Introduction to Programming and Data Structure

Subject Code: CS1L001

Dr Sudipta Saha
Assistant Professor
https://www.iitbbs.ac.in/profile.php/sudipta/

https://sites.google.com/iitbbs.ac.in/dssrg

Computer Science and Engineering,
School of Electrical Sciences
Indian Institute of Technology Bhubaneswar

Overview

Introduction to -

How to make a computer work for you

How to solve problems

How to program

Textbooks:

1. Al Kelley and Ira Pohl. A book on C, 4th Edition, Pearson India, 1998.
2. Brain W. Kernighan & Dennis Ritche, **The C Programming Language**, Prentice Hall of India.

Reference Books:

1. Ellis Horowitz, Satraj Sahni and Susan Anderson-Freed, Fundamentals of Data Structures in C, W. H. Freeman and Company.
2. **Byron Gottfried**, Schaum's Outline of Programming with C, McGraw-Hill.
3. **Let us C – by Kanetkar – BPB publications**

Evaluation

1. End Sem 40
2. Mid Sem 30
3. Internal Assessments 30

Class Test 1	15
Class Test 2	15
Class Test 3	15
Attendance (+..)	15
	100

Average or Best one

Contact hours

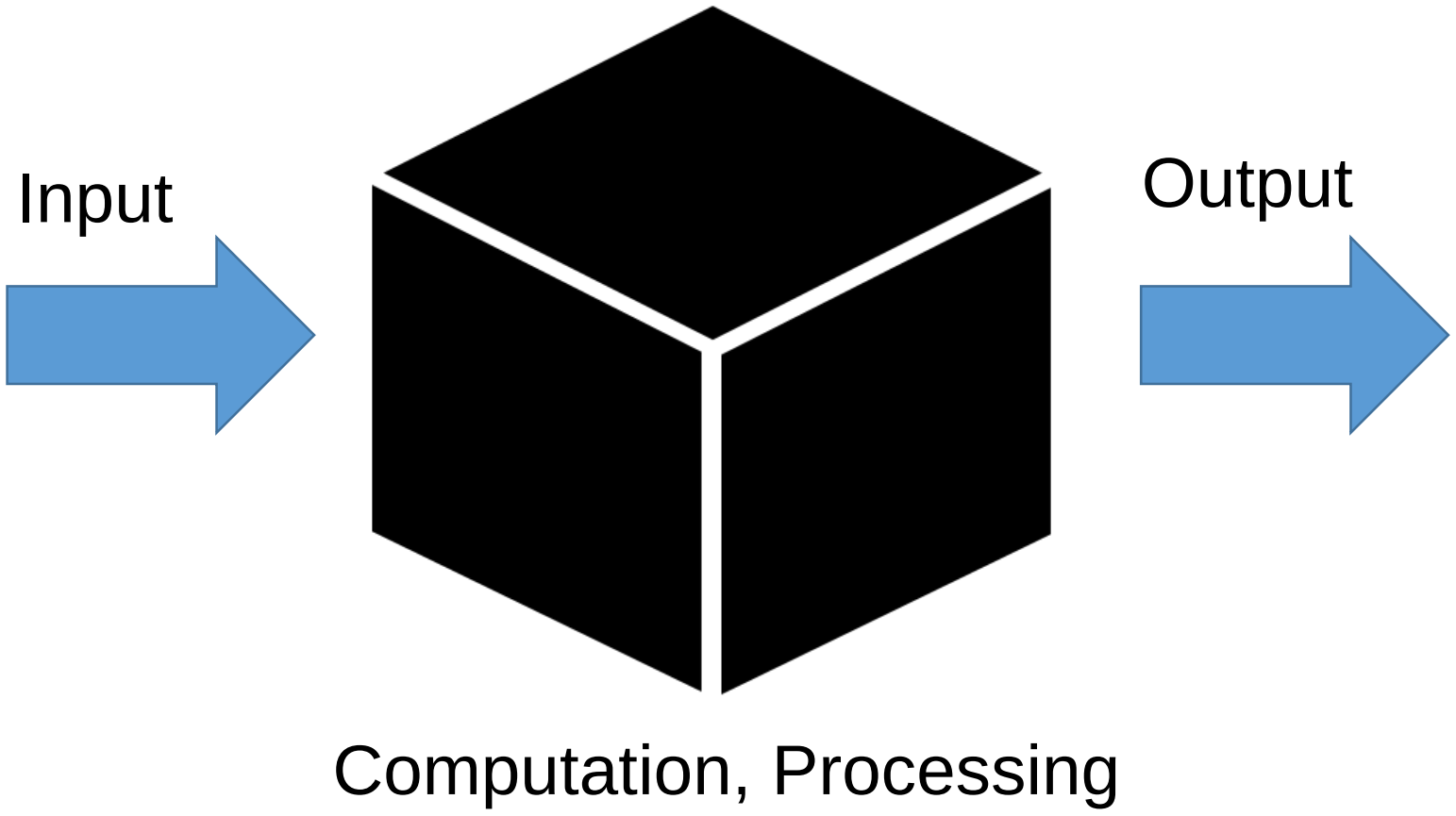
L-T-P: 3-1-0 Credits: 4

Section 1:

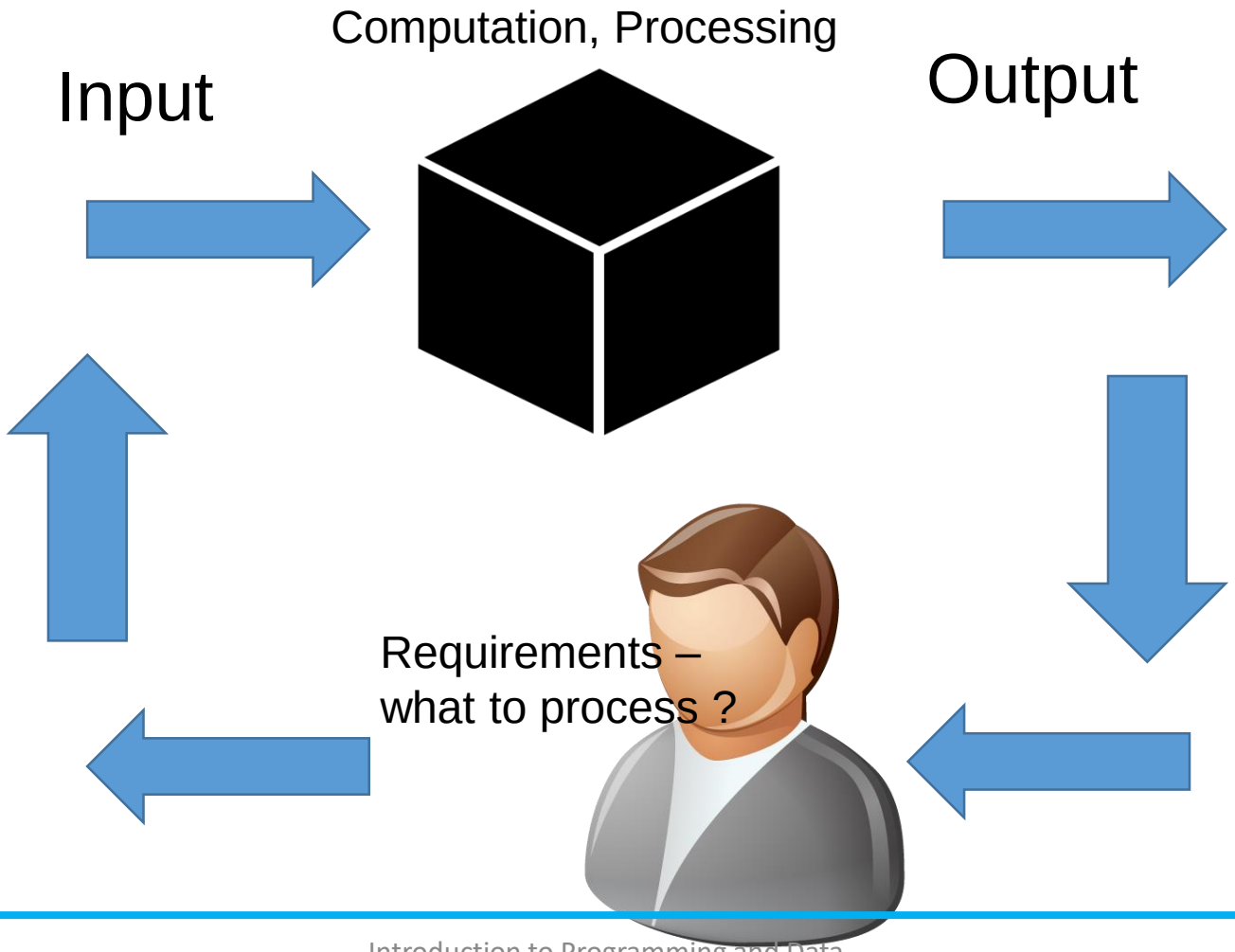
Lecture hours
Monday 10 to 12 : 2 hours
Thursday 8 to 9 : 1hour

