

Mid Term Syllabus

Scientific Computing



Mr. M Usman Mustafa
Department of Physics
Mid Term Syllabus



Introduction to programming

Problem solving is a process of identifying a problem and finding the best solution for it. We solve different problems every day, every problem is different in nature. In our life we are bound to solve problems.

Example1: If you are watching a news channel on your TV and you want to change it to a sports channel, you need to do something i.e. move to that channel by pressing that channel number on your remote. This is a kind of problem solving.

Now, broadly we can say that problem is a kind of barrier to achieve something and problem solving is a process to get that barrier removed by performing some sequence of activities.

A problem may be solved in different ways. One solution may be faster, less expensive and more reliable than other. It is important to select best suitable solution. Different strategies, techniques and tools are used to solve a problem. Computers are used to solve complex problems by developing computer programs. Computer programs contains different instruction for computer. A programmer write instructions and computer executes these instructions to solve a problem.

Different problem-solving techniques are as follows:

- **Program**
- **Algorithm**
- **Flow chart**

Program

A set of instructions that tells a computer what to do is called program. A computer works according to the given instructions in the program. Computer programs written in programming language. A person who develops a program is called programmer. Programmer uses programming languages or tools to write a program.

Algorithm

An algorithm is a step-by-step procedure to solve a problem. A formula or set of steps for solving a particular problem.

Example of Algorithm

Problem 1: Find the area of a Circle of radius r.

Inputs to the algorithm: Radius r of the Circle.

Expected output:

Area of the Circle

Algorithm:

Step1: Read/input the Radius r of the Circle

Step2: Area $\rightarrow \text{PI} * r * r$ // calculation of area

Step3: Print Area

Problem2: Write an **algorithm** to read two numbers and find their sum.

**Expected output:**

Sum of the two numbers

Inputs to the algorithm:

First num1.

Second num2.

Properties of Algorithm

- The given problem should be broken down into simple and meaningful steps.
- The step should be numbered sequentially.
- The step should be descriptive and written in simple English.

Algorithm is written in English called pseudo code. There is no standard to write pseudo code. It is used to specify program logic in an English like manner that is independent of any particular programming language.

Advantage of Algorithm

- Reduce complexity.
- Increase flexibility.
- Ease of understanding

Flowchart

Flow chart is combination of two words “flow” and “chart”. A chart consists of different symbols to display information about any program. Flow indicates the direction of processing that take place in program.







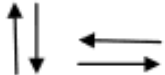
Flow chart is graphical representation of an algorithm. It is the way of visually presenting the data, operations performed on data and sequence of these operations. Flowchart is similar to layout plan of building. A designer draw the layout design of the building before construction of it. Similarly, programmer prefer to design the flow chart before writing the computer program.

Flow chart is designed according to the defined rules.

Flow chart symbols

There are 6 basic symbols commonly used in flowcharting of assembly language Programs: Terminal, Process, input/output, Decision, Connector and Predefined Process. This is not a complete list of all the possible flowcharting symbols; it is the ones used most often in the structure of Assembly language programming.



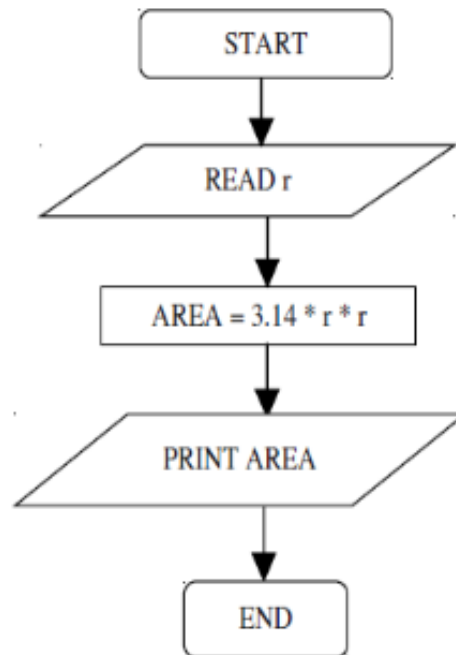
Symbol	Name	Function
	Process	Indicates any type of internal operation inside the Processor or Memory
	input/output	Used for any Input / Output (I/O) operation. Indicates that the computer is to obtain data or output results
	Decision	Used to ask a question that can be answered in a binary format (Yes/No, True/False)
	Connector	Allows the flowchart to be drawn without intersecting lines or without a reverse flow.
	Predefined Process	Used to invoke a subroutine or an Interrupt program.
	Terminal	Indicates the starting or ending of the program, process, or interrupt program
	Flow Lines	Shows direction of flow.

Limitations of Flowchart

The limitations of flow chart are as follows

1. It is difficult to draw flowchart for complex problems.
2. The flow chart has to redesigned if any change is required.

Example: Find the area of a circle of radius r .



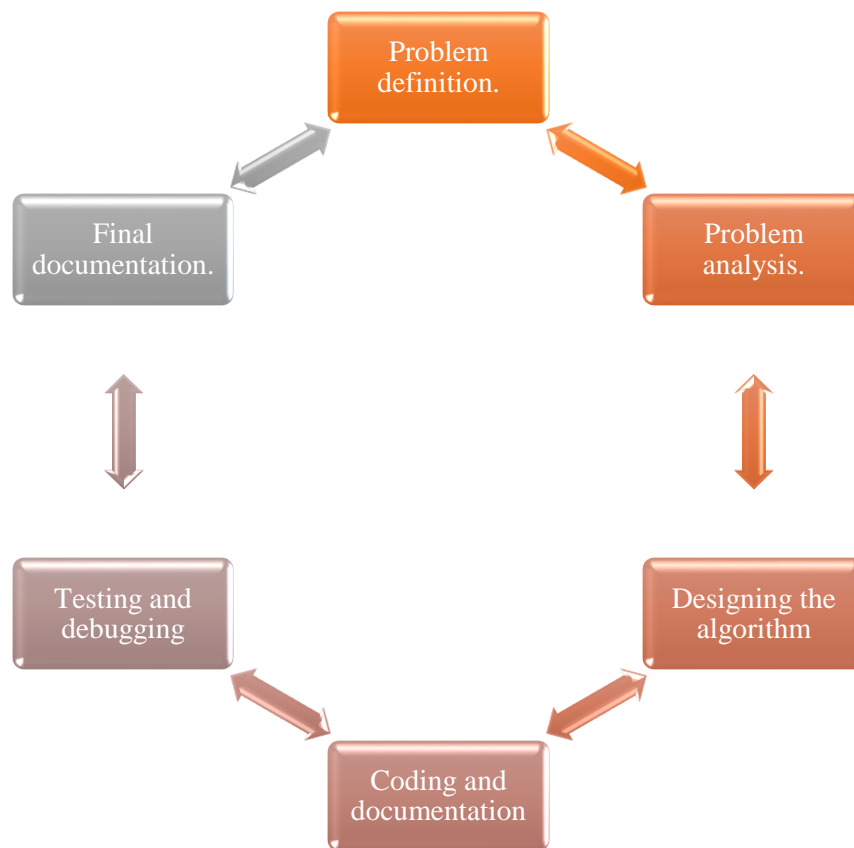
Difference between Algorithm and Flowchart

Flowchart	Algorithm
1. Standard symbols are used to design flowchart.	1. Simple English is used to write algorithm.
2. Flowchart is time consuming.	2. Algorithm is less time-consuming
3. It is difficult to modify.	3. It is easier to modify.
4. It is graphical representation of solution	4. It is a step-by-step procedure to solve a problem.

Program Development Process

A programmer has to go through the following stages to develop a computer program.

- Problem definition.
- Problem analysis.
- Designing the algorithm
- Coding and documentation
- Testing and debugging
- Final documentation.



Problem Definition

In this phase, we define the problem statement and we decide the boundaries of the problem. In this phase we need to understand the problem statement, what is our requirement, what should be the output of the problem solution. These are defined in this first phase of the program development life cycle.

Problem Analysis

In phase 2, we determine the requirements like variables, functions, etc. to solve the problem. That means we gather the required resources to solve the problem defined in the problem definition phase. We also determine the bounds of the solution.

Designing the Algorithm

During this phase, we develop a step by step procedure to solve the problem using the specification given in the previous phase. This phase is very important for program development. That means we write the solution in step by step statements.



Coding & Documentation

This phase uses a programming language to write or implement the actual programming instructions for the steps defined in the previous phase. In this phase, we construct the actual program. That means we write the program to solve the given problem using programming languages like C, C++, Java, etc.,

Testing & Debugging

During this phase, we check whether the code written in the previous step is solving the specified problem or not. That means we test the program whether it is solving the problem for various input data values or not. We also test whether it is providing the desired output or not.

Final Documentation

During this phase, the program is actively used by the users. If any enhancements found in this phase, all the phases are to be repeated to make the enhancements. That means in this phase, the solution (program) is used by the end-user. If the user encounters any problem or wants any enhancement, then we need to repeat all the phases from the starting, so that the encountered problem is solved or enhancement is added.

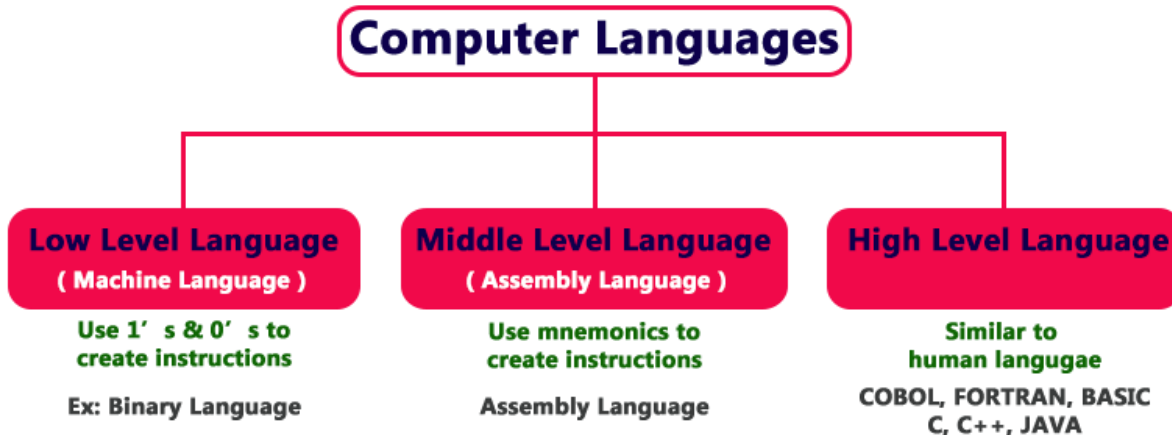


Computer Languages

Generally, we use languages like English, Hindi, etc., to make communication between two persons. That means when we want to make communication between two persons, we need a language through which persons can express their feelings. Similarly, when we want to make communication between user and computer or between two or more computers, we need a language through which user can give information to the computer and vice versa. When a user wants to give any instruction to the computer the user needs a specific language and that language is known as a computer language. The user interacts with the computer using programs and that programs are created using computer programming languages like C, C++, Java, etc., Computer languages are the languages through which the user can communicate with the computer by writing program instructions. Every computer programming language contains a set of predefined words and a set of rules (syntax) that are used to create instructions of a program.

