```
// C++ program to illustrate the reinterpret_cast
#include <iostream>
using namespace std;

int main()
{
	int number = 10;
	// Store the address of number in numberPointer
	int* numberPointer = &number;

	// Reinterpreting the pointer as a char pointer
	char* charPointer
		= reinterpret_cast<char*>(numberPointer);

	// Printing the memory addresses and values
	cout << "Integer Address: " << numberPointer << endl;
	cout << "Char Address: "
		<< reinterpret_cast<void*>(charPointer) << endl;

	return 0;
}


// C++ program to illustrate the static_cast
#include <iostream>
#include <typeinfo>
using namespace std;

int main()
{
```

```
	int num = 10;

	// converting int to double
	double numDouble = static_cast<double>(num);

	// printing data type
	cout << typeid(num).name() << endl;

	// typecasting
	cout << typeid(static_cast<double>(num)).name() << endl;

	// printing double type t
	cout << typeid(numDouble).name() << endl;

	return 0;
}
```

```
// C++ program to illustrate the dynamic_cast
#include <iostream>
using namespace std;

// Base Class
class Animal {
public:
	virtual void speak() const
	{
		cout << "Animal speaks." << endl;
	}
};
```

```
// Derived Class
class Dog : public Animal {
public:
	void speak() const override
	{
		cout << "Dog barks." << endl;
	}
};

// Derived Class
class Cat : public Animal {
public:
	void speak() const override
	{
		cout << "Cat meows." << endl;
	}
};

int main()
{
	// base class pointer to derived class object
	Animal* animalPtr = new Dog();

	// downcasting
	Dog* dogPtr = dynamic_cast<Dog*>(animalPtr);

	// checking if the typecasting is successfull
	if (dogPtr) {
		dogPtr->speak();
	}
```

```
	else {
		cout << "Failed to cast to Dog." << endl;
	}

	// typecasting to other dervied class
	Cat* catPtr = dynamic_cast<Cat*>(animalPtr);
	if (catPtr) {
		catPtr->speak();
	}
	else {
		cout << "Failed to cast to Cat." << endl;
	}

	delete animalPtr;
	return 0;
}


// C++ program to illustrate the const_cast
#include <iostream>
using namespace std;

int main()
{

	const int number = 5;
	// Pointer to a const int
	const int* ptr = &number;

	// int* nonConstPtr = ptr; if we use this
	// instead of without using const_cast
```

```
    // we will get error of invalid conversion
    int* nonConstPtr = const_cast<int*>(ptr);
    *nonConstPtr = 10;

    cout << "Modified number: " << *nonConstPtr;

    return 0;
}
```