Unit 2: Introduction Of C++

UNIT-I Introduction: Introducing Object Oriented Approach, Procedural Programming Language Vs Object Oriented Language. Basic concept of OOPs, operators, tokens, variables, Keywords, Data types, identifiers, characters, typedef statement, constants, Enumerated data type.

Outcome of this unit ->: students should have a clear understanding of the fundamental concepts of object-oriented programming, the difference between procedural and object-oriented approaches, various programming elements like keywords, data types, variables, identifiers, and basic operators. They should also grasp the importance and usage of typedef, constants, and enumerated data types in programming.

C++ history

C++ programming language was developed in 1980 by Bjarne Stroustrup at bell laboratories of AT&T (American Telephone & Telegraph), located in U.S.A.

Bjarne Stroustrup is known as the **founder of C++ language.**

s

It was develop for adding a feature of **OOP (Object Oriented Programming)** in C without significantly changing the C component.

C++ programming is "relative" (called a superset) of C, it means any valid C program is also a valid C++ program.

Faculty: SHAHRUKH KAMAL
Shahrukhkamal7@gmail.com

Unit 2: Introduction Of C++

Language	Year	Developed By
Algol	1960	International Group
BCPL	1967	Martin Richard
B	1970	Ken Thompson
Traditional C	1972	Dennis Ritchie
K & R C	1978	Kernighan & Dennis Ritchie
C++	1980	Bjarne Stroustrup

What is C++?

C++ is a special-purpose programming language developed by **Bjarne Stroustrup** at Bell Labs circa 1980. C++ language is very similar to C language, and it is so compatible with C that it can run 99% of C programs without changing any source of code though C++ is an object-oriented programming language, so it is safer and well-structured programming language than C.

Let's summarize the above differences in a tabular form.

No.	C	C++
1)	C follows the **procedural style programming.**	C++ is multi-paradigm. It supports both **procedural and object oriented.**
2)	Data is less secured in C.	In C++, you can use modifiers for class members to make it inaccessible for outside users.

Faculty: SHAHRUKH KAMAL
Shahrukhkamal7@gmail.com

Unit 2: Introduction Of C++

3)	C follows the **top-down approach.**	C++ follows the **bottom-up approach.**
4)	C does not support function overloading.	C++ supports function overloading.
5)	In C, you can't use functions in structure.	In C++, you can use functions in structure.
6)	C does not support reference variables.	C++ supports reference variables.
7)	In C, **scanf() and printf()** are mainly used for input/output.	C++ mainly uses stream **cin and cout** to perform input and output operations.
8)	Operator overloading is not possible in C.	Operator overloading is possible in C++.
9)	C programs are divided into **procedures and modules**	C++ programs are divided into **functions and classes.**
10)	C does not provide the feature of namespace.	C++ supports the feature of namespace.
11)	Exception handling is not easy in C. It has to perform using other functions.	C++ provides exception handling using Try and Catch block.
12)	C does not support the inheritance.	C++ supports inheritance.

C++ Features

It provides a lot of features that are given below.

1. Simple
2. Abstract Data types
3. Machine Independent or Portable
4. Mid-level programming language
5. Structured programming language
6. Rich Library

Faculty: SHAHRUKH KAMAL
Shahrukhkamal7@gmail.com

Unit 2: Introduction Of C++

7. Memory Management
8. Quicker Compilation
9. Pointers
10. Recursion
11. Extensible
12. Object-Oriented
13. Compiler based
14. Reusability
15. National Standards
16. Errors are easily detected
17. Power and Flexibility
18. Strongly typed language
19. Redefine Existing Operators
20. Modeling Real-World Problems
21. Clarity

---

1) Simple

C++ is a simple language because it provides a structured approach (to break the problem into parts), a rich set of library functions, data types, etc.

2) Abstract Data types

In C++, complex data types called Abstract Data Types (ADT) can be created using classes.

3) Portable

C++ is a portable language and programs made in it can be run on different machines.

4) Mid-level / Intermediate programming language

C++ includes both low-level programming and high-level language so it is known as a mid-level and intermediate programming language. It is used to develop system applications such as kernel, driver, etc.

Unit 2: Introduction Of C++

5) Structured programming language

C++ is a structured programming language. In this we can divide the program into several parts using functions.

6) Rich Library

C++ provides a lot of inbuilt functions that make the development fast. Following are the libraries used in C++ programming are:

- <iostream>
- <cmath>
- <cstdlib>
- <fstream>

7) Memory Management

C++ provides very efficient management techniques. The various memory management operators help save the memory and improve the program's efficiency. These operators allocate and deallocate memory at run time. Some common memory management operators available C++ are new, delete etc.

8) Quicker Compilation

C++ programs tend to be compact and run quickly. Hence the compilation and execution time of the C++ language is fast.

9) Pointer

C++ provides the feature of pointers. We can use pointers for memory, structures, functions, array, etc. We can directly interact with the memory by using the pointers.

10) Recursion

In C++, we can call the function within the function. It provides code reusability for every function.

11) Extensible

C++ programs can easily be extended as it is very easy to add new features into the existing program.

12) Object-Oriented

In C++, object-oriented concepts like data hiding, encapsulation, and data abstraction can easily be implemented using keyword class, private, public, and protected access specifiers. Object-oriented makes development and maintenance easier.

13) Compiler based

C++ is a compiler-based programming language, which means no C++ program can be executed without compilation. C++ compiler is easily available, and it requires very little space for storage. First, we need to compile our program using a compiler, and then we can execute our program.

14) Reusability

With the use of inheritance of functions programs written in C++ can be reused in any other program of C++. You can save program parts into library files and invoke them in your next programming projects simply by including the library files. New programs can be developed in lesser time as the existing code can be reused. It is also possible to define several functions with same name that perform different task. For Example: abs () is used to calculate the absolute value of integer, float and long integer.

15) National Standards

C++ has national standards such as ANSI.

16) Errors are easily detected

It is easier to maintain a C++ programs as errors can be easily located and rectified. It also provides a feature called exception handling to support error handling in your program.

