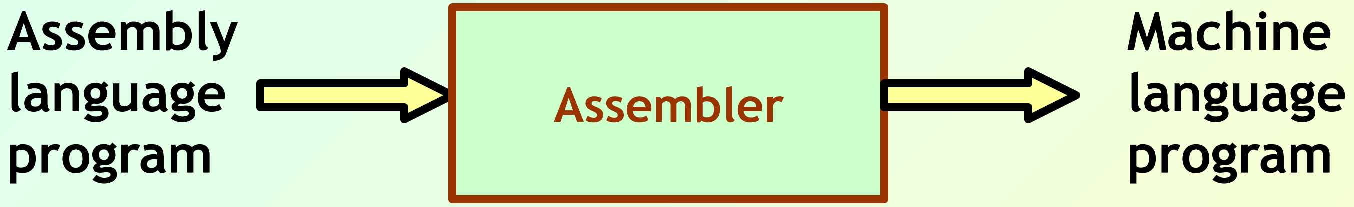
- Two categories:
  1. Application Software
     - Used to solve a particular problem.
     - Editor, financial accounting, weather forecasting, mathematical toolbox, etc.
  2. System Software
     - Helps in running other programs.
     - Compiler, operating system, etc.

## Computer Languages

- Machine Language
  - Expressed in binary.
    - 10110100 may mean ADD, 01100101 may mean SUB, etc.
  - 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.

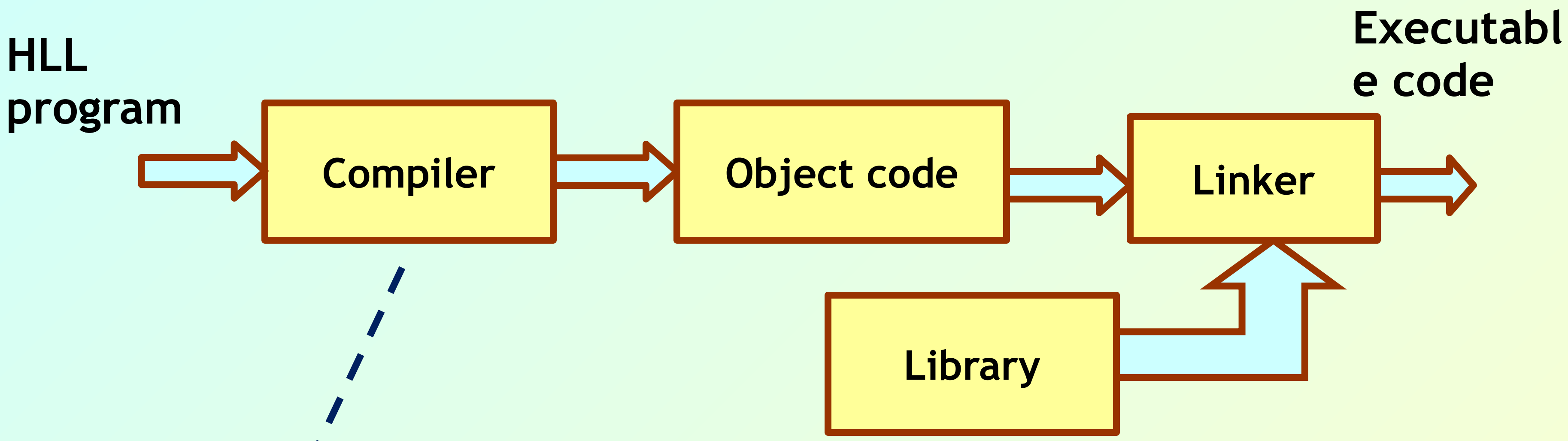
```
MOV    A,X      ;  A = X
ADD    A,Y      ;  A = A + Y
ADD    A,Z      ;  A = A + Z
DIV    A,3      ;  A = A / 3
MOV    RES,A    ;  RES = A
```

**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, C, C++, Java, Python.
  - Requires an elaborate process of translation.
    - Using a software called *compiler*.
  - They are portable across platforms.



gcc compiler will be used in the lab classes

## Operating Systems

- Makes the computer easy to use.
  - Basically the computer is very difficult to use.
  - Understands only machine language.
- Operating systems makes the task of the users easier.
- Categories of operating systems:
  - Single user
  - Multi user (Time sharing, Multitasking, Real time)

## Contd.

- Popular operating systems:

| | |
|---|---|
| – Windows: | single-user multitasking |
| – Unix: | multi-user |
| – Linux: | a free version of Unix |

- The laboratory classes will be based on Linux.

## Contd.

- Question:
  - How many users can work on the same computer?
- 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

## Some Terminologies

- Algorithm / Flowchart / Pseudo-code
  - A step-by-step procedure for solving a particular problem.
  - Should be 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.

## First Look at a C Program

```
/* Program to compute the area of a circle */
#include  <stdio.h>    /* Compulsory, for library files
   */
main()                 /* Function heading */
{
       float  radius, area;     /* Declare variables */
       scanf ("%f", &radius);  /* Read radius */
       area = 3.14159 * radius * radius;
       printf ("Area = %f", area); /* Output to screen
   */
}
```

```
Input:        10
Output:       Area = 314.158997
```

