ADP first class functions – functions treated as variables

In this exercise we will build a simple calculator doing the operations +,-,*,/ to two variables: the accumulator and the input. The start value of the accumulator is 0. The user dialog goes like this: Present the value of the accumulator on the screen and then ask the user to choose one of the operations +,-,*,/. Then you ask the user for an input value. Then you calculate the new value of the accumulator on the basis of the chosen arithmetic operation and the input number. Make 4 void functions one for each of operations +,-,*,/. They have 2 parameters one reference to int and an integer. The add function prototype will look like this: void add(int& acc, int x); and the function increases acc by adding x to it. We will do this calculation in many different ways:

a) Using function pointers

Declare a function pointer operation_ptr, which can point to one of the 4 operation functions you just have made. When you calculate the new value of the accumulator you first set your function pointer to point to the chosen operation. For instance, if the user chooses + then the function pointer is set to point to the add function. After that you use the function pointer to do the chosen operation.

b) A vector of function pointers

Now make a vector of function pointers called operation which can point to your operation functions. Put the addresses of your 4 operation functions into this vector. When you calculate the new value of the accumulator then you use the user's choice of operation as index to call the correct one of the function pointers in the vector. If the value the user enters for choosing to do subtraction is 1, then 1 will be the index you use when you call the function.

c) Inducing functionality into a function template

Make a function template called calculate with one unknown class parameter called Operation_functor. The function is a void function having two parameters: acc being a reference to int and x being an integer. Now make 4 functors, one for each of the 4 operations. When you do the operation chosen by the user (use a switch-case statement a bit similar to what you did in a)) then you call an instantiated (with an appropriate functor) function from the function template.

d) A vector of 4 std::function<> wrappers for the arithmetic operations

Put in this vector a mixture of a function pointer, a lambda, a named lambda and on of the functors that you have already made and put this vector to work just as you did it with the vector of function pointers.