Getting Started

For this course, you will be provided with the in-built compiler of Coding Ninjas. However, if you want to run programs and practice them on your local desktop, there are various compilers out there like Code blocks, VS Code, Dev C++, Atom and many more. We have provided the steps for installing Code Blocks in a separate file.

About Code blocks

Code blocks is an Integrated Development Environment (IDE) for C/C++. To set the path of compiler, follow the given steps:

1. Click on menu **Settings** -> **Compiler.**
2. Then click on the tab **Toolchain Executables.**
3. In the text box under **Compiler's installation directly,** click on the button **Auto Detect**. A pop-up appears.
4. In case, pop-up says, **Couldn't auto detect...,** that means you have downloaded the incorrect setup of code blocks. Uninstall this setup, and install the setup with **MinGW**. Then repeat the above steps.

To create a new file:

1. Follow **File -> New -> Empty File.**
2. Then save that file with extension **.cpp**
3. In order to run and compile the program press F11 or click on the button right next to a play button which says **Build and Run**.

You will get your output window after following step 3.

Looking at the code

C++ code begins with the inclusion of header files. There are many header files available in the C++ programming language which you will discuss while moving ahead with the course.

So, what are these header files?

The names of program elements such as variables, functions, classes, and so on must be declared before they can be used. For example, you can't just write x = 42 without first declaring variable 'x' as:

```
int x = 42;
```

The declaration tells the compiler whether the element is an int, a double, a float, a function or a class. Similarly, header files allow us to put declarations in one location and then import them wherever we need them. This can save a lot of typing in multi-file programs. To declare a header file, we use **#include** directive in every .cpp file. This #include is used to ensure that they are not inserted multiple times into a single .cpp file.

Note: # operator is known as **Macros.**

The following are not allowed, or are considered as bad practices while putting header files into the code:

- Built-in-type definitions at namespace or global scope
- non-inline function definitions
- Non-const variable definitions
- Aggregate definitions
- Unnamed namespaces
- Using directives

Now, moving forward to the code:

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

iostream stands for Input/Output stream, meaning this header file is necessary if you want to take input through the user or print output to the screen. This header file contains the definitions for the functions:

- **cin** : used to take input
- **cout** : used to print output

namespace defines which input/output form is to be used. You will understand these better as you progress in the course.

Note: semicolon (;) is used for terminating a C++ statement. ie. different statements in a C++ program are separated by semicolon

main() function:

Look at the following piece of code:

```
int main() {

    Statement 1;
    Statement 2;
    ...
}
```

You can see the highlighted portion above. Let's discuss each portion stepwise.

Starting with the line:

```
int main()
```

- **int** : This is the return-type of the function. You will get this thing clear once you reach the **Functions** topic.

- **main()** : This is the portion of any C++ code inside which all the commands are written and gets executed.
    - This is the line at which the program will begin executing. This statement is similar to the start block of flowcharts.
    - As you will move further in the course, you will get a clear glimpse of what this function is. Till then, just note that you will have to write all the programs inside this block.
- **{}** : all the code written inside the curly braces is said to be in one block, also known as scope of a particular function. Again, these things will be clear when you will study functions.

For now, just understand that this is the format in which we are gonna write our basic C++ code. From time to time as you will move forward with the course, you will get a clear and better understanding.

Declaring a variable:

To declare a variable, we should always know what type of value it should hold, whether it's an integer (int), decimal number (float, double), character value (char). In general, the variable is declared as follows:

Datatype variableName = VALUE;

Note: Datatype is the type of variable:

- int : Integer value
- float, double : Decimal number
- char : Character values (including special characters)
- bool : Boolean values (true or false)
- long : Contains integer values but with larger size
- short : Contains integer values but with smaller size

Table for datatype and its size in C++: (This can vary from compiler to compiler and system to system depending on the version you are using)

Datatype	Default size
bool	1 byte
char	1 byte
short	2 bytes
int	4 bytes
long	8 bytes
float	4 bytes
double	8 bytes

For example: To declare an integer variable ‘a’ with a value of 5, the structure looks like:

```
int a = 5;
```

Similarly, this way other types of variables can also be declared. There is one more type of variable known as **string** variables which store combinations of characters. You will study that in your further lectures.

Rules for variable names:

- Can’t begin with a number.
- Spaces and special characters except underscore(_) are not allowed.
- C++ keywords (reserved words) must not be used as a variable name.
- C++ is case-sensitive, meaning a variable with name ‘A’ is different from variable with name ‘a’. (Difference in the upper-case and lower-case holds true)

Printing/Providing output:

For printing statements in C++ programs, we use the **cout** statement**.**

For example: If you want to print “Hello World!” (without parenthesis) in your code, we will write it in following way:

```
cout << “Hello World!”;
```

A full view of the basic C++ program is given below for the above example:

Code:

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

int main() {
    cout<<”Hello World!”;
}
```

Output:

```
Hello World!
```

Line separator:

For separating different lines in C++, we use **endl** or **‘\n’**.

For example:

Code:

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

int main() {
    cout<<"Hello World1"<<endl;
    cout<<"Hello World2"<<'\n';

}
```

Output:

```
Hello World1
Hello World2
```

Taking input from the user:

To take input from the user, we use the **cin** statement.

