C++ PROGRAMMING BASICS

WEEK 2

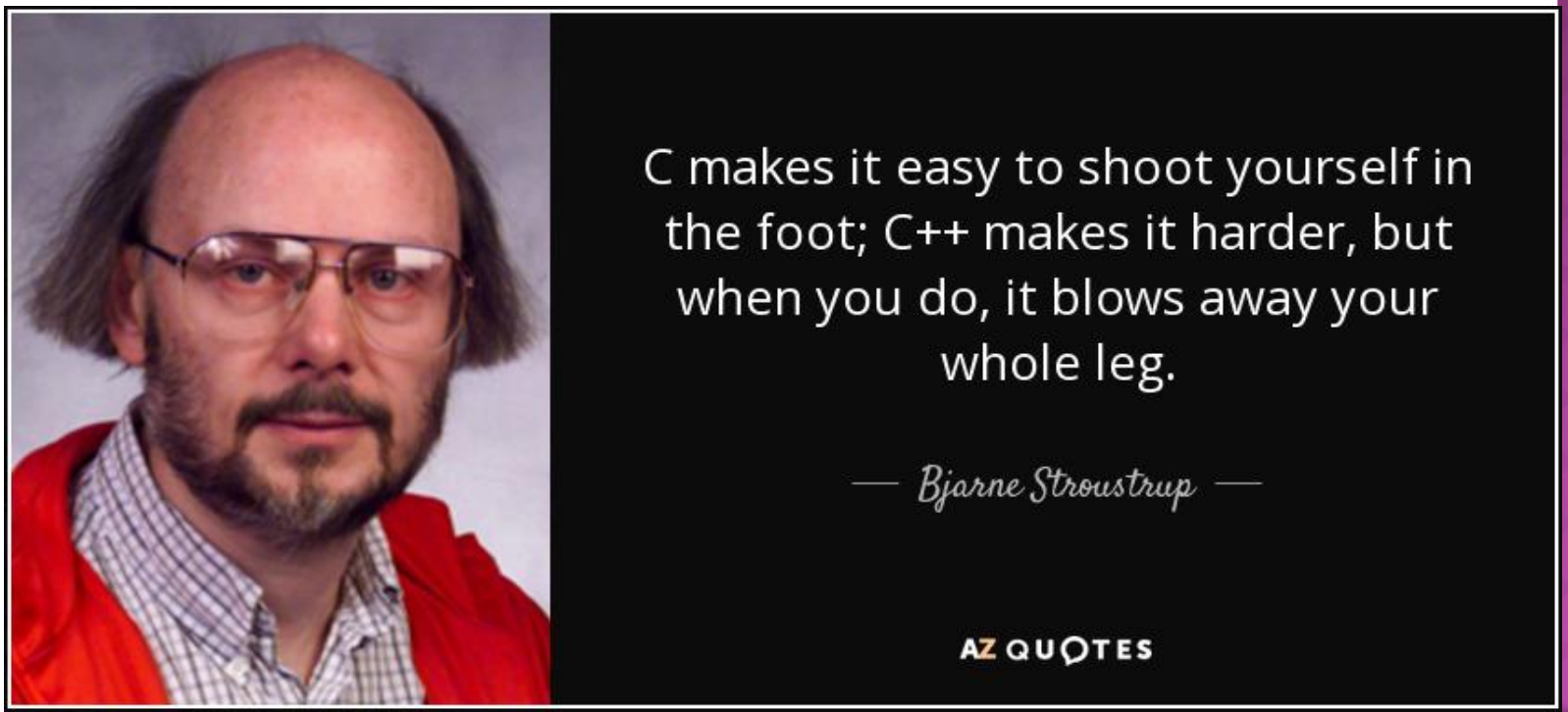
Dr. Yuzana Win

OUTLINE

- Overview of C++ Programming Language
- Variable and Data types
- Expressions and Operators
- Type Conversion

WHAT IS C++?

