C++ Programming Lab #4

Valparaiso University CS Department

March 30, 2023

Learning Objectives: Learn the benefits and use cases of polymorphism, function overloading and operator overloading.

Turn-In Instructions: Submit a screenshot of your code's output from each step and copy that into a word document. Below the output, copy and paste your code.

1. **Polymorphism:** Paste the following code into your main function...

   ```
   int x;
   x = 5 + 10;
   cout << x;
   ```

   Run the code and review the output. A simple and expected result. Now enter the following code...

   ```
   string y;
   string m = "Hello, ";
   string n = "World!";
   y = m + n;
   cout << y;
   ```

   Review the output. Is this the result you expected? Concatenating strings with the "+" is possible thanks to polymorphism. When "adding" two variables of type String, the compiler knows that what we want is to concatenate them.

2. **Function Overloading:** Write a function called myPrinter that takes an integer argument. In that function, use *cout* to print that value in a sentence of your choosing. Assign an integer variable a value, pass it to the function and run your code. It should simply print your sentence with the value you passed.

   Now write a second function with the same name, myPrinter. This time it should accept an argument of a string data type. Within your

function, concatenate the argument with a sentence/clause of your choosing. Again, use *cout* to print the contents of your new sentence.

Call this new function twice within your main function. The first time passing an integer argument to it, the second time passing a string argument to it. The compiler should determine which version of myPrinter to use based on the argument passed. This simplifies our job as coders, removing the need to remember a different name for every function we create when they have mostly the exact same purpose.

3. **Operator Overloading:** Using the keyword *operator*, create a new definition for an operator of your choice. Use a data type in your definition that demonstrates polymorphism, like strings with the + operator.

   Run your code and see that it gives the result you expect.