Tech House

PROGRAMMING BASIC WITH C++

Ye Thu Aung

INTRODUCTION C++ PROGRAMMING ?

Section one

WHAT IS C++

- C++ is a middle-level programming language developed in 1979.
- C++ runs on a variety of platforms, such as Windows, Mac OS, and the various versions of UNIX.
- C++ gives programmers a high level of control over system resources and memory.
- The language was updated 4 major times in 2011, 2014, 2017, and 2020 to C++11, C++14, C++17, C++20.

WHY TO LEARN C++

- C++ is very close to hardware, so you get a chance to work at a low level which gives you lot of control in terms of memory management, better performance and finally a robust software development.

- C++ is an object-oriented programming language which gives a clear structure to programs and allows code to be reused, lowering development costs.

- C++ is the most widely used programming languages in application and system programming.

ENVIRONMENTAL SETUP

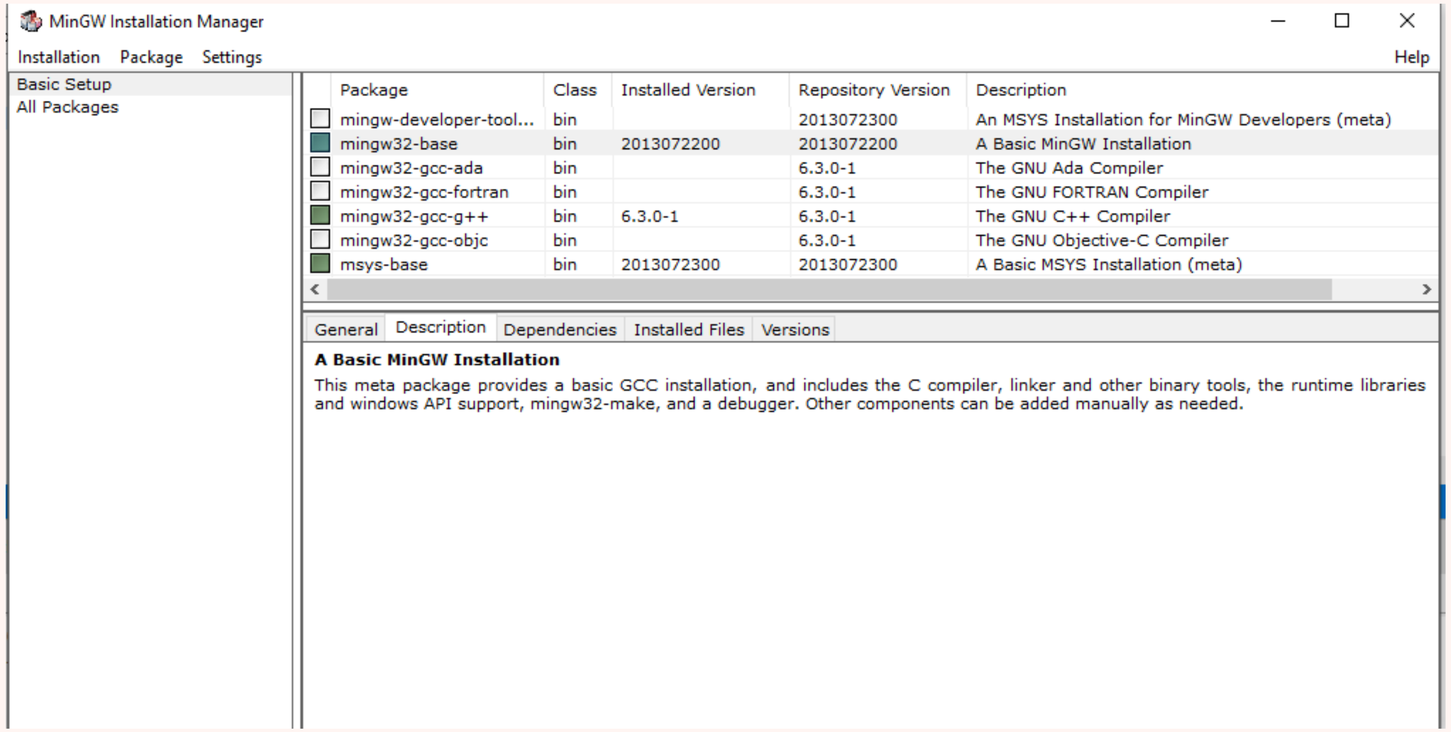
Section two

ENVIRONMENTAL SETUP

- Install C++ compiler
- Install CodeLite IDE
- Configure CodeLite IDE
- Hello World Project
- Command-line test
- Creating user template

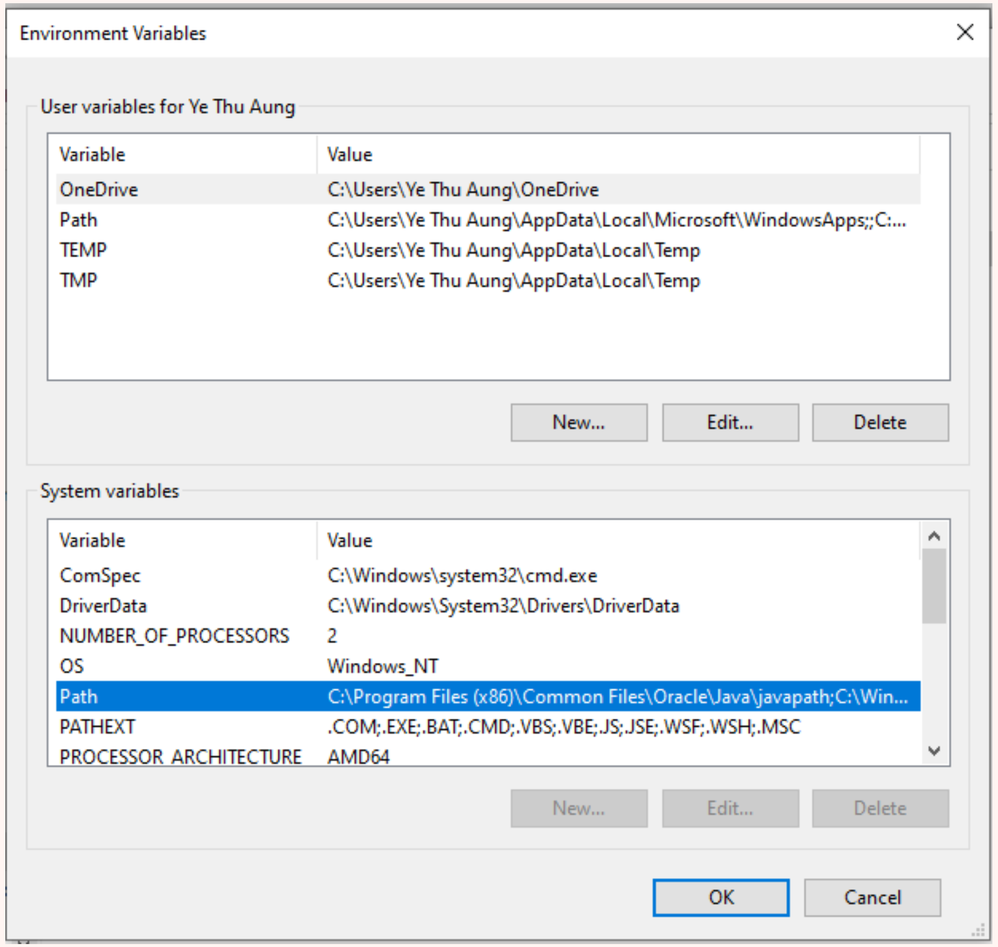
INSTALL C++ COMPLIER

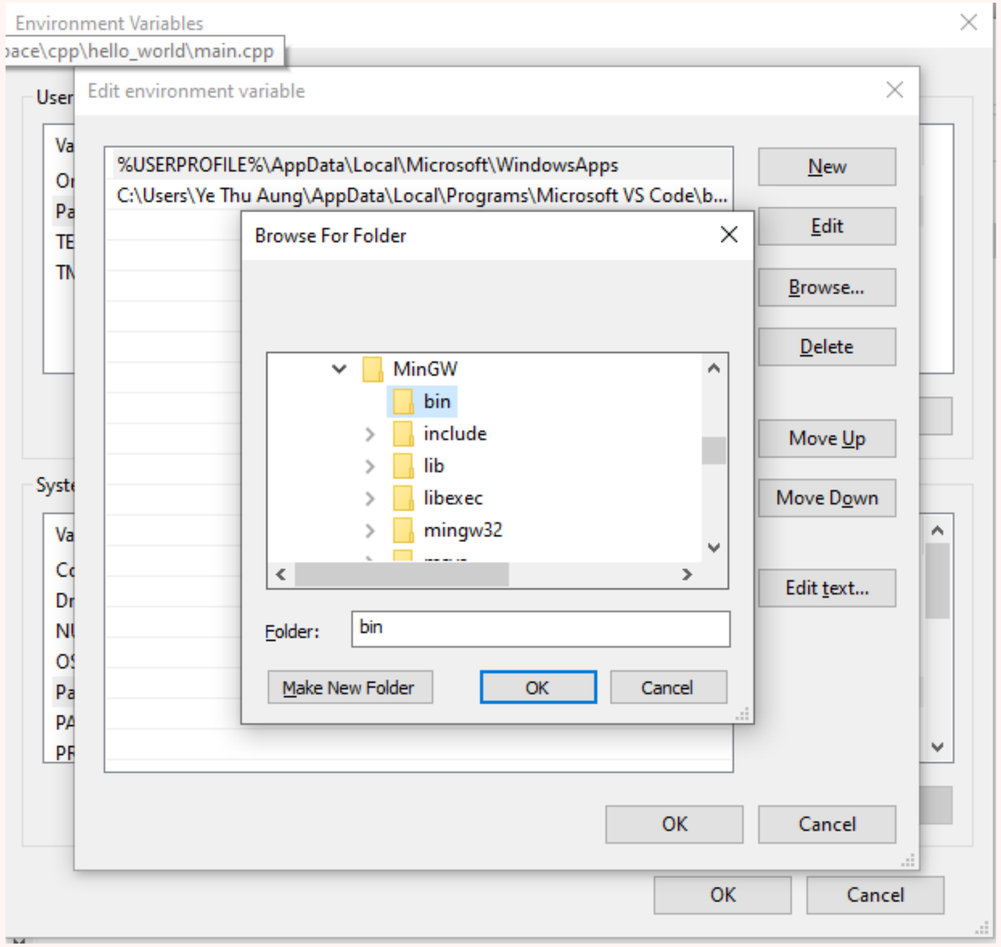
- We will use MinGW as c++ complier
- Download from https://sourceforge.net/projects/mingw/ from source forge
- Install MinGW and configure follow



INSTALL C++ COMPLIER

Edit env variable



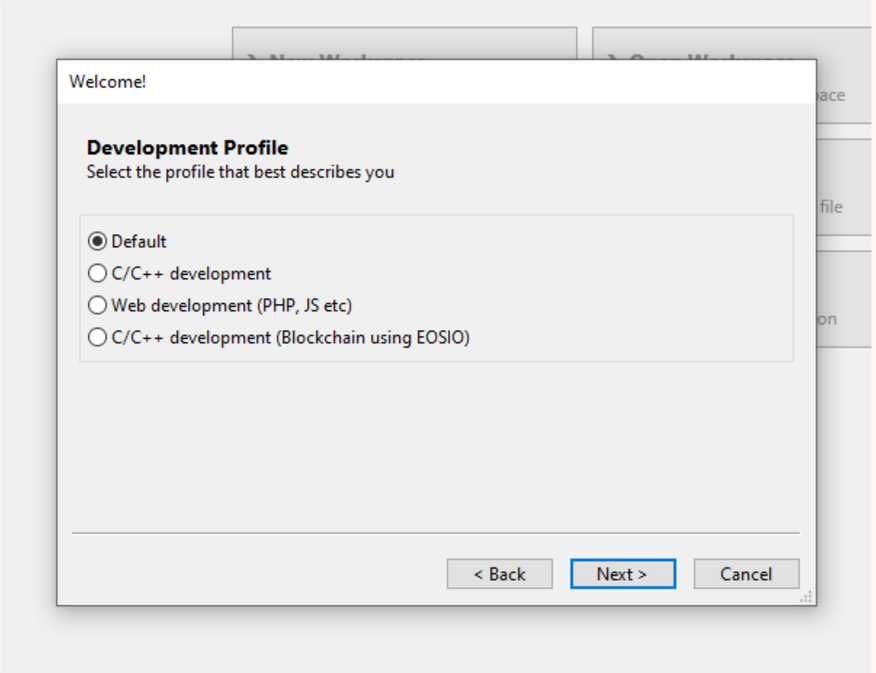


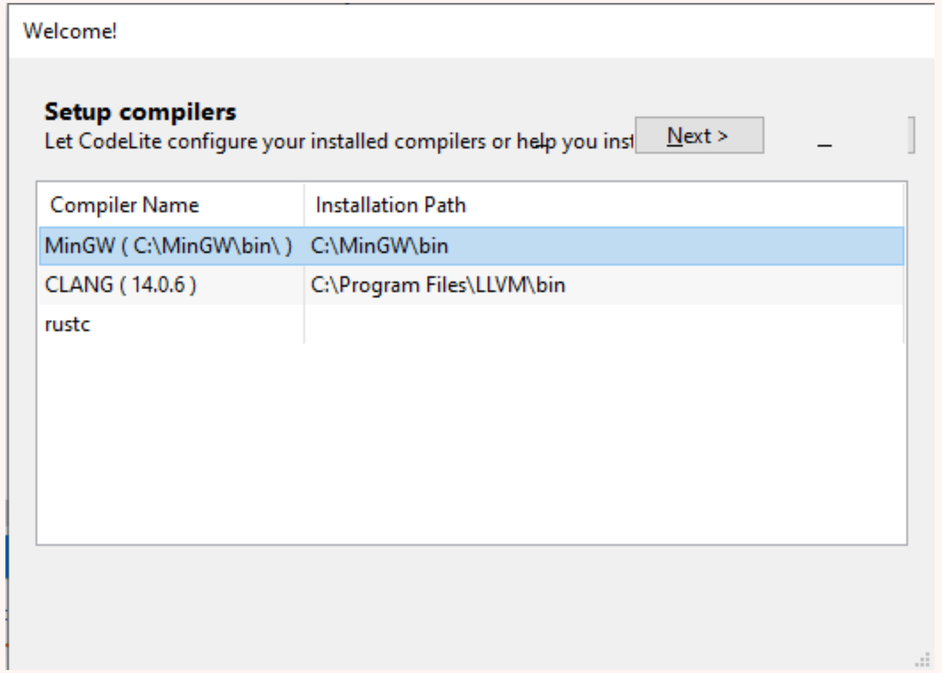
INSTALL CODELITE IDE

- We will use Code Lite as IDE (Integrated Development Environment)
- Download https://codelite.org/
- Install CodeLite

