Fundamentals of Computer Programming

CS-110

Course Instructor: Dr. Momina Moetesum

Operators & Basic Programming Errors

Week 2-B

```
51 template <typename T, typename I>
52 void TestIntegral(const T values[], int num_values) {
53   for (int i = 0; i < num_values; ++i) {
54     T t0 = values[i];
55     I i0 = absl::bit_cast<I>(t0);
56     T t1 = absl::bit_cast<T>(i0);
57     I i1 = absl::bit_cast<I>(t1);
58     ASSERT_EQ(0, memcmp(&t0, &t1, sizeof(T)));
59     ASSERT_EQ(i0, i1);
60   }
61 }
62
63 TEST(BitCast, Bool) {
64   static const bool bool_list[] = { false, true };
65   TestMarshall<bool>(bool_list, ABSL_ARRAYSIZE(bool_list));
66 }
67
68 TEST(BitCast, Int32) {
69   static const int32_t int_list[] =
70     { 0, 1, 100, 2147483647, -1, -100, -2147483647, -2147483647-1 };
71   TestMarshall<int32_t>(int_list, ABSL_ARRAYSIZE(int_list));
72 }
73
74 TEST(BitCast, Int64) {
75   static const int64_t int64_list[] =
76     { 0, 1, 1LL << 40, -1, -(1LL<<40) };
77   TestMarshall<int64_t>(int64_list, ABSL_ARRAYSIZE(int64_list));
```