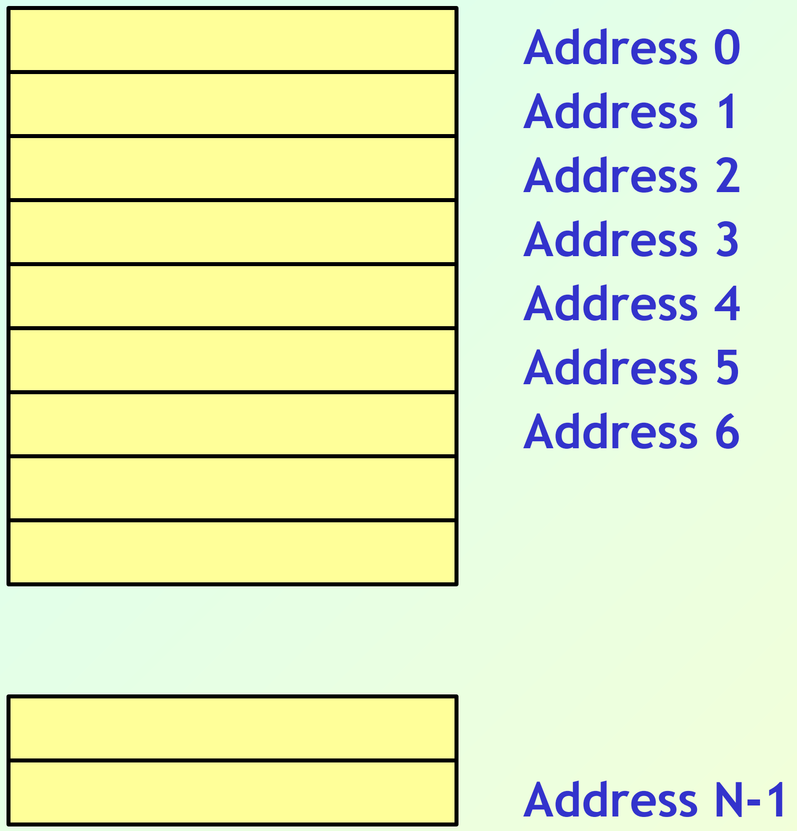
## Variables and Constants

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

## Contd.

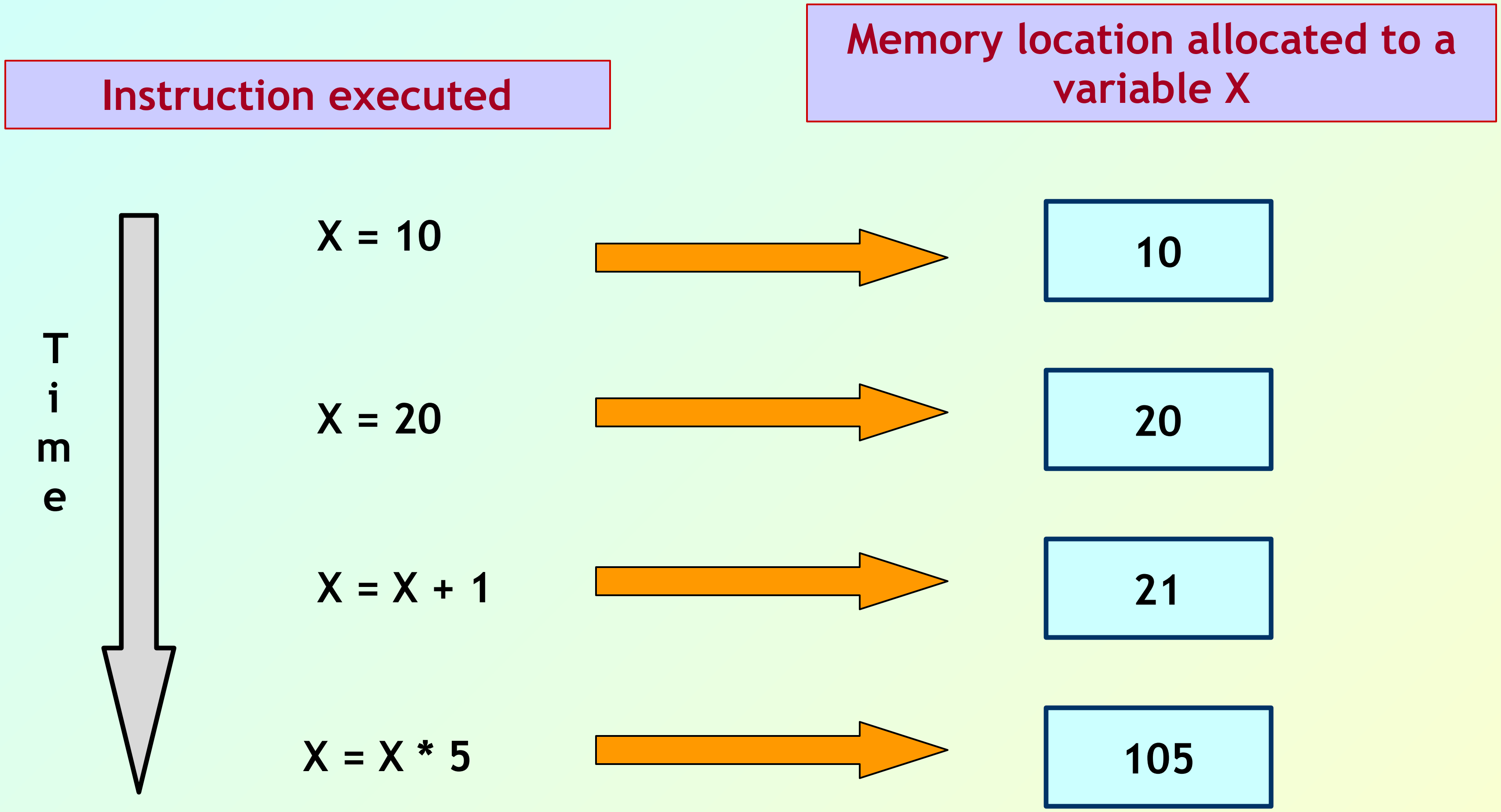
- How does memory look (logically)?
  - As a list of storage locations, each having a unique address.
  - Variables and constants are stored in these storage locations.
  - Variable is like a ***house***, and the name of a variable is like the ***address*** of the house.
    - Different people may reside in the house, which is like the ***contents*** of a variable.

## Memory map

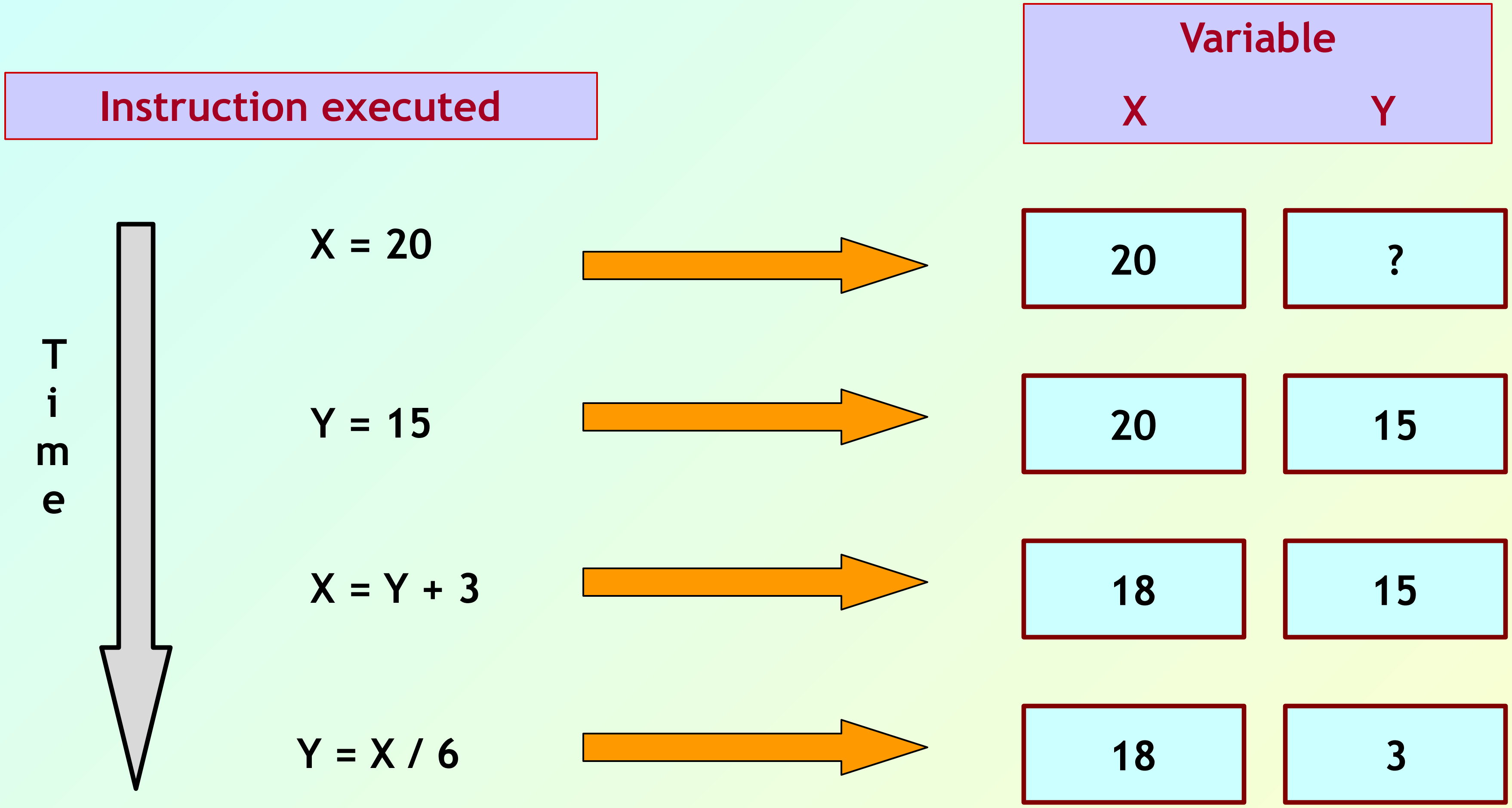


Every variable is mapped to a particular memory address

# Variables in Memory



# Variables in Memory (contd.)



| Instruction executed | Variable X | Variable Y |
|---|---|---|
| X = 20 | 20 | ? |
| Y = 15 | 20 | 15 |
| X = Y + 3 | 18 | 15 |
| Y = X / 6 | 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, -12345.345, 2.65E12, 2.35E-25
  - Character :: can store a single character

    Examples: 'A', 'a', '*', '3', ' ', '+'

## Data Types (contd.)

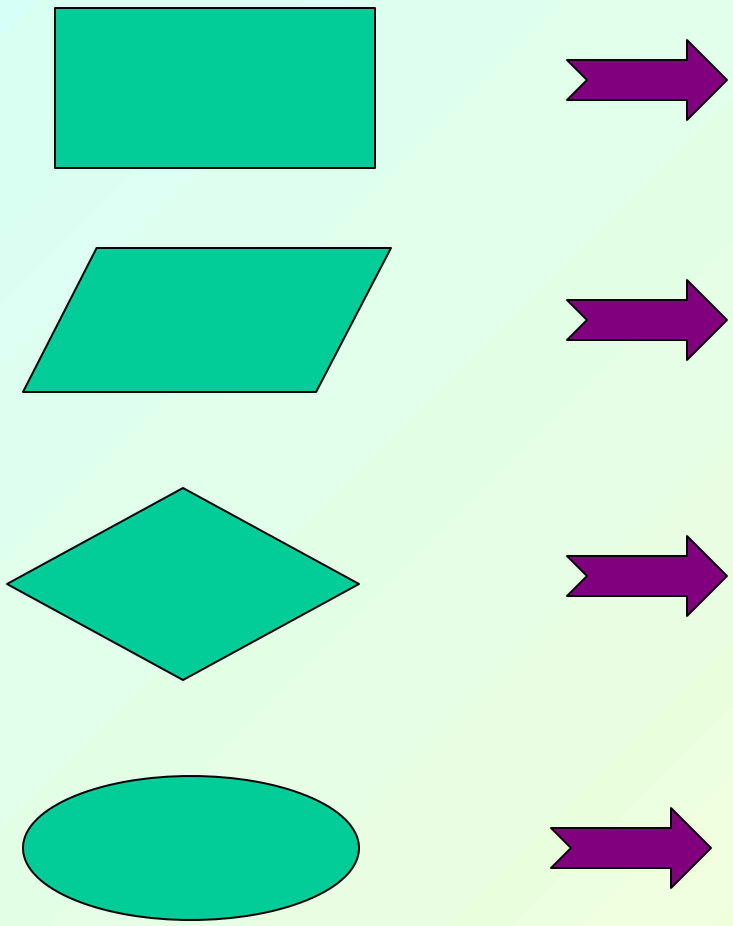
- How are they stored in memory?
  - Integer ::
    - 16 bits
    - 32 bits
  - Float ::
    - 32 bits
    - 64 bits
  - Char ::
    - 8 bits (ASCII code)
    - 16 bits (UNICODE, used in Java)