CONFIGURE CODELITE IDE

Choice development profile as C/C++ development

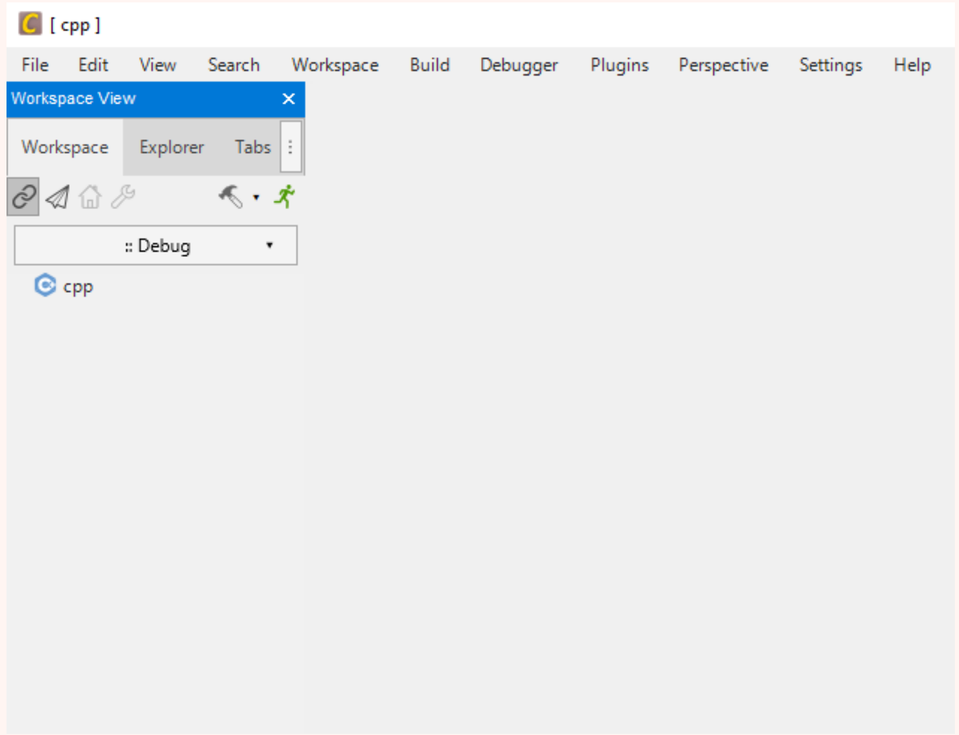


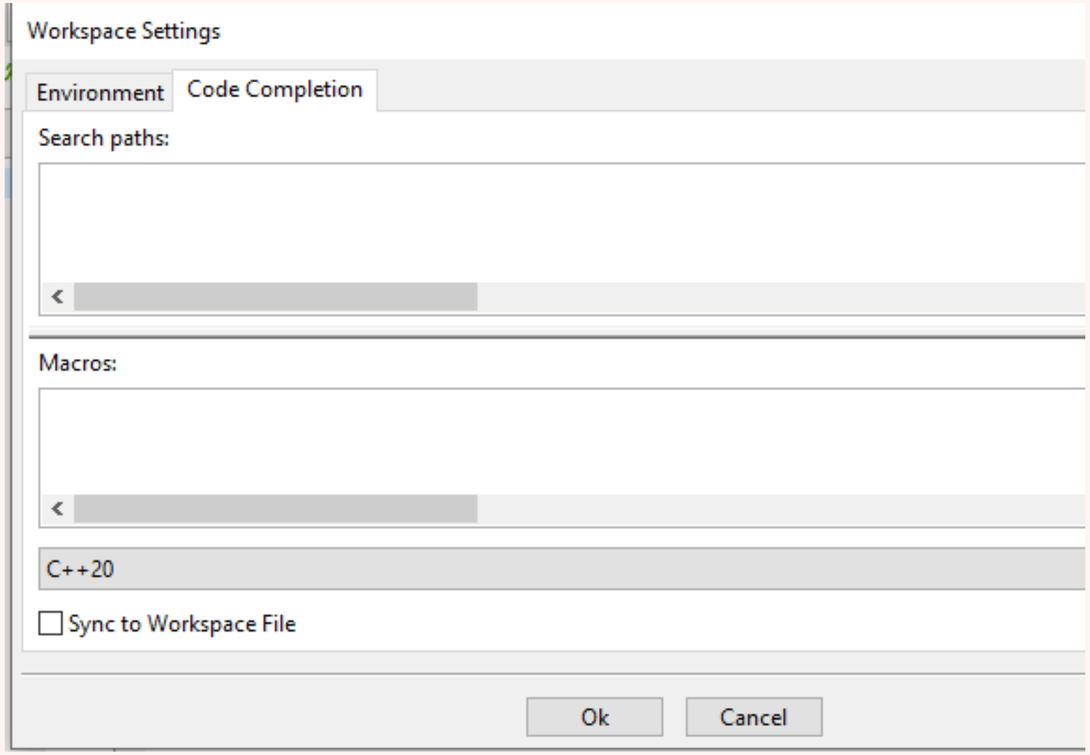


Choice compilers as MinGW

CONFIGURE CODELITE IDE

Create Workspace (new workspace) as cpp

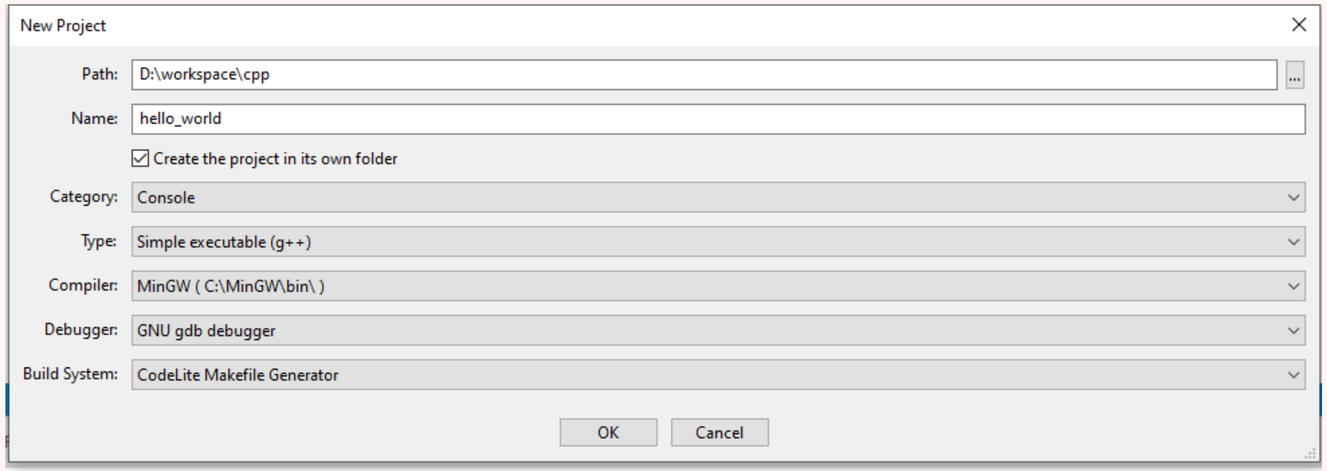


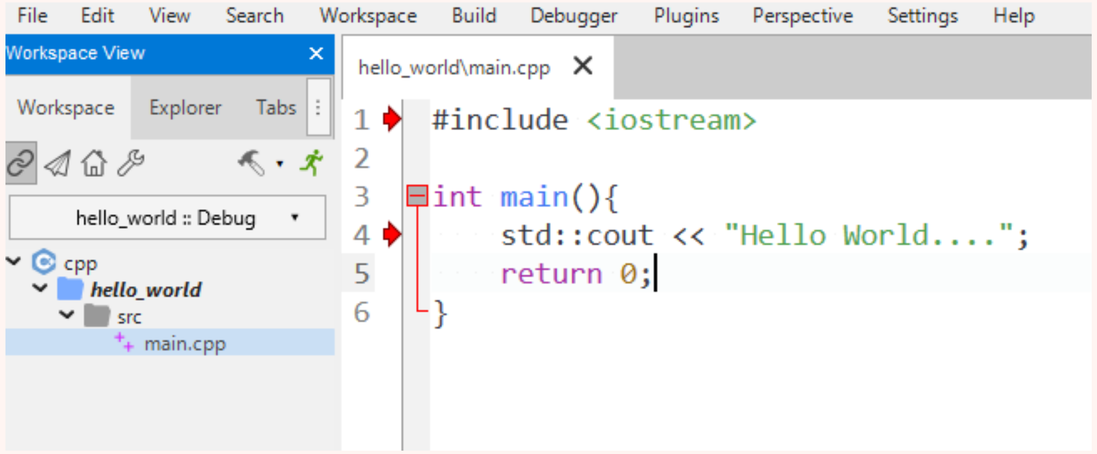


Right click on workspace, enter workspace setting , switch code completion and choice c++20

