Logic Building

31-Jan 2022 to 05 Feb 2022

Day-1

Trainers:

1. Dr. Abhay Kothari
2. Prof. Mubeen Ahmed Khan

SIRT SAGE University,

Department of Computer Science and Engineering

Contents

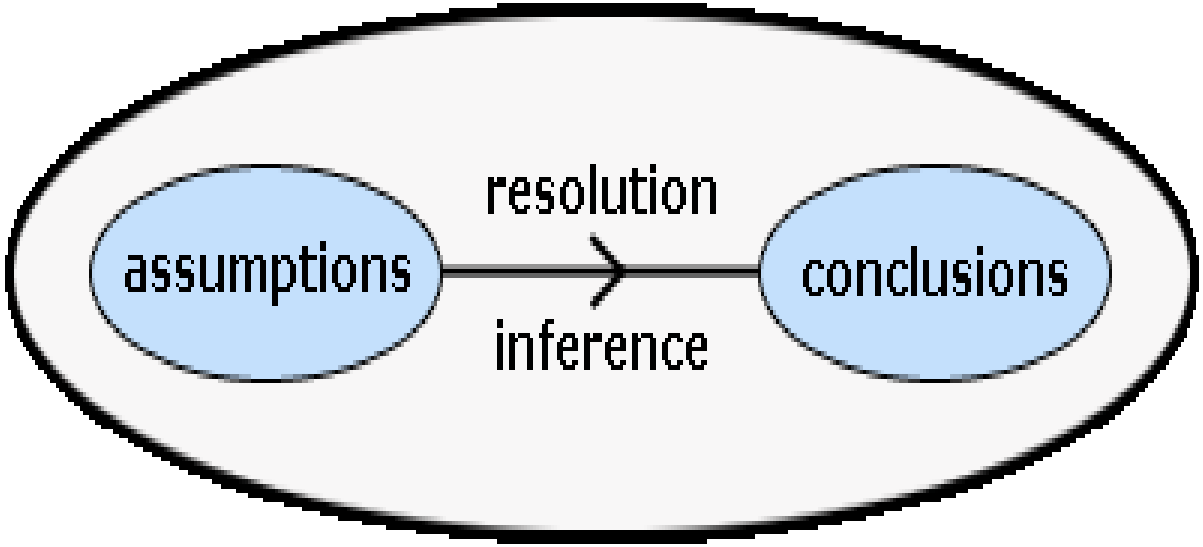
- Introduction to Logics
- Introduction to Programming Languages
- Algorithms
- Flow Charts
- Variables
- Identifiers
- Keywords
- Data Types

What is Logic programming?

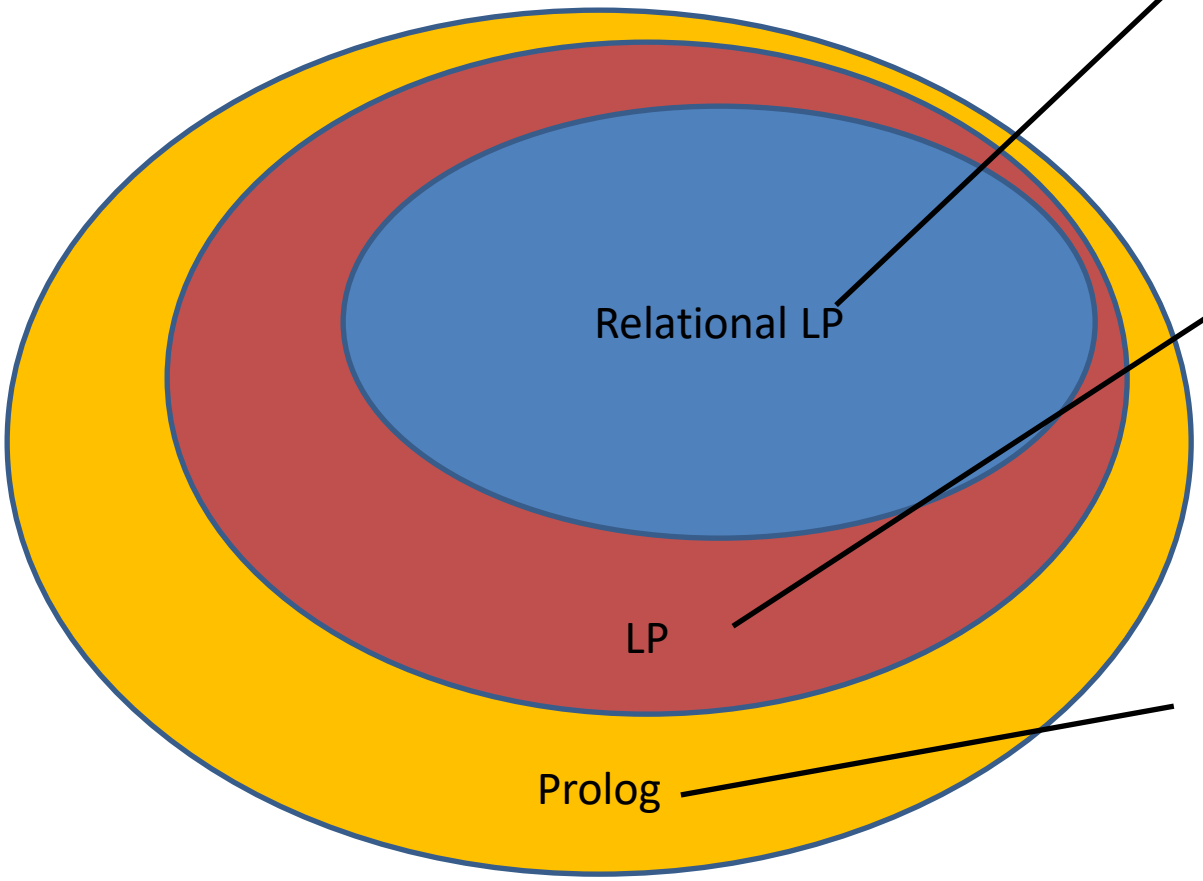
- **Logic Programming** is a method that computer scientists are using to try to allow machines to reason because it is useful for knowledge representation. In **logic programming**, **logic** is used to represent knowledge and inference is used to manipulate it.