Actual number of bits varies from one computer to another

## Problem solving (Typical Flow)

- 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**

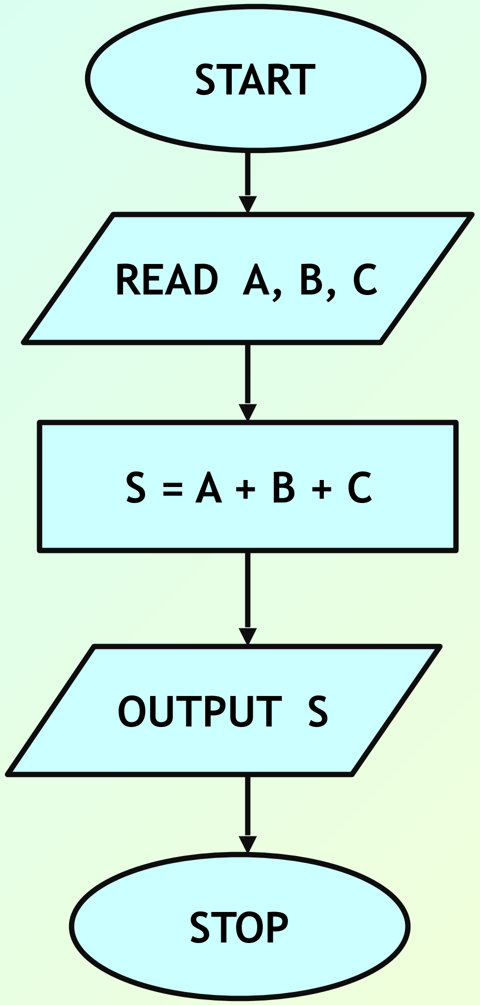
**Decision Box**

**Start / Stop**

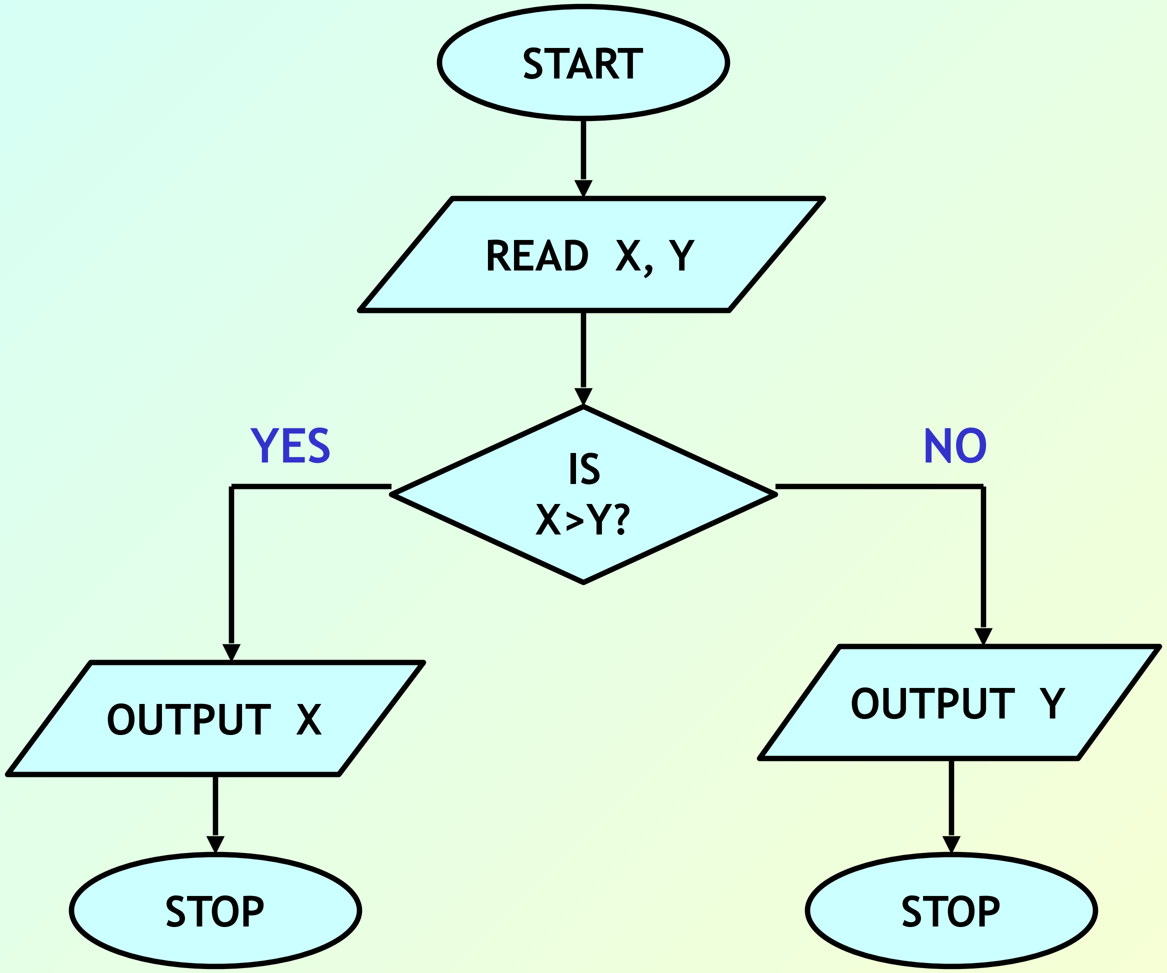
# Contd.