Computer Languages Classification

Over the years, computer languages have been evolved from Low-Level to High-Level Languages. In the earliest days of computers, only Binary Language was used to write programs. The computer languages are classified as follows...



Low-Level Language (Machine Language)

Low-Level language is the only language which can be understood by the computer. **Binary Language** is an example of a low-level language. Low-level language is also known as **Machine Language**. The binary language contains only two symbols 1 & 0. All the instructions of binary language are written in the form of binary numbers 1's & 0's. A computer can directly understand the binary language. Machine language is also known as the **Machine Code**.



As the CPU directly understands the binary language instructions, it does not require any translator. CPU directly starts executing the binary language instructions and takes very less time to execute the instructions as it does not require any translation. Low-level language is considered as the First-Generation Language (1GL).

Advantages

- A computer can easily understand the low-level language.
- Low-level language instructions are executed directly without any translation.
- Low-level language instructions require very less time for their execution.

Disadvantages

- Low-level language instructions are very difficult to use and understand.
- Low-level language instructions are machine-dependent, that means a program written for a particular machine does not execute on another machine.
- In low-level language, there is more chance for errors and it is very difficult to find errors, debug and modify.

Middle-Level Language (Assembly Language)

Middle-level language is a computer language in which the instructions are created using symbols such as letters, digits and special characters. Assembly language is an example of middle-level language. In assembly language, we use predefined words called **mnemonics**. Binary code instructions in low-level language are replaced with mnemonics and operands in middle-level language. But the computer cannot understand mnemonics, so we use a translator called **Assembler** to translate mnemonics into binary language. Assembler is a translator which takes assembly code as input and produces machine code as output. That means, the computer cannot understand middle-level language, so it needs to be translated into a low-level language to make it understandable by the computer. Assembler is used to translate middle-level language into low-level language.

Advantages

- Writing instructions in a middle-level language is easier than writing instructions in a low-level language.
- Middle-level language is more readable compared to low-level language.
- Easy to understand, find errors and modify.

Disadvantages

- Middle-level language is specific to a particular machine architecture, that means it is machine-dependent.
- Middle-level language needs to be translated into low-level language.
- Middle-level language executes slower compared to low-level language.



High-Level Language

A high-level language is a computer language which can be understood by the users. The high-level language is very similar to human languages and has a set of grammar rules that are used to make instructions more easily. Every high-level language has a set of predefined words known as Keywords and a set of rules known as Syntax to create instructions. The high-level language is easier to understand for the users but the computer can not understand it. High-level language needs to be converted into the low-level language to make it understandable by the computer. We use **Compiler** or **interpreter** to convert high-level language to low-level language. Languages like COBOL, FORTRAN, BASIC, C, C++, JAVA, etc., are examples of high-level languages. All these programming languages use human-understandable language like English to write program instructions. These instructions are converted to low-level language by the compiler so that it can be understood by the computer.

Advantages

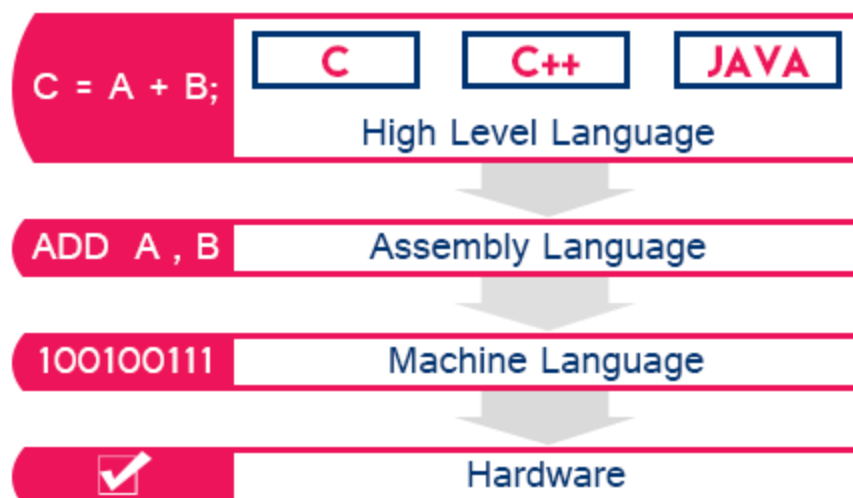
- Writing instructions in a high-level language is easier.
- A high-level language is more readable and understandable.
- The programs created using high-level language runs on different machines with little change or no change.
- Easy to understand, create programs, find errors and modify.

Disadvantages

- High-level language needs to be translated into low-level language.
- High-level language executes slower compared to middle and low-level language

Understanding Computer Languages

The following figure provides a few key points related to computer languages.



From the above figure, we can observe the following key points...

- The programming languages like C, C++, Java, etc., are written in High-level language which is more comfortable for the developers.
- A high-level language is closer to the users.
- Low-level language is closer to the computer. Computer hardware can understand only the low-level language (Machine Language).
- The program written in the high-level language needs to be converted to low-level language to make communication between the user and the computer.
- Middle-level language is not closer to both user and computer. We can consider it as a combination of both high-level language and low-level language.

Difference between low-level language and High level language.

High-Level Languages	Low-Level Languages
1. High-Level Languages are easy to learn and understand.	1. Low-Level Languages are challenging to learn and understand.
2. They are executed slower than lower level languages because they require a translator program.	2. They execute with high speed.
3. They allow much more abstraction.	3. They allow little or no abstraction.
4. They do not provide many facilities at the hardware level.	4. They are very close to the hardware and help to write a program at the hardware level.



5. For writing programs, hardware knowledge is not required.	5. For writing programs, hardware knowledge is a must.
6. The programs are easy to modify.	6. Modifying programs is difficult.
7. A single statement may execute several instructions.	7. The statements can be directly mapped to processor instructions.
8. BASIC, Perl, Pascal, COBOL, Ruby etc are examples of High-Level Languages.	8. Machine language and Assembly language are Low-Level Languages.

Language Processor

To express our idea, we use a suitable language. Same in the computer system, to make a dialogue, a language is required. The language is known as programming language. The programming language is generally used to give command or instruction to computer. It is required to translate into machine format. For a translation of programs, language processor is used. Computers understand instructions only when they are written in the machine language. Machine language is in binary form that is its instructions consist of zeros and ones.

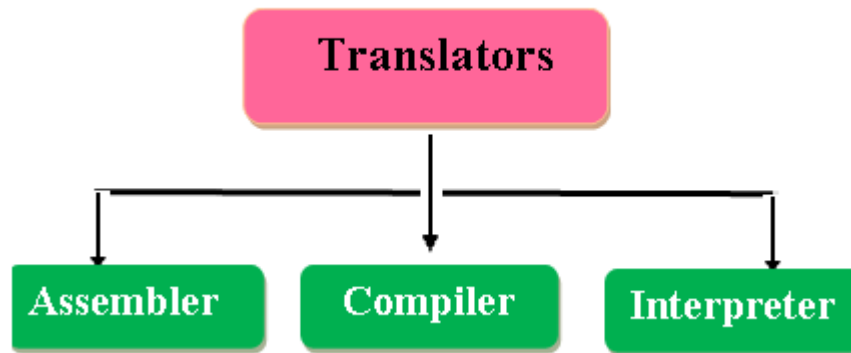
Definition of Language Processor

A language translator is a program which is used to translate an input program written in one programming language into another programming language (output program). Language processor is also called a **language translator**. A program written in any high-level programming language is called **the Source code or program**. To convert the source code into machine code is called the **object code or program**. A Translator translates the source program into the object program that the computer can understand and execute.

Types of Language Processor or Translators

In programming Language processors are three types:

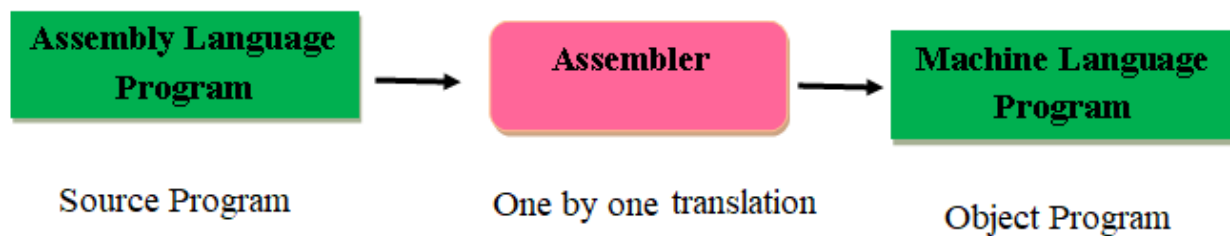
1. Assembler
2. Interpreter
3. Compiler



Assembler

Assembler is a translator which is used to translate the assembly language code into machine language code. Assembly language is a low-level programming language where we use the symbols called mnemonics in place of machine codes. The assembler performs a one to one mapping from mnemonic statement into machine codes and data. The translated program is called as object program.

How Assembler Works



Advantages of Assembler

- Assembler is Very fast translating assembly language to machine code as 1 to 1 relationship.
- Efficiency in execution just like machine level language.

Disadvantages of Assembler

- Assembly language is difficult to understand. it is a low-level programming language.
- It is difficult to maintain.

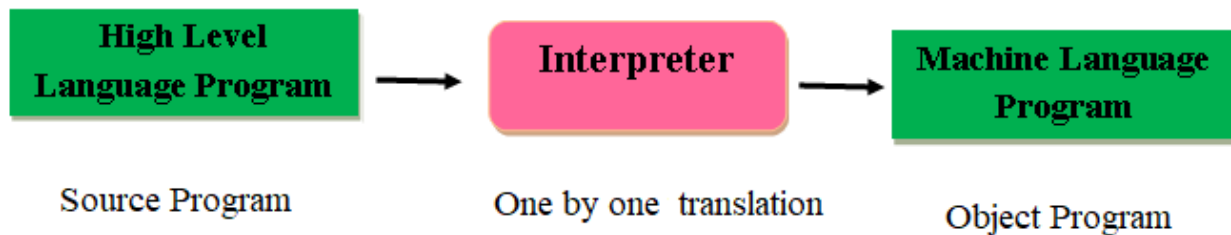
Interpreter

Interpreter converts the source program written in high level language into machine languages. An interpreter converts each statement of the program line by line into machine code. It means an



interpreter translates the one line at a time into machine language and executes it. Then moves towards the next line this goes on till the end of the program. If no error encounters. Interpreter stops and highlights the problem and will not move to next line when any errors are encountered. A interpreter requires a less storage space in primary memory than a compiler.

