1. Introduction

1. Basic Input Output

- Iostream is the header file that allows us to display output and accept input from the console.
- Cout is defined inside the std namespace. To use std namespace we use 'using namespace std'.
- Return 0; is the exit status of the main function.
- Semicolon is the statement terminator.
- End is used to insert a new line.

2. Variables & Literals

A variable is a container to hold data.
Int sum = 0;
sum = 20;

Rules for naming variables:

- A variable name can only have alphabets, numbers and underscore.
- A variable name cannot begin with a number.
- However variable name can start with underscore.
- It is preferred to begin variable names with lowercase.
- A variable name cannot be a keyword. (**Keyword** : The collection of words whose meaning is already explained to the compiler. e.g. int, char, double, continue, etc.)
- Always give meaningful names. e.g. For first name use first_name rather than fn.

List of different literals in C++

a. Integers
   a. Decimal (Base 10) 0, -10, 25, etc.
   b. Octal (Base 8 : 0 - 7) 0o21, 0o77, 0o35, etc.

c. Hexadecimal (Base 16: 0-9,A,B,C,D,E,F) 0x7F, 0x51B, etc.

b. Floating Point

-2, 0.0012, 2e-5 = 2 * 10^-5 = 0.00002

c. Character

'a', 'B', '2', '{', ';'

d. Escape Characters

\r Carriage Return(CR)
\n Newline (Line feed) (LF)
\t Tab

e. String

"good", "y", "Earth is spherical\n"

f. Constants

const int LIGHT_SPEED = 3e8;
LIGHT_SPEED = 4e8; /* Error! As LIGHT_SPEED is constant.

3. Data Types

Fundamental Data types:

Int :

2 or 4 bytes (1 byte = 8 bits) Usually 4 bytes
4 bytes and range is -2147483648 to 2147483647
1 byte = 8 bits
4 bytes = 8 * 4 bits = 32 bits
Leading(left most) bit is preserved for sign
(0 : Positive number,
1: Negative number)
Max signed integer that C++ can support = 2^31 - 1 = 2147483647
int salary = 50000;

Float : 4 bytes

Double : 8 bytes

Double has two times the precision of float.

float area = 12.34;
double volume = 1345.678543;
double distance = 45E15;

Char : 1 byte
Enclosed within single quotes.
char ch = 'A';

wchar_t : 2 bytes
Wide character similar to char but size is 2 bytes.
Used for supporting universal character set.

Bool : 1 byte
true or false
They are generally used in condition statements and loops.
bool cold = true;

Void : 0
Nothing or no value
We will use it for functions and pointers.

Type modifiers:
signed
unsigned
short
long

These modifiers can be applied in conjunction with
int
double
char

4. Type Conversion

2 types:
I) Implicit Conversion

Int to Double	Double to Int

int x = 10;
double y;
y = x;
cout << y << endl;
 	
Output: 10.0
	double x = 12.57;
	int y;
	y = x;
	cout << y << endl;
	
	Output: 12

Data loss during conversion:
As we rise up the below ladder, there is no precision loss during datatype conversion.

long double
double
float
long
int
short
char

II) Explicit Conversion (Type casting)
This is done in 3 ways:

a) **C-style type casting (Cast notation)**

```
int  x = 27;
double y;
y = (double) x;
```

b) **Function notation (C++ style type casting) [Preferred]**

```
double  x = 27.67;
int y;
y = int(x);
```

c) **Type conversion operators.**

static_cast
const_cast
dynamic_cast
reinterpret_cast

5. Operators

There are 6 types:

1. Arithmetic Operators

Operator	Meaning
+	Addition
-	Subtraction
*	Multiplication
/	Division
%	Modulo Remainder

Division Operator : /

7/2 is 3
7.0/2 is 3.5
7/2.0 is 3.5
7.0/2.0 is 3.5
7/(float)2 is also 3.5
…

Increment & Decrement Operators:
++: Increases the value of the operand by 1
-- : Decreases the value of the operand by 1

a++: **Post increment**

```
a = 2;
b = a++;
```

(1. a will be assigned to b
2. Value of a will increase by 1)
a--

++a : **Pre-increment**
--a

2. Assignment Operators

Operator	Meaning
=	a = b; b is being assigned to a (Right to left associativity)
+=	a = a + 1
-=	a = a -1
*=	**a = a - 1**
/=	
%=	

3. Relational Operators

Operator	Meaning
==	Is Equal to
!=	Not Equal to
>	Greater than
<	**Less than**
>=	**Greater than or equals to**
<=	**Less than or equals to**

