Data Structure and Algorithms

Chapter 3

Computation Fundamentals

Objectives

At the end of this chapter, you will be able to:

- –Understand different types of data types
- –Understand Input/Output/Process requirements
- –Understand usage of operators and expressions
- –Understand calls of Expressions, Operators, Expressions and statements

Understanding the Need of Data Types

- Data type is used to identify various types of data
- Different data types are used for storing different kinds of data
- Different data type provides different :
  - Storage
  - Space
  - Format
- Basic types allow minimum memory

Data Types

Size and Range of Data Types on 16 bit machine.

TYPE	SIZE (Bits)	Range
Char or Signed Char	8	-128 to 127
Unsigned Char	8	0 to 255
Int or Signed int	16	-32768 to 32767
Unsigned int	16	0 to 65535
Short int or Signed short int	8	-128 to 127
Unsigned short int	8	0 to 255
Long int or signed long int	32	-2147483648 to 2147483647
Unsigned long int	32	0 to 4294967295
Float	32	3.4 e-38 to 3.4 e+38
Double	64	1.7e-308 to 1.7e+308
Long Double	80	3.4 e-4932 to 3.4 e+4932

Understanding the Need of Variable

- Variable is used to store temporary data
- Variable has a name and data type
  - variable name is used to access and manipulate the value of the variable
  - data type determines the type of data that the variable can store

Different types of data

- Scalar data
- Vector data
- Associated array

Scalar data

Scalar data and data type:

- –Scalar data holds one value at a time
- –Scalar data type is the type of a scalar variable.
- –For example, char, int, float, and double are the most common scalar data types

Scalar Operators

- Produces a new result from one or more other values
  - –For example, + is an operator because it takes two numbers (the operands, like 8 and 7), and produces a new value (15, the result).
- Operators for Numbers
  - –Addition, subtraction, multiplication, and division operators
- Operators for Strings
  - –String values can be concatenated with the " . " operator
- Numeric and String Comparison Operators

Scalar Operators

Comparison	Numeric	String
Equal	==	eq
Not equal	!=	ne
Less than	<	lt
Greater than	>	gt
Less than or equal to	<=	le
Greater than or equal to	>=	ge

Vector data

- Vector data :
  - –is used in GIS
  - –represent the location of a point, a line, and an area
- Vector graphics ត្រូវបានប្រើសម្រាប់ធរណីមាត្រ ។
- Vector graphics មានលក្ខណៈផ្ទុយពី raster graphics:
  - raster graphics គឺជាតំណាងនៃរូបភាពជា array និង pixels
  - ប្រើសម្រាប់តំណាងរូបភាព រូបថត

Associated array

Associative array:

- An abstract data type
- A collection of unique keys and a collection of values
- A key is associated with one value

Associated array

Example : associative container, map, mapping, dictionary, finite map, and in query-processing an index or index file

Key	Value
John	1
Michel	2
Bob	3
Allen	4

Input/Output/Process

Input/Output/Process

- Input
  - –Datasheets
  - –Labor
  - –Computer
  - –Electricity
- Process
  - –Data Entry
  - –File Data Sheets
- Output
  - –Entered Data
  - –Filed Data Sheets

Operators

- Types of operators:
  - –Arithmetic operators
  - –Assignment operators
  - –Unary operators
  - –Comparison operators
  - –Logical operators

Arithmetic

Operator	Description	Example	Explanation
+	Adds the operands	X = y + z	Adds the value of y and z and stores the result in x
-	Subtracts the right operand from the left operand	X = y - z	Subtracts z from y and stores the result in x
*	Multiplies the operands	X = y * z	Multiples the values y and z and stores the result in x
/	Divide the left operand by the right operand	X = y / z	Divides y by z and stores the result in x
%	Calculates the remainder of an integer division	X = y % z	Divides y by z and stores the remainder in x

Assignment

Operator	Description	Example	Explanation
+=	Adds the operands and assigns the result to the left operand	X += y	Adds the value of y to x and stores the result in x Equivalent to x = x + y
-=	Subtracts the right operand from the left operand and stores the result in the left operand	X -= y	Subtracts y from x and stores the result in x Equivalent to x = x - y
*=	Multiplies the left operand by the right operand and stores the result in the left operand	X *= y	Multiples the values x and y and stores the result in x Equivalent to x = x * y
/=	Divide the left operand by the right operand and stores the result in the left operand	X /= y	Divides x by y and stores the result in x Equivalent to x = x / y
%=	Divides the left operand by the right operand and stores the remainder in the left operand	X %= y	Divides x by y and stores the remainder in x Equivalent to x = x % y