HELLO WORLD PROJECT

Create new project enter cpp workspace name as hello_world

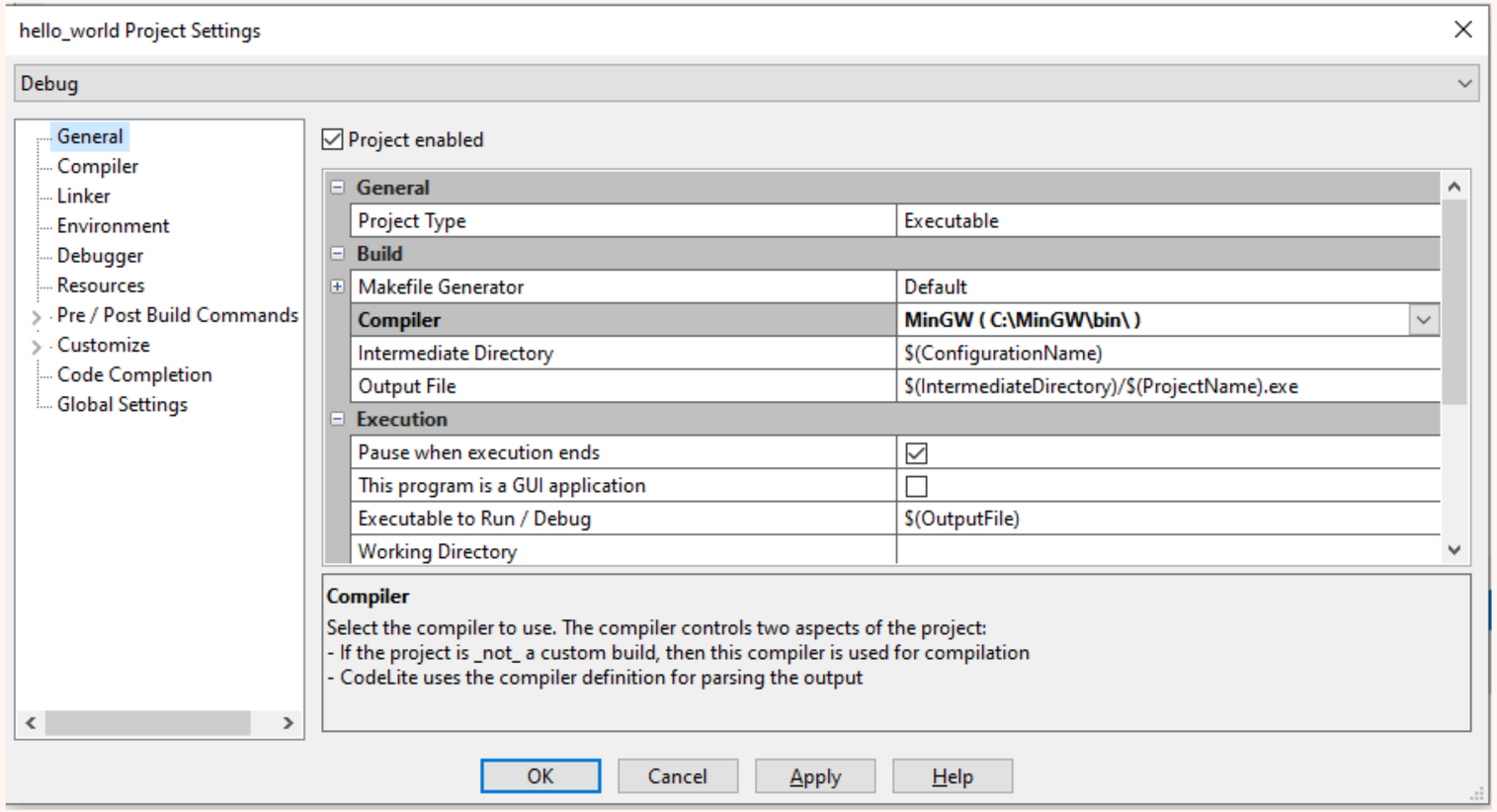




Write following code at hello_world > src > main.cpp

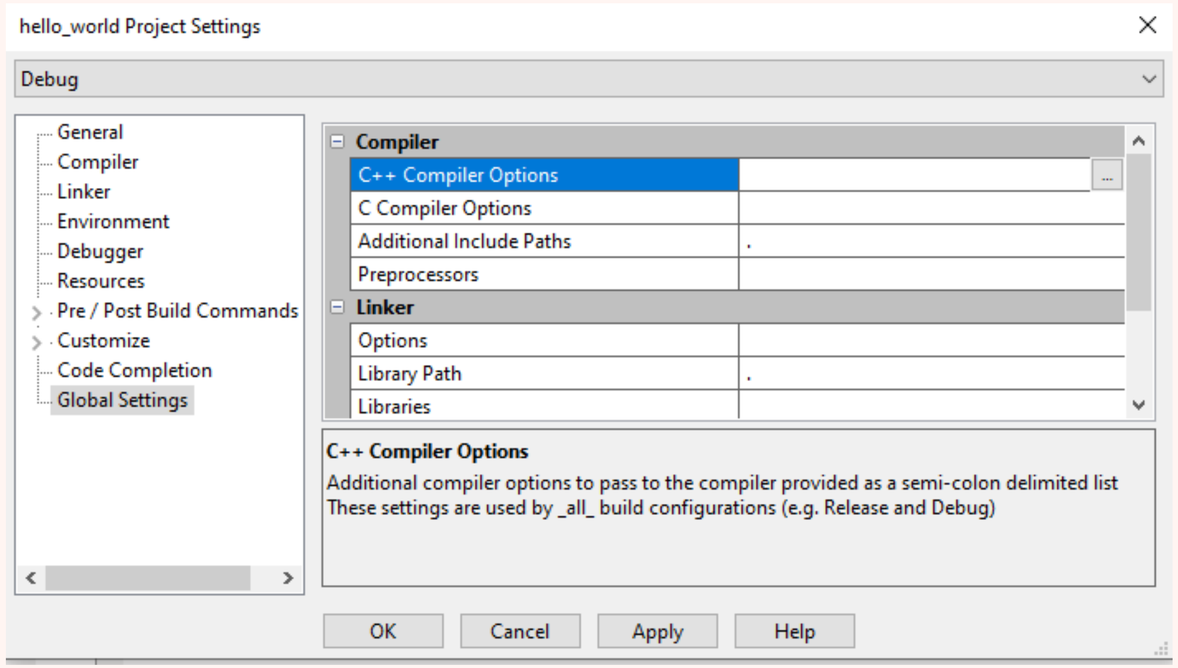
HELLO WORLD PROJECT

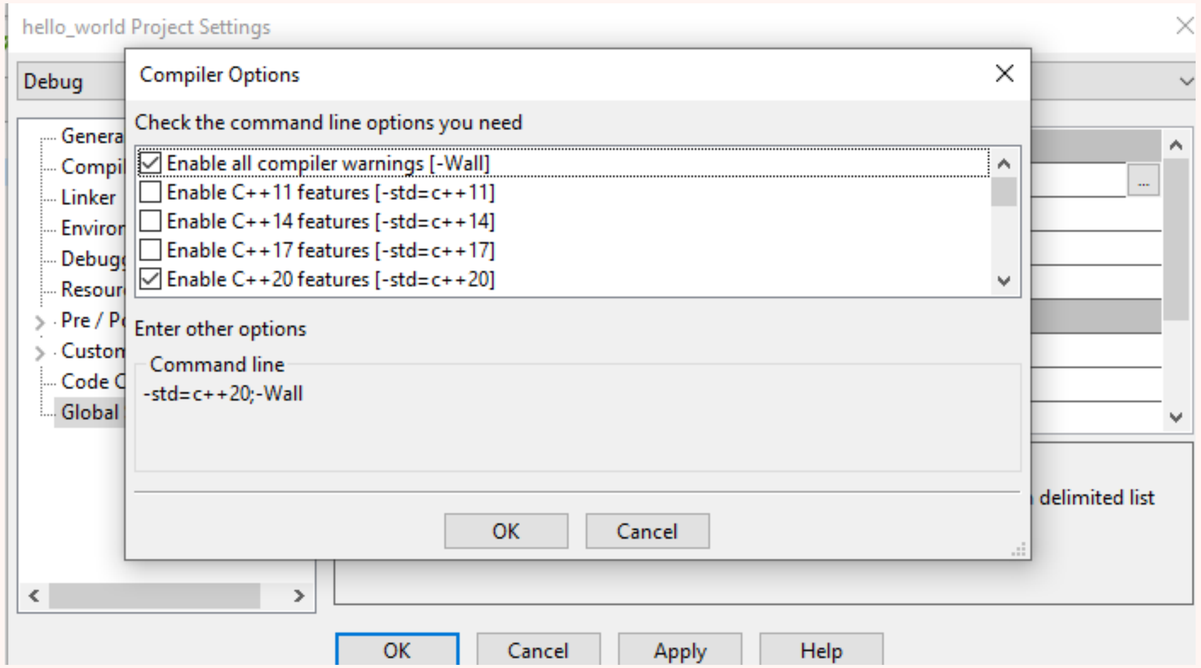
- **Right click on hello_world, go to setting and set complier as minGW**



HELLO WORLD PROJECT

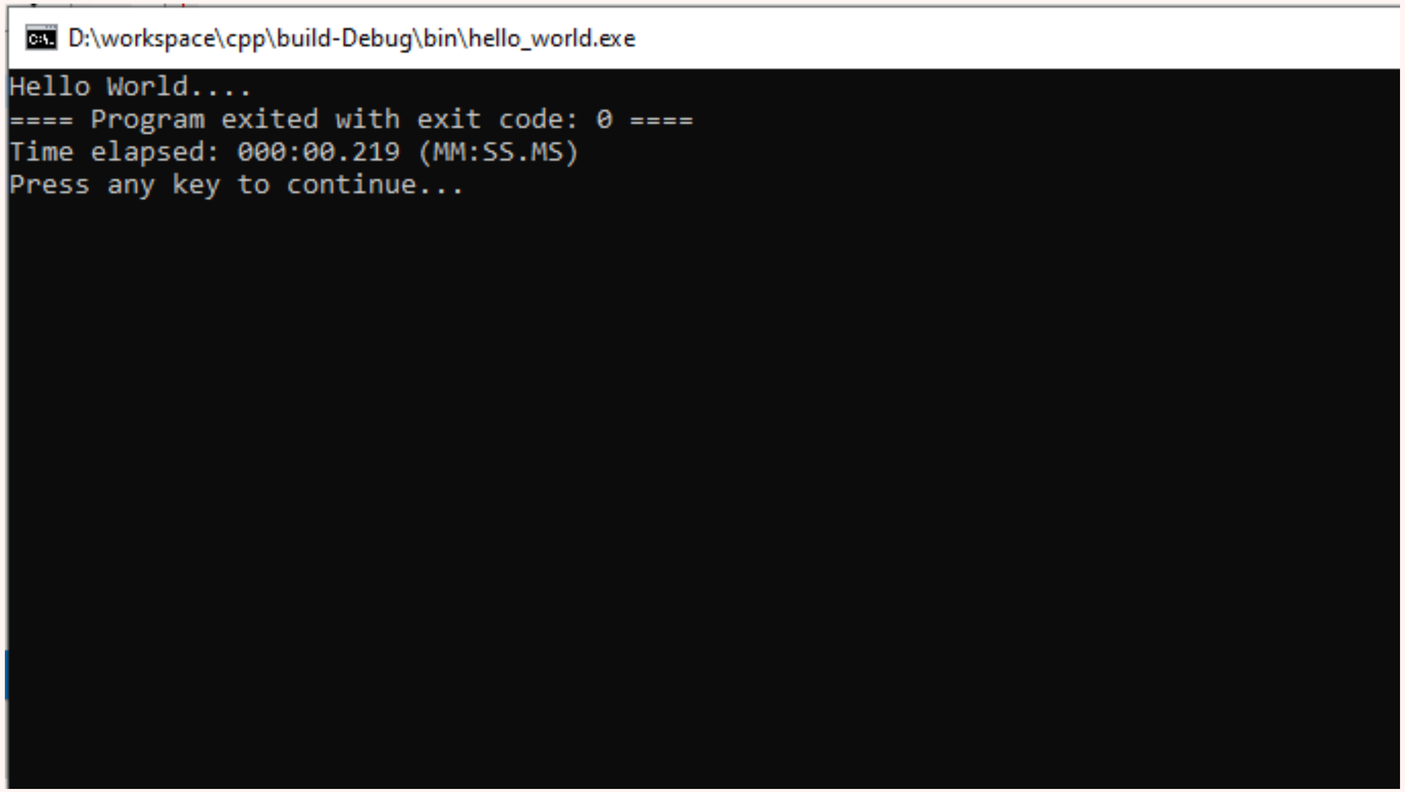
- **Also project setting, at Global settings change c++ compiler options and then apply and ok**





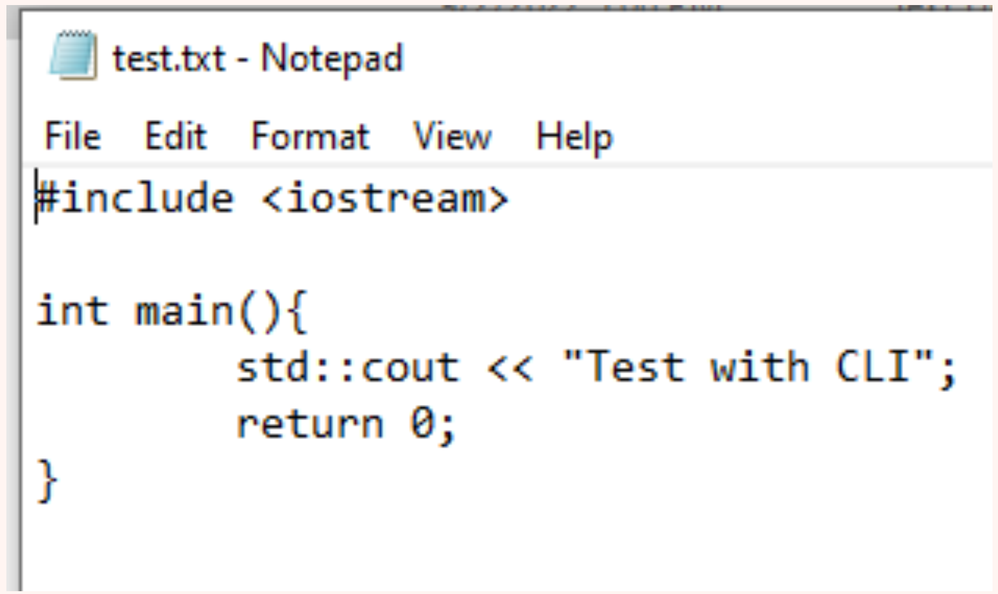
HELLO WORLD PROJECT

- Before run the project need to clean project and build project
- After clean and build, project can run and result following at terminal or command prompt



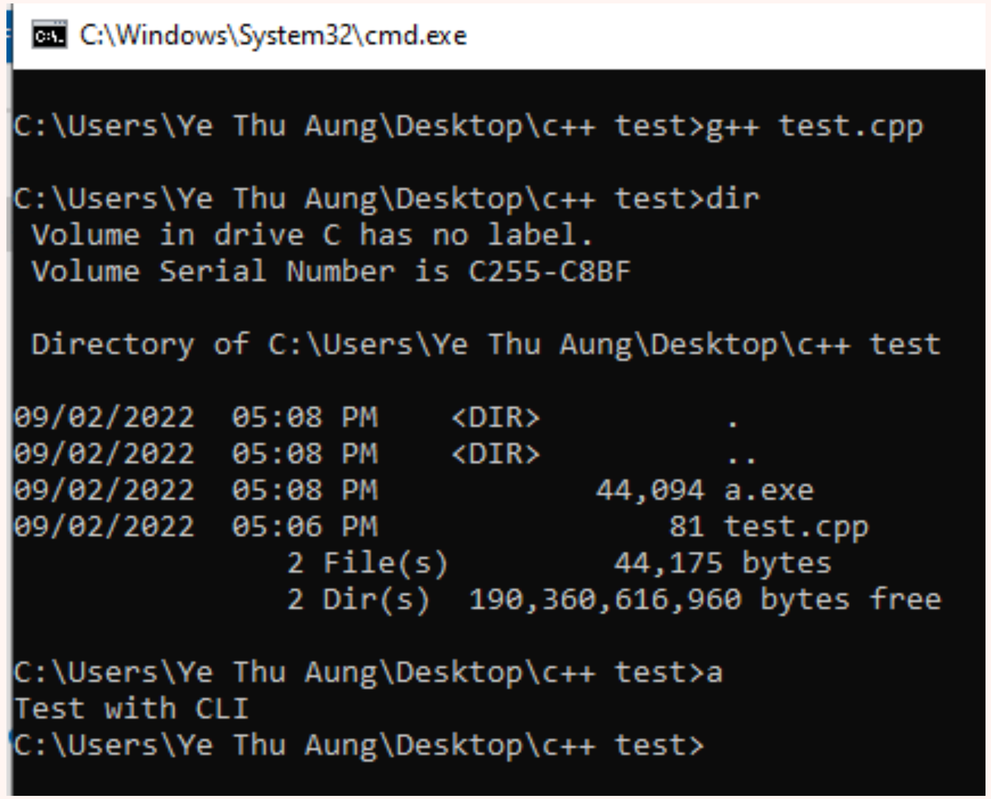
COMMAND-LINE TEST

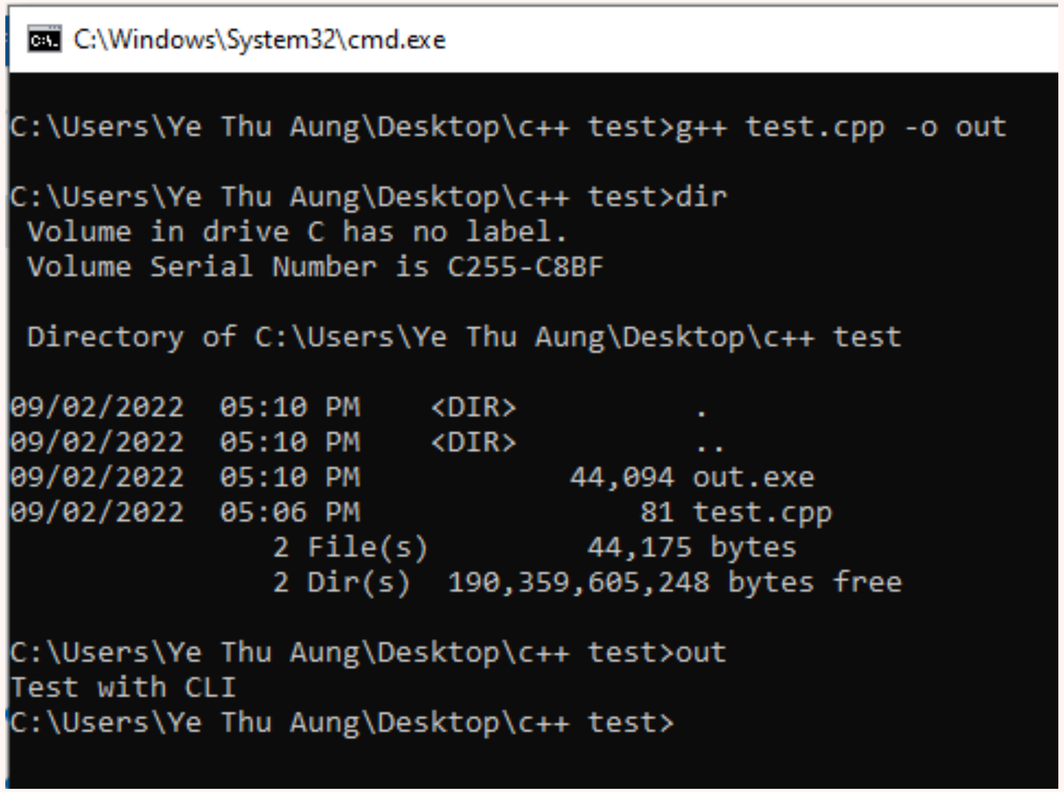
- If complier installed, can run c++ project without IDE.
- First create new folder at desktop, create new txt file, open in notepad, write following code and save as test.cpp



