



## Week 2

### Session 2: Reserved words and Declarative & Executable Statements in C++ program

#### Element 1: Recognize simple data types, reserved words, and declaration and executable statements

ECT 124: Writing Programs using C++



## Performance criteria (PC) for E1

**PC1:** Identify simple data types such characters, integers and floating numbers.

**PC2:** Specify the functionality of reserved words in C++.

**PC3:** Differentiate between declarative and executable statements in C++.

**In this lesson!**



## Learning objectives:

By the end of this lesson, the student should be able to:

- ✓ State the common reserved words in C++ program and recognize its functionality.
- ✓ Understand the declarative and executable statements in C++ program and knowing the differences.





# Class Activity 1

# Open Ended Question

In your opinion, what is defined as RESERVED WORDS in a C++ language?

**Please enter your answer here.**

# Quiz

**Can "reserved words" be used in a variable declaration?**

☐ YES

☐ NO





### reserved word for structure marker

- Keyword used to define the syntax of the language.
- For examples, **if** and **else** for decision making statement and **do** and **while** for loop statement.

### reserved word for standard identifier

- Keyword used to define a specific element of the identifier.
- For examples, **int** and **char** are used for a certain data types.



- For the C++ program, all reserved words are typed in lower case.
- The reserved words for C++ program:

asm	do	inline	short	typeid
auto	double	int	signed	typename
bool	dynamic_cast	long	sizeof	union
break	else	mutable	static	unsigned
case	enum	namespace	static_cast	using
catch	explicit	new	struct	virtual
char	extern	operator	switch	void
class	false	private	template	volatile
const	float	protected	this	wchar_t
const_cast	for	public	throw	while
continue	friend	register	true	
default	goto	reinterpret_cast	try	
delete	if	return	typedef	





- Descriptions for some common reserved words in C++ program:

reserved word	Description/function
break	Terminates a switch or a loop statement.
case	Used specifically within a switch statement to specify a match for the expression.
char	Fundamental data type that defines character objects.
do	Indicates the start of a do-while statement in which the sub-statement is executed repeatedly until the value of the expression is logical-false.
else	Used specifically in an if-else statement.
float	Fundamental data type used to define a floating-point number.
for	Implies the start of a statement to achieve repetitive control.
if	Indicates the start of an if statement to achieve selective control.
int	Fundamental data type used to define integer objects.
return	Returns an object to a function's caller.
void	Absent of a type or function parameter list.
while	Start of a while statement and end of a do-while statement.



## (6) Declaration Statements

- A C++ program is made of a sequence of statements. Each statement is executed by the computer, in the sequence dictated by the program, to achieve the objective of the program.
- A statement in a C++ program is a command that the C++ running environment must execute before going to the next statement.
- A C++ program can consists of various entities such as variables, functions, types, and namespaces. Each of these entities must be declared before they can be used.
- Declaration statement introduces an entity by mentioning its type and giving it a name (an identifier). It is typically followed by a definition to allocate memory for the entity.
- In C++ program, the point at which an entity is declared is the point at which it becomes visible to the compiler. For example, a variable should be declared as close as possible before the point at which it will be used.



# Examples for declaration statements

## Example 1

```
1  #include <iostream>
2  using namespace std;
3
4  int main()
5  {
6      int myAge = 30;
7      cout << "I am " << myAge << " years old.";
8      return 0;
9  }
```

```
C:\Users\mscharizal\Documents\HCT_Aug 2019\01_Lecture Materials and Details
I am 30 years old.
-----
Process exited after 0.0343 seconds with return value 0
Press any key to continue . . .
```

One declaration on a line with an initial value  
**type identifier = InitialValue;**

## Example 2

```
1  #include <iostream>
2  using namespace std;
3
4  int main()
5  {
6      int x = 15, y = 25, z = 50;
7      cout << x + y + z;
8      return 0;
9  }
```

```
C:\Users\mscharizal\Documents\HCT_Aug 2019\0
90
-----
Process exited after 0.03292 seconds
Press any key to continue . . .
```

Multiple declarations of the same type on a line with initial values  
**type identifier<sub>1</sub> = Value<sub>1</sub>, identifier<sub>2</sub> = Value<sub>2</sub>;**





### Example 3

```
1  #include <iostream>
2  using namespace std;
3
4  int main()
5  {
6      int k = 20;
7      int l = 30;
8      int m = 20;
9      int sum = k + l + m;
10     cout << sum;
11     return 0;
12 }
```

C:\Users\mshariza\Documents\HCT\_Aug 201

70

-----

Process exited after 0.05947 second(s)

Press any key to continue . . .

