UNIVERSITI TEKNOLOGI MARA

CSC 128

TOPIC 2: BASIC ELEMENTS OF COMPUTER PROGRAM

By : MOHD SAIFULNIZAM ABU BAKAR

COURSE OUTLINE

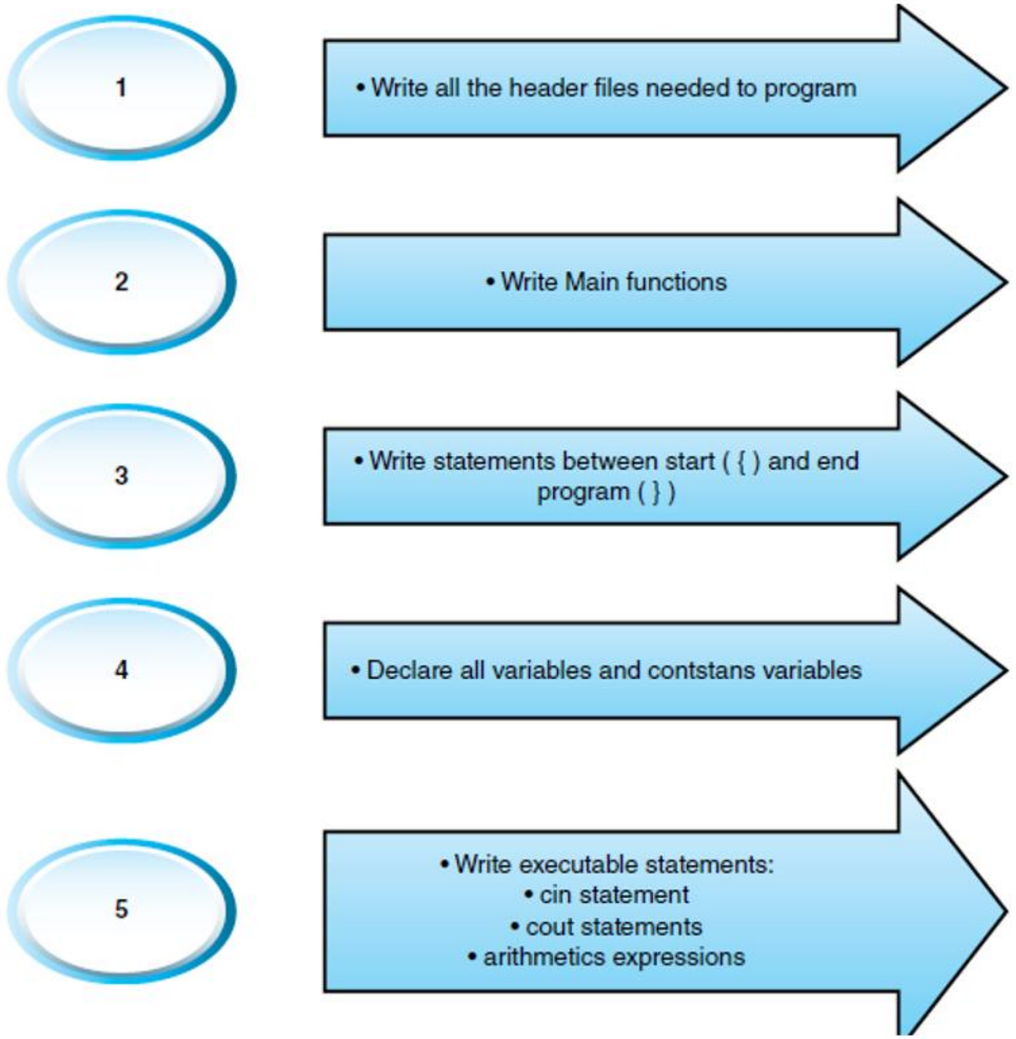
At the end of this chapter, you should be able to:

- Understand the component of a program
- Identify the basic elements needed in writing a program.
- Understand and apply basic element into a program.
- Justify the process of writing the program and error handling.
- Write a simple program.

INTRODUCTION

- In this chapter, we will discuss the basic elements required to write a program using the C++ language.
- Here, we will learn to identify these basic elements. Besides that, we will also learn how to write program code and discuss error handling.
- In the process of learning to write the basic elements in a program, we will learn to write program code for reading input, displaying output, and calculations using arithmetic operators.
- We will also discuss the steps to write the program using C++ language.

BASIC STEPS IN WRITING C++



BASIC COMPONENTS OF A C++ PROGRAM

EXAMPLE 2.2

```
/*                          : another way to represent comments
File Name                   : bmi.cpp
Programmer                  : Aiddamirah
Matrix No                   : 2014253684
Topic                       : Example 1
Program purpose             : To calculate BMI of a user
Date                        : 25 November 2014
*/

#include <iostream>         : preprocessor directives of header file
#include <math.h>           : predefined function
using namespace std;

int main()
{                           : begin block

   double weight, height, bmi;   : variable declaration

   cout << "Enter weight in kilograms: ";
   cin >> weight;

   cout << "Enter height in metres: ";
   cin >> height;

   bmi = weight/pow(height, 2);

   cout << "The BMI is: " << bmi;

   return 0;

}                           : end block
```

Comments [1]

Pre-Processor Directive [2]

Function [3]

SOURCE CODE

Actual text used to write the instructions for a computer program, and this text is then translated into something meaningful the computer can understand

BASIC COMPONENTS OF A C++ PROGRAM

EXAMPLE 2.2

```
/*                          another way to represent comments
File Name            : bmi.cpp
Programmer           : Aiddamirah
Matrix No            : 2014253684
Topic                : Example 1
Program purpose      : To calculate BMI of a user
Date                 : 25 November 2014
*/

#include <iostream>          preprocessor directives of header file
#include <math.h>            predefined function
using namespace std;

int main()
{                            begin block

   double weight, height, bmi;        variable declaration

   cout << "Enter weight in kilograms: ";
   cin >> weight;

   cout << "Enter height in metres: ";
   cin >> height;

   bmi = weight/pow(height, 2);

   cout << "The BMI is: " << bmi;      input, process and output statements

   return 0;
}                            end block
```

- Example 2.2 shows a C++ program that prompt the user to key in the user's weight and height.
- Then, the program will calculate the BMI and display the result.
- General structure of a program includes the elements used in the programming steps.

COMPONENTS OF A C++ PROGRAM

- A **comment [1]** is text writing anywhere in between program statements that is useful as an explanatory statement.
- The comment is not part of program source code, thus will be ignored by compiler in compiling process but very useful for programmers.
- Explain the purpose, parts of the program and keep notes regarding changes to the source code.

Table 2.1
Ways of writing comments

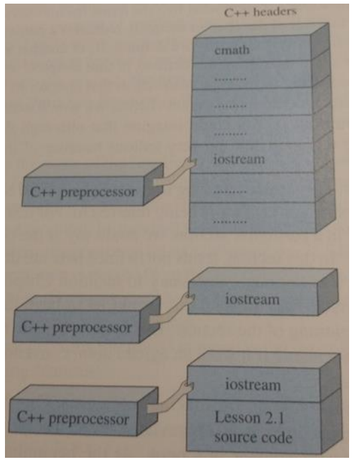
Symbols used	Purpose	Example
//	Use for one line of comment only	//This is an example of a comment. //This is the second line.
/* */	Use for one or more than one line of comments in a group	/*This is an example of a comment. This is the second line.*/

COMPONENTS OF A C++ PROGRAM

- **Pre-Processor Directive [2]** - Also known as header file in C++, thus, included at the top of the program.
- Built-in features in C++ system, automatically can be loaded into a program.
- Example:

  #include<iostream>

This cause C++ preprocessor to take code existing in a file of iostream in C++ system and group with our source code in the program. All the code soon is compiled to produce a single package of binary instruction.



- With above statement, our program is allowed to access C++ I/O(input/output) features. Thus, allow us to easily input and output to the screen using cin and cout in program body.

COMPONENTS OF A C++ PROGRAM

- There are many other header files available that are used as required by the program.
- Table 2.2 lists common header files that are frequently used in C++ programs.

Header file	Description
`iostream`	Used for standard function of input and output statements
`iomanip`	Provides parametric manipulators
`stdlib`	General utilities: memory management, program utilities, string conversions, random numbers
`string`	Used for handling string data
`math`	Used for common mathematical functions
`ctype`	Used to determine the type of character data

COMPONENTS OF A C++ PROGRAM

- Every C++ program has a primary **function [3]** that MUST assigned the name main.
- The name main is mandatory and cannot be altered.
- The compiler searches for the function named main and compiles as the first function executed!
- A complete function consist of name and body as follows:

```
3
4   int main() // the name of function
5   {//open brace indicate beginning of block of code
6
7       //the body of function main
8
9   }//close brace indicate ending of block of code
```

- Line 4 indicates the name of function which usually come with int and empty parenthesis ().
- Line 5 till 9 indicate the body of the main function that MUST begin with { and end with }.
- For the first three chapters, students will learn writing the program using main function. The extended roles of function will be explored in the chapter of user-defined function.