COMMAND-LINE TEST

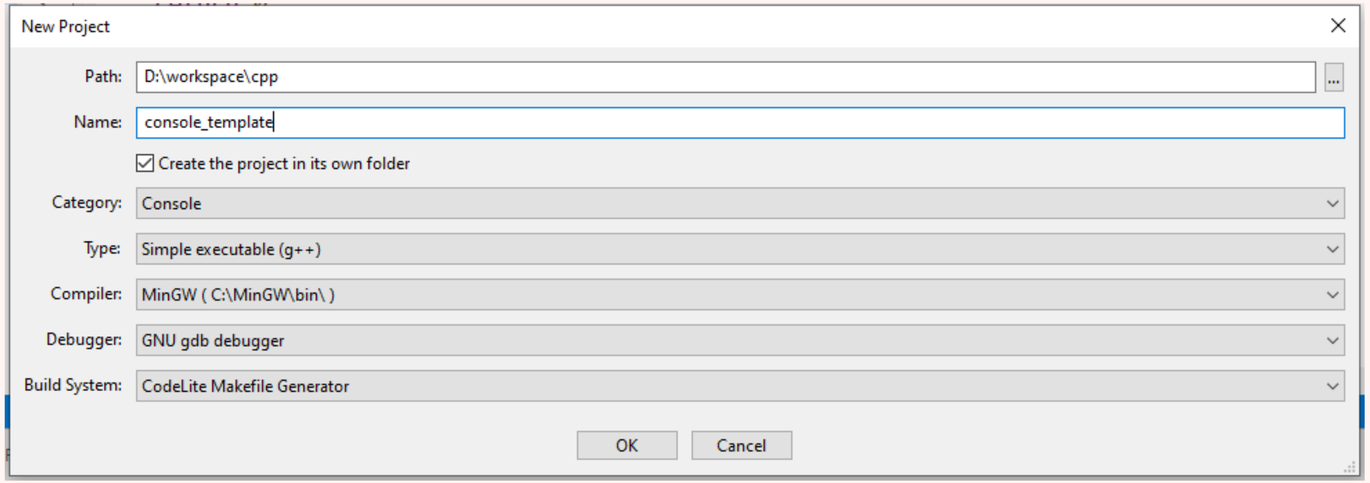
- Open cmd or terminal file created folder, run g++ test.cpp and g++ test.cpp -o out

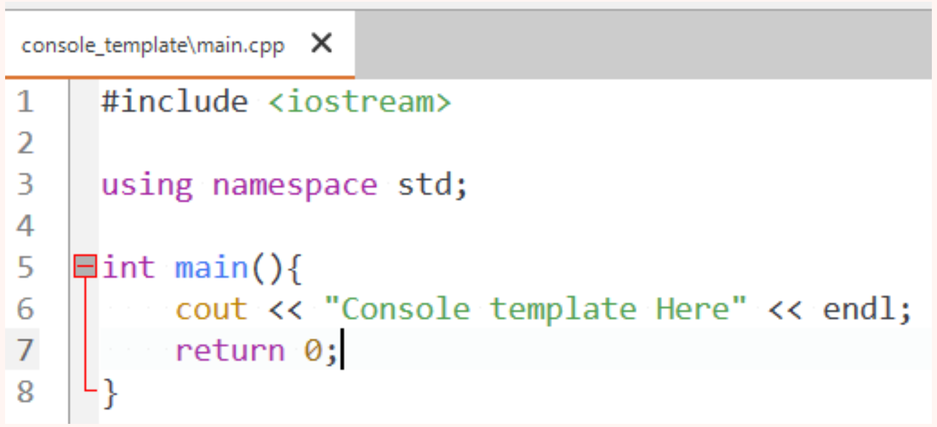




CREATING USER TEMPLATE

- **Create new project enter cpp workspace name as console_template**

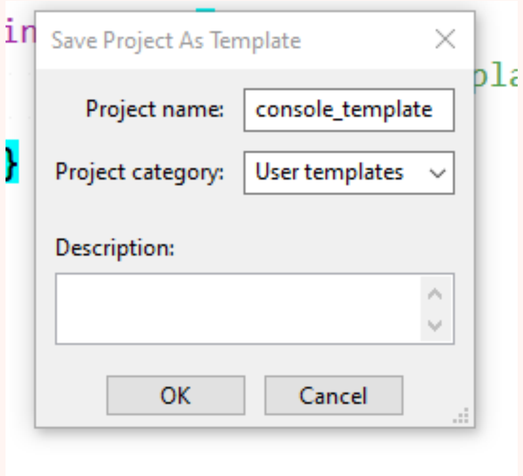


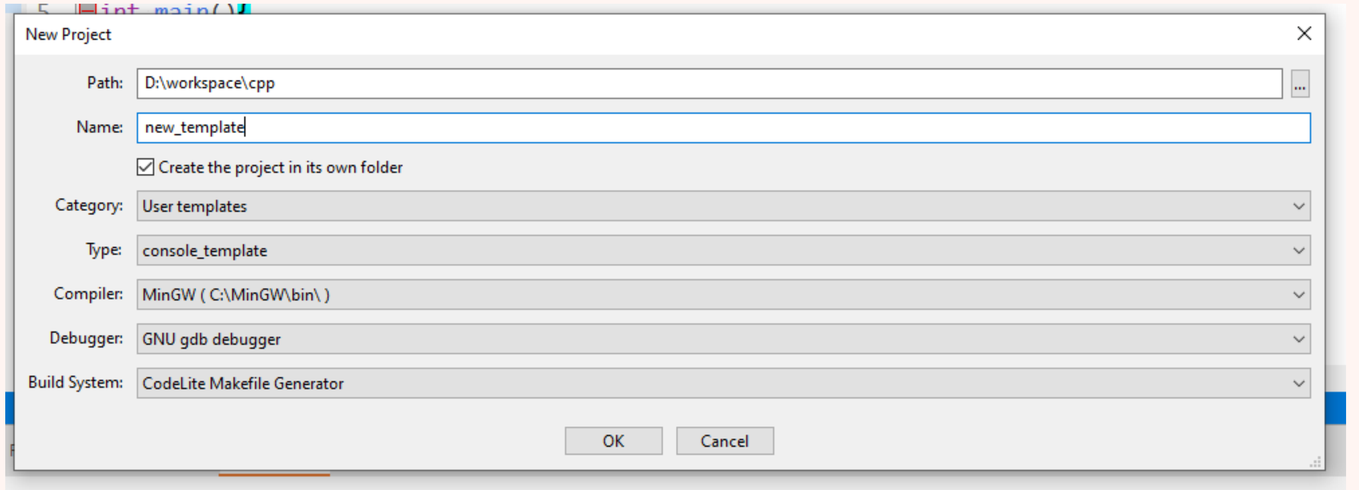


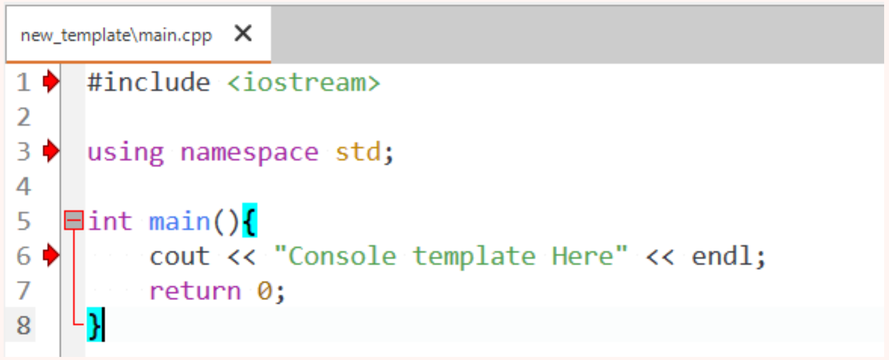
- **Write following code at console_template > src > main.cpp**

CREATING USER TEMPLATE

- Click on project folder, save as template and show following and OK







- Create new project with user template

- And then, created template code appear at main.cpp

VARIABLE AND CONSTANTS

Section three

VARIABLE AND CONSTANT

What is variable?

- Variables are containers for storing data values.
- Each variable in C++ has a specific type, which determines the size and layout of the variable's memory; the range of values that can be stored within that memory; and the set of operations that can be applied to the variable.
- The name of a variable can be composed of letters, digits, and the underscore character.
- C++ variable is case-sensitive

VARIABLE AND CONSTANT

Type of variable?

Type	Description
bool	Stores either value true or false.
char	Typically a single character (one byte). This is an integer type.
int	The most natural size of integer for the machine.
float	A single-precision floating point value.
double	A double-precision floating point value.
void	Represents the absence of type.
wchar_t	A wide character type

VARIABLE AND CONSTANT

Data Type

Type	Typical Bit	Typical Range
char	1 bytes	-127 to 127 or 0 to 255
unsigned char	1 bytes	0 to 255
signed char	1 bytes	-127 to 127
int	4 bytes	-2147483648 to 2147483647
unsigned int	4 bytes	0 to 4294967295
signed int	4 bytes	-2147483648 to 2147483647
short int	2 bytes	-32768 to 32767
unsigned short int	2 bytes	0 to 65,535
signed short int	2 bytes	-32768 to 32767
long int	8 bytes	-2,147,483,648 to 2,147,483,647
signed long int	8 bytes	-2,147,483,648 to 2,147,483,647
unsigned long int	8 bytes	0 to 4,294,967,295
long long int	8 bytes	-(2^63) to (2^63)-1
unsigned long long int	8 bytes	0 to 18,446,744,073,709,551,615
float	4 bytes	
double	8 bytes	
long double	12 bytes	
wchar_t	2 or 4 bytes	1 wide character

VARIABLE AND CONSTANT

Create workspace(section_three) and add new project(variableCheck) for test following code

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

int main(){
   cout << "Size of char : " << sizeof(char) <<endl;
   cout << "Size of unsigned char : " << sizeof(unsigned char) <<endl;
   cout << "Size of signed char : " << sizeof(signed char) <<endl;
   cout << "Size of int : " << sizeof(int) <<endl;
   cout << "Size of unsigned int : " << sizeof(unsigned int) <<endl;
   cout << "Size of signed int : " << sizeof(signed int) <<endl;
   cout << "Size of short int : " << sizeof(short int) <<endl;
   cout << "Size of unsigned short int : " << sizeof(unsigned short int) <<endl;
   cout << "Size of signed short int : " << sizeof(signed short int) <<endl;
   cout << "Size of long int : " << sizeof(long int) <<endl;
   cout << "Size of signed long int : " << sizeof(signed long int) <<endl;
   cout << "Size of unsigned long int : " << sizeof(unsigned long int) <<endl;
   cout << "Size of long long int : " << sizeof(long long int) <<endl;
   cout << "Size of unsigned long long int : " << sizeof(unsigned long long int) <<endl;
   cout << "Size of float : " << sizeof(float) <<endl;
   cout << "Size of double : " << sizeof(double) <<endl;
   cout << "Size of long double : " << sizeof(long double) <<endl;
   cout << "Size of wchar_t : " << sizeof(wchar_t) <<endl;
   return 0;
}
```

```
Size of char : 1
Size of unsigned char : 1
Size of signed char : 1
Size of int : 4
Size of unsigned int : 4
Size of signed int : 4
Size of short int : 2
Size of unsigned short int : 2
Size of signed short int : 2
Size of long int : 8
Size of signed long int : 8
Size of unsigned long int : 8
Size of long long int : 8
Size of unsigned long long int : 8
Size of float : 4
Size of double : 8
Size of long double : 16
Size of wchar_t : 4
```

VARIABLE AND CONSTANT

Variable Declaration

Syntax

Type variableName = value

Create project name as variableDeclaration and write following code

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

int main(){
   // Variable definition:
   int a, b;
   int c;
   float f;
   // actual initialization
   a = 10;
   b = 20;
   c = a + b;

    cout <<  c << endl;
   f = 70.0/3.0;
   cout << f << endl ;
   return 0;
}
```

VARIABLE AND CONSTANT

Variable Literals

Escape sequence	Meaning
\\	\ character
\’	‘ character
\”	“ character
\?	? character
\a	Alert or bell
\b	Backspace
\f	Form feed
\n	Newline
\r	Carriage return
\t	Horizontal tab
\v	Vertical tab