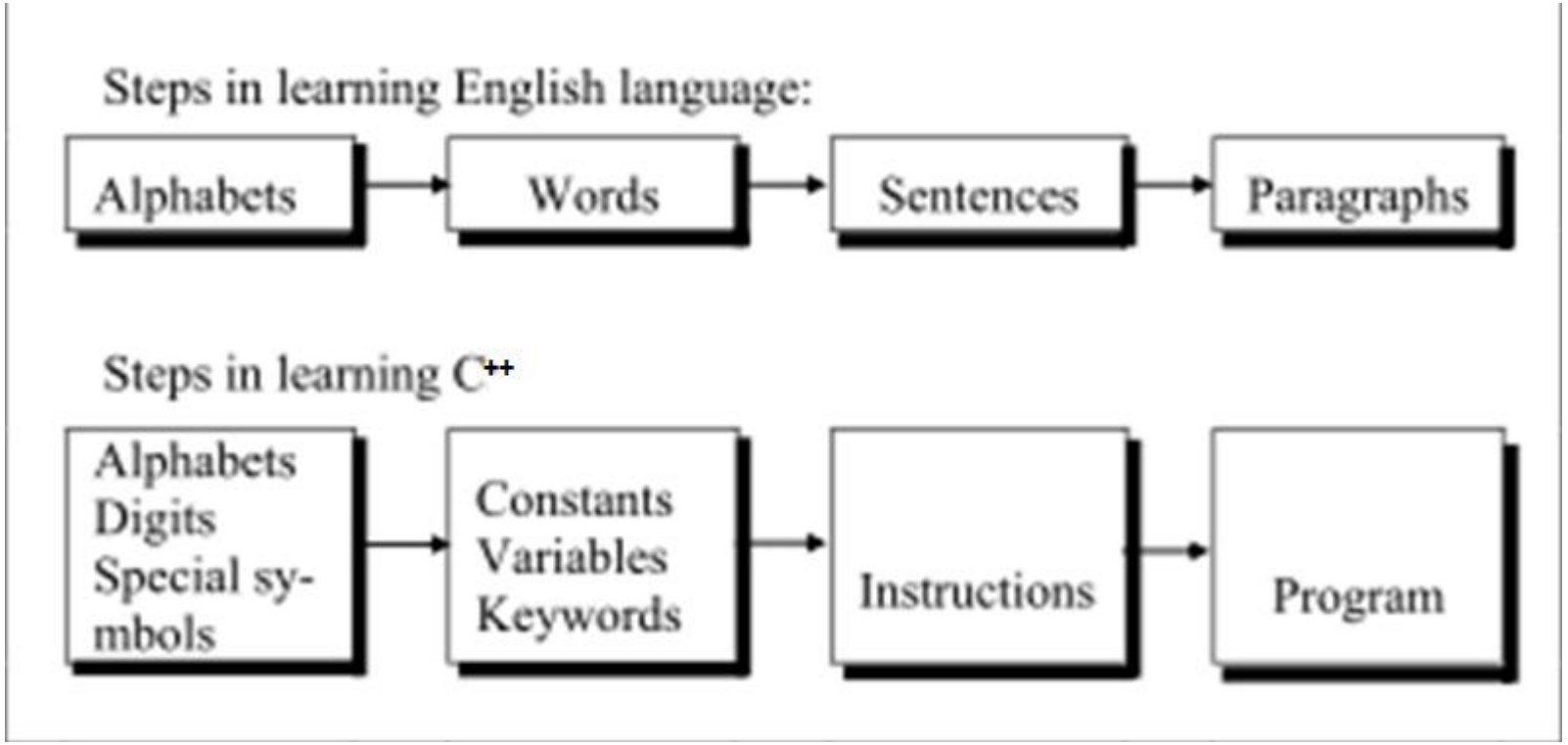
- C++ is a programming language devised by **Bjarne Stroustrup** in 1983 at Bell Laboratories.
- is an extension of C by adding some enhancements to C language.
- combines features of object oriented and the efficiency of C.
  - So, it is an extension of C language.
- **Object-Oriented**



WHY SHOULD WE LEARN C++?