Tutorial
Tuesday 10 to 12 : 1 hour for each group

A Generic Computing system – Components

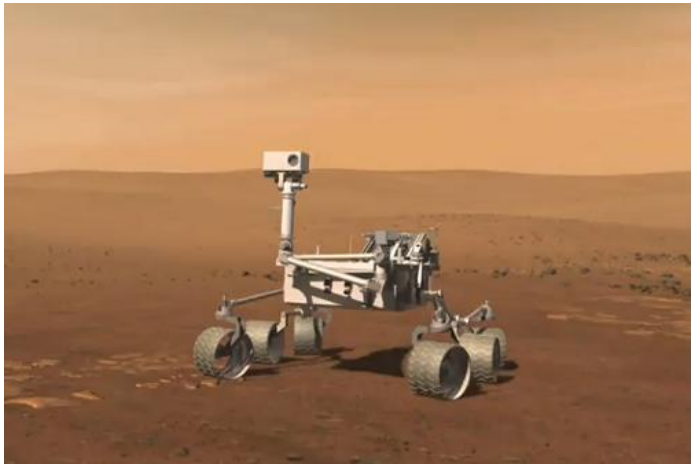


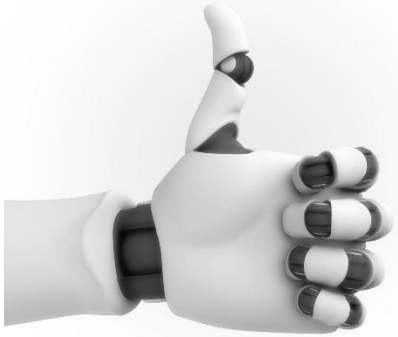
A Generic Computing system - Flow



Examples where it is found ?

Examples where it is found ?





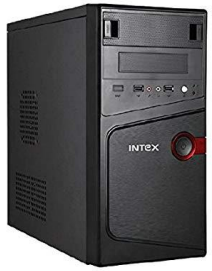
And the list is ever growing …..

A computing system - components

- Input devices – Keyboard, Mouse, webcam, scanner, barcode, Mic
- …..

- Output devices – Monitor, Speaker, Printer,

- A Central unit – Computation / Processing capability - CPU

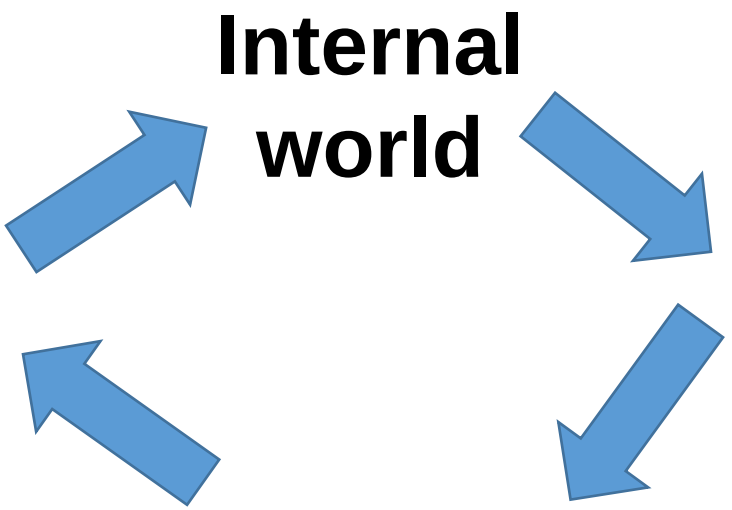


A computing system - components

What about this ?

Human analogy

Cognitive senses
–
Smelling,
Tasting,
Seeing,
Touching,
Hearing



Moving,
Grasping,
Speaking, ….

Then, why we (humans) tried to build computers - ?

Why Computers -

A Slave - that can do works for us

Can carry out the work perfectly

Repeat works as many times as needed

Cannot think or decide itself – a dumb box
(Not fully true now – with ML and AI)

But still there are gaps
(between humans and machines)

So our goal in this course -

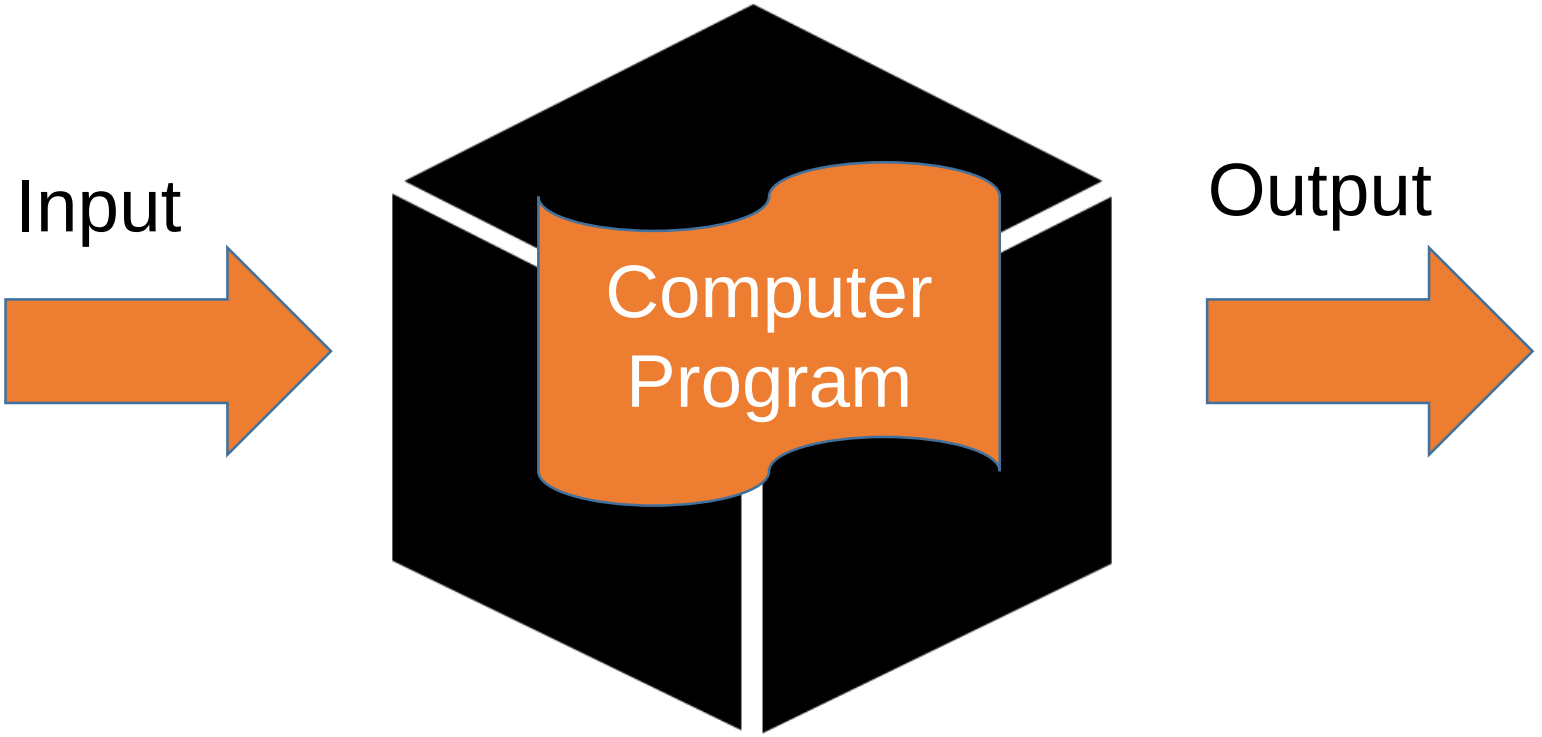
To begin learning – how to make the ‘powerful slave’ work for you –

To ‘program’ the machine (code)

Exploring the black box little more …..

Program ?

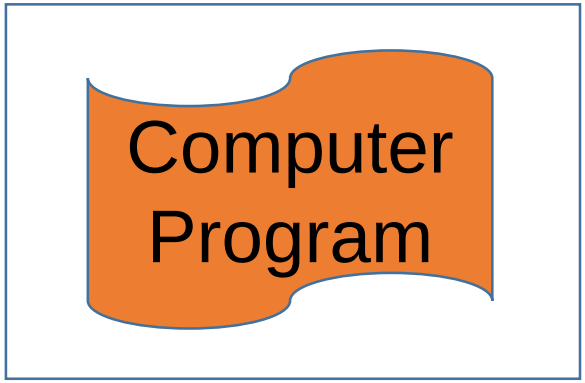
Exploring the black box little more …..



