C++ Concepts:

Variables:

1. Variables are containers for storing data values.Variables in C++ is a name given to a memory location.
2. It is the basic unit of storage in a program.
3. The value stored in a variable can be changed during program execution.
4. A variable is only a name given to a memory location, all the operations done on the variable effects that memory location.
5. In C++, all the variables must be declared before use.
6. In C++, there are different **types** of variables (defined with different keywords), for example:

int - stores integers (whole numbers), without decimals, such as 123 or -123
double - stores floating point numbers, with decimals, such as 19.99 or -19.99
char - stores single characters, such as 'a' or 'B'. Char values are surrounded by single quotes
string - stores text, such as "Hello World". String values are surrounded by double quotes
bool - stores values with two states: true or false

Declaring (Creating) Variables

To create a variable, specify the type and assign it a value:

Syntax:

```
type variableName = value;
```

Where type is one of C++ types (such as int), and variableName is the name of the variable (such as **x** or **myName**).

The **equal sign** is used to assign values to the variable.

// Declaring multiple variables:

```
type variable1_name, variable2_name, variable3_name;
```

A variable name can consist of alphabets (both upper and lower case), numbers, and the underscore '_' character. However, the name must not start with a number.

```
// Declaring float variable

float simpleInterest;

// Declaring integer variable

int time, speed;

// Declaring character variable

char var;
```

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

```
int a=50,b=100; //declaring 2 variable of integer type

float f=50.8; //declaring 1 variable of float type char

c='Z'; //declaring 1 variable of char type
```

Rules For Declaring Variable
The name of the variable contains letters, digits, and underscores.
The name of the variable is case sensitive (ex Arr and arr both are different variables).
The name of the variable does not contain any whitespace and special characters (ex #,$,%,*, etc).
All the variable names must begin with a letter of the alphabet or an underscore(_).

Valid variable names:

```
int x;   //can be letters

int _yz; //can be underscores

int z40;//can be letters and digits
```

Invalid variable names:

```
int 89; Should not be a number
int a b; //Should not contain any whitespace
int double;// C++ keyword CAN NOT BE USED
```

Difference Between Variable Declaration and Definition

The variable declaration refers to the part where a variable is first declared or introduced before its first use.

A variable definition is a part where the variable is assigned a memory location and a value.

Most of the time, variable declaration and definition are done together.
See the following C++ program for better clarification:

C++

```
   // definition and declaration of a variable

   #include <iostream>

   using namespace std;

   int main()

   {
        int a;       //this is declaration of a variable a

        a= 10;       //this is initialisation of a

        int b = 20; //this is definition = declaration + initialisation

     // declaration and definition

     // of variable 'a123'

     char a123 = 'a';

        int _c, _d45, e; //multiple declarations

               // Let us print a variable

               cout << a123 << endl;

               return 0;

}
```

Output: a

You can also declare a variable without assigning the value, and assign the value later:

Example:

```
int myNum;
myNum = 15;
cout << myNum;
```

Note that if you assign a new value to an existing variable, it will overwrite the previous value:

Example:

```
int myNum = 15; // myNum is 15
myNum = 10; // Now myNum is 10
cout << myNum; // Outputs 10
```

C++ Declare Multiple Variables

Declare Many Variables

To declare more than one variable of the same type, use a comma-separated list:

Example

```
int x = 5, y = 6, z = 50;
cout << x + y + z;
```

One Value to Multiple Variables

You can also assign the same value to multiple variables in one line:

Example:

```
int x, y, z;
x = y = z = 50;
cout << x + y + z;
```

C++ Identifiers

All C++ variables must be identified with unique names.
These unique names are called identifiers. Identifiers can be short names (like x and y) or more descriptive names (age, sum, totalVolume).

Note: It is recommended to use descriptive names in order to create understandable and maintainable code:

Example

```
// Good
int minutesPerHour = 60;

// OK, but not so easy to understand what m actually is
int m = 60;
```

The general rules for naming variables are:

Names can contain letters, digits and underscores Names must begin with a letter or an underscore (_) Names are case sensitive (myVar and myvar are different variables)

Names cannot contain whitespaces or special characters like !, #, %, etc.
Reserved words (like C++ keywords, such as int,double) cannot be used as names.

Assignment Statements

Assignment statement are used to assign values to variables. In the example below, we use the assignment operator (=) to assign the value 10 to a variable called x:

Example

```
int x = 10;
```

Output: 10

reference variable

Reference variable is an alternate name of already existing variable.

It cannot be changed to refer another variable and should be initialized at the time of declaration and cannot be NULL.

The operator ‘&’ is used to declare reference variable.