#### Multiple declaration using multiple lines

*type identifier<sub>1</sub> = value<sub>1</sub>;*

*type identifier<sub>2</sub> = value<sub>2</sub>;*

*type identifier<sub>2</sub> = value<sub>2</sub>;*



## (7) Executable Statements

- An executable statement is a procedural step in programming that calls for processing action by the computer, such as reading data, performing arithmetic, or making a decision.
- Executable statements are executed in the sequence they appear in the program code.
- The difference between declaration statements and executables statements:

Declaration statements	Executable statements
Use for naming a variable, constant, or procedure, and can also specify a data type.	Use for initiating actions, such as assigning a value or expression to a variable, evaluating an expression, looping or branching for control flow, or calling a function.



## 7.1 Input-Output statements

- Storing a value into memory is called 'input operation'.
- After executing the computations, the results are stored in memory and the results can be displayed to the user by 'output operation'.
- All I/O operations are performed using input / output functions. The most common I/O functions are supplied through the preprocessor directive **#include<iostream>**.
- The most commonly used I/O functions are **cin** and **cout**.





## Example 4

## Input-Output statements

```
1  #include <iostream>
2  using namespace std;
3
4  int main()
5  {
6      int age;
7
8      cout << "Enter your age: " << endl;
9      cin >> age;
10     cout << "\nYour age is: " << age << endl;
11
12     return 0;
13 }
```

```
C:\Users\mshariza\OneDrive - Higher
Enter your age:
46
Your age is: 46
-----
Process exited after 4.606 seconds.
Press any key to continue . . .
```



## 7.2 Assignment statements

- The assignment statements store a value in a variable.
- Declaration statement must be done before assigning a value to a variable.
- The syntax is: **variable = value;**

- For examples:

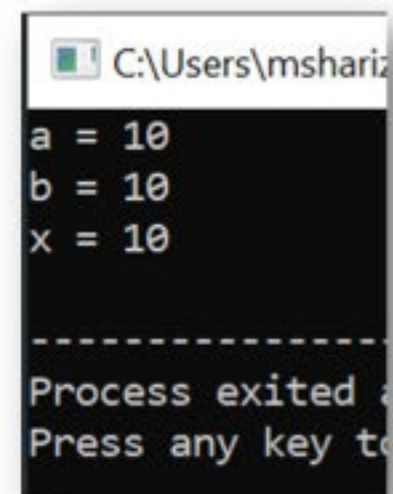
### Examples

```
int a; a = 3; ⇔ int a = 3; ⇔ int a = 3.56;  
double f = 3560.1; ⇔ double f = 3.561e3;  
char c = 'a'; ⇔ char c = 97;  
int a, b, c = 5;  
int a=b=c=5; ⇔ int a=5, b=5, c=5;
```

## Example 5

## Assignment statements

```
1  #include<iostream>
2  using namespace std;
3
4  int main()
5  {
6      int a, b=10, x;
7
8      a = b;
9      x = 10;
10
11     cout<<"a = "<<a<<endl;
12     cout<<"b = "<<b<<endl;
13     cout<<"x = "<<x<<endl;
14
15     return 0;
16 }
```



C:\Users\mshariz

```
a = 10
b = 10
x = 10

-----
Process exited with code 0
Press any key to continue . . .
```





## 7.3 Expression

- The expression store a value in a variable and is used to perform (1) arithmetic operators, (2) relational operators, and (3)logical operations in a C++ program.
- Arithmetic operators** are used to perform calculations like conventional math:
- Assume variables a and b are declared as in the table below:

Operator	Significance
+	Addition
-	Subraction
*	Multiplication
/	Division
%	Remainder