COMPONENTS OF A C++ PROGRAM

- A function contains a block of code that carries out specific task.
- The code itself is C++ statement, therefore a function consist a sequence of statements that perform the work of the function.
- Usually in a function has **input, output** and **operational statements.**
- Every C++ statement ends with semicolon (;).
- Example you want to write a C++ program that display welcoming message of “WELCOME TO UiTM PENANG!”.

```
1   #include<iostream>
2   using namespace std;
3
4   int main() // the name of function
5   {//open brace indicate beginning of block of code
6       cout<<"WELCOME TO UiTM PENANG!";
7
8
9
10  }//close brace indicate ending of block of code
```

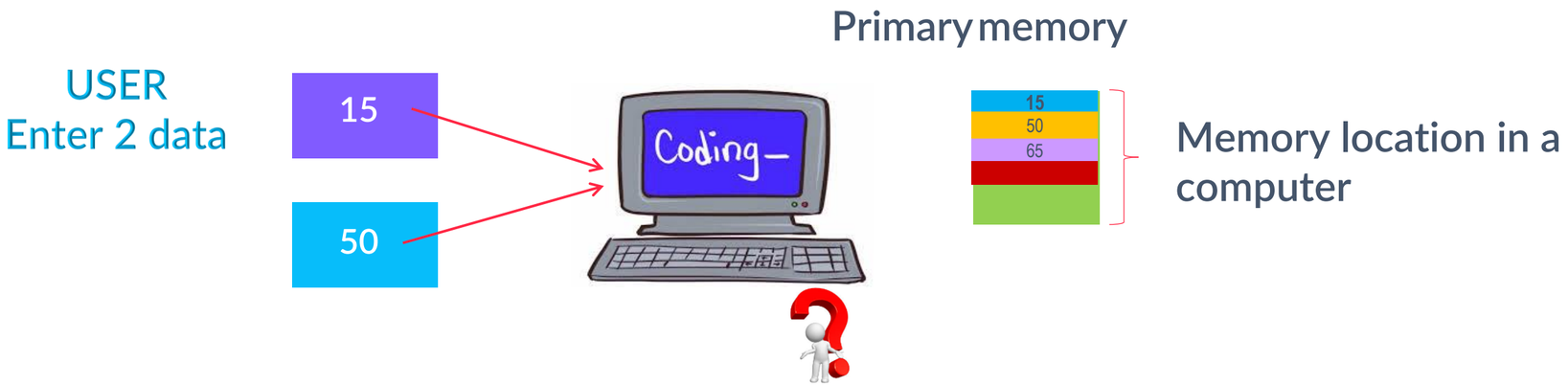
PROGRAM OUTPUT

```
WELCOME TO UiTM PENANG!
--------------------------------
```

- Line 6 indicate output statement in the function main.
- The statement outside the function will not be executed by the computer.

BASIC ELEMENT OF C++ STATEMENTS

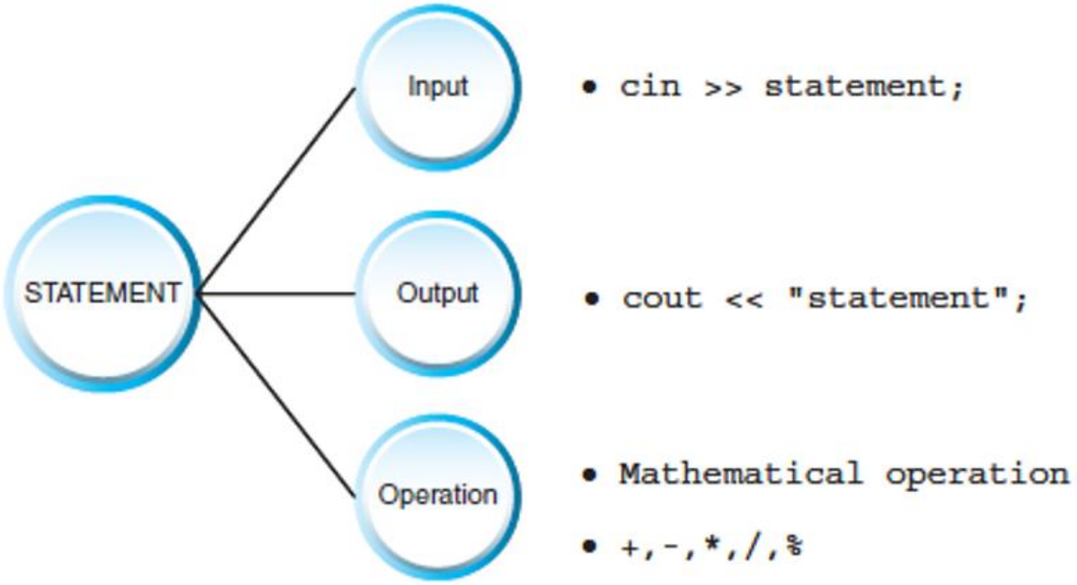
- In general, C++ statements consist of input statement using **cin**, display statement using **cout** and operational statements using group of arithmetic operations.
- Anyhow, when you are writing a program that involves data manipulation, concept of **memory location** become crucial to be understood before implementation.
- Example: You want to write a program which user will key in two numbers and your program will calculate the total.



COMPONENTS OF A C++ PROGRAM

- C++ STATEMENT
- A function consists of a sequence of statements that perform the work of the function.
- Every statement in C++ ends with a semicolon (;).

Figure 2.3
Three types of statements

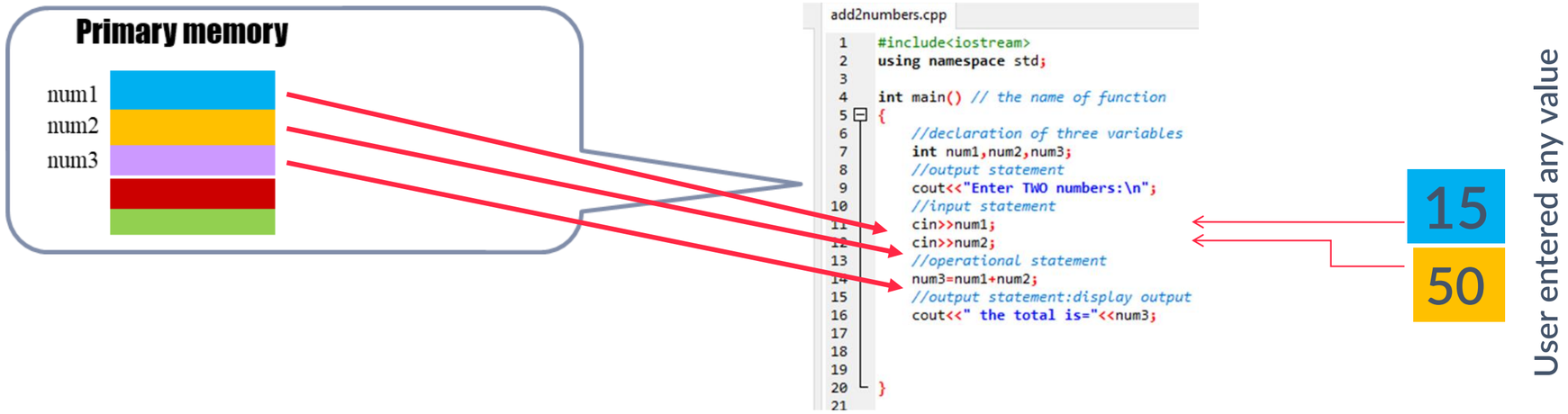


COMPONENTS OF A C++ PROGRAM

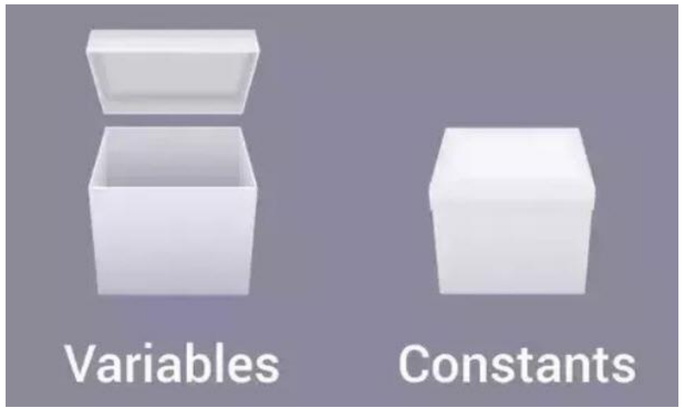
- BRACES
- Body of the main function which is enclosed in braces { }.
- Used to mark the beginning and end of blocks of code in any program.
- The open brace { is placed at the beginning of code after the main function and the close brace } is used to show the closing of code.
- The code after the } will not be read/evaluated by the compiler.