17) Power and Flexibility

C++ is a powerful and flexible language because of most of the powerful flexible and modern UNIX operating system is written in C++. Many compilers and interpreters for other languages such as FORTRAN, PERL, Python, PASCAL, BASIC, LISP, etc., have been

written in C++. C++ programs have been used for solving physics and engineering problems and even for animated special effects for movies.

18) Strongly typed language

The list of arguments of every function call is typed checked during compilation. If there is a type mismatch between actual and formal arguments, implicit conversion is applied if possible. A compile-time occurs if an implicit conversion is not possible or if the number of arguments is incorrect.

19) Redefine Existing Operators

C++ allows the programmer to redefine the meaning of existing operators such as +, -. **For Example,** The "+" operator can be used for adding two numbers and concatenating two strings.

20) Modelling real-world problems

The programs written in C++ are well suited for real-world modeling problems as close as possible to the user perspective.

Difference between procedural programming and object-oriented programming

S.no.	On the basis of	Procedural Programming	Object-oriented programming
1.	**Definition**	It is a programming language that is derived from structure programming and based upon the concept of calling procedures. It follows a step-by-step approach in order to break down a task into a set of variables and routines via a sequence of instructions.	Object-oriented programming is a computer programming design philosophy or methodology that organizes/ models software design around data or objects rather than functions and logic.

Faculty: SHAHRUKH KAMAL
Shahrukhkamal7@gmail.com

2.	**Security**	It is less secure than OOPs.	Data hiding is possible in object-oriented programming due to abstraction. So, it is more secure than procedural programming.
3.	**Approach**	It follows a top-down approach.	It follows a bottom-up approach.
4.	**Data movement**	In procedural programming, data moves freely within the system from one function to another.	In OOP, objects can move and communicate with each other via member functions.
5.	**Orientation**	It is structure/procedure-oriented.	It is object-oriented.
6.	**Access modifiers**	There are no access modifiers in procedural programming.	The access modifiers in OOP are named as private, public, and protected.
7.	**Inheritance**	Procedural programming does not have the concept of inheritance.	There is a feature of inheritance in object-oriented programming.
8.	**Code reusability**	There is no code reusability present in procedural programming.	It offers code reusability by using the feature of inheritance.
9.	**Overloading**	Overloading is not possible in procedural programming.	In OOP, there is a concept of function overloading and operator overloading.
10.	**Importance**	It gives importance to functions over data.	It gives importance to data over functions.
11.	**Virtual class**	In procedural programming, there are no virtual classes.	In OOP, there is an appearance of virtual classes in inheritance.
12.	**Complex problems**	It is not appropriate for complex problems.	It is appropriate for complex problems.

Faculty: SHAHRUKH KAMAL
Shahrukhkamal7@gmail.com

13.	**Data hiding**	There is not any proper way for data hiding.	There is a possibility of data hiding.
14.	**Program division**	In Procedural programming, a program is divided into small programs that are referred to as functions.	In OOP, a program is divided into small parts that are referred to as objects.
15.	**Examples**	Examples of Procedural programming include C, Fortran, Pascal, and VB.	The examples of object-oriented programming are - .NET, C#, Python, Java, VB.NET, and C++.

C++ Operators

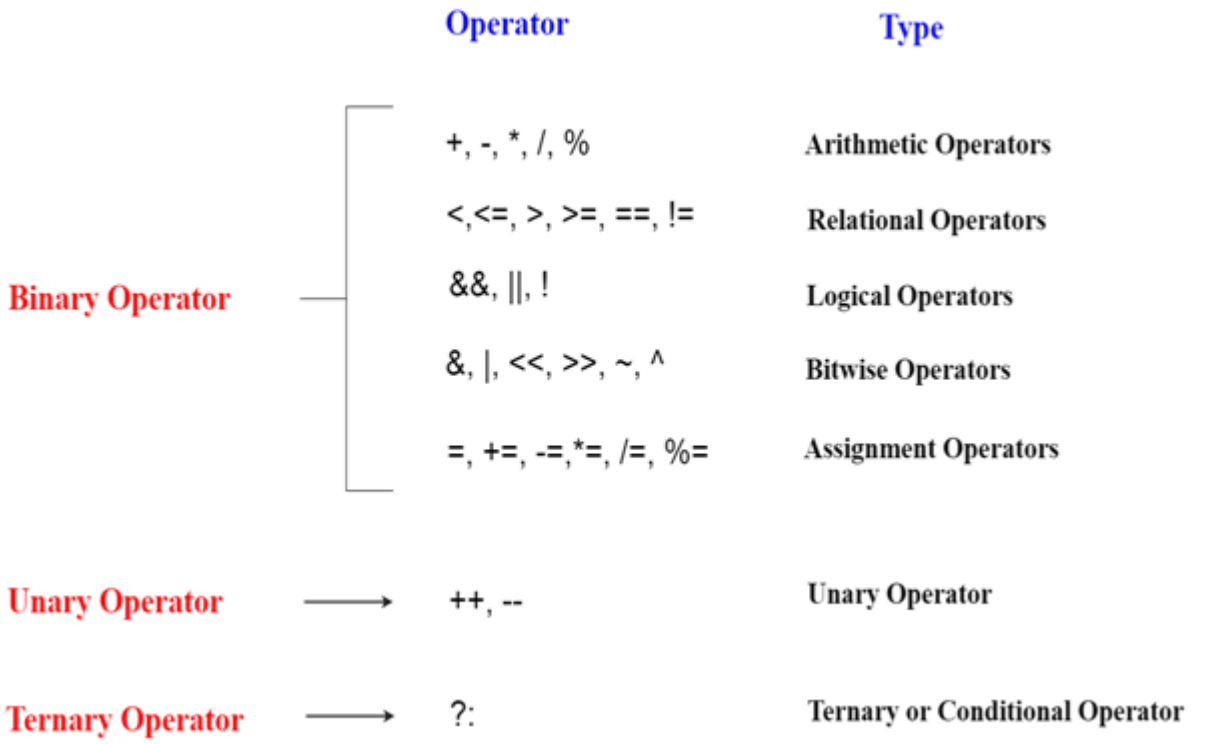
An operator is simply a symbol that is used to perform operations. There can be many types of operations like arithmetic, logical, bitwise etc.