Learnning Objectives

01

To understand basic unary and binary operators

02

To understand operator precedence in C++

03

To practice application of different types of operators

04

To understand the common programming errors

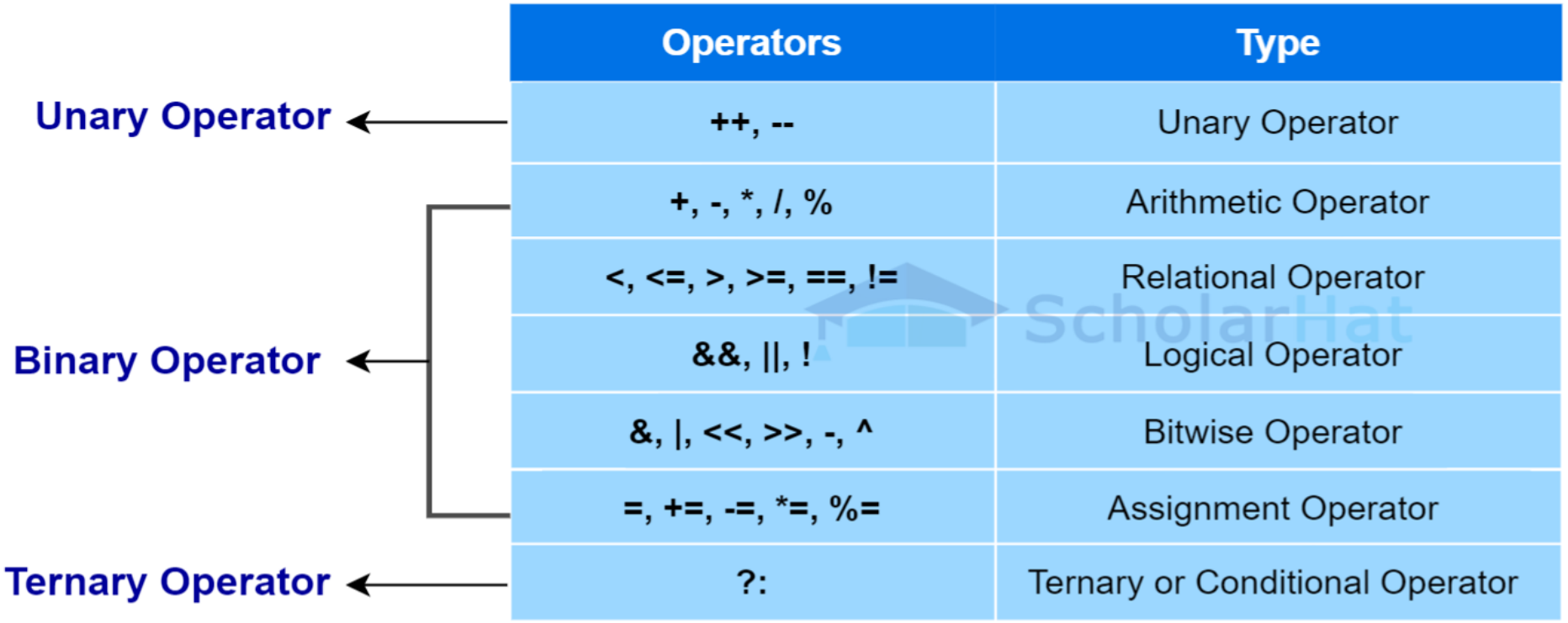
Operators in C++

An **operator** is a symbol that operates on a value to perform specific mathematical or logical computations.

An operator operates the **operands**.

- Unary Operators
- Arithmetic Operators
- Relational Operators
- Logical Operators
- Bitwise Operators
- Assignment Operator
- Ternary or Conditional Operator

Operators in C++



Operators	Type
++, --	Unary Operator
+, -, *, /, %	Arithmetic Operator
<, <=, >, >=, ==, !=	Relational Operator
&&, ||, !	Logical Operator
&, |, <<, >>, -, ^	Bitwise Operator
=, +=, -=, *=, %=	Assignment Operator
?:	Ternary or Conditional Operator

https://www.scholarhat.com/tutorial/cpp/identifiers-and-operators-in-cpp

Arithmetic Operators

- C++ Arithmetic operators are of 2 types:
  - Unary Arithmetic Operator
  - Binary Arithmetic Operator

Arithmetic operators

Operator	Description	Example
+	Adds two operands	A + B will give 30 ,if A=20 and B=10
-	Subtracts second operand from the first	A - B will give -10
*	Multiply both operands	A * B will give 200
/	Divide 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

Example

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

int main() {
    int a, b;
    a = 7;
    b = 2;

    // printing the sum of a and b
    cout << "a + b = " << (a + b) << endl;

    // printing the difference of a and b
    cout << "a - b = " << (a - b) << endl;

    // printing the product of a and b
    cout << "a * b = " << (a * b) << endl;

    // printing the division of a by b
    cout << "a / b = " << (a / b) << endl;

    // printing the modulo of a by b
    cout << "a % b = " << (a % b) << endl;

    return 0;
}
```

Output

```
a + b = 9
a - b = 5
a * b = 14
a / b = 3
a % b = 1
```

Pre- and Post-Increment/Decrement Operators

- In **++x**, the variable's value is first increased/incremented before being utilised in the program.
- In **x++**, a variable's value is assigned before it is increased/incremented.
- Similarly happens for the decrement operator.

```cpp
// C++ Program to demonstrate the
// increment and decrement operators
#include <iostream>
using namespace std;

int main()
{
    int x = 5;

    // This statement Incremented 1
    cout << "x++ is " << x++ << endl;

    // This statement Incremented 1
    // from already Incremented
    // statement resulting in
    // Incrementing of 2
    cout << "++x is " << ++x << endl;

    int y = 10;

    // This statement Decremented 1
    cout << "y-- is " << y-- << endl;

    // This statement Decremented 1
    // from already Decremented
    // statement resulting in
    // Decrementing of 2
    cout << "--y is " << --y << endl;

    return 0;
}
```

Example

```
// Working of increment and decrement operators

#include <iostream>
using namespace std;

int main() {
    int a = 10, b = 100, result_a, result_b;

    // incrementing a by 1 and storing the result in result_a
    result_a = ++a;
    cout << "result_a = " << result_a << endl;


