CSE 333  Section 7 - Casting
Welcome back to section! We’re glad that you’re here :) 

Casting in C++
While in C++, we want to use casts that are more explicit in their behaviour. This gives us a better understanding of what happens when we read our code, because C-style casts can do many (sometimes unwanted) things. There are four types of casts we will use in C++:

\[
\text{static\_cast<to\_type>(expression);} \\
\begin{align*}
\text{★ Convert} & \text{s between pointers of related types.} \\
\text{○ Compiler error if not related.} \\
\text{★ Performs not pointer conversion (e.g. float to int conversion).}
\end{align*}
\]

\[
\text{dynamic\_cast<to\_type>(expression);} \\
\begin{align*}
\text{★ Convert} & \text{s between pointers of related types.} \\
\text{○ Compiler error if not related.} \\
\text{○ Also checks at runtime to make sure it is a `safe` conversion (returns nullptr if not).}
\end{align*}
\]

\[
\text{const\_cast<to\_type>(expression);} \\
\begin{align*}
\text{★ Used to add or remove const-ness.}
\end{align*}
\]

\[
\text{reinterpret\_cast<to\_type>(expression);} \\
\begin{align*}
\text{★ Casts between incompatible types \textit{without changing the data}.} \\
\text{○ The types you are casting to and from must be the same size.} \\
\text{○ Will not let you convert between integer and floating point types.}
\end{align*}
\]

Exercise 1
For each of the following snippets of code, fill in the blank with the most appropriate C++ style cast. Assume that we have the following classes defined:

```cpp
class Base {
public:
    int x;
};
class Derived : public Base {
public:
    int y;
};
```

```
int64_t x = 0x7fffffffffe870;
char* str = ____________________________(x);
```

```
void foo(Base *b) {
    Derived *d = _________________________(b);
    // additional code omitted
}
```

```
Derived *d = new Derived;
Base *b = ______________________________(d);
```

```
double x = 64.382;
int64_t y = ____________________________(x);
```