VARIABLE AND CONSTANT

Variable Literals

Create project name as variableLiterals and write following code

variableLiteral/main.cpp

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

int main(){
    cout << "Slash literals \\" << endl;
    cout << "Single quote literals \'" << endl;
    cout << "Double quote literals \"" << endl;
    cout << "Question marks literals \?" << endl;
    cout << "Alert or bell literals (os songs appear) \a" << endl;
    cout << "Backspace literals \b" << endl;
    cout << "Form feed literals \f" << endl;
    cout << "New Line literals \n" ;
    cout << "Horizontal \t tab " << endl ;
    cout << "Vertical \v tab " << endl ;
    return 0;
}
```

Slash literals \
Single quote literals '
Double quote literals "
Question marks literals ?
Alert or bell literals (os songs appear)
Backspace literals
Form feed literals

New Line literals
Horizontal	tab
Vertical
		tab

VARIABLE AND CONSTANT

Identifiers and Multiple declaration

All C++ variables must be identified with unique names.These unique names are called identifiers.

1. Names can contain letters, digits and underscores
2. Names must begin with a letter or an underscore (_)
3. Names are case sensitive (myVar and myvar are different variables)
4. Names cannot contain whitespaces or special characters like !, #, %, etc.
5. Reserved words (like C++ keywords, such as int) cannot be used as names

Multiple variable declaration

```
int fstNum, secNum;

int a = 3, b = 5;
```

VARIABLE AND CONSTANT

Define and Constant

#define is pre-processor directive while const is a keyword
#define is not scope controlled whereas const is scope controlled

Create project name as defineAndConst and write following code

defineAndConst/main.cpp

```cpp
#include <iostream>
using namespace std;
#define abc 30 //macro definition
const int def = 13;
int main()
{
    cout<<"Define value of abc: "<<abc<<endl;
    #undef abc //undefining abc
    #define abc 300 // redefine abc
    cout<<"Define value of chnaged abc: "<<abc<<endl;

    cout << "Const value of def: " << def << endl;
//    def = 15;
//    cout << "Const value changed def: "<< def << endl;
    return 0;
}
```

VARIABLE AND CONSTANT

User Input

cin is a predefined variable that reads data from the keyboard with the extraction operator (>>)

Create project name as userInputTest and write following code

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

int main(){
    int fstNum,secNum;
    int sum;
    cout << "Enter First Number : ";
    cin >> fstNum;
    cout << "Enter Second Number : ";
    cin >> secNum;
    sum = fstNum + secNum;
    cout << "The Sum of First and Second Number is " << sum << endl;
    return 0;
}
```

STATEMENTS AND OPERATORS

Section four

STATEMENTS AND OPERATORS

What is operator

An operator is a symbol that tells the compiler to perform specific mathematical or logical manipulations.

Type operator

- Arithmetic Operators
- Relational Operators
- Logical Operators
- Bitwise Operators
- Assignment Operators
- Misc Operators

STATEMENTS AND OPERATORS

Arithmetic Operators

Assume variable A holds 10 and variable B holds 20

Operator	Description	Example
+	Adds two operands	A + B will give 30
-	Subtracts second operand from the first	A - B will give -10
*	Multiplies both operands	A * B will give 200
/	Divides numerator by de-numerator	B / A will give 2
%	Modulus Operator and remainder of after an integer division	B % A will give 0
++	Increment operator, increases integer value by one	A++ will give 11
- -	Decrement operator, decreases integer value by one	A-- will give 9

STATEMENTS AND OPERATORS

Arithmetic Operators

Create project name as arithmeticOperator and write following code

```cpp
#include <iostream>

using namespace std;

int main(){
    int a = 20;
    int b = 10;
    int c;
    c = a + b;
    cout << "The value of a + b = " << c << endl;
    c = a - b;
    cout << "The value of a - b = " << c << endl;
    c = a * b;
    cout << "The value of  a * b = " << c << endl;
    c = a / b;
    cout << "The value of a / b = " << c << endl;
    c = a % b;
    cout << "The value of a % b = " << c << endl;
    //increment, decrement / pre and post
    cout << "Increment Postfix of a = " << a++ << endl;
    cout << "Decrement Postfix of a = " << a-- << endl;
    cout << "Increment Prefix of b = " << ++b << endl;
    cout << "Decrement Prefix of b = " << --b << endl;
    return 0;
}
```

STATEMENTS AND OPERATORS

Relational Operators (Comparison operators)

Assume variable A holds 10 and variable B holds 20

Operator	Description	Example
==	Checks if the values of two operands are equal or not, if equal then condition becomes true.	(A == B) is not true.
!=	Checks if the values of two operands are equal or not, if values are not equal then condition becomes true.	(A != B) is true.
>	Checks if the value of left operand is greater than the value of right operand, if yes then condition becomes true.	(A > B) is not true.
<	Checks if the value of left operand is less than the value of right operand, if yes then condition becomes true.	(A < B) is true.
>=	Checks if the value of left operand is greater than or equal to the value of right operand, if yes then condition becomes true.	(A >= B) is not true.
<=	Checks if the value of left operand is less than or equal to the value of right operand, if yes then condition becomes true.	(A <= B) is true.

STATEMENTS AND OPERATORS

Relational Operators

Create project name as relationalOperator and write following code

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

int main(){
    int a = 20;
    int b = 10;
    if( a == b ){
        cout << "a is equal to b" << endl;
    }else{
        cout << "a is not equal b" << endl;
    }
    if( a > b ){
        cout << "a is greater than b" << endl;
    }else{
        cout << "a is not greater than b" << endl;
    }
    if( a < b ){
        cout << "a is less than b" << endl;
    }else{
        cout << "a is not less than b" << endl;
    }
    if( a <= b){
        cout << "a is either less than or equal to b" << endl;
    }
    if( a >= b){
        cout << "a is either greater than or equal to b"<< endl;
    }
    return 0;
}
```

STATEMENTS AND OPERATORS

Logical Operators

Assume variable A holds 1 and variable B holds 0

Operator	Description	Example
&&	Called Logical AND operator. If both the operands are non-zero, then condition becomes true.	(A && B) is false.
||	Called Logical OR Operator. If any of the two operands is non-zero, then condition becomes true.	(A || B) is true.
!	Called Logical NOT Operator. Use to reverses the logical state of its operand. If a condition is true, then Logical NOT operator will make false.	!(A && B) is true.

STATEMENTS AND OPERATORS

Logical Operators

Create project name as logicalOperator and write following code

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

int main(){
    int a = 0;
    int b = 1;
    if(a && b){
        cout << "a && b is true" << endl;
    }else{
        cout << "a && b is false" << endl;
    }
    if(a || b){
        cout << "a || b is true" << endl;
    }else{
        cout << "a || b is false" << endl;
    }
    if(!(a && b)){
        cout << "!(a && b) is true"<< endl;
    }
    return 0;
}
```

STATEMENTS AND OPERATORS

Bitwise Operators

Assume variable A(0011 1100) holds 60(0000 1101) and variable B holds 13

Operator	Description	Example
&	Binary AND Operator copies a bit to the result if it exists in both operands.	(A & B) will give 12 which is 0000 1100
|	Binary OR Operator copies a bit if it exists in either operand.	(A | B) will give 61 which is 0011 1101
^	Binary XOR Operator copies the bit if it is set in one operand but not both.	(A ^ B) will give 49 which is 0011 0001
~	Binary Ones Complement Operator is unary and has the effect of 'flipping' bits.	(~A) will give -61 which is 1100 0011 in 2's complement form due to a signed binary number.
<<	Binary Left Shift Operator. The left operands value is moved left by the number of bits specified by the right operand.	A << 2 will give 240 which is 1111 0000
>>	Binary Right Shift Operator. The left operands value is moved right by the number of bits specified by the right operand.	A >> 2 will give 15 which is 0000 1111

STATEMENTS AND OPERATORS

Bitwise Operators

Create project name as bitwiseOperator and write following code

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

int main(){
    unsigned int a = 60;   // 60 = 0011 1100
    unsigned int b = 13;   // 13 = 0000 1101
    int c = 0;

    c = a & b;             // 12 = 0000 1100
    cout << "The value of a & b is " << c << endl;

    c = a | b;             // 61 = 0011 1101
    cout << "The value of a | b is " << c << endl;

    c = a ^ b;             // 49 = 0011 0001
    cout << "The value of a ^ b is " << c << endl;

    c = ~a;                // -61 = 1100 0011
    cout << "The value of ~a is " << c << endl;

    c = a << 2;            // 240 = 1111 0000
    cout << "The value of 2 left shift of a is " << c << endl;

    c = a >> 2;            // 15 = 0000 1111
    cout << "The value of 2 right shift if a is " << c << endl;
    return 0;
}
```

STATEMENTS AND OPERATORS

Assignment Operators

Operator	Description	Example
=	Simple assignment operator, Assigns values from right side operands to left side operand.	C = A + B will assign value of A + B into C
+=	Add AND assignment operator, It adds right operand to the left operand and assign the result to left operand.	C += A is equivalent to C = C + A
-=	Subtract AND assignment operator, It subtracts right operand from the left operand and assign the result to left operand.	C -= A is equivalent to C = C - A
*=	Multiply AND assignment operator, It multiplies right operand with the left operand and assign the result to left operand.	C *= A is equivalent to C = C * A
/=	Divide AND assignment operator, It divides left operand with the right operand and assign the result to left operand.	C /= A is equivalent to C = C / A
%=	Modulus AND assignment operator, It takes modulus using two operands and assign the result to left operand	C %= A is equivalent to C = C % A
<<=	Left shift AND assignment operator.	C <<= 2 is same as C = C << 2
>>=	Right shift AND assignment operator.	C >>= 2 is same as C = C >> 2
&=	Bitwise AND assignment operator.	C &= 2 is same as C = C & 2
^=	Bitwise exclusive OR and assignment operator.	C ^= 2 is same as C = C ^ 2
|=	Bitwise inclusive OR and assignment operator.	C |= 2 is same as C = C | 2

STATEMENTS AND OPERATORS

Assignment Operators

Create project name as assignmentOperator and write following code

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

int main(){
    int a = 21,c;
    c = a;
    cout << "Simple assignment of c = a is " << c << endl;
    c += a;
    cout << "Additional assignment of c += a is "<< c << endl;
    c -= a;
    cout << "Substract assignment of c -= a is " << c << endl;
    c *= a;
    cout << "Multiplication assignment of c *= a is "<< c << endl;
    c /= a;
    cout << "Division assignment of c /= a is "<< c << endl;
    c = 200;
    c %= a;
    cout << "Modulo assignment of c %= a is "<< c << endl;
    c <<= 2;
    cout << "Left shift assignment of c <<= 2 is" << c << endl;
    c >>= 2;
    cout << "Right shift assignment of c >>= 2 is " << c << endl;
    c &= 2;
    cout << "Logical AND assignment of c &= 2 is " << c << endl;
    c ^= 2;
    cout << "Logical XOR assignment of c ^= 2 is " << c << endl;
    c |= 2;
    cout << "Logical OR assignment of c |= 2 is " << c << endl;
    return 0;
}
```

STATEMENTS AND OPERATORS

Misc Operators

Operator	Description
sizeof	Sizeof operator returns the size of a variable. For example, sizeof(a), where ‘a’ is integer, and will return 4
Condition? X: Y	If Condition is true then it returns value of X otherwise returns value of Y
,	Comma operator causes a sequence of operations to be performed. The value of the entire comma expression is the value of the last expression of the comma-separated list.
. (dot) and -> (arrow)	Member operators are used to reference individual members of classes, structures, and unions.
Cast	Casting operators convert one data type to another. For example, int(2.2000) would return 2.
&	Pointer operator & returns the address of a variable. For example &a; will give actual address of the variable.
*	Pointer operator * is pointer to a variable. For example *var; will pointer to a variable var.

STATEMENTS AND OPERATORS

Misc Operators

Create project name as miscOperator and write following code

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

int main(){
    int a,b,c,d;
    float f = 2.200;
    int  var = 3000;
    int  *ptr;
    int  val;
    cout << "sizeof int value is " << sizeof(a) << endl;

    string res = (1 == 1)? "true": "false";
    cout << "Conditional Operater (1 == 1) ? \"true\":\"false\" is "<< res << endl;

    b = 5;
    c = (b++,b=+10,17+b);
    cout << "Comma Operator of ( b++, b=+4, 17+b ) is " << c << endl;

    d = (int) f;
    cout << "Cast Operator 2.200 (float) to int " << d << endl;

    ptr = &var;
    val = *ptr;
    cout << "Value of var :" << var << endl;
    cout << "Value of ptr :" << ptr << endl;
    cout << "Value of val :" << val << endl;
    return 0;
}
```

STATEMENTS AND OPERATORS

Operators Precedence

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

STATEMENTS AND OPERATORS

Operators Precedence

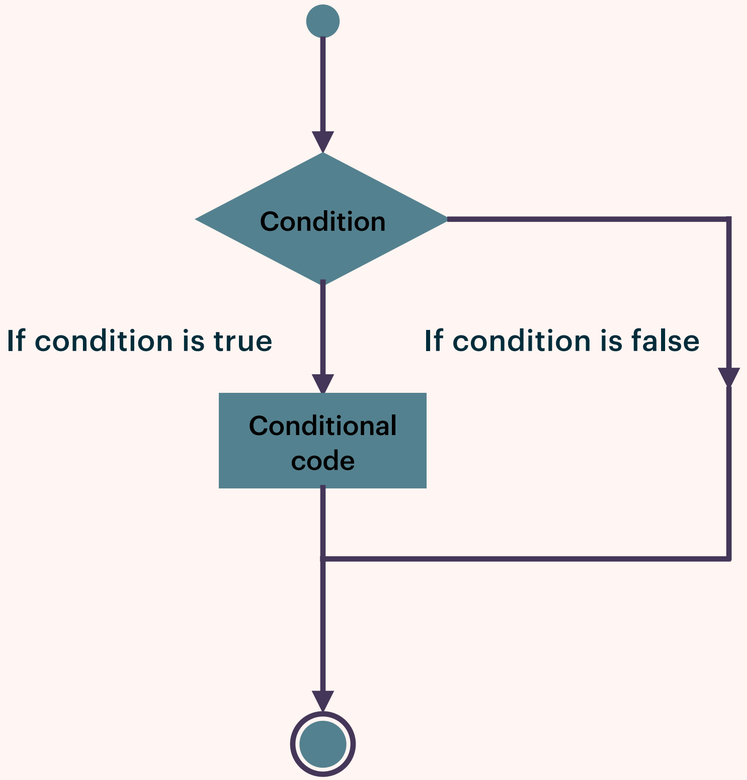
Create project name as operatorPrecedence and write following code

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

int main(){
    int a = 20;
    int b = 10;
    int c = 15;
    int d = 5;
    int e;

    e = (a + b) * c / d;       // ( 30 * 15 ) / 5
    cout << "Value of (a + b) * c / d is :" << e << endl ;

    e = ((a + b) * c) / d;     // (30 * 15 ) / 5
    cout << "Value of ((a + b) * c) / d is  :" << e << endl ;

    e = (a + b) * (c / d);    // (30) * (15/5)
    cout << "Value of (a + b) * (c / d) is  :" << e << endl ;

    e = a + (b * c) / d;      //  20 + (150/5)
    cout << "Value of a + (b * c) / d is  :" << e << endl ;

    return 0;
}