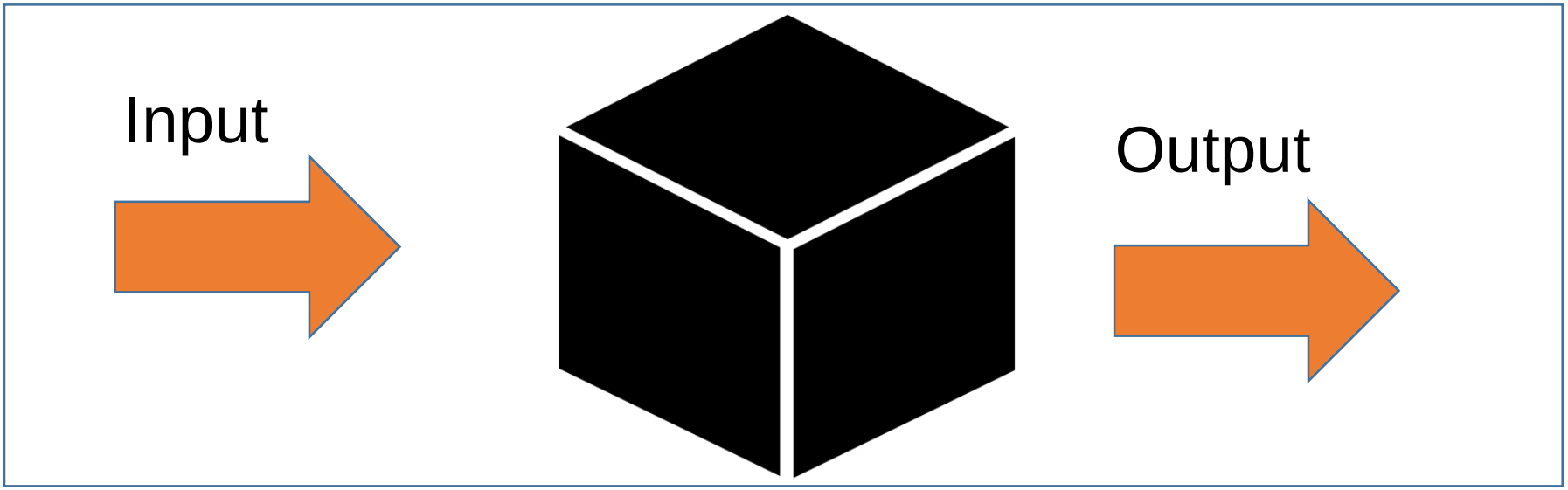
Computation, Processing

Software and Hardware -

Software



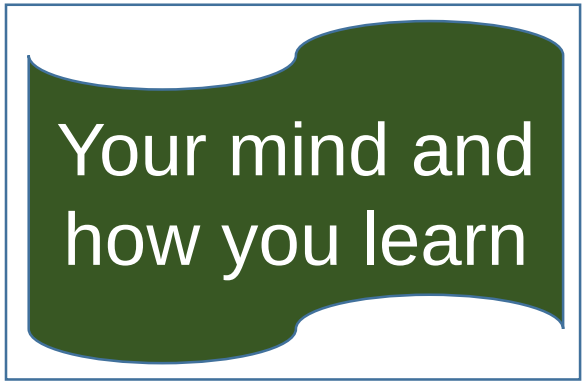
Hardware



Computation, Processing

Human analogy

Software



Hardware

Your Body

What is programming -

Its sort of talking with the computer –

Needs a Language to do so – Like English is a language we use to speak

Computer Languages – are there for the same purpose – **C, C++, Basic, Pascal, Python, Java, C#, …** etc.

Evolution of the languages

Programming languages

- Machine Language
- Assembly language
- Low level Languages
- **Medium Level Languages, C, ….**
- High Level Languages – Python, java, Basic, ….
- Natural Language

How a programming language works

- Computers only understands Machine language
- So all other languages need to be converted to Machine language

- Types of converters –
  - Assembler – for Assembly language
  - Compiler – C compiler, BASIC compiler etc…..
  - Interpreters –
  - Machine independence – Bytecode – over virtual machines – like Java

How a programming language works

Program → Compiler → Machine language program → Execution in the machine (Process)

We need to know the following –

---

a) How to write a program or code/program, edit, save etc.

b) How to compile the code to generate the machine language program

C) How to execute the code

How to write, save, edit code

Help of file systems –

1. Editing tools – **Gedit / Emacs / VI**

2. Open file → Write file → Save file → Open file → edit file → Save file → Rename file → Open file …… Delete file.

3. Directory structure / files …. Navigation through the file system - Various file system related commands (pwd, ls, mv, cd, etc.)

How to write, save, edit code

Compilation

$ gcc hello.c

…… List of errors that are there in the program

…… Or a message saying the compilation went successful….

There will be a file created called - a.out – which is the executable file – it contains machine language code.

How to write, save, edit code

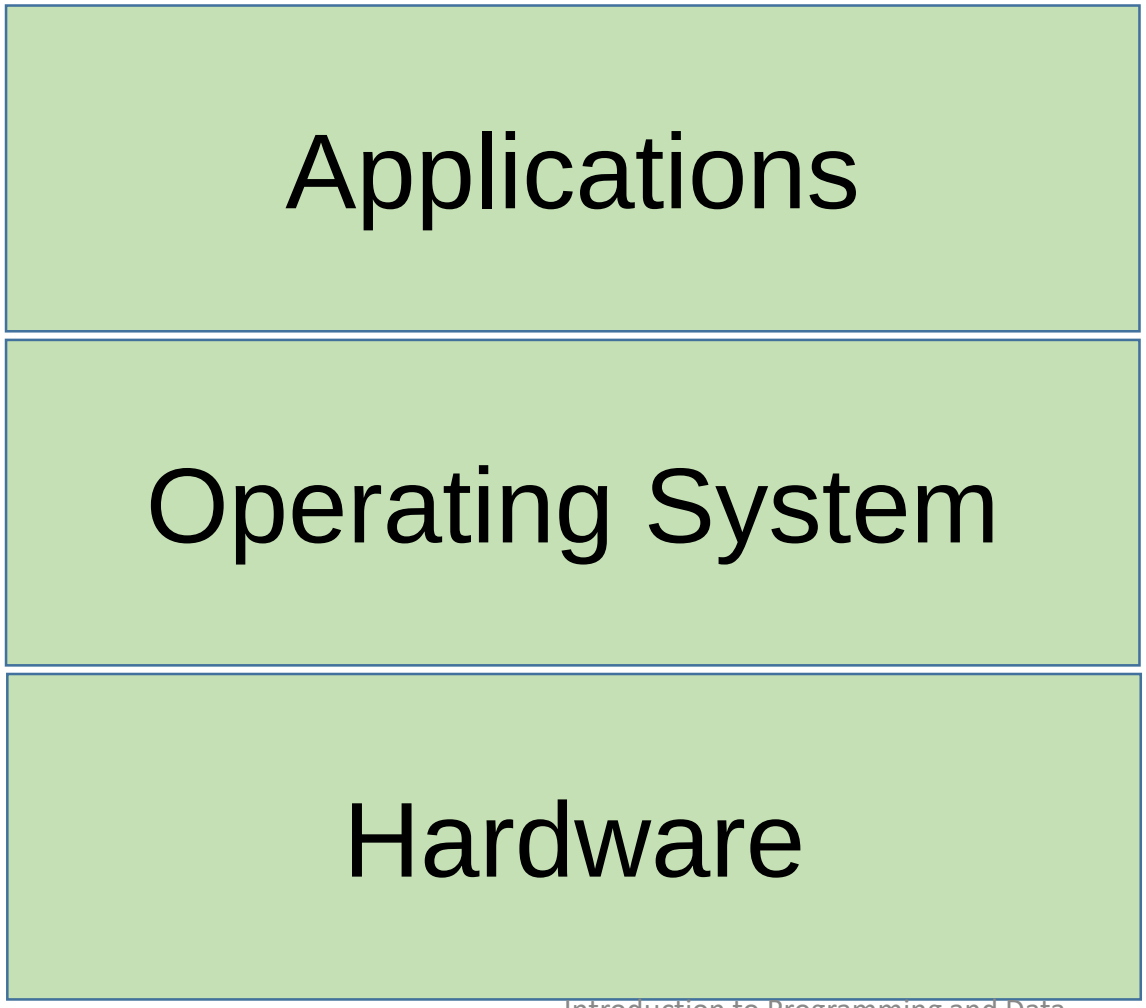
Execution step

$ ./a.out

Hello World !

$.....

Computer system -



Web browser

Win10, Linux, Android …

Examples – CPU, memory, IoT devices etc
Mobile phone

Types and components of programs

Inputs

Outputs

Processing

Types and components of programs

Inputs C statements for inputting contents

scanf, getchar, getch, gets etc.

Outputs printf, putchar, puts etc.

