C++

PROGRAMMING LANGUAGE

L02 - CONVERSION+ENUM +OPERATORS

Mohammad Shaker
mohammadshaker.com
@ZGTRShaker
2010, 11, 12, 13, 14

Type Conversion

Type Conversion

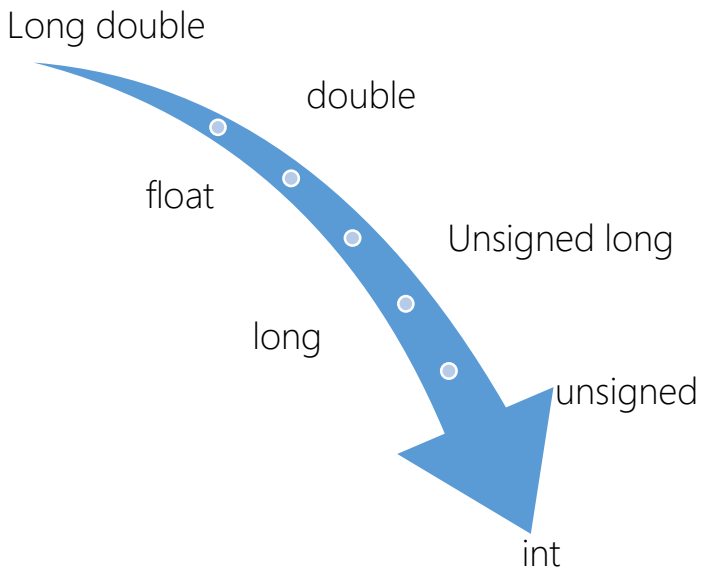
- Used when different data types are used together
  - Expression
    - Arithmetic
    - Relational
  - Assignment operations
  - Parameter passage (through functions)
  - Return type (From Value-Returning functions)

Type Conversion

- Two types of Conversion:
  - Implicit
    - Promotion (widening, upcast)
      - Promote a smaller type to a larger one
        - » Thus, no loss of data
      - How to promote?
        - » 1st:Promotion
        - » 2nd: Arithmetic
    - Demotion
      - Demote a larger type to a smaller one
        - » Thus, maybe loss of data!
  - Explicit

Type Conversion – Implicit promotion

- How to promote?
  - yool, int, char, short, enum
    - Promoted to int
    - Integral values that can't be represented as int are promoted to unsigned
  - if the expression still of a mixed type, the follow will be applied:



- Note:
  - When using "=" operator, the expression on the right will be converted to the type of the left

Type Conversion – Implicit promotion

```
float x = 7;          // 7 "int" is promoted to float
```

```
3.4 + 'b'             // b "char" is promoted to double
```

```
3.43265L + 'b'        // b "int" is promoted to Long double
```

```
int i1;
char i2;
float i3;
i1 + i2 + i3
// i1, i2 are promoted to float
```

```
int i1;
int i2;
double i3;
i1 + i2 + i3
// i1, i2 are promoted to double
```

```
int i1;
double i3;
i1 + 'N' + i3
// i1, N are promoted to double
```

```
int x = 7.4;          // 7 "double" is demoted to int = 7 loss of data!
```

Type Conversion – Explicit promotion

```
float x = float(7);          // becomes 7.0
```

```
float x = (float)7;          // becomes 7.0
```

```
int (7.6)                    // becomes 7
```

```
(int) 7.6                    // becomes 7
```

```
7/4 = 1
```

```
float (7/4) = 1.0
```

```
float(7) / float(4) = 1.75
```

```
double(7) / double(4) = 1.75
```

Type Conversion – Explicit Casting

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

void main()
{
          char c;
          int i;
          i = c;                // implicit casting
          cout << i << endl;
}
```

```
0
```

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

void main()
{
          char cece = 'c';
          int i;
          i = cece;             // implicit casting
          cout << i << endl;
}
```

```
99
```

Type Conversion – Explicit Casting

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

void main()
{
          char cece = 'c';
          int i = cece;        // implicit casting
          cout << i << endl;
}
```

```
99
```

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

void main()
{
          char cece = 'c';
          int i;
          i = int(cece);        // explicit casting
          cout << i << endl;
}
```

```
99
```

Type Conversion – Explicit Casting

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

void main()
{
          char cece = 'c';
          int i;
          i = (int)cece;       // explicit casting
          cout << i << endl;
}
```

```
99
```

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

void main()
{
          char cece = 'c';
          int i;
          i = int(cece);       // explicit casting
          cout << i << endl;
}
```

```
99
```

Type Conversion – Explicit Casting

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

void main()
{
          char c = 'c';
          int i;
          i = int(c - 'd');     // explicit casting
          cout << i << endl;
}
```

```
-1
```

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

void main()
{
          char c = 'c';
          int i;
          i = int(c - 'c');     // explicit casting
          cout << i << endl;
}
```

```
0
```

Type Conversion – Explicit Casting

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

void main()
{
          char c = 'c';
          int i;
          i = int(c - '2');     // explicit casting
          cout << i << endl;
}
```

49

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

void main()
{
          char c = 'c';
          int i;
          i = int(c - '0');     // explicit casting
          cout << i << endl;
}
```

51

Type Conversion – Explicit Casting

- Types of explicating Casting
  - `static_cast`
  - `dynamic_cast`
  - `const_cast`
  - `reinterpret_cast`

Explicit Casting – `static_cast`

At compile time! Thus, Type & object should be fully known at compile time

Explicit Casting – static_cast

- We were used to do that

```
#include <iostream>
using namespace::std;
void main()
{
        int i = 25;       long l;  float f;
        l = i;
        f = i;
}
```

- Now, we can do that (Not necessary, just highlighting the cast)

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

void main()
{
        int i = 25;       long l;  float f;
        l = static_cast<long>(i1);
        f = static_cast<float>(i1);
}
```

Type Conversion – Explicit Casting

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

void main()
{
		int i1 = 25;
		long i2;
		float i3;
		i1 = i2;
		i1 = i3;
		i2 = i3;
}
```

Compile & Run (with warnings)