```

STATEMENTS AND OPERATORS

Statements

Decision making structure for statements



1. If statement
2. If...else statement
3. switch statement

STATEMENTS AND OPERATORS

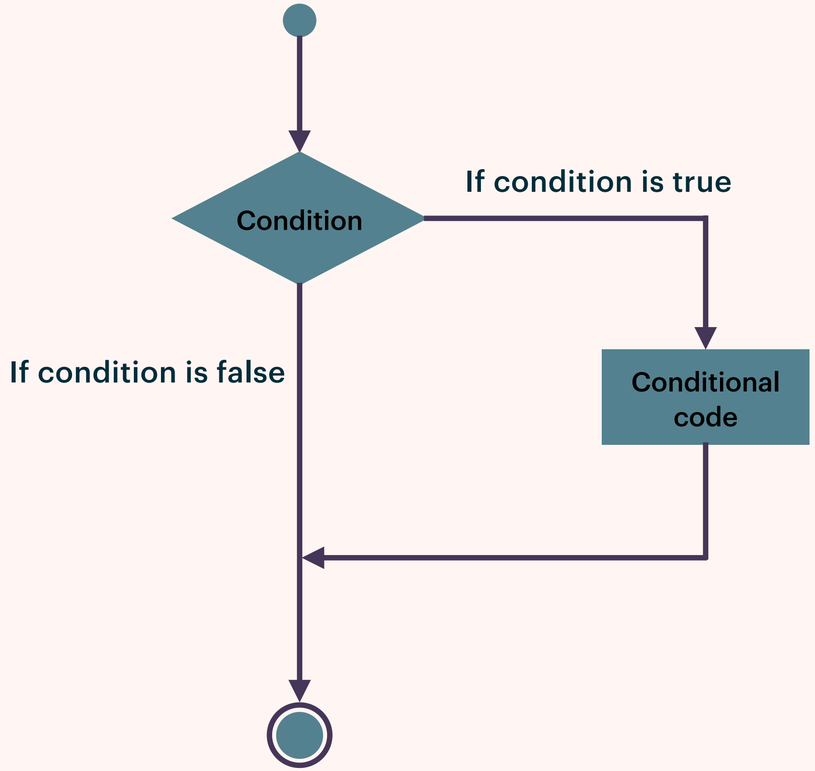
if Statements

Syntax

```
if(boolean_expression){
    //statement will execute if the boolean expression is true
    }
```

Create project name as ifStatement and add following code

```cpp
#include <iostream>

using namespace std;

int main(){
    int a = 10;

    if( a < 20 ){
        cout << "a is less than 20;" << endl;
    }
    cout << "Value of a is " << a << endl;
    return 0;
}
```

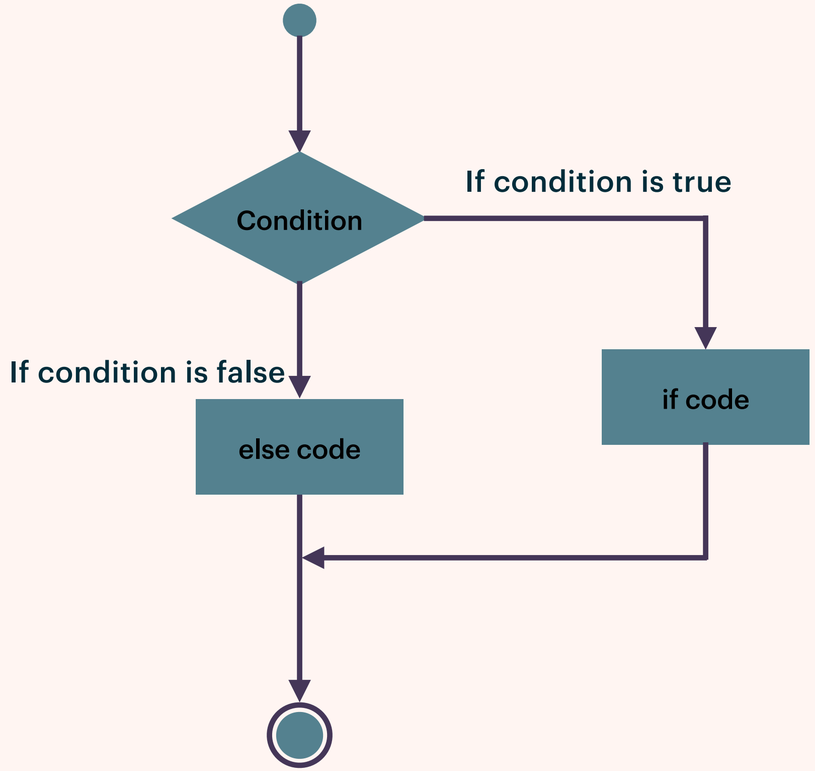


STATEMENTS AND OPERATORS

if...else Statements

Syntax

```
if(boolean_expression){
    //statement will execute if the boolean expression is true
    }else{
    //statement will execute if the boolean expression is false
    }
```



Create project name as ifelseStatement and add following code

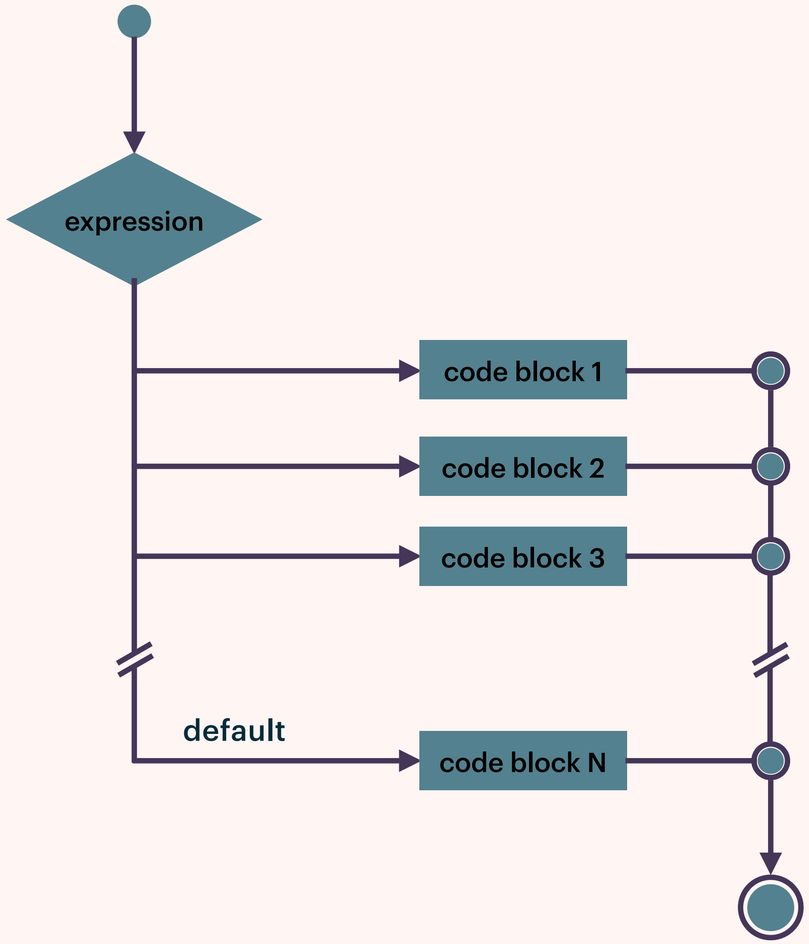
```cpp
#include <iostream>

using namespace std;

int main(){
    int a = 100;

    if( a < 20 ){
        cout << "a is less than 20;" << endl;
    }else{
        cout << "a is not less than 20;" << endl;
    }
    cout << "Value of a is " << a << endl;
    return 0;
}
```

STATEMENTS AND OPERATORS

switch Statements

Syntax

```
switch(expression){
    case constant-expression :
        statement;
        break;
    case constant-expression :
        statement;
        break;
        ....
    default :
       statement;
    }
```



STATEMENTS AND OPERATORS

switch Statements

Create project name as switchStatement and add following code

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

int main(){
    char grade = 'D';

   switch(grade) {
      case 'A' :
         cout << "Excellent!" << endl;
         break;
      case 'B' :
      case 'C' :
         cout << "Well done" << endl;
         break;
      case 'D' :
         cout << "You passed" << endl;
         break;
      case 'F' :
         cout << "Better try again" << endl;
         break;
      default :
         cout << "Invalid grade" << endl;
   }
   cout << "Your grade is " << grade << endl;

   return 0;
}
```

```
You passed
Your grade is D
```

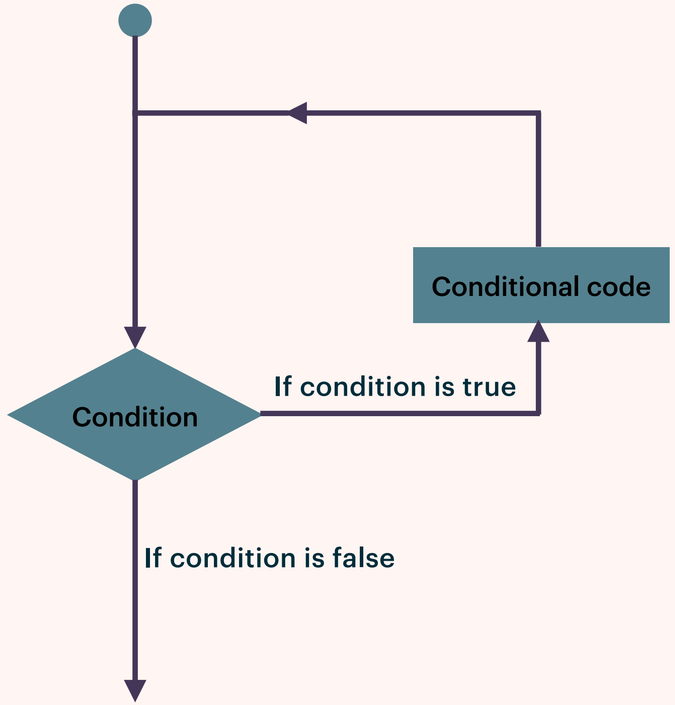
LOOPS

Section five

LOOPS

What is loop?

- Loops can execute a block of code as long as a specified condition is reached.
- A loop statement allows us to execute a statement or group of statements multiple times and following is the general from of a loop statement in most of the programming languages .



LOOPS

Type of loops

- **while**
  - Repeats a statement or group of statements while a given condition is true. It tests the condition before executing the loop body
- **do while**
  - Like a ‘while’ statement, except that it tests the condition at the end of the loop body.
- **for loop**
  - Execute a sequence of statements multiple times and abbreviates the code that manages the loop variable

LOOPS

loop control statement

- **break**
  - ☆ **Terminates the loop or switch statement and transfers execution to the statement immediately following the loop or switch**
- **continue**
  - ☆ **Causes the loop to skip the remainder of its body and immediately retest its condition.**
- **goto**
  - ☆ **Transfers control to the labeled statement**

LOOPS

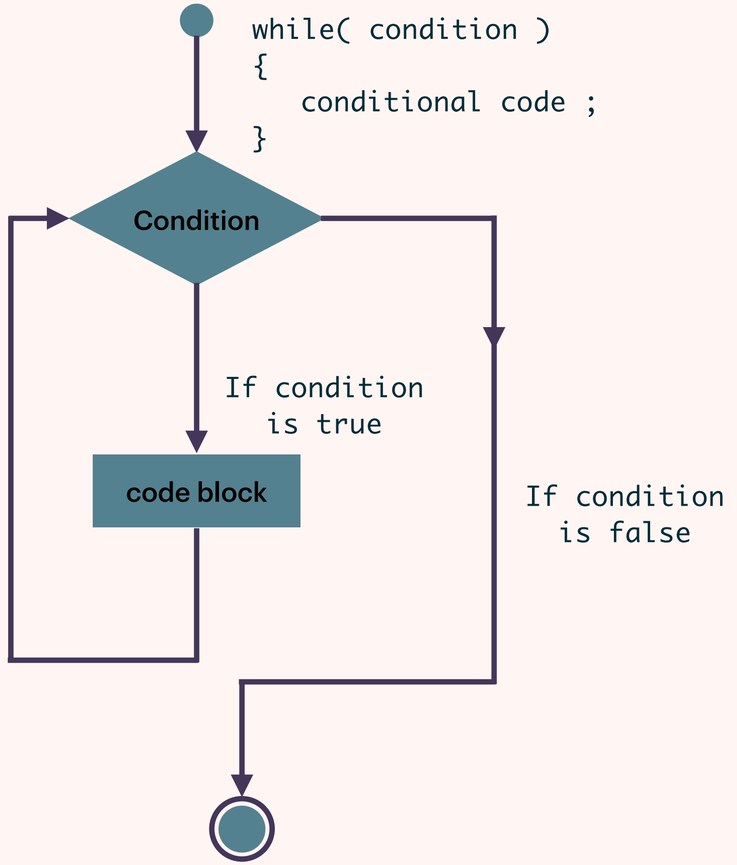
while loop

Syntax