Processing user defined functions, or already defined functions such as **sin, log and many others**

First program – ‘Hello World’

```
1. #include <stdio.h>
2. int main()
3. {
4.     printf("Hello, World!.1\n");
5.     printf("Hello, World!.2 ");
6.     return 0;
6. }
```

preprocessor command

Library stdio

printf and scanf

Return 0 – exist status

Output without \n

Hello, World!.1Hello, World!.2

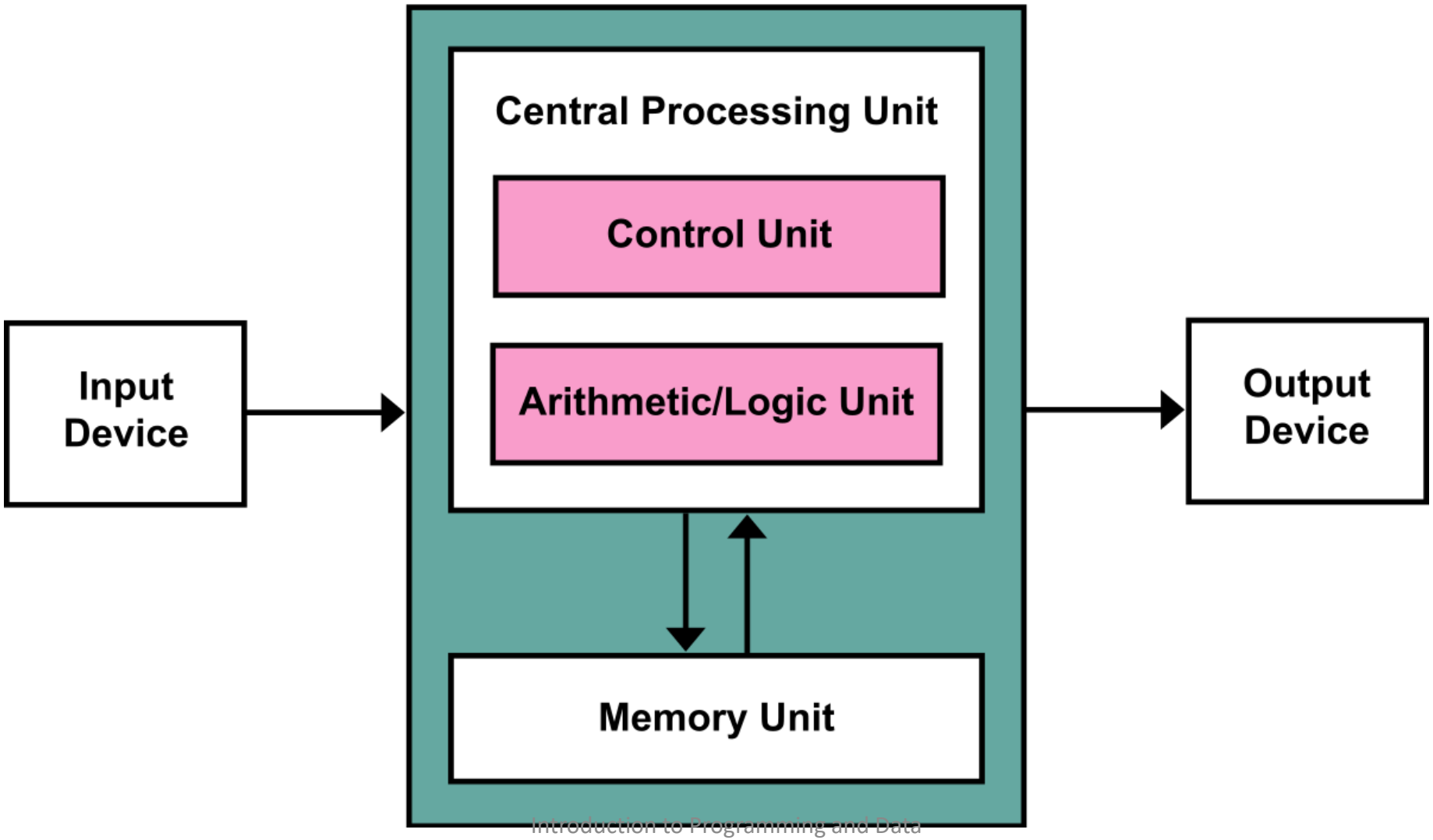
Output with \n

Hello, World!.1
Hello, World!.2

Input and Output

- printf is for output
- Before going to scanf we need understand the concept of variables – hence memory architecture

Von-Neumann Model / Architecture.
Von-Neumann proposed his computer **architecture** design in 1945 which was later known as **Von-Neumann Architecture**



Variables

```
1. #include <stdio.h>
2. int main()
3. {
4.     int testInteger = 5;
5.     printf("Number = %d", testInteger);
6.     return 0;
7. }
```

Variables

```
1. #include <stdio.h>
2. int main()
3. {
4.     float number1 = 13.5;
5.     double number2 = 12.4;
7.     printf("number1 = %f\n", number1);
8.     printf("number2 = %lf", number2);
9.     return 0;
10. }
```

Output

```
number1 = 13.500000
number2 = 12.400000
```

Integer Input/Output

```
1.  #include <stdio.h>
2.  int main()
3.  {
4.      int testInteger;
5.      printf("Enter an integer: \n");
6.      scanf("%d", &testInteger);
7.      printf("Number = %d",testInteger);
8.      return 0;
9.  }
```

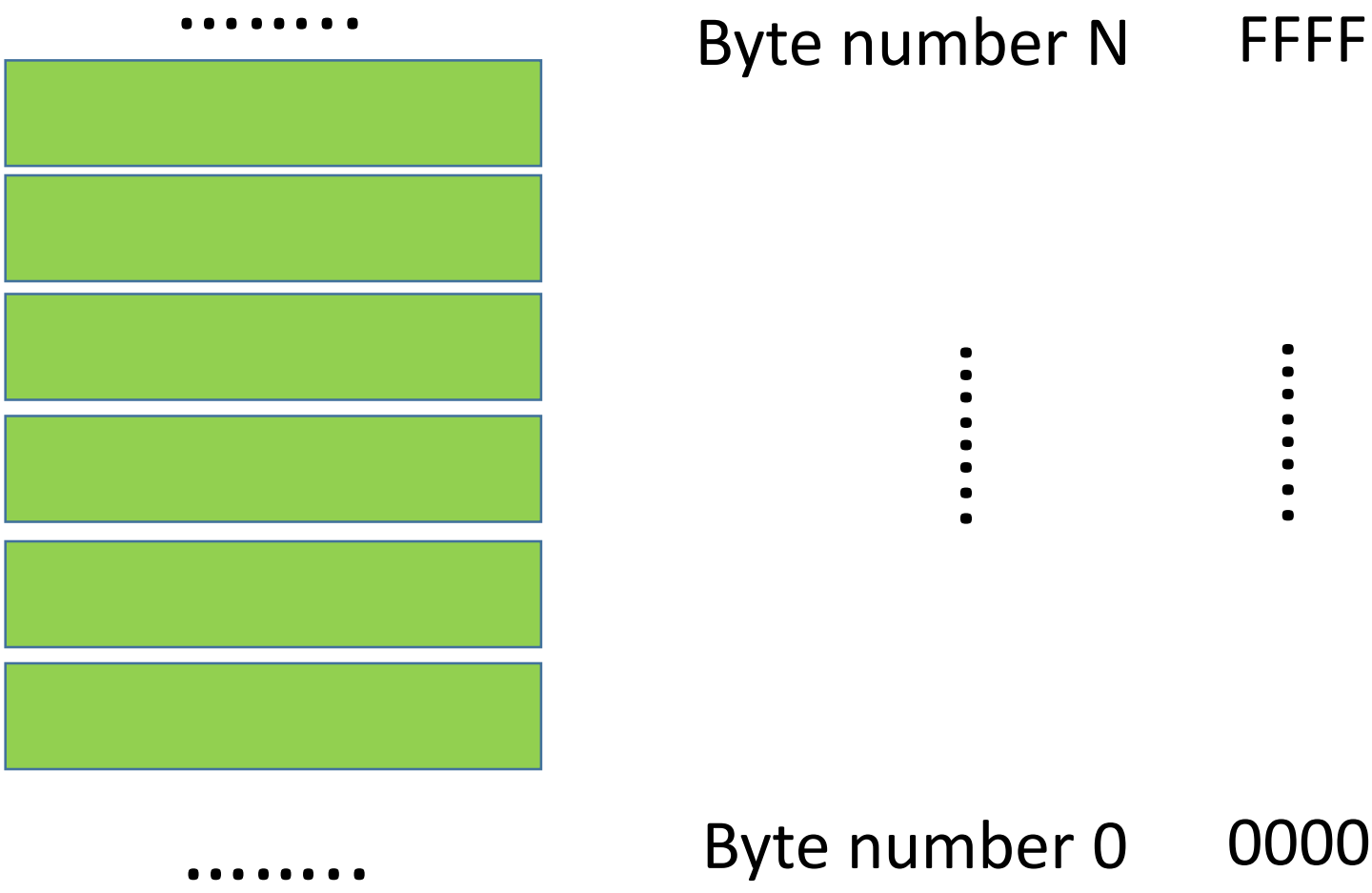
Integer Input/Output -

```
1.  #include <stdio.h>
2.  int main()
3.  {
4.      int testInteger, testl;
5.      printf("Enter two integers: \n");
6.      scanf("%d", &testInteger);
7.      scanf("%d", &testl);
8.      printf("Number = %d,
%d",testInteger, testl);
9.return 0;
10. }
```

Output

```
Enter two integers:
4 10
Number = 4,10
```

Memory model



Variables, Constants and Literals

- A variable is a container (storage area) to hold data.

- C is a strongly typed language.
  - Variable type cannot be changed once it is declared.

```
int number = 5; // integer variable
number = 5.5; // error
double number; // error
```

- Rules for variable name formation

Variables, Constants and Literals

A **Literals** is – whose value cannot be altered

```
Numeric literals –

Decimal: 0, -9, 22 etc.
Octal: 021, 077, 033 etc.
Hexadecimal: 0x7f, 0x2a, 0x521 etc.
```

Floating point literals –
`-2.0, 0.0000234, -0.22E-5`

Character literals – ‘a', 'm', 'F', '2‘

char a = ‘F’;

- char a;
- int I;
- const float PI = 3.14;
- PI..