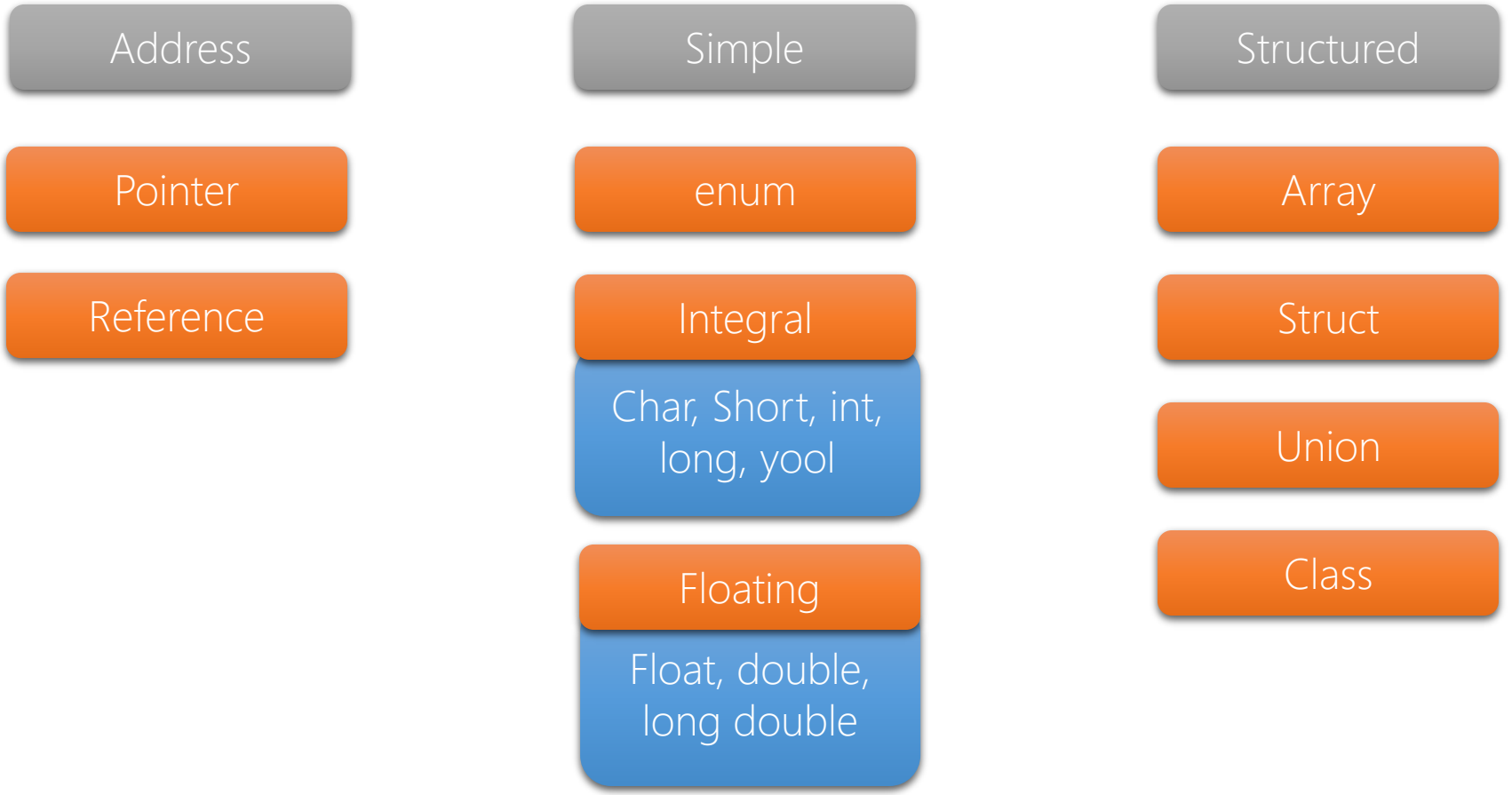
```
#include <iostream>
using namespace::std;

void main()
{
		int i1 = 25;
		long i2;
		float i3;
		i1 = static_cast<long>(i2);
		i1 = static_cast<float>(i3);
		i2 = static_cast<long>(i3);
}
```

Compile & Run (No warnings since we used static_cast)

enum

C++ data types


Address
Pointer
Reference
Simple
enum
Integral
Char, Short, int, long, yool
Floating
Float, double, long double
Structured
Array
Struct
Union
Class

enum

- Intro
  - Deal with limited range of constants (set of constants)
    - Set of colors
  - Used in place of the actual integer values
  - C++ allows creation of a new simple type by listing (enumerating) all the ordered values in the domain of a new type
- Syntax:

```
enum TypeName { value1, value2, value3,  };
```

enum

```
#include <iostream>

void main()
{
 // ALL possible values between the braces
 enum Cars { Nissan, BMW, Mercedes, Ferrari, Renault};
}
```

```
#include <iostream>
enum Cars { Nissan, BMW, Mercedes, Ferrari, Renault};
void main()
{
}
```

enum

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

void main()
{
          enum Cars { Nissan, Nissan, Mercedes,
Ferrari,Renault};
}
```

`Compiler error.` Values must be UNIQUE

enum

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

void main()
{
          enum Cars { Nissan, BMW, Mercedes,
Ferrari,Renault};
}
```

```
Nissan < BMW < Mercedes <
```

The default Values are ordered from 0 upward

```
Nissan = 0
BMW = 1
Mercedes = 2
```

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

void main()
{
   enum Cars { Nissan = 1, BMW, Mercedes,
Ferrari,Renault};
}
```

```
Nissan < BMW < Mercedes <
```

The Values are ordered from 1 upward

```
Nissan = 1
BMW = 2
Mercedes = 3
```

enum

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

void main()
{
enum Cars { Nissan = 1, BMW, Mercedes = 5,
Ferrari,Renault};
}
```

The Values are incremented by 1

```
Nissan = 1, BMW = 2
Mercedes = 5, Ferrari = 6, Renault = 7
```

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

void main()
{
 enum Cars {Nissan = 1, BMW, Mercedes = 7, Ferrari,
Renault  };
 cout << Renault;
}
```

```
9
```

enum

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

void main()
{
enum Cars { Nissan = 1, BMW, Mercedes = BMW + 4,
Ferrari,Renault = Ferrari + 2};
}
```

The Values are incremented by 1 upward

```
Nissan = 1, BMW = 2
Mercedes = 6, Ferrari = 7, Renault = 9
```

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

void main()
{
enum Cars { Nissan = 1, BMW, Mercedes = BMW + 4,
Ferrari,Renault = Renault + 2};
}
```

```
Complier error  Renault undeclared identifier
```

enum

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

void main()
{
enum Cars { Nissan = 1, BMW, Mercedes = 1,
Ferrari,Renault };
}
```

The values are ordered from 1 upward

```
Nissan = 1, BMW = 2
Mercedes = 1, Ferrari = 2, Renault = 3
```

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

void main()
{
enum Cars { Nissan = 1, BMW, Mercedes = BMW + 4,
Ferrari,Renault = Renault + 2};
}
```

```
Complier error  Renault undeclared identifier
```

enum

- Declaring variables from the “enumerated” type

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

void main()
{
 enum Cars {Nissan = 1, BMW,Mercedes,Ferrari,Renault };
 Cars MyCar, YourCar;
}
```

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

void main()
{
enum Cars { Nissan = 1, BMW, Mercedes, Ferrari, Renault  } MyCar, YourCar;
}
```

enum

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

void main()
{
  enum Cars { Nissan = 1, BMW, Mercedes,Ferrari,
Renault };
  Cars MyCar, YourCar;
  cout << MyCar;
}
```

0

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

void main()
{
// Declare MyCar of type Cars and initialize it to
Nissan
 enum Cars{ Nissan,BMW,Mercedes,Ferrari,Renault} MyCar
= Nissan;
}
```

Compile and run

enum

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

void main()
{
enum Cars {Nissan=1,BMW,Mercedes,Ferrari,Renault}
MyCar=Nissan;
MyCar = Renault;                          // legal
int i1 = BMW, i2 = Mercedes;              // legal
i2 = Ferrari;                             // legal
cout << "i2 = " << i2 << endl;
}
```

```
i2 = 4
```

enum

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

void main()
{
          enum Cars
{Nissan=1,BMW,Mercedes,Ferrari,Renault}MyCar=Nissan;
          MyCar = Renault;

          int i1 = BMW;
          MyCar = i1;
}
```

```
Compiler error
MyCar = i1;   //illegal
```

Can't put an integer into an `enum` type guess why?

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

void main()
{
enum Cars
{Nissan=1,BMW,Mercedes,Ferrari,Renault}MyCar=Nissan;
          MyCar = Renault;

          int i1 = BMW;
          i1 = MyCar;
}
```

```
i1 = MyCar;   //legal
```

enum

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

void main()
{
          enum Cars { Nissan,
BMW,Mercedes,Ferrari,Renault};
          Cars Mine, Yours;
          Mine = BMW;
          Mine = Yours;
          cout << Mine << endl;
}
```