```
while(condition){
    statement(s)
}
```

Create project name as whileLoop and add following code

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

int main(){
    // Local variable declaration:
   int a = 10;

   // while loop execution
   while( a < 20 ) {
      cout << "value of a: " << a << endl;
      a++;
   }

   return 0;
}
```



LOOPS

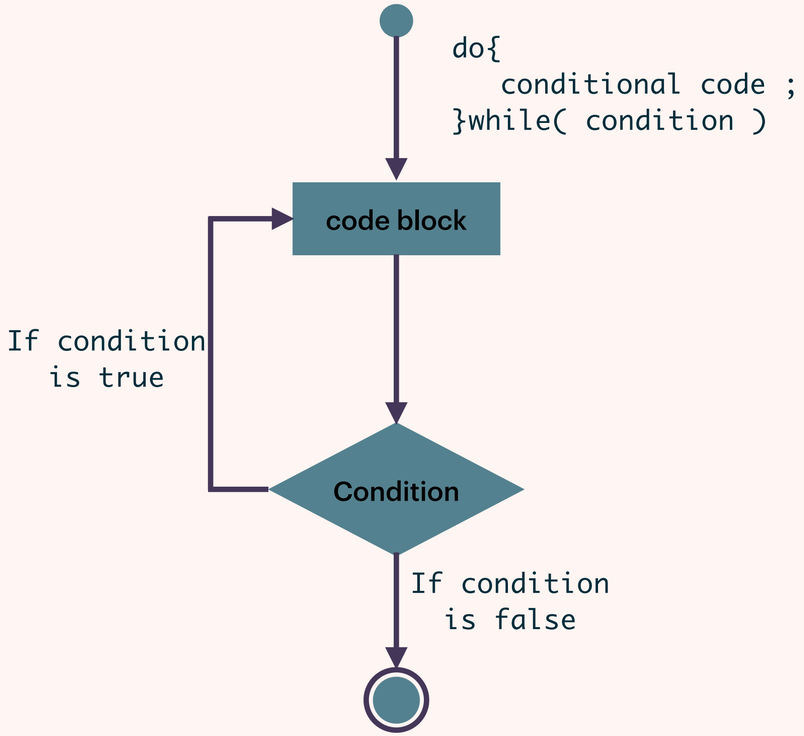
do... while loop

Syntax

```
do {
    statement(s);
}while(condition)
```

Create project name as doWhileLoop and add following code

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

int main(){
    // Local variable declaration:
   int a = 10;

   // do loop execution
   do {
       cout << "value of a: " << a << endl;
       a = a + 1;
   } while( a < 20 );

   return 0;
}
```



LOOPS

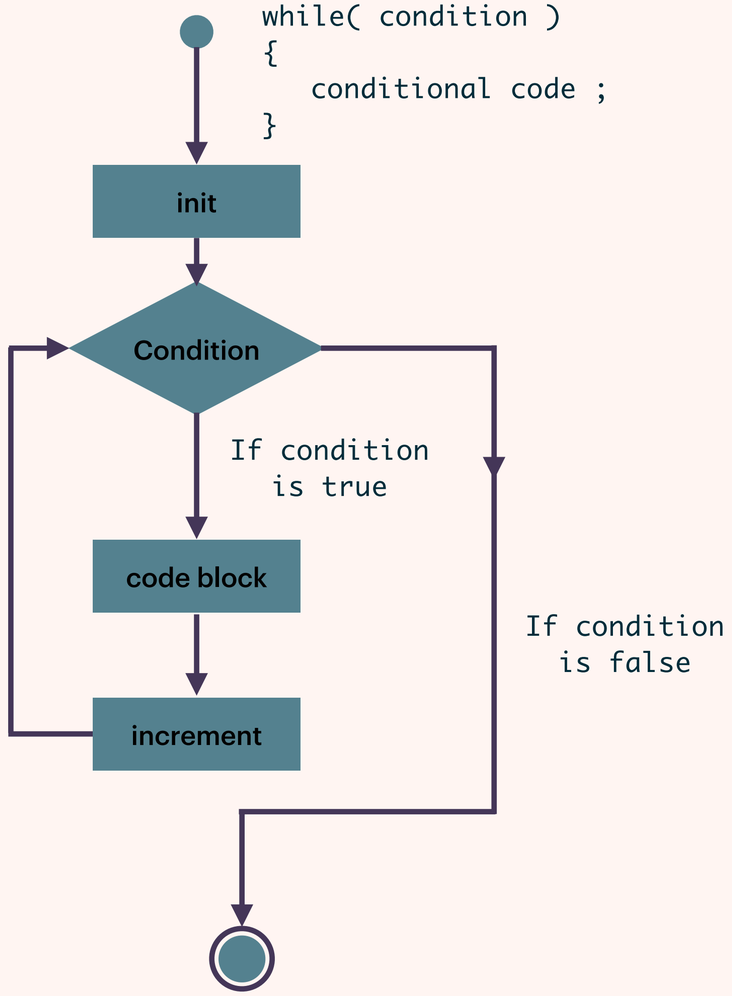
for loop

Syntax

```
for(init; condition; increment){
    statement(s)
}
```

Create project name as forLoop and add following code

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

int main(){
    // for loop execution
   for( int a = 10; a < 20; a = a + 1 ) {
      cout << "value of a: " << a << endl;
   }

   return 0;
}
```

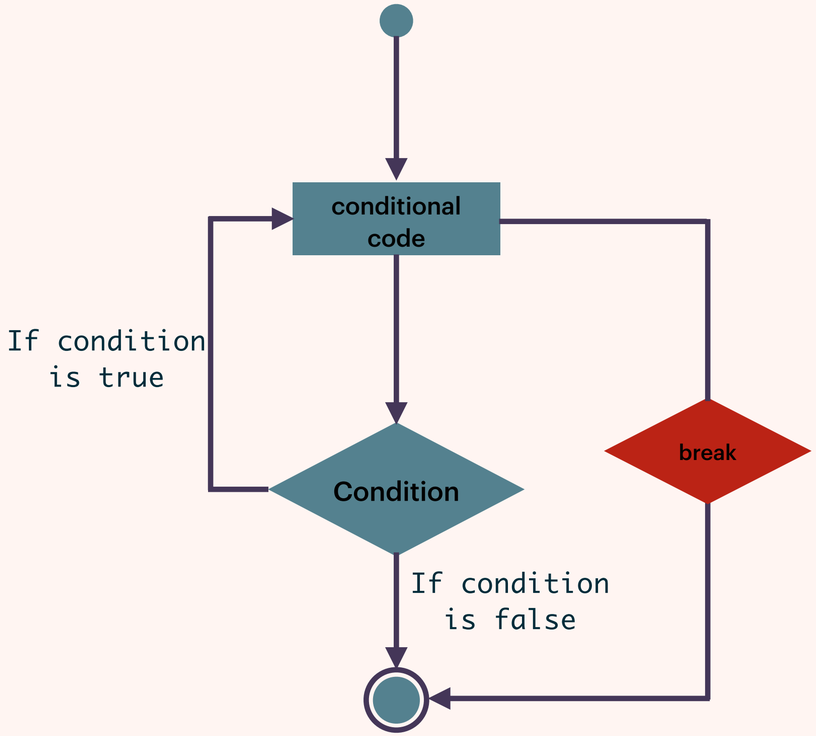


LOOPS

break statement

Create project name as breakStatement and add following code

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

int main(){
    int a = 10;

   do {
      cout << "value of a: " << a << endl;
      a = a + 1;
      if( a > 15) {
         break; // terminate the loop
      }
   } while( a < 20 );

   return 0;
}
```

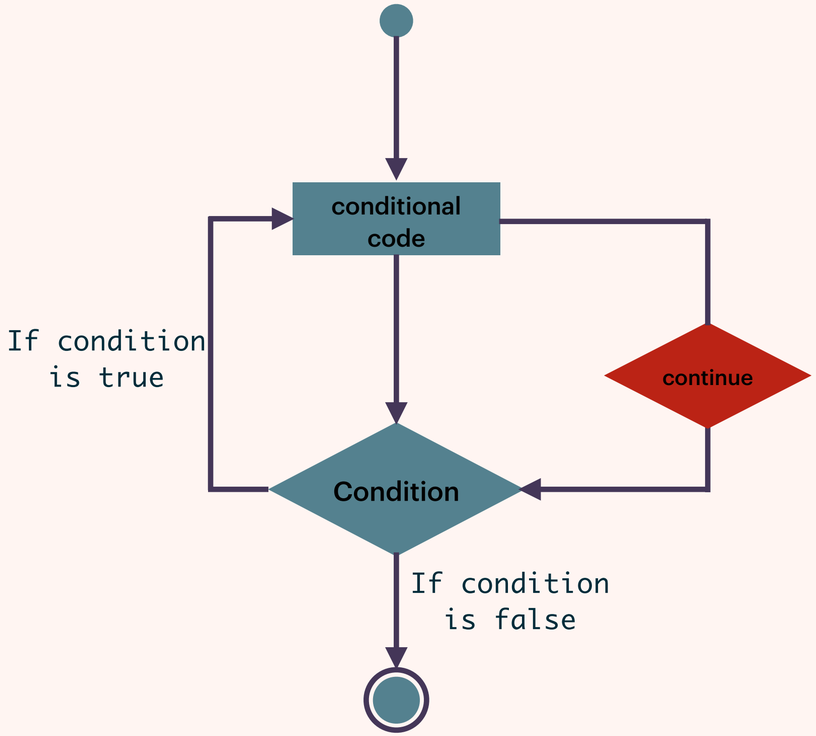


LOOPS

continue statement

Create project name as continueStatement and add following code

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

int main(){
    int a = 10;

    do {
        if( a == 15) {
            a = a + 1;
            continue; // skip the iteration.
        }
        cout << "value of a: " << a << endl;
        a = a + 1;
    }while( a < 20 );

    return 0;
}
```

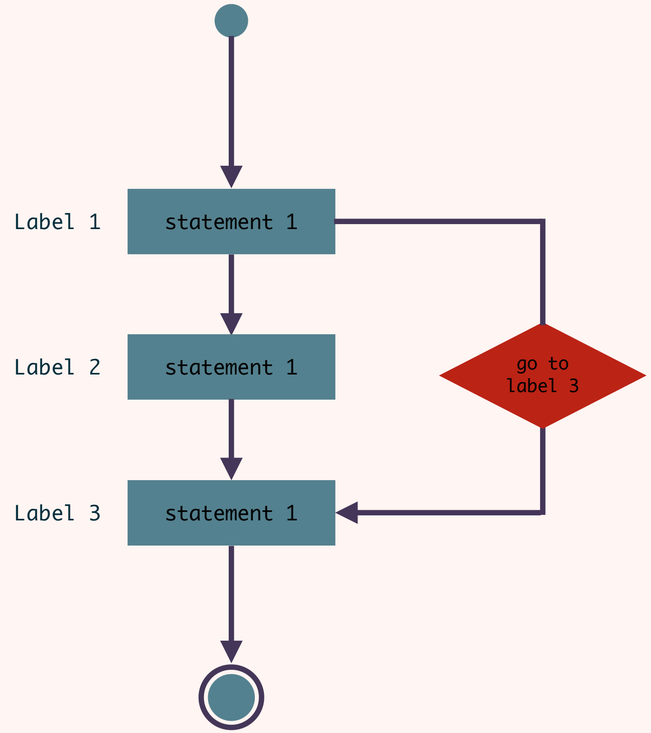


LOOPS

go to statement

Create project name as gotoStatement and add following code

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

int main(){
    int a = 10;

   LOOP:do {
      if( a == 15) {

         a = a + 1;
         goto LOOP; // skip the iteration.
      }
      cout << "value of a: " << a << endl;
      a = a + 1;
   }
   while( a < 20 );

   return 0;
}
```



FUNCTION

Section six

FUNCTION

What is function?

- A function is a block of code which only runs when it is called.
- You can pass data, known as parameters, into a function.
- Functions are used to perform certain actions, and they are important for reusing code: Define the code once, and use it many times.

Syntax

```
void myFunction() {
  //code to be executed
}
```

FUNCTION

Creating & Call function

Create project name as createAndCallFunction and add following code

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

void myFunction(){
    cout << "My Function is executed" << endl;
}

void otherFunction();

int main(){
    myFunction();
    otherFunction();
}

void otherFunction(){
    cout << "Other Function is executed" << endl;
}
```

FUNCTION

Function parameters

Create project name as functionWithParameter and add following code

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

void nameFun(string name){
        cout <<"My name is " << name <<endl;
    }

int main(){
    nameFun("Ko Ko");
    nameFun("Ma Ma");
    nameFun("Mg Mg");
    return 0;
}
```

Syntax

```
void functionName(p1){
  //code to be executed
}
```

FUNCTION

Function multi parameters

Create project name as functionWithMultiParameter and add following code

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

void multiPara(string role, string name){
    cout << "My "<< role << " name is " << name <<endl;
}

int main(){
    multiPara("Father","U Hla");
    multiPara("Mother","Daw Mya");
    multiPara("Brother","Ko Ko");
    return 0;
}
```

Syntax

```
void functionName(p1,p2){
  //code to be executed
}
```

FUNCTION

Function default parameters

Create project name as functionWithDefaultParameter and add following code

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

void showCountry(string country="USA"){
    cout << "I want to visit " << country << endl;
}

int main(){
    showCountry("UK");
    showCountry("Thai");
    showCountry();
    return 0;
}
```

Syntax

```
void functionName(p1=“default value”){
  //code to be executed
}
```

FUNCTION

Function with return

Create project name as functionWithReturn and add following code

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

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

int main(){
    cout << "Sum of 3 and 5 is " << sumNums(3,5) << endl;
    return 0;
}
```

Syntax

```
Int functionName(p1,p2){
  return result; //code to be executed
}
```

FUNCTION

Function overloading

Create project name as functionOverloading and add following code

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

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

double plusFun(double c, double d){
    return c + d;
}

int main(){
    int num1 = plusFun(3, 4);
    double num2 = plusFun(3.3, 4.4);
    cout << "Int Plus Function is " << num1 <<endl;
    cout << "Float Plus Function is" << num2 <<endl;
    return 0;
}
```

Syntax

```cpp
int plusFun(int p1,int p2){
  return result; //code to be executed
}
Float plusFun(float p1,float p2){
  return result; //code to be executed
}
```

FUNCTION

Function recursion

Create project name as functionResursion and add following code

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

int sum(int i){
    if( i > 0){
        return i + sum(i - 1);
    }else{
        return 0;
    }
}

int main(){
    int result = sum(10);
    cout << "Adding 1 to 10 is " << result << endl;
    return 0;
}
```

Process

```
10+sum(10-1);
10+(9+sum(9-1));
10+(9+(8+sum(8-1)));
…..
10+9+8+7+6+5+4+3+2+1+0
```

ARRAYS, STRING AND POINTER

Section seven

ARRAY

What is array?

