Cast operators

C++ supports four new cast operators
- static_cast
- const_cast
- dynamic_cast
- reinterpret_cast

In C, we typecast using the following statement:
(type) expression

In C++, we would write the same as
static_cast <type> (expression)
For example, suppose we want to cast an int to double, using C-style casts, the operation would be as follows:

```
int num1, num2;
double result = ((double)num1) / num2;
```

In C++, it would be written as

```
double result = static_cast <double> (num1) / num2;
```

static_cast has the same power and meaning as the general purpose C-style cast. It has the same kind of restrictions. For example, we can't cast a struct into a double or a double into a pointer using a static_cast.

const_cast is used to cast away the constness or volatileness of an expression. The only thing the const_cast allows us to change is the constness or volatileness of something.
Imagine we have a function init() which takes a non-const object. To this function if we pass a const object it would result in an error. To overcome this problem, we change the constness of the object.

```
void init(account *pa);
const account a1;
```

// Error. Can't pass a const object to a function that takes a non-const object.

```
init(&a1);
```

To overcome this problem, we write the same as

```
init(const_cast <account *> (&a1));
```

dynamic_cast is primarily used to perform safe casts down or across the inheritance hierarchy. It is used to cast pointers or references to base class objects into pointers or references to derived or sibling class in such a way that you can determine whether the casts succeeded.

```
class shape {}
class circle : public shape {}
shape *sp;
```

/* We are passing to draw(), a pointer to circle sp. If sp points to one it is fine, otherwise null pointer is passed. */

```
draw(dynamic_cast <circle *>(sp));
```

dynamic_cast are useful for navigating inheritance hierarchy. They cannot be applied to types lacking virtual functions, nor can they cast away constness.

```
int num1, num2;

// Error, as no inheritance is involved
double result = dynamic_cast <double> (num1) / num2;
const account a1;

// Error, as dynamic_cast can't cast away constness
init(dynamic_cast <account *> (&a1));
```

reinterpret_cast is used to perform type conversions whose result is nearly always implementation defined, hence they are rarely portable. It is commonly used to cast between function pointer types. For example, if we have an array of pointers to functions of a particular type:

```
// fp is a pointer to a function, which has no arguments and returns void
typedef void (*fp)();
fp fparray[5];

Assume a function test(), which is defined as follows:
int test();

If we perform the following operation, we get an error - type mismatch
fparray[0] = &test;

Using reinterpret_cast we can solve this problem.

fparray[0] = reinterpret_cast <fparray> (&test);
```