- The logic used to represent knowledge in logic programming is used because first-order logic is well understood and able to represent all computational problems. Knowledge is manipulated using the resolution inference system which is required for proving theorems in clausal-form logic. The diagram below shows the essence of logic programming.

- **Clausal-Form Logic**



Logic Programming



typeless atomic terms, type safe
decidable
multi-directional

functors,
typeless composite terms,
type safe
partially decidable,
multi-directional

arithmetics,
uni-directional
dynamically typed,
not type safe,
system predicates (e.g. !)

Points to improve your programming logic

- Think to solve
- Practice
- Learn about Data Structures and Algorithms
- Learn programming paradigms
- Look at other people's code

Introduction to Programming Languages

- Programming languages are the languages that are designed to perform various computational tasks such as mathematical, general purpose, engineering, scientific and so on used in our day to day life span.

Types of programming languages:

- First generation
- Second generation
- Third generation
- Fourth generation

First Generation Language

- Merits
  - As machine language statements are written in binary language. It makes fast and efficient use of computer
  - Requires no translator to translate the code
- Demerits
  - It is difficult to learn and understand the instruction in machine language
  - It is very difficult to program in machine language. The programmers has to know details of hardware to write programs.
  - It is difficult to find errors in program written in machine language

Second Generation Language

- Assembly Language
- Human readable notations for machine language which is used to control computer operations.
- The programmer write the code using symbolic instructions codes that are meaningful abbreviations or mnemonics. It is also called low level programming languages. During the execution the assembler converts it into machine language.
- Merits
  - Easier to understand as compared to machine language
  - Code can be easily modified, if required
  - Easy to locate and correct errors
- Demerits
  - Machine dependent, programmers needs to understand the hardware
  - Assembler is required to translate the code from the assembly language to machine language

Third generation language

- 3GL or procedural language use series of English like words, which are close to human language, to write instructions.
- Called high level programming languages
- Complex but easy to read, understand and maintain.
- During execution, the programs written in high level languages need to translated using either compiler or interpreter. Eg. C, Pascal, Fortran, COBOL, ADA and so on.
- Merits:
  - Similar to English vocabulary
  - Can be translated into many machine languages and can run on any computers
- Demerits:
  - Each time the compiler or interpreter is needed to translate the high level language into machine code
  - The object code is generated which is insufficient as compared to equivalent assembly language.

Fourth generation Language

- 4GL is non procedural language enables to access database. Examples of 4GL languages are SQL, Focus, FoxPro, Power builder
- Merits:
  - The GL language are easy to understand and learn
  - The application can be created in very less
  - These are less prone to error
  - The automation tool can used for developing the tasks
- Demerits
  - These language consume lot of memory space
  - These languages are less flexible
  - The 4GL languages has poor control over hardware and hence tend to use less machine recourses

Program design

- Top Down
- Bottom up

Programming Paradigm

- Procedural Programming C
- Object Oriented Programming C++,JAVA
- Functional Programming
- Logic Programming PROLOG

Features of C Programming Language

- **Procedural Language:** In a procedural language like C step by step predefined instructions are carried out.
- **Fast and Efficient:** C programming language as the been middle-level language provides programmers access to direct manipulation with the computer hardware but higher-level languages do not allow this.
- **Modularity:** The concept of storing C programming language code in the form of libraries for further future uses is known as modularity.
- **Statically Type:** C programming language is a statically typed language. Meaning the type of variable is checked at the time of compilation but not at run time.
- **General Purpose Language:** From system programming to photo editing software, the C programming language is used in various applications. Some of the common applications where it's used are as follows:
  - Operating systems: Windows, Linux, iOS, Android, OXS
  - Databases: PostgreSQL, Oracle, MySQL, MS SQL Server etc.
- **Rich set of built-in Operators:** It is a diversified language with a rich set of built-in operators which are used in writing complex or simplified C programs.
- **Libraries with rich Functions:** Robust libraries and functions in C help even a beginner coder to code with ease.
- **Portability:** C language is lavishly portable as programs that are written in C language can run and com
- **Easy to Extend:** Programs written in C language can be extended means when a program is already written in it then some more features and operations can be added to it. pile on any system with either none or small changes.
- **Middle-Level Language:** As it is a middle-level language so it has the combined form of both capabilities of assembly language and features of the high-level language.