There are following types of operators to perform different types of operations in C language.

- Arithmetic Operators
- Relational Operators
- Logical Operators
- Bitwise Operators
- Assignment Operator
- Unary operator
- Ternary or Conditional Operator
- Misc Operator

Faculty: SHAHRUKH KAMAL
Shahrukhkamal7@gmail.com

Unit 2: Introduction Of C++



Precedence of Operators in C++

The precedence of operator species that which operator will be evaluated first and next. The associativity specifies the operators direction to be evaluated, it may be left to right or right to left.

Let's understand the precedence by the example given below:

```
1. int data=5+10*10;
```

The "data" variable will contain 105 because * (multiplicative operator) is evaluated before + (additive operator).

The precedence and associativity of C++ operators is given below:

Category	Operator	Associativity
Postfix	() [] -> . ++ - -	Left to right
Unary	+ - ! ~ ++ - - (type)* & sizeof	Right to left

Multiplicative	* / %	Left to right
Additive	+ -	Right to left
Shift	<< >>	Left to right
Relational	< <= > >=	Left to right
Equality	== !=/td>	Right to left
Bitwise AND	&	Left to right
Bitwise XOR	^	Left to right
Bitwise OR	|	Right to left
Logical AND	&&	Left to right
Logical OR	||	Left to right
Conditional	?:	Right to left
Assignment	= += -= *= /= %=>>= <<= &= ^= |=	Right to left
Comma	,	Left to right

C++ Program

Before starting the abcd of C++ language, you need to learn how to write, compile and run the first C++ program.

To write the first C++ program, open the C++ console and write the following code:

```
1. #include <iostream.h>
2. #include<conio.h>
3. void main() {
4.    clrscr();
5.    cout << "Welcome to C++ Programming.";
6.    getch();
```

7. }

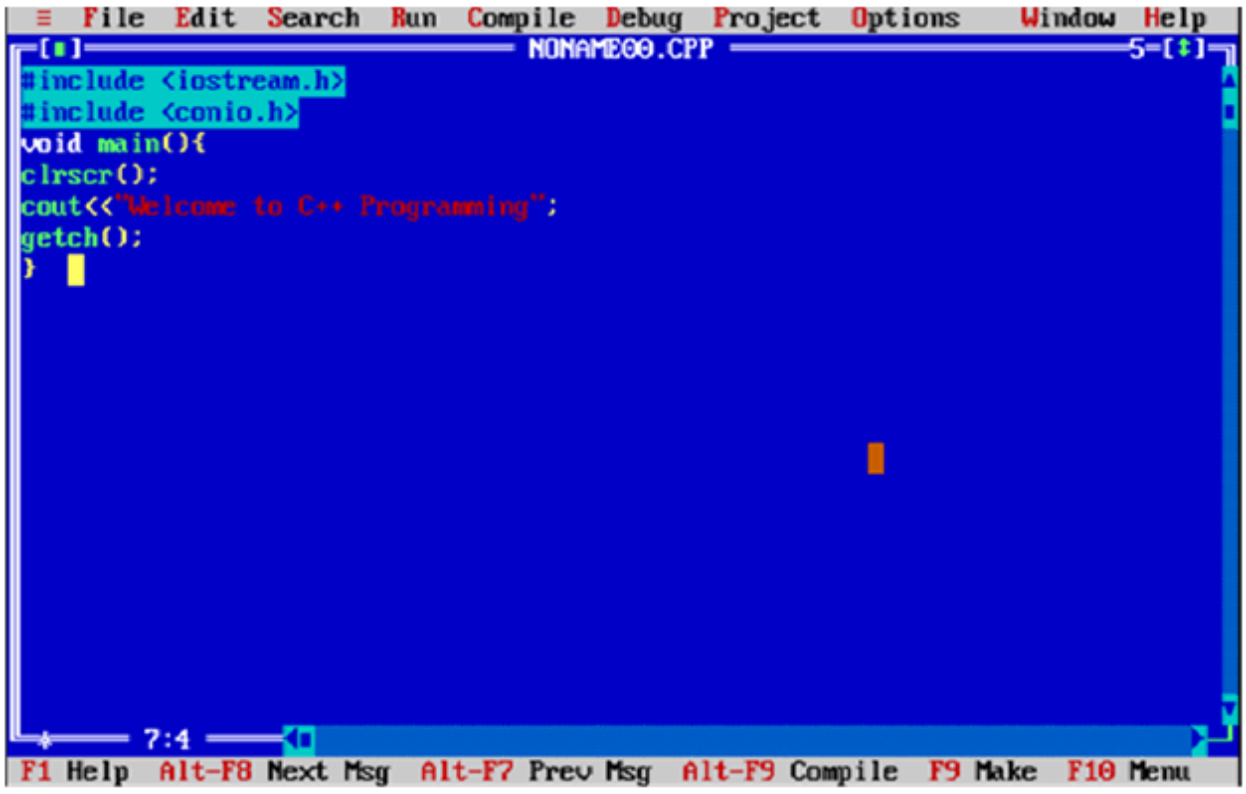
#include<iostream.h> includes the **standard input output** library functions. It provides **cin** and **cout** methods for reading from input and writing to output respectively.

#include <conio.h> includes the **console input output** library functions. The getch() function is defined in conio.h file.

void main() The **main() function is the entry point of every program** in C++ language. The void keyword specifies that it returns no value.

cout << "Welcome to C++ Programming." is **used to print the data "Welcome to C++ Programming."** on the console.

getch() The getch() function **asks for a single character**. Until you press any key, it blocks the screen.



How to compile and run the C++ program

There are 2 ways to compile and run the C++ program, by menu and by shortcut.

By menu

Now **click on the compile menu then compile sub menu** to compile the c++ program.

Then **click on the run menu then run sub menu** to run the c++ program.

By shortcut

Or, press ctrl+f9 keys compile and run the program directly.