How Interpreter Works



Advantages of Interpreter

- Interpreter is that it makes easy to trace out and correct errors in the source program.
- Interpreters over compilers are that an error is found immediately.

Disadvantages of Interpreter

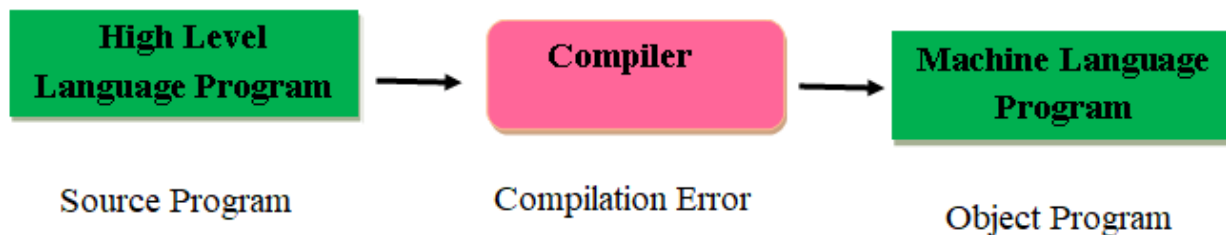
- It is a time-consuming process of translating and executing statements one by one.
- Programs execution is slow.

Compiler

The purpose of compiler is same as interpreter but unlike interpreters which translate the source program line by line, compiler are the translators, which translate the entire program into machine codes. If source program contains errors, the compiler highlights a list of errors at the end of the execution of the program. i.e. a compiler translates the whole program before execution. A compiler is a larger program and occupies more memory space. It is costlier than interpreter



How Compiler Works



Advantages of compiler

- Producers and executable file, and therefore the program can be run without need of the source code.
- A compiler converts a high-level program that can be executed many times.

Disadvantages of compiler

- It is slow to execute as you have to finish the whole program.
- It is not easy to debug as errors are shown at the end of the execution.

Difference between Compiler and Interpreter

S. No.	Compiler	Interpreter
1	A compiler takes the whole program as a single unit and compiles it at once	1. Interpreter each line in translated or converted one by one and executed
2	It stores an object file.	2. It does not store an object file.
3	Occupies more memory space	3. Occupies less memory space
4	Program execution is very fast.	4. Program execution is slow.
5	Debugging is harder	5. Debugging is easier
6	Translator program is required to translate the program each time you want to run the program.	6. Translator program is not required to translate the program each time you want to run the program.



S. No.	Compiler	Interpreter
7	More useful for commercial purpose	7. More useful for learning purpose
9	Compiler are good for a very long program	8. Interpreter is good for small programs.
10	Example: C compiler, PASCAL compiler, FORTRAN compiler etc.	9. Example: Basic interpreter.



Introduction to C++

Historical Perspective

The C++ programming language was created by Bjarne Stroustrup and his team at Bell Laboratories (AT&T, USA) to help implement simulation projects in an object-oriented and efficient way. The earliest versions, which were originally referred to as “C with classes,” date back to 1980. As the name C++ implies, C++ was derived from the C programming language:

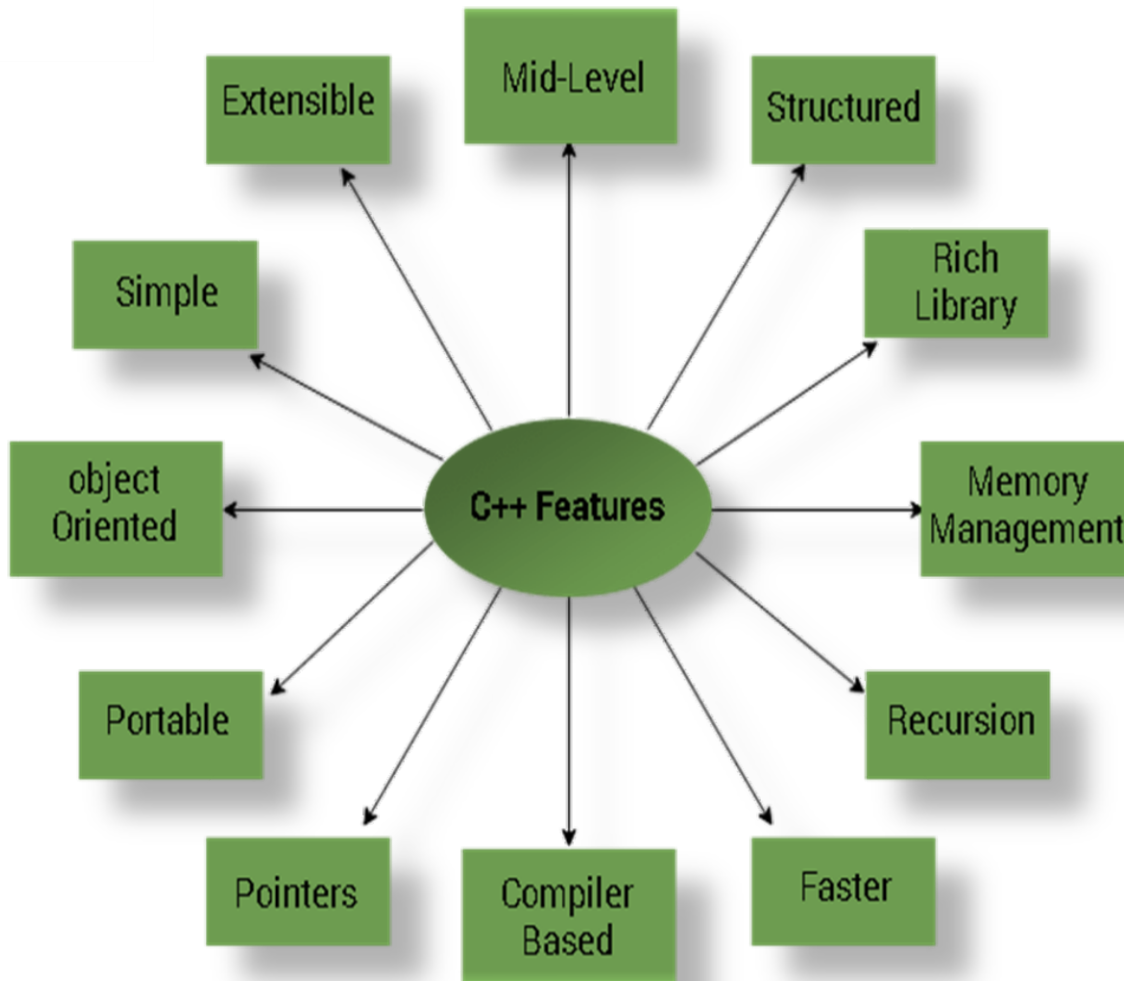
++ is the increment operator in C. As early as 1989 an ANSI Committee (American National Standards Institute) was founded to standardize the C++ programming language. The aim was to have as many compiler vendors and software developers as possible agree on a unified description of the language in order to avoid the confusion caused by a variety of dialects. In 1998 the ISO (International Organization for Standardization) approved a standard for C++ (ISO/IEC 14882)

Characteristics of C++

- Convenient Language
- Well-Structure Language
- Case Sensitive
- Machine Independence
- Object Oriented
- C compatibility
- Modular programming

Basic Structure of C++

- The format of writing program in C++ is called structure of C++. The structure of C++ program is very flexible. It increases power of language.
- Preprocessor Directives.
- Main function ()
- Program Body



Preprocessor Directives

Preprocessor directives are lines included in the code of our programs that are not program statements but directives for the preprocessor. These lines are always preceded by a pound sign (#). The preprocessor is executed before the actual compilation of code begins, therefore the preprocessor digests all these directives before any code is generated by the statements. These preprocessor directives extend only across a single line of code. As soon as a newline character is found, the preprocessor directive is considered to end. No semicolon (;) is expected at the end of a preprocessor directive.

Include

The first preprocessor directive which we will cover is one which you should already know quite a lot about! We've been using the `#include` directive ever since our first C++ program, but we've



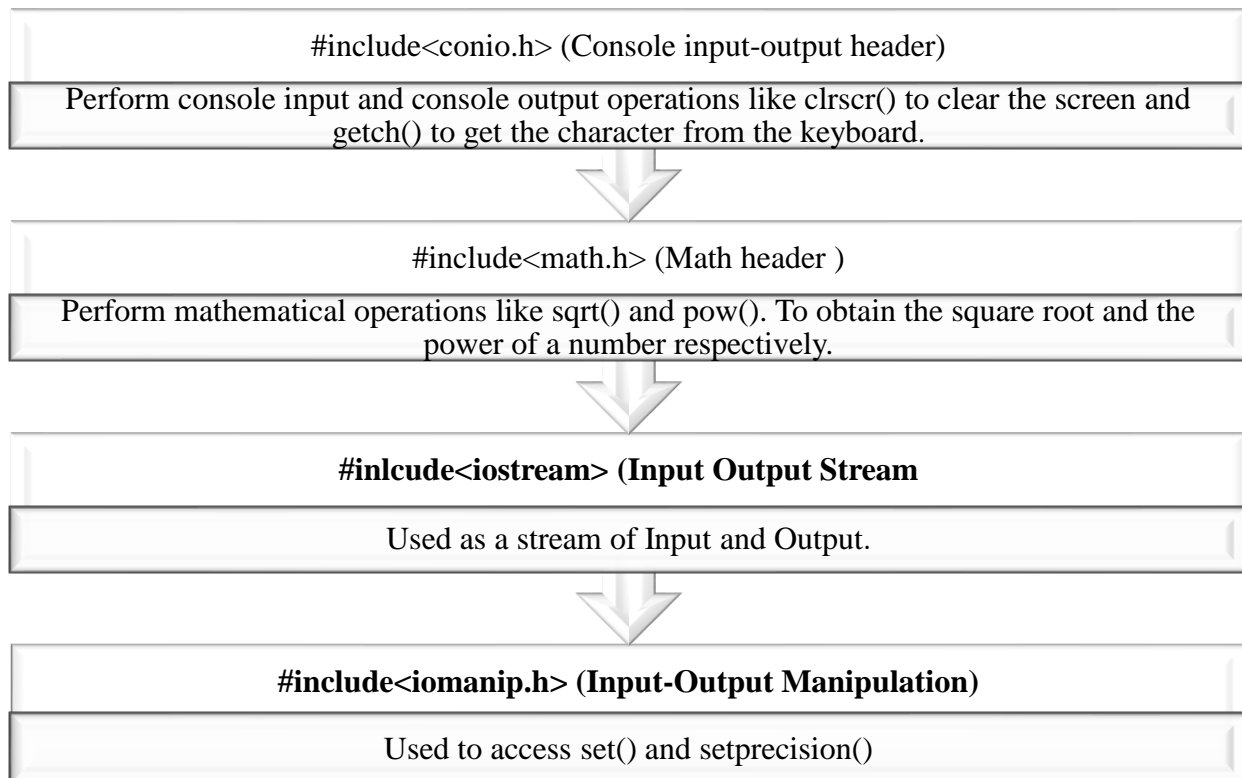
sort of glossed over what it does. When the preprocessor finds an 'include' directive, it fetches the file specified and dumps it in the place of the directive - so in the case of `<iostream>`, the preprocessor looks into the directories where it knows it might find these kind of files, and then dumps the file (e.g. `iostream.h`) where the directive is present.

