C++ casting

Score: 4.1/5 (92 votes)

Casting is a conversion process wherein data can be changed from one type to another. C++ has two types of conversions:

Implicit conversion: Conversions are performed automatically by the compiler without the programmer's intervention.

ex.
```
int iVariable = 10;
    float fVariable = iVariable; //Assigning an int to a float will trigger a conversion.
```

Explicit conversion: Conversions are performed only when explicitly specified by the programmer.

ex.
```
int iVariable = 20;
    float fVariable = (float) iVariable / 10;
```

In C++, there are four types of casting operators.
```
- static_cast
- const_cast
- reinterpret_cast
- dynamic_cast
```

In this article we will only be looking into the first three casting operators as dynamic_cast is very different and is almost exclusively used for handling polymorphism only which we will not be addressing in this article.

static_cast
Format:
static_cast<type>(expression);
ex.
```
float fVariable = static_cast<float>(iVariable); /*This statement converts iVariable which is of type int to float. */
```

By glancing at the line of code above, you will immediately determine the purpose of the cast as it is very explicit. The static_cast tells the compiler to attempt to convert between two different data types. It will convert between built-in types, even when there is a loss of precision. In addition, the static_cast operator can also convert between **related**pointer types.

ex.
```
int* pToInt = &iVariable;
    float* pToFloat = &fVariable;

    float* pResult = static_cast<float*>(pToInt); //Will not work as the pointers are not related (they are of different types).
```

const_cast
Format:
const_cast<type>(expression);
ex.
```
void aFunction(int* a)
{
    cout << *a << endl;
}

int main()
{
    int a = 10;
    const int* iVariable = &a;

    aFunction(const_cast<int*>(iVariable));
/*Since the function designer did not specify the parameter as const int*, we can strip the const-ness of the pointer iVariable to pass it into the function.
Make sure that the function will not modify the value. */

    return 0;
}
```

Probably one of the most least used cast, the const_cast does not cast between different types. Instead it changes the "const-ness" of the expression. It can make something const what was not const before, or it can make something volatile/changeable by getting rid of the const. Generally speaking, you will not want to utilise this particular cast in your programs. If you find yourself using this cast, you should stop and rethink your design.

reinterpret_cast
Format:
reinterpret_cast<type>(expression);

Arguably one of the most powerful cast, the reinterpret_cast can convert from any built-in type to any other, and from any pointer type to another pointer type. However, it cannot strip a variable's const-ness or volatile-ness. It can however convert between built in data types and pointers without any regard to type safety or const-ness. This particular cast operator should be used only when absolutely necessary.

Hopefully this article was helpful to anyone whose struggling to grasp the theory of casting.

Happy programming.