Features of C++ Programming Language

- Object Oriented Programming
- Machine Independent
- Simple
- Intermediate Level Programming Language
- Compiler-Based
- Dynamic Memory Allocation
- Integration and Extendibility

Features of Functional Programming Language

- **State does not exist:** FP Programs does not contain state. That means all Data is Immutable Data and Functions cannot change state.
- **Low importance of Order of Execution:** In FP Languages, we write programs with a set of Independent Functions. Functions contain a set of statements. In FP, the order of execution of those Functions does not have much importance because they do not have state and all Functions work independently. Even if we change the order of execution still they produce the same results.
- **Stateless Programming Model:** All FP Programs uses Immutable data and Functions, which cannot modify that data. That means FP Languages support Stateless Programming Model.
- **Functions are first class citizens:** In FP Languages, Functions are first class objects. Functions are independent units, we can execute them in any order.
- **Primary Manipulations Units:** In FP Languages, Primary Manipulations units are Functions and Data Structures because All Programs are made up of using these units.
- **Modular Programming:** In FP Languages, we need to write smaller and independent units, called Pure Functions to support Stateless Programming model. That means FP supports better Modularity than OOP.
- **Higher-order Functions and Lazy Evaluation:** Functional Programming Languages should support Higher-order functions and Lazy Evaluation features.
- **Primary Flow Controls:** FP Languages don't use Flow Controls like For…Loop, Do…While Loop, While…Loop etc and also don't use Conditional statements like If..Else or Switch Statements. All FP Languages write programs using the following things:
  - Functions
  - Function calls & Function calls with Recursion
- **Abstraction, Encaspulation,Inheritance and Polymorphism:** Like OOP, FP Languages supports all 4 concepts : Abstraction,Encaspulation,Inheritance and Polymorphism. FP Languages supports Inheritance with Type Classes or Implicits. They support Polymorphism with the help of Generics. It is also known as Parametric Polymorphism.

Features of Logic Programming

- Logical programming can be used to express knowledge in a way that does not depend on the implementation, making programs more flexible, compressed and understandable.
- It enables knowledge to be separated from use, i.e. the machine architecture can be changed without changing programs or their underlying code.
- It can be altered and extended in natural ways to support special forms of knowledge, such as meta-level of higher-order knowledge.
- It can be used in non-computational disciplines relying on reasoning and precise means of expression.

Algorithms

- Collection of unambiguous instructions occurring in some specific sequences
- An algorithm is a step-by-step method for solving some problem.

Algorithms generally have the following characteristics:

- **Input:** The algorithm receives input. Zero or more quantities are externally supplied.
- **Output:** The algorithm produces output. At least one quantity is produced.
- **Precision:** The steps are precisely stated. Each instruction is clear and unambiguous.
- **Feasibility:** It must be feasible to execute each instruction.
- **Flexibility:** It should also be possible to make changes in the algorithm without putting so much effort on it.
- **Generality:** The algorithm applies to a set of inputs.
- **Finiteness:** Algorithm must complete after a finite number of instruction have been executed.

Analysis (Complexity) of Algorithms

- The Analysis of an algorithm refers to the process of deriving estimates for the time and space needed to execute the algorithm.
- It is important to estimate the time (e.g., the number of steps) and space (e.g., the number of variables) required by algorithms. Knowing the time and space required by algorithm allows us to compare the algorithms that solve the same problem. For example, if one algorithm takes n steps to solve a problem and another algorithm takes n^2 steps to solve the same problem, we would prefer the first algorithm. This estimation of time and space needed to execute the algorithm is called the time and space complexity of the algorithm.

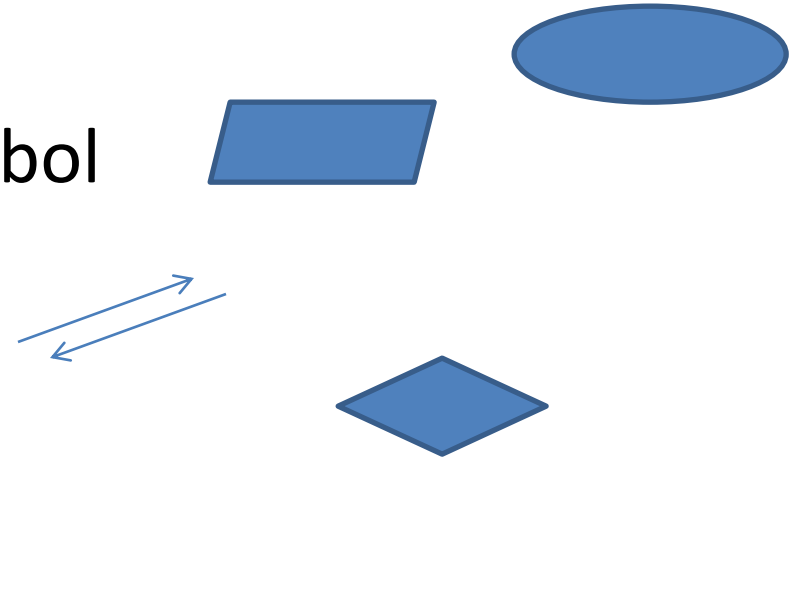
Complexity

- Time complexity: the time complexity is the amount of time taken by the algorithm to execute
- Space complexity: The space complexity is an amount of space taken by the algorithm.

Flow Charts

- Graphical representation of algorithm
- Uses symbols and logical flow
- Helps in understanding the program which is not written on computer for execution

1. Start and stop symbol
2. Input and output symbol
3. Process
4. Flow of information
5. Decision symbol
6. Connector



Variables

- A memory location which can store some data/ value of different types
- Variable is a name given to a particular storage area inside the memory
- Every variable having a data type, which define the size of the variable in memory and the range of values that can be stored within the variable.
- A variable is a code which tells the C compiler- where and how much space to create inside memory storage.
- A variable syntax contains data type, variables name and value which we want to store.
- <data-type><variable-name>;
- Int num1;
- Double principal;
- Float time,rate,si;

Identifiers

Type Name	Bytes	Other Names	Range of Values
int	4	**signed**	-2,147,483,648 to 2,147,483,647
unsigned int	4	**unsigned**	0 to 4,294,967,295
__int8	1	**char**	-128 to 127
unsigned __int8	1	**unsigned char**	0 to 255
__int16	2	**short, short int, signed short int**	-32,768 to 32,767
unsigned __int16	2	**unsigned short, unsigned short int**	0 to 65,535