- C++ is reliable, efficient and fast language
- Unix, Linux, Windows are written in C++
- Embedded and Mobile devices use C++
- Gaming frameworks are written in C++
- C++ offers better interaction with H/W

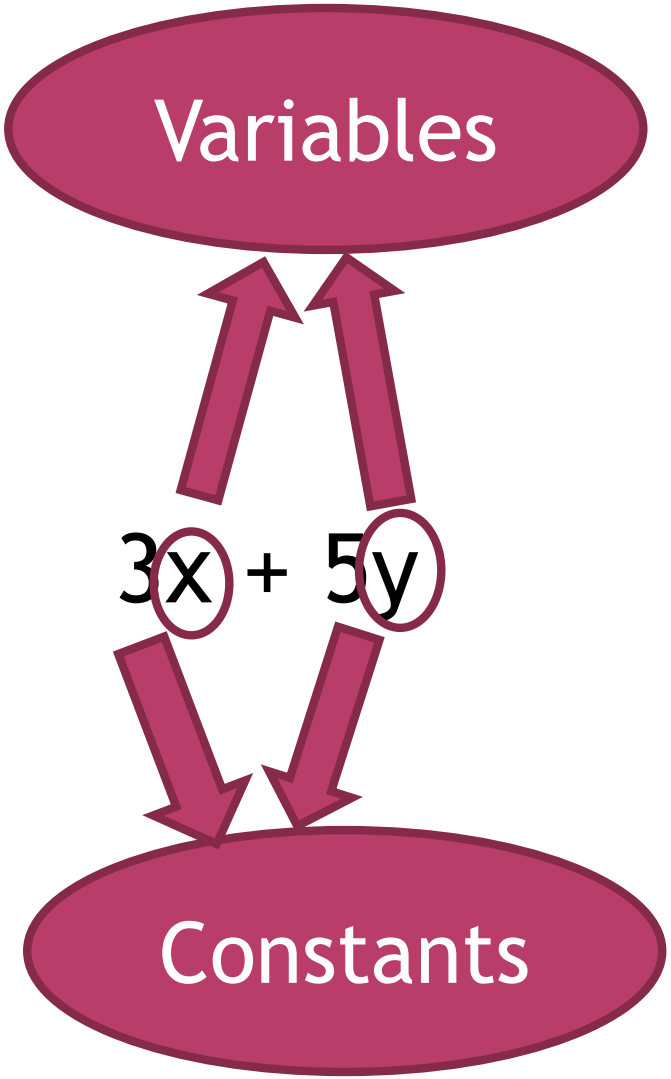
IN THE BEGINNING ----



ALPHABETS, DIGITS, SPECIAL SYMBOLS

Alphabets	A, B, ….., Y, Z a, b, ……, y, z
Digits	0, 1, 2, 3, 4, 5, 6, 7, 8, 9
Special symbols	~ ‘ ! @ # % ^ & * ( ) _ - + = | \ { } [ ] : ; " ' < > , . ? /

CONSTANTS AND VARIABLES



TYPES OF C++ CONSTANTS
1. INTEGER CONSTANT

- Example

  426 +75 -62

Rules:

1. At least one digit
2. No decimal point

29 29.0

3. Either positive or negative
   Default: positive
4. No comma or spaces

3,500 4 7 3

5. Valid Range: -32768 to 32767

2. FLOATING POINT CONSTANT

- Example

  426.5 +75.23 -62.03

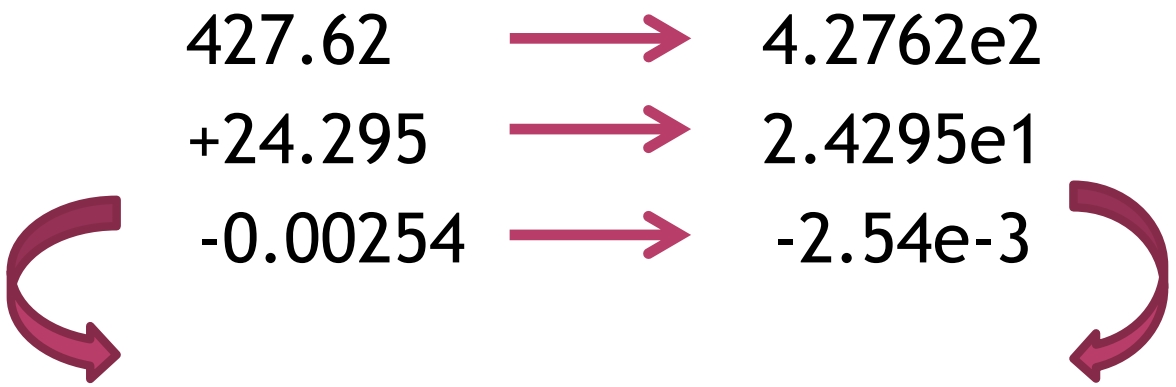
Rules:

