```cpp
//////// Lecture 67 - Type Conversion I (Basic Type to Basic Type)
int main(){
    int a = 5, b = 2;

    // Implicit Type Conversion
    float f = a;
    // The compiler will apply the cast implicitly, but the implicit cast may not be correct in all the cases.
    // What if we divide this variable with b?
    float f = a/b; // you expect the value in the float variable to be 2.5, but it is actually 2.00.
    // The result is 2.
    // the reason for the incorrect value is that a is of type integer, b is also of type integer, neither of the integers will be promoted to floating point numbers, and the decimal portion of the number is lost. So we have to manually apply a cast on the variable a.

    // You can apply the cast as a C-style cast, like this.
    float f = (float)a/b;
    // C-style cast are discouraged in C++, because they don't check for the validness of the cast. That is why C++ provides a casting operator called as 'static_cast'. Static_Cast is superior to C style casting because it checks if the cast is valid or not.

    // Do this instead:
    float f = static_cast<float>(a)/b;

    // Why is static_cast preferred in C++?
    // We can create a pointer to the integer variable:
    // But wait - the type of pointer will be a character pointer.
    int d;
    char *p = &d; // The compiler will not allow this conversion because the types are different - if
```

```
    you try to build this, there is an error -
    cannot convert from int* to char*.

// But we can apply a C-style cast to convert and
    the compiler will then compile the code without
    any errors.
char *p = (char*) &a;
// This will compile without any errors. This type
    of casting is not right, because you're trying
    to cast between different pointer types.
// If we had used static_cast in place of the C
    cast,
char *p = static_cast<char*>(&a);
// If we build this, it does not allow the cast
    because pointer types are not convertible. So
    static_Cast does not allow us to convert int* to
    char*.

// REINTERPRET CAST
// but if you know what you're doing and you want
    to store the address of this integer variable
    into the character pointer then you can make
    this cast in C++ using the operator:
char *p = reinterpret_cast<char*> (&a);

// A reinterpret-cast allows casting between
    different types. Even if the types are not
    related. A C style cast would have worked in its
    place, but the advantage of reinterpret cast is
    that if there are any qualifiers on the source
    then they are not discarded.  MEANING?

// C style casts will discard the qualifiers.
    This means if we have a constant integer and we
    try to create a pointer to this,
const int x = 1;
int* p = &x;  // this doesn't work.
// If we use a C style cast, this will compile
    fine, but this would be a source of bugs.
int *p = (int*)&x; // this works, but it shouldn't

// So a Cstyle cast discards the qualifiers, but
```

```
        if we had used a reinterpret cast, it does not
        discard the qualifiers, and this will NOT ALLOW
        the cast to be performed.
    int *p = reinterpret_cast<int*> (&x); // this
        doesn't work!

    // CONST CAST
    // we can make this work by using a const cast.
    int *p = const_cast<int*> (&x); // correct!
}
// In summary, C style casts should be avoided.
    Instead, we should use the casting operators. In
    general, casting between types should be avoided.
    One more advantage of using reinterpret cast over
    Cstyle cast is that it is easy to search for places
    that uses 'reinterpret_cast'. all these casting
    operators work at COMPILE TIME.

```