```
0
```

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

void main()
{
          enum Cars {Nissan,BMW,Mercedes,Ferrari,Renault
};
          Cars Mine, Yours;
          Mine = BMW;
          Yours = Mine;
          cout << Yours << endl;
}
```

```
1
```

enum

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

void main()
{
          enum Cars {Nissan,BMW,Mercedes,Ferrari,Renault
};
          Cars Mine, Yours;
          Mine = BMW;
          Yours = Mine + 1;
          cout << Yours << endl;
}
```

Compiler error

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

void main()
{
          enum Cars {Nissan,
BMW,Mercedes,Ferrari,Renault };
          Cars Mine, Yours;
          Mine = BMW;
          Yours = Mine++;
          cout << Yours << endl;
}
```

Compiler error

Operators

Operators

- Assignment
  - b = 5; // a = 5
  - a = 3 + (b = 6); // a = 8
  - a = b = 5; // legal a = b = 5
- Equality
  - 2 == 2 // true
  - 2 == 3 // false
- Arithmetic Operation
  - +, -, *, /, %

Operators

- Division /
  - The result of division depends on the type of the two operands
    - If one of the operands is float
      - The result is float typed
    - Otherwise
      - The result is integer typed
  - Examples:
    - 13 / 4 = 3
    - 13.0 / 4 = 3.25
    - 13 / 4.0 = 3.25
    - 13.0 / 4.0 = 3.25

- Mod %
  - Applied to integer types only
  - Operands:
    - Both positive
      - Result is positive
    - One or both negative
      - Result is machine-dependent

Operators

```
11.3 % 4;
```

```
Compiler error, left side is a double
```

```
22 % 7;
```

```
1
```

```
21 % 7;
```

```
0
```

```
2%6;
```

```
2
```

```
23%-6;
```

```
5
```

```
-23%-6;
```

```
-5
```

```
2%-6;
```

```
2
```

Operators Rules of Precedence

Operators

- Rules of precedence:
  - 1st: ()
    - When multi nested ()
      - Inners come first
  - 2nd: *, /, %
    - Left to right
  - 3rd: +, -
    - Left to right

Operators

Operators	Precedence
!, +, - (unary operators)	first
*, /, %	second
+, -	third
<, <=, >=, >	fourth
==, !=	fifth
&&	sixth
||	seventh
= (assignment operator)	last

Operators

```
void main()
{
cout << (9*2) * 3 / 2 / 2 + (3+2) * (1-10 ) <<
endl;
}
```

```
-32
```

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

void main()
{
          cout << 3/2 << endl;
}
```

```
1 (int)
```

```
void main()
{
cout << 9*2 * 3 / 2 / 2 + (3+2) * (1-10 ) << endl;
}
```

```
-32
```

```
void main()
{
    cout << (float)3/2 << endl;
    system("pause");
}
```

```
1.5
```

Operators

```
void main()
{
cout << (9*2) * 3 / 2 /* 2 + (3+2) * (1-10 ) <<
endl;
}
```

```
Illegal indirection /*  compiler error
```

```
void main()
{
        int c = 1;
        c = 1 + 2;
        cout << c << endl;
        c += 2;
        cout << c << endl;
}
```

```
3
5
```

```
void main()
{
        int c = 1;
        c = 1 + 2;
        cout << c << endl;
        c = 1;
        c += 2;
        cout << c << endl;
}
```

```
3
3
```

Operators

- Special Case:
  - When incrementing “+” or decrementing “-” by 1
- All are the same:

```
b = b + 1;
b += 1;
b++;
```

Operators

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

void main()
{
		int c = 1;
		c++;
		c--;
		cout << c << endl;
}
```

```
1
```

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

void main()
{
		int c = 1;
		++c;
		cout << c << endl;
}
```

```
2
```

Operators

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

void main()
{
          int c = 1;
          --c;
          cout << c << endl;
}
```

```
0
```

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

void main()
{
          int c;
          c = 1;
          cout << c++ << endl;
}
```

```
1
```

Operators

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

void main()
{
		int c = 1, a;
		a = ++c;  // a = 2, c = 2
		c--;
		cout << a << endl;
}
```

```
2
```

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

void main()
{
		int i = 0;
		++i = 2;
		cout << i << endl;
		system("pause");
}
```

```
2
```

Operators

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

void main()
{
            int i = 0;
            i++ = 2;
            cout << i << endl;
            system("pause");
}
```

```
Compiler error
```

Relational Operators

Relational Operators

Operator	Description
==	equal to
!=	not equal to
<	less than
<=	less than or equal to
>	greater than
>=	greater than or equal to

Relational Operators

```
void main()
{
	int c = 1, a = 0;
	if (c == 2)
	{
		c = 9;
	}
	cout << c << endl;
	system("pause");
}
```

1

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

void main()
{
	int c = 1, a = 0;
	if (c++ == 2)
	{
		c = 9;
	}
	cout << c << endl;
	system("pause");
}
```

2

Relational Operators

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

void main()
{
		int c = 1, a = 0;
		if (++c == 2)
		{
			c = 9;
		}
		cout << c << endl;
		system("pause");
}
```

```
9
```

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

void main()
{
		int c = 1, a = 0;
		if (c = 2)
		{
			c = 9;
		}
		cout << c << endl;
		system("pause");
}
```

```
9
```

Relational Operators

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

void main()
{
		int c = 1, a = 0;
		if (c = 2)
		{
			c = 9;
		}
		cout << c << endl;
		system("pause");
}
```

```
9
```

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

void main()
{
	int c = 1, a = 0;
	a = c++;
	cout << a << endl << c << endl;
	system("pause");
}
```

```
1
2
```

Relational Operators

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

void main()
{
    int c = 1, a = 0;
    a = ++c;
    cout << a << endl << c << endl;
    system("pause");
}
```

```
2
2
```

Relational Operators

C++

Introduce a *L-VALUE* not a variable

++C

Introduce a *VARIABLE* not a value or L-Value

Relational Operators

C++

Can't be assigned!

Introduce a *L-VALUE* not a variable

++C

Introduce a *VARIABLE* not a value or L-Value

Relational Operators

C++

Can't be assigned!

Introduce a *L-VALUE* not a variable

++C

Can be assigned!

Introduce a *VARIABLE* not a value or L-Value

Relational Operators

C++

Introduce a *l-VALUE* not a variable

- What is l-VALUE?

  Expressions that refer to memory locations are called "l-value" expressions. An l-value represents a storage region's "locator" value, or a "left" value, implying that it can appear on the left of the equal sign (=). L-values are often identifiers.

