

Fundamentals of Computer Programming



Chapter 1 Basic Concepts of Programming

Chere L. (M.Tech)
Lecturer, SWEG, AASTU

- Basics of Program Development
 - ✓ *What is computer programming?*
 - ✓ *Reasons to study programming*
- Introduction to Program Development Life Cycle
- An Overview of Programming Languages and Paradigms
- Compilation Process and running programs:
 - ✓ *Compiling, Debugging and Linking*
- Structure (Anatomy) of C++ program
 - ✓ *Preprocessor and library functions*
 - ✓ *main() function*
 - ✓ *Statement and Block*
- Why C++ programming language

Outline

- Basic Elements of program (Syntax and Semantics)
 - ✓ *Input/output standards*
 - ✓ *Variables and Data types*
 - ✓ *Constants*
 - ✓ *Operators and expression*
- Debugging and Programming Errors
 - ✓ *Error types*
 - ✓ *Debugging errors*
- Formatting Program
 - ✓ *Comment, braces, Indentation and White space*
- Formatted Input-Output

1. Basics of Program Development

■ Computer Program

- ✓ *Self-contained set of explicit and unambiguous instructions that tells the computer to perform certain specific tasks.*
- ✓ *It tells what to do and How to do it (the driver is a program)*
- ✓ **HOW** → the computer program should determine the operations to be carried out by the machine in order to solve a particular problem and produce a specific result.

■ Computer Programming

- ✓ Also shortened as Programming/Coding
- ✓ Skills acquired to design and develop computer programs
- ✓ The process of writing, testing, debugging, troubleshooting, and maintaining the source code of computer programs.

■ Software Development Life Cycle

- ✓ The overall process of software development

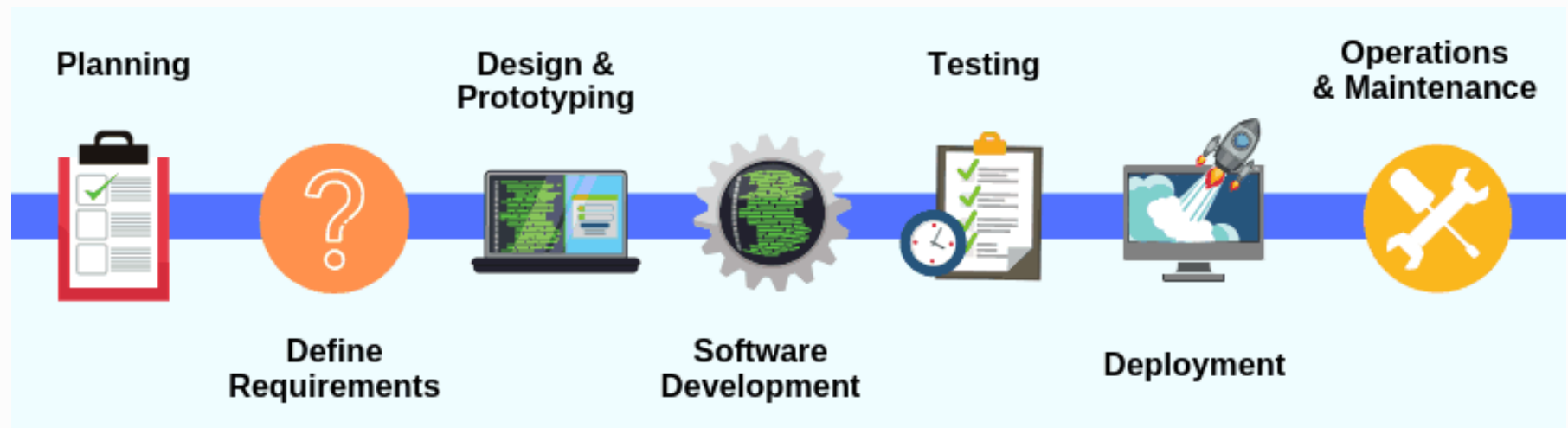
Some of the reasons to study Computer Programming

Cont. . .

- The job prospects are great
 - *We are currently living in a world where computers are found nearly everywhere and most objects are now connected to them*
 - We are living in the digital age and the growth of technology does not seem to be coming to a stop.
- Programming makes things easier for you
 - *A simple computer program is capable of turning things around as you want.*
 - *For example, if you're using a Smartphone to chat app or switch on/off your electrical appliances, if you're unlocking your car with the push of a button, then you must know that all these things are using some kind of program*
- Coding develops structured and creative or logical thinking
- Learning to program teaches you persistence
- You'll learn how to learn and become more detail-orientated

Program Development Life Cycle (1/2)

- Also referred as **Software development procedure**
- It is a conceptual model that describes the **stages** involved in a software development project from an **initial phase (planning)** **through maintenance** of the completed application
- It is **scientific method approach** and **systems approach** that used in science and engineering, and in quantitative analysis respectively



Program Development Life Cycle (2/2)



- Why SDLC? --- reading assignment

Programming Languages and Paradigms

Programming Languages

- An artificial language that can be used to control the behavior of a machine, particularly a computer.
- Like natural language (such as Amharic), are defined by syntactic and semantic rules which describe their structure and meaning respectively.
- More specifically programming languages
 - A set of different category of written symbols that instruct computer hardware to perform specified operations required by the designer.
 - It provides a linguistic framework for describing computations.
 - A set of rules for communicating an algorithm with computer systems (i.e. it provides a way of telling a computer what operations to perform).
 - A tool for developing executable models for a class of problem domains.

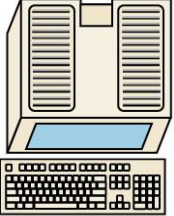

Programming Languages and Paradigms (cont'd)

Types of Programming Languages

- Basically there are two types of programming language.



Generations of Programming Languages

Generations					Human
1st	2nd	3rd	4th	5th	
Machine 	Machine Language O-1 Long, difficult programming	Assembly Language Assemble repetitive instructions, shorter code	Procedural Languages Include commands, shorter code	Nonprocedural Languages Application generators, commands specify results	Intelligent Languages Natural-language processing
Progress →					Natural Language 

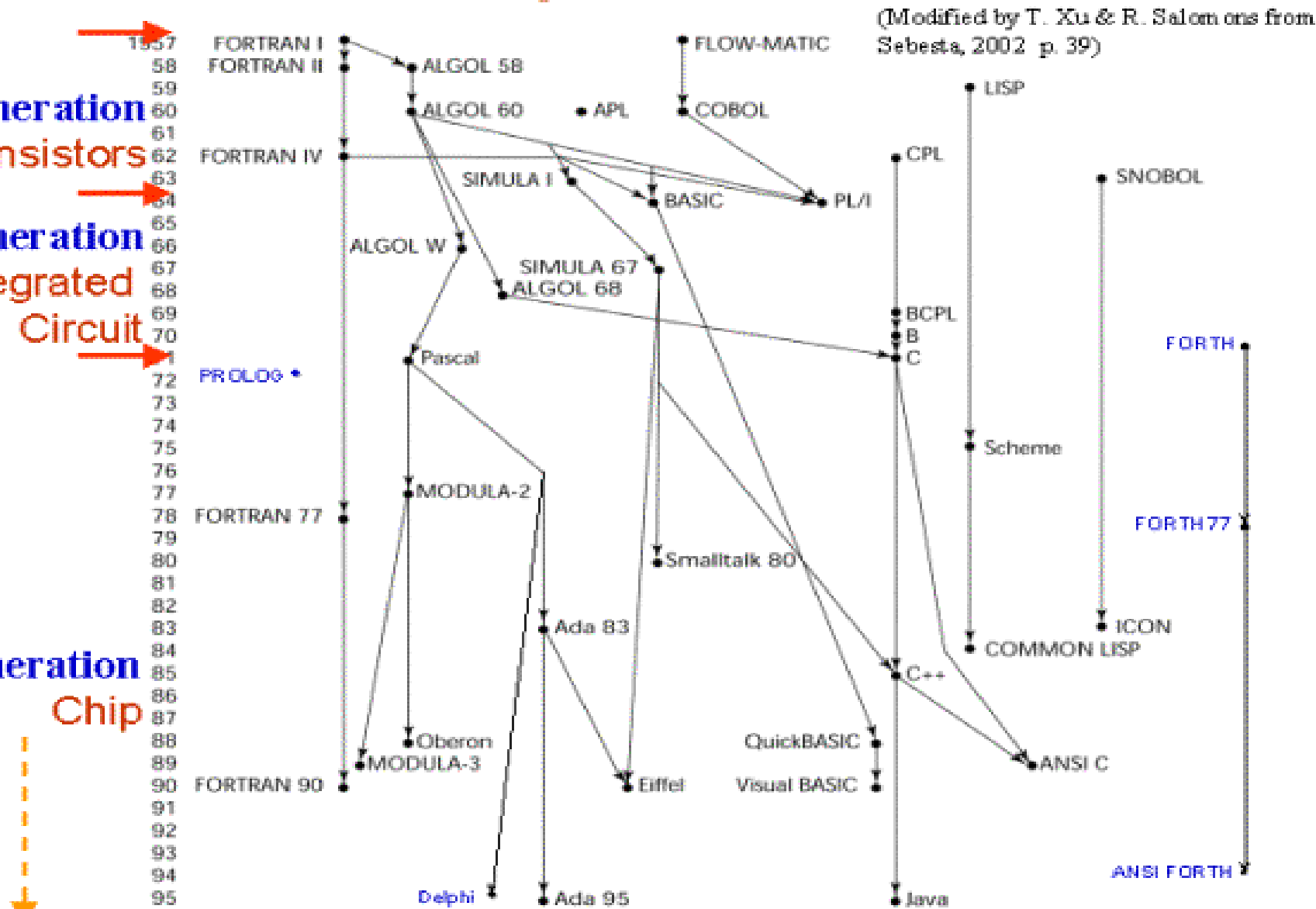
Programming Languages and Paradigms (cont'd)