Data Types

Type Name	Bytes	Other Names	Range of Values
__int32	4	**signed**, **signed int**, **int**	-2,147,483,648 to 2,147,483,647
unsigned __int32	4	**unsigned**, **unsigned int**	0 to 4,294,967,295
__int64	8	**long long**, **signed long long**	-9,223,372,036,854,775,808 to 9,223,372,036,854,775,807
unsigned __int64	8	**unsigned long long**	0 to 18,446,744,073,709,551,615
bool	1	none	**false** or **true**

Data Types

Type Name	Bytes	Other Names	Range of Values
char	1	none	-128 to 127 by default 0 to 255 when compiled by using /J
signed char	1	none	-128 to 127
unsigned char	1	none	0 to 255
short	2	**short int, signed short int**	-32,768 to 32,767
unsigned short	2	**unsigned short int**	0 to 65,535
long	4	**long int, signed long int**	-2,147,483,648 to 2,147,483,647

Data Types

Type Name	Bytes	Other Names	Range of Values
unsigned long	4	**unsigned long int**	0 to 4,294,967,295
long long	8	none (but equivalent to **__int64**)	-9,223,372,036,854,775,808 to 9,223,372,036,854,775,807
unsigned long long	8	none (but equivalent to **unsigned __int64**)	0 to 18,446,744,073,709,551,615
enum	varies	none	
float	4	none	3.4E +/- 38 (7 digits)
double	8	none	1.7E +/- 308 (15 digits)
long double	same as **double**	none	Same as **double**
wchar_t	2	**__wchar_t**	0 to 65,535

THANK YOU

Day 1 Lab

- By Prof. Mubeen Ahmed Khan
- Dr. Abhay Kothari

Contents of Lab1

- Algorithm for addition, subtraction, Multiplication and division of two numbers
- Algorithm for finding smaller among two, three and four numbers
- Flow chart for addition of two numbers
- Flow chart for average of three numbers

Algorithm for addition of two numbers

- Step1: Start
- Step2: Declare variables num1 and num2
- Step3: Read Values num1 and num2
- Step4: Add num1 and num2 and sum←num1+num2
- Step5: Display sum
- Step6: Stop

Algorithm for subtraction of two numbers

- Step1: Start
- Step2: Declare variables num1 and num2
- Step3: Read Values num1 and num2
- Step4: Add num1 and num2 and sub←num1-num2
- Step5: Display sub
- Step6: Stop

Algorithm for finding smaller among two

- Step1: Begin
- Step2: get num1
- Step3: get num2
- Step4: if(num1<num2)
- Step5: Display num1 is smaller than num2
- Step6: Else Display num2 is smaller
- Step7: end if
- Step8: end

Algorithm for finding smaller among three numbers

- Step1: Start
- Step2: Read the values of a,b,c
- Step3: if(a>b) and (a>c) Than
  - Print a is larger
  - Elseif(b>c)
  - Print b is largest
  - Else
  - Print c is largest
- Step4: Stop

Smaller among 4 numbers

```
1.    INPUT a, b, c, d
2.    SET minn := a, maxx := a
3.    IF b < minn THEN
4.        minn : = b
5.    END IF
6.    IF c < minn THEN
7.        minn := c
8.    END IF
9.    IF d < minn THEN
10.       minn := d
11.   END IF
12.   IF b > maxx THEN
13.       maxx := b
14.   END IF
15.   IF c > maxx THEN
16.       maxx := c
17.   END IF
18.   IF d > maxx THEN
19.       maxx : =d
20.   END IF
21.   PRINT minn, maxx
```

Algorithm for calculating average of numbers

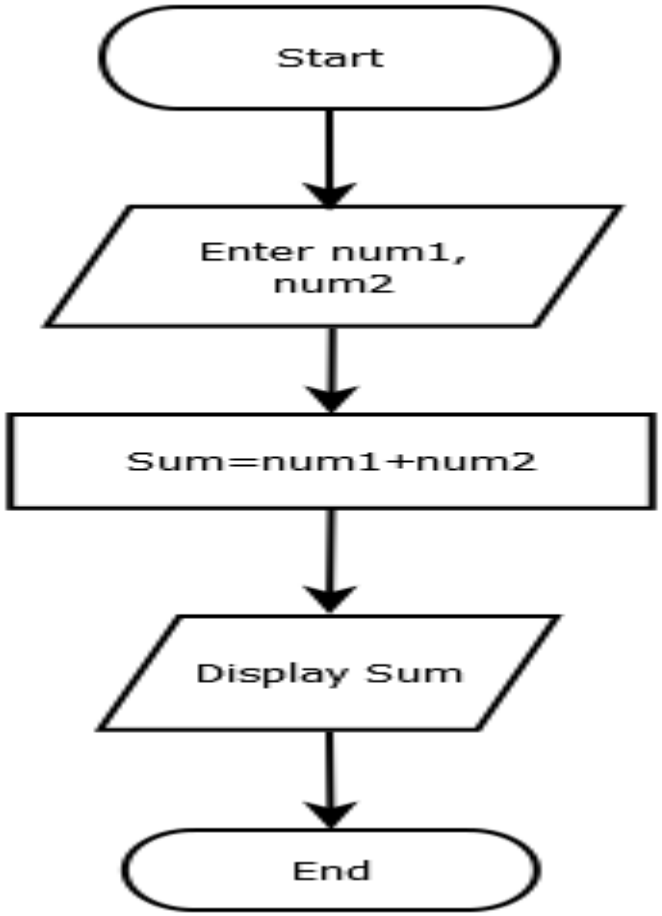
Algorithm Example

To calculate Average of three numbers:

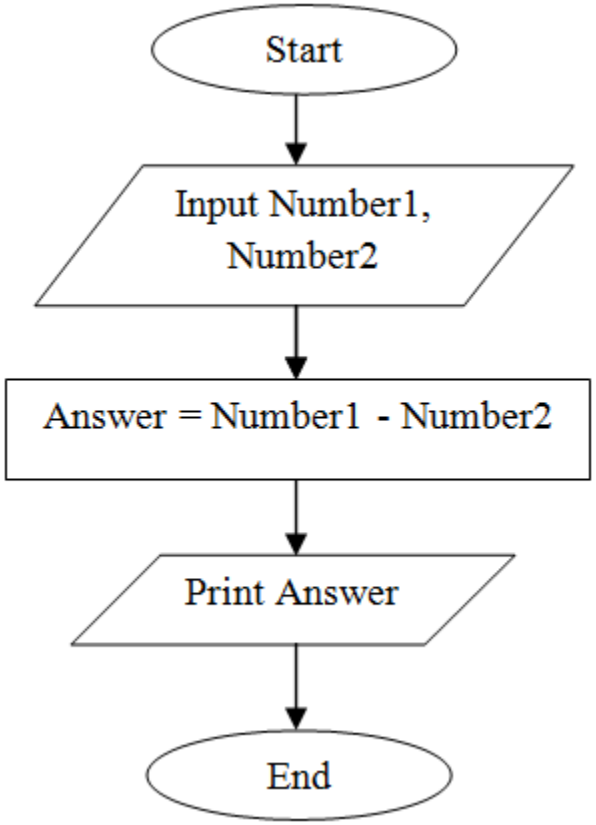
Algorithm:

STEP 1	START
STEP 2	INPUT X,Y,Z
STEP 3	COMPUTE SUM = X + Y + Z
STEP 4	COMPUTE AVG = SUM/3
STEP 5	PRINT SUM
STEP 6	PRINT AVG
STEP 7	END

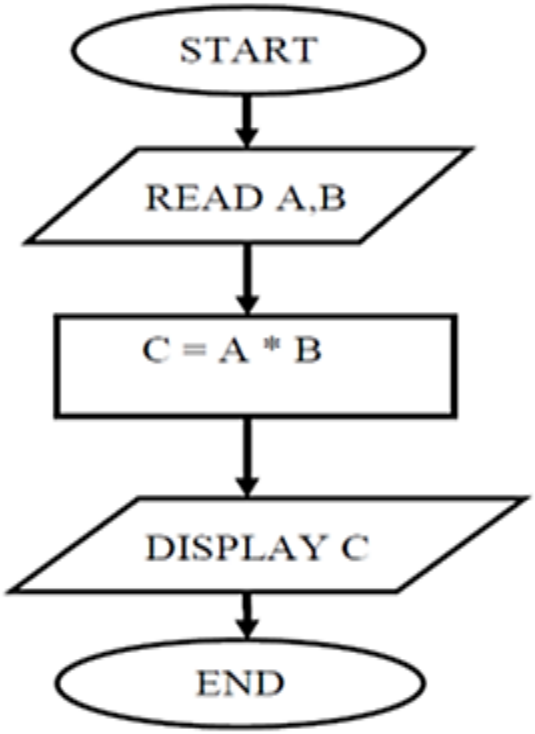
Flow chart for addition of two numbers



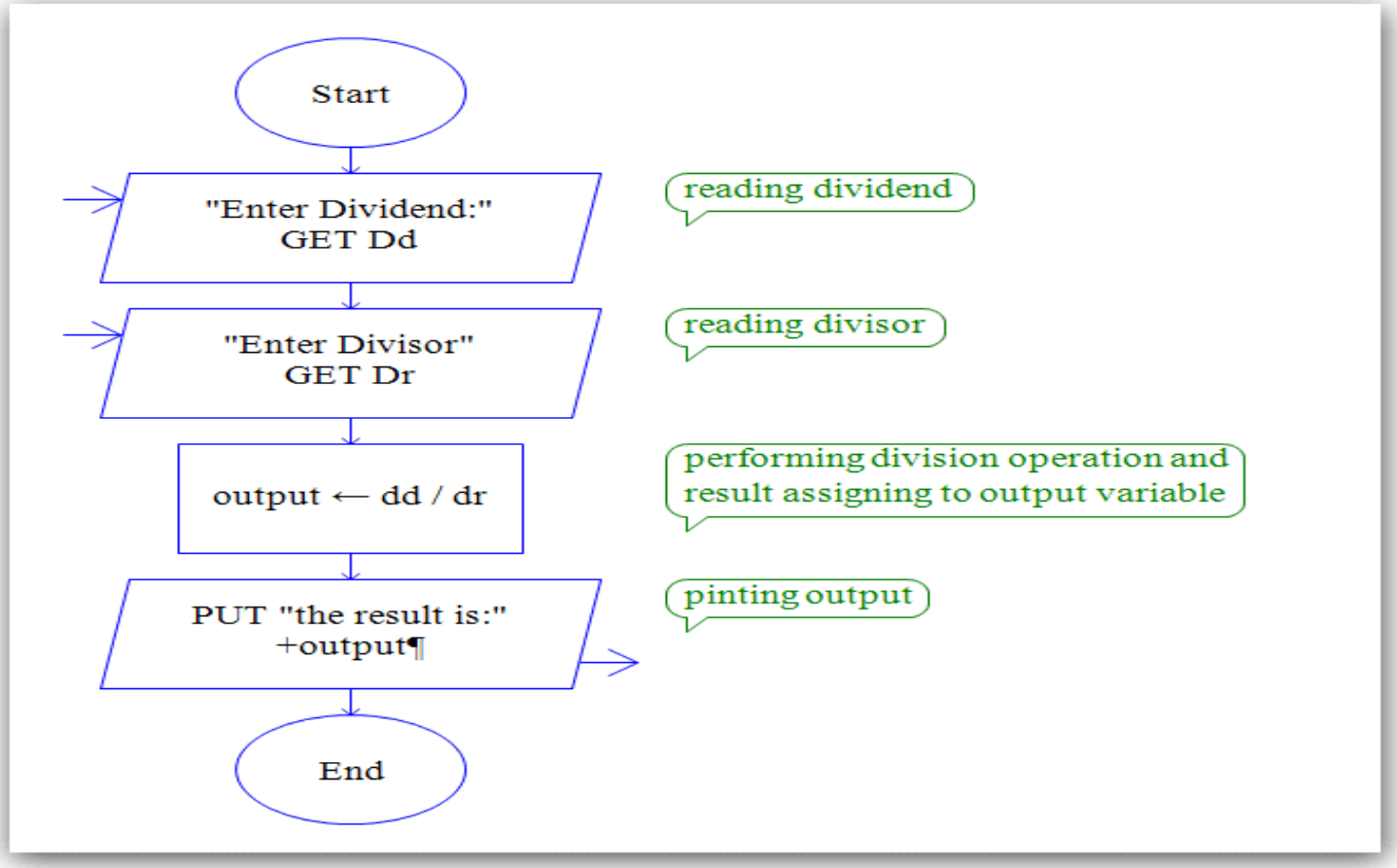
Flow chart for subtraction of two numbers