Files specified using triangular brackets (e.g. `<iostream>`) will be looked for in the directories that the compiler has "noted down", and files specified using double quotes will be looked for in the same directory as your project. As such, you can create your own custom '.cpp' and '.h' files, and include these using double quotes with the include directive.

Syntax of Header

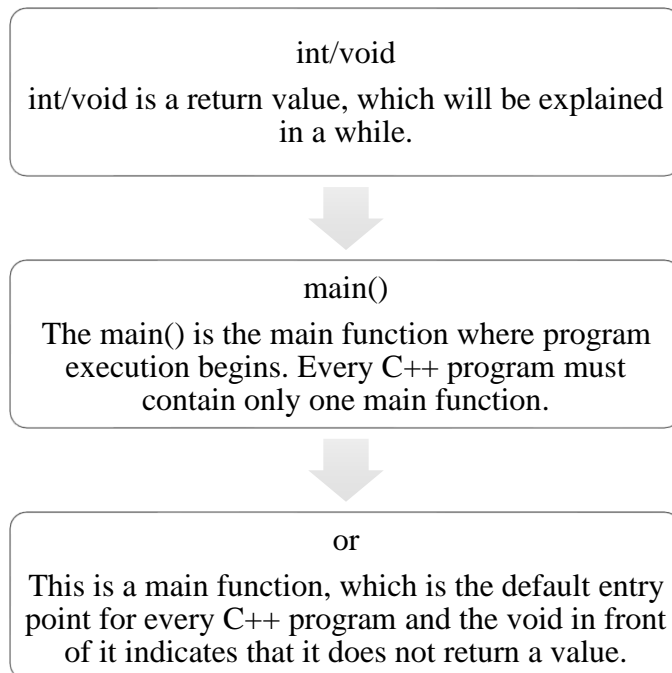
`#include<Header file.h>`

C++ Header Files



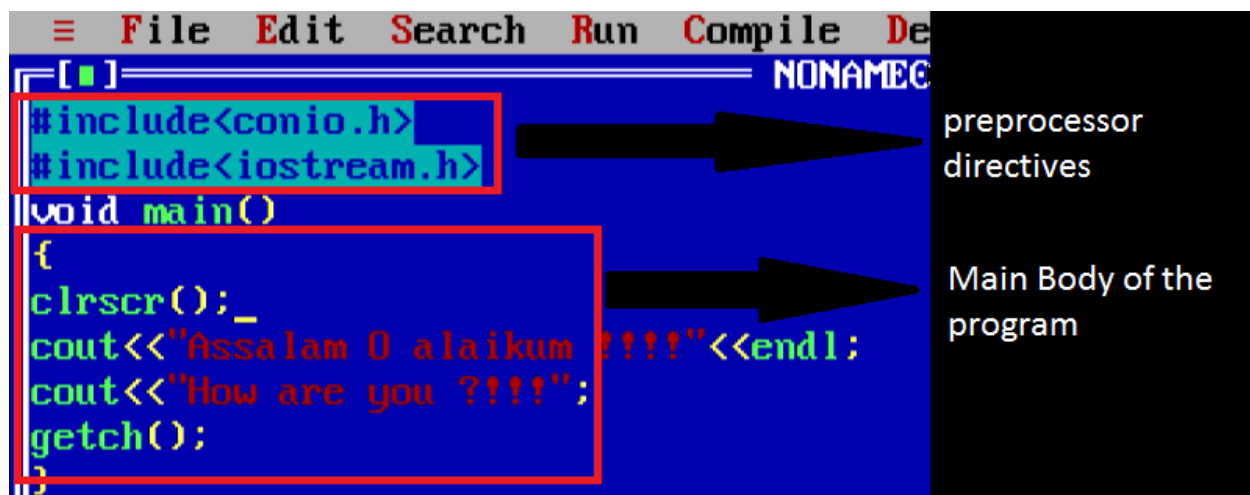


Main Function



Main Body of Program

Main body of C++ consist of statements. Start and end with the curly braces. As shown in fig.





Creating and Running C++

Program

Generally, the programs created using programming languages like C, C++, Java, etc., are written using a high-level language like English. But, the computer cannot understand the high-level language. It can understand only low-level language. So, the program written in the high-level language needs to be converted into the low-level language to make it understandable for the computer. This conversion is performed using either Interpreter or Compiler.

Popular programming languages like C, C++, Java, etc., use the compiler to convert high-level language instructions into low-level language instructions. A compiler is a program that converts high-level language instructions into low-level language instructions. Generally, the compiler performs two things, first it verifies the program errors, if errors are found, it returns a list of errors otherwise it converts the complete code into the low-level language. To create and execute C++ programs in the Windows Operating System, we need to install Turbo C++ software. We use the following steps to create and execute C++ programs in Windows.

Step 1

Create Source Code

Write program in the Editor &
save it with **.cpp** extension

Step 2

Compile Source Code

Press **Alt + F9** to compile

Step 3

Run Executable Code

Press **Ctrl + F9** to run

Step 4

Check Result

Press
Alt + F5
to open UserScreen

Creating a Source Code

Source code is a file with C++ programming instructions in a high-level language. To create source code, we use any text editor to write the program instructions. The instructions written in the source code must follow the C++ programming language rules. The following steps are used to create a source code file in Windows.

- Click on the **Start** button



- Select **Run**
- Type **cmd** and press Enter
- Type **cd c:\TC\bin** in the command prompt and press **Enter**
- Type **TC** press **Enter**
- Click on **File -> New** in C Editor window
- Type the **program**
- Save it as **Filename** (Use shortcut key **F2** to save)

Compile Source Code (Alt + F9)

The compilation is the process of converting high-level language instructions into low-level language instructions. We use the shortcut key **Alt + F9** to compile a C program in **Turbo C++**. The compilation is the process of converting high-level language instructions into low-level language instructions.

Whenever we press **Alt + F9**, the source file is going to be submitted to the Compiler. On receiving a source file, the compiler first checks for the Errors. If there are any Errors then compiler returns List of Errors, if there are no errors then the source code is converted into **object code** and stores it as a file with **.obj** extension. Then the object code is given to the **Linker**. The Linker combines both the **object code** and specified **header file** code and generates an **Executable file** with a **.exe** extension.

Executing / Running Executable File (Ctrl + F9)

After completing compilation successfully, an executable file is created with a **.exe** extension. The processor can understand this **.exe** file content so that it can perform the task specified in the source file.

We use a shortcut key **Ctrl + F9** to run a C program. Whenever we press **Ctrl + F9**, the **.exe** file is submitted to the **CPU**. On receiving **.exe** file, **CPU** performs the task according to the instruction written in the file. The result generated from the execution is placed in a window called **User Screen**.

Check Result (Alt + F5)

After running the program, the result is placed into **User Screen**. Just we need to open the User Screen to check the result of the program execution. We use the shortcut key **Alt + F5** to open the User Screen and check the result.



C++ Tokens

Every C++ program is a collection of instructions and every instruction is a collection of some individual units. Every smallest individual unit of a c++ program is called token. Every instruction in a C++ program is a collection of tokens. Tokens are used to construct c programs and they are said to be the basic building blocks of a C++ program. In a c program tokens may contain the following.

1. Keywords
2. Identifiers
3. Operators
4. Special Symbols
5. Constants
6. Strings
7. Data values

In a C++ program, a collection of all the keywords, identifiers, operators, special symbols, constants, strings, and data values are called tokens.

Debugging in Turbo C++

Error is an illegal operation performed by the user which results in abnormal working of the program. Programming errors often remain undetected until the program is compiled or executed. Some of the errors inhibit the program from getting compiled or executed. Thus, errors should be removed before compiling and executing. The most common errors can be broadly classified as follows.

Types of errors

There are basically three types of errors that you must contend with when writing computer programs:

- Syntax errors
- Runtime errors
- Logic errors

Generally speaking, the errors become more difficult to find and fix as you move down the above list.



Syntax errors

In effect, syntax errors represent *grammar errors* in the use of the programming language. Common examples are:

- Misspelled variable and function names
- Missing semicolons
- Improperly matches parentheses, square brackets, and curly braces
- Incorrect format in selection and loop statements

Runtime errors

Runtime errors occur when a program with no syntax errors asks the computer to do something that the computer is unable to reliably do. Common examples are:

- Trying to divide by a variable that contains a value of zero
- Trying to open a file that doesn't exist

There is no way for the compiler to know about these kinds of errors when the program is compiled.

Logic errors

Logic errors occur when there is a design flaw in your program. Common examples are:

- Multiplying when you should be dividing
- Adding when you should be subtracting
- Opening and using data from the wrong file
- Displaying the wrong message

Comments

The program that you write should be clear not only to you, but also to the reader of your program. Part of good programming is the inclusion of comments in the program. Typically, comments can be used to identify the authors of the program, give the date when the program is written or modified, give a brief explanation of the program, and explain the meaning of key statements in a program. In the programming examples, for the programs that we write, we will not include the date when the program is written, consistent with the standard convention for writing such books.

Comments are for the reader, not for the compiler. So when a compiler compiles a program to check for the syntax errors, it completely ignores comments. Comments are shown in green.

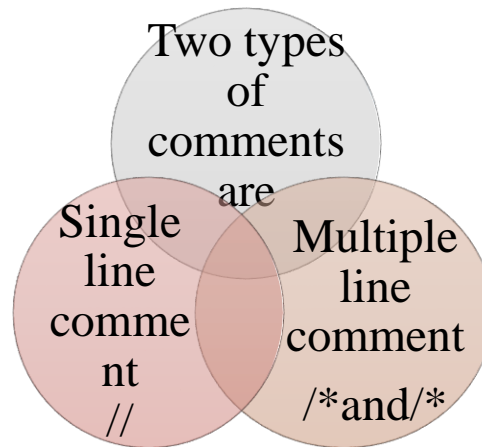


The program in Example 2-1 contains the following comments:

// This is a C++ program. It prints the sentence:

// Welcome to C++ Programming.

Types of Comments



Example

```

#include<iostream.h>
#include<conio.h>
void main()
{
    int sum=0,n1,n2;
    clrscr();
    cout<<"Enter two number: ";
    cin>>n1>>n2;    // Input two numbers    ——— Single Line Comment
    sum=n1+n2;
    /*
       Enter two number
       add these number
       and store in sum
    */
    cout<<"Sum: "<<sum;
    getch();
}
  
```

endl Manipulator

- it is important and most common used manipulator.