    // decrementing b by 1 and storing the result in result_b
    result_b = --b;
    cout << "result_b = " << result_b << endl;

    return 0;
}
```

Output

```
result_a = 11
result_b = 99
```

Other Unary Operators

Unary operators are the operators that perform operations on a single operand to produce a new value.

Types of unary operators

Types of unary operators are mentioned below:

- Unary minus (–)
- Increment (++)
- Decrement (--)
- NOT (!)
- Addressof operator (&)
- sizeof()

Assignment Operators

- Assignment operators are used to assigning value to a variable.
- The left side operand of the assignment operator is a variable and right side operand of the assignment operator is a value.
- The value on the right side must be of the same data-type of the variable on the left side otherwise the compiler will raise an error.

Operator	Name of Operator	Example
+=	Addition Assignment	a = 10; c = a += 5; (ie, a = a + 5) c = 15
-=	Subtraction Assignment	a = 10; c = a -= 5; (ie. a = a – 5) c = 5
*=	Multiplication Assignment	a = 10; c = a *= 5; (ie. a = a * 5) c = 50
/=	Division Assignment	a = 10; c = a /= 5; (ie. a = a / 5) c = 2
%=	Modulus Assignment	a = 10; c = a %= 5; (ie. a = a % 5) c = 0

Relational Operators

Relational operators are used for the comparison of two values to understand the type of relationship a pair of number shares.

OPERATOR	MEANING	EXAMPLE	RESULT
<	Less than	1<2	True
>	Greater than	1>2	False
<=	Less than or equal to	1<=2	True
>=	Greater than or equal to	1>=2	False
==	Equal to	1==2	False
!=	Not equal to	1!=2	True

Example

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

int main() {
    int a, b;
    a = 3;
    b = 5;
    bool result;

    result = (a == b);    // false
    cout << "3 == 5 is " << result << endl;

    result = (a != b);   // true
    cout << "3 != 5 is " << result << endl;

    result = a > b;    // false
    cout << "3 > 5 is " << result << endl;

    result = a < b;    // true
    cout << "3 < 5 is " << result << endl;

    result = a >= b;   // false
    cout << "3 >= 5 is " << result << endl;

    result = a <= b;   // true
    cout << "3 <= 5 is " << result << endl;

    return 0;
}
```

Logical Operators

Logical AND table:

Logical AND			
False	&&	False	False
False	&&	True	False
True	&&	False	False
True	&&	True	True

Logical OR table:

Logical OR			
False	||	False	False
False	||	True	True
True	||	False	True
True	||	True	True

```cpp
// C++ program to demonstrate working of
// logical operators
#include <iostream>
using namespace std;

int main()
{
    int a = 10, b = 4, c = 10, d = 20;

    // logical operators

    // logical AND example
    if (a > b && c == d)
        cout << "a is greater than b AND c is equal to d\n";
    else
        cout << "AND condition not satisfied\n";

    // logical OR example
    if (a > b || c == d)
        cout << "a is greater than b OR c is equal to d\n";
    else
        cout << "Neither a is greater than b nor c is equal "
                " to d\n";

    // logical NOT example
    if (!a)
        cout << "a is zero\n";
    else
        cout << "a is not zero";

    return 0;
}
```

Examples

```cpp
int main() {
    bool result;

    result = (3 != 5) && (3 < 5);     // true
    cout << "(3 != 5) && (3 < 5) is " << result << endl;

    result = (3 == 5) && (3 < 5);    // false
    cout << "(3 == 5) && (3 < 5) is " << result << endl;

    result = (3 == 5) && (3 > 5);    // false
    cout << "(3 == 5) && (3 > 5) is " << result << endl;

    result = (3 != 5) || (3 < 5);    // true
    cout << "(3 != 5) || (3 < 5) is " << result << endl;

    result = (3 != 5) || (3 > 5);    // true
    cout << "(3 != 5) || (3 > 5) is " << result << endl;

    result = (3 == 5) || (3 > 5);    // false
    cout << "(3 == 5) || (3 > 5) is " << result << endl;

    result = !(5 == 2);    // true
    cout << "!(5 == 2) is " << result << endl;

    result = !(5 == 5);    // false
    cout << "!(5 == 5) is " << result << endl;

    return 0;
}
```