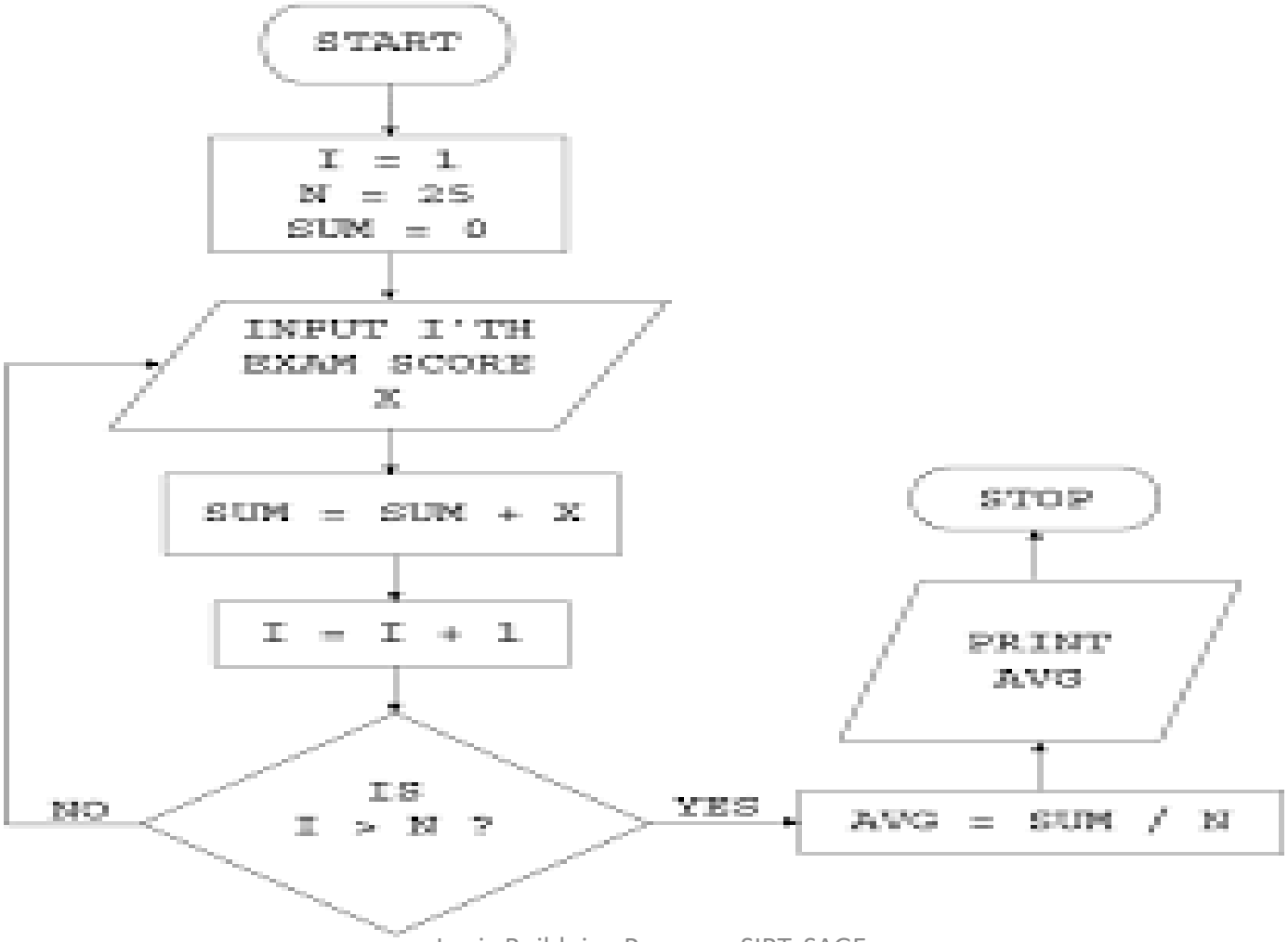
Flow chart for subtraction of two numbers