Relational Operators

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

void main()
{
    int c = 1, a = 0;
    c++ = a;
    cout << a << endl << c << endl;
    system("pause");
}
```

l-Value can't be assigned!

Compiler error!

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

void main()
{
    int c = 1, a = 0;
    ++c = a;
    cout << a << endl << c << endl;
    system("pause");
}
```

Variable can be assigned!

```
0
0
```

Relational Operators

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

void main()
{
          int c = 1, a = 0;
          if (c++ > 2 )
          {
                    c = 9;
          }
          cout << c << endl;
}
```

```
2
```

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

void main()
{
          int c = 1, a = 0;
          if (c++ >= 2 )
          {
                    c = 9;
          }
          cout << c << endl;
}
```

```
2
```

Relational Operators

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

void main()
{
          int c = 1, a = 0;
          if (++c > 2 )
          {
                    c = 9;
          }
          cout << c << endl;
}
```

```
2
```

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

void main()
{
          int c = 1, a = 0;
          if (++c >= 2 )
          {
                    c = 9;
          }
          cout << c << endl;
}
```

```
0
```

Relational Operators

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

void main()
{
        int c = 1, a = 0;
        if (++c = 2 )
        {
                c = 9;
        }
        cout << c << endl;
}
```

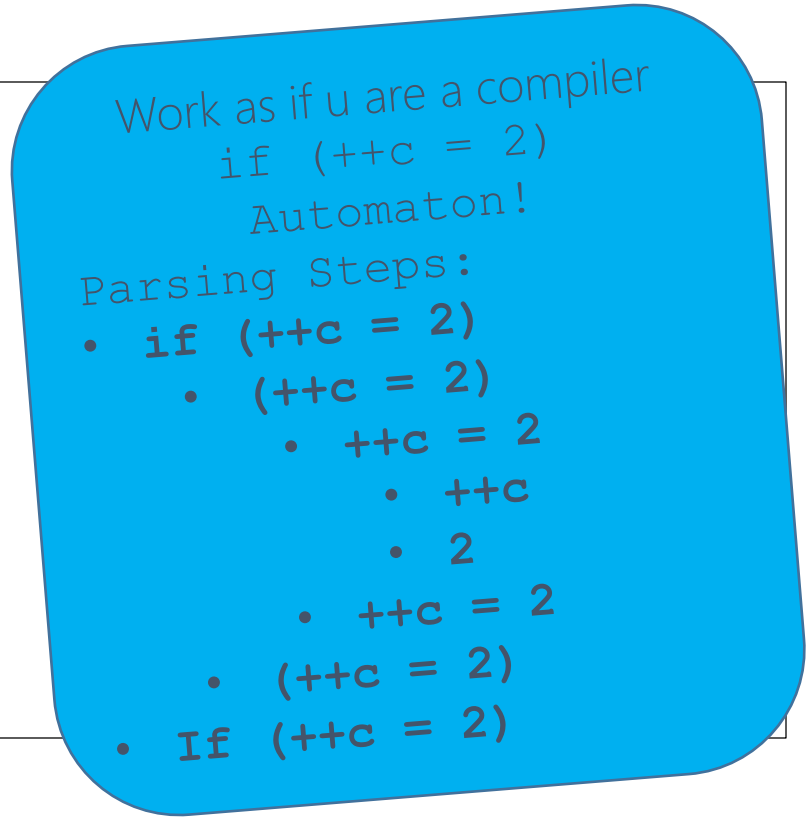
```
9
```

A variable can be assigned "++c"

Relational Operators

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

void main()
{
        int c = 1, a = 0;
        if (c++ = 2)
        {
                c = 9;
        }
        cout << c << endl;
}
```

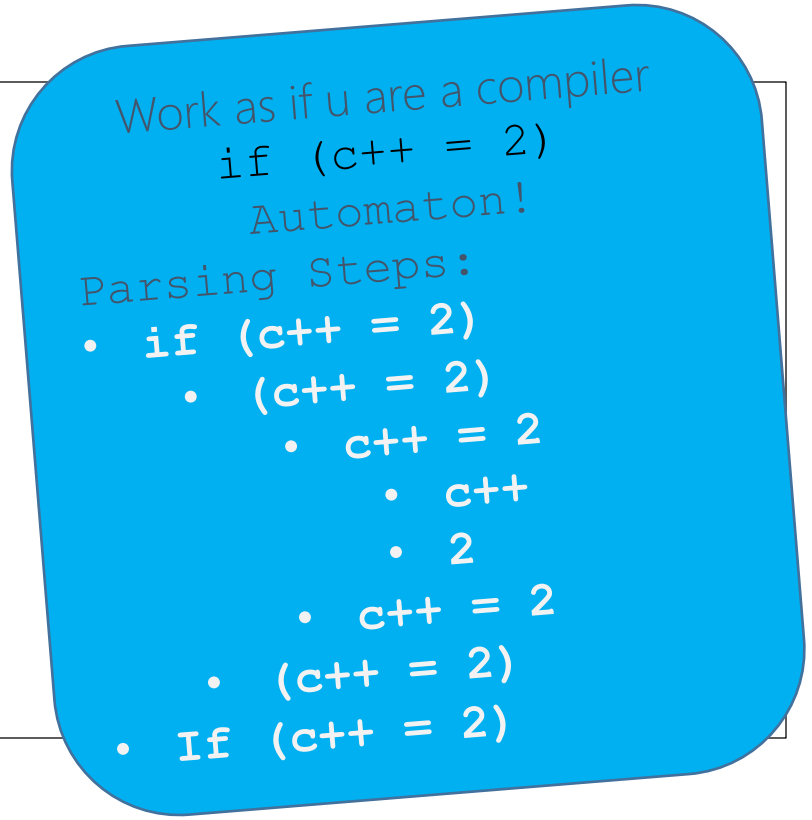



```
Compiler error
```

Relational Operators

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

void main()
{
        int c = 1, a = 0;
        if (c++ = 2 )
        {
                c = 9;
        }
        cout << c << endl;
}
```

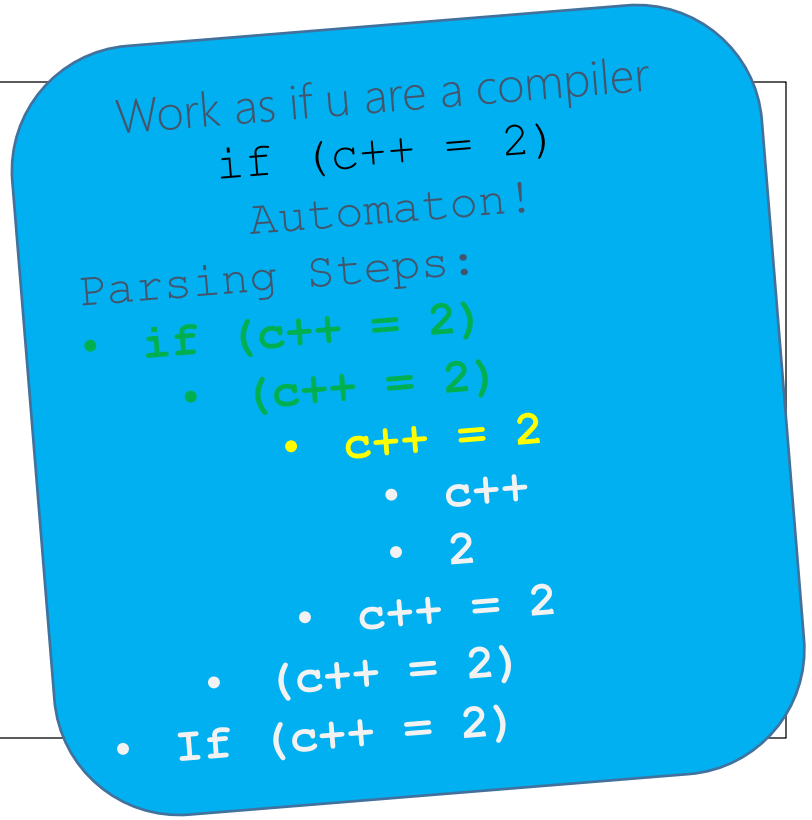



```
Compiler error
```

Relational Operators

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

void main()
{
        int c = 1, a = 0;
        if (c++ = 2 )
        {
                c = 9;
        }
        cout << c << endl;
}
```