1st Generation Vacuum Tubes, Magnetic drums

2nd Generation Transistors

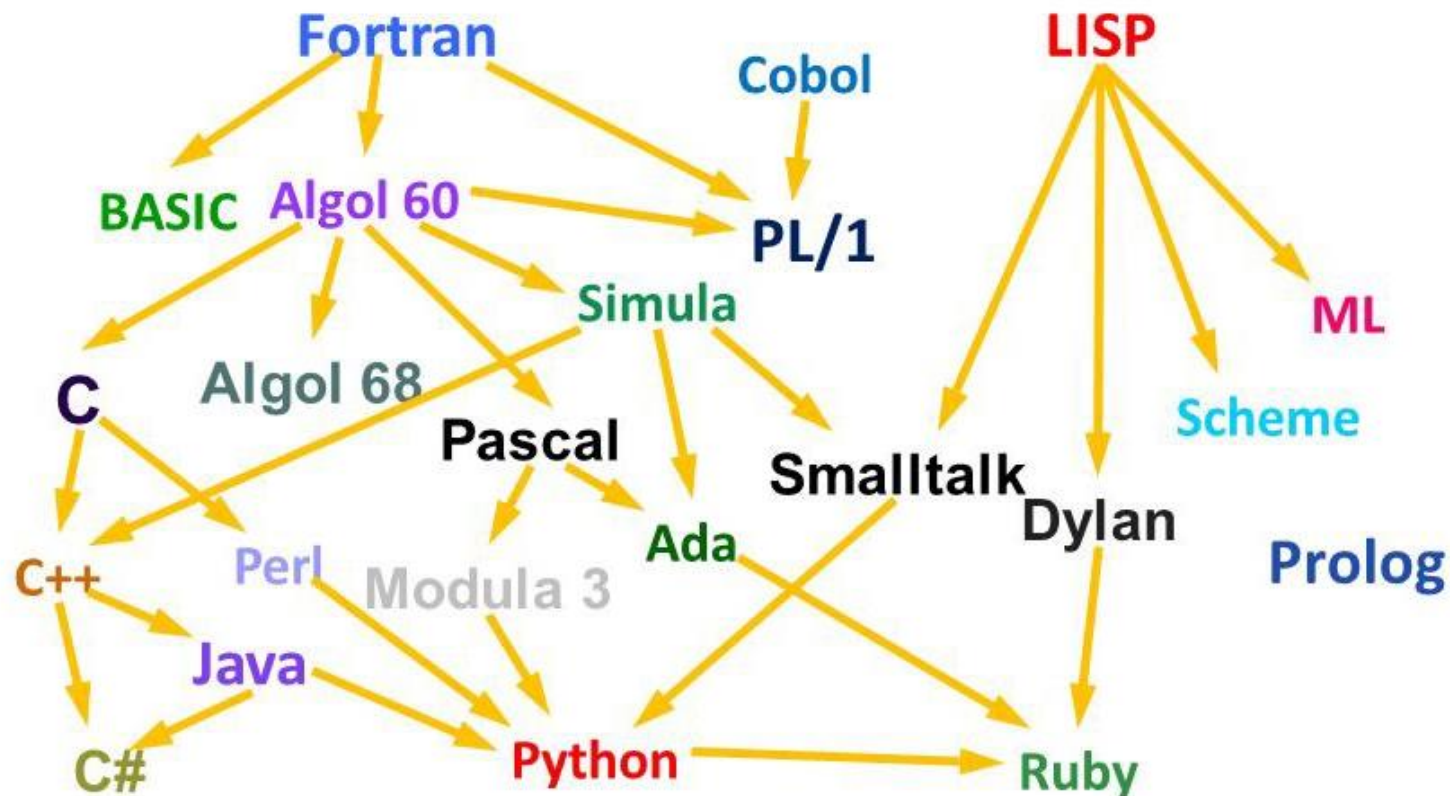
3rd Generation Integrated Circuit

4th Generation Chip



A family tree of languages

Some of the 2400 + programming languages



Programming Languages and Paradigms (cont'd)

Programming Paradigms

- A style, or “way,” of programming
- Describes the way in which a particular computer program should be written.
- Doesn't refer to specific programming language
- Different programming languages are designed to follow a particular paradigm or may be multiple paradigms.
- The major programming paradigms include the following
 1. Imperative (Procedural) Programming
 2. Logical / Declarative Programming
 3. Functional Programming
 4. Object-Oriented Programming

- <https://www.cs.ucf.edu/~leavens/ComS541Fall97/hw-pages/paradigms/major.html>
- https://www.cs.bham.ac.uk/research/projects/poplog/paradigms_lectures/lecture1.html
- <https://hackr.io/blog/programming-paradigms>

Programming Languages and Paradigms (cont'd)

■ Brief summary of programming paradigms

No	Paradigms	Descriptions		Example
1	Imperative (Procedural) Programming	Programs as statements that directly change computed state (datafields)	Program = algorithms + data - good for decomposition	C, C++, Java, PHP, Python, Ruby
2	Logical (Declarative) Programming	Defines program logic, but not detailed control flow	Program = facts + rules - good for searching	Prolog, SQL, CSS
3	Functional Programming	Treats computation as the evaluation of mathematical functions avoiding state	Program = functions . functions - good for reasoning	C++, Lisp, Python, JavaScript
4	Object- Oriented Programming	Treats data fields as objects manipulated through predefined methods only	Program = objects + messages - good for modelling	C++, C#., Java, PHP, Python

Anatomy of C++ program

```
// sample C++ program ← comment
#include <iostream> ← preprocessor directive
using namespace std; ← which namespace to use

int main() ← beginning of function named main
{ ← beginning of block for main
    cout << "Hello, there!"; ← output statement
                                ↑ string literal
    return 0; ← send 0 to operating system
} ← end of block for main
```

Output



Hello, there!

Structure of C++ Program (cont'd)

■ Statements

- ✓ Independent unit in a program, just like a sentence in the English language.
- ✓ The individual instructions of a program that performs a piece of programming action
- ✓ It is a fragments of the program that are executed in sequence
- ✓ Every C++ statement has to end with a semicolon.
- ✓ E.g. input/output statement

■ Blocks

- ✓ Also called compound statement
- ✓ A group of statements surrounded by braces { }.
- ✓ All the statements inside the block are treated as one unit.
- ✓ There is no need to put a semi-colon after the closing brace to end a complex statement
- ✓ E.ge main() function

Structure of C++ Program (cont'd)

■ main () function

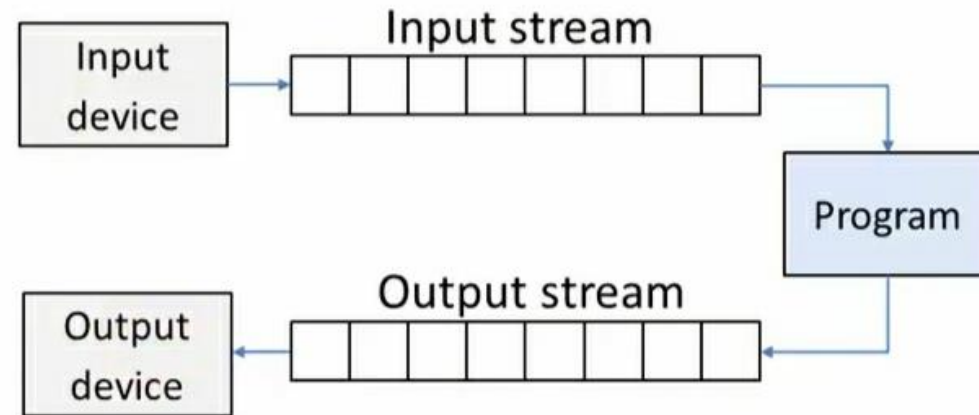
- ✓ It is where program execution starts in C++
- ✓ Every C++ must contain main () function
- ✓ The main () function can be made to return a value to the operating system (the caller)
- ✓ The return type is specified by Data type where as the return value is mentioned by with return keyword.
- ✓ The main () function must followed by block i.e. { }

Note

- ✓ Instead of using numeric value of zero and non-zero, you can also use EXIT_SUCCESS or EXIT_FAILURE, which is defined in the cstdlib header (i.e., you need to "#include <cstdlib>".

I/O Streams

- Data stream refers to the flow of data between program and its environment particularly input/output devices.
- Data stream typically exist in the form of characters or numbers

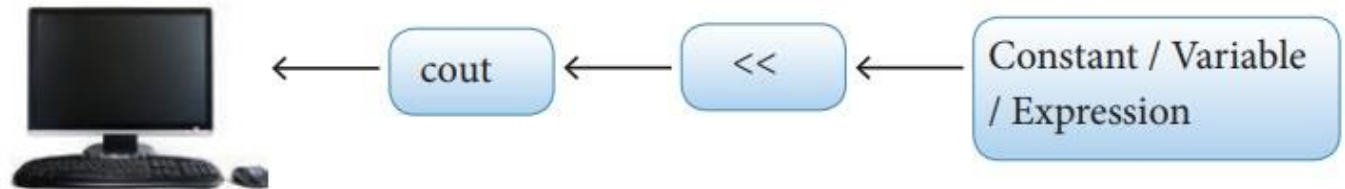


- An **input** stream is data for the program to use typically originates at the keyboard
- An **output** stream is the program's output. Destination is typically the monitor.

I/O Streams (cont'd)

■ C++ standard OUTPUT Stream (cout)

- **cout** (console **out**put) is the object to display information on the computer's screen in C++.

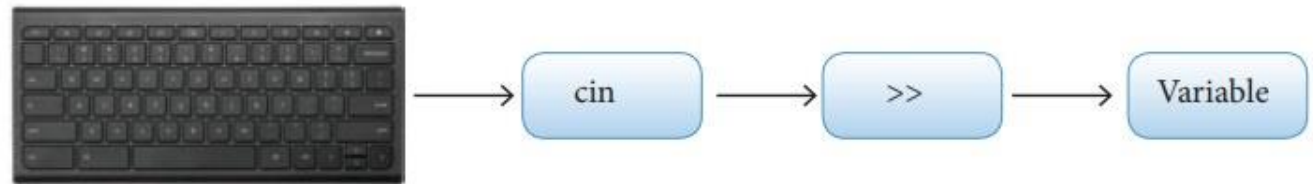


- Syntax: **cout<<data;**
- The **insertion operator** "<<" inserts data into **cout**
- **cout** is an output stream sending data to the monitor
- **data** refers to data to be printed which can be literal, variable, expression or constant.
- More than one **data** separated by **insertion operator** can be sent to screen with single **cout** output stream.

I/O Streams (cont'd)

■ C++ standard INPUT Stream (cin)

- **cin** (console input) is the object to read data typed at the keyboard in C++.



- Syntax: **cin>>variable;**
- **cin** is an input stream bringing data from the keyboard
- The **extraction operator (>>)** removes data to be used
- **variable** refers to memory location where the data will be stored.
- More than one **variable** separated by **extraction operator** can be used to capture data from keyboard with single **cin** input stream.

I/O Streams (cont'd)

- Sample program to demonstrate the standard Input/output stream

```
#include <iostream>
using namespace std;

int main()
{
    int room_width, room_length, room_area = 0;

    //prompt user to enter value to terminal using cout
    //and then get an input from keyboard using cin
    cout<<"Enter the width of the room: "<<endl;
    cin>>room_width;
    cout<<"Enter the length of the room: "<<endl;
    cin>> room_length;

    /*the above output and input stream statement can be also written as below
    cout << "Please enter width and length of the room:" << endl;
    cin >> room_width >> room_length;
    */

    //calculate the area of a room
    room_area = room_width*room_length;

    //display the area to the terminal
    cout<<"The area of the room is "<< room_area<<" square feet."<<endl;
    return 0;
}
```

C/C++ Library Functions and preprocessor

- Library functions also called “**built-in**” functions
- These are functions that are implemented in C/C++ and ready to be directly incorporated in our program as per our requirements
- Library functions in C/C++ are declared and defined in special files called “**Header Files**” which we can reference in our C/C++ programs using the “**include**” directive.
- Some of C/C++ library header files are tabularized as below

Headers	Description	Functions
iostream	This header contains the prototype for standard input and output functions used in C++	cin, cout, cerr, get(), getline() etc.
cmath/math	This is the header containing various math library functions.	sqrt(), pow(base,exponent), exp(x), fabs(x), log(x), log10(x), sin(x), cos(x), tan(x), asin(x), acos(x), atan(x) etc.
iomanip	This header contains stream manipulator functions that allow us to format the stream of data.	setw (int n), setfill(char c), setprecision(int n) etc.