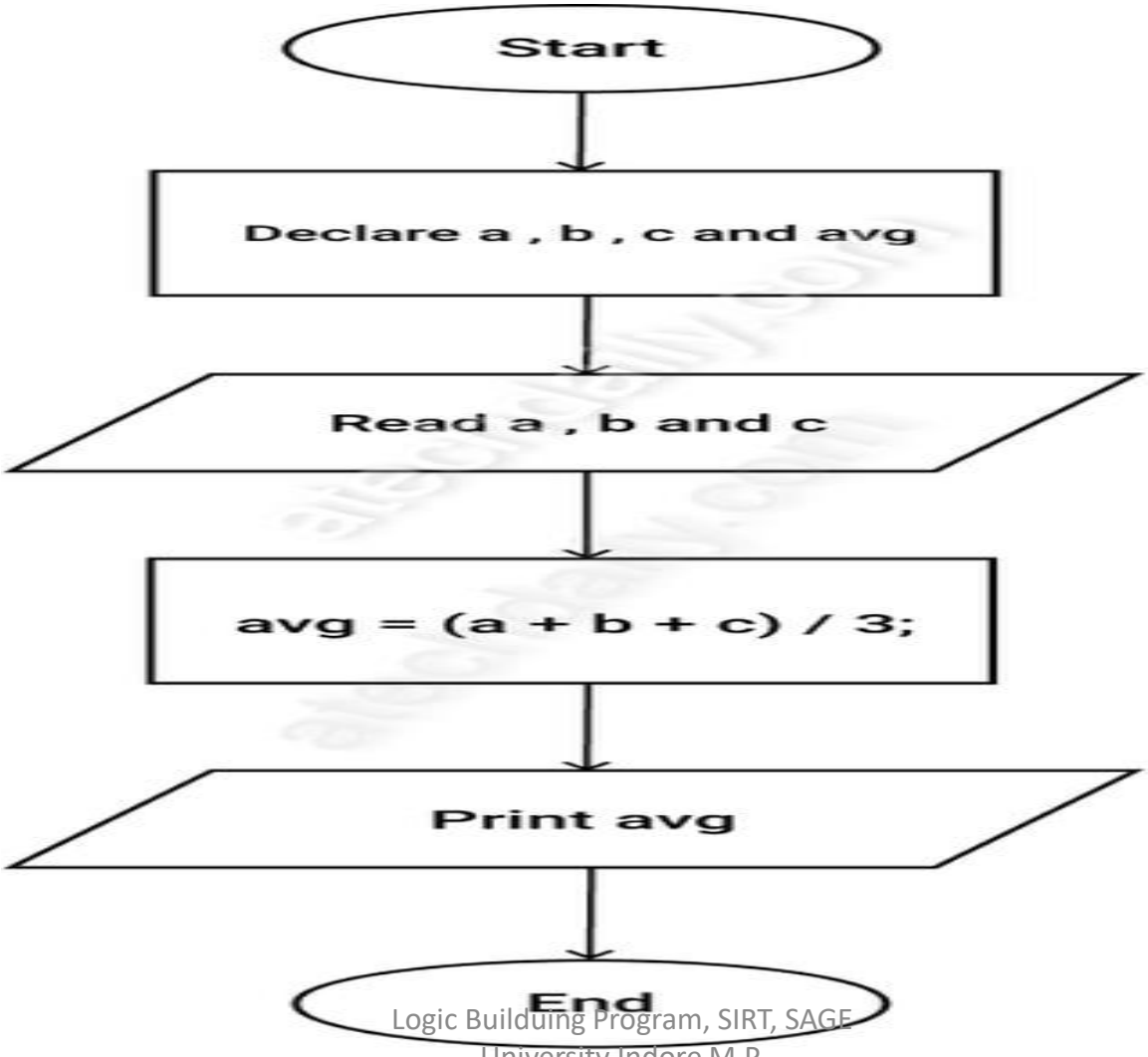
Division of two numbers



Flow chart for calculating average



Flow chart for calculating average 3 numbers



- THANK YOU

Assignment-1

- Write a program to print even/odd numbers up to the given range.
- Write a program to print table by simple addition method
- Write a program to show the given number is even or odd.
- Write a program to convert Fahrenheit to Celsius (c=(f-32)*5/9)
- Write a program to print the ASCII value of a given number
- Write a program to calculate lowest out of three numbers.

Pogram to print even/odd numbers u to the given range

Program

```
# include <iostream.h>
# include <conio.h>
void main()
{
    int range;
    clrscr();
    cout << "Enter range : ";
    cin >> range;
    int i=1;
    while(i<=range)
    {
        if(i%2!=0)
            cout << i << "\t";
            i++;
    }
    getch();
}
```

Output

```
Enter range : 30
1   3   5   7   9   11
13  15  17  19  21  23
25  27  29
```