BASIC ELEMENT OF C++ STATEMENTS

- In order to integrate the program with memory location in the computer, proper declaration has to be made prior to the implementation of the data.
- Usually the declaration is made in the function. The data itself can be represent as a variable or a constant depends on the purpose of the data.



WHAT WILL COVER

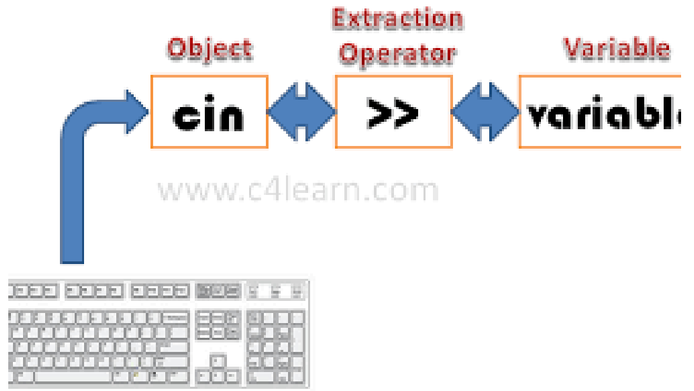


VARIABLE & CONSTANT

- Variable, constant , identifier
- Data types
- Variable declaration

OPERATIONAL STATEMENT

- Arithmetic operation
- Assignment statement
- Math library
- Compound statement



INPUT AND OUTPUT STATEMENT

- Cin / cout statement
- Data types
- Predefined output formatting

VARIABLES

- Variables can be defined as a memory allocation that will hold data and the values will **keep on changing** during program execution.
- All the data needed to solve the problem is known as input data, while the resulting information produced is known as output data.

CONSTANT VARIABLES

- Data that has a **fixed value** are declared as constant.
- For example, PI = 3.142, whereby the value of PI is fixed for any situation.
- The data of a constant is **unchangeable** throughout the program

IDENTIFIER

"An identifier is a name given (by programmer) to a variable, constant variable, function's name or label."

NAMING IDENTIFIERS

the rules to write identifier name

1. An identifier may contain letter, numbers and only the special characters underscore (_)
2. An identifier must begin with a letter or underscore (_) and may NOT begin with a number
3. An identifier may NOT contain blanks (no space between identifiers).
4. An identifier may NOT be a special/ reserved/keyword and symbols ($,&,%,*,@)
5. An identifier names in C++ can range from 1 to 255 characters

Example:
To name variable / constant

1

- contain letter, numbers and only the special characters underscore (_)

✅ priceRate1
✅ weight
✅ pass2read
✅ temperateSample3

❌ 1stPlace
❌ Pass%Percentage

2

- An identifier must begin with a letter or underscore (_) and may NOT begin with a number

✅ priceRate1
✅ Weight
✅ companyName
✅ _myHouseID

❌ 3Sample
❌ #VolumeSpehere

Example:
To name variable / constant

3

- An identifier may NOT contain blanks (no space between identifiers).

✅ discountRate
✅ schoolID
✅ monthsalary
✅ temperateSample 3

❌ Discount Rate
❌ School IDNumber
❌ month salary
❌ Temperate Sample 3

4

- An identifier may NOT be a special/ reserved/keyword and symbols ($,&,%,*,@)

✅ Item_Rate
✅ dateofbirth
✅ TEMPLATE
✅ integer

Special symbol
+ - * / ; <= & % , # @ ^ > < = { }

❌ Item-Rate
❌ date@birth
❌ template (reserved word)
❌ void (reserved word)
❌ int (reserved word)

List of Keywords and Special Symbols

Sample of keywords				
asm	default	goto	register	throw
auto	delete	if	return	try
break	do	inline	short	typedef
case	double	int	signed	union
catch	else	long	sizeof	unsigned
char	enum	new	static	virtual
class	extern	operator	struct	void
const	float	private	switch	volatile
continue	for	protected	template	while
	friend	public	this	
Special symbols				
+ - * / ; < = & % , # @ ^ > < = { }				

Table 2.3
Keywords, special words and symbols

Example:
To name variable / constant

Figure 2.5
Examples of legal and illegal identifiers

Table 2.4
Valid and invalid variable names (identifiers)

Identifier	Valid/Invalid	Explanation
`Student Number`	Invalid	Has a space between the two words
`main`	Invalid	Uses a keyword
`Pass_Percentage%`	Invalid	Has a symbol (%)
`template`	Invalid	Uses a keyword
`123place`	Invalid	Starts with a number
`_1stplace`	Valid	Starts with an underscore
`MyNumber`	Valid	No space between words

DATA TYPES IN C++

- All the variables and constants have to be declared before they are used in the program.
- To declare the variables, we have to identify the categories of data and name each uniquely.
- It is because once we declare the variable, a memory space will be provided.
- So, to allocate the size, we have to base it on the variable categories.
- Data type represents the size and type of a variable