You will see the following output on user screen.

You can view the user screen any time by pressing the **alt+f5** keys.

C++ Basic Input/Output

C++ I/O operation is using the stream concept. Stream is the sequence of bytes or flow of data. It makes the performance fast.

If bytes flow from main memory to device like printer, display screen, or a network connection, etc, this is called as **output operation.**

If bytes flow from device like printer, display screen, or a network connection, etc to main memory, this is called as **input operation.**

I/O Library Header Files

Let us see the common header files used in C++ programming are:

Header File	Function and Description
<iostream>	It is used to define the **cout, cin and cerr** objects, which correspond to standard output stream, standard input stream and standard error stream, respectively.
<iomanip>	It is used to declare services useful for performing formatted I/O, such as **setprecision and setw.**
<fstream>	It is used to declare services for user-controlled file processing.

Standard output stream (cout)

The **cout** is a predefined object of **ostream** class. It is connected with the standard output device, which is usually a display screen. The cout is used in conjunction with stream insertion operator (<<) to display the output on a console

Let's see the simple example of standard output stream (cout):

```
1. #include <iostream>
2. using namespace std;
3. int main( ) {
4.    char ary[] = "Welcome to C++ tutorial";
5.    cout << "Value of ary is: " << ary << endl;
6. }
```

Output:

```
Value of ary is: Welcome to C++ tutorial
```

Standard input stream (cin)

The **cin** is a predefined object of **istream** class. It is connected with the standard input device, which is usually a keyboard. The cin is used in conjunction with stream extraction operator (>>) to read the input from a console.

Let's see the simple example of standard input stream (cin):

Unit 2: Introduction Of C++

```
1. #include <iostream>
2. using namespace std;
3. int main( ) {
4.   int age;
5.    cout << "Enter your age: ";
6.    cin >> age;
7.    cout << "Your age is: " << age << endl;
8. }
```

Output:

```
Enter your age: 22
Your age is: 22
```

Standard end line (endl)

The **endl** is a predefined object of **ostream** class. It is used to insert a new line characters and flushes the stream.

Let's see the simple example of standard end line (endl):

```
1. ss#include <iostream>
2. using namespace std;
3. int main( ) {
4. cout << "C++ Tutorial";
5. cout << " Javatpoint"<<endl;
6. cout << "End of line"<<endl;
7. }
```

Output:

```
C++ Tutorial Javatpoint
End of line
```

C++ Variable

A variable is a name of memory location. It is used to store data. Its value can be changed and it can be reused many times.

Unit 2: Introduction Of C++

It is a way to represent memory location through symbol so that it can be easily identified.

Let's see the syntax to declare a variable:

```
type variable_list;
```

The example of declaring variable is given below:

```
int x;
float y;
char z;
```

Here, x, y, z are variables and int, float, char are data types.

We can also provide values while declaring the variables as given below:

```
int x=5,b=10;  //declaring 2 variable of integer type
float f=30.8;
char c='A';
```

Rules for defining variables

A variable can have alphabets, digits and underscore.

A variable name can start with alphabet and underscore only. It can't start with digit.

No white space is allowed within variable name.

A variable name must not be any reserved word or keyword e.g. char, float etc.

Valid variable names:

```
int a;
int _ab;
int a30;
```

Invalid variable names:

```
int 4;
```

```
2. int x y;
3. int double;
4.
```

5. C++ Data Types

6. A data type specifies the type of data that a variable can store such as integer, floating, character etc.

7. **Data Types in C++**
8. There are 4 types of data types in C++ language.

Types	Data Types
Basic Data Type	int, char, float, double, etc
Derived Data Type	array, pointer, etc
Enumeration Data Type	enum
User Defined Data Type	structure

9. Basic Data Types

10. The basic data types are integer-based and floating-point based. C++ language supports both signed and unsigned literals.

Unit 2: Introduction Of C++

11. The memory size of basic data types may change according to 32 or 64 bit operating system.
12. Let's see the basic data types. It size is given according to 32 bit OS.

Data Types	Memory Size	Range
Char	1 byte	-128 to 127
signed char	1 byte	-128 to 127
unsigned char	1 byte	0 to 127
Short	2 byte	-32,768 to 32,767
signed short	2 byte	-32,768 to 32,767
unsigned short	2 byte	0 to 32,767
Int	2 byte	-32,768 to 32,767
signed int	2 byte	-32,768 to 32,767
unsigned int	2 byte	0 to 32,767
short int	2 byte	-32,768 to 32,767
signed short int	2 byte	-32,768 to 32,767
unsigned short int	2 byte	0 to 32,767
long int	4 byte	
signed long int	4 byte	
unsigned long int	4 byte	
Float	4 byte	
Double	8 byte	
long double	10 byte	

Unit 2: Introduction Of C++

13. C++ Keywords

14. A keyword is a reserved word. You cannot use it as a variable name, constant name etc. **A list of 32 Keywords in C++ Language which are also available in C language are given below.**

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

15. **A list of 30 Keywords in C++ Language which are not available in C language are given below.**

Asm	dynamic_cast	namespace	reinterpret_cast	bool
Explicit	new	static_cast	false	catch
Operator	template	friend	private	class
This	inline	public	throw	const_cast
Delete	mutable	protected	true	try
Typeid	typename	using	virtual	wchar_t

C++ Identifiers

C++ identifiers in a program are used to refer to the name of the variables, functions, arrays, or other user-defined data types created by the programmer. They are the basic requirement of any language. Every language has its own rules for naming the identifiers.

In short, we can say that the C++ identifiers represent the essential elements in a program which are given below:

Unit 2: Introduction Of C++

- **Constants**
- **Variables**
- **Functions**
- **Labels**
- **Defined data types**

Some naming rules are common in both C and C++. They are as follows:

- Only alphabetic characters, digits, and underscores are allowed.
- The identifier name cannot start with a digit, i.e., the first letter should be alphabetical. After the first letter, we can use letters, digits, or underscores.
- In C++, uppercase and lowercase letters are distinct. Therefore, we can say that C++ identifiers are case-sensitive.
- A declared keyword cannot be used as a variable name.