| | |
|---|---|
| ↓ | Flow of control |
| ● | Connector |

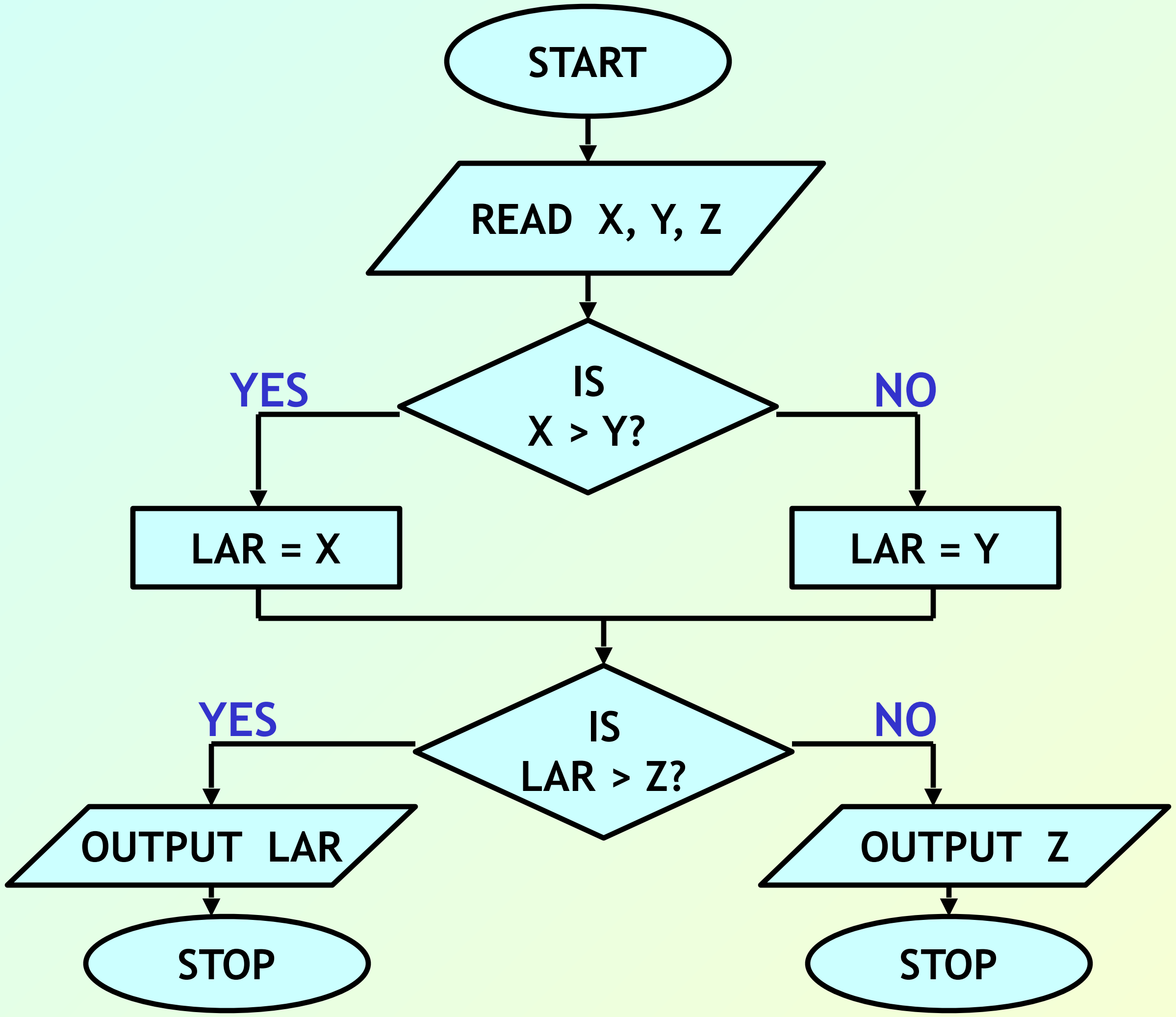
## Example 1: *Adding three numbers*



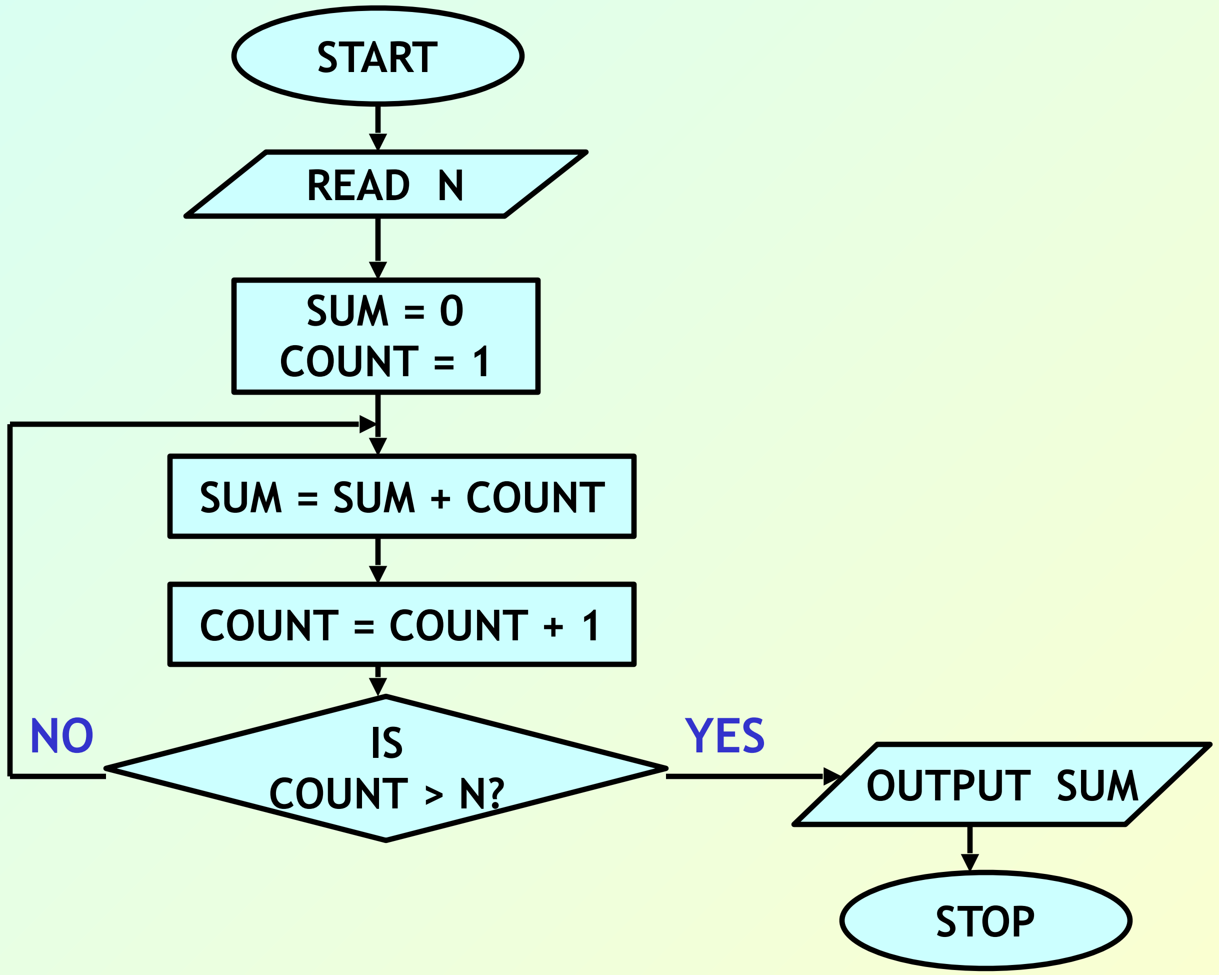
## Example 2: *Larger of two numbers*



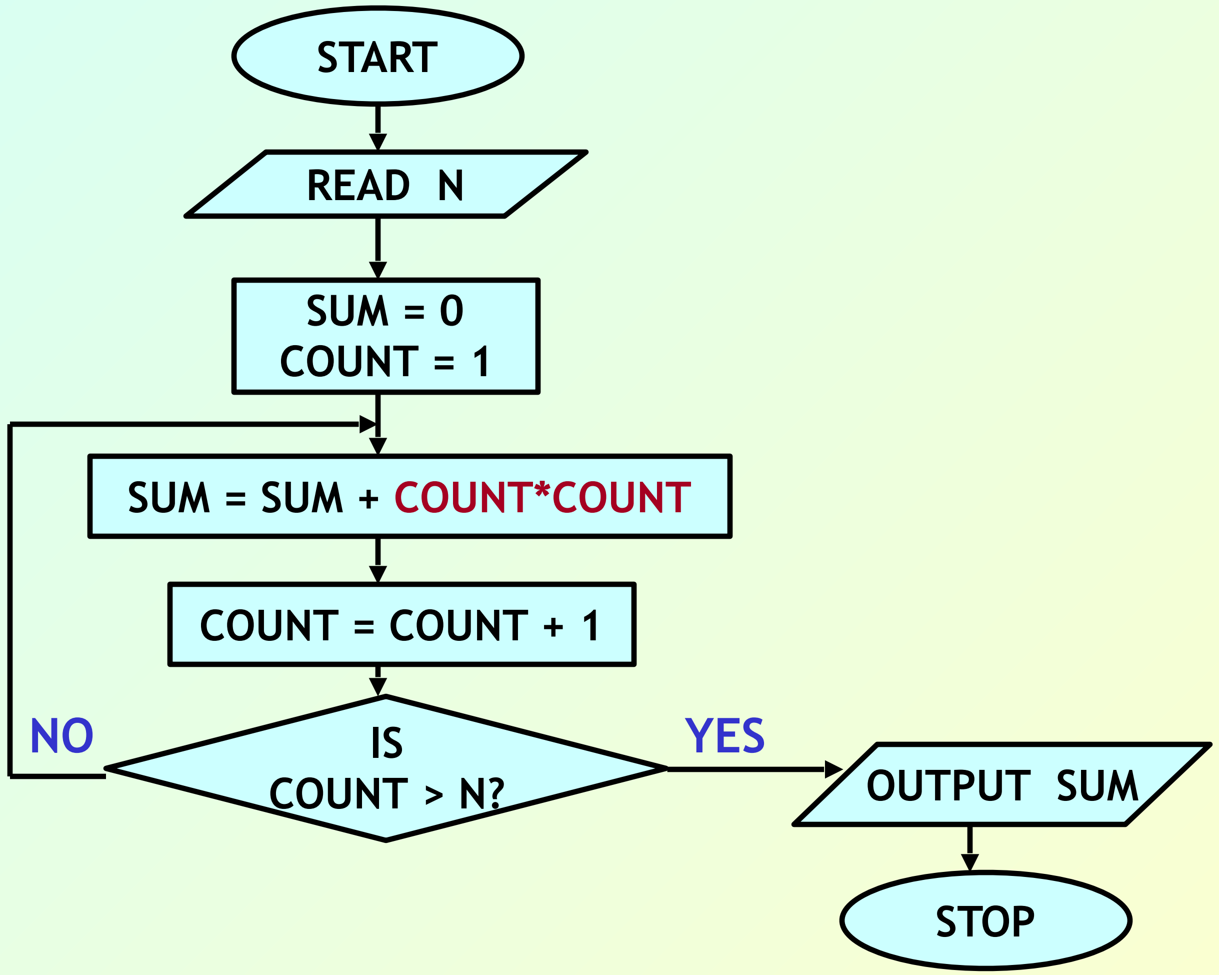
## Example 3: *Largest of three numbers*