The following is the syntax of reference variable.

```
datatype variable_name; // variable declaration

datatype& refer_var = variable_name; // reference variable
```

Here,

datatype − The datatype of variable like int, char, float etc.

variable_name − This is the name of variable given by user.

refer_var − The name of reference variable.

The following is an example of reference variable.

Example:

```
#include <iostream>
using namespace std;
int main() {
   int a = 8;
   int& b = a;
   cout << "The variable a : " << a;
   cout << "\nThe reference variable r : " << b;
   return 0;
}
```

Output

```
The variable a : 8
The reference variable r : 8
```

In the above program, a variable of integer type is declared and initialized with a value.

```
int a = 8;
```

The variable b is declared which is referring variable a.

```
int& b = a;
```

Symbolic constant

Constants

When you do not want others (or yourself) to change existing variable values, use the const keyword (this will declare the variable as "constant", which means unchangeable and read-only):

Example

```
const int myNum = 15; // myNum will always be 15
myNum = 10; // error: assignment of read-only variable 'myNum'
```

You should always declare the variable as constant when you have values that are unlikely to change:

Example

```
const int minutesPerHour = 60;
const float PI = 3.14;
```

Output:

```
60
3.14
```

Input and Output: cin, cout

You have already learned that cout is used to output (print) values. Now we will use cin to get user input.
cin is a predefined variable that reads data from the keyboard with the extraction operator (>>).
In the following example, the user can input a number, which is stored in the variable x.
Then we print the value of x:

Example

```
int x;
cout << "Type a number: "; // Type a number and press enter
cin >> x; // Get user input from the keyboard
cout << "Your number is: " << x; // Display the input value
```

Output:

```
Type a number: Your number is: 2
```

Good To Know:

cout is pronounced "see-out". Used for output, and uses the insertion operator (<<)
cin is pronounced "see-in". Used for input, and uses the extraction operator (>>)

Creating a Simple Addition of two numbers:

In this example, the user must input two numbers.
Then we print the sum by calculating (adding) the two numbers:

Example

```
#include <iostream>

using namespace std;

int main() {

  int x, y;

  int sum;

  cout << "Type a number: ";

  cin >> x;

  cout << "Type another number: ";

  cin >> y;

  sum = x + y;

  cout << "Sum is: " << sum;

  return 0;

}
```

Output:

```
Type a number: 2
Type another number: 3 Sum is: 5
```

Escape Sequences:

The escape sequences are special non-printing characters that are used to control the printing behavior of the output stream objects (such as ‘cout’).

These characters are not displayed in the output.

An escape sequence is prefixed with a backslash () and a coded character is used to control the printing behavior.

The backslash \ is called an escape character.

The escape sequence is used inside a string constant or independently.

These are written in single or double-quotes.

The escape sequence can be inserted in any position of the string such as:

At the beginning of the string.

In the middle of the string.

At the end of the string etc.

For example, the escape sequence ‘\n’ is used to insert a new line.

The cursor moves from the current position on the output device to the beginning of the next line.

If escape sequence ‘\n’ is inserted at the beginning of the string then the string will be printed after printing a blank line e.g.

Cout<<”\nWelcome”;

```
#include <iostream>
using namespace std;
int main() {
string txt = "We are the so-called "Vikings" from the north.";
cout << txt;return 0;
}
```

//string txt = "We are the so-called "Vikings" from the north.";
The above line that was mentioned in the above code will generate an error.
Because its a syntax error.

The solution to avoid this problem, is to use the **backslash escape character**.

The backslash (\) escape character turns special characters into string characters:

Escape character	Result	Description
\'	'	Single quote
\"	"	Double quote
\\	\	Backslash

The sequence \" inserts a double quote in a string:

Example

```
string txt = "We are the so-called \"Vikings\" from the north.";
```

Output: We are the so-called "Vikings" from the north.

The sequence \' inserts a single quote in a string:

Example

```
string txt = "It\'s alright.";
```

Output: It's alright.

The sequence \\ inserts a single backslash in a string:

Example

```
#include <iostream>
using namespace std;
int main() {
  string txt = "The character \\ is called backslash.";
  cout << txt;
  return 0;
}
```

Output:The character \ is called backslash.

Other popular escape characters in C++ are:

Escape Character	**Result**
\n	New Line
\t	Tab

```
#include <iostream>
using namespace std;
int main() {
        string txt = "Hello\nWorld!";
        cout << txt;
        return 0;
        }
```

Output:

```
Hello
World!
```

Another way to insert a new line, is with the endl manipulator:

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

int main() {
  cout << "Hello World!" << endl;
  cout << "I am learning C++";
  return 0;
}
```

Output:

```
Hello World!
I am learning C++
```

This program use \t (tab escape sequence) to insert tab in output. Here we use two cout (output) statements, one with \t and other without \t to see the impact on output.