- Arrays are used to store multiple values in a single variable, instead of declaring separate variables for each value.
- To declare an array, define the variable type, specify the name of the array followed by square brackets and specify the number of elements it should store

Syntax

```
type arrayName [ arraySize ];
```

ARRAY

Creating & Output array

Create project name as createAndOutputArray and add following code

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

int main(){
    string cars[4] = {"Volvo","BMW","Ford","Mazada"};
    for(int i = 0; i < sizeof(cars); i++){
        cout << cars[i] << endl;
    }
    return 0;
}
```

ARRAY

Modifying array

Create project name as modifyingArray and add following code

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

int main(){
    string cars[4] = {"Volvo","BMW","Ford","Mazada"};
    cout << "Changing array" << endl;
    cars[0] = "Toyota";
    for (string k : cars) {
        cout << k << "\n";
    }
    return 0;
}
```

ARRAY

Multi-Dimensional array

Create project name as multiDimensionalArray and add following code

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

int main(){
    //Two Dimensional Array
    string nums[2][4] = {
          { "1", "2", "3", "4" },
          { "5", "6", "7", "8" }
        };
    //Three Dimensional Array
    string letters[2][2][2] = {
          {
            { "A", "B" },
            { "C", "D" }
          },
          {
            { "E", "F" },
            { "G", "H" }
          }
        };
    cout << "Outputing number 7 from array " << nums[1][2] <<endl;
    cout << "Outputing letter F from array " << letters[1][0][1] << endl;
    return 0;
}
```

ARRAY

Two-Dimensional array

Create project name as accessAndModifyTwoDimensionArray and add following code

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

int main(){
    int nums[2][5] = {
        {1, 2, 3, 4, 5},
        {6, 7, 8, 9, 10}
    };
    nums[1][3] = 11;
    for(int i = 0; i < 2; i++){
        for(int j = 0; j < 5; j++){
            cout << "Index " << i << " , " << j << " is " << nums[i][j] << endl;
       }
            cout << endl;
    }
    return 0;
}
```

ARRAY

Three-Dimensional array

Create project name as accessAndModifyThreeDimensionArray and add following code

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

int main (){
    string letters[2][3][4]={
        {
            {"AA", "AB" ,"AC", "AD"},
            {"BA", "BB", "BC", "BD"},
            {"CA", "CB", "CC", "CD"},
        },
        {
            {"DA", "DB" ,"DC", "DD"},
            {"EA", "EB", "EC", "ED"},
            {"FA", "FB", "FC", "FD"},
        }
    };
    letters[1][2][0] = "ForeverAlone";
    for(int i = 0; i < 2; i++){
        for(int j = 0; j < 3; j++) {
            for(int k = 0; k < 4; k++){
                cout << "Index " << i << " " << j << " " << k << " is " << letters[i][j][k] << endl;
            }
            cout << endl;
        }
        cout << endl;
    }
}
```

ARRAY

No length array

Create project name as arrayWithNoLength and add following code

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

int main(){
//     int nums[4];
//     nums = {2,3,5,6,7};
    string str[] = {"a","b"};
    for(string i: str){
        cout << i << endl;
    }
    return 0;
}
```

ARRAY

Passing Array to Function

Create project name as passingArrayToFunction and add following code

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

double getAverage(int arr[], int size);

int main(){
    int balance[5]={10,20,30,40,50};
    double avg;
    avg = getAverage(balance,5);
    cout << "Average value is: " << avg << endl;

    return 0;
}

double getAverage(int arr[], int size){
    int i,sum=0;
    double avg;

    for(i = 0; i < 5; i++){
        sum += arr[i];
    }
    avg = sum / size;
    return avg;
}
```

STRING

What is string?

- **This string is actually a one-dimensional array of characters which is terminated by a null character '\0'.**

String Example with char

Create project name as stringExample and add following code

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

int main () {
   char greeting[6] = {'H', 'e', 'l', 'l', 'o', '\0'};
   cout << "Greeting message: ";
   cout << greeting << endl;
   return 0;
}
```

STRING

String Functions

Functions	Descriptions
strcpy(s1, s2);	Copies string s2 into string s1.
strcat(s1, s2);	Concatenates string s2 onto the end of string s1.
strlen(s1);	Returns the length of string s1.
strcmp(s1, s2);	Returns 0 if s1 and s2 are the same; less than 0 if s1<s2; greater than 0 if s1>s2.
strchr(s1, ch);	Returns a pointer to the first occurrence of character ch in string s1.
strstr(s1, s2);	Returns a pointer to the first occurrence of string s2 in string s1.

STRING

String Function with Cstring

Create project name as stringFunctionWithCstring and add following code

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

int main () {
   char str1[10] = "Hello";
   char str2[10] = "World";
   char str3[10];
   int  len ;

   // copy str1 into str3
   strcpy( str3, str1);
   cout << "strcpy( str3, str1) : " << str3 << endl;

   // concatenates str1 and str2
   strcat( str1, str2);
   cout << "strcat( str1, str2): " << str1 << endl;

   // total lenghth of str1 after concatenation
   len = strlen(str1);
   cout << "strlen(str1) : " << len << endl;

   return 0;
}
```

STRING

String Function with C++string

Create project name as stringFunctionWithC++string and add following code

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

int main () {
   string str1 = "Hello";
   string str2 = "World";
   string str3;
   int  len ;

   // copy str1 into str3
   str3 = str1;
   cout << "str3 : " << str3 << endl;

   // concatenates str1 and str2
   str3 = str1 + str2;
   cout << "str1 + str2 : " << str3 << endl;

   // total length of str3 after concatenation
   len = str3.size();
   cout << "str3.size() :  " << len << endl;

   return 0;
}
```

POINTER

What is pointer?

- **A pointer is a variable whose value is the address of another variable**

Pointer Example

Create project name as pointerExample and add following code

```cpp
#include <iostream>

using namespace std;
int main () {
   int  var1;
   char var2[10];
   cout << "Address of var1 variable: ";
   cout << &var1 << endl;
   cout << "Address of var2 variable: ";
   cout << &var2 << endl;
   return 0;
}
```

POINTER

Pointer to value

Create project name as poinerToValue and add following code

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

int main () {
   int  var = 20;   // actual variable declaration.
   int  *ip;        // pointer variable
   ip = &var;       // store address of var in pointer variable
   cout << "Value of var variable: ";
   cout << var << endl;
   // print the address stored in ip pointer variable
   cout << "Address stored in ip variable: ";
   cout << ip << endl;
   // access the value at the address available in pointer
   cout << "Value of *ip variable: ";
   cout << *ip << endl;
   return 0;
}
```

POINTER

Null Pointer

Create project name as nullPoimter and add following code

```
#include <iostream>

using namespace std;
int main () {
   int  *ptr = NULL;
   cout << "The value of ptr is " << ptr ;

   return 0;
}
```

POINTER

Pointer Arithmetic(increment)

Create project name as incrementPointer and add following code

```cpp
#include <iostream>
using namespace std;
const int MAX = 3;

int main () {
   int  var[MAX] = {10, 100, 200};
   int  *ptr;
   ptr = var;

   for (int i = 0; i < MAX; i++) {
      cout << "Address of var[" << i << "] = ";
      cout << ptr << endl;

      cout << "Value of var[" << i << "] = ";
      cout << *ptr << endl;
      // point to the next location
      ptr++;
   }
   return 0;
}
```

POINTER

Pointer Arithmetic(decrement)

Create project name as decrementPointer and add following code

```
#include <iostream>
using namespace std;
const int MAX = 3;

int main () {
   int  var[MAX] = {10, 100, 200};
   int  *ptr;
   // let us have address of the last element ir
   ptr = &var[MAX-1];
   for (int i = MAX; i > 0; i--) {
      cout << "Address of var[" << i << "] = ";
      cout << ptr << endl;
      cout << "Value of var[" << i << "] = ";
      cout << *ptr << endl;
      // point to the previous location
      ptr--;
   }
   return 0;
}
```

POINTER

Array of Pointer

Create project name as arrayOfPointer and add following code

```
#include <iostream>
using namespace std;
const int MAX = 3;

int main () {
   int  var[MAX] = {10, 100, 200};
   int *ptr[MAX];

   for (int i = 0; i < MAX; i++) {
      ptr[i] = &var[i]; // assign the address of
   }

   for (int i = 0; i < MAX; i++) {
      cout << "Value of var[" << i << "] = ";
      cout << *ptr[i] << endl;
   }
   return 0;
}
```

POINTER

Pointer to Pointer

Create project name as pointerToPointer and add following code

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

int main () {
    int  var;
    int  *ptr;
    int  **pptr;
    var = 3000;
   // take the address of var
   ptr = &var;
   // take the address of ptr using address of operator &
   pptr = &ptr;

   // take the value using pptr
   cout << "Value of var :" << var << endl;
   cout << "Value available at *ptr :" << *ptr << endl;
   cout << "Value available at **pptr :" << **pptr << endl;
   return 0;
}
```

CLASSES AND OBJECTS

Section eight

CLASSES AND OBJECTS

What are classes and objects

- Classes and objects are the two main aspects of object-oriented programming.
- A class definition starts with the keyword class followed by the class name; and the class body, enclosed by a pair of curly braces.
- A class provides the blueprints for objects, so basically an object is created from a class.

Syntax of class

```
class box {
  public:
      double length;
      double breadth;
      double height;
}
```

Syntax of object

```
class box {
  public:
      double length;
      double breadth;
      double height;
}
```

CLASSES AND OBJECTS

Using classes and objects

Create project name as usingClassesAndObjects and add following code

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

class Box {
   public:
      double length;   // Length of a box
      double breadth;  // Breadth of a box
      double height;   // Height of a box
};
int main() {
   Box Box1;        // Declare Box1 of type Box
   Box Box2;        // Declare Box2 of type Box
   double volume = 0.0;     // Store the volume of a box here
   // box 1 specification
   Box1.height = 5.0;
   Box1.length = 6.0;
   Box1.breadth = 7.0;
   // box 2 specification
   Box2.height = 10.0;
   Box2.length = 12.0;
   Box2.breadth = 13.0;
   // volume of box 1
   volume = Box1.height * Box1.length * Box1.breadth;
   cout << "Volume of Box1 : " << volume <<endl;
   // volume of box 2
   volume = Box2.height * Box2.length * Box2.breadth;
   cout << "Volume of Box2 : " << volume <<endl;
   return 0;
}
```

CLASSES AND OBJECTS

Inside class definitions

Create project name as InsideClassDefinition and add following code

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

class myClass{
    public:
        void myMethod(){
            cout << "Hello Inside Class" << endl;
        }

};

int main(){
    myClass myObj;
    myObj.myMethod();
    return 0;
}
```

CLASSES AND OBJECTS

Outside class definitions

Create project name as outsideClassDefinition and add following code

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

class myClass{
    public:
        void myMethod();
};

void myClass::myMethod(){
    cout <<"Hello Outside class" <<endl;
}

int main() {
  myClass myObj;      // Create an object of MyClass
  myObj.myMethod();  // Call the method
  return 0;
}
```

CLASSES AND OBJECTS

Outside class with parameter

Create project name as outsideClassWithParameter and add following code

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

class Car {
  public:
    int speed(int maxSpeed);
};

int Car::speed(int maxSpeed) {
  return maxSpeed;
}

int main() {
  Car myObj; // Create an object of Car
  cout << myObj.speed(200) << endl;
  return 0;
}
```

CLASSES AND OBJECTS

Class Access Modifiers or Access Specifiers

- **Data hiding is one of the important features of Object Oriented Programming which allows preventing the functions of a program to access directly the internal representation of a class type.**

- **There are three type of access modifier public, private and protected**

public => members are accessible from outside the class.

private => members cannot be accessed (or viewed) from outside the class

protected => members cannot be accessed from outside the class, however, they can be accessed in inherited classes.

CLASSES AND OBJECTS

Private access modifier or specifier

Create project name as privateAccessModifier and add following code

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

class MyClass {
  public:    // Public access specifier
    int x;   // Public attribute
  private:   // Private access specifier
    int y;   // Private attribute
    int z;   // ommited access specifier, will be private
};

int main() {
  MyClass myObj;
  myObj.x = 25;  // Allowed (x is public)
  myObj.y = 50;  // Not allowed (y is private)
  myObj.z = 100  // Not allowed (z is private)
  return 0;
}
```

CLASSES AND OBJECTS

Get and Set of private access modifier

Create project name as getAndSetOfPrivateModifier and add following code

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

class MyClass {
  public:    // Public access specifier
    int x;   // Public attribute
    void setY(int yValue);
    int getY( void );
  private:   // Private access specifier
    int y;   // Private attribute]
};

void MyClass::setY(int yValue){
    y = yValue;
}

int MyClass::getY(void){
    return y;
}

int main() {
  MyClass myObj;
  myObj.setY(10);
  cout << "Getting private of y is " << myObj.getY() << endl;
  return 0;
}
```

CLASSES AND OBJECTS

Protected access modifier or specifier

Create project name as privateAccessModifier and add following code

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

class MyClass {
  public:    // Public access specifier
    int x;   // Public attribute
  private:   // Private access specifier
    int y;   // Private attribute
    int z;   // ommited access specifier, will be private
};

int main() {
  MyClass myObj;
  myObj.x = 25;  // Allowed (x is public)
  myObj.y = 50;  // Not allowed (y is private)
  myObj.z = 100  // Not allowed (z is private)
  return 0;
}
```