C/C++ Library Functions (cont'd)

Headers	Description	Functions
cstdlib	contains various functions related to conversion between text and numbers, random numbers, and other utility functions.	abs(x), atof(const char* str), atoi(const char* str), atol(const char* str), atoll(const char* str) etc.
ctime	Contains function prototypes related to date and time manipulations in C++.	localtime(), clock(), difftime() etc.
cctype	Contain function prototypes that test the type of characters (digit, alphabet, etc.). It also has prototypes that are used to convert between uppercase and lowercase	toupper(ch), tolower(ch), isalpha(ch), isalnum(ch), isupper(ch), isdigit(ch), islower()
cstring	This header file includes function prototypes for C/C++-style string-processing functions.	strcpy (), strcat(), strcmp() etc.
fstream	Function as prototypes for functions that perform input/output from/to files on disk are included in fstream header.	open(), close(), put(), read(), write() etc.
climits	This header file has the integral size limits of the system.	CHAR_MIN, CHAR_MAX, INT_MIN, INT_MAX, UINT_MAX, LONG_MIN, LONG_MAX
cfloat	This header file contains the size limits for floating-point numbers on the system.	FLT_DIG, DBL_DIG, LDBL_DIG, FLT_MANT_DIG, DBL_MANT_DIG, LDBL_MANT_DIG

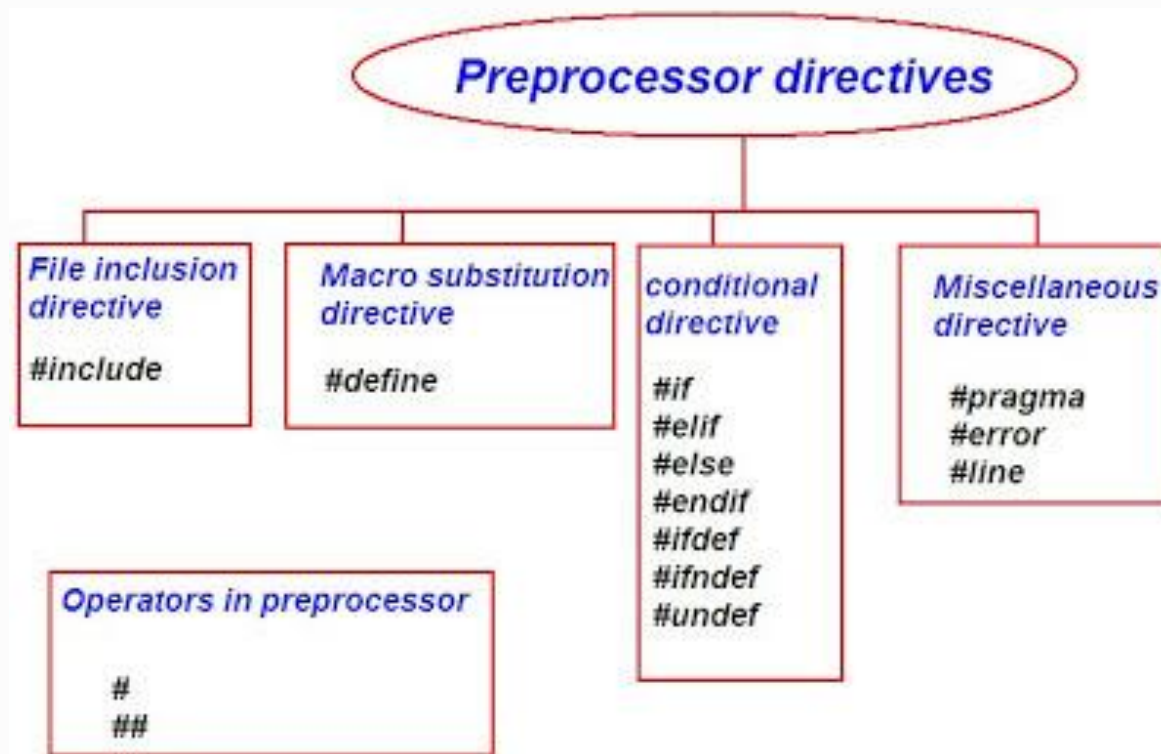
C/C++ Library Functions (cont'd)

Namespace: *using namespace std;*

- Namespace designed to overcome a difficulty of like the below
 - ✓ Sometimes a developer might be writing some code that has a function called `xyz()` and there is *another library available which is also having same function xyz()*.
 - ✓ The compiler has no way of knowing which version of `xyz()` function you are referring to within your code.
- Generally namespace designed to differentiate similar functions, variables etc. with the same name available in different libraries.
- There are several ways in which a namespace used in our program
 - ✓ Option 1: `using namespace std;`
 - ✓ Option 2: `using std::cout;` `using std::cin;`where “std” is identifier of the namespace

Preprocessor Directives

- A ***preprocessor*** is a program that invoked by the compiler before the program is converted to machine language
- In C++ preprocessor obeys command called *preprocessor directives*, which indicate that certain manipulations are to be performed on the program before translation begins.



Preprocessor Directives (cont'd)

■ A *file preprocessor*

#include <filename> versus #include “/path/filename.h”

■ #include <filename>

- Looks in the IDE C++ developer tools standard include directories that were created at install time for the file.

■ #include “/path/filename.h”

- Looks for the filename.h file in the /path/ directory path.
- Used when you are creating your own files (we will learn later how to do) OR when you have downloaded some 3rd party non-standard C++ header files.

Comments

- Used to document parts of the program
- Are ignored by the compiler
- Type of comments -- Single comment and Multi line comment

- **Single comment** - Begin with `//` through to the end of line:

e.g.1) `int length = 12; // length in inches`

e.g. 2) `// calculate rectangle area`
`area = length * width;`

- **Multi line comment** - Begin with `/*` and end with `*/` and

- ✓ Can span multiple lines

e.g. `/* this is a multi-line
comment
*/`

- ✓ Can begin and end on the same line

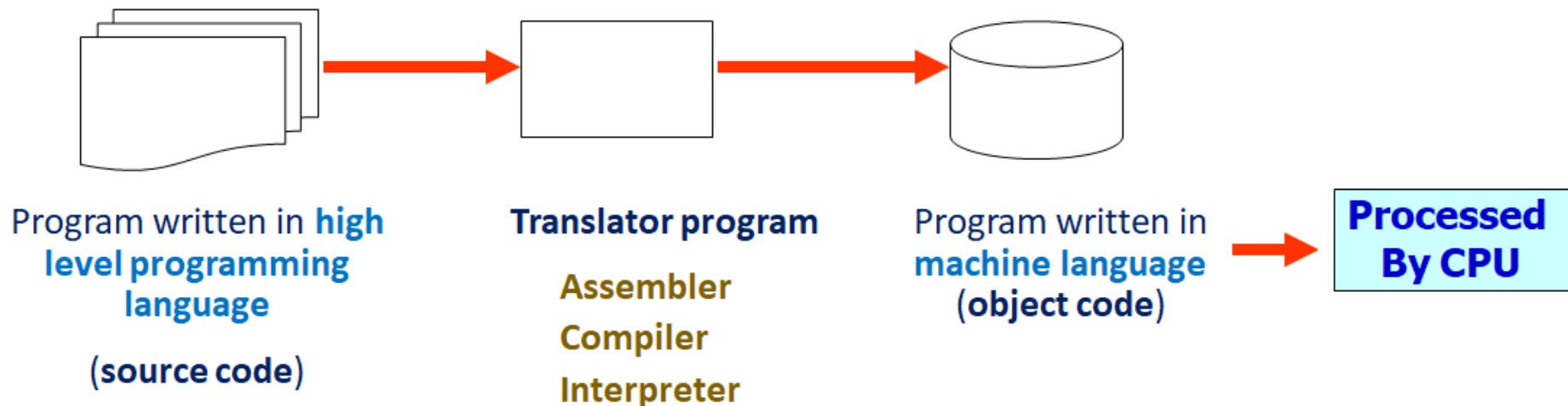
e.g. `int area; /* calculated area */`

Comments (cont'd)

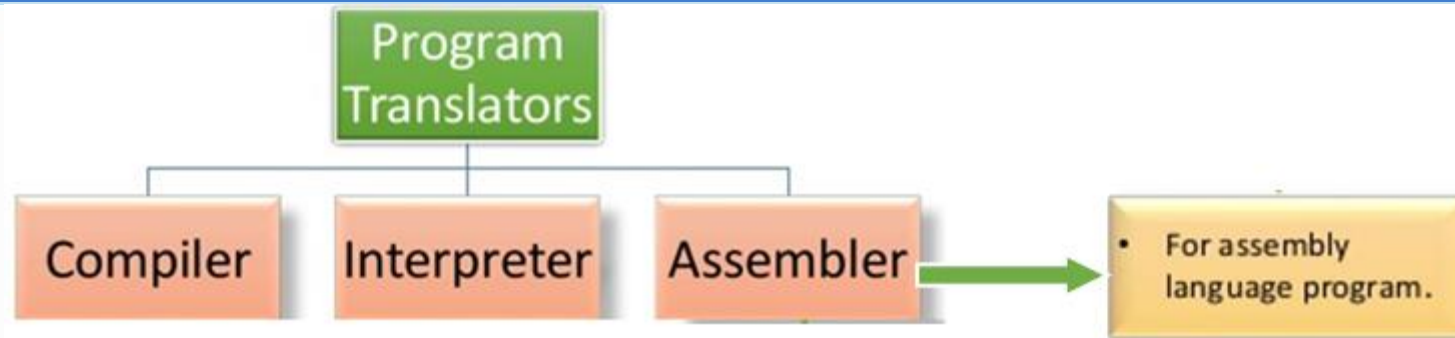
- **Purpose:** Intended for persons reading the source code of the program
 - ✓ Indicate the purpose of the program
 - ✓ Describe the use of variables
 - ✓ Explain complex sections of code
- **Comment Guidelines --where**
 - ✓ Towards the top of your program
 - ✓ Before every function you create
 - ✓ Before loops and test conditions
 - ✓ Next to declared variables.

Translation and Execution process (1/4)

- The only language that the computer understands is the machine language.
- Therefore any program that is written in either assembly or high level language must be translated to machine code so that the computer could process it.
- This process is called **Program translation**/ and performed by special software called **translator**