- endl manipulator is used to shift the statement in the next line.
- It stands for “endl of line”.
- Syntax of this operator is as <<endl;

Example

```
≡ File Edit Search Run Compile Debug Pr  
[■] NONAME00.CPP  
#include<iostream.h>  
#include<conio.h>  
void main()  
{  
clrscr();  
cout<<"my name is iqra"<<endl;  
cout<<"this is my first program";  
getch();  
}
```

Exercise

Write a C++ program that outputs the following text on screen:

- Oh what a happy day!
- This is my first Program.
- My name is (your name). and roll no. is (roll no).

The following program contains several errors:



```
*/ Now you should not forget your glasses //
```

```
#include <iostream,h>
```

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

```
Void main()
```

```
{
```

```
cout << "If this text",
```

```
cout >> " appears on your display, "
```

```
cout << 'you can pat yourself on '
```

```
<< " the back!" << ;
```

```
getch();
```

```
)
```

Resolve the errors and run the program to test your changes



Tokens in C++

(Keywords, Identifiers, Constants, Strings, Operators, Special Symbols)

Tokens and keywords are undoubtedly notable features of C++. Tokens act as building blocks of a program. Just like a living cell is the smallest possible unit of life, tokens in C++ are referred to as the smallest individual units in a program. Keywords in C++ help the user in framing statements and commands in a language. Each keyword conveys a unique connotation to the compiler to perform a specific task. Just like the combination of words helps us in framing sentences, the combination of keywords helps us in framing statements to perform logical operations in a programming language. Simply combining keywords wouldn't help to serve the purpose.

As we need to use proper grammar to form a meaningful sentence, we need to be well-acquainted with the **syntax of C++** to instruct the compiler what to do. If these statements are not formed in a logical manner, they would sound gibberish and you would get a compilation error. Let's discuss the concept of Tokens with Character set in C++ in detail.



C++ Character Set

C++ Character set is basically a set of valid characters that convey a specific connotation to the compiler. We use characters to represent letters, digits, special symbols, white spaces, and other characters. The C++ character set consists of 3 main elements. They are:

1. **Letters:** These are alphabets ranging from A-Z and a-z (both uppercase and lowercase characters convey different meanings)
2. **Digits:** All the digits from 0 – 9 are valid in C++.



3. **Special symbols:** There are a variety of special symbols available in C++ like mathematical, logical and relational operators like +, -, *, /, \, ^, %, !, @, #, ^, &, (,), [,], ; and many more.

Tokens in C++

As discussed earlier, **tokens in C++ are the smallest individual unit of a program.**

The following tokens are available in C++ which are similar to that seen in C with the addition of certain exclusive keywords, strings, and operators:

- Keywords
- Identifiers
- Constants
- Strings
- Special symbols
- Operators

C++ Keywords

Keywords in C++ refer to the pre-existing, reserved words, each holding its own position and power and has a specific function associated with it. It is important to note that we cannot use C++ keywords for assigning variable names as it would suggest a totally different meaning entirely and would be incorrect. Here is a list of keywords available in C++ according to the latest standards:

alignas	alignof	asm	auto	bool	break
case	catch	char	char16_t	char32_t	class
const	constexpr	const_cast	continue	decltype	default
delete	double	do	dynamic_cast	else	enum
explicit	export	extern	FALSE	float	for
friend	goto	if	inline	int	long
mutable	namespace	new	noexcept	nullptr	operator
private	protected	public	register	reinterpret_cast	return
short	signed	sizeof	static	static_assert	static_cast
struct	switch	template	this	thread_local	throw
TRUE	try	typedef	typeid	typename	union
unsigned	using	virtual	void	volatile	wchar_t
while	–	–	–	–	–

Table 1 C++ reserved words. C++ reserves these words for specific purposes in program construction. None of the words in this list may be used as an identifier; thus, you may not use any of these words to name a variable



C++ Identifiers

C++ allows the programmer to assign names of his own choice to variables, arrays, functions, structures, classes, and various other data structures called identifiers. The programmer may use the mixture of different types of character sets available in C++ to name an identifier.

Rules for C++ Identifiers

There are certain rules to be followed by the user while naming identifiers, otherwise, you would get a compilation error. These rules are:

1. **First character:** The first character of the identifier in C++ should positively begin with either an alphabet or an underscore. It means that it strictly cannot begin with a number.
2. **No special characters:** C++ does not encourage the use of special characters while naming an identifier. It is evident that we cannot use special characters like the *exclamatory mark* or the “@” symbol.
3. **No keywords:** Using keywords as identifiers in C++ is strictly forbidden, as they are reserved words that hold a special meaning to the C++ compiler. If used purposely, you would get a compilation error.
4. **No white spaces:** Leaving a gap between identifiers is discouraged. White spaces incorporate blank spaces, newline, carriage return, and horizontal tab.
5. **Word limit:** The use of an arbitrarily long sequence of identifier names is restrained. The name of the identifier must not exceed 31 characters, otherwise, it would be insignificant.
6. **Case sensitive:** In C++, uppercase and lowercase characters connote different meanings.

Here is a table which illustrates the valid use of Identifiers:

Special Symbols

Apart from letters and digits, there are some special characters in C++ which help you manipulate or perform data operations. Each special symbol has a specific meaning to the C++ compiler.

Here is a table which illustrates some of the special characters in C:



Identifier Name	Valid or Invalid	Correction or alternative, if invalid	Elucidation if invalid
5th_element	Invalid	element_5	It violates Rule 1 as it begins with a digit
_delete	Valid	–	–
school.fee	Invalid	school_fee	It violates Rule 2 as it contains a special character ‘.’
register[5]	Invalid	Register[5]	It violates Rule 3 as it contains a keyword
Student[10]	Valid	–	–
employee name	Invalid	employee_name	It violates Rule 4 as it contains a blank space
perimeter()	Valid	–	–

Special Character	Trivial Name	Function
[]	Square brackets	The opening and closing brackets of an array symbolize single and multidimensional subscripts.
()	Simple brackets	The opening and closing brackets represent function declaration and calls, used in print statements.
{ }	Curly braces	The opening and closing curly brackets to denote the start and end of a particular fragment of code which may be functions, loops or conditional statements
,	Comma	We use commas to separate more than one statements, like in the declaration of different variable names
#	Hash / Pound / Preprocessor	The hash symbol represents a preprocessor directive used for denoting the use of a header file



*	Asterisk	We use the asterisk symbol in various respects such as to declare pointers, used as an operand for multiplication
~	Tilde	We use the tilde symbol as a destructor to free memory
.	Period / dot	The use the dot operator to access a member of a structure

C++ Operators

Operators are tools or symbols which are used to perform a specific operation on data. Operations are performed on operands. Operators can be classified into three broad categories according to the number of operands used.

Unary: It involves the use of one a single operand. For instance, '!' is a unary operator which operates on a single variable, say 'c' as !c which denotes its negation or complement.

Binary: It involves the use of 2 operands. They are further classified as:

- Arithmetic
- Relational
- Logical
- Assignment
- Bitwise
- Conditional

Ternary

Summary

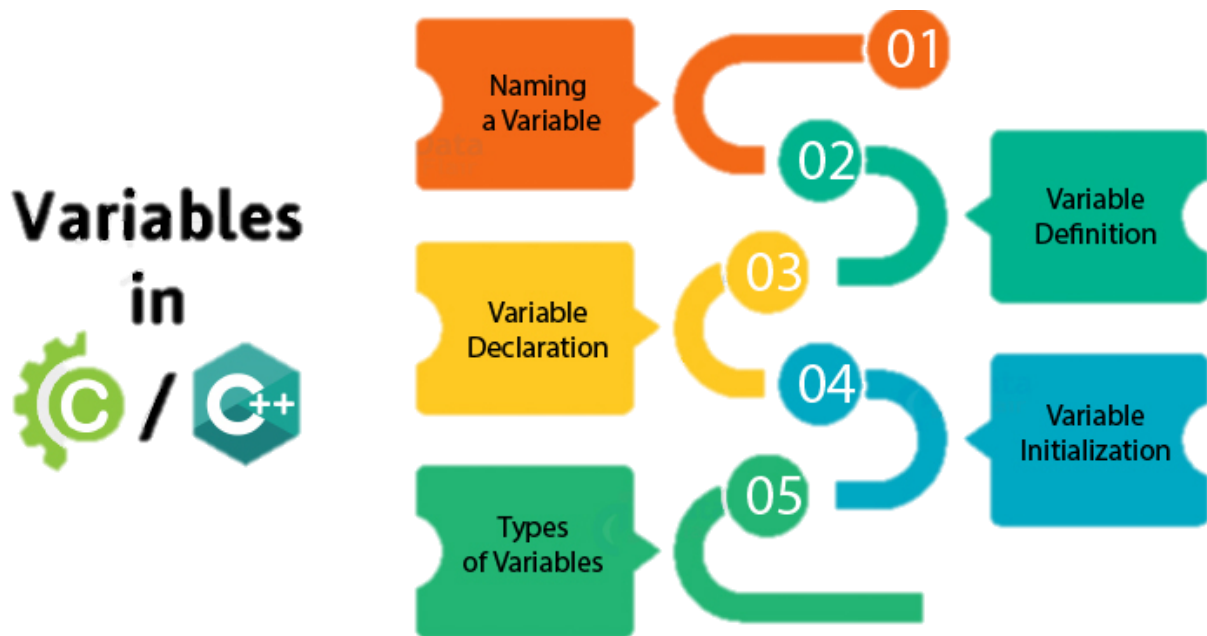
Tokens in C++ are called as building blocks of a program. All the sub-parts of tokens are: keywords, identifiers, constants, strings, special symbols, and operators equally important and play a vital role in framing the programs in C++. Every part is discussed well, refer all the links above and get a detailed description of each part.



Variables in C++

When we hear about variables, we tend to think about mathematical formulas and questions in which variables are unknown values or quantities limited to some numbers. But, in the C/C++ programming language, variables connote a different meaning. In mathematical terms, we use representations of variables such as x or y that indicates an unknown value that we are supposed to find. Here, *variables in the C and C++ programming language are the basic units, which help us to build C Programs*. So, without wasting time, start exploring C/C++ variables.

A C/C++ program performs many tasks and operations that help you to resolve many problems by storing values into computer memory. But, how does the compiler understand the names given to these values? A variable helps to specify the existence of these values by defining and declaring them.



In simple words, *a variable is a storage space associated with a unique name to identify them*. When you want to store some data on your system in the computer memory, is it possible for you to be able to remember these memory addresses? The answer is no, and that is the reason why we use variables.

