Assignment statements, arithematic expressions

- In C++, the assignment operator (=) works by assigning left part the value of the right part and not the vice versa.

- Multiplication and Division have the precedence over Addition and Subtraction (which have equal precedence amongst themselves)- So we can say C++ follows B,O,D=M=%,A=S instead of BODMAS and O doesn't exist because of course it doesn't

  Operators of same precedence would be evaluated from left to right

- While using division, if both the operands are integers, C++ will just give the integral quotient as the answer and ignore the remainder

  Example of the above statement:

  ```
  int b=12, c=5;
  double a=b/c;
  cout<<a<<endl;
  /*This will actually print 2.0 instead of 2.4 as although a is double, C++ don't
  no fucks*/
  //In order to get a non-integral answer, we use explicit typecasting (?)
  double d=(double)b/c;
  cout<<d; //This prints 2.4
  ```

- C++ tries its best to fit things in properly

  ```
  int a=2.7;
  float b=123456789;
  cout<<a<<endl<<b;
  /*This will print-
  2
  123456780
  Reason- int only stores the integral part of the operand
  float can only store numbers as large as that in 24 bits so it stores only 1.2345678e8 or something like that (not  100% sure if it wil
  */
  ```

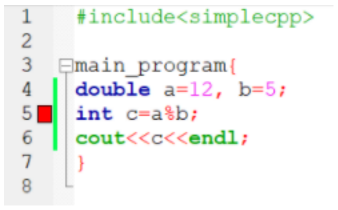
- You go for implicit typecasting on the basis of storage space. Thus a long int while in an operation with an int will give result in long int.

```
int a=27;
double b=a/100; //This stores b=0.0
double c=a/100.0; //This stores c=0.27
```

- C++ also suffers from limited precision

```
float a, b=1.5;
double c=6.023e23
a=c+b; //a will actually store 6.023e23 and not the actual sum as float has only 7 digits in significand
```

- Modulo (%) is hella sexy. It can find the remainder when first operand is divided by second operand. However, both operands need to be integers.



```
#include<simplecpp>

main_program{
double a=12, b=5;
int c=a%b;
cout<<c<<endl;
}
```