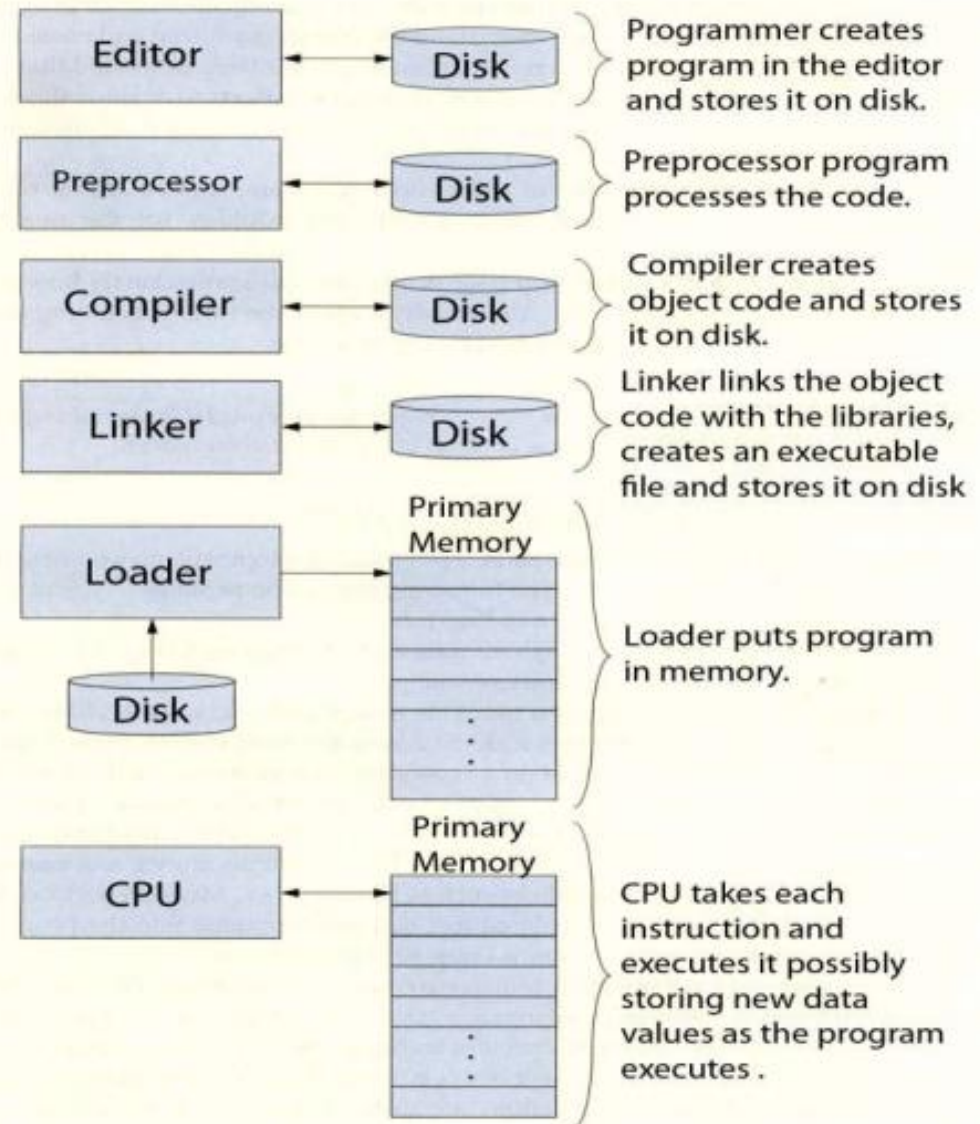
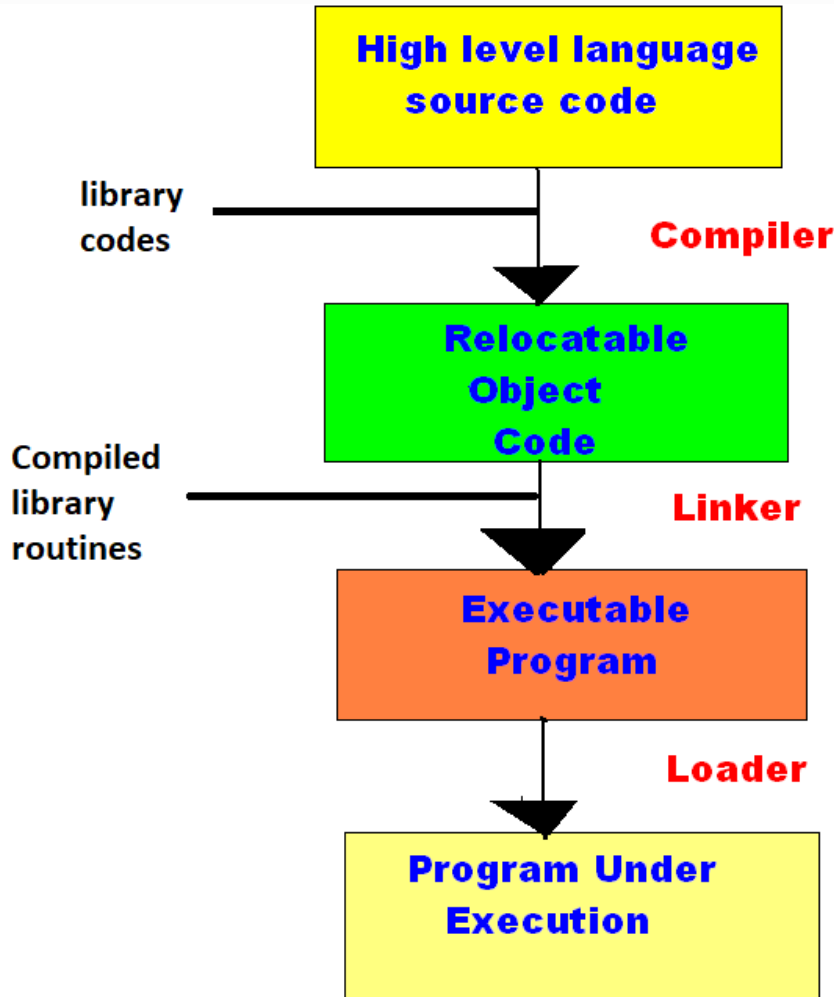


Translation and Execution process (2/4)

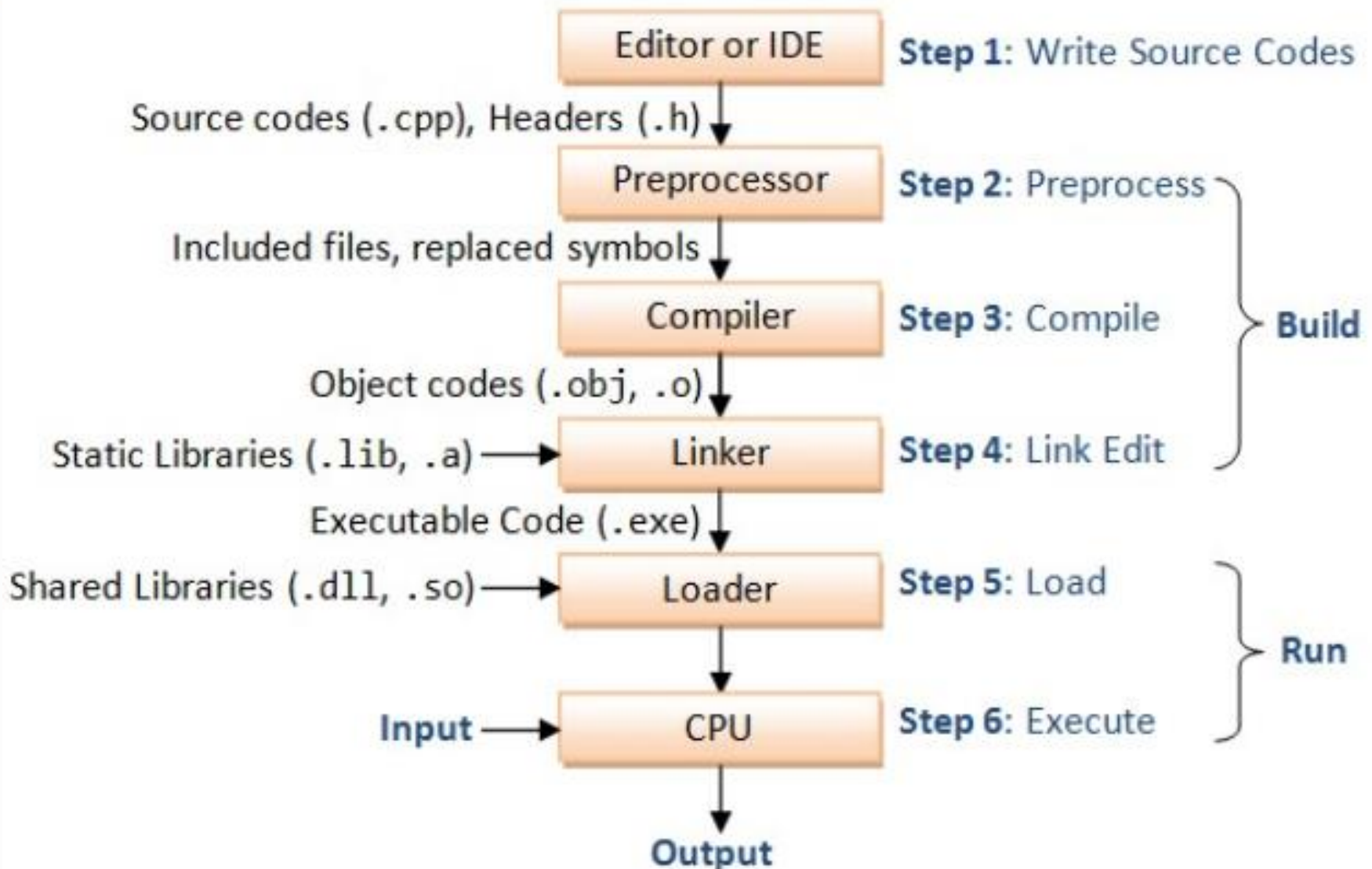


Compiler	Interpreter
Compiler scans the whole program in one go	Translates program one statement at a time.
As it scans the code in one go, the errors (if any) are shown at the end together.	Considering it scans code one line at a time, errors are shown line by line.
Converts the source code into object code.	does not convert source code into object code
More memory requirement due to creation object. However, comparatively it faster	Requires less memory. However, it is slower
C, C++, C# etc.	Python, Ruby, Perl, SNOBOL, MATLAB, etc.

Translation and Execution process (3/4)



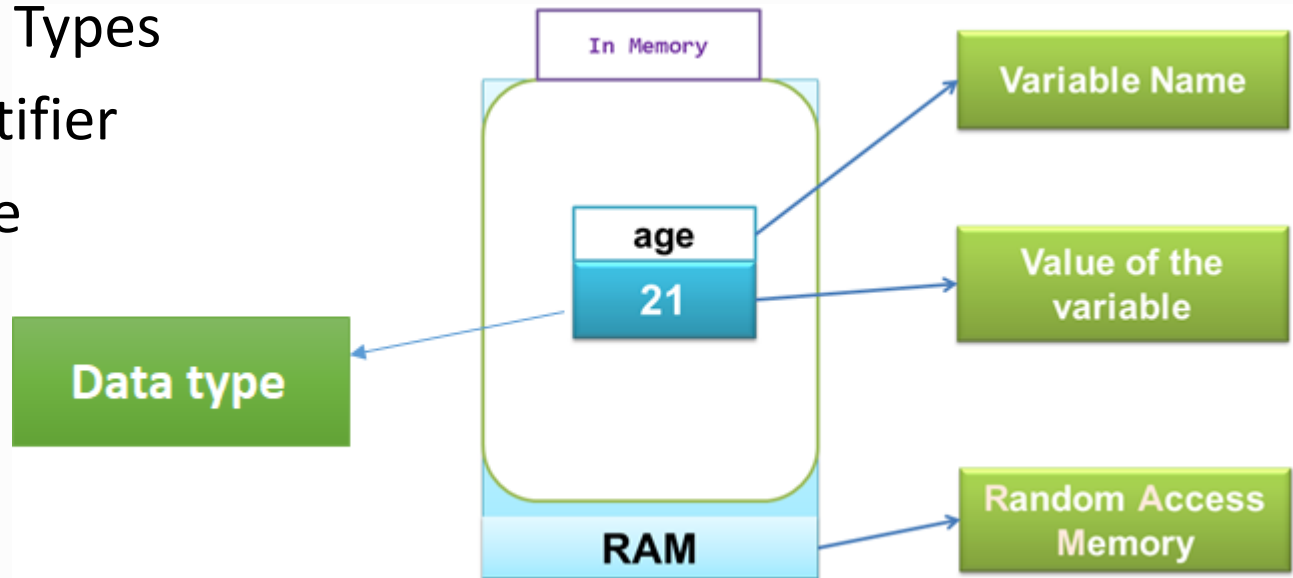
Translation and Execution process (4/4)