For example, suppose we have two identifiers, named as 'FirstName', and 'Firstname'. Both the identifiers will be different as the letter 'N' in the first case in uppercase while lowercase in second. Therefore, it proves that identifiers are case-sensitive.

Valid Identifiers

The following are the examples of valid identifiers are:

1. Result
2. Test2
3. _sum
4. power

Invalid Identifiers

The following are the examples of invalid identifiers:

```
1. Sum-1   // containing special character '-'.
2. 2data   // the first letter is a digit.
3. break   // use of a keyword.
```

The major difference between C and C++ is the limit on the length of the name of the variable. ANSI C considers only the first 32 characters in a name while ANSI C++ imposes no limit on the length of the name.

Unit 2: Introduction Of C++

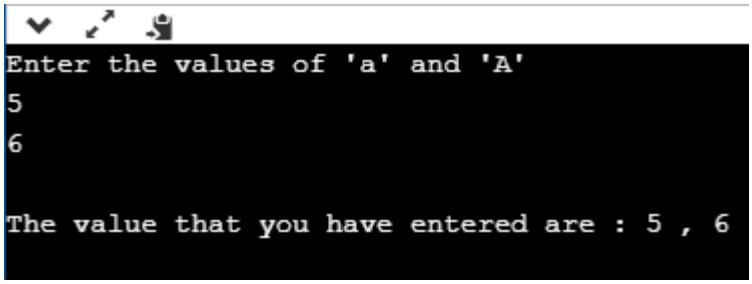
Constants are the identifiers that refer to the fixed value, which do not change during the execution of a program. Both C and C++ support various kinds of literal constants, and they do have any memory location. For example, 123, 12.34, 037, 0X2, etc. are the literal constants.

Let's look at a simple example to understand the concept of identifiers.

```
1. #include <iostream>
2. using namespace std;
3. int main()
4. {
5.     int a;
6.     int A;
7.     cout<<"Enter the values of 'a' and 'A'";
8.     cin>>a;
9.     cin>>A;
10.    cout<<"\nThe values that you have entered are : "<<a<<" , "<<A;
11.    return 0;
12. }
```

In the above code, we declare two variables 'a' and 'A'. Both the letters are same but they will behave as different identifiers. As we know that the identifiers are the case-sensitive so both the identifiers will have different memory locations.

Output



What are the keywords?

Keywords are the reserved words that have a special meaning to the compiler. They are reserved for a special purpose, which cannot be used as the identifiers. For example, 'for', 'break', 'while', 'if', 'else', etc. are the predefined words where predefined words are those words whose meaning is already known by the compiler. Whereas,

Faculty: SHAHRUKH KAMAL
Shahrukhkamal7@gmail.com

the identifiers are the names which are defined by the programmer to the program elements such as variables, functions, arrays, objects, classes.

Differences between Identifiers and Keywords

The following is the list of differences between identifiers and keywords:

Identifiers	Keywords
Identifiers are the names defined by the programmer to the basic elements of a program.	Keywords are the reserved words whose meaning is known by the compiler.
It is used to identify the name of the variable.	It is used to specify the type of entity.
It can consist of letters, digits, and underscore.	It contains only letters.
It can use both lowercase and uppercase letters.	It uses only lowercase letters.
No special character can be used except the underscore.	It cannot contain any special character.
The starting letter of identifiers can be lowercase, uppercase or underscore.	It can be started only with the lowercase letter.
It can be classified as internal and external identifiers.	It cannot be further classified.
Examples are test, result, sum, power, etc.	Examples are 'for', 'if', 'else', 'break', etc.

C++ Expression

C++ expression consists of operators, constants, and variables which are arranged according to the rules of the language. It can also contain function calls which return values. An expression can consist of one or more operands, zero or more operators to compute a value. Every expression produces some value which is assigned to the variable with the help of an assignment operator.

Unit 2: Introduction Of C++

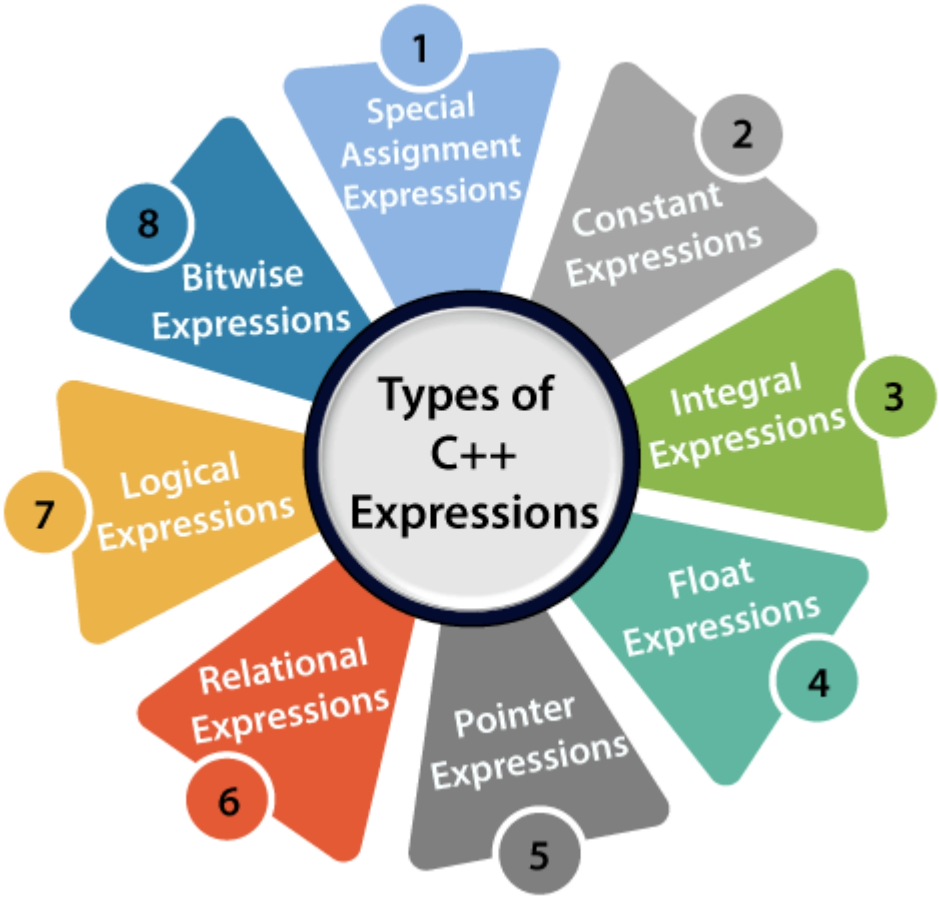
Examples of C++ expression:

