C++

- #include <iostream> → Imports the iostream library, providing input/output like cout.
- using namespace std → Let's you use standard elements like cout directly (instead of std::cout)
- main function : int main() {---} → core of every c++ program; execution starts here.
- // → double dash, used for single line comment
- /* */ → multi line comment
- You can perform methamatical operation like (addition, subtraction, multiplication etc) with cout

  eg → cout << 21+40;
- output a text using double quotes (" ")

  eg → cout << "Hey Everyone";
- endl → Keyword used to go to newline in the output
- " " → can be used for empty space

  Eg → cout << 2 << " " << 2;
- A variable is like a labelled box where you can store data.

  int age = 25;

Data Types

int	Stores integers	4
long long	stores large integers	8
float	stores decimal numbers	4
double	stores large decimal numbers	8
char	stores single character	1
bool	stores true & false	1

- A variable can only contain alphabets, numbers and underscores.
- A variable name cannot start with a number.
- A variable name cannot have spaces in between.
- variable names are case-sensitive.
- variable names cannot be of the reserved keywords in C++.
- fixed modifier → This allows us to output numbers in fixed-point notation, we generally get 6 digits after decimal with this.

Eg →

```
cout << fixed << num << endl;
```

Output → 9152345.678912

- setprecision modifier → We can set the number of digits to be printed after the decimal.

Eg →

```
double num = 9152345.6789118;
cout << setprecision(9) << num << endl;
```

Output → 9152345.678911800

Type conversion

we can convert one data type into another as per our use.

There are two type:

- Implicit Type conversion

Here, C++ automatically converts a variable or a value to the appropriate data type itself

* Eg →

```
int x = 7;
float y = x;
```

Here the integer 7, which is stored in x is converted to a float and stored in y as 7.0

- Explicit Type conversion

Here, we manually convert a value to another type of data as per our use.

* Eg →

```
float num = 7.89;
cout << int(num) << endl;
```

Here, we declare a float with value 7.89. Then we convert it to integer the output it. This converts 7.89 to integer and output 7.

Integer to boolean conversion

0 is false, and anything other then 0 is true

* Eg →

```
cout << bool(0) << endl;
cout << bool(1) << endl;
cout << bool(-83) << endl;
cout << bool(0.003) << endl;
```

output →

```
0
1
1
1
```

Ascii

When we convert 'C' to integer it gave 67 and when we convert 68 to character it gives 'D'.
This is because each character has an integer value also known as it's ASCII value.

- Lower case and upper case have different ascii values.

Character Operations

As characters are stored in integer form, we can also manipulate characters using arithmetic operations.

eg →

```
cout<< char('a' + 2);
cout << char('b' - 1);
```

output → c a

as 'a' is stored as 97, we perform 'a'+2 so it is 99 which is 'c' same when we do 'b'-1, it is 'a'.

Converting uppercase to lowercase

- The ascii value of upper case letters lies between 65 and 90 with 65 being 'A' and 90 being 'Z'.
- Similarly the ascii value of lower case letters lies between 97 and 122. with 97 being 'a' and 122 being 'z'

Thus we can convert 'A' (65) to 'a' (97) by adding 32.

eg

```
char ch = 'A';
cout << char(ch+32) << endl;
```

output → a

But we don't need to remember this value to be added to convert an upper case letter to lower case.
This is just the difference between the ascii value of the lower case and their upper counter parts.

Eg →
```
char ch = 'D';
char lower_case = char(ch + ('a' - 'A'));
cout << lower_case << endl;
```
Output d

lower case to upper case

Eg →
```
char ch = 'r';
char upper_case = char(ch + ('A' - 'a'));
Cout upper_case << endl;
```
Output R

Arithmetic operations

use + for addition:
```
int a = 6;
int b = 3;
cout << a+b;
```
output 9

use - for subtraction:
```
cout<<a-b;
```
output 3

use * for multiplication:
```
cout << a * b;
```
output 18

use / (forward slash) for division:
```
cout << a/b;
```
output 2

use % (percent symbol, called modulo operator) for the divisi
reminder
```
cout << a % b;
```
output 0

use ++ for increasing the value of a variable by 1:
```
a++;
cout <<a;
```
output 7

use -- for decreasing the value of a variable by 1:
```
b--;
Cout << b;
```
output 2

Assignment operator

Help you to set or change the value of a variable.

*Basic assignment

```
int length;
length = 15;
```

* Compound Assignment operators

It is just the shorthand way of performing operations on a variable and assigning the result back to the variable.

- without using the compound assignment operator we write -
  - length = 15;
  - length = length + 5;
- using compound assignment operator
  - length = 15;
  - length += 5;

other operators

- x -= 5 (subtract 5 from x and assigns the result back to x)
- x *= 3 (multiply x by 3 and assigns the result back to x)
- x /= 3 (divides x by 3 and assigns the result back to x)
- x %= 3 (finds the reminder when x is divided by 3 and assigns the result back to x)

Increment and Decrement Operator

- Pre Increment / Decrement operator → Increase or decrease the value by 1 first and then returns the value.

```
int a = 5;
cout << ++a << endl;
```

output → 6

*It increases the value to 6 first then output the value.