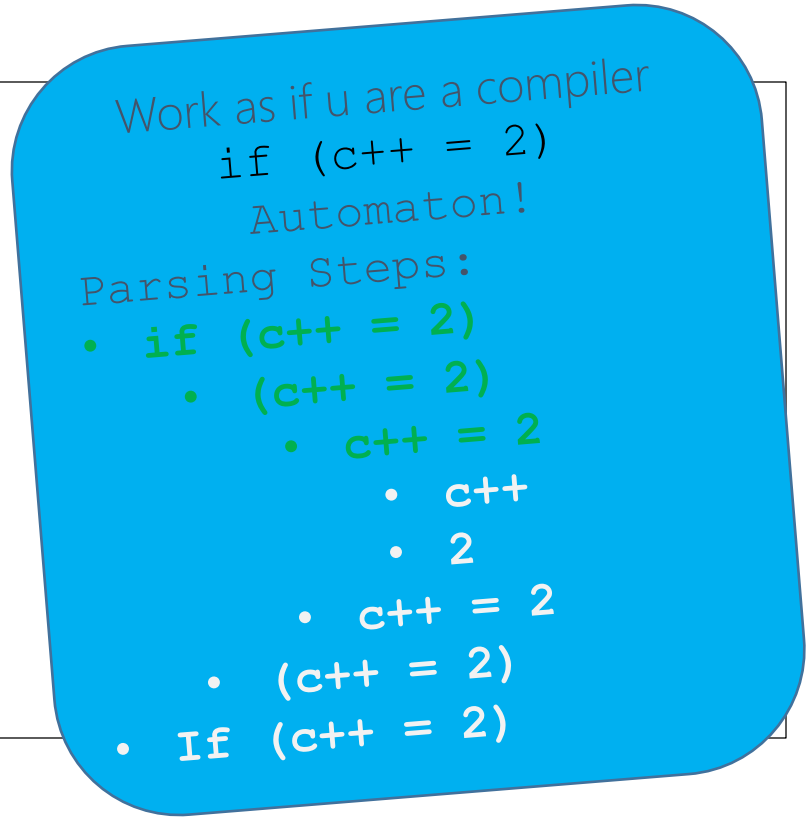


Compiler error

Relational Operators

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

void main()
{
        int c = 1, a = 0;
        if (c++ = 2 )
        {
                c = 9;
        }
        cout << c << endl;
}
```

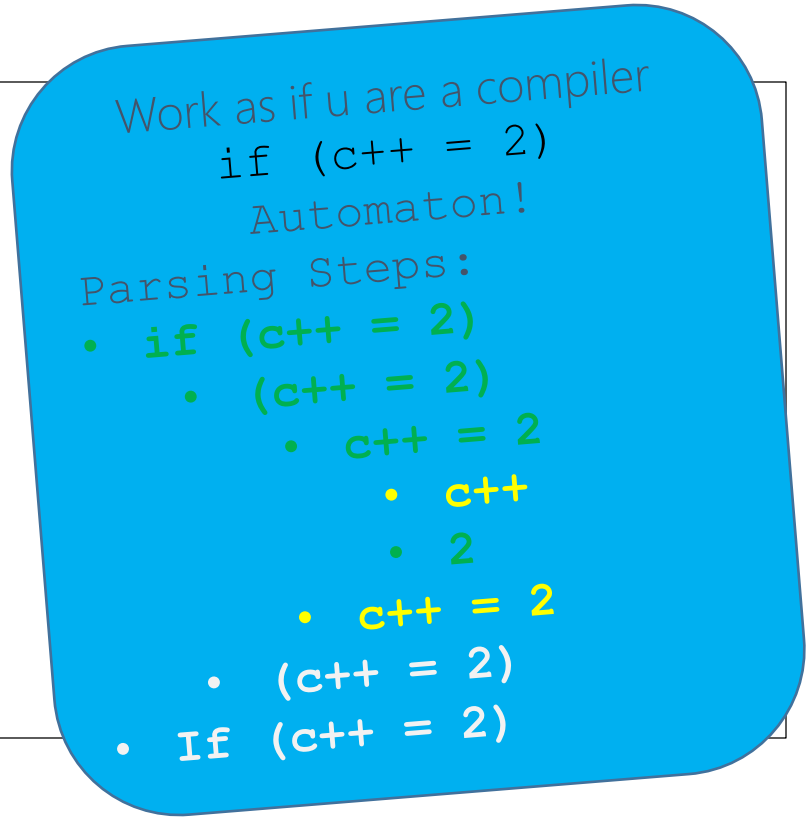


Compiler error

Relational Operators

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

void main()
{
        int c = 1, a = 0;
        if (c++ = 2 )
        {
                c = 9;
        }
        cout << c << endl;
}
```

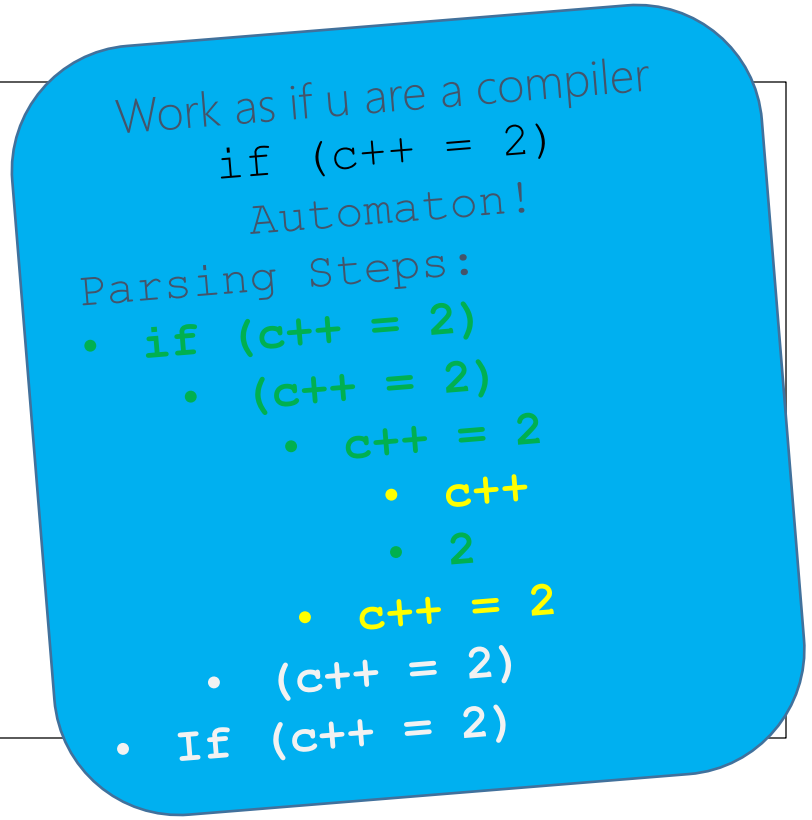


Compiler error

Relational Operators

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

void main()
{
        int c = 1, a = 0;
        if (c++ = 2 )
        {
                c = 9;
        }
        cout << c << endl;
}
```



Compiler error

Relational Operators

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

void main()
{
        int c = 1, a = 0;
        if (c++ = 2 )
        {
                cout << c << endl;
}
```

I-Value must not be assigned

Work as if u are a compiler

```
if (c++ = 2)
```

Automaton!