BASIC ELEMENT OF C++ STATEMENTS

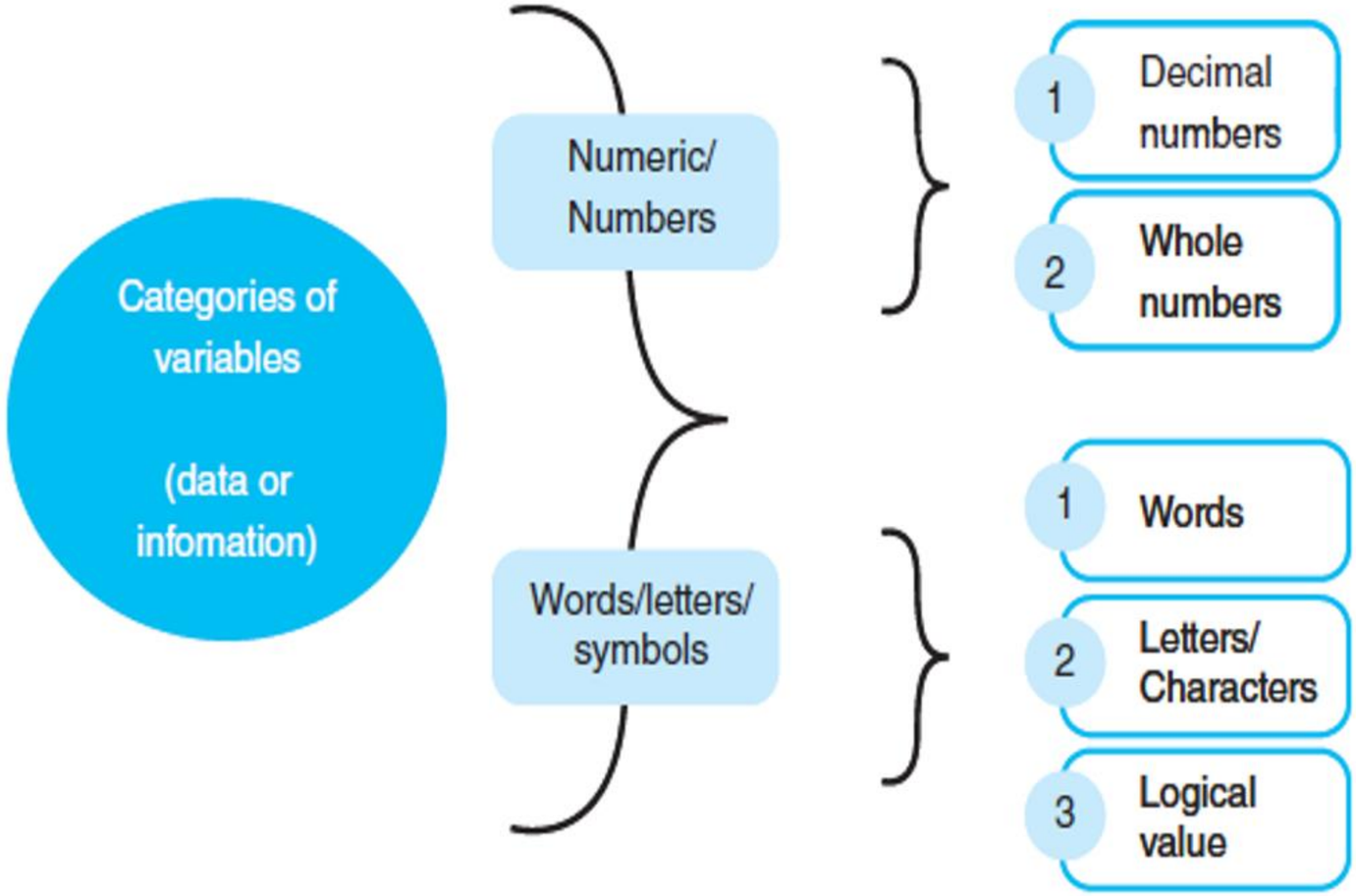


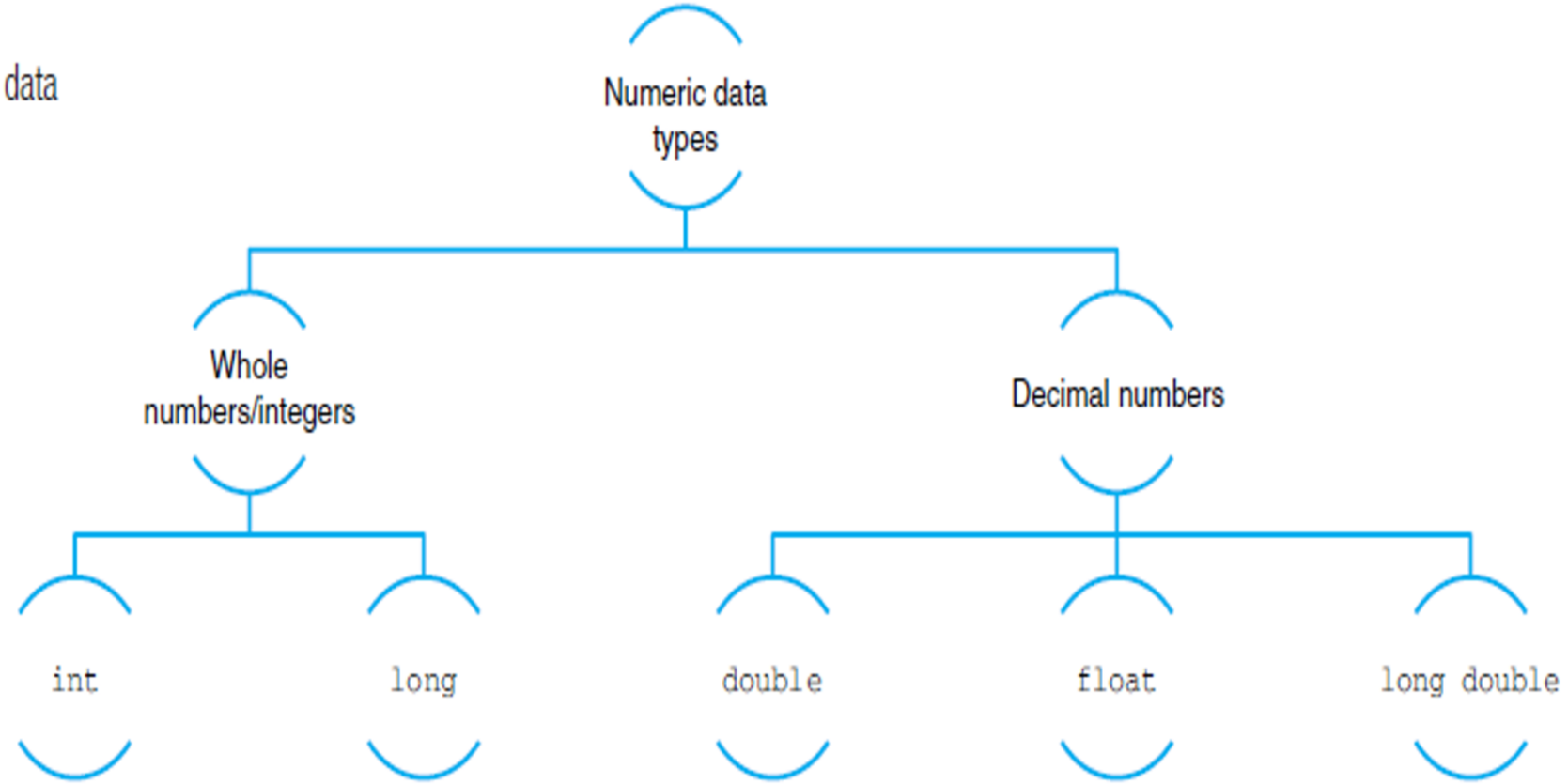
Figure 2.6
Categories of variables

DATA TYPES IN C++

- Once we declare a variable, a memory space will be provided. The allocation size of memory has to be justified and it is based on variable categories.
- Therefore, variable's declaration must enclose appropriate data type represents the size and type of a variable.
- Data type will explain the size of memory needed to hold the value of a variable. We can split them to numeric data type, character data type, string data type and logical value data type.

NUMERIC DATA TYPE

Figure 2.7
Types of numeric data



CHARACTER - DATA TYPES

- char is for variables with a single character or letter.
- char reserves a smaller size of storage for this data type.
- This char value will be enclosed in single quotation marks. The following are some char variable values:

  'A' 'b' 'G' '*' '$' '8' '12'

- Sometimes, char can also be used for holding more than individual letters.
- The statement shows that the variable gender will hold seven letters. The following is an example for this kind of char data type:

  'm', 'a', 'l', 'e'

STRING - DATA TYPES

- A string, also known as a sequence of characters, is used for variables such as name, address, object code, telephone numbers, car plate numbers and other variables that are made up of a combination of numbers, words and symbols.
- String variables are enclosed in double quotation marks.
- This string data type has a large storage size to hold the combination of values.
- The following are some examples showing the values of the string data type:
- "Eric Soh“ "AGP103“ "AP100“ "012-123125"

LOGICAL - DATA TYPES

- A variable that has a value of true or false is known as a logical value data type.
- The data type for a logical value is bool.
- The bool data type is usually used in a looping structure.
- Next slide shows example of every data type and their explanation

DESCRIPTION OF DATA TYPES

Table 2.5 Data types

Category	Data type	Explanation	Example	Size in memory (storage-byte)
Numeric	float	Variables in decimal numbers (smaller size)	Weight = 3.15 Price = 50.50 Marks = 96.5	4
	double	Variables in decimal numbers (large size)	Area_of_circle = 89.263 TotalIncome = 1500.50 TotalWaste = 15638.36	8
	int	Variables in whole numbers	Number_of_staff = 200 BookNo = 26 TotalCustomers = 150	4
	long	Variables in whole numbers (larger size)	Area = 134263985 Total_cars = 25056392 TotalBooks = 1531425	8
Character	char	Variables in single letter	code = 'A' ProgramCode = 'C'	1
		Variables in a combination of words, symbols and numbers	Gender[7] = "male" or Gender[7] = {'m', 'a', 'l', 'e'}	>1
String	string	Variable in a combination of symbols and numbers	Name = "Amin" Address = "Block A 3-5-8, My Condo" StaffId = "LEC1253"	>8
Logical value	bool	Variable with a value of either true or false	ans = true	2

EXAMPLE

- Define a correct data type for the following variables

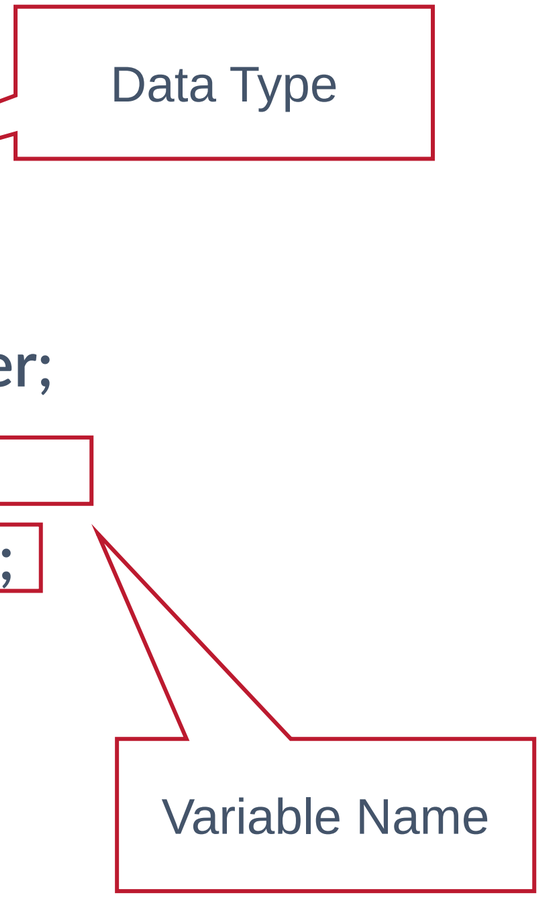
1. Number of students - int (integer-whole number)
2. Customer discount rate – double / float (decimal)
3. Book name - string (word)
4. Book category code – char (character)

VARIABLE AND CONSTANT DECLARATION

- The process of clarifying the variable name and data type of a variable is known as variable declaration.
- The declaration is a C++ statement, so it should end with a semicolon (;).
- In the declaration statement, we should write the data type, followed by the variable's name.
- The general way to write a variable declaration is as follows:

```
dataType variable_name ;
```

VARIABLE DECLARATION EXAMPLE



NUMERIC

- int age;
- int studentNumber;
- double totalPrice;
- float temperature;

NON NUMERIC

- char itemCode;
- char code [6];
- bool status;
- string name= “ ”;

VARIABLE DECLARATION

Table 2.6
Variable declarations and their descriptions

Declaration	Description
`int no_of_customers;`	Makes a variable named `no_of_customers` of data type `int`
`long List_numberOfBookSales;`	Makes a variable named `List_numberOfBookSales` that is of type `long`
`char bookCategory = 'S';`	Makes a variable named `bookCategory` of `char` data type and stores a single character 'S' in it
`double average_testmark;`	Makes a variable named `average_testmark` of data type `double`
`char gender[7];`	Makes a variable named `gender` of data type `char` that has 7 as its size
`string bookName;`	Makes a variable named `bookName` of data type `string` that will receive more than one character
`string Address;`	Makes a variable named `Address` of data type `string` that will receive more than one character or symbol
`bool final = false;`	Makes a variable named `final` with data type `bool` and assigns 'false' to it.

