A ternary operator is of form *exp1 ? exp2 : exp3;* which is read as,

'*if exp1 is true, exp2 will be executed, or else, exp3 is executed.'*

E.g. (n%2 == 0) ? printf ("Even") : printf ("Odd");

If we try to convert if (a < b) printf ("A"); in terms we of ternary operator, we may write,

Attempt 1: (a < b) ? printf ("A") : ;

But the problem with this attempt is we cant leave the statement after : , we have to write something. So we add a statement that wont affect the functioning of our program, called 'Dummy statement'.

Attempt 2: (a < b) ? printf ("A") : printf ("");  OR

(a < b) ? printf ("A") : (z = 9);  //z is variable used only here

Note: Parentheses after : part for assignments and expressions is compulsory whereas that after ? is optional.

We are allowed to write only 1 statement after ? and ;. This is the reason ?: don't replace if-else because they can have blocks of statements.

*exp1 ? (exp2; exp3;) : (exp4; exp5;) ;* is invalid.

However, ternary operators like if-else can be nested. The trick to read nested conditional operator is to read them as 'if-else-then' expressions. We will learn this in following example:

*Any expression before ? must be placed after 'if', '?' is replaced by 'then' and the colon':' is replaced by 'else'.*

Hence,

result = i%2 == 0 ? "a" :  i%3 == 0 ? "b" :  i%5 == 0 ? "c" : "d"

is read as,

result = if i%2 == 0, then "a"

else if i %3 == 0 then "b"

else if i%5 == 0 then "c"

else "d"

The ternary operator has return type. The return type depends upon exp2, and the convertibility of exp3 into exp2 as per casting rules. If they are not compatible, compiler throws an error.

The following program compiles without error. The return type is expected to be float (as that of exp2) and exp3 (int type) is convertible to float.

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

int main()
{
	int test = 0;
	float fvalue = 3.11f;

	cout << (test ? fvalue : 0) << endl;
	return 0;
}
```

The following program will not compile, because the compiler is unable to find return type of ternary operator. The implicit conversion of 'char array' and 'int' is unavailable.

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

int main()
{
	int test = 0;
	cout << (test ? "a string" : 0) << endl;
```

```
    return 0;
}
```

Predict the output:

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

int main()
{
    int test = 0;
    cout << "first character " << '1' << endl;
    cout << "second character " << (test ? 3 : '1') << endl;
    return 0;
}
```

Output:

first character 1

second character 49

Since the return type was int as per exp2, char was converted to return type int. Hence ASCII of 1 got printed viz. 49.

Consider the following function:

```
int sumprod (int x, int y, int z, int code)
{
        code == 1 ? return (x+y+z) : return (x*y*z);
}
```

The above function produces a compilation error as return statements cannot be used in ternary operators.

So the correct way is to use operator within return statement:

```
int sumprod (int x, int y, int z, int code)
{
        return (code == 1 ? (x+y+z) : (x*y*z));
}
```