Operator	Description	Example
+	Add two operands	int a = 3, b = 6; int c = a+b; // c = 9
-	Subtract second operand from the first	int a = 9, b = 6; int c = a-b; // c = 3
*	Multiply two operands	int a = 3, b = 6; int c = a*b; // c = 18
/	Divides first operand by the second operand	int a = 12, b = 6; int c = a/b; // c = 2
%	Modulo which returns the remainder an integer division	int a = 8, b = 6; int c = a%b; // c = 2



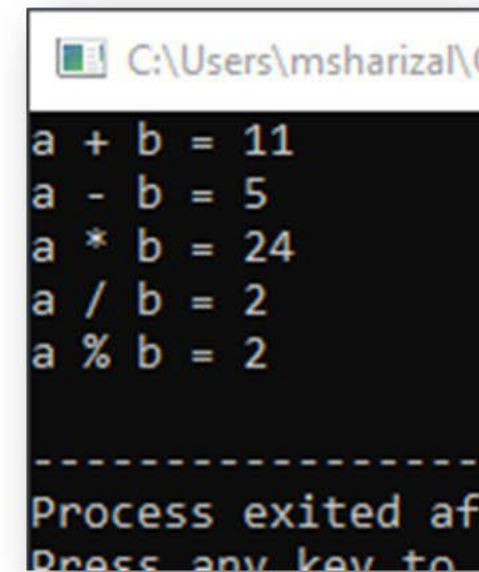


## Example 6

## Arithmetic operators



```
1  #include <iostream>
2  using namespace std;
3
4  int main()
5
6  {
7      int a = 8, b = 3;
8
9      // Addition operator
10     cout << "a + b = " << (a + b) << endl;
11
12     // Subtraction operator
13     cout << "a - b = " << (a - b) << endl;
14
15     // Multiplication operator
16     cout << "a * b = " << (a * b) << endl;
17
18     // Division operator
19     cout << "a / b = " << (a / b) << endl;
20
21     // Modulo operator
22     cout << "a % b = " << (a % b) << endl;
23
24     return 0;
25 }
```



```
C:\Users\msharizal\O
a + b = 11
a - b = 5
a * b = 24
a / b = 2
a % b = 2
-----
Process exited af
Press any key to
```

The MODULO operator(%) operator should only be used with int data type!



- **Relational operators** which also known as comparison operators are utilized to assess the connection between two values by comparison.
- A Boolean value of true or false is returned by these operators after they have evaluated the criteria. If the expression is true, it returns 1 whereas if the expression is false, it returns 0.
- There are six types of comparison operators in C++ as shown in the table below.

this expression	is true if
<code>x == y</code>	x is equal to y
<code>x != y</code>	x is not equal to y
<code>x &lt; y</code>	x is less than y
<code>x &gt; y</code>	x is greater than y
<code>x &lt;= y</code>	x is less than or equal to y
<code>x &gt;= y</code>	x is greater than or equal to y





- Examples for comparing two variables defined as int a and int b are illustrated below.

Operator	Symbol	Description	Example
Equal To	==	Checks if both operands are equal.	<pre>int a = 3, b = 6; a==b; // returns false</pre>
Greater Than	>	Checks if first operand is greater than the second operand.	<pre>int a = 3, b = 6; a&gt;b; // returns false</pre>
Greater Than or Equal To	>=	Checks if first operand is greater than or equal to the second operand.	<pre>int a = 3, b = 6; a&gt;=b; // returns false</pre>
Less Than	<	Checks if first operand is lesser than the second operand.	<pre>int a = 3, b = 6; a&lt;b; // returns true</pre>
Less Than or Equal To	<=	Checks if first operand is lesser than or equal to the second operand.	<pre>int a = 3, b = 6; a&lt;=b; // returns true</pre>
Not Equal To	!=	Checks if both operands are not equal.	<pre>int a = 3, b = 6; a!=b; // returns true</pre>



- We will now look into some examples of relational operators.

## Equal To :

```
1 #include <iostream>
2 using namespace std;
3
4 int main()
5 {
6     int x = 5;
7     int y = 3;
8     cout << (x == y);
9
10    return 0;
11 }
12 // returns 0 (false) because 5 is not equal to 3
```

0  
-----  
Process exited after 0.0519 s  
Press any key to continue . .