4. Logical Operators

&& : Logical AND (Binary Operator)
|| : Logical OR. (Binary Operator)
! : Logical NOT (Unary operator)

5. Bitwise Operators

& : Binary AND
| : Binary OR
^ : Binary XOR (Exclusive OR)
~ : Binary One's complement
<< : Shift left (Multiplication : 1 bit left shift leads to multiplication by 2)
>> : Shift right (Division : 1 bit right shift leads to division by 2)

6. Other Operators

Ternary Operators:
<expression>?<if True, this block gets executed>:<If False, this block is executed.>

6. Comments

Single line comment : // Comment
Multi line comment :
/*
Comment
*/

2. Flow Control

1. If - Else

A. If statement

```
if (condition_is_true){
        //body of if statement
}
```

B. If…else statement

```
if (condition_is_true){
        //body of if statement
}
```

```
int main(){
    int n = 501;

    if (n % 2 == 0){
        cout << n << " is an even number." << endl;
    }
    else{
        cout << n << " is an odd number." << endl;
    }


}
```

```
else{
        //body of the else statement
}
```

C. If…else if…else statement

```
if (condition1_is_true){
        //code block 1
}
else if(condition2_is_true){
        // code black 2
}
else{
        //code block 3
}
```

```
int units_consumed = 60, bill_amt;

if (units_consumed < 50)
    bill_amt = units_consumed * 2;
else if (units_consumed < 100)
    bill_amt = 100 + (units_consumed - 50) * 4;
else if (units_consumed < 250)
    bill_amt = 300 + (units_consumed - 100) * 7;
else
    bill_amt = 1350 + (units_consumed - 250) * 12;
```

D. **Nested If … else**

```
if (condition_is_true){
        if (sub_condition1_is _true){
                // Code Block 1
        }
        else{
                // Code Block 2
        }
}
else {
        if (sub_condition2_is _true){
                // Code Block 3
        }
        else{
                // Code Block 4
        }
}
```

2. For Loop

```
for(initialisation; condition; increment){
        // Code block for for loop
}
```

```
for(int i=0; i < 5; i++){
    cout << i << endl;
}
```

3. While Loop

```
//Initialisation block;
while(condition){
   // body of the loop
  // increment block
}
```

4. Do….While loop

```
do{
      // body of the loop
}while(condition);     // DON'T FORGET THE TERMINATING SEMICOLON
```

5. Break statement

When the condition for break is true, the control (of the code) comes out of the immediate loop.

6. Continue statement

When the condition for continue is true, the control (of the code) skips the successive lines and goes for the next iteration.

7. Switch statement

```
switch(expression){
        case constant1:
                // block1
                break;
        case constant2:
                // block2
                break;
        case constant3:
                // block3
                break;
        …
        default:
                // default_block;
}
```

8. Goto Statement

goto label;
….
….
….
label:
Statement;

3. Functions

A function is a reusable block of code that performs a specific task.

2 types of functions:

a. Standard Library Functions (cout, can, etc.)

b. User defined functions.

The **syntax** to declare a function:

returnType functionName (parameter1, parameter2,….){

// function body

}

Function Parameters:

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

void printNum(int num){
    cout << "printing the number " << num << endl;
}

int main(){

    printNum(10);
    printNum(20);
    printNum(40);

    return 0;
}
```

Function Prototype & Return Statement:

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

int add(int, int);
void printNum(int);


int main(){
    int a = 1, b = 2, sum;

    sum = add(1, 2);
    printNum(sum);

    return 0;
}

int add(int a, int b){
    return (a + b);
}

void printNum(int num){
    cout << "printing the number " << num << endl;
}
```

Benefits:
>> Reusable code. Declare one and use multiple times.
>> Makes the program easier as each small task is divided into a function.
>> Increases readability.

C++ Standard Library Functions:
Built-in functions in C++ programming.
Common library functions : sqrt(), abs(), isdigit(), etc.
In order to use library functions, we usually need to include the corresponding header file in which these functions are defined.
e.g. To use sqrt() and abs() we need to include the header file cmath.

Function Types:
a. No argument & no return value
b. No argument but return value
c. With argument but no return value
d. With argument and return value

Function Overloading:
Functions having the same name, different arguments (in terms of number and datatype)
Only changing the

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

void display(int var1, double var2){
    cout << "Integer : " << var1 ;
    cout << " & Double : " << var2 << endl;
}

void display(double var){
    cout << "Double : " << var << endl;
}

void display(int var){
    cout << "Integer : " << var << endl;
}

int main(){
    int a = 5;
    double b = 2.3;

    display(a);
    display(a, b);
    display(b);

    return 0;
}
```

return type does not qualify a function for overloading.