Variables, Constants and Literals

Escape sequences

\b	Backspace
\f	Form feed
\n	Newline
\r	Return
\t	Horizontal tab
\v	Vertical tab
\\	Backslash
\'	Single quotation mark
\"	Double quotation mark
\?	Question mark
\0	Null character

Constants

A variable whose value cannot be changed

```
const double PI = 3.14;

PI = 2.9; //Error
```

String constants

- "good"

```
//string constant
```

- ""

```
//null string constant
```

- " "

```
//string constant of six white space
```

String constants

- "x"

//string constant having a single character.

- "Earth is round\n"

//prints string with a newline

Data Types

Type	Size (bytes)	Format Specifier
int	at least 2, usually 4	%d
char	1	%c
float	4	%f
Double	8	%lf
short int	2 usually	%hd

Data Types

Type	Size (bytes)	Format Specifier
signed char	1	%c
unsigned char	1	%c
long double	at least 10, usually 12 or 16	%Lf

Data Types

Type	Size (bytes)	Format Specifier
unsigned int	at least 2, usually 4	%u
long int	at least 4, usually 8	%li
long long int	at least 8	%lli
unsigned long int	at least 4	%lu
unsigned long long int	at least 8	%llu
signed char	1	%c
unsigned char	1	%c
long double	at least 10, usually 12 or 16	%Lf

int

Integers are whole numbers
Can have both zero, positive and negative
No decimal values.

0, -5, 10

```
int id;
```

```
int id, age;     // Multiple variables together
```

The size of int is usually 4 bytes (32 bits).
it can take 2^{32} distinct states from
-2147483648 to 2147483647.

Float and double

Hold real numbers

float salary;

double price;

float normalizationFactor = 22.442e2;

float (single precision float) is 4 bytes.
double (double precision float) is 8 bytes.

char

char - character type variables.

char test = 'h';

Type specifier – for size

Type specifier ‘long’.

long a;
long long b;
long double c;

a and b can store integer values.
c can store a floating-point number.

Small integer ([−32,767, +32,767] range -
short d;

Type modifier

signed and unsigned

Alter the data storage of a data type by using them

For example,

unsigned int x; // Can hold: -2^{31} to $2^{31}-1$

int y; // Can hold 0 to $2^{32}-1$

What is a byte

What is a bit

Number system

Unary

Binary

Decimal

Hex

How to determine size of a variable using program ?

Input / Output in C

Character I/O

```
#include <stdio.h>
int main()
{
    char chr;
    printf("Enter a character: ");
    scanf("%c",&chr);
    printf("You entered %c.", chr);
    return 0;
}
```

Input / Output in C

Output after execution:

Enter a character: g

You entered g.

Input / Output in C

```
1.  #include <stdio.h>
2.  int main()
3.  {
4.      char x;
5.      printf("Enter a character: ");
6.      scanf("%c", &x);
7.      // When %c is used, a character is displayed
8.      printf("You entered %c.\n",x);
9.      // When %d is used, ASCII value is displayed
10.     printf("ASCII value is % d.", x);
11.     return 0;
12. }
```

Input / Output in C

Output after execution:

Enter a character: g

You entered g.

ASCII value is 103.

I/O Multiple Values

```
#include <stdio.h>
int main()
{
    int a;
    float b;
    printf("Enter integer and then a float: ");

    // Taking multiple inputs
    scanf("%d%f", &a, &b);
    printf("You entered %d and %f", a, b);
    return 0;
}
```

I/O Multiple Values

Output

Enter integer and then a float: -3 3.4

You entered -3 and 3.400000

Keywords and Identifiers

Keywords:

Predefined, reserved words
Special meanings to the compiler
Part of the syntax
Cannot be used as an identifier

int money;

int is a keyword
money is a variable of type int (integer)

Keywords and Identifiers

Some Keywords in C

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

Keywords and Identifiers

Identifier

Name given to entities for example -
Variables, functions, structures etc.

Identifiers must be unique.

int money;
double accountBalance;

Names must be different from keywords

Keywords and Identifiers

Rules to form the identifiers

Valid identifier
Letters (uppercase and lowercase), Digits, underscores

First letter → Either a letter or an Underscore

Keywords cannot be used as identifiers

No rule on length identifier can be
Problem maybe there – for length higher than 31 characters

C Operators

A symbol that operates on a value or a variable.

+ is an operator to perform addition.

C Arithmetic Operators
C Increment and Decrement Operators
C Assignment Operators
C Relational Operators
C Logical Operators
C Bitwise Operators
Comma Operator
The sizeof operator

Arithmetic Operators

For mathematical operations

Addition, Subtraction, Multiplication, Division etc. on Numerical values (constants and variables).

+	addition or unary plus
-	subtraction or unary minus
*	multiplication
/	division
%	remainder after division (modulo division)

Arithmetic Operators

```
1. #include <stdio.h>
2. int main()
3. {
4.     int a = 9,b = 4, c;
5.     c = a+b;
6.     printf("a+b = %d \n",c);
7.     c = a-b;
8.     printf("a-b = %d \n",c);
9.     c = a*b;
10.    printf("a*b = %d \n",c);
11.    c = a/b;
12.    printf("a/b = %d \n",c);
13.    c = a%b;
14.    printf("Remainder when a divided by b = %d \n",c);
15.    return 0;
16. }
```

Expressions

B = 5;

C = 6;

A = B + C;

B = B – C;

B – C;

Unary, Binary, Ternary…

Arithmetic Operators

Output:

a+b = 13
a-b = 5
a*b = 36
a/b = 2
Remainder when a divided by b=1

Note:
9/4 = 2.25. However, the output is 2 in the program.
Remainder works only on integer

Arithmetic Operators

a = 5.0,

b = 2.0,

c = 5

d = 2

In output ----

a/b = 2.5

a/d = 2.5

c/b = 2.5

Possible only either one of the operands is a float

Conversion from Centigrade to Fahrenheit

- int C = ….;
- float F = 0;
- You know this formula - C/5 = (F-32)/9;
- How to use this formula for conversion of C to F or F to C.
- F = f(C) OR F = 9C/5 + 32;
- Convert this formula to an expression – containing variables, literals, constants, and operators.
- Expression: F = (9 *(C/5)) + 32;

Conversion from Centigrade to Fahrenheit

- Expression: F = (9 *(C/5)) + 32;
- C is integer and F is float.
- C = 5, what is F ? F = 9 + 32 = 41 – Correct ….
- Print F → 41.0
- C = Something less than 5 – what will happen ??
- C = 1, C = 2 , C = 3, C = 4 – all are 32. Incorrect ….

Conversion from Centigrade to Fahrenheit

- Expression: F = (9 *(C/5)) + 32;
- Imagine, Only C is Float – will it solve ?
- Yes. Because – C/5 will be float 9 * float – will also be float and 32 + float will be also a float.

Conversion from Centigrade to Fahrenheit

- Expression: F = (9 *(C/5)) + 32;
- Imagine, Only 5 is Float – will it solve ? 5.0
- Yes. Because – C/5 will be float 9 * float – will also be float and 32 + float will be also a float.

Conversion from Centigrade to Fahrenheit

- Expression: F = (9 *(C/5)) + 32;

- Imagine, Only 9 is Float – will it solve ? 9.0 and considering the following two expressions –

- 1. Expression: F = (9.0 * C)/5 + 32;
- 2. Expression: F = 9.0*C/5 + 32;

gcc – Wall a.c
man gcc

- **Warning**
- Error
- **Compilation error (gcc)**
- Run time error

```
float f;
int i
….
f = i;                  // Warning
f = (float)i;           // No warning
….
i = f;                  // Warning
i = (int)f;             // No warning
```

Increment and Decrement Operators

C programming has two operators to change the value of an operand (constant or variable) by 1.

++ → increases the value by 1; **a = a+1, a++**

-- → decreases the value by 1; **a = a-1, a--**

Unary operators : only one operand

Examples on increment and decrement operators

```
1.    #include <stdio.h>
2.    int main()
3.    {
4.        int a = 10, b = 100;
5.        float c = 10.5, d = 100.5;
6.        printf("++a = %d \n", ++a);
7.        printf("--b = %d \n", --b);
8.        printf("++c = %f \n", ++c);
9.        printf("--d = %f \n", --d);
10.       return 0;
11.   }
```

Examples on increment and decrement operators

Output

++a = 11

--b = 99

++c = 11.500000

--d = 99.500000

Operators ++ and -- are used as prefixes

Can also be used as postfixes: a++ and a--

- Postfix

a++

a--

- Prefix

++a

--a

More on increment and decrement

- Pre-increment:
  - ++a increments the value and immediately returns it.

- Post-increment
  - a++ also increments the value (in the background) but returns unchanged value of the variable – (it is executed later).