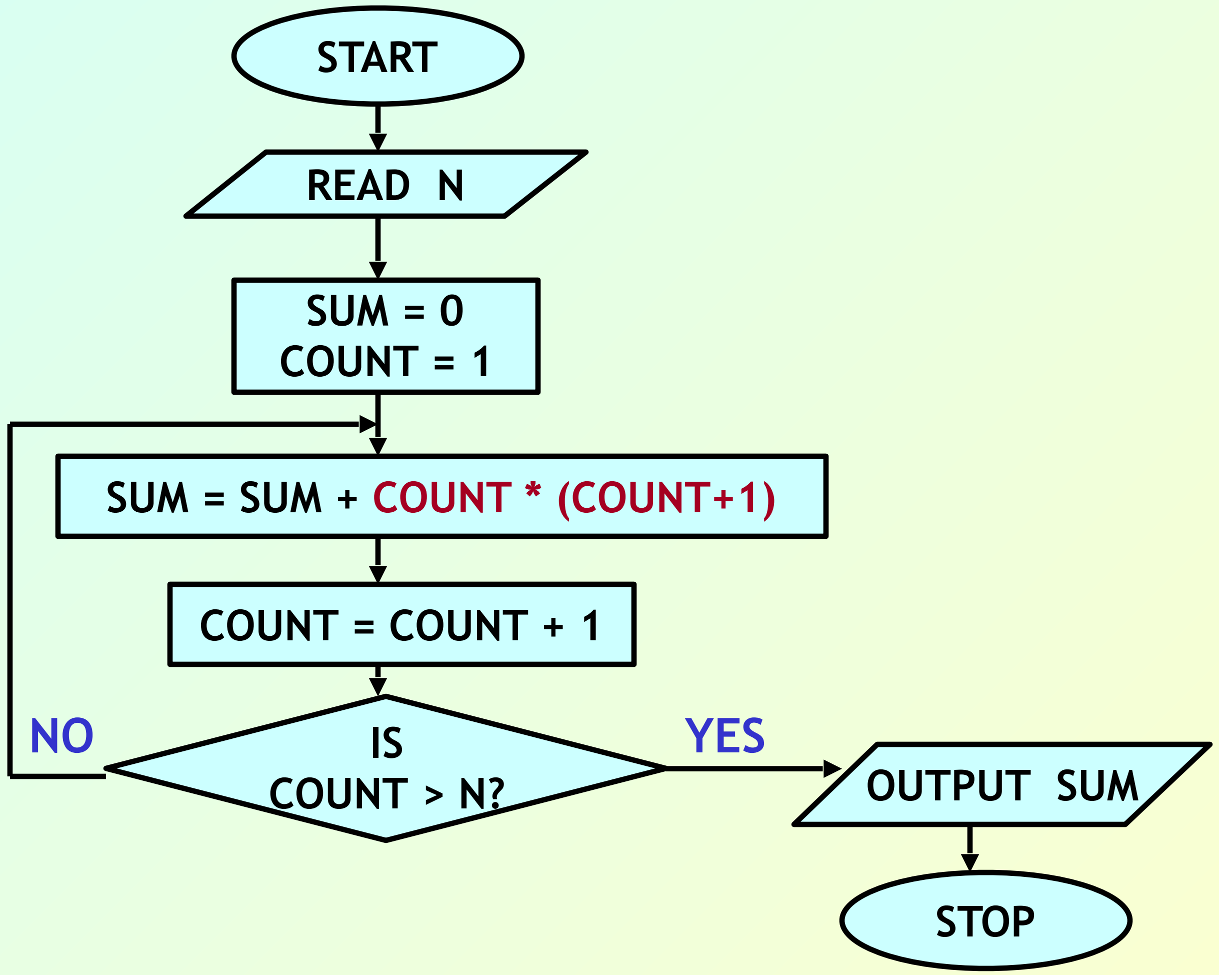
## Example 4: *Sum of first N natural numbers*



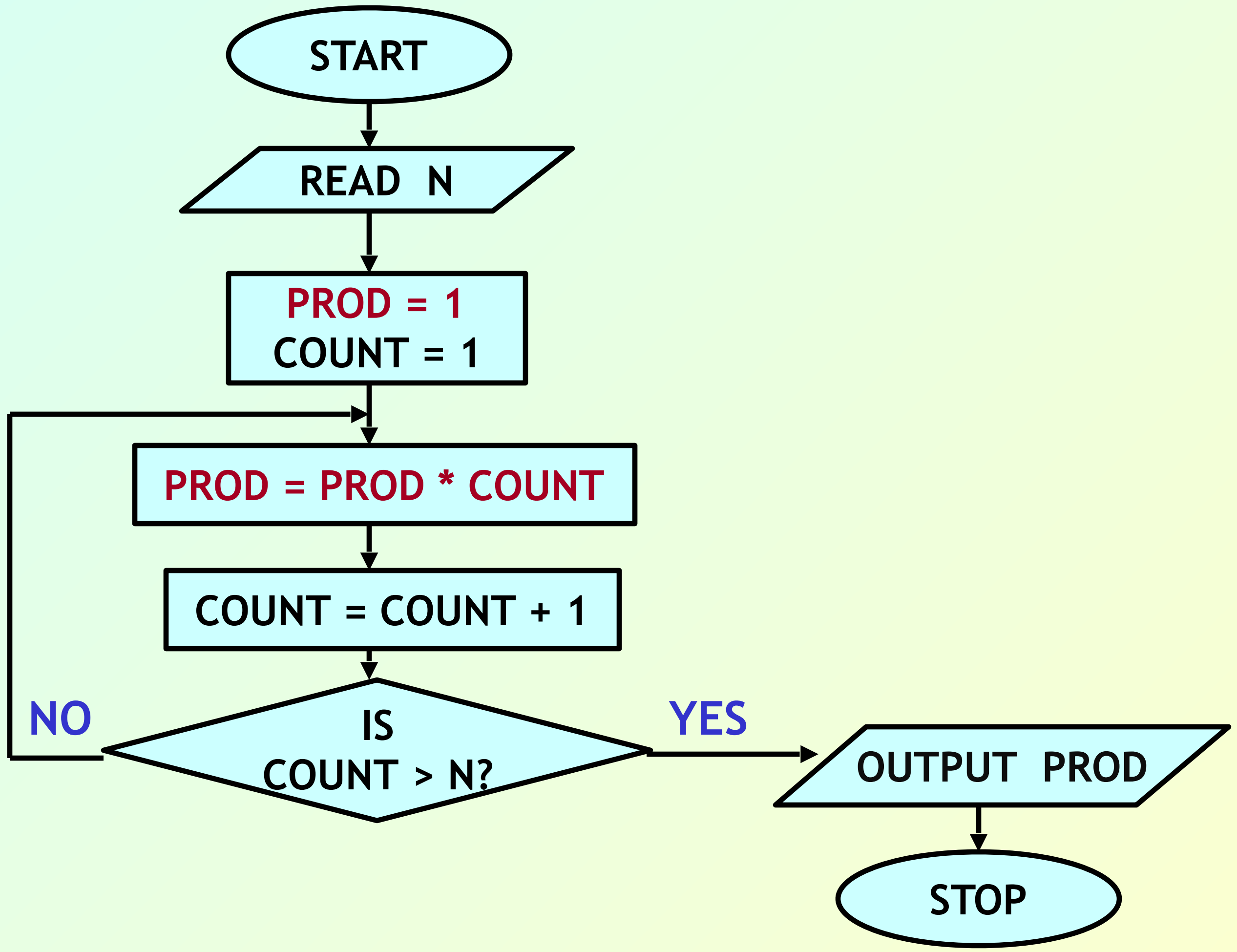
## Example 5: $SUM = 1^2 + 2^2 + 3^2 + N^2$