Output

```
(3 != 5) && (3 < 5) is 1
(3 == 5) && (3 < 5) is 0
(3 == 5) && (3 > 5) is 0
(3 != 5) || (3 < 5) is 1
(3 != 5) || (3 > 5) is 1
(3 == 5) || (3 > 5) is 0
!(5 == 2) is 1
!(5 == 5) is 0
```

C++ Bitwise Operators

- The **& (bitwise AND)** in C or C++ takes two numbers as operands and does AND on every bit of two numbers. The result of AND is 1 only if both bits are 1.
- The **| (bitwise OR)** in C or C++ takes two numbers as operands and does OR on every bit of two numbers. The result of OR is 1 if any of the two bits is 1.
- The **^ (bitwise XOR)** in C or C++ takes two numbers as operands and does XOR on every bit of two numbers. The result of XOR is 1 if the two bits are different.
- The **<< (left shift)** in C or C++ takes two numbers, the left shifts the bits of the first operand, and the second operand decides the number of places to shift.
- The **>> (right shift)** in C or C++ takes two numbers, right shifts the bits of the first operand, and the second operand decides the number of places to shift.
- The **~ (bitwise NOT)** in C or C++ takes one number and inverts all bits of it.

Bitwise Operator Truth Table

X	Y	X&Y	X|Y	X^Y	~(X)
0	0	0	0	0	1
0	1	0	1	1	1
1	0	0	1	1	0
1	1	1	1	0	0

Example

```
// C++ Program to demonstrate use of bitwise operators
#include <iostream>
using namespace std;

int main() {
    // a = 5(00000101), b = 9(00001001)
    int a = 5, b = 9;

    // The result is 00000001
    cout<<"a = " << a <<","<< " b = " << b <<endl;
    cout << "a & b = " << (a & b) << endl;

    // The result is 00001101
    cout << "a | b = " << (a | b) << endl;

    // The result is 00001100
    cout << "a ^ b = " << (a ^ b) << endl;

    // The result is 11111010
    cout << "~a = " << (~a) << endl;

    // The result is 00010010
    cout<<"b << 1" <<" = "<< (b << 1) <<endl;

    // The result is 00000100
    cout<<"b >> 1 "<<"= " << (b >> 1 )<<endl;

    return 0;
}
```

Output

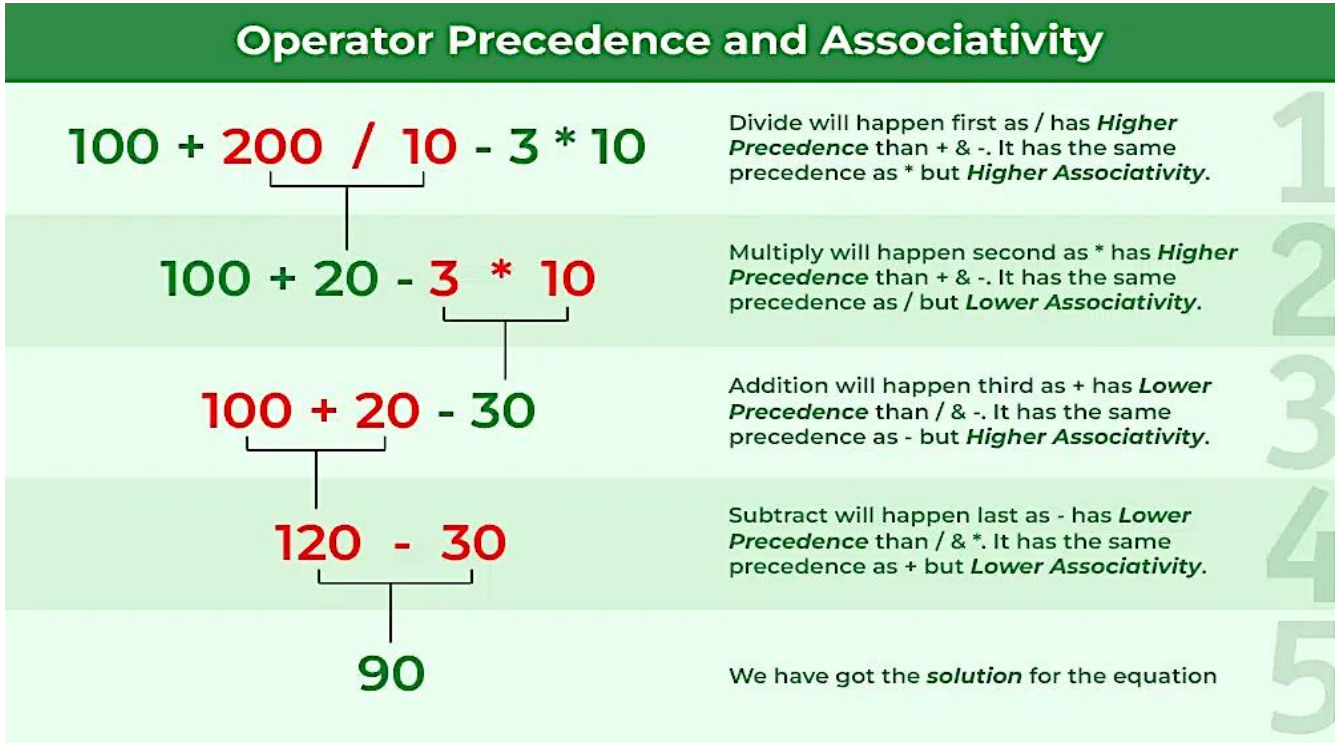
```
a = 5, b = 9
a & b = 1
a | b = 13
a ^ b = 12
~a = -6
b << 1 = 18
b >> 1 = 4
```