Variables and Data types

■ Variables

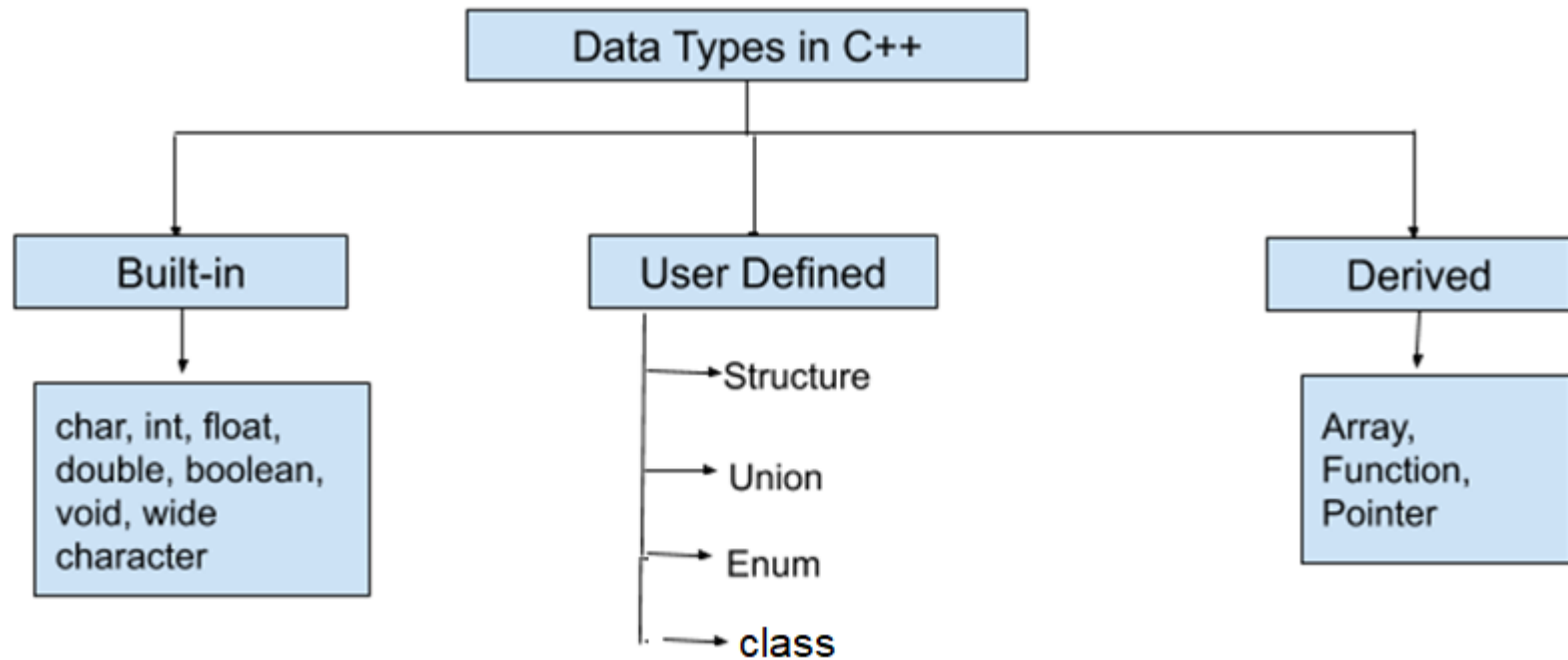
- ✓ A primary goal of computer program is data processing
- ✓ **Variable** is a *named reserved place (storage location)* in memory, which stores *a piece of data* of a particular data type.
- ✓ As a result it can be used in various computations in a program
- ✓ All variables have three important components
 - Data Types
 - Identifier
 - Value



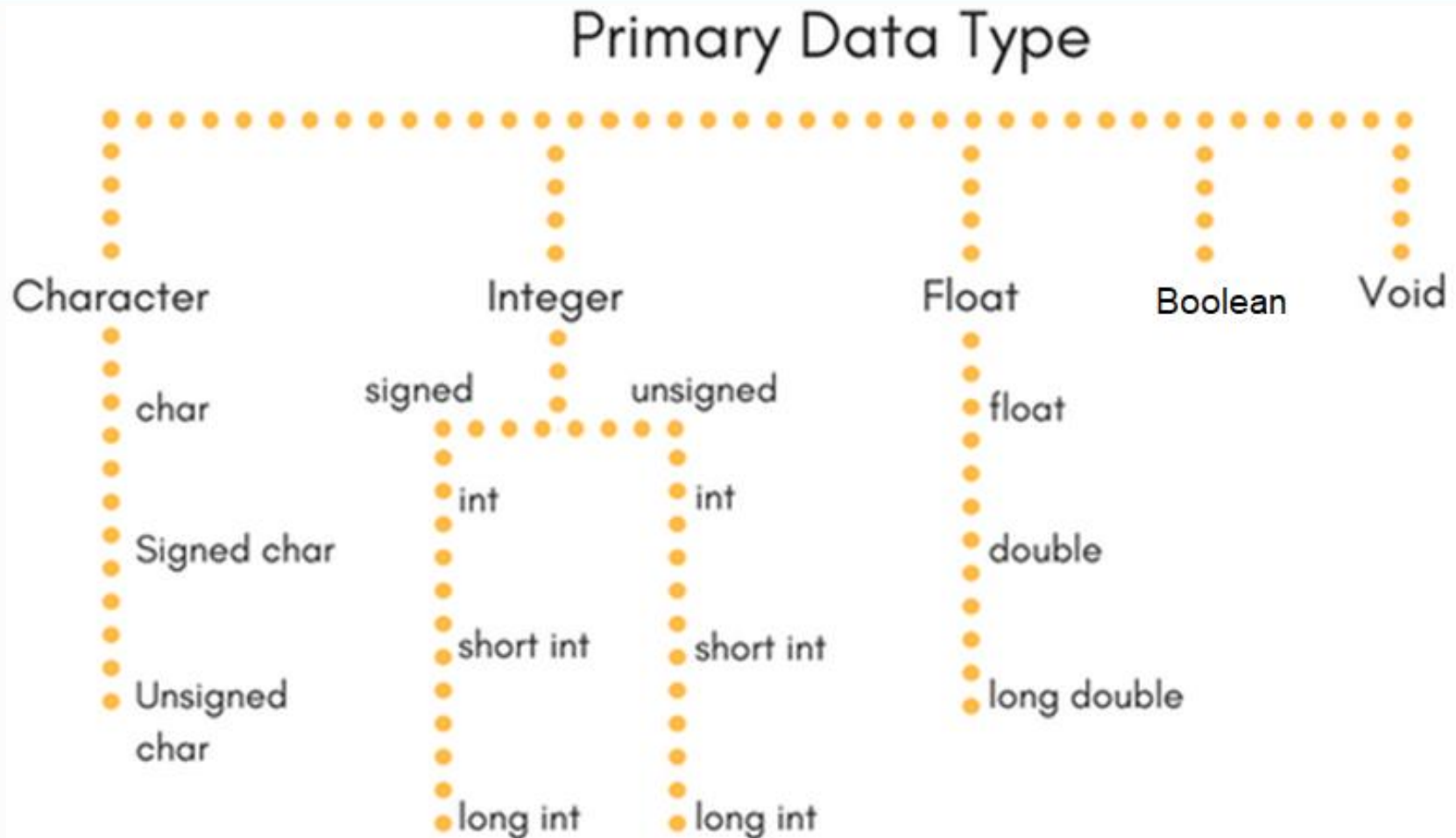
Variables and Data types (cont'd)

■ Data Types

- ✓ Describes the property of the data and the size of the reserved memory.
- ✓ Established when the variable is defined
- ✓ Data types supported by C++ can be classified as follow



Variables and Data types (cont'd)



- **Signed** - either negative or positive
- **Unsigned** - refers to positive integer only

Variables and Data types (cont'd)

- Data types size and range of value

16 Bit Compiler like Turbo C++

Data Type	Size (Byte)
char	1
short int	2
int	2
long int	4
float	4
double	8
long double	10
void	Nothing

32 Bit Compiler like Linux GCC

Data Type	Size (Byte)
char	1
short int	2
int	4
long int	4
float	4
double	8
long double	12
void	Nothing

Variables and Data types (cont'd)

▪ Data types size and range of value

Type	Size (in bytes)	Range
char	1	-127 to 127 or 0 to 255
unsigned char	1	0 to 255
int	4	-2147483648 to 2147483647
unsigned int	4	0 to 4294967295
short int	2	-32768 to 32767
unsigned short int	2	0 to 65,535
long int	4	-2147483648 to 2147483647
unsigned long int	4	0 to 4294967295
float	4	+/- 3.4e +/- 38 (~7 digits)
double	8	+/- 1.7e +/- 308 (~15 digits)

Variables and Data types (cont'd)

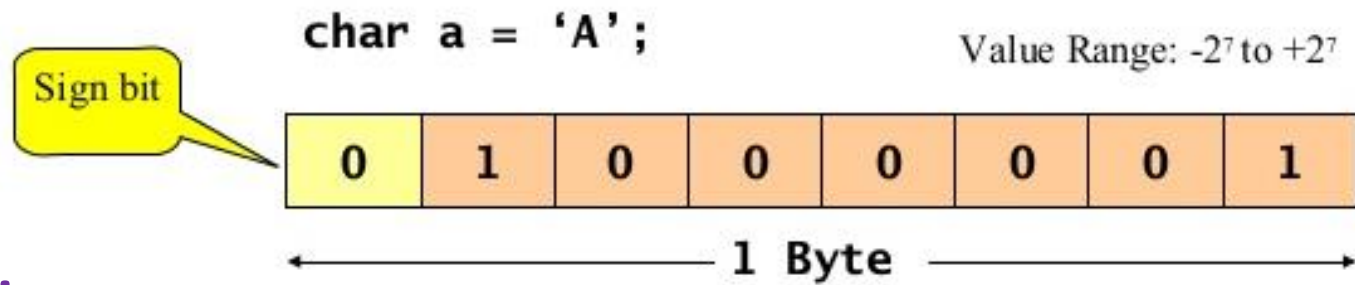
- The program below prints the size of C++ primitive data types

```
//Print Size of Fundamental Types
#include <iostream>
using namespace std;

int main(){
    char var;
    cout<<"sizeof(char) is "<<sizeof(var)<<" bytes "<<endl;
    cout<<"sizeof(char) is "<<sizeof(char)<<" bytes "<<endl;
    cout<<"sizeof(short) is "<<sizeof(short)<<" bytes "<<endl;
    cout<<"sizeof(int) is "<<sizeof(int)<<" bytes "<<endl;
    cout<<"sizeof(long) is "<<sizeof(long)<<" bytes "<<endl;
    cout<<"sizeof(float) is "<<sizeof(float)<<" bytes "<<endl;
    cout<<"sizeof(double) is "<<sizeof(double)<<" bytes\n";
    cout<<"sizeof(long double) is "<<sizeof(long double)<<" bytes\n";
    cout<<"sizeof(bool) is "<<sizeof(bool) <<" bytes " <<endl;
    return 0;
}
```

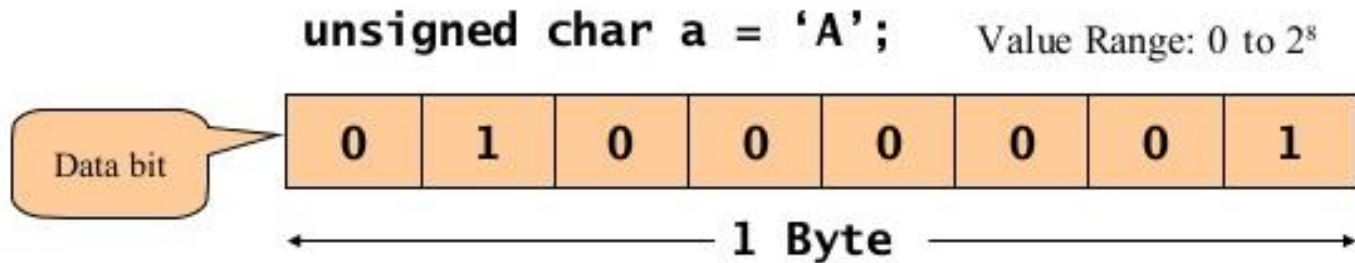
Variables and Data types (cont'd)

■ Signed Vs. Unsigned



■ +ve – 0 bit

■ -ve ---> 1 bit



■ By default

✓ integer Data types are Signed where as char data types are Unsigned

✓ float, double, long double is also Signed, and cannot restricted to be Unsigned

Variables and Data types (cont'd)

Mapping Signed \leftrightarrow Unsigned

Bits	Signed		Unsigned
0000	0	\longleftrightarrow =	0
0001	1		1
0010	2		2
0011	3		3
0100	4		4
0101	5		5
0110	6		6
0111	7		7
1000	-8	\longleftrightarrow +/- 16	8
1001	-7		9
1010	-6		10
1011	-5		11
1100	-4		12
1101	-3		13
1110	-2		14
1111	-1		15

Variables and Data types (cont'd)