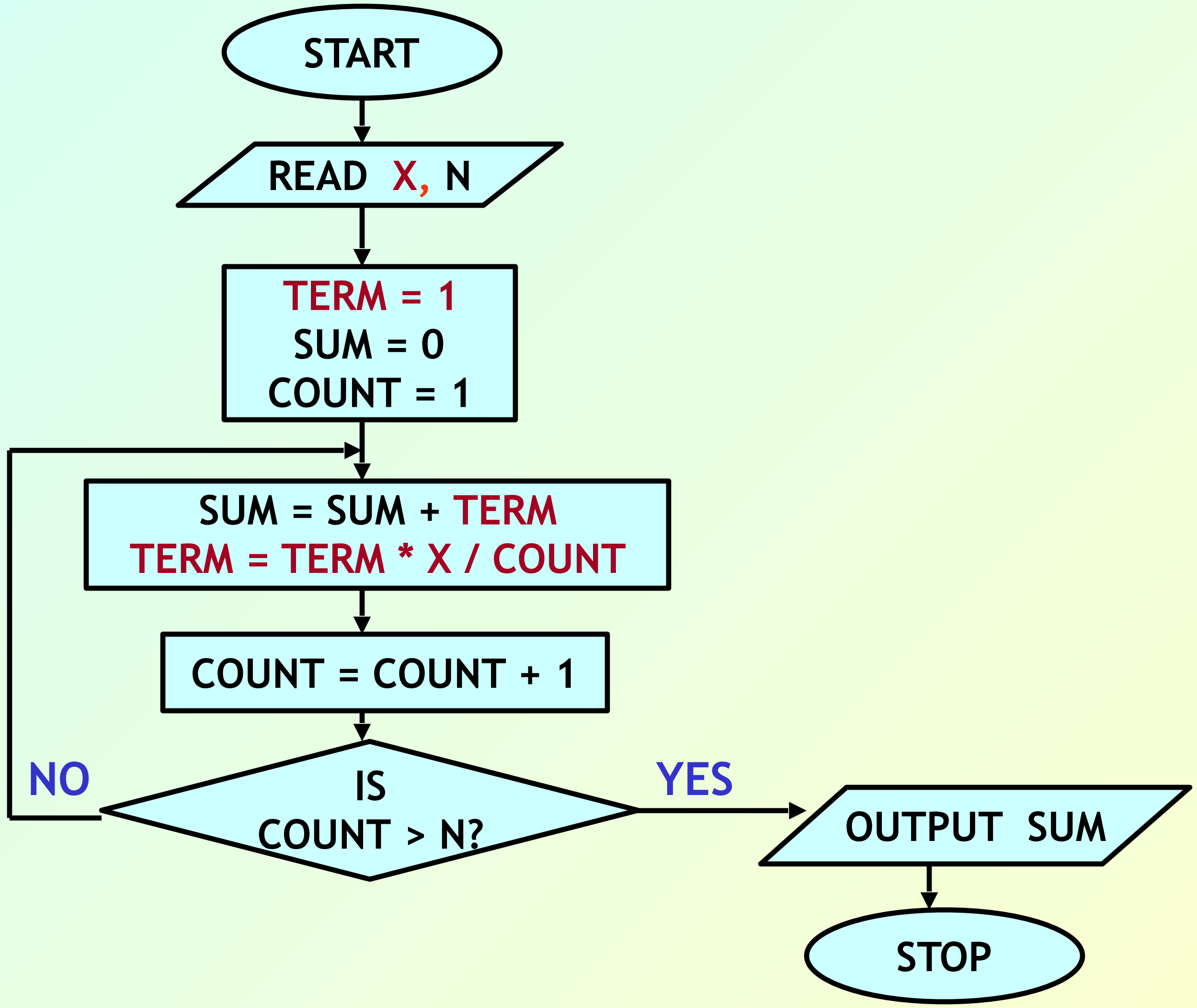
## Example 6: *SUM = 1.2 + 2.3 + 3.4 + to N terms*