## Less Than or Equal To :

```
1 #include <iostream>
2 using namespace std;
3
4 int main()
5 {
6     int x = 5;
7     int y = 3;
8     cout << (x <= y);
9
10    return 0;
11 }
12 // returns 0 (false) because 5 is neither less than or equal to 3
```

0  
-----  
Process exited after 0.03424 s  
Press any key to continue . . .

## Greater Than :

```
1 #include <iostream>
2 using namespace std;
3
4 int main()
5 {
6     int x = 5;
7     int y = 3;
8     cout << (x > y);
9
10    return 0;
11 }
12 // returns 1 (true) because 5 is greater than 3
```

1  
-----  
Process exited after 0.04855 s  
Press any key to continue . .

## Not Equal To :

```
1 #include <iostream>
2 using namespace std;
3
4 int main()
5 {
6     int x = 5;
7     int y = 3;
8     cout << (x != y);
9
10    return 0;
11 }
12 // returns 1 (true) because 5 is not equal to 3
```

1  
-----  
Process exited after 0.03496 s  
Press any key to continue . . .





- **Logical operators** are used to check whether an expression is true or false. If the expression is true, it returns 1 whereas if the expression is false, it returns 0.
- The three types of logical operators in C++ are logical AND, logical OR, and logical NOT.

Operator	Symbol	Description	Example
Logical AND	&&	Returns true only if all the operands are true or non-zero.	<pre>int a = 3, b = 6; a&amp;&amp;b; // returns true</pre>
Logical OR		Returns true if either of the operands is true or non-zero.	<pre>int a = 3, b = 6; a  b; // returns true</pre>
Logical NOT	!	Returns true if the operand is false or zero.	<pre>int a = 3; !a; // returns false</pre>



- These operators let you combine many conditions and determine the truth value of each one using truth tables as described below.

### “Truth” table for logical operators

A	B	A && B	A    B
true	true	true	true
true	false	false	true
false	true	false	true
false	false	false	false

A	!A
true	false
false	true

- Some basic examples:

Example	Result
(5<2)&&(5>3)	False
(5<2)   (5>3)	True
!(5<2)	True

#### Remark:

True can be value 1 or non-zero whereas False value is 0.





- We will now look into examples with the C++ coding as given below.

### Logical AND:

```
1  #include <iostream>
2  using namespace std;
3
4  int main()
5  {
6      int x = 5;
7      int y = 3;
8      cout << (x > 3 && x < 10);
9      return 0;
10 }
11 // returns true (1) because 5 is greater than 3 AND 5 is less than 10
```

```
1
-----
Process exited after 0.04232
Press any key to continue .
```

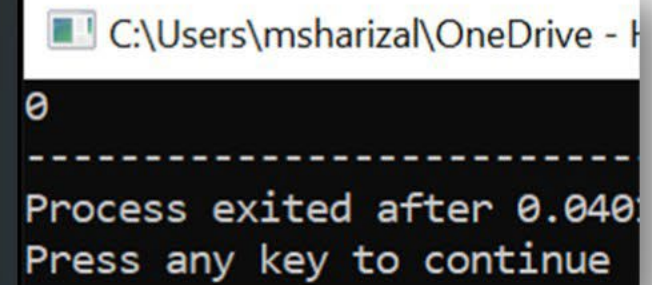
### Logical OR:

```
1  #include <iostream>
2  using namespace std;
3
4  int main()
5  {
6      int x = 5;
7      int y = 3;
8      cout << (x > 3 || x < 4);
9      return 0;
10 }
11 // returns true (1) because one of the conditions are true (5 is greater than 3, but 5 is not less than 4)
```

```
1
-----
Process exited after 0.0
Press any key to continu
```

## Logical NOT:

```
1  #include <iostream>
2  using namespace std;
3
4  int main()
5  {
6      int x = 5;
7      int y = 3;
8      cout << (!(x > 3 && x < 10));
9      return 0;
10 }
11 // returns false (0) because ! (not) is used to reverse the result
```



C:\Users\msharizal\OneDrive - H

0

-----

Process exited after 0.040

Press any key to continue



## **(7) Class Activity 2**



Matching Pairs



# Thank You