- Post Increment / Decrement operator → Return the value then increase or decrease the value by 1.

```
int a = 5;
cout << a++ << endl;
cout << a << endl;
```

output → 5
6

- It output the value first, so the first output is 5.
- Then it increments the value, which becomes 6.
- Then the incremented value is outputted.

we can also increment or decrement operators.

```
Char ch = 'a';
ch++;
cout << ch << endl;
```

Output → b

Relational Operators → Helpers to compare two values or variables.

gives us a result of either 1(true) or 0(false) based on the comparison.

Eg →

```
int height1 = 12;
int height2 = 14;
cout << (height1 < height2);
```

Output → 1

* It is important to enclose the condition in brackets else C++ will throw an error.

Other Relational operators

- a > b
- a == b
- a != b
- a >= b
- a <= b

* logical operators help you combine multiple conditions to check if they are True or False. They are often used to make decisions based on multiple criteria.

&& (logical AND: returns 1(true) only if both conditions are true)
(a > 5 && a < 10)

|| (logical OR: returns 1 if at least 1 cond^n is true)
(a > 15 || a < 5)

! (logical NOT: reverses the result of the cond^n)
(!(a > 5))

Operator Precedence and Associativity

- Precedence → It tells the priority of operations.
- Associativity → It tells what order the operations with the same priority will be performed in.

→ Operator precedence means that some operations are done before others when evaluating an expression. Operators with higher precedence are evaluated before those with lower precedence.

→ order of operator precedence in C++ from highest to lowest.

- Paranthesis: ( )
- Postfix operators: ++, --
- unary operators: +, -, !, ~, ++, --, (type)
- multiplicative operators: *, /, %
- Additive operators: +, -
- Relational operators: <, >, <=, >=
- Equality operators: ==, !=
- Logical AND operator: &&
- Logical OR operator: ||
- Assignment operator: =, +=, -= ... and so on

→ Operator Associativity

when an expression contains operators of the same precedence level, associativity determines their order of evaluation.

- Left-Associative: Operators are evaluated from left to right. For instance a+b-c, addition and subtraction, being left-associative, will first evaluate a+b, and then subtract c from the result
- Right-Associative: Although less common in C++, some operators are right-associative, meaning they are evaluated from right to left. An example is the assignment operator =

In a=b=c, c is assigned to b, and then the resulting value of b is assigned to a.

String

Creating a String

Strings are used for storing text.

A string variable contains a collection of characters surrounded by double quotes. Before declaring a variable as string the string datatype must be used.

Eg →

```
string name= "Chef";
cout << name;
```

Output → Chef

Concatenation

The "+" sign can be used between strings to add them together to make a new string.

This is called concatenation.

Eg →

```
string x= "Good";
string y= "Work";
cout << x+y;
```

Output → GoodWork

→ You can add space between words by using an empty double quotes (" ") with a space inside or by providing a space in the end of a word.

Eg →

```
cout << x + " " + y;
```

Output → Good Work

String length

We can use the length() function to get the length of a string.

We can use the length() function to get the length of a string.

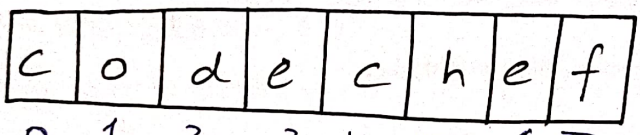
Eg →

```
string s = "hello";
int a = s.length();
cout << a;
```

output → 5

Printing characters from a string.

We can access the characters in a string by referring to its index number inside the square brackets []. We use the concept of indexing which means every character in a string corresponds to a number called the index number.



Index no:s are 0 1 2 3 4 5 6 7

- Indexing always starts from 0 when going left to right.
- So the first character of a string s is s[0], the second is s[1] and so on.

Changing characters in a String

To change the value of a specific character in the string.

- We refer to the index number to access the specific character using the indexing operator [].
- We then assign the new character (enclosed in single quotes '') to it.

Eg →

```
string myString = "chef";
myString[2] = 'e';
cout << myString;
```

Output → chef

Substring

A substring is a small part of a string.
For example:
"World" is a substring of the string "Hello World!"
In C++, the substr() is a string function which is used to extract a substring from a given string.
It takes two values:

- the starting index of the substring
- the number of characters in the substring

Eg →

```
string str = "Hello, World!";
cout << str.substr(7,5);
```

output → World

It goes to 7th index which is 'W' and takes 5 characters starting from 'W', which results in 'World'.
We can even give only the starting index to the substr function.
In that case it will start from the starting index and go till the end of the string.

→ `cout << str.substr(5);`

output → , World!

Taking User Inputs

We have already seen that cout is used to output (print) values.
Now, we will use cin to get user output.

* cin is predefined keyword that reads data from the keyboard with the extraction operator >>.