Unary

Operator	Description	Example	Explanation
++	An increment operator increases the value of the operand by one	x++	Equivalent to x = x + 1
--	The decrement operator decreases the value of the operand by one	x--	Equivalent to x = x - 1

Comparison

Operator	Description	Example	Explanation
==	Evaluates whether or not the operands are equal	x == y	Returns true if the values are equal and false otherwise
!=	Evaluates whether or not the operands are not equal	x != y	Returns true if the values are not equal and false otherwise
>	Evaluates whether or not the left operand is greater than the right operand	x > y	Returns true if x is greater than y and false otherwise
<	Evaluates whether or not the left operand is less than the right operand	x < y	Returns true if x is less than y and false otherwise
>=	Evaluates whether or not the left operand is greater than or equal to the right operand	x >= y	Returns true if x is greater than or equal to y and false otherwise
<=	Evaluates whether or not the left operand is less than or equal to the right operand	x <= y	Returns true if x is less than or equal to y and false otherwise

Logical

Operator	Description	Example	Explanation
&&	Evaluates to true, if both the conditions evaluate to true, false otherwise	x > 5 && y < 10	The result is true if condition1 (x>5) and condition2 (y<10) are both true. If one of them is false, the result is false.
||	Evaluates to true, if at least one of the conditions evaluate to true, false if none of the conditions evaluate to true.	x > 5 || y < 10	The result is true if condition1 (x>5) or condition2 (y<10) or both evaluate to true. If both the conditions are false, the result is false.
!	The effect of the ! operator is that the logical value of its operand is reversed.	!(x ==0)	The result is true, if the condition (x==0) is false. If the same condition is true then the result is false.