1. At least one digit
2. Must have a decimal point
3. Either positive or negative

   Default: positive
4. No comma or spaces

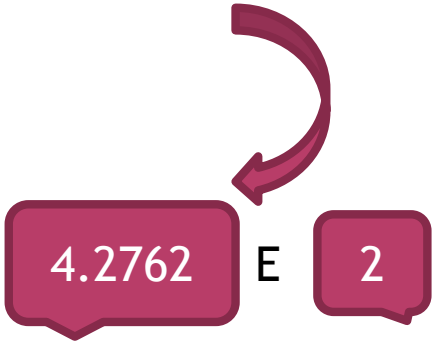
5. Valid Range: -3.4×10^{38} to 3.4×10^{38}

FORMS OF FLOATING POINT CONSTANTS



Fractional Form

Exponential Form



Mantissa

Exponent

3. CHARACTER CONSTANT

- A character constant is a single alphabet, a single digit or a single special symbol enclosed within single inverted commas.
- The maximum length of a character constant can be 1 character.

Example

‘A’ ‘I’ ‘5’

Are they OK?

’Z’ ‘name’

VARIABLE AND DATA TYPES

VARIABLES AND DATA TYPES

Variable

- A variable is an identifier that represents a value.
- The value represented by the identifier may be changed during the execution of the program.

Data and Data Types

- Data represent raw facts.
- Data type indicated the type of value represented or stored.

Data Type Defines

- It also specifies the number of bytes to be reserved in memory
- The range of value can be represented
- Type of operation to be performed on data

VARIABLES AND DATA TYPES (CONT.)

Data types may be broadly classified into two types.

- Primitive or Primary or Basic data types
- Compound or Derived or User defined data types

There are four basic primitive data types:

char - used to declare variables for storing character (textual) data.

int - used to declare variables that store integer values.

float - used to declare variables that store real (floating point) values.

double - used to declare variables that store double precision values.

VARIABLES AND DATA TYPES (CONT.)

Derived Data Types

- Derived data types are a combination of primitive data types.

- They are used to represent a collection of data. They are
  - Arrays
  - Structure
  - Unions
  - Typedef
  - Enumerated
  - Pointers

VARIABLES AND DATA TYPES (CONT.)

Size of the Data types

- Size of each data types (Machine Dependent)
  - have different ranges of values
  - require different amounts of memory

Data Type	Values	Storage (in bytes)
bool	**true** and **false**	1
int	-2147483648 to 2147483647	4
char	-128 to 127	1
short	-32,768 to 32,767	2
float	3.4×10^{-38} to 3.4×10^{38}	4
long	-2,147,483,648 to 2,147,483,648	4
double	1.7×10^{-308} to 1.7×10^{-308}	8

VARIABLES AND DATA TYPES (CONT.)

- Type Qualifiers
  - Data Types can be augmented by additional information.
  - A number of qualifiers or modifiers may be assigned to these basic types to represent the number of bits utilized in memory

- Some simple “type qualifiers” are listed below:
  - short
  - long
  - unsigned
  - signed

RULES FOR BUILDING VARIABLE NAMES

- First character must be an alphabet
- Rest can be alphabets, digits, or underscores

  Example: pop98 si_int
- Length <= 8 (Usually)
- No commas or spaces
- Variable names are case sensitive

  Example: abc ABC Abc aBc AbC

All are different

VARIABLES DECLARING & INITIALIZING

- **Declaring a Variable**

  To declare a variable in C++, do:

  data_type *variable-name*;

  Where type can be:

  - int //integer
  - double //real number
  - char //character
  - float // floating point

  Examples:

  ```
  int i,j,k;
  float x,y,z;
  char ch;
  ```

- **Initializing a Variable**

  You can initialize a variable when you declare it.

  int total_score = 0;

VARIABLES AND DATA TYPES (CONT.)

Constants

- Constant it is a bit like a variable declaration except the value cannot be changed.
- There are two types of constants.
  - **Symbolic constants**
  - **Constant variables**, also called read-only variables.
- A **symbolic** constant is defined in the preprocessor area of the program and is valid throughout the entire program.
- The preprocessor **#define** is more flexible method to define *constants* in a program. A symbolic constant is defined as follows.

```
For e.g.    #define  max    100
            #define  pi     3.14
```

- Each reference to max in program will cause the value of 100 to be applied. This value cannot be changed by the program. Because there is no memory allocated for the symbolic constants.