```
int test() { ... }

int test(int a) { ... }

double test(int a ){ ... }  # Error. You can keep 3 or 5

float test(double a) { ... }

int test(int a, double b) { ... }

int main(){
    test(10);
}
```

In C++ many standard Library functions are overloaded. E.g. sqrt() can accept int, float, double, etc.

Default Argument:

a. Once you provide a default value for a parameter, all subsequent parameters must also have default values.

void add(int a, int b=3, int c, int d) ; Invalid
void add(int a, int b=3, int c, int d=5) ; Invalid
void add(int a, int c, int b = 3, int d=5) ; Valid

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

void display(char a = '%', int b = 3);

int main(){
    int count = 5;

    cout << "No argument passed : " ;
    display();

    cout << "First argument passed : ";
    display('#');

    cout << "Both arguments passed : ";
    display('$', count);

    return 0;
}

void display(char c, int n){
    for (int i=1; i <= n ; i++){
        cout << c;
    }
    cout << endl;
}
```

b. You should also declare the default values in the function prototype / declaration.

Storage Class :
5 major types:
a. Local
b. Global
c. Static Local

d. Register
e. Thread Local

1. Simple Functions
2. Function Types
3. Function Overloading
4. Default Argument
5. Storage Class
6. Recursion
7. Return reference

4. Array & Strings

1. Arrays

An array is a variable that can **store multiple values** of the **same type.**

Syntax: datatype arrayName[arraySize];

double grades[5]; // grade is an array that can hold a max of 5 doubles.

grades

Index: 0	1	2	3	4
Value : 99	**95**	**89**	**97**	**98**
Address: 9796	9800	9804	9808	9812

Accessing elements:

arrayName[index]

A. Indexing starts with 0
B. The last element has an index of (n-1) where n is the length of the array.
C. Elements of an array have consecutive addresses. e.g. if address of x[1] is 9800 then x[2] will be stored at 9804.

Array Initialization: double grades[5] = {99, 95, 89, 97, 98};

Array Declaration & Initialization: int x[] = {9, 10, 95, 22, 101, 89, 97, 98};

Array Initialization with empty members:
double grades[10] = {99, 95, 89, 97, 98, 94}; // 6 numbers initialised, 4 numbers are empty

2. Multidimensional Arrays

Declaration of a 2D array : int x[4][3]; // This array can store 12 elements. There will be 4 rows and 3 columns

x

x[0][0] = 20	x[0][1] = 25	x[0][2] = 15
x[1][0]	x[1][1]	x[1][2]
x[2][0]	x[2][1]	x[2][2]
x[3][0]	x[3][1]	x[3][2]

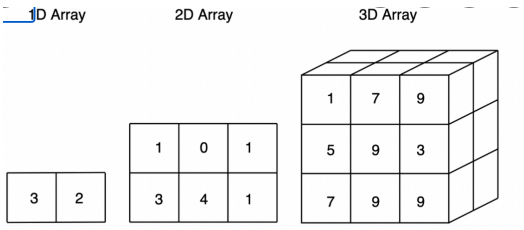
Declaration of 3D array :
int x[3][4][2];
Rubic Cube : int x[3][3][3];

Initialization of 2D array :
int x[2][3] = {10, 11, 2,7,5, 18}; // Not preferred
int x[2][3] = {{10, 11, 2}, {7, 5, 18}};

10	11	2
7	5	18

Initialization of 3D array :

```
int x[][][] = {
      {{1,7,9}, {5, 9, 3}, {7,9,9}},
      {{4,17,8}, {50, 9, 13},
{72,39,95}}
}
```



3. **Function & Array** :

Syntax for passing arrays as function parameters:

```
returnType functionName(datatype arrayName[arraySize]){
    //code
}

int total(int marks[5]){
    //code
}
```

4. String

5. File Input Output
   1. Open a file
   2. Read from, Write & Append to a file
   3. Text vs Binary file
   4. OS related commands
   5. Advanced File concepts

6. Structures
   1. Structure
   2. Structures & Functions
   3. C++ Pointers to Structure
   4. Enumeration

7. Object & Classes
8. Pointers
9. Inheritance
10. Application Development