Operator Precedence and Associativity

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

Examples

```
exp = 100 + (200 / 10) - 3 * 10
```

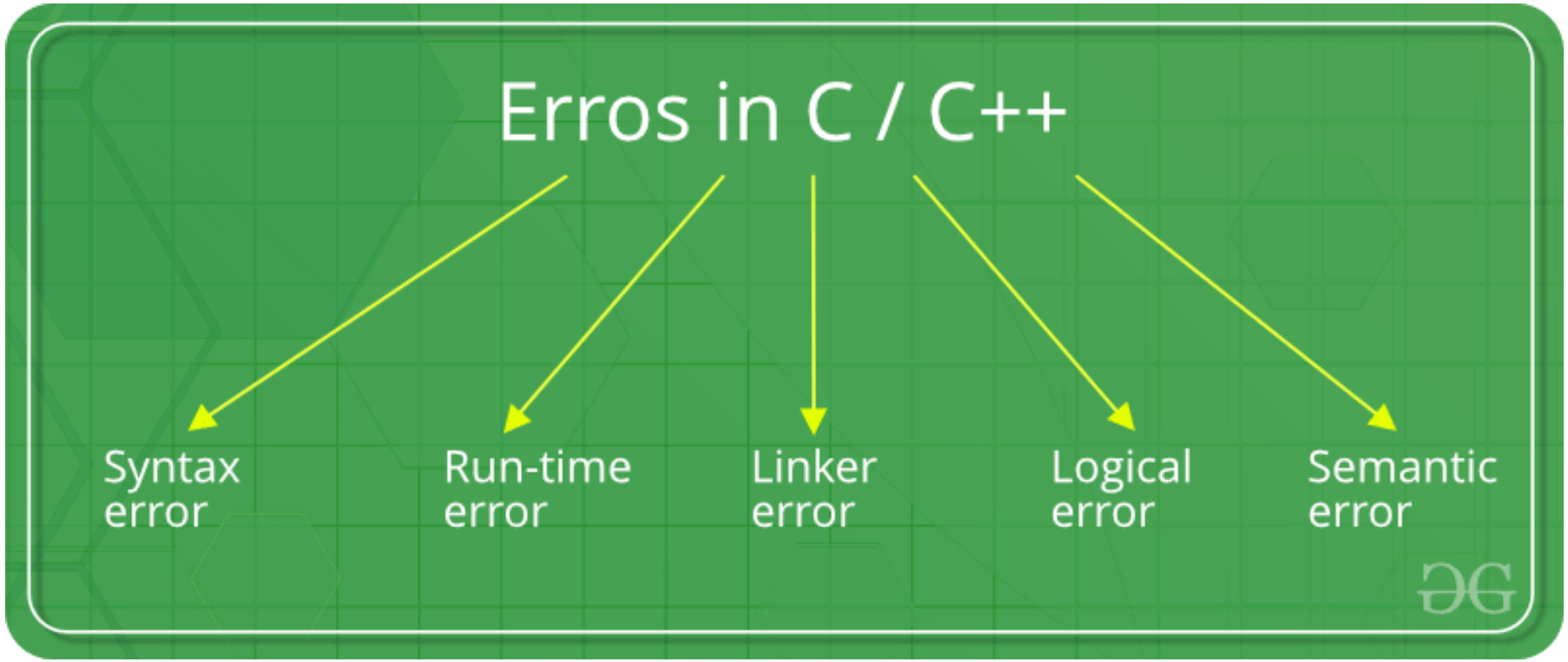


Try This Now!!

```
exp = 100 + 200 / 10 - 3 * 10
```

Types of Errors

https://www.geeksforgeeks.org/errors-in-cc/



Syntax Errors

Errors that occur when you **violate the rules** of writing C/C++ syntax are known as syntax errors. This compiler error indicates something that must be fixed before the code can be compiled. All these errors are detected by compiler and thus are known as compile-time errors.

Most frequent syntax errors are:

- Missing Parenthesis (**}**)
- Printing the value of variable without declaring it
- Missing semicolon like this:

```
// C++ program to illustrate
// syntax error
#include <iostream>
using namespace std;
void main()
{
        int x = 10;
        int y = 15;
        cout << " "<< (x, y) // semicolon missed
}
```

Syntax Errors

Errors that occur when you **violate the rules** of writing C/C++ syntax are known as syntax errors. This compiler error indicates something that must be fixed before the code can be compiled. All these errors are detected by compiler and thus are known as compile-time errors.

Most frequent syntax errors are:

- Missing Parenthesis (})
- Printing the value of variable without declaring it
- Missing semicolon like this:

```
// C++ program to illustrate
// syntax error
#include <iostream>
using namespace std;
int main(void)
{
        while(.) // while() cannot contain "." as an argument.
        {
                cout <<"hello";
        }
        return 0;
}
```

Runtime Error

Errors which occur during program execution(run-time) after successful compilation are called run-time errors. One of the most common run-time error is division by zero also known as Division error. These types of error are hard to find as the compiler doesn't point to the line at which the error occurs.