For example: If you want to input a number from the user:

```
int n;
cin >> n;
```

A full view of the basic C++ program in which you want to take input of 2 numbers and then print the sum of them:

Code:

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

int main() {
    int a, b, sum;
    cin >> a;
    cin >> b;
    sum = a + b;
    cout << "Sum of two numbers: ";
    cout << sum << endl;
}
```

Input:

```
1
2
```

Output:

```
Sum of two numbers: 3
```

Operators in C++

There are 3 types of operators in C++

- Arithmetic operators
- Relational operators
- Logical operators

Discussing each of them in detail...

Arithmetic operators:

These are used in mathematical operations in the same way as that in algebra.

OPERATOR	DESCRIPTION
+	Add two operands
-	Subtracts second operand from the first
*	Multiplies two operands
/	Divides numerator by denominator
%	Calculates Remainder of division

Relational operators:

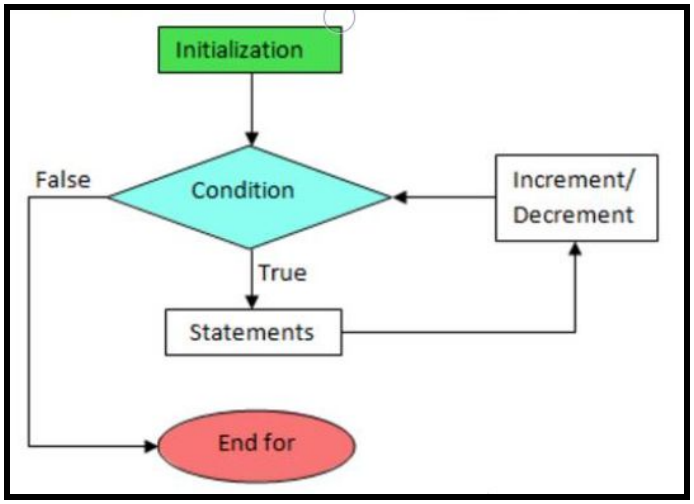
C++ relational operators specify the relation between two variables by comparing them.

Following table shows the relational operators that are supported by C++.

OPERATOR	DESCRIPTION
==	Checks if two operands are equal
!=	Checks if two operands are not equal
>	Checks if operand on the left is greater than operand on the right
<	Checks if operand on the left is lesser than operand on the right
>=	Checks if operand on the left is greater than or equal to operand on the right
<=	Checks if operand on the left is lesser than or equal to operand on the right

Logical operators:

C++ supports 3 types of logical operators. The result of these operators is a boolean value i.e., True(BITWISE ‘1’) or False(BITWISE ‘0’). Refer the visual representation below:



OPERATOR	DESCRIPTION

&&	Logical AND
||	Logical OR
!	Logical NOT

How is data Stored ?

- **For integers:**

  The most commonly used is a signed 32-bit integer type. When you store an integer, its corresponding binary value is stored. There is a separate way of storing positive and negative numbers. For positive numbers the integral value is simply converted into binary value while for negative numbers their 2's complement form is stored.

  ### For negative numbers:

  Computers use 2's complement in representing signed integers because:

  - There is only one representation for the number zero in 2's complement, instead of two representations in sign-magnitude and 1's complement.
  - Positive and negative integers can be treated together in addition and subtraction. Subtraction can be carried out using the **addition logic.**

  For example: int i = -4;

  Steps to calculate Two's Complement of -4 are as follows:

  Step1: Take Binary Equivalent of the positive value (4 in this case)

  ```
  0000 0000 0000 0000 0000 0000 0000 0100
  ```

  Step2: Write 1's complement of binary representation by inverting the bits

```
1111 1111 1111 1111 1111 1111 1111 1011
```

Step3: Find 2's complement by adding 1 to the corresponding 1's complement

```
 1111 1111 1111 1111 1111 1111 1111 1011
+0000 0000 0000 0000 0000 0000 0000 0001
-----------------------------------------------------------------
 1111 1111 1111 1111 1111 1111 1111 1100
```

Thus, integer -4 is represented by the above binary sequence in C++.

- **For float and double values:**

  In C++, any value declared with a decimal point is by-default of type double (which is generally of 8-bytes). If we want to assign a float value (which is generally of 4-bytes), then we must use 'f' or 'F' literal to specify that the current value is "float".

  For example:

```
float var = 10.4f          // float value
Double val = 10.4          // double value
```

- **For character values:**

  Every character has a unique integer value, which is called ASCII value. As we know, systems only understand binary language and thus everything has to be stored in the form of binaries. So, for every character there is a corresponding integer code-ASCII code and the binary equivalent of this code is actually stored in memory when we try to store a character.

  For ASCII values, refer the link below:

  https://ascii.cl/

- **Adding int to char**

When we add int to char, we are basically adding two numbers i.e., one corresponding to the int and the other corresponding to the ASCII code for the character.

For example:

Code:

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

int main () {
    cout<< 'a' + 1;
}
```

Output:

```
98
```

Explanation:

The **ASCII value of 'a' is 97**, so it printed **97+1 = 98** as the output.