WELCOME TO TUTORIAL 10

VISITOR DP, TYPE CASTING C++
VISITOR DESIGN PATTERN

• Earlier in the course we saw dynamic dispatch, which allows selecting the intended method to call in a class hierarchy based on the runtime type of the object.

• What if we want to select the correct method to call based on two different hierarchies?

• Consider a Tree with nodes of various degrees:

• TreeNode.cc
VISITOR DESIGN PATTERN

- However this solution does not take polymorphism into account: what if we have a TreeNode reference?

To solve this problem, we have to make use of dynamic dispatch in both hierarchies. This technique is called double dispatch.

- TreeNode.cc
VISITOR DESIGN PATTERN

• The client code calls t.accept(v) where t is a reference to a TreeNode and v is a reference to a TreeVisitor.
• In accept() methods, the call to the virtual method v.visit() makes sure that v is the correct visitor.
• It seems like UnaryTreeNode::accept() and BinaryTreeNode::accept() have the same code; however, the types of *this are different in those methods. This will make sure the compiler select the correct overloaded method to call.
• Adding a new visitor does not require any implementation change for the TreeNode hierarchy. This is called the visitor pattern.
• This solution promotes low coupling and high cohesion. How?
TYPE CASTING IN C++

• Static Cast
• Dynamic Cast
• Const Cast
• Reinterpret Cast
• This is the simplest type of cast which can be used. It is a **compile time cast**. It does things like implicit conversions between types (such as int to float, or pointer to void*).
DYNAMIC CASTING

• Dynamic casting uses (RTTI) Run Time Type Checking.
• RTTI (Run-time type information) is a mechanism that exposes information about an object’s data type at runtime and is available only for the classes which have at least one virtual function. It allows the type of an object to be determined during program execution
CONSTD CAST

• const_cast is used to cast away the constness of variables.

1. const_cast can be used to change non-const class members inside a const member function.

2. const_cast can be used to pass const data to a function that doesn’t receive const. For example, in the following program fun() receives a normal pointer, but a pointer to a const can be passed with the help of const_cast.
3) It is undefined behavior to modify a value which is initially declared as const. Consider the following program. The output of the program is undefined. The variable ‘val’ is a const variable and the call ‘fun(ptr1)’ tries to modify ‘val’ using const_cast.

4) const_cast is considered safer than simple type casting. It’s safer in the sense that the casting won’t happen if the type of cast is not same as original object. For example, the following program fails in compilation because ‘int *’ is being typecasted to ‘char *’
REINTERPRET CAST

• It is used to convert one pointer of another pointer of any type, no matter either the class is related to each other or not.

• It does not check if the pointer type and data pointed by the pointer is same or not.

• Syntax
  • data_type *var_name = reinterpret_cast<data_type *>(pointer_variable);
PURPOSE

• reinterpret_cast is a very special and dangerous type of casting operator. And is suggested to use it using proper data type i.e., (pointer data type should be same as original data type).

• It can typecast any pointer to any other data type.

• It is used when we want to work with bits.

• If we use this type of cast then it becomes a non-portable product. So, it is suggested not to use this concept unless required.

• It is only used to typecast any pointer to its original type.

• Boolean value will be converted into integer value i.e., 0 for false and 1 for true.