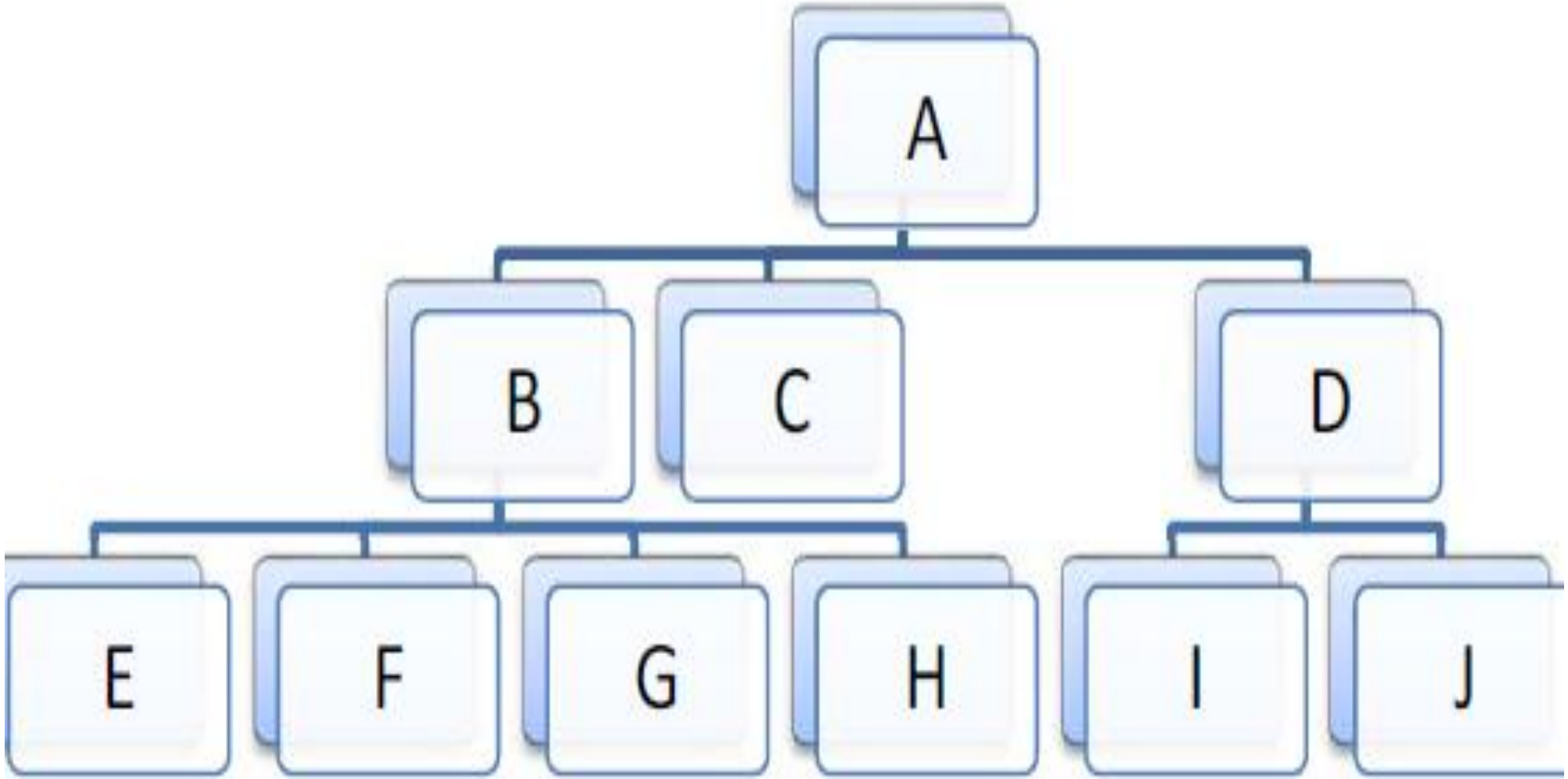
ការធ្វើគម្រោង

ការរៀបចំគម្រោង គឺជាវិធីមួយទៅតាមគម្រោងការវិភាគ Algorithms របស់ចំណោទបញ្ហាមួយមុនពេលឈានដល់សរសេរកម្មវិធី ។ ការរៀបចំកម្មពិធីចាំបាច់ត្រូវបង្កើតយុទ្ធសាស្ត្រទាំងឡាយដោយធ្វើការប្រមូល ពួកវាដាក់បញ្ចូលគ្នាវិធីសាស្ត្រដែលយើងនឹងលើកយកមកធ្វើការដោះស្រាយនោះគឺការធ្វើគម្រោងពីលើចុះក្រោម Top down Design ។

Top down Design

គឺជាវិធានធ្វើគំរោងតាងភ្ជាប់ជាមួយកម្មវិធី ។ គោលការណ៍ Top down design ត្រូវបានអនុវត្តន៍តាមលំនាំបែងចែកចំណោទបញ្ហាដែលមានទ្រង់ទ្រាយធំនិងមានលក្ខណៈសុគស្មាញអោយទៅជាបញ្ហាតូចៗ ជាបន្តបន្ទាប់ដែលតាងដោយ Module ហើយការបែងចែក Module ត្រូវបានអនុវត្តជាបន្តបន្ទាប់ អោយក្លាយជាបញ្ហាមូលដ្ឋានងាយស្រួលក្នុងការដោះស្រាយ ។

យើងបាន structure ទូទៅរបស់ Top down design តាងដោយ module ដូចខាងក្រោម:


A
B
C
D
E
F
G
H
I
J

ការបែងចែក Module កំណត់ដោយ:

- Module ដំបូងត្រូវមានកម្រិតសូន្យ
- Module ទាំងឡាយ ដែលមានកម្រិត i+1)Level i+1) ត្រូវបានគេបែងចែកចេញពីកម្រិតi)Level i) ជាមួយ i ≥0 (i=1,2,3,...) ។

ការបង្កើតឡើងនូវ Top Down Design សម្រាប់ទៅតាមចលនការដោះស្រាយរបស់បញ្ហាយើងមានគោលការណ៍អនុវត្តរបស់ចំណោទបញ្ហាត្រូវការគម្រោងដូចខាងក្រោម:

Idea	Top down	Program
យុទ្ធសាស្ត្រទូទៅសំរាម ដោះស្រាយបញ្ហា ↓ ការបង្កើតផ្នែកជាចំបងៗ របស់បញ្ហា ↓ ផ្នែកនិមួយៗជាពិស្តារ	Module ដំបូង(Level 0) ↓ Sub Module ↓ Module ស្ថិតនៅ Level ចុងក្រោយ	កំណត់សរសេរកម្មវិធី ↓ Sub program ↓ Source code

Stepwise Refinement

គឺជាវិធានសំរាប់ធ្វើគំរោងតាងភ្ជាប់ជាមួយ Algorithms វាមានឥទ្ធិពលលើ Top down design ។ នៅពេលអនុវត្ត Algorithms របស់ Stepwise Refinement រួមមានភាសាពីរដែលបានបង្ហាញចេញគឺភាសាធម្មជាតិ និងភាសាកម្មវិធីអោយឈ្មោះថា Pseudo code រឺ Pseudo code language ដោយអនុវត្តជាបណ្តើរៗពី ភាសាធម្មជាតិឆ្លងតាម Pseudo code និងឈានទៅដល់ភាសាកម្មវិធីដែលយើងជ្រើសរើស និងដែលចាប់ផ្តើមពីចំនុច (what?) ដល់ចំនុចធ្វើដូចម្តេច (How?) ។