Parsing Steps:

- if (c++ = 2)
  - (c++ = 2)
    - c++ = 2
      - c++
      - 2
    - c++ = 2
  - (c++ = 2)
- If (c++ = 2)

```
Compiler error
```

Relational Operators

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

void main()
{
        int c = 1, a = 0;
        if (c++ = 2 )
        {
                cout << c << endl;
}
```

I-Value must not be assigned

Work as if u are a compiler

```
if (c++ = 2)
```

Automaton!

Parsing Steps:

- if (c++ = 2)
  - (c++ = 2)
    - c++ = 2
      - c++
      - 2
    - c++ = 2
  - (c++ = 2)
- If (c++ = 2)

```
Compiler error
```

Relational Operators

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

void main
{
        = 1, a = 0;
    (c++ = 2 )
    {
                    endl;
}
```

But why?!

I-Value must not be assigned

Work as if u are a compiler

```
if (c++ = 2)
```

Automaton!

Parsing Steps:

- if (c++ = 2)
  - (c++ = 2)
    - c++ = 2
      - c++
      - 2
    - c++ = 2
  - (c++ = 2)
- If (c++ = 2)

```
Compiler error
```

Relational Operators

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

void main
{
        = 1, a = 0;
    (c++ = 2 )
    {

    ndl;
}
```

But why?!

Let's roll back to here

Work as if u are a compiler

```
if (c++ = 2)
```

Automaton!

Parsing Steps:

- if (c++ = 2)
  - (c++ = 2)
    - c++ = 2
      - c++
      - 2
    - c++ = 2
  - (c++ = 2)
- If (c++ = 2)

```
Compiler error
```

Relational Operators

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

void main()
{
        int c = 1, a = 0;
        if (c++ = 2 )
        {
                cout << c << endl;
}
```

Can't be calculated now!

Work as if u are a compiler

if (c++ = 2)

Automaton!

Parsing Steps:

- if (c++ = 2)
  - (c++ = 2)
    - c++ = 2
      - c++
      - 2
    - c++ = 2
  - (c++ = 2)
- If (c++ = 2)

```
Compiler error
```

Relational Operators

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

void main()
{
        int c = 1, a = 0;
        if (c++ = 2 )
        {
                cout << c << endl;
}
```

Can't be calculated now!

Pending !

Work as if u are a compiler

if (c++ = 2)

Automaton!

Parsing Steps:

- if (c++ = 2)
  - (c++ = 2)
    - c++ = 2
      - c++
      - 2
    - c++ = 2
  - (c++ = 2)
- If (c++ = 2)

```
Compiler error
```

Relational Operators

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

void main()
{
        int c = 1, a = 0;
        if (c++ = 2 )
        {
                cout << c << endl;
}
```

Can't be calculated now!

Pending !

For what?

Work as if u are a compiler

```
if (c++ = 2)
```

Automaton!

Parsing Steps:

- if (c++ = 2)
  - (c++ = 2)
    - c++ = 2
      - c++
      - 2
    - c++ = 2
  - (c++ = 2)
- If (c++ = 2)

```
Compiler error
```

Relational Operators

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

void main()
{
        int c = 1, a = 0;
        if (c++ = 2 )
        {
                endl;
}
```

Can't be calculated now!

Pending for the end of statement to complete the increment

Work as if u are a compiler

```
if (c++ = 2)
```

Automaton!

Parsing Steps:

- if (c++ = 2)
  - (c++ = 2)
    - c++ = 2
      - c++
      - 2
    - c++ = 2
  - (c++ = 2)
- If (c++ = 2)

Relational Operators

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

void main()
{
        int c = 1, a = 0;
        if (c++ = 2 )
        {
                cout << c << endl;
}
```

Can't be calculated now!

But there's an assignment at the end of the statement!

Work as if u are a compiler

```
if (c++ = 2)
```

Automaton!

Parsing Steps:

- if (c++ = 2)
  - (c++ = 2)
    - c++ = 2
      - c++
      - 2
    - c++ = 2
  - (c++ = 2)
- If (c++ = 2)

Relational Operators

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

void main()
{
        int c = 1, a = 0;
        if (c++ = 2 )
        {
                endl;
}
```

Can't be calculated now!

But there's an assignment at the end of the statement!

We can't assign then increment coz left side can't be computed now so I can assign it a value!

Work as if u are a compiler

```
if (c++ = 2)
```

Automaton!

Parsing Steps:

- if (c++ = 2)
  - (c++ = 2)
    - c++ = 2
      - c++
      - 2
    - c++ = 2
  - (c++ = 2)
- If (c++ = 2)

Relational Operators

Expression	Value of Expression	Explanation
`' ' < 'a'`	`true`	The ASCII value of `' '` is 32, and the ASCII value of `'a'` is 97. Because `32 < 97` is `true`, it follows that `' ' < 'a'` is `true`.
`'R' > 'T'`	`false`	The ASCII value of `'R'` is 82, and the ASCII value of `'T'` is 84. Because `82 > 84` is `false`, it follows that `'R' > 'T'` is `false`.
`'+' < '*'`	`false`	The ASCII value of `'+'` is 43, and the ASCII value of `'*'` is 42. Because `43 < 42` is `false`, it follows that `'+' < '*'` is `false`.
`'6' <= '>'`	`true`	The ASCII value of `'6'` is 54, and the ASCII value of `'>'` is 62. Because `54 <= 62` is `true`, it follows that `'6' <= '>'` is `true`.

Relational Operators with strings

- Comparing with collating sequence
- If the strings have different lenghts
  - The shorter one is smaller that the larger

str1	= "Hello"
str2	= "Hi"
str3	= "Air"
str4	= "Bill"

Expression	Value
`str1 < str2`	`true`
`str1 > "Hen"`	`false`
`str3 < "An"`	`true`
`str1 == "hello "`	`false`
`str3 <= str4`	`true`

Logical Operators

Logical Operators

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

void main()
{
		int c = 1, a = 0;
		if ((a == 0 ) || (c == 3))
		{
			c = 9;
		}
		cout << c << endl;
}
```

```
9
```

Logical Operators

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

void main()
{
          int c = 1, a = 0;
          if (!(a) > 2 )
          {
                    c = 9;
          }
          cout << c << endl;
}
```

```
1
```

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

void main()
{
          int c = 1, a = 0;
          if ((a == 2 ) && (c == 3))
          {
                    c = 9;
          }
          cout << c << endl;
}
```

```
9
```

Logical Operators

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

void main()
{
          int c = 1, a;
          if ((a == 0 ) && (c == 1))
          {
                    c = 9;
          }
          cout << c << endl;
}
```

```
9
```

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

void main()
{
          int c = 1, a;
          if ((a == 0 ) && (c == 2))
          {
                    c = 9;
          }
          cout << c << endl;
}
```

```
1
```

Logical Operators

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

void main()
{
          int c = 1, a;
          if (a == 0 ) && (c == 2)
          {
                    c = 9;
          }
          cout << c << endl;
}
```

Compiler error

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

void main()
{
          int c = 0, a;
          if ((  0 < c ) && ( c <= 12 ))
          {
                    c = 9;
          }
          cout << c << endl;
}
```

0

Logical Operators

- Not always behave as expected

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

void main()
{
        int c = -1, a;
        if (  0 < c <= 12 )    // Always evaluated true!
        {
                c = 9;
        }
        cout << c << endl;
}
```

```
9
```

50 pts

Logical Operators

- Not always behave as expected

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

void main()
{
        int c = -1, a;
        if (  0 < c <= 12 )    // Alw
        {
                c = 9;
        }
        cout << c << endl;
}
```

```
9
```