VARIABLE DECLARATION FOR SAME DATA TYPES

NUMERIC

```
int age;
int studentNumber;
Int adult;
```

```
int age, studentNumber , adult ;
```

Separated by coma

```
double totalPrice;
float temperature;
```

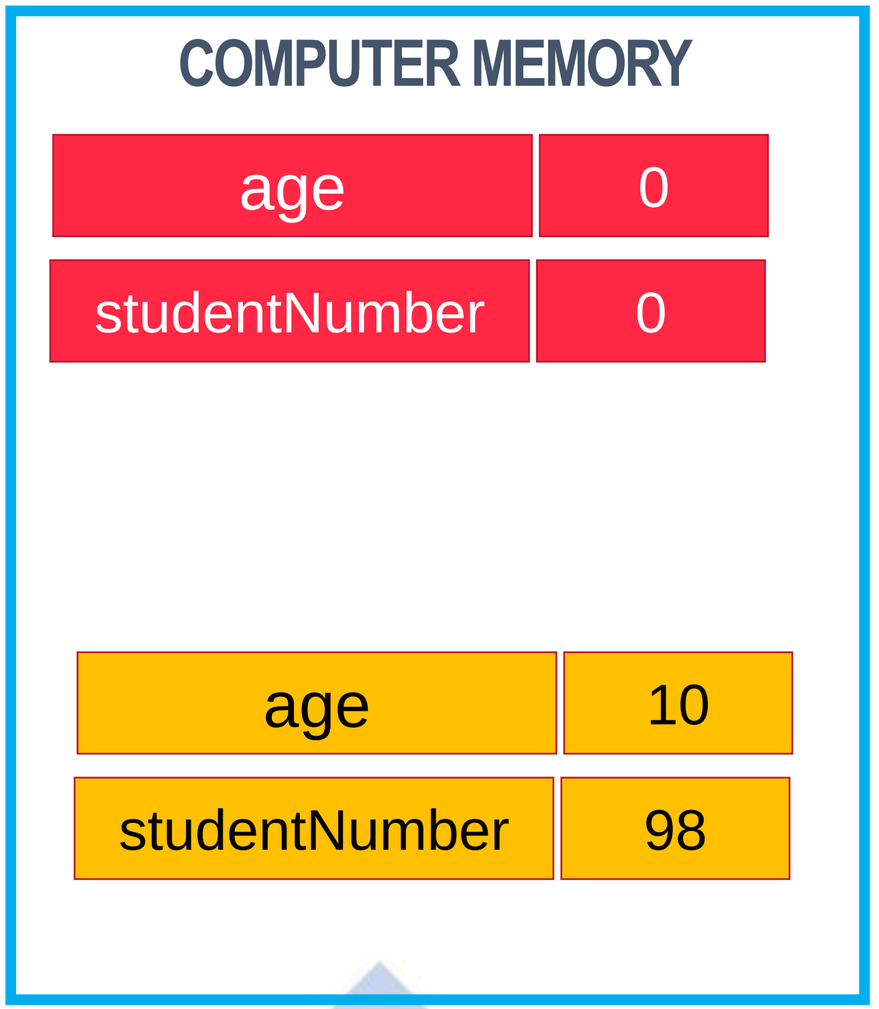
```
double totalPrice ,  temperature;
```

Separated by coma

VARIABLE DECLARATION & ASSIGN

DECLARE ONLY

- int age;
- int studentNumber;



DECLARE AND ASSIGN VALUE

- int age = 10;
- int studentNumber=98;

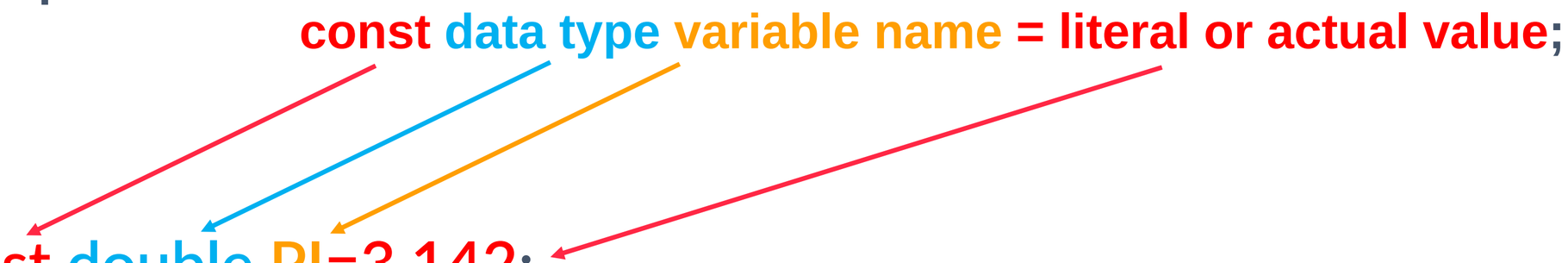
CONSTANT DECLARATION

- Constant variables must also be declared before they can be used.
- A constant variable is a variable that has a value that never changes.
- To declare the constant variable, we will use the literal value, which means that we are using the constant value. A literal value is any fixed or constant value used in a program.
- The syntax of a declaration statement for a constant variable is:

```
const data type variable name = literal or actual value;
```

CONSTANT DECLARATION

- Examples:




```
const double PI=3.142;
const double GRAVITY=9.8;
const char sizeCode= 'M';
const string companyName= "ABC  Interprise";
```

ARITHMETIC EXPRESSIONS

ARITHMETIC EXPRESSIONS

- Arithmetic expressions are the mathematical formula used to solve the problem statement.
- Usually, in solving a problem, we will do some calculations or evaluations to find the solution.
- In C++ programming, there are a few operators defined in the arithmetic expressions.
- The operators in C++ expressions are similar to normal mathematical expressions.
- The difference between the two expressions is the symbols used to represent them.
- Operators represent the symbols for instructions or commands that have to be performed.

ARITHMETIC EXPRESSIONS

- There are five basic operators used in C++ expressions and there are:

Operation	MATH operator	C++ operator	Example
addition	+	+	c=a+b;
subtraction	-	-	c=a-b;
multiplication	x	*	c=a*b;
division	÷	/	c=a/b;
modulus (return remainder)	mod	%	c=a%b;

ARITHMETIC EXPRESSIONS

- Example:

Process	Symbol/Operator	Example
Addition	+	• 40.5 + 2.2 = 42.7 • 28 + 5 = 33
Subtraction	–	• 60 - 30 = 30 • 50.2 - 20 = 30.2
Multiplication	*	• 7 * 3 = 21 • 10.5 * 3 = 31.5
Division	/	• 7 / 3 = 2 • 10.0 / 3 = 0.3333
Modulus (remainder)	%	• 29 % 9 = 2 • *Never use modulus with floating-point values.*

ARITHMETIC EXPRESSIONS

- We have to convert the mathematical formula or algebraic expression to the arithmetic expression used in programming.
- Table 2.6 provides samples of algebraic expressions converted to arithmetic expressions.