Avoid confusion

- #include <stdio.h>
  int main()
  {
  int i=5;
  printf(“%d\n”,++i);
  return 0;
  }
- **Output**: 6

Avoid confusion

- #include <stdio.h>
  int main()
  {
  int i=5;
  printf(“%d , %d\n”,++i ,++i);
  return 0;
  }
- **Output:** 7 , 7

Avoid confusion

- #include <stdio.h>
  int main()
  {
  int i=5;
  printf(“%d\n”,i++);
  return 0;
  }
- **Output**: 5

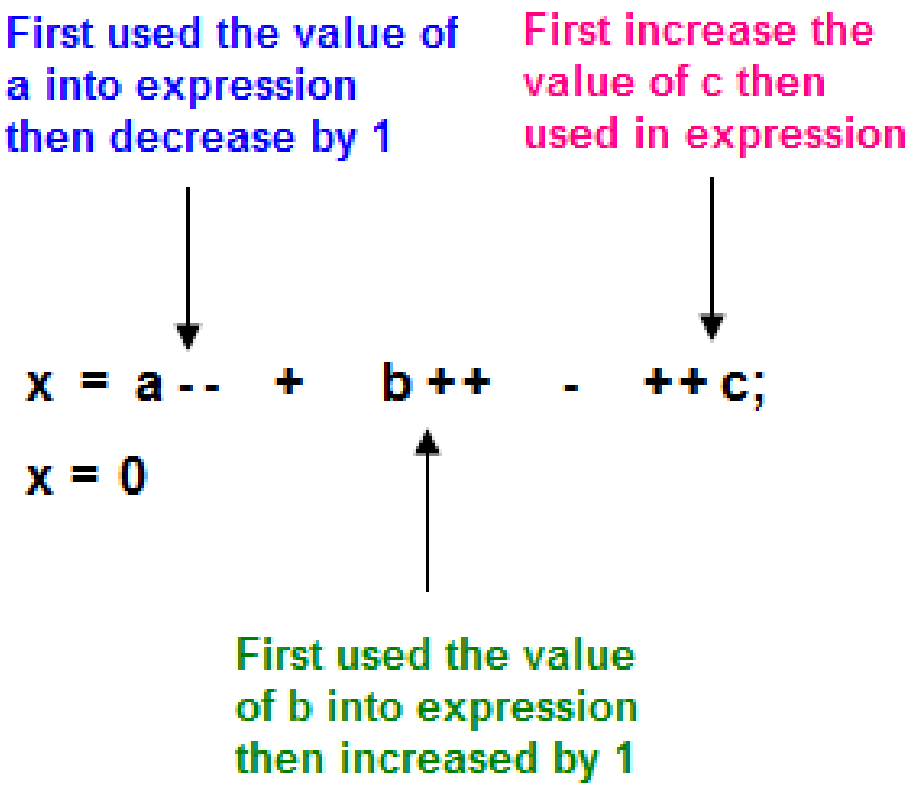
Avoid confusion

- #include <stdio.h>
  int main()
  {
  int i=5;
  printf("%d , %d\n",i++ , i++);
  return 0;
  }
- **Output:** 6 , 5
- The execution of Unary Operator (Post-increment or Pre-increments) happens form Right to left.

Avoid confusion

```
#include<stdio.h>
 void main()
{
        int x,a,b,c;
        a = 2;
        b = 4;
        c = 5;
        x = a-- + b++ - ++c;
        printf("x: %d",x);
}
```

Avoid confusion



Assignment operators

Assigning a value to a variable

Transfer the content of the RHS to the LHS

Operator	***Example***	***Meaning***
=	a = b	a = b
+=	a += b	a = a+b
-=	a -= b	a = a-b
*=	a *= b	a = a*b
/=	a /= b	a = a/b
%=	a %= b	a = a%b

Assignment operations

- In any assignment operation
  - The left hand side must be a variable – LHS
  - The right hand side can be a general expression containing variables, literals, constants, operators – RHS
  - So LHS is restricted

Assignment operators

```
1.  #include <stdio.h>
2.  int main()
3.  {
4.      int a = 5, c;
5.      c = a;                          // c is 5
6.      printf("c = %d\n", c);
7.      c += a;                         // c is 10
8.      printf("c = %d\n", c);
9.      c -= a;                         // c is 5
10.     printf("c = %d\n", c);
11.     c *= a;                         // c is 25
12.     printf("c = %d\n", c);
13.     c /= a;                         // c is 5
14.     printf("c = %d\n", c);
15.     c %= a;                         // c = 0
16.     printf("c = %d\n", c);
17.     return 0;
18. }
```

Assignment operators

Output

1. c = 5
2. c = 10
3. c = 5
4. c = 25
5. c = 5
6. c = 0

Relational Operators

Checks the relationship between two operands

If true, it returns 1;

If false, it returns value 0.

Relational operators are used in decision making and loops.

Operator	Meaning of Operator	Example
==	Equal to	5 == 3 is evaluated to 0
>	Greater than	5 > 3 is evaluated to 1
<	Less than	5 < 3 is evaluated to 0
!=	Not equal to	5 != 3 is evaluated to 1
>=	Greater than or equal to	5 >= 3 is evaluated to 1
<=	Less than or equal to	5 <= 3 is evaluated to 0

Relational operators

- 5 + 3 – result is 8
- 5 > 3 – results of this is what ? Yes or no, 1, 0
- X=1; X >= 8 – what will be the result ?? 0
- X=10; X >= 8 – what will be the result ?? 1
- X = 10; X = X>=8; Print X – what will be the result ?

Relational Operators

```
1. #include <stdio.h>
2. int main()
3. {
4.     int a = 5, b = 5, c = 10;
5.     printf("%d == %d is %d \n", a, b, (a == b)==c );
6.     printf("%d == %d is %d \n", a, c, a == c);
7.     printf("%d > %d is %d \n", a, b, a > b);
8.     printf("%d > %d is %d \n", a, c, a > c);
9.     printf("%d <= %d is %d \n", a, b, a <= b);
10.    printf("%d < %d is %d \n", a, c, a < c);
11.    return 0;
12.}
```

Relational Operators

1. **5 == 5 is 1**
2. 5 == 10 is 0
3. 5 > 5 is 0
4. 5 > 10 is 0
5. 5 <= 5 is 1
6. 5 < 10 is 1

- A = B;
- A and B will contain the same value

Relational Operators

```
1. #include <stdio.h>
2. int main()
3. {
4.     float a = 5;
5.     int b = 5, c = 10;
6.     printf("%d == %d is %d \n", a, b, a == b);
7.     printf("%d == %d is %d \n", a, c, a == c);
8.     printf("%d > %d is %d \n", a, b, a > b);
9.     printf("%d > %d is %d \n", a, c, a > c);
10.    printf("%d <= %d is %d \n", a, b, a <= b);
11.    printf("%d < %d is %d \n", a, c, a < c);
12.    return 0;
13.}
```

Relational Operators

```
1. #include <stdio.h>
2. int main()
3. {
4.     int a = 5, b = 5, c = 10;
5.     printf("%d != %d is %d \n", a, b, a != b);
6.     printf("%d != %d is %d \n", a, c, a != c);
7.     printf("%d >= %d is %d \n", a, b, a >= b);
8.     printf("%d >= %d is %d \n", a, c, a >= c);
9.     printf("%d <= %d is %d \n", a, b, a <= b);
10.    printf("%d <= %d is %d \n", a, c, a <= c);
11.    return 0;
12.}
```

Relational Operators

1. 5 != 5 is 0
2. 5 != 10 is 1
3. 5 >= 5 is 1
4. 5 >= 10 is 0
5. 5 <= 5 is 1
6. 5 <= 10 is 1

Discussion on Midsem

Week 1

1. Operators, precedence rules, associativity rules
2. Formula evaluation
3. If – Else statements

Week 2

1. Switch case statements
2. For loop / While loop

Week 3

1. Break and continue
2. Array

Concepts behind the logical operators

- AND logic
- OR logic
- NOT logic

AND

- AND logic

- 0 AND 0 → 0
- 1 AND 0 → 0
- 0 AND 1 → 0
- 1 AND 1 → 1

- AND can be thought of as a function also –
- AND (X1, X2) = outcome – either 0 or 1

AND

- && represents the AND operation

- (a>b && c>d) value of this will be 0 if any one of the sub-conditions is false.
- If both are true then the whole condition is evaluated to be true

OR

- OR logic

- 0 OR 0 → 0
- 0 OR 1 → 1
- 1 OR 0 → 1
- 1 OR 1 → 1

OR

- || represents the OR operation

- (a>b || c>d) value of this will be 1 if any one of the sub-conditions is true.
- If both are false then the whole condition is evaluated to be false

NOT

- NOT logic

- 0 to NOT→ 1
- 1 to NOT → 0

Boolean values

- 0 is False
- 1 is True

Logical operator

An expression is evaluated to be true or false – solves – how to concatenate multiple relations.

Decision making

Operator	***Meaning***	***Example***
&&	Logical AND	**True only if all operands are true** Say, c = 5 and d = 2 ((c==5) && (d>5)) is 0
||	Logical OR	**True only if either one operand is true** Say, If c = 5 and d = 2 ((c==5) || (d>5)) is 1
!	Logical NOT	**True only if the operand is 0** Say, c = 5 !(c==5) equals to 0.

Logical operator

```
1.  #include <stdio.h>
2.  int main()
3.  {
4.      int a = 5, b = 5, c = 10, result;
5.      result = (a == b) && (c > b);
6.      printf("(a == b) && (c > b) is %d \n", result);
7.      result = (a == b) && (c < b);
8.      printf("(a == b) && (c < b) is %d \n", result);
9.      result = !(a != b);
10.     printf("!(a == b) is %d \n", result);
11.     result = !(a == b);
12.     printf("!(a == b) is %d \n", result);
13.     return 0;
14. }
```

Logical operator

Outputs:

(a == b) && (c > b) is 1

(a == b) && (c < b) is 0

!(a != b) is 1

!(a == b) is 0

Logical operator

Explanations:

(a == b) && (c > b) is 1
since (a == b) and (c > b) is 1 (true).