```
// C++ program to illustrate run-time error
#include <iostream>
#include <bits/stdc++.h>
using namespace std;
void main()
{
        int n = 9, div = 0;
        // wrong logic
        // number is divided by 0,
        // so this program abnormally terminates
        div = n/0;
        cout << "result = "<< div;
}
```

Linker Error

These are errors generated when the executable of the program cannot be generated. This may be due to wrong function prototyping, incorrect header files. One of the most common linker error is writing **Main()** instead of **main()**.

```
// C++ program to illustrate
// linker error
#include <bits/stdc++.h>
using namespace std;
void Main() // Here Main() should be main()
{
        int a = 10;
        cout << " "<< a;
}
```

Logical Error

On compilation and execution of a program, desired output is not obtained when certain input values are given. These types of errors which provide incorrect output but appears to be error free are called logical errors. These are one of the most common errors done by beginners of programming.

```
// C++ program to illustrate logical error
int main()
{
        int i = 0;
        // logical error : a semicolon after loop
        for(i = 0; i < 3; i++);
        {
                cout << "loop ";
                continue;
        }
        return 0;
}
```

Semantic Error

This error occurs when the statements written in the program are not meaningful to the compiler.

```
// C++ program to illustrate semantic error
int main()
{
        int a, b, c;
        a + b = c; //semantic error
}
```

Acknowledgment

- Content of these slides are taken from:
  - https://www.geeksforgeeks.org/
  - https://www.tutorialspoint.com/
  - https://www.programiz.com/