Variables in C/C++ programming help us to store values depending upon the size of the variable. With the help of variables, we can decide what amount and type of data store in a variable. When you assign a data type and name to some space in the memory, variables are defined.

Variables reserve some memory in the storage space, that you can access later in the program. By declaring a variable, you inform the operating system to reserve memory indicated by some name. i.e. variable name.



Naming a Variable in C/C++

You need to follow some rules, before naming a variable in C and C++:

1. A variable must not start with a digit.
2. A variable can begin with an alphabet or an underscore.
3. Variables in C and C++ are case-sensitive which means that uppercase and lowercase characters are treated differently.
4. A variable must not contain any special character or symbol.
5. White spaces are not allowed while naming a variable.
6. Variables should not be of the same name in the same scope.
7. A variable name cannot be a keyword.
8. The name of the variable should be unique.

Let's see some examples of both valid and invalid variable names.

Valid variable names

- Ticketdata
- _ticketdata
- ticket_data

Invalid variable names

- 56ticketdata
- ticket@data
- ticket data

Variable Definition

A variable definition in C and C++ defines the variable name and assigns the data type associated with it in some space in computer memory. After giving its definition, this variable can be used in the program depending upon the scope of that variable. By defining a variable, you indicate the name and data type of the variable to the compiler. The compiler allocates some memory to the variable according to its size specification.

Data Types in C and C++

A programming language cannot work without a wide range of data types as each type has its own significance and utility to perform various tasks. Here, reasons are mentioned why we require different data types in C and C++ Programming:

- At the time of the variable declaration, it becomes convenient for the user to distinguish which type of data variable stores.
- It makes it clear for the user to identify the return value of the function based on the data type.



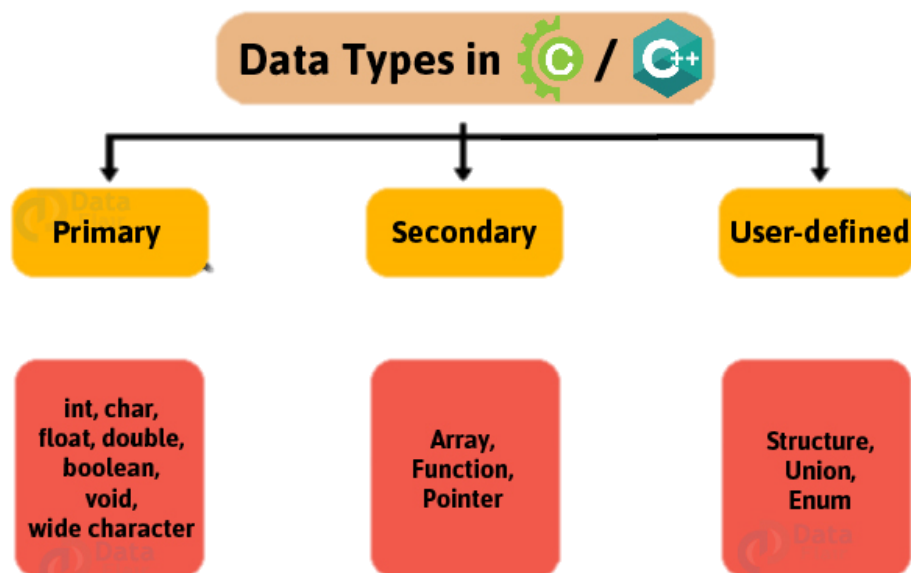
- If parameters are passed to the function, it becomes easy for the user to give input according to the given format.

Data types in C and C++ refer to the *characteristics of data stored into a variable*. For instance, while working with mathematical problems, in order to simplify things for us, we look for a specific type of data, let's say, we want to find the factorial of a number. We know that only for whole numbers, the factorial of that number exists which is also a whole number. In order to eliminate all scopes of errors and reduce run-time of the program, we would preferably assign such a data type to the input and output such that it only covers the range of whole numbers.

Clearly, from the above discussion, you might have inferred that memory occupied by different data types would be different. Therefore, a different amount of space in the computer memory would be allocated for them and hence the run time of the program would be reduced, increasing the efficiency of the program.

Types of Data Types in C and C++

According to the **conventional classification**, these are data types in C++



Primary Data Type

Primary (Fundamental) data types in C programming includes the 4 most basic data types, that is:

- **int**: It is responsible for storing integers. The memory it occupies depends on the compiler (32 or 64 bit). In general, int data type occupies 4 bytes of memory when working with a 32-bit compiler.
- **float**: It is responsible for storing fractions or digits up to 7 decimal places. It is usually referred to as a single-precision floating-point type. It occupies 4 bytes of memory



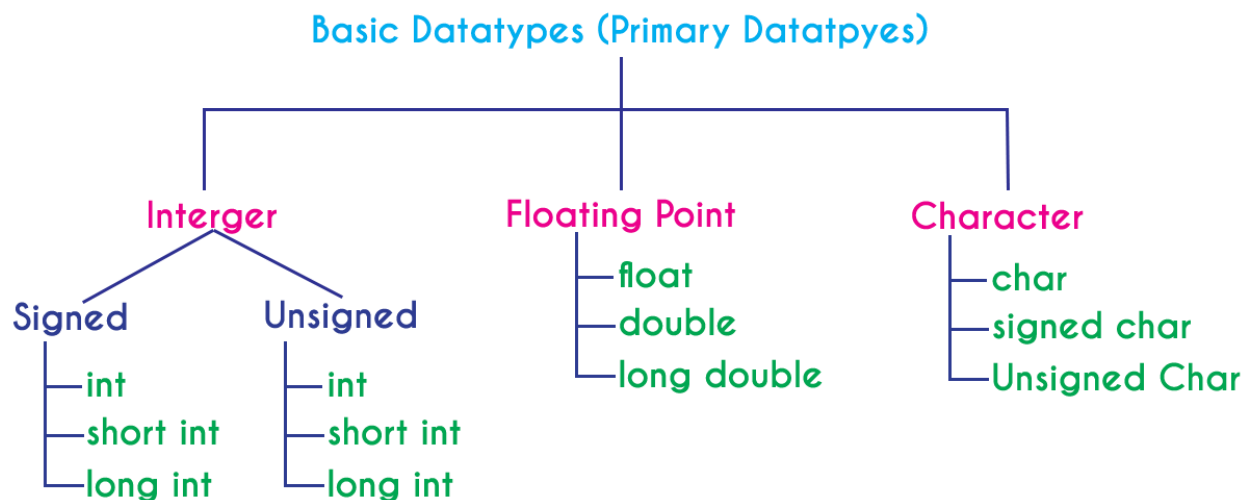
- **char:** It can be used to store a set of all characters which may include alphabets, numbers and special characters. It occupies 1 byte of memory being the smallest addressable unit of a machine containing a fundamental character set.
- **double:** It is responsible for storing fractions or digits up to 15-16 decimal places. It is usually referred to as a double-precision floating-point type.
- **void (Null) data type:** It indicates zero or no return value. It is generally used to assign the null value while declaring a function.

In C++, in addition to the primary data types available in C, there are few more data types available in the C++ programming language. They are:

- **bool:** It refers to a boolean/logical value. It can either be true or false.
- **wchar_t:** It refers to a wide character whose size is either 2 or 4 bytes. It is similar to the char data type but the only difference is the space occupied in the computer memory.
- **string:** Instead of declaring an array of characters to enter a string data type, C++ gives you the provision to declare the “string” data type.

Along with data types, comes modifiers. Modifiers basically alter the function of a data type and make it more specific by the inclusion of additional provisions. These include:

- **Signed:** It is used to store zero, positive or negative values.
- **Unsigned:** It can store only zero or positive values.
- **Long:** It is used to store large integer numbers. The size of the long type is 8 bytes.
- **Short:** It is used to store small integer numbers. Its size is of 2 Bytes.



Here is a table that would specify the range of various data primary data types along with their modifiers. It is in accordance with a 32-bit compiler.



Data Type	Memory (In Bytes)	Range
int	4	-2,147,483,648 to 2,147,483,647
short int	2	-32,768 to 32,767
unsigned int	4	0 to 4,294,967,295
unsigned short int	2	0 to 65,535
long int	4	-2,147,483,648 to 2,147,483,647
unsigned long int	4	0 to 4,294,967,295
long long int	8	-(2 ⁶³) to (2 ⁶³)-1
unsigned long long int	8	0 to 18,446,744,073,709,551,615
char	1	-128 to 127
Signed char	1	-128 to 127
Unsigned char	1	0 to 255
float	4	–
double	8	–
long double	12	–

Here is a table that would help you to explore a wide range of format specifiers used for various purposes.

	Integer	Floating Point	Double	Character
What is it?	Numbers without decimal value	Numbers with decimal value	Numbers with decimal value	Any symbol enclosed in single quotation
Keyword	int	float	double	char
Memory Size	2 or 4 Bytes	4 Bytes	8 or 10 Bytes	1 Byte
Range	-32768 to +32767 (or) 0 to 65535 (Incase of 2 bytes only)	1.2E - 38 to 3.4E + 38	2.3E-308 to 1.7E+308	-128 to + 127 (or) 0 to 255
Type Specifier	%d or %i or %u	%f	%ld	%c or %s
Type Modifier	short, long signed, unsigned	No modifiers	long	signed, unsigned
Type Qualifier	const, volatile	const, volatile	const, volatil	const, volatile



Rules for Defining Variables in C and C++

1. Must contain datatype of that variable.

Example: `int start;`

`float width;`

`char choice;`

2. The variable name should follow all the rules of the naming convention.
3. After defining the variable, terminate the statement with a semicolon otherwise it will generate a termination error. **Example:** `int sum;`
4. The variable with the same data type can work with a single line definition. **Example:** `float height, width, length;`

Defining Variables in C and C++ with Example

```
int var;
```

Here, a variable of integer type with the variable name `var` is defined. This variable definition allocates memory in the system for `var`.

Another example,

```
char choice;
```

When we define this variable named `choice`, it allocates memory in the storage space according to the type of *data type in C*, i.e., character type.

Variable Declaration

There is a huge difference between defining a variable and declaring a variable.

By declaring a variable in C and C++, we simply tell the compiler that this variable exists somewhere in the program. But the declaration does not allocate any memory for that variable.

Declaration of variable informs the compiler that some variable of a specific type and name exists.

The definition of variable allocates memory for that variable in the program.

So, it is safe to say that the variable definition is a combination of declaration and memory allocation. A variable declaration can occur many times but variable definition occurs only once in a program, or else it would lead to wastage of memory.



variable
declaration

```
int x = 27;
```

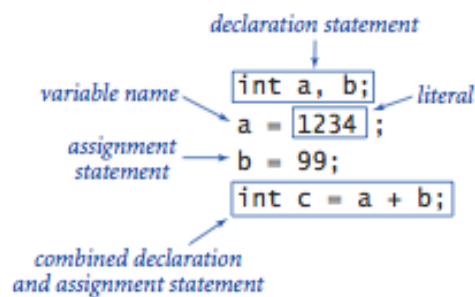
variable
assignment

Variable Initialization

Variable initialization means *assigning some value to that variable*. The initialization of a variable and declaration can occur in the same line.

```
int demo = 23;
```

By this, you initialize the variable demo for later use in the program.