Data type Value Range

This program demonstrate the value range that **int**, **unsigned**, **short int**, and **unsigned short** store.

```
#include <iostream>
using namespace std;

int main(){

    short num1;          unsigned short num2;
    int num3;             unsigned int num4;
    cout<<"Enter value of num1: ";
    cin>>num1;
    cout<<"Value of num1 you entered is "<<num1;
    cout<<"\n \nEnter value of num2: ";
    cin>>num2;
    cout<<"Value of num2 you entered is "<<num2;
    cout<<"Enter value of num3: ";
    cin>>num3;
    cout<<"Value of num3 you entered is "<<num3;
    cout<<"\n \nEnter value of num4: ";
    cin>>num4;
    cout<<"Value of num4 you entered is "<<num4;

    return 0;
}
```

Variables Naming Conventions

- An **identifiers** refers to the name of variable that needed to uniquely identify each variable, so as **to store a value** to the variable and **retrieve the value** stored
- **Rules of Making Identifier (Conventions)**
 - ✓ Identifier name can be the combination of alphabets (a – z and A - Z), digit (0 -9) or underscore (_).
E.g. sum50, avgUpto100 etc.
 - ✓ First character must be either alphabet or underscore
E.g. _sum, class_strength, height ----- valid
123sum, 25th_var ----- invalid
 - ✓ No space and No other special symbols(!,@,%,\$,*,(,),-,+,= etc) except underscore
E.g. _calulate, _5, a_

Variables Naming Conventions (Cont'd)

- ✓ Variable name should not be a keyword or reserve word
Invalid names: interrupt, float, asm, enum etc.
- ✓ Variable name must be unique and case sensitive

Good practice of Identifier

- ✓ It is important to choose a name that is *self-descriptive* and closely reflects the meaning of the variable.
- ✓ Avoid using single alphabet and meaningless names like below
a, b, c, d, i1, j99
- ✓ Some of allowed single alphabets include x, y, i, j, k
- ✓ Use came rule if the identifier constructed from more than words
thefontSize, roomNumber, xMax, yMin,
xTopLeft, thisIsAVeryLongName

C++ Key Words

- Certain words are reserved by C++ for specific purposes and have a unique meaning within a program.
- These are called **reserved words** or **keywords**.
- cannot be used for any other purposes.
- All reserved words are in lower-case letters.

and	continue	goto	public	try
and_eq	default	if	register	typedef
asm	delete	inline	reinterpret_cast	typeid
auto	do	int	return	typename
bitand	double	long	short	union
bitor	dynamic_cast	mutable	signed	unsigned
bool	else	namespace	sizeof	using
break	enum	new	static	virtual
case	explicit	not	static_cast	void
catch	export	not_eq	struct	volatile
char	extern	operator	switch	wchar_t
class	false	or	template	while
compl	float	or_eq	this	xor
const	for	private	throw	xor_eq
const_cast	friend	protected	true	

Variables declaration and Initialization

■ Variable Declaration

- ✓ Variable must first be declared (defined) before they can be used.
- ✓ Variables can be created in a process known as **declaration** which instructs the computer to *reserve a memory location with the name and size*
- ✓ Declaration syntax:
DataType Identifier;
- ✓ Declaration Examples:
 - float double moon_distance;
 - double average, m_score, total_score;
 - int age, num_students;

Variables declaration and Initialization (cont'd)

■ Variable Initialization

- ✓ When a variable is assigned a value at the time of declaration
- ✓ Declaration and initialization can be combined using two methods

Method 1: DataType Identifier = Initial value;

Method 2: DataType Identifier (Initial value);

- ✓ Example:

```
int length = 12;  
double width = 26.3, area = 0.0;  
int length (12), width (5);
```

- ✓ We can initialize some or all variables:

```
int length = 12, width, area (0.0);
```

User Defined Data types

- “**typedef**” Keyword is used to define a data type by the programmer
- Evaluate the below two sample codes

```
#include<iostream.h>
void main()
{
    unsigned short int num1;
    ...
    ...
    unsigned short int num2;
    ...
    ...
    unsigned short int num3;
    ...
    ...
}
```



```
#include<iostream.h>
typedef unsigned short int USHORT;
void main()
{
    USHORT width = 9;
    ...
}
```

Scope of Variables

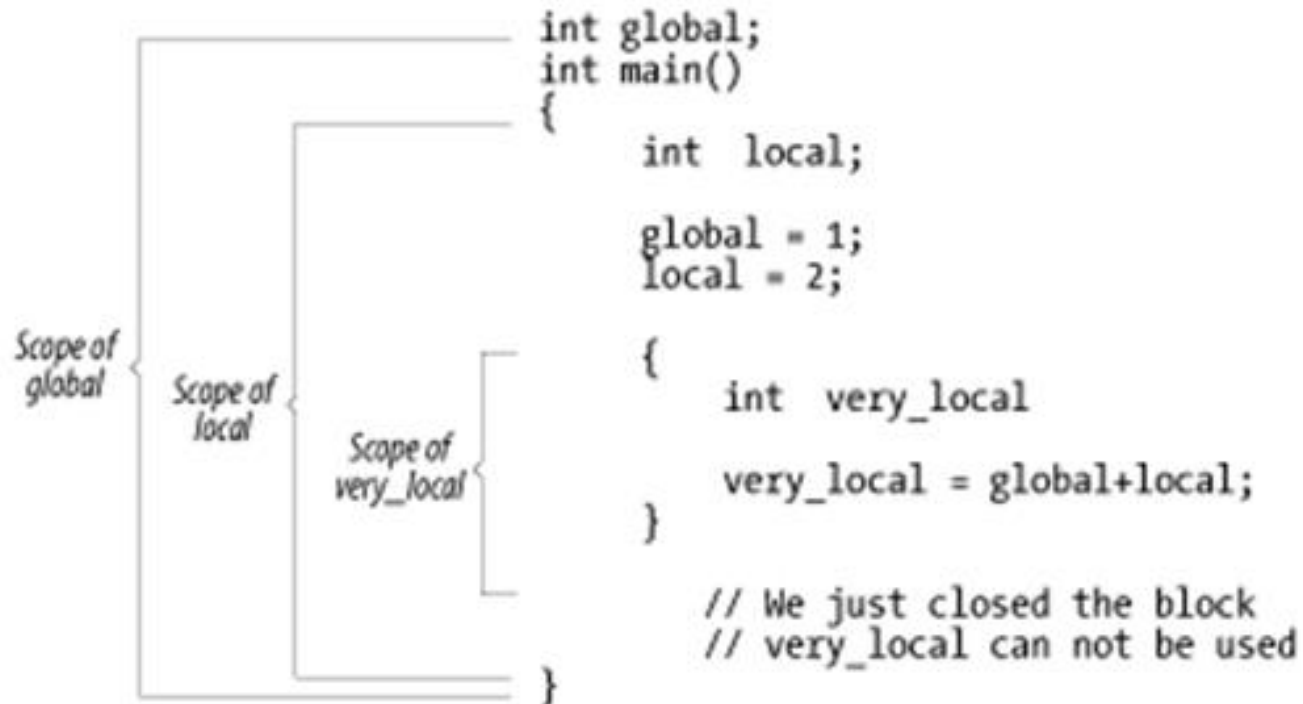
- Variables in a program can be declared just about anywhere.
- However, the accessibility, visibility and length of life of a variable depend on where the variable is declared.
- Scope of a variable** is the boundary or block in a program where a variable can be accessed

Global Variable

- referred/accessed anywhere in the code

Local Variable

- Accessed only inside a block within which they are declared



Scope of Variables (cont'd)

- Sample program to demonstrate scope of variable

```
#include <iostream>
using namespace std;

int age = 34; //global variable

int main(){

    int age = 14; //local variable

    cout<<"Local varibale "<<age<<endl;

//use resolution operator to use global varibale
    cout<<"Global varibale "<<::age<<endl;

    return 0;
}
```

Constants and literals (1/8)

■ Constant

- ✓ Like a variable a data storage locations in the computer memory that has a ***fixed value*** and that do not change their content during the execution of a program.
- ✓ Must be initialized when they are created by the program,
- ✓ The programmer can't assign a new value to a constant later.
- ✓ Usually defined for values that will be used more than once in a program but not changed

E.g., $\text{PI} = 3.14$, $\text{Sun_Seed} = 3 \times 10^8$

- ✓ It is customary to use all capital letters, joined with underscore in constant identifiers to distinguish them from other kinds of identifiers.

E.g., `MIN_VALUE`, `MAX_SIZE`.

Constants and literals (2/8)

(a) Defining constant using const keyword

- ✓ Syntax: `const DataType Identifier = value;`
- ✓ Example: `const float PI = 3.14;`

(b) Defining constant the #define preprocessor

- ✓ Syntax: `#define Identifier value`
- ✓ Example: `#define PI 3.14`

```
#include <iostream>
using namespace std;

int main() {
    const int    LENGTH = 10, WIDTH  = 5;
    const char   NEWLINE = '\n';
    int area;

    area = LENGTH * WIDTH;
    cout << area << NEWLINE;
    return 0;
}
```

```
#include <iostream>
using namespace std;

#define LENGTH 10.5
#define WIDTH  2
#define NEWLINE '\n'

int main() {
    float area = LENGTH * WIDTH;
    cout << area << NEWLINE;
    return 0;
}
```

Constants and literals (3/8)

■ Literals

- ✓ are data used for representing fixed values.
- ✓ They can be used directly in the code.
- ✓ For example: 1, 2.5, 'c' etc.



Constants and literals (4/8)

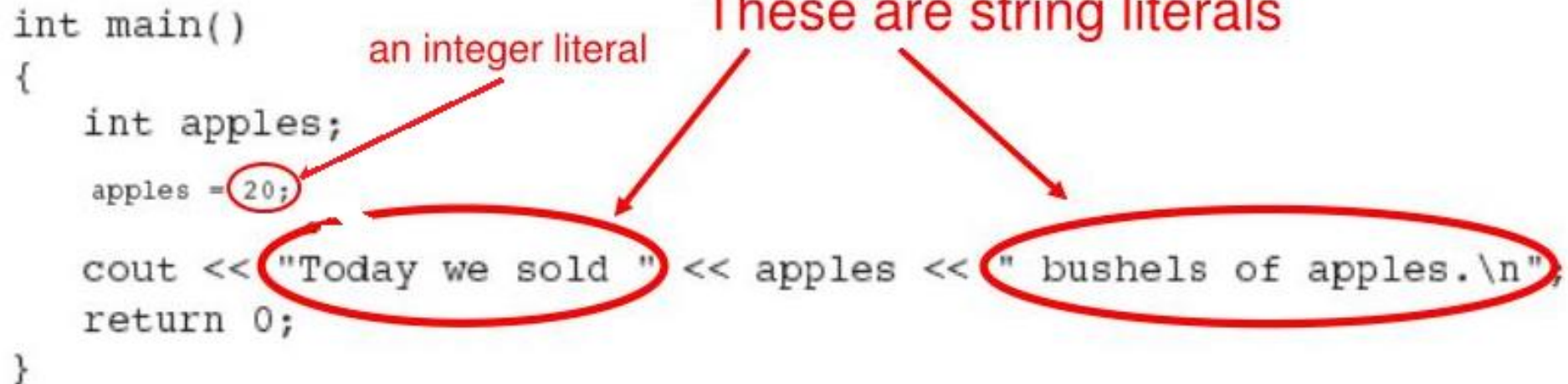
■ Sample program

```
// This program has literals and a variable.  
#include <iostream>  
using namespace std;
```

```
int main()  
{  
    int apples;  
    apples = 20;  
    cout << "Today we sold " << apples << " bushels of apples.\n";  
    return 0;  
}
```

an integer literal

These are string literals



Constants and literals (5/8)