VARIABLES AND DATA TYPES (CONT.)

Declaring and Initializing constants

- A **constant variable** is declared and initialized in the variable declaration section of the program and cannot be modified thereafter.
- The type of value stored in the constant must also be specified in the declaration.
- In C++, we are using **const** as a keyword to declare constant variables.
- For example, an integer and float constants can be declared as follows:

```
const  int size = 100;
const  float pi=3.14;
```

VARIABLES AND DATA TYPES (CONT.)

- The **`string`** type
  - a programmer-defined type
  - requires **`#include <string>`**
- **A string is a sequence of characters**

```
"Hello"
"Mingalarpar"
"Mom"
```

C++ KEYWORDS

⇒ What are keywords?

- Words whose meaning already stands explained
- cannot be used as identifiers or variable names

⇒ What are Reserved words?

- Same as keywords

<u>Are they OK?</u>

integer a	real b	character c
int float	float = 3	char = 3.14

C++ KEYWORDS

- Common C and C++ Keywords

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

- C++ Keywords

bool	namespace	friend
public	class	virtual
protected	template	operator
private	Inline	using
new	delete	typeid

GENERAL FORM OF A C++ PROGRAM

```
// Program description
#include directives
using namespace std;
int main()
{
      constant declarations
      variable declarations
      executable statements
      return 0;
}
```

FIRST C++ PROGRAM

```
/* Calculation of simple interest */
#include <iostream>
using namespace std;
main( )
{
  int   p, n ;
  float   r, si ;

  p = 1000 ;
  n = 3 ;
  r = 8.5 ;

  /* formula for simple interest */
  si = p * n * r / 100 ;

  cout << "Simple interest is \n" << si ;
}
```

Notes:

- C++ requires a semicolon at the end of **every** statement.
- cout is a standard C++ function -- called from main.
- \n signifies new line character for formatted output.

STRUCTURE OF C++ PROGRAM

Preprocessor Area

- This section is used to include source file inclusion (#include)

Function prototype

- User defined functions prototype specification.

Global Variables Declaration

- This section is used to declare global variables

main function

- Starting and ending point of execution

User defined functions

- This section contains many user defined functions

SOURCE CODE MANAGEMENT

- Define all constants at the top of file (#define)
- Use comments to leave notes to self and other about code functionality (// or /* */)
- Make function and variable names meaningful
- One function should has 3-10 lines total, otherwise decompose into smaller functions.

PREPROCESSOR DIRECTIVES

- Commands supplied to the preprocessor
  - Runs before the compiler
  - Modifies the text of the source code before the compiler starts
- Syntax
  - start with **#** symbol
  - **#include <headerFileName>**
- Example **#include <iostream>**

HEADER FILES

- A header file is a file containing C++ declarations to be shared between several source files.
- Including a header file produces the same results as copying the header file into each source file that needs it.
- With a header file, the related declarations appear in only one place.
- The header file eliminates the labor of finding and changing all the copies as well as the risk that a failure to find one copy will result in inconsistencies within a program.
- It allows code to know about other functions provided by other code.
  - System header files declare the interfaces to parts of the operating system.
  - Own header files contain declarations for interfaces between the source files of your program.

HEADER FILES

Header File	Purpose
<iostream>	Input and output function
<stdio>	Input and output functions
<ctype>	Character testing functions
<string>	String operation functions
<math>	Mathematical Functions
<stdlib>	Number conversion, storage allocation Functions
<time>	Date and time manipulation functions

* < > indicates system header file

LIBRARY FILES

- Contains actual implementation of functions

iostream	stdio	ctype	string	math	stdlib	time
cin cout	fopen() fclose() fread() fwrite()	isdigit() islower() isupper() tolower() toupper()	strcpy() strlen() strchr() strrchr() strstr()	sqrt() pow() ceil() floor() sin() tan()	atoi() atof() atol() abs() div()	clock() time() difftime() ctime() localtime()