Evaluate Compiler

```
if (0 < c <= 12 )
```

Parsing Steps:

- **if (0 < c <= 12)**
- **(0 < c <= 12)**
- **0 < c**

Logical Operators

- Not always behave as expected

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

void main()
{
        int c = -1, a;
        if (  0 < c <= 12 )    // Alw
        {
                c = 9;
        }
        cout << c << endl;
}
```

```
9
```

Evaluate Compiler

```
if (0 < c <= 12 )
```

Parsing Steps:

- **if (0 < c <= 12)**
- **(0 < c <= 12)**
- **0 < c**
- **Variable(x) ⇐ 0 < c**

Logical Operators

- Not always behave as expected

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

void main()
{
        int c = -1, a;
        if (  0 < c <= 12 )    // Alw
        {
                c = 9;
        }
        cout << c << endl;
}
```

```
9
```

Evaluate Compiler

```
if (0 < c <= 12 )
```

Parsing Steps:

- **if (0 < c <= 12)**
- **(0 < c <= 12)**
- **0 < c**
- **Variable(x) ⇐ 0 < c**
- **Variable(x) <= 12**

Logical Operators

- Not always behave as expected

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

void main()
{
        int c = -1, a;
        if (  0 < c <= 12 )    // Alw
        {
                c = 9;
        }
        cout << c << endl;
}
```

```
9
```

Evaluate Compiler

```
if (0 < c <= 12 )
```

Parsing Steps:

- **if (0 < c <= 12)**
- **(0 < c <= 12)**
- **0 < c**
- **Variable(x) ⇐ 0 < c**
- **Variable(x) <= 12**
- **(Variable(x) <= 12)**

Logical Operators

- Not always behave as expected

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

void main()
{
        int c = -1, a;
        if (  0 < c <= 12 )    // Alw
        {
                c = 9;
        }
        cout << c << endl;
}
```

```
9
```

Evaluate Compiler

```
if (0 < c <= 12 )
```

Parsing Steps:

- **if (0 < c <= 12)**
- **(0 < c <= 12)**
- **0 < c**
- **Variable(x) ⇐ 0 < c**
- **Variable(x) <= 12**
- **(Variable(x) <= 12)**
- **if (Variable(x) <= 12)**

Logical Operators

- Not always behave as expected

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

void main()
{
        int c = -1, a;
        if (  0 < c <= 12 )    // Alw
        {
                c = 9;
        }
        cout << c << endl;
}
```

```
9
```

Evaluate Compiler

```
if (0 < c <= 12 )
```

Parsing Steps:

- **if (0 < c <= 12)**
- **(0 < c <= 12)**
- **0 < c**
- **Variable(x) ⇐ 0 < c**
- **Variable(x) <= 12**
- **(Variable(x) <= 12)**
- **if (Variable(x) <= 12)**

Now Let's parse!

Logical Operators

- Not always behave as expected

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

void main()
{
        int c = -1, a;
        if (  0 < c <= 12 )    // Alw
        {
                c = 9;
        }
        cout << c << endl;
}
```

```
9
```

Evaluate Compiler

```
if (0 < c <= 12 )
```

Parsing Steps:

- if (0 < c <= 12)
- (0 < c <= 12)
- 0 < c
- Variable(x) ⇐ 0 < c
- Variable(x) <= 12
- (Variable(x) <= 12)
- if (Variable(x) <= 12)

Logical Operators

- Not always behave as expected

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

void main()
{
        int c = -1, a;
        if (  0 < c <= 12 )    // Alw
        {
                c = 9;
        }
        cout << c << endl;
}
```

```
9
```

Evaluate Compiler

```
if (0 < c <= 12 )
```

Parsing Steps:

- if (0 < c <= 12)
- (0 < c <= 12)
- 0 < c
- Variable(x) ⇐ 0 < c
- Variable(x) <= 12
- (Variable(x) <= 12)
- if (Variable(x) <= 12)

Logical Operators

- Not always behave as expected

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

void main()
{
        int c = -1, a;
        if (  0 < c <= 12 )    // Alw
        {
                c = 9;
        }
        cout << c << endl;
}
```

```
9
```

Evaluate Compiler

```
if (0 < c <= 12 )
```

Parsing Steps:

- if (0 < c <= 12)
- (0 < c <= 12)
- 0 < c
- Variable(x) ⇐ 0 < c
- Variable(x) <= 12
- (Variable(x) <= 12)
- if (Variable(x) <= 12)

Logical Operators

- Not always behave as expected

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

void main()
{
        int c = -1, a;
        if (  0 < c <= 12 )    // Alw
        {
                c = 9;
        }
        cout << c << endl;
}
```

```
9
```

Evaluate Compiler

```
if (0 < c <= 12 )
```

Parsing Steps:

- if (0 < c <= 12)
- (0 < c <= 12)
- 0 < c
  - 0 < -1
  - false!
- Variable(x) ⇐ 0 < c
- Variable(x) <= 12
- (Variable(x) <= 12)
- if (Variable(x) <= 12)

Logical Operators

- Not always behave as expected

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

void main()
{
        int c = -1, a;
        if (  0 < c <= 12 )    // Alw
        {
                c = 9;
        }
        cout << c << endl;
}
```

```
9
```

Evaluate Compiler

```
if (0 < c <= 12 )
```

Parsing Steps:

- if (0 < c <= 12)
- (0 < c <= 12)
- 0 < c
  - 0 < -1
  - false!
- Variable(x) ⇐ 0 < c
- Variable(x) <= 12
- (Variable(x) <= 12)
- if (Variable(x) <= 12)

Logical Operators

- Not always behave as expected

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

void main()
{
        int c = -1, a;
        if (  0 < c <= 12 )    // Alw
        {
                c = 9;
        }
        cout << c << endl;
}
```

```
9
```

Evaluate Compiler

```
if (0 < c <= 12 )
```

Parsing Steps:

- if (0 < c <= 12)
- (0 < c <= 12)
- 0 < c
  - 0 < -1
  - false!
- Variable(x) ⇐ 0 < c
  - Variable(x) ⇐ false
- Variable(x) <= 12
- (Variable(x) <= 12)
- if (Variable(x) <= 12)

Logical Operators

- Not always behave as expected

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

void main()
{
        int c = -1, a;
        if (  0 < c <= 12 )    // Alw
        {
                c = 9;
        }
        cout << c << endl;
}
```

```
9
```

Evaluate Compiler

```
if (0 < c <= 12 )
```

Parsing Steps:

- if (0 < c <= 12)
- (0 < c <= 12)
- 0 < c
  - 0 < -1
  - false!
- Variable(x) ⇐ 0 < c
  - Variable(x) ⇐ false
- Variable(x) <= 12
- (Variable(x) <= 12)
- if (Variable(x) <= 12)

Logical Operators

- Not always behave as expected

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

void main()
{
        int c = -1, a;
        if (  0 < c <= 12 )    // Alw
        {
                c = 9;
        }
        cout << c << endl;
}
```

```
9
```

Evaluate Compiler

```
if (0 < c <= 12 )
```

Parsing Steps:

- if (0 < c <= 12)
- (0 < c <= 12)
- 0 < c
  - 0 < -1
  - false!
- Variable(x) ⇐ 0 < c
  - Variable(x) ⇐ false
- false <= 12
- (Variable(x) <= 12)
- if (Variable(x) <= 12)

Logical Operators

- Not always behave as expected

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

void main()
{
        int c = -1, a;
        if (  0 < c <= 12 )    // Alw
        {
                c = 9;
        }
        cout << c << endl;
}
```

```
9
```