1. (a+b) - c
2. (x/y) -z
3. 4a2 - 5b +c
4. (a+b) * (x+y)

An expression can be of following types:

- Constant expressions
- Integral expressions
- Float expressions
- Pointer expressions
- Relational expressions
- Logical expressions
- Bitwise expressions
- Special assignment expressions

Unit 2: Introduction Of C++



If the expression is a combination of the above expressions, such expressions are known as compound expressions.

Constant expressions

A constant expression is an expression that consists of only constant values. It is an expression whose value is determined at the compile-time but evaluated at the run-time. It can be composed of integer, character, floating-point, and enumeration constants.

Constants are used in the following situations:

- It is used in the subscript declarator to describe the array bound.
- It is used after the case keyword in the switch statement.
- It is used as a numeric value in an **enum**
- It specifies a bit-field width.
- It is used in the pre-processor **#if**

Faculty: SHAHRUKH KAMAL
Shahrukhkamal7@gmail.com

Unit 2: Introduction Of C++

In the above scenarios, the constant expression can have integer, character, and enumeration constants. We can use the static and extern keyword with the constants to define the function-scope.

The following table shows the expression containing constant value:

Expression containing constant	Constant value
x = (2/3) * 4	(2/3) * 4
extern int y = 67	67
int z = 43	43
static int a = 56	56

Let's see a simple program containing constant expression:

```
1. #include <iostream>
2. using namespace std;
3. int main()
4. {
5.     int x;       // variable declaration.
6.     x=(3/2) + 2;  // constant expression
7.     cout<<"Value of x is : "<<x;  // displaying the value of x.
8.     return 0;
9. }
```

In the above code, we have first declared the 'x' variable of integer type. After declaration, we assign the simple constant expression to the 'x' variable.

Output

```
Value of x is : 3
```


Faculty: SHAHRUKH KAMAL
Shahrukhkamal7@gmail.com

Unit 2: Introduction Of C++

Integral Expressions

An integer expression is an expression that produces the integer value as output after performing all the explicit and implicit conversions.

Following are the examples of integral expression:

1. (x * y) -5
2. x + **int**(9.0)
3. where x and y are the integers.

Let's see a simple example of integral expression:

```
1.  #include <iostream>
2.  using namespace std;
3.  int main()
4.  {
5.      int x;  // variable declaration.
6.      int y;  // variable declaration
7.      int z;  // variable declaration
8.      cout<<"Enter the values of x and y";
9.      cin>>x>>y;
10.     z=x+y;
11.     cout<<"\n"<<"Value of z is :"<<z; //  displaying the value of z.
12.     return 0;
13. }
```

In the above code, we have declared three variables, i.e., x, y, and z. After declaration, we take the user input for the values of 'x' and 'y'. Then, we add the values of 'x' and 'y' and stores their result in 'z' variable.

Output

```
Enter the values of x and y
8
9
Value of z is :17
```

Let's see another example of integral expression.

```
1. #include <iostream>
2. using namespace std;
3. int main()
4. {
5.    int x;   // variable declaration
6.    int y=9;    // variable initialization
7.    x=y+int(10.0);    // integral expression
8.   cout<<"Value of x : "<<x;   // displaying the value of x.
9.   return 0;
10.}
```

In the above code, we declare two variables, i.e., x and y. We store the value of expression (y+int(10.0)) in a 'x' variable.

Output

```
Value of x : 19
```

Float Expressions

A float expression is an expression that produces floating-point value as output after performing all the explicit and implicit conversions.

The following are the examples of float expressions:

1. x+y
2. (x/10) + y
3. 34.5
4. x+float(10)

Let's understand through an example.

```
1. #include <iostream>
2. using namespace std;
3. int main()
4. {
5.
6.    float x=8.9;     // variable initialization
7.    float y=5.6;      // variable initialization
```

```
8.    float z;          // variable declaration
9.    z=x+y;
10.   std::cout <<"value of z is :"  << z<<std::endl;  // displaying the value of z.
11.
12.
13.    return 0;
14. }
```

Output

```
value of z is :14.5
```

Let's see another example of float expression.

```
1.  #include <iostream>
2.  using namespace std;
3.  int main()
4.  {
5.     float x=6.7;    // variable initialization
6.     float y;     // variable declaration
7.     y=x+float(10);   // float expression
8.     std::cout <<"value of y is :"  << y<<std::endl;  // displaying the value of y
9.     return 0;
10. }
```

In the above code, we have declared two variables, i.e., x and y. After declaration, we store the value of expression (x+float(10)) in variable 'y'.

Output

```
value of y is :16.7
```

Pointer Expressions

A pointer expression is an expression that produces address value as an output.

The following are the examples of pointer expression:

1. &x
2. ptr

3. ptr++
4. ptr-

Let's understand through an example.

```
#include <iostream>
using namespace std;
int main()
{

   int a[]={1,2,3,4,5};  // array initialization
   int *ptr;       // pointer declaration
   ptr=a;    // assigning base address of array to the pointer ptr
   ptr=ptr+1;   // incrementing the value of pointer
std::cout <<"value of second element of an array : "  << *ptr<<std::endl;
   return 0;
}
```

In the above code, we declare the array and a pointer ptr. We assign the base address to the variable 'ptr'. After assigning the address, we increment the value of pointer 'ptr'. When pointer is incremented then 'ptr' will be pointing to the second element of the array.