Summary

In this tutorial, we focused on all significant points related to variables. Follow these rules for variables in C and C++:

1. Every variable uses some memory in the storage space when it is defined not declared.
2. Initializing a variable can be done along with a definition.
3. There are various types of variables that C support and these variables are categorized based on their scope.



Input & Output in C++

In C++, **input and output (I/O) operators** are used to take **input** and display **output**. The **operator** used for taking the **input** is known as the extraction or get from **operator** (>>), while the **operator** used for displaying the **output** is known as the insertion or put to **operator** (<<).

The process of getting something to computer is known as **output**. the process of giving something to computer is known as **input**.

Streams in C++

C++ I/O is based on streams, which are a sequence of bytes flowing in and out of the programs (just like water and oil flowing through a pipe). I/O systems in C++ are designed to work with a wide variety of devices including terminals, disks and tape drives. The input-output system supplies an interface to the programmer that is independent of the actual device being accessed. This interface is known as a stream. A stream is a sequence of bytes which acts either as a source from which input data can be obtained or as a destination to which output data can be sent. The source stream which provides data to the program is called the input stream and the destination stream which receives output from the program is called the output stream.

Follow the steps below to perform input and output:

- Construct a stream object.
- Connect (Associate) the stream object to an actual IO device
- Perform input/output operations on the stream, via the functions defined in the stream's public interface in a device-independent manner.
- Disconnect (Dissociate) the stream to the actual IO device (e.g., close the file).
- Free the stream object.

Unformatted input/output operations

You have already used the cin and cout (pre-defined in the iostream file) for input and output of data of various types. This has been made possible by overloading the operators << and >> to recognize all the basic C++ types.

- cin standard input stream
- cout standard output stream



Example

```
#include <iostream.h>
#include <conio.h>
void main()
{
    clrscr();
    int age;
    cout << "Enter your age: ";
    cin >> age;
    cout << "Your age is: " << age << endl;
}
```

Example

Write a program that adds two floating point numbers and shows the sum on screen.

```
#include <iostream.h>
#include <conio.h>
void main()
{
    clrscr();
    int var1, var2, sum;
    var1 = 24.34;
    var2 = 41.50;
    sum = var1+var2;
    cout << var1 << "+" << var2 << "=" << sum;
    getch();
}
```



Example

```
#include <iostream.h>
#include <conio.h>
void main()
{
    clrscr();
    int a, b, sum;
    cout<<"enter the value of a";
    cin>>a;
    cout<<"enter the value of b";
    cin>>b;
    sum = a+b;
    sub = a -b;
    cout <<a<< "+"<<b<<"="<<sum<<endl;
    cout<<a<<"-"<<b<<"="<<sub;
    getch();
}
```

Escape Sequences

Escape Sequence	Meaning
<code>\newline</code>	Ignored
<code>\\</code>	Backslash (\)
<code>\'</code>	Single quote (')
<code>\"</code>	Double quote (")
<code>\a</code>	ASCII Bell (BEL)
<code>\b</code>	ASCII Backspace (BS)
<code>\f</code>	ASCII Formfeed (FF)
<code>\n</code>	ASCII Linefeed (LF)
<code>\r</code>	ASCII Carriage Return (CR)
<code>\t</code>	ASCII Horizontal Tab (TAB)
<code>\v</code>	ASCII Vertical Tab (VT)
<code>\ooo</code>	ASCII character with octal value <i>ooo</i>
<code>\xhh...</code>	ASCII character with hex value <i>hh...</i>



Example

```
#include<iostream.h>
#include<conio.h>
void main()
{
    clrscr();
    cout<<"this\bThis\t article is \n
    best for \"beginners\"";
    getch();
}
```

Manipulators in C++

Manipulators are instructions to the output stream that modify the output in various ways.

“setprecision” manipulator

Sets the decimal precision to be used to format floating-point values on output operations. Preprocessor directive for setprecision is “<iomanip.h>”

Syntax

```
cout<<setprecision(n);
```

Example

```
cout<<setprecision(5)<<c;
getch();
}
```



Example

```
// setprecision example
#include <iostream.h>
#include <conio.h>
#include <iomanip.h>
void main()
{
    clrscr();
    double f=3.14159;
    cout << setprecision(5) << f << "\n";
    cout << setprecision(9) << f << "\n";
    cout << setprecision(5) << f << "\n";
    cout << setprecision(9) << f << "\n";
    getch();
}
```



Arithmetic Operators

It includes basic arithmetic operations like addition, subtraction, multiplication, division, modulus operations, increment, and decrement.

The Arithmetic Operators in C and C++ include:

1. **+** (**Addition**) – This operator is used to add two operands.
2. **-** (**Subtraction**) – Subtract two operands.
3. ***** (**Multiplication**) – Multiply two operands.
4. **/** (**Division**) – Divide two operands and gives the quotient as the answer.
5. **%** (**Modulus operation**) – Find the remains of two integers and gives the remainder after the division.
6. **++** (**Increment**) – Used to increment an operand.
7. **--** (**Decrement**) – Used to decrement an operand.

You can use the operators **+**, **-**, *****, and **/** with both integral and floating-point data types. You use **%** with only the integral data type to find the remainder in ordinary division. When you use **/** with the integral data type, it gives the quotient in ordinary division. That is, integral division truncates any fractional part; there is no rounding. Since high school, you have been accustomed to working with arithmetic expressions such as the following:

- i. -5
- ii. $8 - 7$
- iii. $3 + 4$
- iv. $2 + 3 * 5$
- v. $5.6 + 6.2 * 3$
- vi. $x + 2 * 5 + 6 / y$

Formally, an arithmetic expression is constructed by using arithmetic operators and numbers. The numbers appearing in the expression are called operands. The numbers that are used to evaluate an operator are called the operands for that operator. In expression (i), the symbol **-** specifies that the number 5 is negative. In this expression,

- has only one operand. Operators that have only one operand are called unary operators. In expression (ii), the symbol **-** is used to subtract 7 from 8. In this expression, **-** has two operands, 8 and 7.

Operators that have two operands are called **binary operators**.

Unary operator:

An operator that has only one operand.

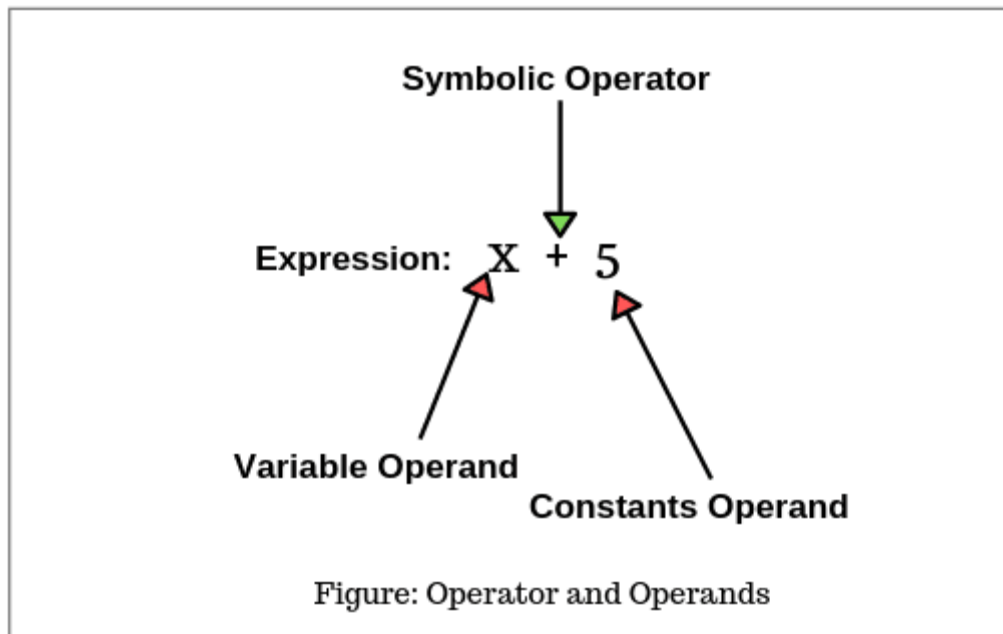
Binary operator:

Scientific Computing

Department of Physics



An operator that has two operands.



The unary arithmetic operators

Operator	Significance
+ -	Unary sign operators
++	Increment operator
--	Decrement operator



Table for Arithmetic Operators in C and C++

Operator	Operand	Operation	Elucidation
+	a, b	$a + b$	Addition
-	a, b	$a - b$	Subtraction
*	a, b	$a * b$	Multiplication
/	a, b	a / b	Division
%	a, b	$a \% b$	Modulus operator – to find the remainder when two integral digits are divided
++	a	$a ++$	Increment
--	a	$a --$	Decrement

Arithmetic Expression	Result	Description
$2 + 5$	7	
$13 + 89$	102	
$34 - 20$	14	
$45 - 90$	-45	
$2 * 7$	14	
$5 / 2$	2	In the division $5 / 2$, the quotient is 2 and the remainder is 1. Therefore, $5 / 2$ with the integral operands evaluates to the quotient, which is 2.
$14 / 7$	2	
$34 \% 5$	4	In the division $34 / 5$, the quotient is 6 and the remainder is 4. Therefore, $34 \% 5$ evaluates to the remainder, which is 4.
$4 \% 6$	4	In the division $4 / 6$, the quotient is 0 and the remainder is 4. Therefore, $4 \% 6$ evaluates to the remainder, which is 4.

Example-1

The following C++ program evaluates the preceding expressions:

```
// This program illustrates how integral expressions are
// evaluated
#include<iostream.h>
```

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

Scientific Computing

Department of Physics



```

void main()
{
    clrscr();

    cout << "2 + 5 = " << 2 + 5 << endl;
    cout << "13 + 89 = " << 13 + 89 << endl;
    cout << "34 - 20 = " << 34 - 20 << endl;
    cout << "45 - 90 = " << 45 - 90 << endl;
    cout << "2 * 7 = " << 2 * 7 << endl;
    cout << "5 / 2 = " << 5 / 2 << endl;
    cout << "14 / 7 = " << 14 / 7 << endl;
    cout << "34 % 5 = " << 34 % 5 << endl;
    cout << "4 % 6 = " << 4 % 6 << endl;

    getch();
}

```

Example-2

The following C++ program evaluates various floating-point expressions. (The details of how the expressions are evaluated are left as an exercise for you.)