```
#include <iostream>

using namespace std;

int main() {

      / Output without Escape Sequence
    cout<<" First Second Third Forth Fifth";

      / Output with Escape Sequence
    cout<<" First \t Second \t Third \t Forth \t Fifth";


return 0;
}
```

Output:

```
&& "/Desktop/IT Dept/FYIT/SEM 1/C++ Program/"areaofrectangle
 First Second Third Forth Fifth
 First   Second         Third   Forth   Fifth%
```

Include Directives and Namespaces

Using Directives

You use the "using" directive to import the entire namespace into a program or another namespace.

This directive eliminates the need to use the "using namespace" every time.

So it is better to use the "using" if you need several functions or classes from the namespace. Otherwise, if you only need to use it once or twice, "using namespace" would be a better choice.

Standard Namespace

The std is a short form of standard, the std namespace contains the built-in classes and declared functions.

You can find all the standard types and functions in the C++ "std" namespace. There are also several namespaces inside "std."

Example **without using namespace**

```
#include<iostream>
int main()
{
std:: cout <<”Enter the numbers to multiply:”;
int num1 = 3;
int num2 = 4;
std::cin>>num1;
std::cin>>num2;
std::cout<<"The product of two numbers is:"<<(num1*num2);
return 0;
}
```

Here std is used in front of cin and cout along with scope resolution operator, which indicates that the object cin and cout are defined inside the namespace whose name is std.

The std is the standard library, and both cin and cout are defined inside this scope.

using namespace std

It means that we can use names for objects and variables from the standard library.

Exmaple:

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

int main() {
  cout << "Hello C++";
  return 0;
}
```

Indenting and Comments,

Indentation refers to the spaces at the beginning of a code line.

```
int main(){
        int number1, number2;
        cout<<"Enter the two Integers:";
```

Where in other programming languages the indentation in code is for readability only.

Comments

Comments can be used to explain C++ code, and to make it more readable.
It can also be used to prevent execution when testing alternative code.
Comments can be singled-lined or multi-lined.

Single-line Comments

Single-line comments start with two forward slashes (//).
Any text between // and the end of the line is ignored by the compiler (will not be executed).
This example uses a single-line comment before a line of code:

Example

```
// This is a comment cout
<< "Hello World!";
```

This example uses a single-line comment at the end of a line of code:

Example

```
cout << "Hello World!"; // This is a comment
```

Multi-line Comments

Multi-line comments start with /* and ends with */.
Any text between /* and */ will be ignored by the compiler:

Example

```
/* The code below will print the words Hello World!
to the screen, and it is amazing */
cout << "Hello World!";
```

Single or multi-line comments?
It is up to you which you want to use. Normally, we use // for short comments, and /* */ for longer.

Operator Precedence:

Operator precedence determines the grouping of terms in an expression.
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.

The associativity of an operator is a property that determines how operators of the same precedence are grouped in the absence of parentheses. This affects how an expression is evaluated.
Certain operators have higher precedence than others.
If there are multiple operators in a single expression, the operations are not evaluated simultaneously. Rather, operators with higher precedence have their operations evaluated first.

for example, the multiplication operator has higher precedence than the addition operator:
For example x = 7 + 3 * 2; here, x is assigned 13, not 20 because operator * has higher precedence than +, so it first gets multiplied with 3*2 and then adds into 7.

Here, operators with the highest precedence appear at the top of the table, those with the lowest appear at the bottom. Within an expression, higher precedence operators will be evaluated first.

Let us consider an example:

```
int x = 5 - 17 * 6;
```

Here, the multiplication operator * is of higher level precedence than the subtraction operator -. Hence, 17 * 6 is evaluated first.

As a result, the above expression is equivalent to

```
int x = 5 - (17 * 6);
```

If we wish to evaluate 5 - 17 first, then we must enclose them within parentheses:

```
int x = (5 - 17) * 6;
```

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

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

C++ Operators Associativity

Example 2: Operators Associativity

```
#include <iostream>
using namespace std;
int main() {
  int a = 1;
  int b = 4;
// a -= 6 is evaluated first
  b += ;
a -= 6;
//a=a-6
//a=1-6
//a=-5
//b=b+6
//b=4+6
b+=a-=6;  //a = -5 , //b= -1
  cout << "a = " << a << endl; ;
  cout << "b = " << b;
}
```

Output

```
a = -5
b = 10
```