(a == b) && (c < b) is 0
since (c < b) is 0 (false).

Logical operator

Explanations:

!(a != b) is 1

since (a != b) is 0 (false). So, !(a != b) is 1 (true).

!(a == b) is 0 since

(a == b) is 1 (true). Hence, !(a == b) is 0 (false).

Non zero is True and Zero is false

- 5 && 6 -
- -5 && 6 - 1 && 1 = 1
- 0 && 7 -
- -1|| 2
- ‘c’ && ‘a’ - Check – ‘\0’ && 1
- 1.23 && 3.45
- 0 || !(‘c’) - 0 || !(1) = 0 || 0 = 0
- 0 || !(‘0’)

Sizeof operator

Unary operator

Returns the size of data

(constants, variables, array, structure, etc).

Sizeof operator

```
1.    #include <stdio.h>
2.    int main()
3.    {
4.        int a;
5.        float b;
6.        double c;
7.        char d;
8.        printf("Size of int=%lu bytes\n",sizeof(a));
9.        printf("Size of float=%lu bytes\n",sizeof(b));
10.       printf("Size of double=%lu bytes\n",sizeof(c));
11.       printf("Size of char=%lu byte\n",sizeof(d));
12.       return 0;
13.   }
```

Sizeof operator

Sample Output

Size of int = 4 bytes

Size of float = 4 bytes

Size of double = 8 bytes

Size of char = 1 byte

Comma operator

Comma operators are used to link related expressions together.

[Read in details from Book – what are the applications]

For example:

int a, c = 5, d;

Other operators

Bitwise operator

Ternary operator ?:,

reference operator &,

dereference operator *

member selection operator ->

Bitwise operator

Bit-level operation – for fast computation and power saving

Operators	Meaning of operators
&	Bitwise AND
|	Bitwise OR
^	Bitwise exclusive OR
~	Bitwise complement
<<	Shift left
>>	Shift right

Confusion ??

A = 5 + 7/3 + 1 – 9/4*5;

Which one will be executed first ?

In which order the same type of operators will act ?

Confusion ??

A = 5 + 7/3 + 1 – 9/4*5;

Which one will be executed first ?

Precedence rule

In which order the same type of operators will act ?

Associativity

Operators	Associtivity
`()  []  ->  .`	left to right
`!  ~  ++  ==  +  -  *  &  (type)  sizeof`	right to left
`*  /  %`	left to right
`+  -`	left to right
`<<  >>`	left to right
`<  <=  >  >=`	left to right
`==  !=`	left to right
`&`	left to right
`^`	left to right
`|`	left to right
`&&`	left to right
`||`	left to right
`?:`	left to right
`=  +=  -=  *=  /=  %=  &=  ^=  |=  <<=  >>=`	right to left
`,`	left to right

Examples -

Example 1:
Swap two Variables Using Temporary Variable

```
1.    #include <stdio.h>
2.    int main()
3.    {
4.        double firstNumber, secondNumber, temporaryVariable;
5.        printf("Enter first number: ");
6.        scanf("%lf", &firstNumber);
7.        printf("Enter second number: ");
8.        scanf("%lf",&secondNumber);
9.        temporaryVariable = firstNumber;
10.       firstNumber = secondNumber;
11.       secondNumber = temporaryVariable;
12.       printf("\nAfter swapping, firstNumber = %.2lf\n", firstNumber);
13.       printf("After swapping, secondNumber = %.2lf", secondNumber);
14.       return 0;
15.   }
```

Examples -

Output

Enter first number: 1.20
Enter second number: 2.45

After swapping, firstNumber = 2.45
After swapping, secondNumber = 1.20

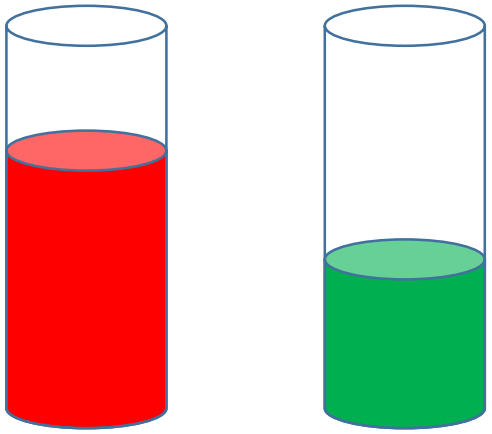
Note:

Step 1: **temporaryVariable** is assigned the value of firstNumber.
Step 2: value of **firstNumber** is assigned to **secondNumber.**
Step 3: **temporaryVariable** (which holds the initial value of **firstNumber**) is assigned to **secondNumber**)

Examples -

Analogy:

How to exchange the content of two glasses having liquids of two different color ?

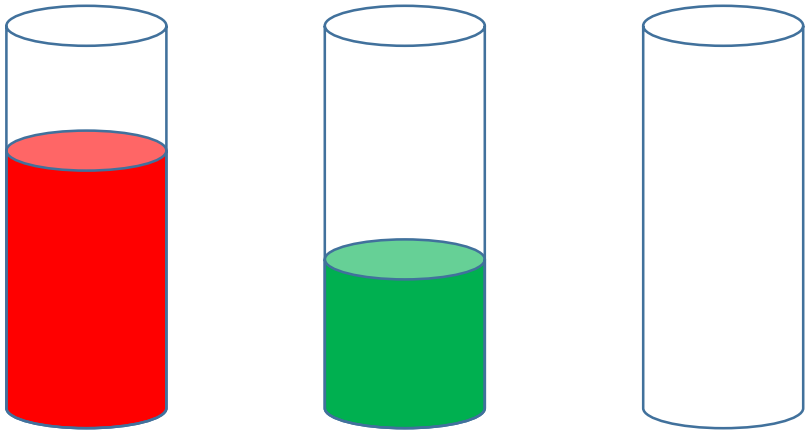


Glass 1 Glass 2

Examples -

Analogy:

Use a third glass

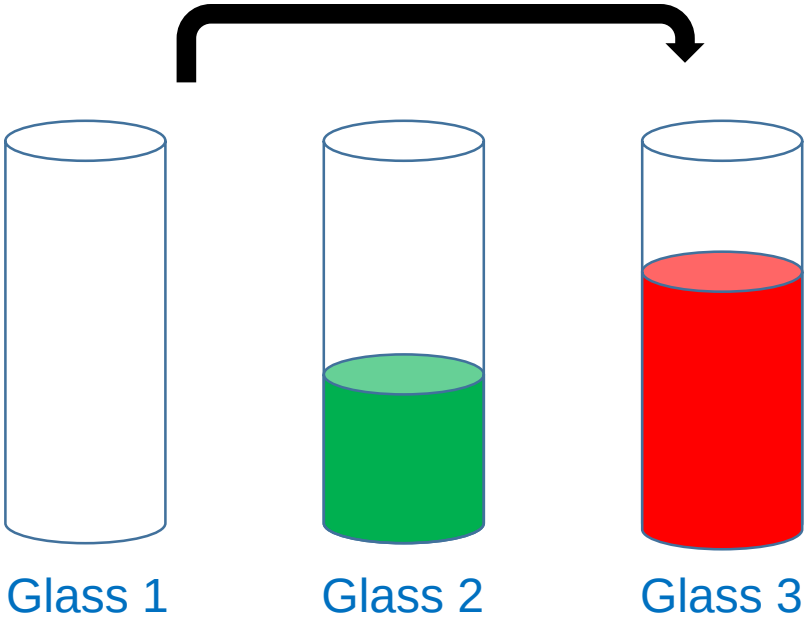


Glass 1 Glass 2 Glass 3

Examples -

Step 1

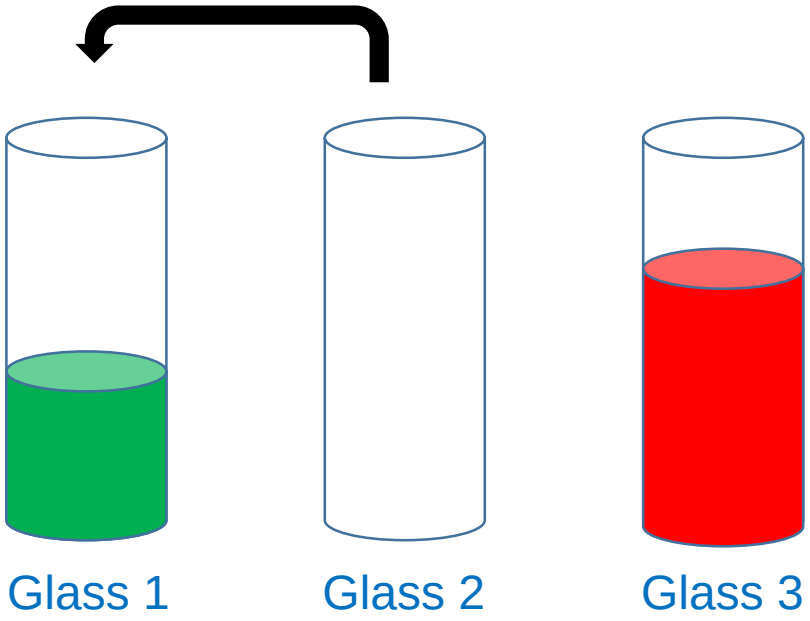
Transfer content from glass 1 to glass 3