Algebraic Expression	Arithmetic Expression
$14 - 10 + 63$	16 – 10 + 63
$56 - \frac{9}{2}$	56 – 9 /2
$\frac{16 + 3 + 85}{2}$	(16 + 3 + 85) / 2
5^3	5*5*5

Table 2.6
Conversion of algebraic expressions to arithmetic expressions

ARITHMETIC EXPRESSIONS

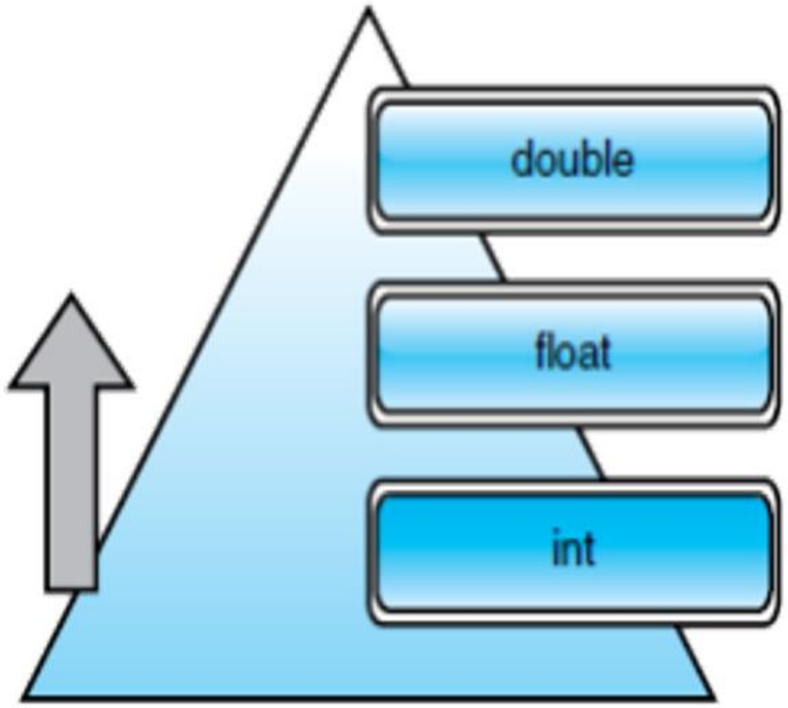
Algebraic expression	C++ arithmetic expression
$14 - 10 + 63$	`14 - 10 + 63`
$56 - \frac{9}{2}$	`56 - 9 / 2`
$\frac{16 + 3 + 85}{2}$	`(16 + 3 + 85) / 2`
5^3	`5 * 5 * 5` OR `pow(5, 3)`
$2x^2$	`2 * x * x` OR `2 * pow(x, 2)`
$2xb - d$	`(2 * x * b) - d`

We have to convert the **mathematical formula or algebraic expression** to the **arithmetic expression** used in programming.

Table 2.8
Conversion of algebraic expressions to C++ arithmetic expressions

ARITHMETIC EXPRESSIONS

- In situations where mixed numeric types appear in an expression, the computer will replace all variables with copies of the highest precision type.
- Here, the solution will be solve based on a hierarchy of operations and there is also a need for a hierarchy of types.



The hierarchy starts from integer, to float, and then to double.

ARITHMETIC EXPRESSIONS

Expression	Output
11 + 36	47
18 - 2	16
5 * 6	30
40 / 8	5
21 / 2	10
35 / 4	8
8 % 2	0
12 % 5	2

Table 2.9
Examples of arithmetic expressions

MIXED TYPE EXPRESSION

- A mixed-type expression means that different data type values are being calculated.
- For example, to solve (3/6.0), the computer will promote the lower operand in the hierarchy to the higher type before the calculation is carried out.
- The expression (3/6.0) is converted to (3.0/6.0) before the division is performed. So the answer for the division is 0.5.
- Sometimes, type changes can be made explicit by placing the value to be changed in parentheses and placing the name of the new type before it. This process is called type casting or type conversion.

MIXED TYPE EXPRESSION

- Given:

```
int X;    double Y;
Y = 3.4; X = 12;

Y = int (Y) + X ;
So, answer is Y = 15
Y = X + Y ;
So, answer is Y = 15.4
```

ASSIGNMENT STATEMENT

- An assignment operator (=) is an operator that shows where the value is assigned to.
- If given x=5, it means that value 5 is stored by variable x.
- Every variable in a program is given a value explicitly before any attempt is made to use it.
- It is also very important that the value assigned is of the correct type.

```
variable = expression;     // (eg: int num=5;)
variable = constant;      // (eg: const double PI=3.142;)
num1 = 45; //the value of num1 is 45
num2 = num1; //the value of num2 takes the value of num1 = 45
```

ASSIGNMENT STATEMENT

- Besides simple assignment (=), we also have a situation of (==) whereby this assignment is used to do comparison between the right and left side of assignments (==). For example:

```
if (ans == ‘y’)
{
    True statements;
}
else
{
     False statements;
}
```

ASSIGNMENT STATEMENT

- The assignment operator (=) also enables the storage of a value in memory.
- The value is stored at a location in memory that is accessed by the variable on the left-hand side of the assignment operator.

Example 1:

```
x = x+1;
//means add one to x and then assign the resulting value back to x.
```

Example 2:

```
x =3 //means value of 3 is assigned to x variable
x ==3 //means x is equal to 3
```

C++ COMPOUND ASSIGNMENTS

Operation	Symbol	Example	Equivalent to
Simple assignment	=	`y = 1;`	`y = 1;`
Addition/assignment	+=	`y += 10;`	`y = y + 10;`
Subtraction/assignment	-=	`y -= 2;`	`y = y - 2;`
Multiplication/assignment	*=	`y *= 5;`	`y = y * 5;`
Division/assignment	/=	`y /= 2;`	`y = y / 2;`
Modulus/assignment	%=	`y %= 2;`	`y = y % 2;`

Table 2.10 C++ compound assignments

Operator	Example	Meaning
+=	`total += 2`	`total = total + 2`
-=	`total -= 2`	`total = total - 2`
*=	`total *= 2`	`total = total * 2`
/=	`total /= 5`	`total = total / 5`
%=	`total %= 5`	`total = total % 5`

Table 2.11 C++ compound expressions

PRECEDENCE AND ASSOCIATIVITY

- In C++, we routinely have a few operators in a mixed expression.
- Which one should be evaluated first? Here, we have to follow the precedence rules to perform the mixed expression as shown in table 2.10.
- We have to draw the parentheses once we start to evaluate the expression.
- Precedence shows the sequence of arrangement or order of operators that should be evaluated first in mixed expression.
- Associativity is the process that specifies the order to perform which calculation first if two or more expressions have the same priority.

PRECEDENCE AND ASSOCIATIVITY

Table 2.12
Precedence of operators in C++ programming

Operator(s)	Operation	Precedence
( )	Parentheses	Evaluated first, inside-out if nested, or left to right if same level
*, /, %	Multiply, divide, modulus	Evaluated second, left to right
+, –	Add, subtract	Evaluated last, left to right

EXAMPLE 2.4

a 2 + 2 * 5
Solution:
2 + (2 * 5)
= 2 + 10 (evaluate *)
= 12 (evaluate +)

b 2 * 3 / 2 % 2
Solution: (evaluate operators from left to right)
= 6 / 2 (evaluate *)
= 3%2 (evaluate %)
= 1

c 2 – 3 + 2
Solution: (evaluate operators from left to right)
= –1 + 2 (evaluate –)
= 1 (evaluate +)

d 3 * 7 – 6 + 2 * 5 / 4 + 6
Solution:
(3 * 7) – 6 + ((2 * 5)/4) + 3
= 21 – 6 + (10 / 4) + 3 (evaluate *)
= 21 – 6 + 2 + 3 (evaluate /)
= 15 + 2 + 3 (evaluate –)
= 17 + 3 (evaluate first +)
= 20 (evaluate += result)

B	Bracket
O	Order L—R
D	Division
M	Multiplication
A	Addition
S	Substraction

INCREMENT AND DECREMENT OPERATOR

- Increment is the process of adding value to a variable.
- Decrement is the process of subtracting the value in a statement.
- Prefix is a process of performing the evaluation immediately. Example: ++a or --a means increase/decrease (evaluate) before assign the value.
- Postfix is a process of performing the expression after the evaluation. Example: a++ or a-- means increase/decrease (evaluate) after assign the value.

INCREMENT AND DECREMENT OPERATOR

Operators Name	Operator	Meaning
Increment postfix	a++	a=a+1
Increment prefix	++a	a=a+1
Decrement postfix	a--	a=a–1
Decrement prefix	--a	a=a–1

Table 2.11
List of incremental and decremental operators (postfix and prefix expressions)

INCREMENT AND DECREMENT OPERATOR

EXAMPLE 2.4

Increment expression:

i. Given a=4
a++
it means: a=a+1
= 4 + 1 (replace the a value, 4, with it)
= 5 (evaluate +)

ii. Given a=4
++a
it means: a=a+1
= 4 + 1 (replace the a value, 4, with it)
= 5 (evaluate +)

Decrement expression:

1. Given a=4
a--
it means: a=a-1
= 4 -1 (replace the a value, 4, with it)
=3 (evaluate -)

2. Given a=4
--a
it means: a=a-1
= 4- 1 (replace the a value, 4, with it)
= 3 (evaluate -)

EXAMPLE 2.5

Decrement expression

a Given `x = 6`
`x--`
it means: `x = x - 1`
`= 6 - 1` (replace x with the value 6)
`= 5` (evaluate –)

b Given `x = 6`
`x--`
it means: `x = x - 1`
`= 6 - 1` (replace x with the value 6)
`= 5` (evaluate –)

INCREMENT AND DECREMENT OPERATOR

```
int main()
{
    int a=7,b=8;

    cout<< a<<" "<<b<<endl;      // 7  8
    //prefix - immediately change
    cout<<++a<<" "<<++b<<endl;  // 8  9
    cout<< a<<" "<<b<<endl;      // 8 9
    //postfix - change next statement
    cout<<a++<<" "<<b++<<endl;  // 8 9
    cout<< a<<" "<<b<<endl;      //9  10

}
```

```
7 8
8 9
8 9
8 9
9 10
```