■ C++ Escape characters

Escape Sequence	Meaning
<code>\n</code>	New Line
<code>\t</code>	Horizontal Tab
<code>\b</code>	BackSpace
<code>\r</code>	Carriage Return
<code>\a</code>	Audible bell
<code>\'</code>	Printing single quotation
<code>\"</code>	printing double quotation
<code>\?</code>	Question Mark Sequence
<code>\\</code>	Back Slash
<code>\f</code>	Form Feed
<code>\v</code>	Vertical Tab
<code>\0</code>	Null Value
<code>\nnn</code>	Print octal value
<code>\xhh</code>	Print Hexadecimal value

Escape Sequence	ASCII value
<code>\n</code>	10
<code>\t</code>	9
<code>\v</code>	11
<code>\b</code>	8
<code>\a</code>	7
<code>\"</code>	34
<code>\'</code>	39
<code>\?</code>	63
<code>\\</code>	92
<code>\0</code>	0
<code>\r</code>	013

Constants and literals (6/8)

■ Special characters

Character	Name	Meaning
#	Pound sign	Beginning of preprocessor directive
< >	Open/close brackets	Enclose filename in #include
()	Open/close parentheses	Used when naming a function
{ }	Open/close brace	Encloses a group of statements
" "	Double open/close quotation marks	Encloses string of characters
` `	Single open/close quotation marks	Encloses character

Constants and literals (7/8)

■ Enumerated Constant

- ✓ Use to declare **multiple integer** constants using single line with different features.
- ✓ Cannot take any other data type than integer
- ✓ *enum* types can be used to set up collections of named integer constants.
- ✓ Enables programmers to define variables and restrict the value of that variable to a set of possible values which are integer.
- ✓ Use the keyword *enum* is short for "enumerated".
- ✓ Syntax: **enum identifier {enumerated list};**

```
#define SPRING 0  
  
#define SUMMER 1  
  
#define FALL 2
```



```
enum COLOR {  
  
    RED,  
    BLUE,  
    GREEN,  
    WHITE,  
    BLACK};
```

Constants and literals (8/8)

enum sample program

```
#include <iostream>
using namespace std;

enum days {Monday = 1, Teusday,
           Wednesday, Thrusday,
           Friday, Sataruday, Sanduy};

int main(){

    cout<<"Yesterday is "<<days(3)<<endl;
    cout<<"Todday is "<<days(4)<<endl;
    cout<<"Tomorrow is "<<days(5)<<endl;

    return 0;
}
```

```
enum Season {
    WINTER,      // = 0 by default
    SPRING,      // = WINTER + 1
    SUMMER,      // = WINTER + 2
    AUTUMN       // = WINTER + 3
};

enum {SUNDAY=1, MONDAY, TUESDAY, ...}; // nameless

enum Color {BLACK,RED,GREEN,YELLOW,BLUE,WHITE=7,GRAY};
//          0      1      2      3      4      7      8
```

Operators and expression (1/ 17)

■ Operators

- ✓ A symbol that tells the computer to perform certain **mathematical (or) logical manipulations** (makes the machine to take an action).
- ✓ Used in programs to manipulate data and variables.
- ✓ Different Operators act on one or more operands
- ✓ Based on the number of the operands the operators act on, operators are grouped as
 - **Unary** – only single operand used
 - **Binary** – require two operands
 - **Ternary** – require three operands

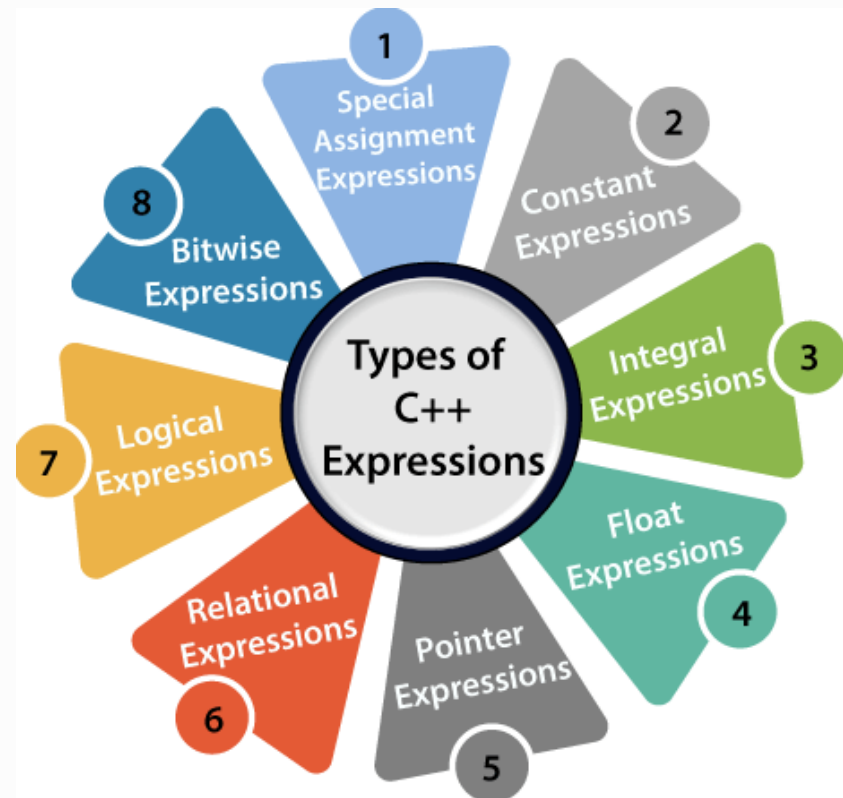
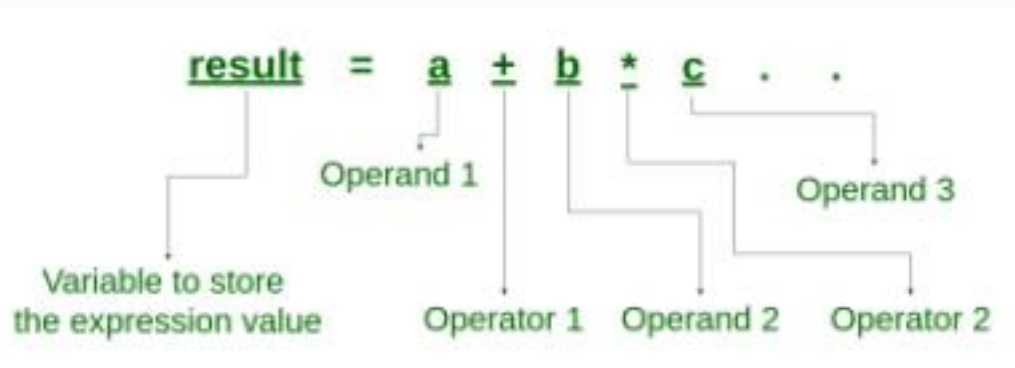
Operators and expression (2/17)

	Operator	Type
Binary Operator	+, -, *, /, %	Arithmetic Operators
	<, <=, >, >=, ==, !=	Relational Operators
	&&, , !	Logical Operators
	&, , <<, >>, ~, ^	Bitwise Operators
	=, +=, -=, *=, /=, %/=	Assignment Operators
Unary Operator	→ ++, --	Unary Operator
Ternary Operator	→ ?:	Ternary or Conditional Operator
Miscellaneous Operators	→ comma, sizeof, insetion, extraction, sizeof(), typecasting	

Operators and expression (3/17)

■ Expression

- ✓ A combination of *operators and operands* (variables or literal values), that can be evaluated to *yield a single value of a certain type*.
- ✓ It can also be viewed as any statement that evaluates to a value (returns a value)
- ✓ Examples:
 - `x=3.2;` returns the value 3.2
 - `x = a + b;`



Operators and expression (4/17)

Arithmetic Operators

SYMBOL	OPERATION	EXAMPLE	VALUE OF ans
+	addition	ans = 7 + 3;	10
-	subtraction	ans = 7 - 3;	4
*	multiplication	ans = 7 * 3;	21
/	division	ans = 7 / 3;	2
%	modulus	ans = 7 % 3;	1

Points to be considered

- ✓ Arithmetic expression
- ✓ Mixed type operations - *two operands belong to different types*
- ✓ Type casting – *implicitly type casting*
- ✓ Overflow/Under Flow – *what happen if the computation result exceed the storage capacity of the result variable*

$$\frac{1 + 2a}{3} + \frac{4(b + c)(5 - d - e)}{f} - 6\left(\frac{7}{g} + h\right)$$

Operators and expression (5/17)

■ Overflow/Underflow and wrapping around

- ✓ Arithmetic overflow/underflow happens when an arithmetic operation results in a value that is outside the range of values representable by the given data type
- ✓ **Wrapping around unsigned integers** - when the program runs out of positive number, the program moves into the largest negative number and then counts back to zero
- ✓ **Wrapping around signed integers** - when the program runs out of positive number, the program moves into the largest negative number and then counts back to smallest negative value

Because of overflow the value of `num2` & `num` either **wrapping around** or the program generate **garbage value**



```
#include <iostream>
using namespace std;

int main(){

    unsigned short num2 = 88945;
    int num = 3465891;

    cout<<"\n \n Value of Var1: "<<num1<<endl;
    cout<<"\n \n Value of Var2: "<<num2<<endl;

    return 0;
}
```

Operators and expression (6/17)

■ Exercise

(1) Given below constants

Base pay rate = 180.25 ETB

regular work hours ≤ 40 hrs.

Over time payment rate = 120 ETB

Write a program that read hours worked and calculate hourly wages, overtime and total wages.

(2) Write a program which inputs a temperature reading expressed in Fahrenheit and outputs its equivalent in Celsius, using the formula

$$^{\circ}C = \frac{5}{9} (^{\circ}F - 32)$$

Operators and expression (7/17)

■ Assignment Operators

- ✓ Copy/put the value/content (of RHS) ----> to a variable (of LHS)
- ✓ Evaluates an expression (of RHS) and assign the resultant value to a variable (of LHS)

Single assignment operators

Compound assignment operators

Multiple assignment
a = b = c = 36

Operator	Example	Equivalent Expression
=	$m = 10$	$m = 10$
+=	$m += 10$	$m = m + 10$
-=	$m -= 10$	$m = m - 10$
*=	$m *= 10$	$m = m * 10$
/=	$m /=$	$m = m/10$
%=	$m \% = 10$	$m = m \% 10$
<<=	$a <<= b$	$a = a << b$
>>=	$a >>= b$	$a = a >> b$
>>>=	$a >>>= b$	$a = a >>> b$
&=	$a \&= b$	$a = a \& b$
^=	$a \wedge= b$	$a = a \wedge b$
=	$a = b$	$a = a b$

Operators and expression (8/17)

■ Relation (Comparison) Operators

- ✓ Evaluated to true/false
- ✓ Used to compare two operands that must evaluated to numeric
- ✓ Characters and Boolean are valid operands as they are represented by numeric. Character can be evaluated by their ASCII value

Operators	Meaning	Example	Result
<	Less than	5<2	False
>	Greater than	5>2	True
<=	Less than or equal to	5<=2	False
>=	Greater than or equal to	5>=2	True
==	Equal to	5==2	False
!=	Not equal to	5!=2	True
===	Equal value and same type	5 === 5	True
		5 === "5"	False
!==	Not Equal value or Not same type	5 !== 5	False
		5 !== "5"	True

Operators and expression (9/17)

■ Logical Operators

- ✓ Evaluated to true/false
- ✓ Used to compare two logical statements
- ✓ any non-zero value can be used to represent the **logical true**, whereas only **zero** represents the **logical false**

Operator	Operation	Description
&&	AND	The logical AND combines two different relational expressions in to one. It returns 1 (True), if both expression are true, otherwise it returns 0 (false).
	OR	The logical OR combines two different relational expressions in to one. It returns 1 (True), if either one of the expression is true. It returns 0 (false), if both the expressions are false.
!	NOT	NOT works on a single expression / operand. It simply negates or inverts the truth value. i.e., if an operand / expression is 1 (true) then this operator returns 0 (false) and vice versa