INPUTTING VARIABLES

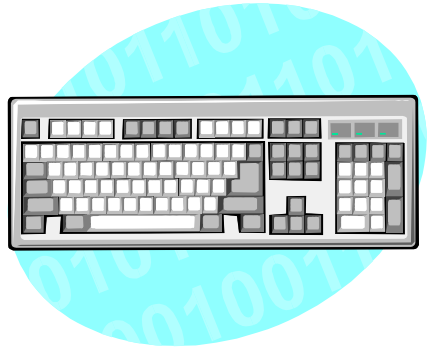
cin >> variable-name;

Meaning: read the value of the variable called <variable-name> from the user

- Storing data in the computer's memory requires two steps
  1. Allocate the memory by declaring a variable
  2. Have the program fetch a value from the input device and place it in the allocated memory location

```
cin >> a;
cin >> b >> c;
cin >> my-character;
```

123

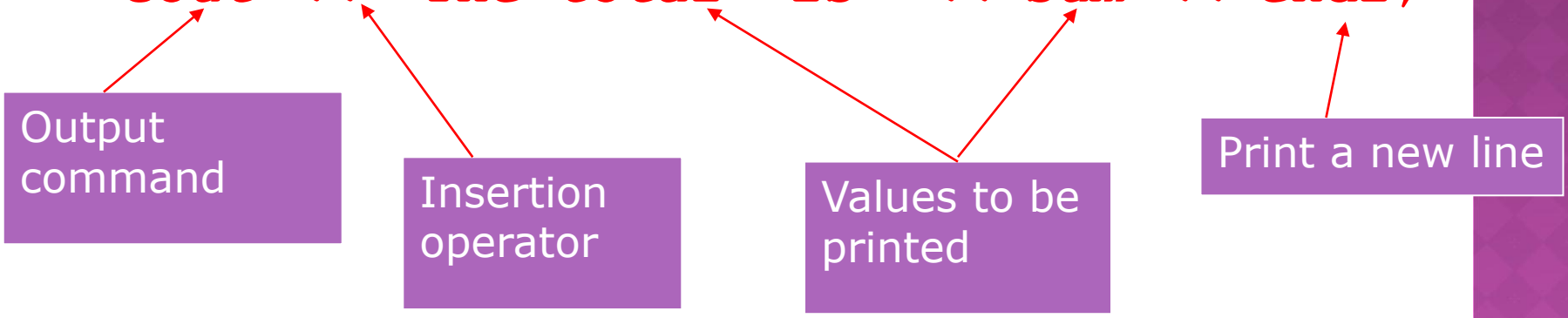


PRINTING OUTPUT

- **cout << variable-name;**

  Meaning: print the value of variable <variable-name> to the user
- Syntax for screen output:
  `cout << expression;`…
- Example

```
cout << "The total  is "<< sum << endl;
```



PRINTING A LINE OF TEXT

Escape Sequence	Description
`\n`	Newline. Position the screen cursor to the beginning of the next line.
`\t`	Horizontal tab. Move the screen cursor to the next tab stop.
`\r`	Carriage return. Position the screen cursor to the beginning of the current line; do not advance to the next line.
`\a`	Alert. Sound the system bell.
`\\`	Backslash. Used to print a backslash character.
`\"`	Double quote. Used to print a double quote character.

EXAMPLE

```
#include <iostream>
using namespace std;
int main()
{
        long pop1 = 2425785, pop2=47, pop3=9761;
        cout << “LOCATION” << “POPULATION” << endl
                << “Portcity”   << pop1     << endl
                << “Hightown” << pop2    << endl
                << “Lowville” << pop3 << endl;
        system(“pause”);
        return 0;
}
```

Output

```
LOCATION POP.
Portcity 2425785
Hightown 47
Lowville 9761
```

SETW MANIPULATOR

```
// demonstrates need for setw manipulator
#include <iostream>
#include <iomanip>
using namespace std;
int main()
{
        long pop1 = 2425785, pop2=47, pop3=9761;
      cout << setw(8) << "LOCATION" << setw(12) << "POPULATION" << endl
           << setw(8) << "Portcity" << setw(12) << pop1 << endl
           << setw(8) << "Hightown" << setw(12) << pop2 << endl
           << setw(8) << "Lowville" << setw(12) << pop3 << endl;
      system("pause");
      return 0;
}
```