ឧទាហរណ៍ :

ចូរធ្វើគំរោង Stepwise Refinement របស់ចំណោទបញ្ហាបង្កើត Matrix (a_{ij}) ដែលមាន n rows និង n columns ដោយបង្ហាញចេញតំលៃធាតុនីមួយៗរបស់វានៅលើបន្ទាត់ស្របជាមួយអង្កត់ទ្រូងទី១ និងរាល់ធាតុដែលស្ថិតនៅលើអង្កត់ទ្រូងនោះ ។

Ex: $n = 3 \rightarrow (a_{ij})_{3x3}$

$$\Rightarrow a_{ij} = \begin{bmatrix} 20 & 9 & 40 \\ 7 & 6 & 30 \\ 80 & 4 & 50 \end{bmatrix}$$

ចំណោទបញ្ហានេះ Algorithms សំរាប់ដោះស្រាយរួមមានបីភារកិច្ចគឺ:

១-កំនត់តំលៃ Row និង Column របស់ Matrix

២ -កំនត់តំលៃធាតុនីមួយៗរបស់ Matrix a_{ij}

៣-បង្ហាញចេញធាតុនីមួយៗរបស់វានៅលើបន្ទាត់ស្របជាមួយអង្កត់ទ្រូង និងរាល់ធាតុស្ថិតនៅលើអង្កត់ទ្រូងនោះយើងមាន Algorithms របស់ Stepwise refinement នៅក្នុងចំណោទ

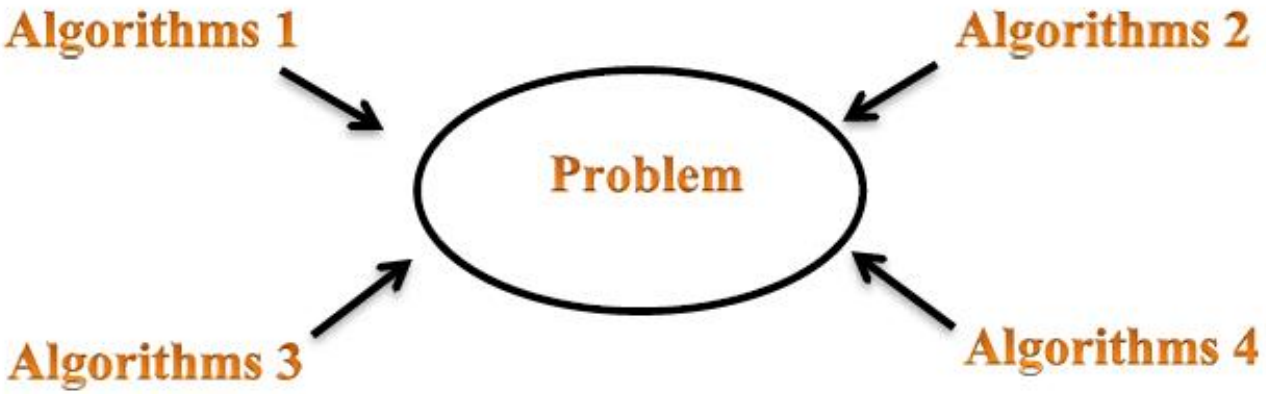
ចំពោះការកិច្ចទី១និងទី២ គេអាចសរសេរតាម C++ ដូចខាងក្រោម៖

1) cin>>n;
2) for (i=1; i<=n; i++){
   for (j=1; j<=n; j++)
   { cin>>a[i,j];}}

ការកច្ចនី៣

- ★ ជាមួយ j=n-1 បង្ហាញចេញ១ធាតុ(a_{01}=40) គឺធាតុមាន Row ទី0 និង Column ទីj ។
- ★ ជាមួយ j=n-2 បង្ហាញចេញ២ធាតុ(a_{01}=9; a_{12}=30) គឺធាតុមាន Row ទី0 និងColumn ទីj និងធាតុមានRow ទី1 និងColumn ទីj+1 ។
- ★ ជាមួយ j=n-3 បង្ហាញចេញ៣ធាតុ(a_{00}=20; a_{11}=6; a_{22}=50) គឺធាតុមាន Row ទី0 និងColumn ទីj , ធាតុមានRow ទី1 និងColumn ទីj+1 និងធាតុមានRow ទី2 និងColumn ទីj+2 ។

Algorithms Analysis



នៅក្នុងការដោះស្រាយបញ្ហាណាមួយ យើងមានមធ្យោបាយជាច្រើនសម្រាប់អនុវត្តទៅលើបញ្ហានោះ។ ប៉ុន្តែគេគប្បីជ្រើសរើសនូវមធ្យោបាយណាមួយដែលមានលក្ខណៈប្រសើរបំផុត ពោលគឺងាយយល់ ងាយបង្កើតកម្មវិធី និងចំណាយពេលអស់តិចក្នុងពេលដំណើរការកម្មវិធី។