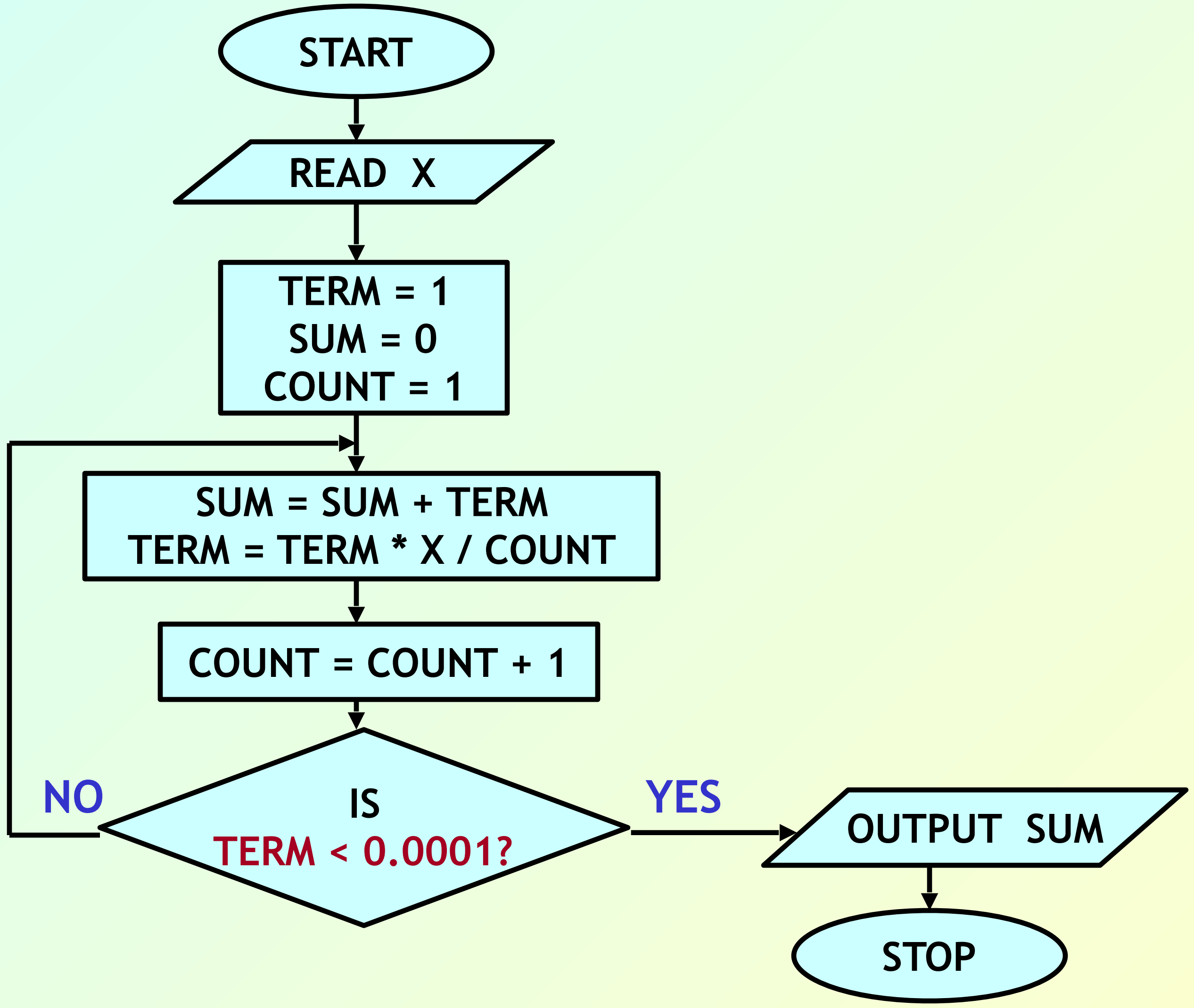
## Example 7: *Computing Factorial*



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



## Example 9: *Computing $e^x$ series up to 4 decimal places*



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

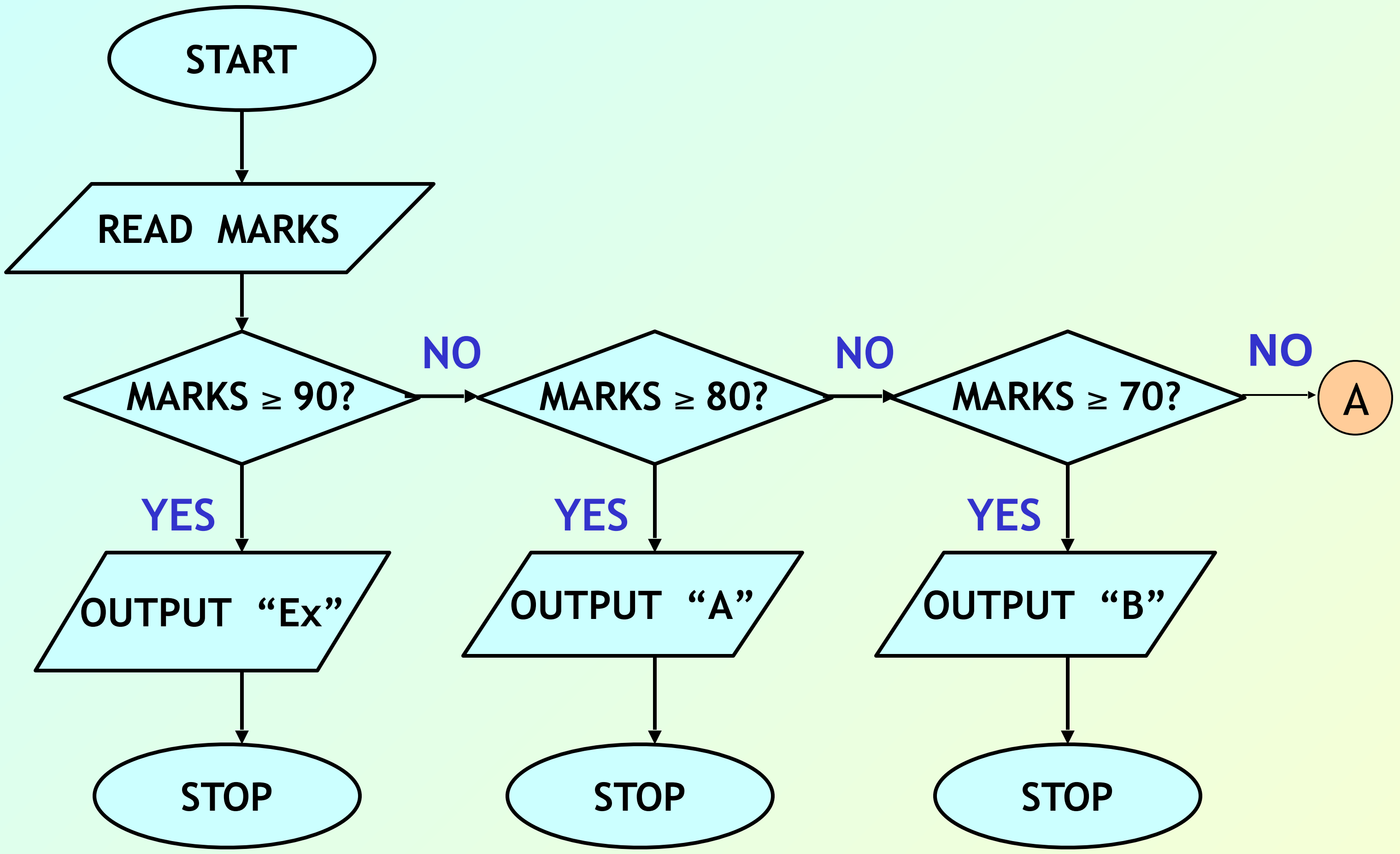
$$\boxed{ax^2 + bx + c = 0}$$

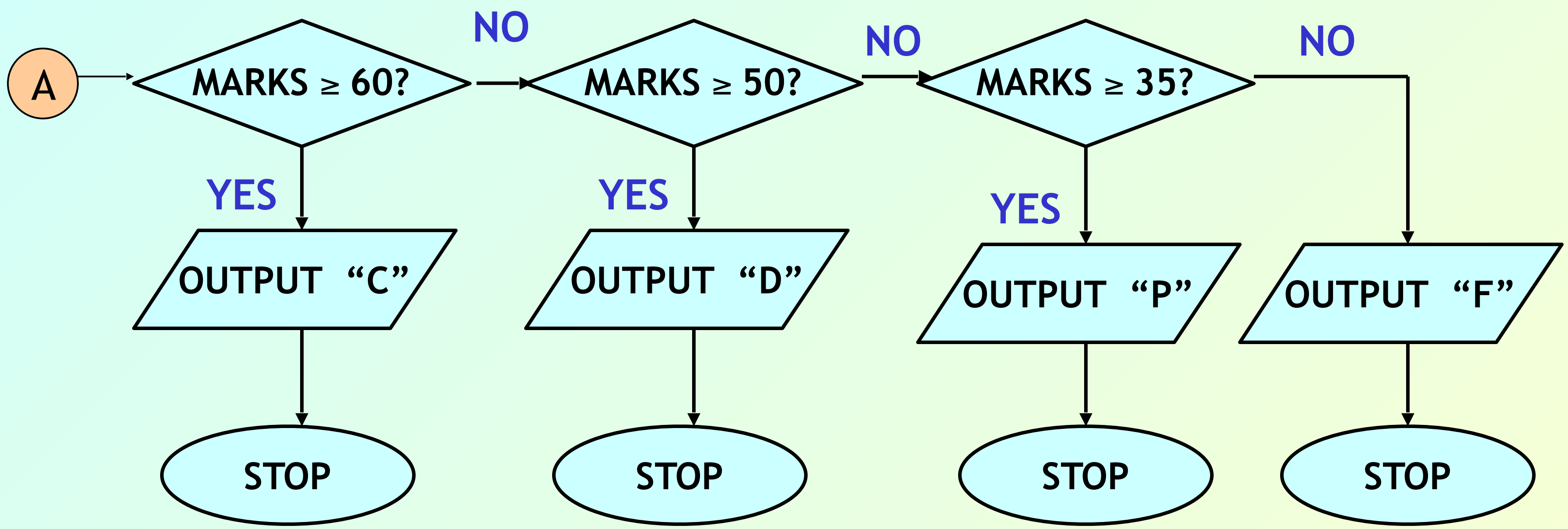
**TRY YOURSELF**

## Example 11: *Grade computation*

| | | |
|---|---|---|
| MARKS ≥ 90 | ➔ | Ex |
| 89 ≥ MARKS ≥ 80 | ➔ | A |
| 79 ≥ MARKS ≥ 70 | ➔ | B |
| 69 ≥ MARKS ≥ 60 | ➔ | C |
| 59 ≥ MARKS ≥ 50 | ➔ | D |
| 49 ≥ MARKS ≥ 35 | ➔ | P |
| 34 ≥ MARKS | ➔ | F |

## *Grade Computation (contd.)*




A
MARKS ≥ 60?
NO
YES
OUTPUT “C”
STOP
MARKS ≥ 50?
NO
YES
OUTPUT “D”
STOP
MARKS ≥ 35?
NO
YES
OUTPUT “P”
STOP
OUTPUT “F”
STOP

## Programming in C

## Sample C program #1

```
#include <stdio.h>
main()
    {
        printf ("\n Our first look at a C program \n");
    }
```

## Sample C program #2

```
/* Compute the sum of two integers */

#include <stdio.h>
main()
    {
         int   a, b, c;
         a = 10;
         b = 20;
         c = a + b;
         printf ("n The sum of %d and %d is %d\n", a,b,c);
    }
```

## Sample C program #3

```
#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);
    }
```

## Sample C program #4

```
#include <stdio.h>
#define     PI    3.1415926

/* Compute the area of a circle */
main()
    {
         float   radius, area;
         float   myfunc (float radius);

         scanf ("%f", &radius);
         area = myfunc (radius);
         printf ("\n Area is %f \n", area);
    }
```

```
float   myfunc (float r)
    {
         float   a;
         a = PI * r * r;
         return (a);      /* return result */
    }
```

## Introduction to C

- C is a general-purpose, structured programming language.
  - Also contains additional features which allow it to be used at a lower level.
- C can be used for applications programming as well as for systems programming.
- There are only 32 keywords and its strength lies in its built-in functions.
- C is highly portable, since it relegated much computer-dependent features to its library functions.

## History of C

- Originally developed in the 1970's by Dennis Ritchie at AT&T Bell Laboratories.
- 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

## 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.

## Contd.

- Each compound statement is enclosed within a pair of braces: '{' and '}'
  - The braces may contain combinations of elementary statements and other compound statements.
- Comments may appear anywhere in a program, enclosed within delimiters '/*' and '*/'.
  - Example:

```
a = b + c;    /* ADD TWO NUMBERS */
```

## Example of a Function

```
/* Compute the sum of two integers */
// You can also give comments like this

#include <stdio.h>
main()
    {
        int   a, b, c;

        a = 10;
        b = 20;
        c = a + b;
        printf ("\n The sum of %d and %d is %d\n",
   a,b,c);
    }
```

## 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.

## Contd.

- Program indentation
  - Use proper indentation.
  - Structure of the program should be immediately visible.

## Indentation Example #1 :: Good Style

```
#include <stdio.h>
#define     PI     3.1415926
/* Compute the area of a circle */

main()
 {
   float  radius, area;
   float  myfunc (float radius);

   scanf ("%f", &radius);
   area = myfunc (radius);
   printf ("\n Area is %f \n", area);
 }
```

```
float myfunc (float r)
 {
   float   a;
   a = PI * r * r;
   return (a);
      /* return result */
 }
```

## Indentation Example #1 :: Bad Style

```
#include <stdio.h>
#define    PI    3.1415926
/* Compute the area of a circle */
main()
{
float   radius, area;
float   myfunc (float radius);
scanf ("%f", &radius);
area = myfunc (radius);
printf ("\n Area is %f \n", area);
}
```

```
float   myfunc (float r)
{
float   a;
a = PI * r * r;
return (a);
   /* return result */
}
```

## Indentation Example #2 :: 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))    /* 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);
    }
```

## Indentation Example #2 :: 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))/* 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);
}
```

## 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 | | |

# 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.
  - 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.

## Contd.

- Keywords
  - Reserved words that have standard, predefined meanings in C.
  - Cannot be used as identifiers.
  - OK within comments.
  - Standard C keywords:

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| **auto** | **break** | **case** | **char** | **const** | **continue** | **default** | **do** |
| **double** | **else** | **enum** | **extern** | **float** | **for** | **goto** | **if** |
| **int** | **long** | **register** | **return** | **short** | **signed** | **sizeof** | **static** |
| **struct** | **switch** | **typedef** | **union** | **unsigned** | **void** | **volatile** | **while** |

## 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
double
for
```

## 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

Typically occupies 8 bytes (64 bits) in memory.

## Contd.

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