INCREMENT AND DECREMENT OPERATOR

```
int main(){
    //declaration
    int a=1, b=2, c=3;
    //------------------ operational statements
    a--; b++; c--;
    ++a;--b;++c;
    a--;b++;c--;
    cout<< "a=" <<a<<" b=" <<b<<" c=" <<c<<endl;
    cout<< a+2<<endl;
    cout<<--b<<endl;
    cout<<c--<<endl;
    cout<<c<<endl;
}
```

```
a=0 b=3 c=2
2
2
2
1
```

INCREMENT AND DECREMENT OPERATOR

EXAMPLE 2.7

```
//Program to demonstrate the increment expression using
//postfix and prefix operators.

#include <iostream> //header file
using namespace std;

//main function
int main()
{  //program starts

   //declarations
   int a = 4, b = 5;

   //display statements
   cout << a++ << " " << b << endl;   //postfix
   cout << ++a << " " << ++b << endl; //prefix
   cout << a << " " << b << endl;

   return 0;
}  //program ends
```

```
OUTPUT:
4 5
6 6
6 6
```

EXAMPLE 2.8

```
//Program to demonstrate the increment expression using
//postfix and prefix operators.

#include <iostream> //header file
using namespace std;

//main function
int main()
{   //program starts

   //declarations
   int a = 4, b = 5;

   //display statements
   cout << a++ << " " << b-- << endl; //postfix
   a = a++;
   cout << a << " " << ++b << endl;   //prefix
   b = b--;
   cout << a << " " << b << endl;

   return 0;
}   //program ends
```

```
OUTPUT:
4 5
5 5
5 5
```

UNARY AND BINARY OPERATORS

- Unary operators are operators that operate on a single operand while binary operators are operators that operate on two operands.
- some examples of unary and binary operators.

Unary operators		Binary operators	
++	increment operator	+	addition operator
−−	decrement operator	−	subtraction operator
&	address of operator	*	multiplication operator

Unary operators		Binary operators	
−	unary minus operator	/	division operator
~	negation operator (one's complement)	==	equality comparison operator
!	logical NOT	<	less than operator

MATHEMATICAL LIBRARY FUNCTIONS

- Besides the five basic operators in arithmetic expression, we have other mathematical operators that have been defined in the mathematical library function.
- For example, algebraic expression X^2, whereby it means x power of 2. We are unable to write the expression in C++ programming. To use those operators, we have to refer to the maths library function.
- While writing the program, we need to include the math.h header file that will allow the computer to refer to the library.

MATHEMATICAL LIBRARY FUNCTIONS

Function	Operation
`sqrt(x)`	Returns the square root of the argument `x`
`pow(x, y)`	Returns `x` raised to the power of `y`
`pow10(y)`	Returns 10 raised to the power of `y`
`sin( )`	Returns the sine of argument (radians)
`hypot(a, b)`	Returns the hypotenuse of a right triangle with sides `a` and `b`
`tan( )`	Returns the tangent of the argument (radians)
`log( )`	Returns the natural log of the argument
`log10( )`	Returns the base 10 log of the argument
`abs( )`	Returns the absolute value of the argument

MATHEMATICAL LIBRARY FUNCTIONS

EXAMPLE 2.9

```
//Program using functions in the math.h library

#include <iostream> //header files
#include <math.h>
using namespace std;

//main function
int main( )
{   //program starts

    int number1 = 5, power;   //variable declaration and initialization
    power = pow(number1, 2);  //calculate the number raised to the power of 2

    cout << "Number = " << number1 << endl;
    cout << "The number raised to the power of 2 = " << power << endl;


    return 0;
}   //program ends
```

```
OUTPUT:
Number = 5
The number raised to the power of 2 = 25
```

math function

pow (x,y)

COMPUTER MEMORY

number1	5
power	25

MATHEMATICAL LIBRARY FUNCTIONS

EXAMPLE 2.10

```
//Program using functions in the math.h library

#include <iostream> //header files
#include <math.h>
using namespace std;

//main function
int main()
{   //program starts

    int number1 = 25, root; //declarations
    root = sqrt(number1);   //calculate the square root of the number

    cout << "Number = " << number1 << endl;
    cout << "Square root of the number = " << root << endl;

    return 0;
}   //program ends
```

```
OUTPUT:
Number = 25
Square root of the number = 5
```

math function

sqrt (x)

COMPUTER MEMORY

number1	25
root	5

MATHEMATICAL LIBRARY FUNCTIONS

EXAMPLE 2.11

```
//Program using functions in the math.h library

#include <iostream> //header files
#include <math.h>
using namespace std;

//main function
int main()
{   //program starts

    int length_a = 3, length_b = 4; //declarations

    cout << "\nLength 1 = " << length_a << endl;
    cout << "\nLength 2 = " << length_b << endl;

    cout << "\nHypotenuse = " << hypot(length_a, length_b);
                               //calculate the hypotenuse

    return 0;
}   //program ends
```

```
OUTPUT:
Length 1 = 3
Length 2 = 4
Hypotenuse = 5
```

math function

hypot (a,b)

COMPUTER MEMORY

length_a	3
length_b	4

INPUT AND OUTPUT STATEMENTS

- After the declaration steps, we will continue with reading data from the user, which is also known as the input process.
- In the input process, the value that we use is known as the input variable.
- The input process can be done in two ways.
- Firstly, by assigning the value to the variable.
- The second way is by reading the value from the user.
- Standard input stream, cin, is used to represent the input statement.

INPUT STATEMENTS

- The cin statement is used to get input from the user using the keyboard.
- The stream extraction operator, >>, is capable of handling all of the basic data types in a way that is transparent to the programmer.
- To perform the read process in a C++ statement, we will use the syntax below:

```
cin>> input variable name;
```

INPUT STATEMENTS

- The cin statement is used to get input from the user using the keyboard.
- The stream extraction operator, >>, is capable of handling all of the basic data types in a way that is transparent to the programmer.
- To perform the read process in a C++ statement, we will use the syntax below:

```
cin>> input variable name;
```

Example 1:

```
i. cin>>num1;
ii. cin>>code;
```

INPUT STATEMENTS

- If we want to read more than one variable in one statement, we can apply the step below:

```
cin>>num1>>num2;
```

- Method 1 : cin one by one

```
cout<<“Please enter your height”;
cin>>height;
cout<<“Please enter your weight”;
cin>>weight;
```

- Method 2 : cin all in one

```
cout<<“Please enter your height and weight”;
cin>>height>>weight;
```

INPUT STATEMENTS

- If we want to read more than one variable in one statement, we can apply the step below:

```
cin>>num1>>num2;
```

Method 1 : cin one by one

```
cout<<“Please enter your height”;
cin>>height;
cout<<“Please enter your weight”;
cin>>weight;
```

Method 2 : cin all in one

```
cout<<“Please enter your height and weight”;
cin>>height>>weight;
```

INPUT STATEMENTS

cin>> int/double/float

```
int age; double cgpa;

  cout<<“Please enter your age”;
  cin>>age;
  cout<<“Please enter your cgpa”;
  cin>>cgpa;
```

cin>>single char

```
char raceCode;

  cout<<“Please enter your race“;
  cin>>raceCode;
```

INPUT STATEMENTS – CHAR[ ] / STRING

- Input statements for **char[]** and **string** are different compared to numeric and single character variables.
- The reading process uses the **getline** keyword which informs the compiler to accept a string value.
- The reading process for **char[]** and **string** data types can be performed using the general syntax as shown in the following example:

```
//for char with size
cin.getline(variable, length);
//for string data type
getline(cin, variable);
```

INPUT STATEMENTS – CHAR[ ]/STRING

EXAMPLE 2.13

```
//Program to read char data type

#include <iostream> //header file
using namespace std;

//main function
int main( )
{
    char student_name[10];

    //ask user to input string
    cout << "Please enter your name: ";

    //input char[] variable
    cin.getline(student_name, 10);

    return 0;
}   //program ends
```

EXAMPLE 2.14

```
//Program to read string data type

#include <iostream> //header file
using namespace std;

//main function
int main()
{
    string student_name;

    //ask user to input string
    cout << "Please enter your name: ";
    getline(cin, student_name);

    return 0;
}   //program ends
```

OUTPUT STATEMENTS

- For display instructions and output in C++, we will use the cout statement.
- For display instructions, we will write the instruction in double quote (“ ”) after the cout<< and end the statement with semicolon (;).
- The syntax of the display instructions is shown below:

```
cout<< “instruction statements”;
```

OUTPUT STATEMENTS

- For display output in C++, we will used the cout statement and place the variable name, and end the statement with a semicolon (;). The syntax of the display output is as follows:

```
cout<< variablename1<<variablename2;
```