Output

```
LOCATION  POPULATION
Portcity     2425785
Hightown          47
Lowville        9761
```

EXPRESSIONS AND OPERATORS

EXPRESSIONS AND OPERATORS

Expressions

- There are three types of expressions:
  - Arithmetic Expressions
  - Relational Expressions
  - Logical Expressions
- **Arithmetic Expression**
  - An expression which involves arithmetic operators, is arithmetic Expression.
  - The **arithmetic operators** are,

+	Addition
-	Subtraction
*	Multiplication
/	Division
%	Modulus

ARITHMETIC OPERATORS

```
// assign.cpp
// demonstrates arithmetic assignment operators
#include <iostream>
using namespace std;
int main()
{
        int ans = 27;
        ans += 10;              //same as: ans = ans + 10;
        cout << ans << “, ” ;
        ans -= 7;               //same as: ans = ans - 7;
        cout << ans << “, ”;
        ans *= 2;               //same as: ans = ans * 2;
        cout << ans << “, ”;
        ans /= 3;               //same as: ans = ans / 3;
        cout << ans << “, ”;
        ans %= 3;               //same as: ans = ans % 3;
        cout << ans << endl;
        return 0;
}
```

REMAINDER OPERATORS

```
// remaind.cpp
// demonstrates remainder operator
#include <iostream>
using namespace std;
int main()
{
        cout <<  6 % 8 << endl     // 6
             <<  7 % 8 << endl     // 7
             <<  8 % 8 << endl     // 0
             <<  9 % 8 << endl     // 1
             << 10 % 8 << endl;    // 2
        return 0;
}
```

EXPRESSIONS AND OPERATORS (CONT.)

- **Relational Expression**
  - An expression which involves relational operators, is Relational Expression.
  - The **Relational operators** are

<	Less than
>	Greater than
< =	Less than or equal to
> =	Greater than or equal to
= =	Equal to
! =	Not equal to

EXPRESSIONS AND OPERATORS (CONT.)

- **Logical Expression**
  - An expression which involves logical operators, is Logical Expression.
  - Logical operators are usually used with conditional statements.
  - The **logical operators** are

&&	logical AND (binary operator-requires 2 operands)
||	logical OR (binary operator-requires 2 operands)
!	logical NOT (Unary operator-requires 1 operand)

EXPRESSIONS AND OPERATORS (CONT.)

Assignment Statements

- Assignment Statement is used to assign a value to a variable. In C++, the **assignment operator** is "=".
- C++ allows multiple assignment statements using "="
- for example:

```
a = b = c = d = 3;
```

- This kind of assignment is only possible if all the variables types in the statement are the same.

EXPRESSIONS AND OPERATORS (CONT.)

Overview of C++ operators

- C++ operators fall into the following categories:
  - **Postfix** operators, which follow a single operand. (**a++**)
  - **Unary prefix** operators, which precede a single operand. (**++a**)
  - **Binary** operators, which take two operands and perform a variety of arithmetic and logical operations. (**+** , **-** , **&&**)
  - The **conditional** operator (a **ternary** operator), which takes three operands and evaluates either the second or third expression, depending on the evaluation of the first expression. (**?:**)
  - **Assignment** operators, which assign a value to a variable. (**=**)
  - The **comma** operator, which guarantees left-to-right evaluation of comma-separated expressions. (**,**)

Table 1 Overview of C++ operators

Operator	Example	Description/Meaning
()	`f()`	Function call
[]	`a[10]`	Array reference
->	`s->a`	Structure and union member selection
.	`s.a`	Structure and union member selection
+ [unary]	`+a`	Value of a
- [unary]	`-a`	Negative of a
* [unary]	`*a`	Reference to object at address a
& [unary]	`&a`	Address of a
~	`~a`	One's complement of a
++ [prefix]	`++a`	The value of a after increment
++ [postfix]	`a++`	The value of a before increment
- - [prefix]	`--a`	The value of a after decrement
- - [postfix]	`a--`	The value of a before decrement
`sizeof`	`sizeof (t1)`	Size in bytes of object with type t1
`sizeof`	`sizeof e`	Size in bytes of object having the type of expression e

Table 1 Overview of C++ operators (cont.)