```
short int  flag;
long int  result;
unsigned int  count, age;
```

## Some Examples of Data Types

- int

  0,  25, –156,  12345, –99820

- char

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

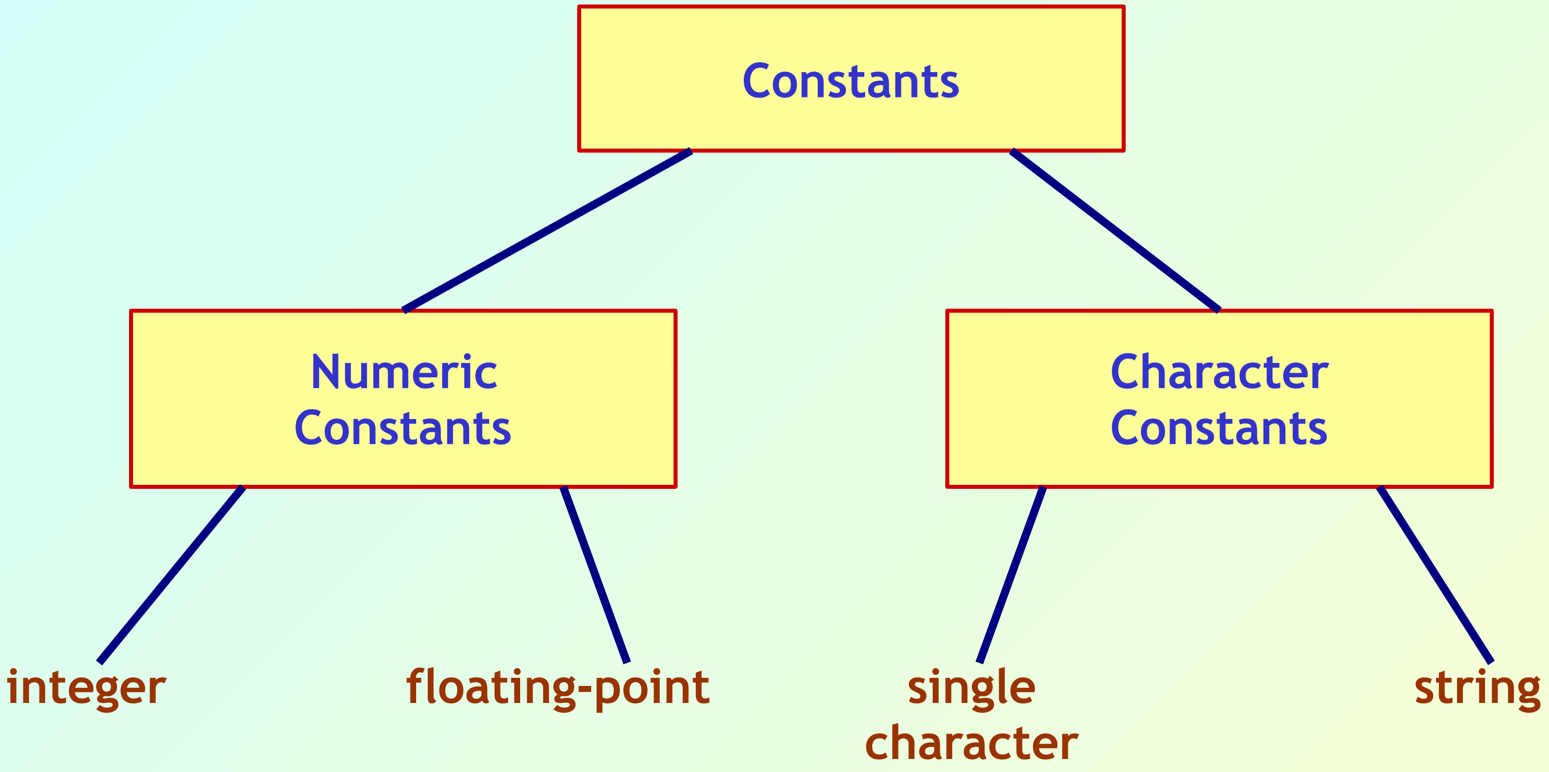
- float

  23.54, –0.00345,  25.0

  2.5E12,  1.234e–5

E or e means "10 to the power of"

## Constants


Constants
Numeric Constants
Character Constants
integer
floating-point
single character
string

## 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 | $(2^{31} - 1)$ |
| Minimum | :: | – 2147483648 | $(-2^{31})$ |

<For 2's complement representation>

## 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.

## Variables

- It is a data name that can be used to store a data value.
- Unlike constants, a variable may take different values in memory during execution.
- Variable names follow the same naming convention for identifiers.

  Examples :: temp, speed, name2, current, my_salary

## Example

```
int     a, b, c;
char    x;

a = 3;
b = 50;
c = a – b;
x = ‘d’;

b = 20;
a = a + 1;
x = ‘G’;
```

## Declaration of Variables

- There are two purposes:
  - It tells the compiler what the variable name is.
  - 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: %f\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.

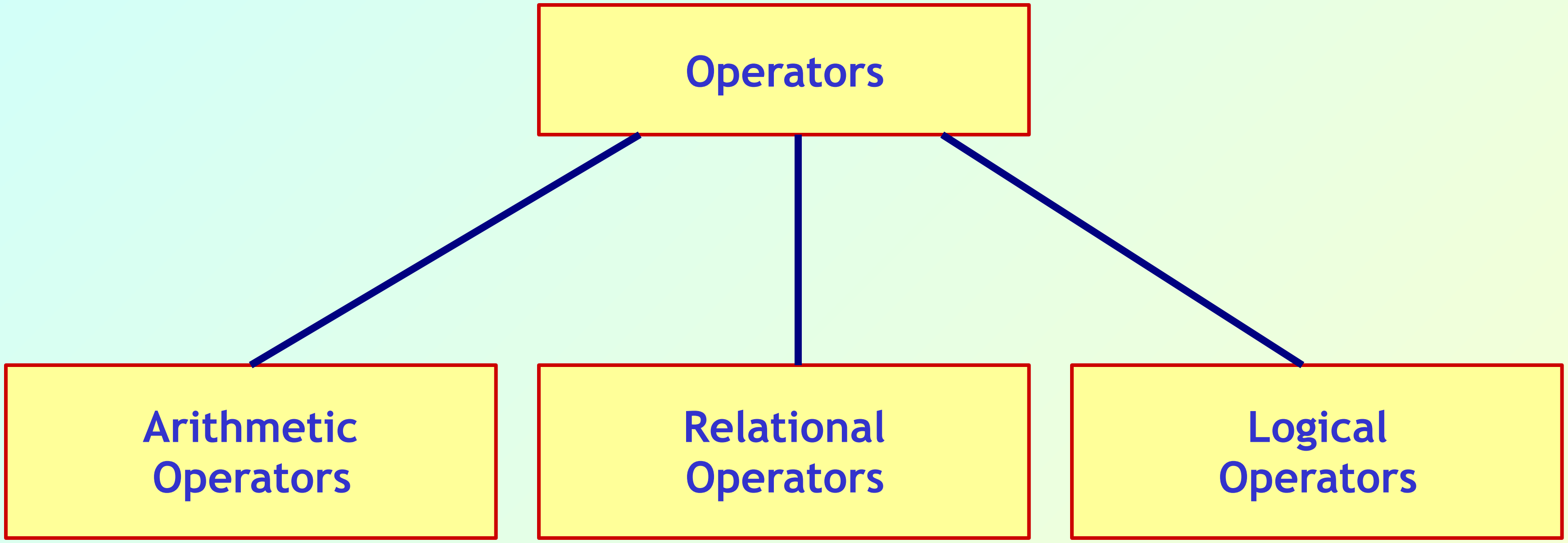
- A value can also be assigned to a variable at the time the variable is declared.

```
 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 | ➔ | a + (b * c) – (d / e) |
| a * – b + d % e – f | ➔ | a * (– b) + (d % e) – f |
| a – b + c + d | ➔ | (((a – b) + c) + d) |
| x * y * z | ➔ | ((x * y) * z) |
| a + b + c * d * e | ➔ | (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.
- Examples:

```
    (12 + 3) / 6            gives the value 2
    (2 / 3) * 3             gives the value 0
    (12 * 3) / 7 + 3 * 2    gives the value 11
```

## 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       gives the value 2
  25 / 10.0    gives the value 2.5
```

- Some more issues will be considered later.

## Type Casting

- Temporarily convert the type of a variable before being used in an expression.
  – Expressed by specifying the *desired* type in parenthesis before the variable/expression.
- Examples:

```
int  a = 10, b = 4, c;   float  x, y;
x = (float) a / b;     /* x will be 2.5 */
y = (float) (a / b);   /* y will be 2.0 */
c = (int) x * 4;       /* c will be 8 */
a = (int) (x * 4);     /* a will be 10 */
```

## 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** |

- The result of comparison is "true" or "false".
  - The value 0 is considered as "false", and any non-zero value as "true".

## 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);


if (1)    /* will be always true */
   …………
```

## 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 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 |

- Examples:

```
if  ((i > 2) && (i < 10))
  printf ("\n i lies between 3 and 9");

if  ((flag == 'A') || (flag == 'a')
  printf ("\n Either lower or uppercase A");
```

## 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.

- 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);
```