Output

```
value of second element of an array : 2
```

Relational Expressions

A relational expression is an expression that produces a value of type bool, which can be either true or false. It is also known as a boolean expression. When arithmetic expressions are used on both sides of the relational operator, arithmetic expressions are evaluated first, and then their results are compared.

The following are the examples of the relational expression:

1. a>b
2. a-b >= x-y
3. a+b>80

Unit 2: Introduction Of C++

Let's understand through an example

```
1. #include <iostream>
2. using namespace std;
3. int main()
4. {
5.     int a=45;    // variable declaration
6.     int b=78;    // variable declaration
7.     bool y= a>b;   // relational expression
8.     cout<<"Value of y is :"<<y;  // displaying the value of y.
9.     return 0;
10. }
```

In the above code, we have declared two variables, i.e., 'a' and 'b'. After declaration, we have applied the relational operator between the variables to check whether 'a' is greater than 'b' or not.

Output

```
Value of y is :0
```

Let's see another example.

```
1. #include <iostream>
2. using namespace std;
3. int main()
4. {
5.  int a=4;     // variable declaration
6.  int b=5;     // variable declaration
7.  int x=3;     // variable declaration
8.  int y=6;    // variable declaration
9.  cout<<((a+b)>=(x+y));   // relational expression
10. return 0;
11. }
```

In the above code, we have declared four variables, i.e., 'a', 'b', 'x' and 'y'. Then, we apply the relational operator (>=) between these variables.

Output

Faculty: SHAHRUKH KAMAL
Shahrukhkamal7@gmail.com

```
1
```

Logical Expressions

A logical expression is an expression that combines two or more relational expressions and produces a bool type value. The logical operators are '&&' and '||' that combines two or more relational expressions.

The following are some examples of logical expressions:

1. a>b && x>y
2. a>10 || b==5

Let's see a simple example of logical expression.

```
1. #include <iostream>
2. using namespace std;
3. int main()
4. {
5.  int a=2;
6.  int b=7;
7.  int c=4;
8. cout<<((a>b)||(a>c));
9. return 0;
10. }
```

Output

```
0
```

Bitwise Expressions

A bitwise expression is an expression which is used to manipulate the data at a bit level. They are basically used to shift the bits.

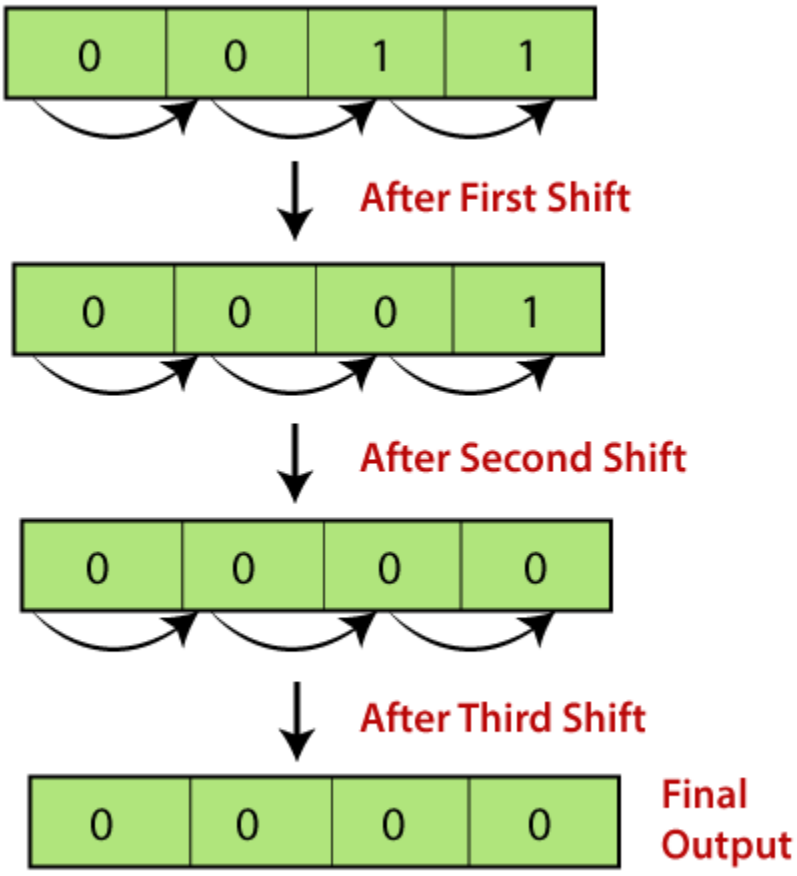
For example:

x=3

x>>3 // This statement means that we are shifting the three-bit position to the right.

Unit 2: Introduction Of C++

In the above example, the value of 'x' is 3 and its binary value is 0011. We are shifting the value of 'x' by three-bit position to the right. Let's understand through the diagrammatic representation.



Let's see a simple example.

```
1. #include <iostream>
2. using namespace std;
3. int main()
4. {
5.  int x=5;   // variable declaration
6. std::cout << (x>>1) << std::endl;
7. return 0;
8. }
```

In the above code, we have declared a variable 'x'. After declaration, we applied the bitwise operator, i.e., right shift operator to shift one-bit position to right.

Output

```
2
```

Let's look at another example.

```
1. #include <iostream>
2. using namespace std;
3. int main()
4. {
5.  int x=7;   // variable declaration
6. std::cout << (x<<3) << std::endl;
7. return 0;
8. }
```

In the above code, we have declared a variable 'x'. After declaration, we applied the left shift operator to variable 'x' to shift the three-bit position to the left.

Output

```
56
```

Special Assignment Expressions

Special assignment expressions are the expressions which can be further classified depending upon the value assigned to the variable.

- **Chained Assignment**

Chained assignment expression is an expression in which the same value is assigned to more than one variable by using single statement.

For example:

```
1. a=b=20
2.  or
3. (a=b) = 20
```

Let's understand through an example.