- If we want to combine an instruction and display output in one statement, it can be written like this:

```
cout<<“instruction statements”<< variablename;
```

-

OUTPUT STATEMENTS

EXAMPLE 2.15

```
//Program to display instruction and output

#include <iostream> //header file
using namespace std;

//main function
int main()

 {
    //declare and assign values to variables
    int age;
    age = 42;
    string name = "Alan" ;

    //display string and values
    cout << "Hi, my name is " << name << "." << endl;
    cout << "I am " << age << " years old." << endl;

    return 0;
 }  //program ends
```

```
OUTPUT:
Hi, my name is Alan.
I am 42 years old.
```

EXAMPLE 2.16

```
//Program to display simple mathematical values

#include <iostream>        //header file
using namespace std;

//main function
int main()
{
   //declare and assign values to variables
   double weight1 = 60.5, weight2 = 80.5, total_weight;

   //calculate total weight
   total_weight = weight1 + weight2;

   //display values
   cout << "Weight 1: " << weight1 << " kg" << endl;
   cout << "Weight 2: " << weight2 << " kg" << endl;
   cout << "Total weight: " << total_weight << " kg";

   return 0;
}  //program ends
```

```
OUTPUT:
Weight 1: 60.5 kg
Weight 2: 80.5 kg
Total weight: 141.0 kg
```

FORMATTING OUTPUT STATEMENTS

- There are two ways to format the display.
- We can use an escape sequence and a predefine function to format the output in cout statement.
- An example of formatting an output statement is to format the output in two decimal points.
- Besides that, we can also arrange the output to look like a table format, etc.

FORMATTING OUTPUT STATEMENTS

- We can perform simple formatting in our cout statements. The table below shows the list of escape sequence formatting available in C++. To use the escape sequence, we have to write the formatting in between the double quotation marks in the cout statement.
- The general syntax to apply the escape sequence is as follows:

```
cout << "instruction statements (escape sequence)";
```

FORMATTING OUTPUT STATEMENTS

Escape sequence	Meaning
\n	new line—go to the next line
\t	horizontal tab
\v	vertical tab
\b	backspace
\a	alert/beep
\\	print a backslash character
\?	print a question mark character
\"	print a double quotation mark character
\'	print a single quotation mark character

FORMATTING OUTPUT STATEMENTS

EXAMPLE 2.17

```
//Program to format the display output using \n

#include <iostream>        //header file
using namespace std;

//main function
int main()
{
   //display output
   cout << "Hi. Good morning.\n";
   cout << "How are you?";
    return 0;
 }  //program ends
```

```
OUTPUT:
Hi. Good morning.
How are you?
```

EXAMPLE 2.18

```
//Program to format the display output using endl

#include <iostream>        //header file
using namespace std;

//main function
int main( )
{
   //display output
   cout << "Hi. Good morning." << endl;
   cout << "How are you?";

   return 0;
}  //program ends
```

```
OUTPUT:
Hi. Good morning.
How are you?
```

FORMATTING OUTPUT STATEMENTS

EXAMPLE 2.19

```
//Program to format the display output using \t and \"

#include <iostream>        //header file
using namespace std;

//main function
int main()
{
    //display output
    cout << "Hi, Adam. \t\t\"Good morning.\"";

    return 0;
}   //program ends
```

```
OUTPUT:
Hi, Adam.           "Good morning."
```

FORMATTING OUTPUT STATEMENTS

EXAMPLE 2.14

```
//Program to format to display output in newline using \t & \".

#include<iostream>//header files
using namespace std;
//main function
int main()
{     //display output
      cout<<"Hi. Amin\t\t\"Good Morning\"";
      return 0;
}//program ends
```

OUTPUT

```
Hi.Amin          "Good Morning"
```

FORMATTING OUTPUT STATEMENTS

- There are some predefined functions that can used to format the output by including the standard library iomanip header file.
- Table below shows some of the predefined functions that can be used for formatting.

Manipulator	Action
`setw(n)`	Sets field width to `n`
`setprecision(n)`	Sets floating-point precision to `n`
`setfill('n')`	Sets fill character to `n`
`ws`	Extracts white space character
`endl`	Inserts a new line in the output stream, then flushes the output stream

FORMATTING OUTPUT STATEMENTS

- setw(n) is a predefined function that will format statements in a width n.
- In other words, setw(n) sets the width. A general way to use setw(n) is as follows:

```
cout <<setw(n)<<"instruction statements";
```

FORMATTING OUTPUT STATEMENTS

- The program in Example 2.20 shows formatting the output using setw and setfill. setfill is used to fill the empty space in the width provided with the character assigned.
- To prove the setw width is correct, we can combine it with setfill as shown in Example 2.20.

EXAMPLE 2.20

```
//Program to format to display output using setw and setfill

#include <iostream>        //header files
#include <iomanip>
using namespace std;

//main function
int main()
{
   //declare variables and assign values
   int age = 21;
   string name = " Alan Wong" ;

   cout << "\n\n\n"; //create 3 empty lines of spacing
   cout << setw(20) << " NAME";
   cout << setw(20) << " AGE" << endl;
    //display a line under the heading
    cout << setw(20) << "----------";
    cout << setw(20) << "----------" << endl;

    //display values
    cout << setw(20) << setfill('*') << name;
    cout << setw(20) << age << endl;

    return 0;
}   //program ends
```

```
OUTPUT:



                NAME                 AGE
           ---------            ---------
********** Alan Wong******************21
```

FORMATTING OUTPUT STATEMENTS

EXAMPLE 2.21

```
//Program to calculate the BMI, formatted to display output
//up to 2 decimal places

#include <iostream>        //header files
#include <iomanip>
#include <math.h>
using namespace std;

//main function
int main()
{
    //declare and assign values to variables
    int age = 21;
    string name = "Alan Wong";
    double weight = 78, height = 1.8, bmi;
    bmi = weight/pow(height, 2);

    cout.setf(ios::fixed);
    cout.precision(2);
    cout << "\n\n\n"; //create 3 empty lines of spacing
    cout << setw(10) << "NAME";
    cout << setw(10) << "AGE";
    cout << setw(10) << "WEIGHT";
    cout << setw(10) << "HEIGHT";
    cout << setw(10) << "BMI" << endl;

    //display line under the heading
    cout << setw(10) << "---------";
    cout << setw(10) << "---------";
    cout << setw(10) << "---------";
    cout << setw(10) << "---------";
    cout << setw(10) << "---------" << endl;

    //display values
    cout << setw(10) << name;
    cout << setw(10) << age;
    cout << setw(10) << weight;
    cout << setw(10) << height;
    cout << setw(10) << bmi << endl;

    return 0;
}   //program ends
```

OUTPUT:

```
      NAME       AGE    WEIGHT    HEIGHT       BMI
--------- --------- --------- --------- ---------
Alan Wong         21     78.00      1.80     24.07
```

ERRORS IN C++ PROGRAMMING

- After writing a complete program, the next step is to compile and test.
- Compiling is a process of checking for errors in the program.
- There are three types of errors defined in C++ programming.
- After detecting errors, we have to debug [The process of correcting the errors] them.

ERRORS IN C++ PROGRAMMING

when we break the rules of writing a C++ program. This error will be identified after compiling the program.

SYNTAX ERRORS

01

LOGICAL ERRORS

02

when we get wrong or unexpected results. This type of error may happen if we use the wrong formula or the wrong variable in the formula.

RUNTIME ERRORS

03

can also be detected once we preview the result. This can lead to the result not being displayed or a program that will not stop running.

ERRORS IN C++ PROGRAMMING

Syntax errors

Examples:

```
cout << Area:".
int tot age;
cin << num;
```

Logical errors

Example:

```
x = 2;
y = 6;
ans = x + y;
cout << ans;
```

Output: 4

Runtime errors

Example:

```
a = 1;
b = 0;
result = a/b;
cout << "RESULT = "
     << result;
```

CONCLUSION

- The steps to write a C++ program are:
    1. Start with common line and header file,
    2. Continue with the main function.
    3. Write declaration statements.
    4. Write the executable statements that include the cin, cout and the process statements in between the open and close braces that show the limitation of the C++ program.
- Input is the process of reading data from the user into computer.
- Output is the process of producing the result from the computer.

CONCLUSION

- Variable refers to input and output data in the problem statement.
- Variable is also defined as the allocation of memory whereby the value will keep on changing during the program's execution.
- Each variable has to be declared.
- Statements in C++ should end with a semicolon (;).
- Data type refers to the category that represents the size in a memory.
- The two main categories of variables are numeric/number and word/letter or symbol.

CONCLUSION

- There are five basic operators used in the arithmetic expression, which are (+), (-), (/) ,(*) and (%).
- Precedence and associativity are used while evaluating the arithmetic expression to solve problems based on the order and priority.
- For formatting the output display, we can use the escape sequence methods or used the predefined functions in iomanip header file methods.
- There are three types of errors, which are syntax errors, run time errors and logical errors.