Example:

```
// Return true if x is between 0 and 100 (inclusive)
(x >= 0) && (x <= 100)

// wrong to use 0 <= x <= 100

// Return true if x is outside 0 and 100 (inclusive)
(x < 0) || (x > 100) //or
!((x >= 0) && (x <= 100))
```

Exercise:

- Given the year, month (1-12), and day (1-31), write a Boolean expression which returns true for dates before October 15, 1582 (G.C. cut over date)

Operators and expression (10/17)

■ Bitwise Operators

- ✓ It refers to the testing, setting or shifting of actual bits in a byte or word.
- ✓ There are bitwise operators

Operator	Operation	Result							
&	a & b	a	0	1	0	0	0	0	1
		b	0	0	0	0	1	1	1
		a & b	0	0	0	0	0	0	1
		$(a \& b) = 0000\ 0001_2 = 1_{10}$							
	a b	a	0	1	0	0	0	0	1
		b	0	0	0	0	1	1	1
		a b	0	1	0	0	1	1	1
		$(a b) = 01001111_2 = 79_{10}$							
^	a ^ b	a	0	1	0	0	0	0	1
		b	0	0	0	0	1	1	1
		a ^ b	0	1	0	0	1	1	0
		$(a \wedge b) = 0100\ 1110_2 = 78_{10}$							

Operators and expression (11/17)

- **Increment/Decrement Operators (++/--)**
 - ✓ Used to automatically increment and decrement the value of a variable by 1
 - ✓ Provide a convenient way of, respectively, adding and subtracting 1 from a numeric variable
- **Prefix increment/decrement (++Operand or --Operand)**
 - ✓ Adds/subtracts 1 to the operand & result is assigned to the variable on the left before any other operation performed
- **Postfix increment/decrement (Operand++ or Operand--)**
 - ✓ First assigns the value to the variable on the left or use the current value of operand in operation & then increments/decrements the operand

Operators and expression (12/17)

■ Example 1

```
#include <iostream>
using namespace std;

main() {
    int a = 21, c ;

    c = a++;    //postfix increment
    cout<<"Value of c after assigned a++ is :"<<c<<endl;
    cout<<"Value of a is  after a++:" << a << endl ;

    c = ++a;    //prefix increment
    cout << Value of c after assigned ++a is :"<<c<<endl;
    cout<<"Value of a is  after ++a:" << a << endl ;
    return 0;
}
```


Operators and expression (13/17)

■ Example 2

```
#include <iostream>
using namespace std;

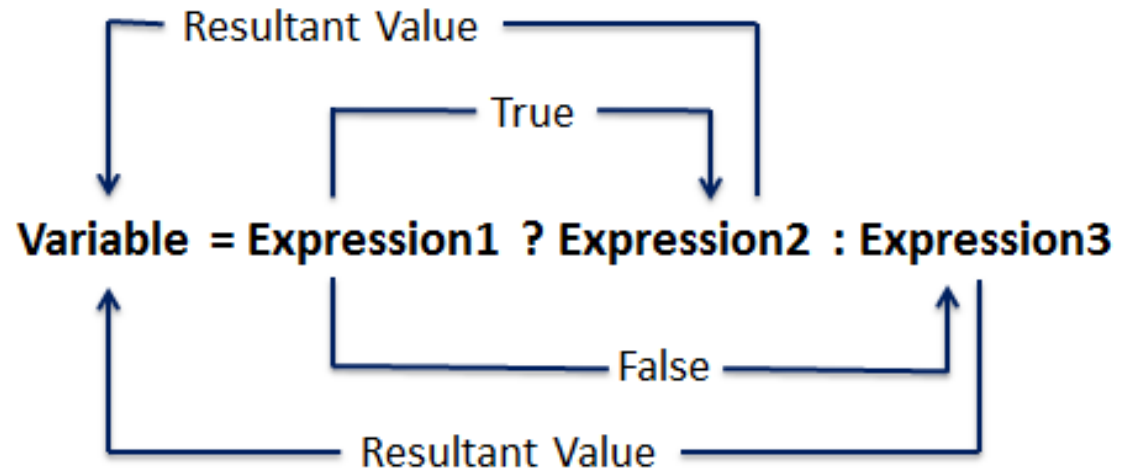
main() {
    int a = 21, c ;
    a++; //postfix increment
    c = a;
    cout<<"Value of c after assigned a++ is :"<<c<<endl;
    cout<<"Value of a is  after a++:" << a << endl ;

    ++a; //prefix increment
    c = a;
    cout << Value of c after assigned ++a is :"<<c<<endl;
    cout<<"Value of a is  after ++a:" << a << endl ;
    return 0;
}
```

Operators and expression (14/17)

■ Miscellaneous Operators

- ✓ Conditional operator (ternary operator)



```

//Conditional operator
#include <iostream>
using namespace std;

int main(){

    int i, j;
    cout<<"Enter two number separated by space: ";
    cin>>i>>j;
    int x = i>=j?i:j;
    cout<<"The larger number is "<<x<<endl;

    return 0;
}
    
```

Output:

i = 10 j = 5
10>=5? 10: 5

The larger
number is 10

Operators and expression (15/17)

■ Miscellaneous Operators

✓ Type casting operator (using **bracket**)

```
average = (double)sum / 100;    // Cast sum from int to double before division
average = sum / (double)100;    // Cast 100 from int to double before division
average = sum / 100.0;
average = (double)(sum / 100);  // Won't work. why?

// C++ also support function-style type casting in the form of new-type(operand)
average = double(sum) / 100;    // Same as (double)sum / 100
```

✓ Type casting operator (using **static-cast**) – provides error message

```
double d = 5.5;
int i = static_cast<int>(d);
float f = static_cast<float>(i);
long l = static_cast<long>(d);
```

Operators and expression (16/17)

```
/* Test Type Casting (TestTypeCast.cpp)
*/
#include <iostream>
#include <iomanip>
using namespace std;

int main() {
    // Print floating-point number in fixed
    // format with 1 decimal place
    cout << fixed << setprecision(1);

    // Test explicit type casting
    int i1 = 4, i2 = 8;
    cout << i1 / i2 << endl;           // 0
    cout << (double)i1 / i2 << endl;   // 0.5
    cout << i1 / (double)i2 << endl;   // 0.5
    cout << (double)(i1 / i2) << endl; // 0.0
}
```

```
double d1 = 5.5, d2 = 6.6;
cout << (int)d1 / i2 << endl;   // 0
cout << (int)(d1 / i2) << endl; // 0
```

```
// Test implicit type casting
d1 = i1; // int implicitly casts to double
cout << d1 << endl; // 4.0
```

```
// double truncates to int! (Warning?)
i2 = d2;
cout << i2 << endl; // 6
}
```

Operator precedence (17/17)

- Refers to the order an expression that contain more than one operators will be evaluated

Level	Operator	Order
Highest	++ -- (post fix)	Right to left
	sizeof() ++ -- (prefix)	Right to left
	* / %	Left to right
	+ -	Left to right
	< <= > >=	Left to right
	== !=	Left to right
	&&	Left to right
		Left to right
	? :	Left to right
	= ,+=, -=, *=, /=, ^=, %=, &=, =, <<=, >>=	Right to left
	,	Left to right

Debugging program Errors (1/2)

■ Errors

- ✓ The problems or the faults that occur in the program, which makes the behavior of the program abnormal
- ✓ Also known as the bugs or faults
- ✓ Detected either during the time of compilation or execution

■ Debugging

- ✓ The process of removing program bugs and correcting it

■ Types of errors

- ✓ There are three major errors in programming namely
 1. *Syntax errors – also include semantics error*
 2. *Logical errors*
 3. *Run-time errors*

Debugging program Errors (2/2)

■ Syntax errors

- ✓ When there is the violation of the grammatical (Syntax) rule.
- ✓ Detected at the compilation time.
- ✓ e.g. missing of semicolon, Identifier not declared

■ Logical Errors:

- ✓ Produced due to wrong logic of program. Tough to identify and even tougher to remove.
- ✓ Variables are used before initialized/assigned value
- ✓ Misunderstanding of priority and associativity of operators

■ Runtime Errors:

- ✓ Generated during execution phase due to mathematical or some operation generated at the run time.
- ✓ Division by zero,
- ✓ Square root of negative number,
- ✓ File not found, Trying to write a read only file

Formatted Input/output (1/2)

- Formatted console input/output functions are used for performing input/output operations at console and the resulting data is formatted and transformed.

Functions	Description
width(int width)	Using this function, we can specify the width of a value to be displayed in the output at the console.
fill(char ch)	Using this function, we can fill the unused white spaces in a value(to be printed at the console), with a character of our choice.
setf(arg1, arg2)	Using this function, we can set the flags, which allow us to display a value in a particular format.
peek()	The function returns the next character from input stream, without removing it from the stream.
ignore(int num)	The function skips over a number of characters, when taking an input from the user at console.
putback(char ch)	This function appends a character(which was last read by get() function) back to the input stream.
precision(int num_of_digits)	Using this function, we can specify the number of digits(num_of_digits) to the right of decimal, to be printed in the output.

Formatted Input/output (2/2)

```
//The fill() and setf() function defines  
//width of the next value to be  
//displayed in the output at the console.
```

```
#include<iostream>  
#include<iomanip>  
using namespace std;
```

```
int main()  
{  
char ch = 'a';
```

```
//calling the fill () or setfill() function  
// to fill the white spaces with a value of  
// a character of our choice  
//setfill() -- requires to include iomanip  
cout<<setfill('*')<<setw(5);  
//cout.fill('*')<<setw(5); -- alternative  
cout<<ch <<"\n";
```

```
//width of the next value to be  
//displayed in the output will  
//not be adjusted to 5 columns.
```

```
int i = 1;  
cout<<i;  
}
```

```
//The width() and setw() function defines  
//width of the next value to be  
//displayed in the output at the console.
```

```
#include<iostream>  
#include<iomanip>  
  
using namespace std;
```

```
int main()  
{  
char ch = 'a';
```

```
//Adjusting the width of the next  
//value to displayed to 5 columns.  
//use either setw(n) or cout.width(n)  
//setw() -- requires to include iomanip  
cout<<setw(5); //cout.width(5) -- alternative  
cout<<ch <<"\n";
```

```
int i = 1;  
//width of the next value to be  
//displayed in the output will  
//not be adjusted to 5 columns.  
cout<<i;  
}
```

Formatting codes (1/1)

■ Proper commenting

- Avoid improper/unnecessary and over comments
- Comment your codes liberally

■ Braces

- Place the beginning brace at the end of the line, and align the ending brace with the start of the statement

■ Indentation and whitespace

- Indent the body of a block by an extra 3 (or 4 spaces), according to its level and provide enough space/newline between statements, operators, operands/variables and other programming elements

■ Naming (Identifiers)

- Use self-contained identifier (variable naming)
 - e.g., row, col, size, xMax, numStudents.
- Avoid using single-alphabet and meaningless names, such as a, b, c, d except common one like i, j, x, y

Summary of C++ Statements (1/1)

- Prep-processor statements
- Declarative Statements
 - ✓ Variable declaration
 - ✓ Variable Initialization
 - ✓ Constant definition
- Executable statements
 - ✓ Input/output statements
 - ✓ Assignment statements
 - ✓ Expression
 - ✓ Selection statements
 - ✓ Loop statements
- Special Statements (break, continue, return, exit)

Reading Resources/Materials

- *Chapter 1, 2 & 3:* Problem Solving With C++ [10th edition, University of California, San Diego, 2018; Walter Savitch;
- *Chapter 2 & 3:* An Introduction to Programming with C++ (8th Edition), 2016 Cengage Learning; Diane Zak
- *Chapter 2:* C++ how to program, 10th edition, Global Edition (2017); P. Deitel , H. Deitel

Thank You
For Your Attention!!

Any Questions