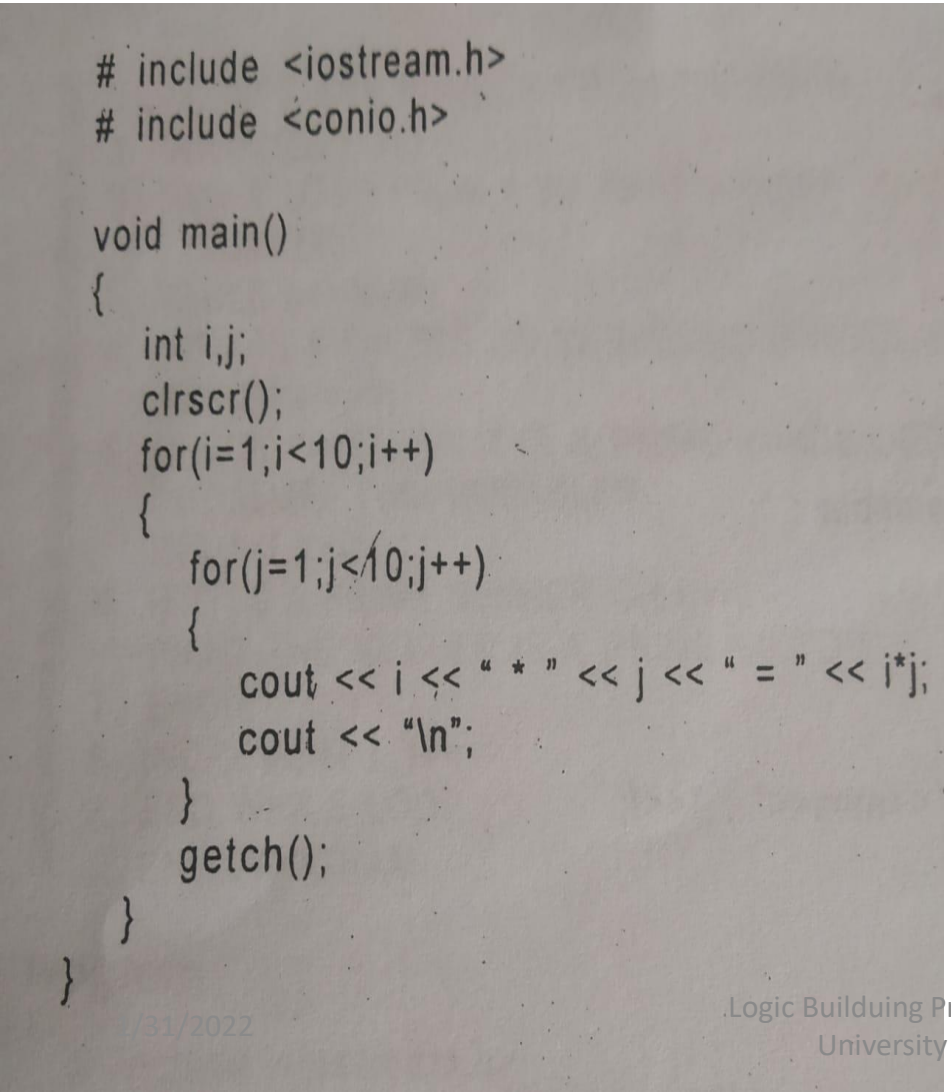
Program

```
# include <iostream.h>
# include <conio.h>
void main()
{
    int range;
    clrscr();
    cout << "Enter range : ";
    cin >> range;
    int i=1,c=0,s=0;
    while(i<=range)
    {
        if(i%2==0)
        {
            s=s+i;
            c++;
        }
        i++;
    }
    cout <<"average of all even numbers upto "<<range<<" : "<<s/c;
    getch();
}
```

Output

```
Enter range : 20
average of all even numbers upto 20 : 11
```

Write a program to print table by simple addition method



```
# include <iostream.h>
# include <conio.h>

void main()
{
   int i,j;
   clrscr();
   for(i=1;i<10;i++)
   {
      for(j=1;j<10;j++)
      {
         cout << i << " * " << j << " = " << i*j;
         cout << "\n";
      }
      getch();
   }
}
```

```
#include<stdio.h>
#include<conio.h>
int main()
{
 int num, i, tab;
printf("Enter the number: ");
 scanf("%d", &num);
printf("\nTable of %d is:\n", num);
 for(i=1; i<=10; i++)
{
 tab = num*i;
printf("%d * %2d = %2d\n", num, i, tab);
}
getch();
 return 0;
 }
```

Write a program to convert Fahrenheit to Celsius

```
# include <iostream.h>
# include <conio.h>
void main()
{
   int a[20],d,e,o,i;
   clrscr();
   cout <<"enter dimension : ";
   cin >>d;
   cout <<"\nenter "<<d<<"values : ";
   for(i=0;i<d;i++)
   cin >>a[i];
   e=o=0;

   for(i=0;
   {
      if(a[i]%2==0)
      e++;
      else
      o++;
   }
   cout <<"\nelements of array a : \n";
   for(i=0;i<d;i++)
   cout  <<a[i]<<"\t";;
   cout <<"\nno.of odd numbers  : "<<o;
   cout <<"\nno.of even numbers : "<<e;
   getch();
}
```

```
# include <iostream.h>
# include <conio.h>
void main()
{
   float c,f;
   clrscr();
   cout <<"enter fahrenheit : ";
   cin >>f;

   c=(f-32)*5.0/9.0;
   cout <<"\nfahrenheit : "<<f;
   cout <<"\ncelsius    : "<<c;
   getch();
}
```

Write a program to print the ASCII value of a given number

```
# include <iostream.h>
# include <conio.h>
void main()
{
    int i;
    clrscr();
    for(i=0;i<256;i++)
    {
        if(i!=26 &&i!=29)
            cout <<i<<" = "<<(char)i<<"\t";
    }
    getch();
}
```

Write a program to calculate lowest out of three numbers.

```
# include <iostream.h>
# include <conio.h>
void main()
{
   int a,b,c;
   clrscr();
   cout << "Enter three values : ";
   cin >> a >> b >> c;
   if(a<b && a<c)
      cout << a <<" is the lowest value ";
   else if(b<c)
      cout << b <<" is the lowest value ";
   else
      cout << c <<" is the lowest value ";
   getch();
}
```

- Exit