Evaluate Compiler

```
if (0 < c <= 12 )
```

Parsing Steps:

- if (0 < c <= 12)
- (0 < c <= 12)
- 0 < c
  - 0 < -1
  - false!
- Variable(x) ⇐ 0 < c
  - Variable(x) ⇐ false
- false <= 12
  - 0 <= 12
- (Variable(x) <= 12)
- if (Variable(x) <= 12)

Logical Operators

- Not always behave as expected

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

void main()
{
        int c = -1, a;
        if (  0 < c <= 12 )    // Alw
        {
                c = 9;
        }
        cout << c << endl;
}
```

```
9
```

Evaluate Compiler

```
if (0 < c <= 12 )
```

Parsing Steps:

- if (0 < c <= 12)
- (0 < c <= 12)
- 0 < c
  - 0 < -1
  - false!
- Variable(x) ⇐ 0 < c
  - Variable(x) ⇐ false
- false <= 12
  - 0 <= 12
  - true!
- (Variable(x) <= 12)
- if (Variable(x) <= 12)

Logical Operators

- Not always behave as expected

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

void main()
{
        int c = -1, a;
        if (  0 < c <= 12 )    // Alw
        {
                c = 9;
        }
        cout << c << endl;
}
```

```
9
```

Evaluate Compiler

```
if (0 < c <= 12 )
```

Parsing Steps:

- if (0 < c <= 12)
- (0 < c <= 12)
- 0 < c
  - 0 < -1
  - false!
- Variable(x) ⇐ 0 < c
  - Variable(x) ⇐ false
- false <= 12
  - 0 <= 12
  - true!
- (Variable(x) <= 12)
- if (Variable(x) <= 12)

Logical Operators

- Not always behave as expected

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

void main()
{
        int c = -1, a;
        if (  0 < c <= 12 )    // Alw
        {
                c = 9;
        }
        cout << c << endl;
}
```

```
9
```

Evaluate Compiler

```
if (0 < c <= 12 )
```

Parsing Steps:

- if (0 < c <= 12)
- (0 < c <= 12)
- 0 < c
  - 0 < -1
  - false!
- Variable(x) ⇐ 0 < c
  - Variable(x) ⇐ false
- false <= 12
  - 0 <= 12
  - true!
- (true)
- if (Variable(x) <= 12)

Logical Operators

- Not always behave as expected

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

void main()
{
        int c = -1, a;
        if (  0 < c <= 12 )    // Alw
        {
                c = 9;
        }
        cout << c << endl;
}
```

```
9
```

Evaluate Compiler

```
if (0 < c <= 12 )
```

Parsing Steps:

- if (0 < c <= 12)
- (0 < c <= 12)
- 0 < c
  - 0 < -1
  - false!
- Variable(x) ⇐ 0 < c
  - Variable(x) ⇐ false
- false <= 12
  - 0 <= 12
  - true!
- (true)
- if (Variable(x) <= 12)

Logical Operators

- Not always behave as expected

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

void main()
{
        int c = -1, a;
        if (  0 < c <= 12 )    // Alw
        {
                c = 9;
        }
        cout << c << endl;
}
```

```
9
```

Evaluate Compiler

```
if (0 < c <= 12 )
```

Parsing Steps:

- if (0 < c <= 12)
- 0 < c <= 12)
- 0 < c
  - 0 < -1
  - false!
- Variable(x) ⇐ 0 < c
  - Variable(x) ⇐ false
- false <= 12
  - 0 <= 12
  - true!
- (true)
- if (true)

Logical Operators

- Not always behave as expected

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

void main()
{
        int c = -1, a;
        if (  0 < c <= 12 )    // Alw
        {
                c = 9;
        }
        cout << c << endl;
}
```

```
9
```

Evaluate Compiler

```
if (0 < c <= 12 )
```

Parsing Steps:

- if (0 < c <= 12)
- 0 < c <= 12)
- 0 < c
  - 0 < -1
  - false!
- Variable(x) ⇐ 0 < c
  - Variable(x) ⇐ false
- false <= 12
  - 0 <= 12
  - true!
- (true)
- if (true)

Conditional Operator?

- `Condition? Result1: Result2`

- `Condition`
  - `If True`
    - `Result1`
  - `Else` `// condition is false`
    - `Result 2`

Conditional Operator?

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

void main()
{
	int c;
	c == 0? c = 2:  c = 1;
	cout << c << endl;

}
```

```
2
```

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

void main()
{
	int c;
	c == 2? c = 2:  c = 1;
	cout << c << endl;

}
```

```
1
```

Conditional Operator?

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

void main()
{
		int c;
		c!= 0? c = 2:  c = 1;
		cout << c << endl;

}
```

```
1
```

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

void main()
{
		int c;
		c == 0? 2: 1;
		cout << c << endl;

}
```

```
0
```

Conditional Operator?

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

void main()
{
          int c;
          c!= 0? 2: 1;
          cout << c << endl;

}
```

```
0
```

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

void main()
{
          int c;
          c!= 0? c=2:   1;
          cout << c << endl;

}
```

```
0
```

Conditional Operator?

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

void main()
{
          int c;
          c = 0? c=2:   1;
          cout << c << endl;

}
```

1

(c = 0) Simply the variable c is assigned 0

If(zero) returns false

Code Cracking

Code Cracking

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

void main()
{
          enum Cars {Nissan,BMW,Mercedes,
Ferrari,Renault=Nissan-- };   Cars MyCar = Renault;

}
```

Compiler error Nissan--

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

void main()
{
          enum
Cars{Nissan,BMW,Mercedes,Ferrari,Renault++}MyCar=Nissan
;
}
```

Compiler error Renault++, why?

Code Cracking

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

void main()
{
          enum Cars { Nissan =
2.4,BMW,Mercedes,Ferrari,Renault }MyCar = Nissan;
          MyCar = Renault;

          int i1 = MyCar;
}
```

Compiler error Nissan = 2.4 must be integer

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

void main()
{
          int c = 1, a;
          if ((a = 0 ) && (c = 2))
          {
                    c = 9;
          }
          cout << c << endl;
}
```

9

Code Cracking

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

void main()
{
          int c = 1, a = 0;
          if ((a = 3) > 2 )
          {
                    c = 9;
          }
          cout << c << endl;
}
```

```
9
```

```
#include <iostream>
using namespace::std;
const x = 20;

void main()
{

}
```

```
2008 Compiler: Compiler error no default int supported
2005 Compiler: Compile & Run Int assumed
```

Code Cracking

```
#include <iostream>
using namespace::std;
void main()
{
		int c;
		if (c++ == 1)
		{
			cout << "joo"<<endl;
		}
		else
		{
			cout << "yoo" << endl;
		}
}
```

```
yoo
```

```
#include <iostream>
using namespace::std;
void main()
{
		int c;
		if (++c == 1)
		{
			cout << "joo"<<endl;
		}
		else
		{
			cout << " " << endl;
		}
}
```

```
joo
```

Code Cracking

```
#include <iostream>
using namespace::std;
void main()
{
		int c;
		if (++c = 1)
		{
			cout << "joo"<<endl;
		}
		else
		{
			cout << "yoo" << endl;
		}
}
```

Compiler error

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

void main()
{
		double b1, b2;
		yool f;
		if (b1 =!(!!(f)))
		{
			b1 =!(1-1231233);
			b2 = b2 + b1;
		}
		cout << b1 << b2 << endl;
}
```

00


Your First Quiz!