តើយើងត្រូវធ្វើយ៉ាងណាដើម្បីអោយជ្រើសរើសបាននូវ Algorithms មួយដែលមានលក្ខណៈល្អ រឺអន់ថយ យើងមានកត្តាជាច្រើនសំរាប់ធ្វើការវាយតំលៃលើ Algorithms តាមពេលវេលាអនុវត្ត រឺកំរិតស្មុកស្មាញដោយប្រើគោលការណ៍កំនត់ចេញនូវតំលៃប្រហាក់ប្រហែល តាងដោយនិមិត្តសញ្ញា O( ) ។ Big.oh និមិត្តសញ្ញា O( ) ត្រូវបានតាងដើម្បីធ្វើការវាស់ពេលវេលាអនុវត្តន៏របស់ Algorithms តាមរយៈ function មួយចំនួនអាស្រ័យដោយទំហំ Data កំនត់ដោយ parameter n ហើយ function តែងត្រូវបានប្រើប្រាស់រួមមាន:

$\log_2 n$, n , $n \log n$, n^2 ,n^3 , 2^n

Below is the list of growth rates you will come across in the following chapters.

Time Complexity	Name	Example
1	Constant	Adding an element to the front of a linked list
$logn$	Logarithmic	Finding an element in a sorted array
n	Linear	Finding an element in an unsorted array
$nlogn$	Linear Logarithmic	Sorting n items by 'divide-and-conquer' - Mergesort
n^2	Quadratic	Shortest path between two nodes in a graph
n^3	Cubic	Matrix Multiplication
2^n	Exponential	The Towers of Hanoi problem

យើងមានតារាងតម្លៃត្រឹមត្រូវរបស់ function ទាំងនេះដូចខាងក្រោម៖

$\log_2 n$	n	$n \log_2 n$	n^2	n^3	2^n
0	1	0	1	1	2
1	2	2	4	8	4
2	4	8	16	64	16
3	8	24	64	512	256
4	16	64	256	4069	65536
5	32	160	1024	32768	429496

ឧបមាថា c ជាចំនួនថេរ , nជាទំហំទិន្នន័យ ហើយ $f(n)$ជាអនុគមន៍ពេលវេលា process របស់ Algorithms។

យើងមានទម្រង់ $c * f(n)$ +អង្គមានកម្រិតយឺតជាង

- ពេលវេលាអនុគមន៍ ឬកម្រិតស្មុគស្មាញរបស់វា

$O(f(n))$

ដើម្បីគណនាផលបូក $Sum = 1! + 2! + 3! + \cdots + n!$ គេមានAlgorithms ពីរ ក្នុងការដោះស្រាយ។ តើ Algorithm ណាមួយដែលមានលក្ខណៈប្រសើជាងគេ?

+ Algorithms ទីមួយ

```
long sum(int n)
    {
        inti,j;
        longs,p;s=0;
        for(i=1;i<=n;i++)
        {
        p=1;
        for(j=1;j<=i;j++)
            p=p*j;
        s=s+p;

        }
        return (s):
```

⇨ Statement ដែលអនុវត្តច្រើនជាងគេគឺ:

```
for(i=1;i<=n;i++)

    for(j=1;j<=i;j++)

        p=p*j;
```

- បើ i=1 Statement អនុវត្ត:1
- បើ i=2Statement អនុវត្ត:2
- បើ i=nStatement អនុវត្ត:n

- $T(n) = 1 + 2 + 3 + \cdots + n = \frac{n(n+1)}{2}$

ដូចនេះកម្រិតស្មុគស្មាញរបស់ Algorithmsគឺ $O(n^2)$

+ Algorithms ที่ดี

```
long sum(int n)
    {
     int i;
     long s=0,p=1;
     for(i=1;i<=n;i++)
          {
               p=p*i;
               s=s+p;
          }

     return (s);
```

⇨ Statement ដែលអនុវត្តច្រើនជាងគេគឺ:

```
for(i=1;i<=n;i++)

    p=p*i;
```

- ពេលវេលាអនុវត្តនៃ $T(n) = ?$

ដូចនេះកម្រិតស្មុគស្មាញរបស់ Algorithms គឺ $T(n) = O(n)$

យើងសង្កេតឃើញថា Algorithms ទី២មានលក្ខណៈប្រសើរជាងAlgorithms ទី១។


THANK
YOU