Examples -

Step 2

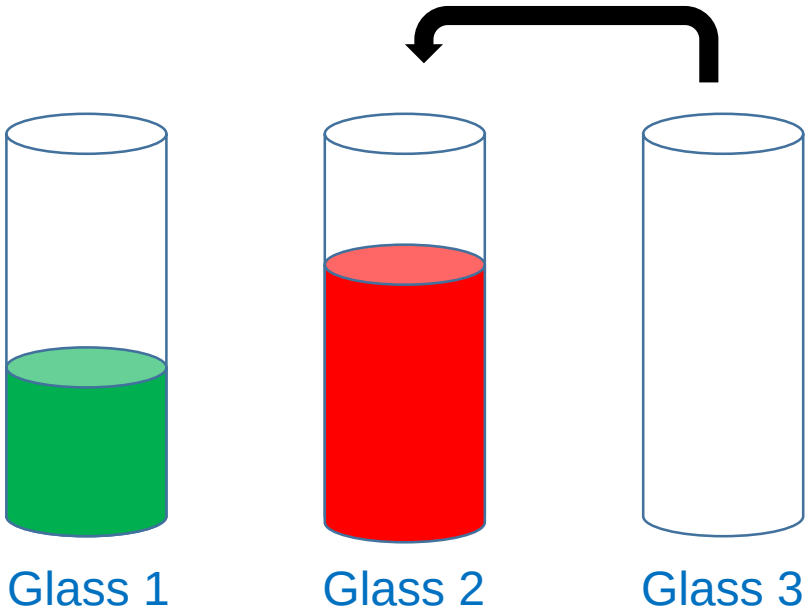
Transfer content from glass 2 to glass 1



Examples -

Step 3

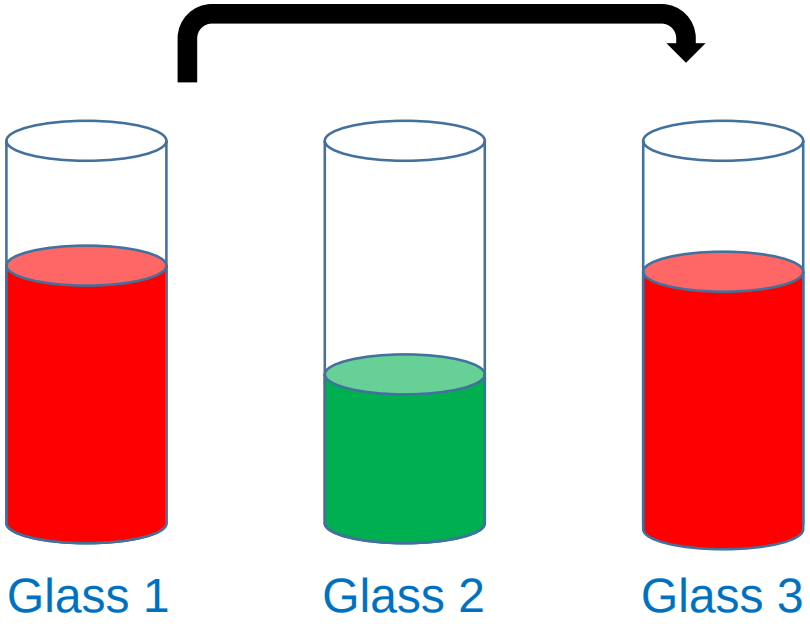
Transfer content from glass 3 to glass 2



Examples -

Step 1 *(Actual scenario if a glass is a variable)*

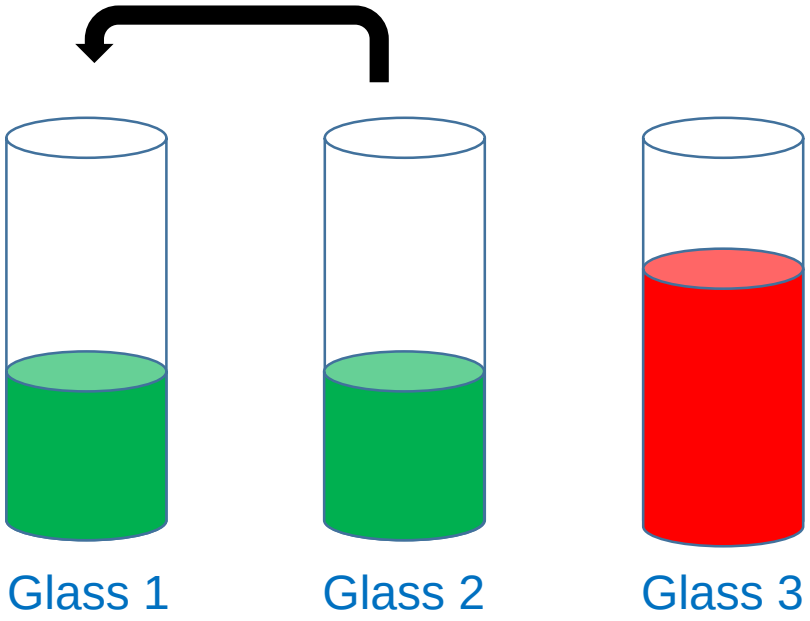
Transfer content from glass 1 to glass 3



Examples -

Step 2 *(Actual scenario if a glass is a variable)*

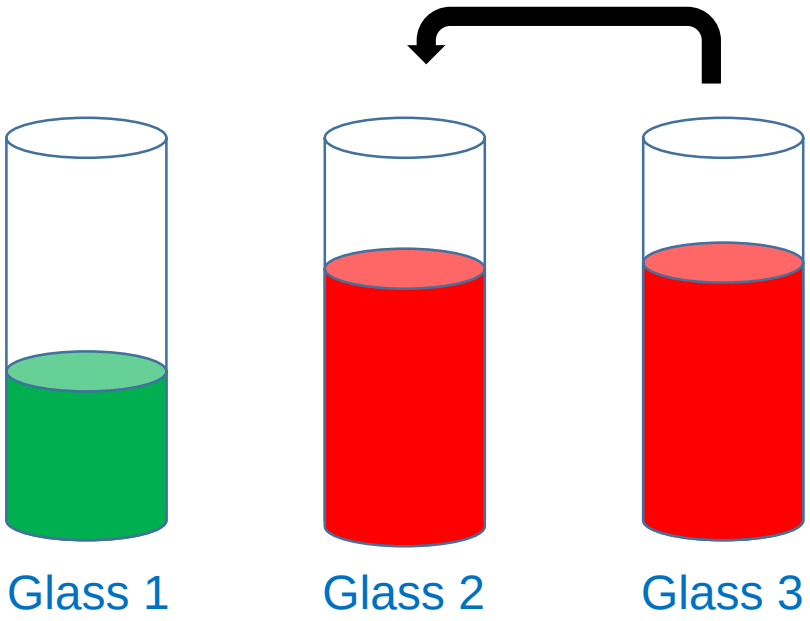
Transfer content from glass 2 to glass 1



Examples -

Step 3 *(Actual scenario if a glass is a variable)*

Transfer content from glass 3 to glass 2



Examples -

Example 2: *Swap without Temporary Variables*

```
1.    #include <stdio.h>
2.    int main()
3.    {
4.        double firstNumber, secondNumber;
5.        printf("Enter first number: ");
6.        scanf("%lf", &firstNumber);
7.        printf("Enter second number: ");
8.        scanf("%lf",&secondNumber);
9.        // Swapping process
10.       firstNumber = firstNumber - secondNumber;
11.       secondNumber = firstNumber + secondNumber;
12.       firstNumber = secondNumber - firstNumber;
13.       printf("\nAfter swapping, firstNumber = %.2lf\n", firstNumber);
14.       printf("After swapping, secondNumber = %.2lf", secondNumber);
15.       return 0;
16.   }
```

Examples -

Output

Enter first number: 10.25

Enter second number: -12.5

After swapping, firstNumber = -12.50

After swapping, secondNumber = 10.25

Examples -

What other mathematical operations can support the same ?

Ternary Operator

- +, -
- A + B – there are two operands ….
- A * B ---

- A>B ? 1 : 2
- OPRND1 ? OPRND2 : OPRND3

if(OPRND1)
    Execute OPRND2
Else
    Execute OPRND3

Ternary Operator: Example

```
If(a==1)
        printf(“abcd\n”)
else
        printf(“efgh\n”)
-----------

(a==1)? printf(“abcd\n”):printf(“efgh\n”);
```

Ternary Operator: Example

a>b ? e==f : g==h

With brackets -

a>b ? (e==f ? printf(“yes”): printf(“no”)):
(g>h ? printf(“1”): printf(“2”))

Without brackets -

What is the rule for ternary operator ??
When multiple ternary operators are mixed together ??

- D==b>c?a:n==g?s:t – Do it on your own / Check in program….

……

- A=0;
- B=1;
- C=2
- Printf(“%d”,A?B:C);