// This program illustrates how floating-point expressions
// are evaluated.

```

#include<iostream.h>

#include<conio.h>

void main()
{
    clrscr();

    cout << "5.0 + 3.5 = " << 5.0 + 3.5 << endl;
    cout << "3.0 + 9.4 = " << 3.0 + 9.4 << endl;
    cout << "16.3 - 5.2 = " << 16.3 - 5.2 << endl;
    cout << "4.2 * 2.5 = " << 4.2 * 2.5 << endl;
    cout << "5.0 / 2.0 = " << 5.0 / 2.0 << endl;
    cout << "34.5 / 6.0 = " << 34.5 / 6.0 << endl;
    cout << "34.5 / 6.5 = " << 34.5 / 6.5 << endl;

    getch();
}

```




Order of Precedence

When more than one arithmetic operator is used in an expression, C++ uses the operator precedence rules to evaluate the expression. According to the order of precedence rules for arithmetic operators, $*, /, \%$ are at a higher level of precedence than $+, -$. Note that the operators $*, /$, and $\%$ have the same level of precedence. Similarly, the operators $+$ and $-$ have the same level of precedence.

When operators have the same level of precedence, the operations are performed from left to right. To avoid confusion, you can use parentheses to group arithmetic expressions.

$3 * 7 - 6 + 2 * 5 / 4 + 6$

means the following:

$(((3 * 7) - 6) + ((2 * 5) / 4)) + 6$	
$= ((21 - 6) + (10 / 4)) + 6$	(Evaluate $*$)
$= ((21 - 6) + 2) + 6$	(Evaluate $/$. Note that this is an integer division.)
$= (15 + 2) + 6$	(Evaluate $-$)
$= 17 + 6$	(Evaluate first $+$)
$= 23$	(Evaluate $+$)

Example

`cout<<3+4*5;`
 Result = 23
 Whereas
`Cout<<(3+5)*5;`
 Result= 35

Because arithmetic operators are evaluated from left to right, unless parentheses are present, the associativity of the arithmetic operators is said to be from left to right.



Mixed Expression	Evaluation	Rule Applied
$3 / 2 + 5.5$	$= 1 + 5.5$ $= 6.5$	$3 / 2 = 1$ (integer division; ($1 + 5.5$ $= 1.0 + 5.5$ $= 6.5$)
$15.6 / 2 + 5$	$= 7.8 + 5$ $= 12.8$	$15.6 / 2$ $= 15.6 / 2.0$ $= 7.8$ $7.8 + 5$ $= 7.8 + 5.0$ $= 12.8$
$4 + 5 / 2.0$	$= 4 + 2.5$ $= 6.5$	$5 / 2.0 = 5.0 / 2.0$ $= 2.5$ $4 + 2.5 = 4.0 + 2.5$ $= 6.5$
$4 * 3 + 7 / 5 - 25.5$	$= 12 + 7 / 5 - 25.5$ $= 12 + 1 - 25.5$ $= 13 - 25.5$ $= -12.5$	$4 * 3 = 12$; $7 / 5 = 1$ (integer division; $12 + 1 = 13$; $13 - 25.5 = 13.0 - 25.5$ $= -12.5$

Precedence of arithmetic operators

Precedence	Operator	Grouping
<div style="text-align: center;"> High Low </div>	++ -- (postfix)	left to right
	++ -- (prefix) + - (sign)	right to left
	* / %	left to right
	+ (addition) - (subtraction)	left to right

Example-4

The following C++ program evaluates the preceding expressions:
 // This program illustrates how mixed expressions are evaluated.

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



```
#include<conio.h>

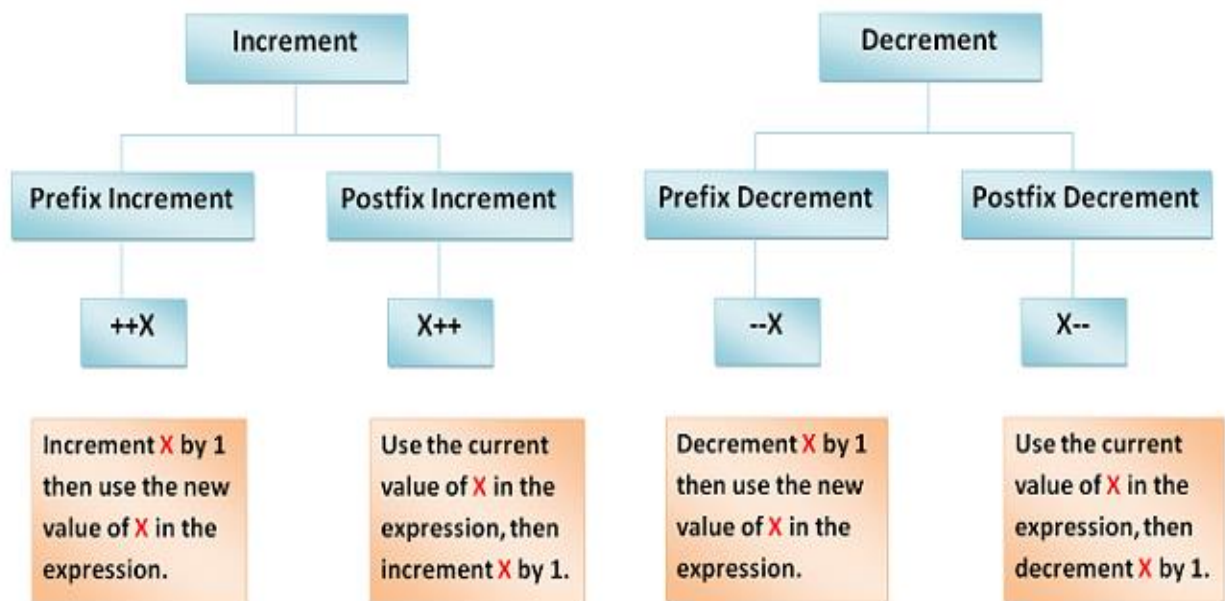
void main()

{
clrscr();

cout << "3 / 2 + 5.5 = " << 3 / 2 + 5.5 << endl;
cout << "15.6 / 2 + 5 = " << 15.6 / 2 + 5 << endl;
cout << "4 + 5 / 2.0 = " << 4 + 5 / 2.0 << endl;
cout << "4 * 3 + 7 / 5 - 25.5 = "
    << 4 * 3 + 7 / 5 - 25.5
    << endl;

getch();
}
```

Increment & Decrement Operator





Example

Increment and Decrement Operators			
<div>Increment</div> <div>Pre-increment Post-increment</div> <div>$Y = ++X$ $Y = X++$</div>		<div>Decrement</div> <div>Pre-decrement Post-decrement</div> <div>$Y = --X$ $Y = X--$</div>	
Expression	Initial Value of X	Final Value of X	Final Value of Y
$Y = ++X$	4	5	5
$Y = X++$	4	5	4
$Y = --X$	4	3	3
$Y = X--$	4	3	4



Practice Exercise

- Write a program that produces the following output:

```
*****
*   Programming Assignment 1   *
*   Computer Programming I    *
*   Author: ???               *
*   Due Date: Thursday, Jan. 24 *
*****
```

- Write four different program statements the increment the value of an integer variable sum.

Given the following declaration:

```
int x = 2;
```

Indicate what each of the following C++ statements would print.

- `cout << "x"<< endl;`
- `cout << 'x'<< endl;`
- `cout << x << endl;`
- `cout << "x + 1"<< endl;`
- `cout << 'x'+ 1 << endl;`
- `cout << x + 1 << endl;`

- Write a C++ program that receives two integer values from the user. The program then should print the sum (addition), difference (subtraction), product (multiplication), quotient (division), and remainder after division (modulus). Your program must use only integers.

A sample program run would look like

```
Please enter the first number: 10
Please enter the second number: 2
10 + 2 = 12
10 - 2 = 8
10 * 2 = 20
10 / 2 = 5
10 % 2 = 0
```

- Write a C++ program that receives two double-precision floating-point values from the user. The program then should print the sum (addition), difference (subtraction), product



(multiplication), and quotient (division). Your program should use only integers. A sample program run would look like

```
Please enter the first number: 10
Please enter the second number: 2.5
10 + 2.5 = 12.5
10 - 2.5 = 7.5
10 * 2.5 = 25
10 / 2.5 = 4
```

- What is printed by the following code fragment?

```
int x1 = 2, y1, x2 = 2, y2;
y1 = ++x1;
y2 = x2++;
cout << x1 << " " << x2 << endl;
cout << y1 << " " << y2 << endl;
```

- Write a program to find the area of circle, circumference of circle, area of cube, volume of cylinder.
- What is another way to write the following declaration and initialization?
- `Int x = 10;`
- In C++ can you declare more than variable in the same declaration statement? If so, how?
- Rewrite the following code fragment so that the code behaves exactly the same but does not use `endl`

```
cout << endl << "Aye!" << endl << "Bye!" << endl;
```

- Will the following lines of code print the same thing? Explain why or why no

```
cout << 6 << endl;
cout << "6" << endl;
```

- Classify each of the following as either a legal or Illegal C++ identifier



- (a) **fred**
- (b) **if**
- (c) **2x**
- (d) **-4**
- (e) **sum_total**
- (f) **sumTotal**
- (g) **sum-total**
- (h) **sum total**
- (i) **sumtotal**
- (j) **While**
- (k) **x2**
- (l) **Private**
- (m) **public**
- (n) **\$16**

- Write a program that performs all mathematical operators using two variables.
- Write a program that solve the following expression
- $a*b/(-c*31\%13)*d$
- Write a program to calculate the area of the square.
- Write a program to display following output using single cout

```

*
***
****
*****
*****
*****
*****

```

- Write a program, use different values of variable using set precision manipulator.
- Write a program to calculate the simple interest. It inputs simple amount, rate of interest and the number of years and displays the simple interest.
- Write a program that input divider and dividend. It then calculate and display quotient and remainder.
- Write a program that inputs distance traveled and the speed of the vehicle. Calculate the time required to reach the destination
- Write a program that input time in seconds and converts it into hh-mm-ss format, as shown in fig



Output:
Enter time in seconds: 5300
HH-MM-SS= 1:28:20

- Write a program that gets temperature from the user in Celsius and convert it into Fahrenheit using the formula $F = 9/5 * C + 32$.
- Write a program that convert person's height into centimeter by using the formula $2.54 * \text{height}$
- Write a program to calculate the final velocity of an object by taking following values as an input from the user v_i = initial velocity, a = acceleration, t = time, by using formula
 - $V_f = v_i + at$
- Write a program that inputs a five-digit number and reverse the number. Output should be like

Output:
Enter 5-digit number: 92174
Number in reverse order is 47129

- Write a program that input an even and an odd number through keyboard, multiply even with 9, and odd with 6, and add both results. It subtract the result from 1000. And finally prints the difference
- Write a program to print following message

C
O M
P U T
E R I S
A W O R L
D O F S
C I E
N C
E

- Write a program that inputs age in year and out put in months and days.