Lab 10

The Warm up

Previously, we have reused a function by overloading. Now we will try template function to have the same result.

|  |
| --- |
| **Note**  Recall from lecture slides how we can create a template function:  template <typename T>  T add ( T item1, T item2) {  return item1 + item2;  } |

1. Create a template **Add** function in the main.cpp file.
2. Use it to add two integer, float and string variables respectively. For example:

Int v1 = 10, v2 = 20;

int sum = Add<int>(v1, v2);

1. Create a Money class.

|  |
| --- |
| **Money** |
| - m\_amount : double  - m\_unit : string |
| + Money()  + Money(amount : double)  + Money (amount: double, unit : string)  + getValue() : double  + getUnit() : string  + operator+(m1 : Money&, m2 : Money&) : Money **friend** |
| Note: If unit value is not specified, it’s by default “dollar”. |

1. Use the Add function to add two money objects.

|  |
| --- |
| **Note**  Remember the add function takes two variables of same type and tries to add them. But if the type is a class what would happen? Recall that we used operator overloading in lab 8 to print an object. Here you have to overload the **+ operator** to add the amount values of the money objects. *Don’t forget to make the function as* ***friend****.* |

1. Print the result in console.

Exercise 1

From class lectures we already know about standard library templates, one of which is *List*. The STL *List* can sort the items in the list if the items are comparable. We will use this useful sort function to sort the items in the list.

1. Create a new sort function in the main.cpp file and do the following in that function.
2. Add **#include<list>** in the header.
3. Then create an object of list type in the sort function.
4. Take file input from *numbers.txt* file and put them in the list.
5. Use the sort function to sort them.
6. Print the sorted list.

Exercise 2

In this exercise we will create our own *List* template class.

|  |
| --- |
| **MyList** |
| - m\_size : int  - numOfItems : int  - m\_list : T\* |
| + MyList()  + MyList(size : int)  + add(item : const T)  + remove(index : int)  + get(index : int)  - increaseSize() |
| Note:  a. Remove function will remove the item in index position. Then shift the items that were to it’s right to left filling up the empty space.  b. IncreaseSize function will increase the size by 5.  c. Get function will not remove the item; it will just make a copy and return it.  d. While adding new item in Add function, if the list is full, first increase the size by calling the increaseSize function. |

1. Create a new project.
2. Create the template *MyList* class.
3. Create a main.cpp and main function.
4. Create a MyList object of type string with size 2.

List<string> names(2);

1. Add five strings: “Jack”, “Mary”, “John”, “Robert”, “Catherine” in the list.
2. Get the 5th item and print it. If it prints “Catherine” then your ***add*** and **g*et*** functions are working fine.
3. Remove the 3rd item “John”.
4. Get the 3rd item and print it. If it prints “Robert”, your ***remove*** is also fine.

Bonus Quiz

Create a text file and write the answers:

1. What would happen if we create a vector of size 10 and the vector is already filled up and we want to push a new item?
2. In a stack, let’s say we have entered four items, “John”, “Mary”, “Jack”, “Robert”. If we pop for the first time, which item will pop out?
3. If the same four items are stored in the queue, which item will pop out?