Operator	Example	Description/Meaning
+ [binary]	`a + b`	a plus b
- [binary]	`a - b`	a minus b
* [binary]	`a * b`	a times b
/	`a / b`	a divided by b
%	`a % b`	Remainder of a/b
>>	`a >> b`	a, right-shifted b bits
<<	`a << b`	a, left-shifted b bits
<	`a < b`	1 if a < b; 0 otherwise
>	`a > b`	1 if a > b; 0 otherwise
<=	`a <= b`	1 if a <= b; 0 otherwise
>=	`a >= b`	1 if a >= b; 0 otherwise
==	`a == b`	1 if a equal to b; 0 otherwise
!=	`a != b`	1 if a not equal to b; 0 otherwise
& [binary]	`a & b`	Bitwise AND of a and b
|	`a | b`	Bitwise OR of a and b
^	`a ^ b`	Bitwise XOR (exclusive OR) of a and b
&&	`a && b`	Logical AND of a and b (yields 0 or 1)
||	`a || b`	Logical OR of a and b (yields 0 or 1)
!	`!a`	Logical NOT of a (yields 0 or 1)

Table 1 Overview of C operators (cont.)

Operator	Example	Description/Meaning
?:	`a ? e1 : e2`	Expression e1 if a is nonzero; Expression e2 if a is zero
=	`a = b`	a, after b is assigned to it
+=	`a += b`	a plus b (assigned to a)
-=	`a -= b`	a minus b (assigned to a)
*=	`a *= b`	a times b (assigned to a)
/=	`a /= b`	a divided by b (assigned to a)
%=	`a %= b`	Remainder of a/b (assigned to a)
>>=	`a >>= b`	a, right-shifted b bits (assigned to a)
<<=	`a <<= b`	a, left-shifted b bits (assigned to a)
&=	`a &= b`	a AND b (assigned to a)
|=	`a |= b`	a OR b (assigned to a)
^=	`a ^= b`	a XOR b (assigned to a)
,	`e1,e2`	e2 (e1 evaluated first)

EXPRESSIONS AND OPERATORS (CONT.)

Order of Precedence

- All operators have a priority, and high priority operators are evaluated before lower priority ones. Operators of the same priority are evaluated from left to right.

Precedence of C++ Operators

Category	Operator	Associativity
Postfix	() [] -> . ++ - -	Left to right
Unary	+ - ! ~ ++ - - (type) * & sizeof	Right to left
Multiplicative	* / %	Left to right
Additive	+ -	Left to right
Shift	<< >>	Left to right
Relational	< <= > >=	Left to right
Equality	== !=	Left to right
Bitwise AND	&	Left to right
Bitwise XOR	^	Left to right
Bitwise OR	|	Left to right
Logical AND	&&	Left to right
Logical OR	||	Left to right
Conditional	?:	Right to left
Assignment	= += -= *= /= %= >>= <<= &= ^= |=	Right to left
Comma	,	Left to right

TYPE CONVERSION

TYPE CONVERSION

- C++ provides a mechanism which allows the programmer to change the default data type of a given arithmetic expression. This is called typecasting.

  e.g.

```
float sum;
sum = (int) (1.5 * 3.8);
```

- The type cast (int) tells the C++ compiler to interpret the result of (1.5 * 3.8) as the integer 5, instead of 5.7. Because sum is of type float, the value stored will therefore be 5.0.

TYPE CONVERSION (CONT.)

- C++ provides two types of typecasting:
  - implicit - C++ compiler can converts the value of one data type into another by default.
  - explicit - User has to enforce the compiler to convert the one data type value to another data type value by using typecasting operator.

    [(data type) expression]

- Another two terms associated with type casting are :
  - Narrowing: Converting the higher data type value to lower data type value.(explicit typecast)
  - Widening : Converting the lower data type value to higher data type value. (implicit cast)

e.g. **float to (int or char) - narrowing**
(char or int) to float - widening

SUMMARY

- Various data types are built into C++: char, int, long, and short are the integer types and float, double, and long double are the floating-point types.
- A variable name can be of maximum 31 characters
- Do not use a keyword as a variable name
- Input/Output in C++ can be achieved using *cin* and *cout* functions.
- C++ employs the arithmetic operators +, -, * , and /.
- The remainder operator, %, returns the remainder of integer division.
- The arithmetic assignment operator +=, +-.
- The increment and decrement operators ++ and – increase or decrease a variable by 1.