Unit 2: Introduction Of C++

```
1. #include <iostream>
2. using namespace std;
3. int main()
4. 
5.  int a;   // variable declaration
6.  int b;   // variable declaration
7.  a=b=80;  // chained assignment
8.  std::cout <<"Values of 'a' and 'b' are : " <<a<<","<<b<< std::endl;
9.  return 0;
10. }
```

In the above code, we have declared two variables, i.e., 'a' and 'b'. Then, we have assigned the same value to both the variables using chained assignment expression.

Output

```
Values of 'a' and 'b' are : 80,80
```

Note: Using chained assignment expression, the value cannot be assigned to the variable at the time of declaration. For example, int a=b=c=90 is an invalid statement.

- **Embedded Assignment Expression**

An embedded assignment expression is an assignment expression in which assignment expression is enclosed within another assignment expression.

Let's understand through an example.

```
1. #include <iostream>
2. using namespace std;
3. int main()
4. {
5.  int a;  // variable declaration
6.  int b;  // variable declaration
7.  a=10+(b=90);  // embedded assignment expression
8.  std::cout <<"Values of 'a' is " <<a<< std::endl;
9.  return 0;
10. }
```

In the above code, we have declared two variables, i.e., 'a' and 'b'. Then, we applied embedded assignment expression (a=10+(b=90)).

Output

```
Values of 'a' is 100
```

- **Compound Assignment**

A compound assignment expression is an expression which is a combination of an assignment operator and binary operator.

For example,

```
1. a+=10;
```

In the above statement, 'a' is a variable and '+=' is a compound statement.

Let's understand through an example.

```
1. #include <iostream>
2. using namespace std;
3. int main()
4. {
5.   int a=10;   // variable declaration
6.   a+=10;    // compound assignment
7.   std::cout << "Value of a is :" <<a<< std::endl; // displaying the value of a.
8.   return 0;
9. }
```

In the above code, we have declared a variable 'a' and assigns 10 value to this variable. Then, we applied compound assignment operator (+=) to 'a' variable, i.e., a+=10 which is equal to (a=a+10). This statement increments the value of 'a' by 10.

Output

```
Value of a is :20
```

C++ 'Using' vs 'Typedef'

Unit 2: Introduction Of C++

C++ has two keywords that can be used to define new types: typedef and using. Both of these keywords allow you to create a new type name that can be used to declare variables, but they work in slightly different ways.

typedef is an older keyword that has been part of the C++ language since the beginning. It is used to create a new type name that is an alias for an existing type. For example, you could use typedef to create a new type named MyInt that is an alias for the built-in int type:

Syntax

```
typedef int MyInt;
```

After this definition, you can use MyInt to declare variables just like you would use int:

Snippet

```
MyInt x = 5;
```

Using is a newer keyword that was introduced in C++11. It works in a similar way to typedef, but it has a more flexible syntax and can be used in more places in your code. For example, you can use using to define a new type name in the same way as typedef:

Snippet

```
using MyInt = int;
```

This creates a new type named MyInt, which is an alias for the built-in int type. You can then use MyInt to declare variables just like you would use int.

In general, using is considered to be a more modern and flexible way to define new type names in C++. typedef is still supported for backward compatibility, but the newest code should use using instead.

Here is an example of using the using keyword to define a new type named MyInt that is an alias for the built-in int type:

C++ code (Example-1)

```
#include <iostream>

// Define a new type named MyInt that is an alias for int
using MyInt = int;
// The main driver code functionality ends from here
```

```
6.  int main()
7.  {
8.      // Declare a variable of type MyInt
9.      MyInt x = 5;
10.
11.     // Print the value of the variable
12.     std::cout << "x = " << x << std::endl;
13.
14.     return 0;
15. // The main driver code functionality ends from here
16. }
```

This code will print the following output:

```
x = 5
```

Note that using is not limited to defining type aliases for built-in types like int. You can use it to define type aliases for any type, including user-defined types, template types, and more.

C++ code (Example-2)

```
1.  #include <iostream>
2.  #include <vector>
3.
4.  using namespace std;
5.
6.  int main()
7.  {
8.      // Create a vector of integers with an initial capacity of 5 elements
9.      vector<int> numbers(5);
10.
11.     // Add some elements to the vector
12.     numbers.push_back(10);
13.     numbers.push_back(20);
14.     numbers.push_back(30);
15.     numbers.push_back(40);
16.     numbers.push_back(50);
```

```
17.
18.    // Print the size and capacity of the vector
19.    cout << "Size: " << numbers.size() << endl;
20.    cout << "Capacity: " << numbers.capacity() << endl;
21.
22.    // Remove the last element from the vector
23.    numbers.pop_back();
24.
25.    // Print the size and capacity of the vector after removing an element
26.    cout << "Size: " << numbers.size() << endl;
27.    cout << "Capacity: " << numbers.capacity() << endl;
28.
29.    // Clear all elements from the vector
30.    numbers.clear();
31.
32.    // Print the size and capacity of the vector after clearing
33.    cout << "Size: " << numbers.size() << endl;
34.    cout << "Capacity: " << numbers.capacity() << endl;
35.
36.    return 0;
37. }
```

Output:

```
Size: 10
Capacity: 10
Size: 9
Capacity: 10
Size: 0
Capacity: 10
```

C++ code (typedef example)

```
1.  #include <iostream>
2.  #include <vector>
3.  using namespace std;
4.
5.  // Create an alias for a vector of integers
6.  typedef vector<int> IntVector;
7.
```

```
8.  int main()
9.  {
10.    // Create a vector of integers using the alias
11.    IntVector numbers;
12.
13.    // Add some elements to the vector
14.    numbers.push_back(10);
15.    numbers.push_back(20);
16.    numbers.push_back(30);
17.
18.    // Print the elements of the vector
19.    for (IntVector::iterator it = numbers.begin(); it != numbers.end(); ++it)
20.    {
21.        cout << *it << " ";
22.    }
23.
24.    return 0;
25. }
```

Output:

```
10 20 30
```

In this code, we use the 'typedef' keyword to create an alias called 'IntVector' for a 'vector' of integers. We then use this alias to create a 'vector' object and add some elements to it.