Here's a structured table that lists **types of typecasting** (rows) and their key characteristics (columns):

Type	Purpose	Syntax	Usage	Scope
Implicit Casting	Automatic conversion to a compatible type (type promotion).	*Done by the compiler automatically.*	Promotes smaller to larger types (e.g., int to float).	General use, safe.
Static Cast	Compile-time checked conversion between related types.	static_cast<Type>(value)	Numeric conversions, upcasting in inheritance.	Limited to valid types.
Dynamic Cast	Runtime-checked conversion for polymorphic types.	dynamic_cast<Type>(value)	Safe downcasting (base to derived class).	Polymorphic hierarchy.
Const Cast	Adds/removes const or volatile qualifiers.	const_cast<Type>(value)	Modifying a const object when necessary.	Limited to qualifiers.
Reinterpret Cast	Low-level conversion to reinterpret memory layout of an object.	reinterpret_cast<Type>(value)	Type punning, pointer manipulation (e.g., int* to char*).	Risky, low-level tasks.
C-style Cast	Combines multiple cast types (e.g., static, const, and reinterpret cast).	(Type)value	Used for simple and quick casting (discouraged in modern C++).	General but unsafe.
Function-style Cast	Constructor-like syntax for casting.	Type(value)	Equivalent to C-style cast but more explicit in intent.	General but unsafe.

1. Implicit Casting (Type Promotion)

Automatically performed by the compiler.

```
#include <iostream>

int main() {

   int intVal = 10;

   float floatVal = intVal; // Implicit
conversion from int to float

   std::cout << "Float value: " <<
floatVal << std::endl;

   return 0;

}
```

2. Static Cast

Used for conversions between compatible types, checked at compile time.

```
#include <iostream>

int main() {

    double doubleVal = 9.99;

   int intVal =
static_cast<int>(doubleVal); //
Converts double to int

   std::cout << "Integer value: " <<
intVal << std::endl;

   return 0;

}
```

3. Dynamic Cast

Used for safe downcasting in polymorphic class hierarchies.

```
#include <iostream>
#include <typeinfo>
class Base {
   virtual void func() {} // Makes the class polymorphic
};
class Derived : public Base {};


int main() {
   Base* basePtr = new Derived();
   Derived* derivedPtr = dynamic_cast<Derived*>(basePtr); // Downcasting
   if (derivedPtr) {
      std::cout << "Dynamic cast successful." << std::endl;
   } else {
      std::cout << "Dynamic cast failed." << std::endl;
   }
   delete basePtr;
   return 0;
}
```

4. Const Cast

Used to add or remove the const qualifier.

```
#include <iostream>
int main() {
   const int constVal = 42;
   int& modifiableVal = const_cast<int&>(constVal); // Removes const
   modifiableVal = 50; // Modifying the value
   std::cout << "Modified value: " << modifiableVal << std::endl;
   return 0;
}
```

5. Reinterpret Cast

Performs low-level reinterpretation of memory.

```
#include <iostream>
int main() {
   int intVal = 65;
   char* charPtr = reinterpret_cast<char*>(&intVal); // Reinterprets int as char*
   std::cout << "Character value: " << *charPtr << std::endl; // Outputs 'A'
   return 0;
}
```

6. C-style Cast

Quick but unsafe type conversion.

```
#include <iostream>
int main() {
   double doubleVal = 5.67;
   int intVal = (int)doubleVal; // C-style cast can be double,float,char,string
   std::cout << "Integer value: " << intVal << std::endl;
   return 0;
}
```

7. Function-style Cast

Uses constructor-like syntax for casting.

```
#include <iostream>
int main() {
   double doubleVal = 8.42;
   int intVal = int(doubleVal); // Function-style cast
   std::cout << "Integer value: " << intVal << std::endl;
   return 0;
}
```