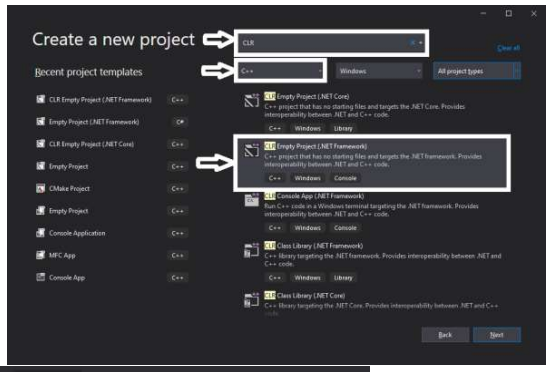
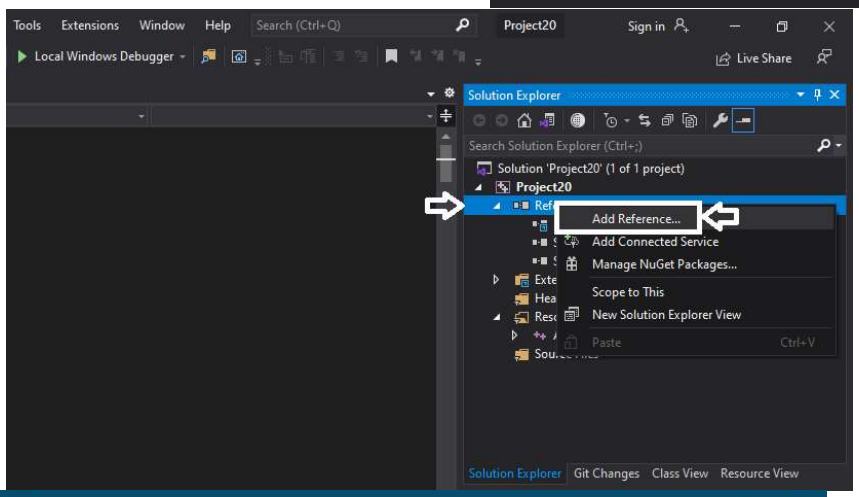
Домашнє завдання №22.2

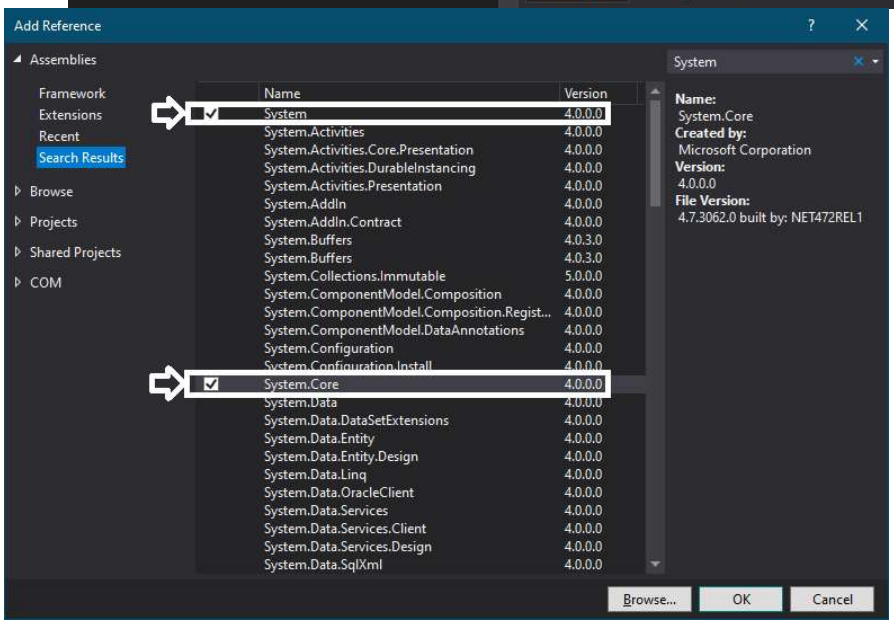
C++/CLI дозволяє писати програми, які водночас можуть містити і звичайний некерований код*(англ. unmanaged code)* написаний на C++ для компіляції безпосередньо в машинний код, і керований код *(англ. managed code)* написаний на C++ для .NET.

Виконати домашнє завдання №22.1 повторно за допомогою C++/CLI.



**** коментар:*** *це завдання тотожне до завдань №19, №20, №21 та №22.1; таким чином можна порівняти різні засоби програмування; далі наводиться приклад повністю виконаного завдання; для компіляції на ПК за допомогою Visual Studio слід створити пустий проект для* ***.NET Framework*** *версії* ***4*** *та обов'язково добавити у проект залежності* ***System*** *та* ***System.Core****.*





Вибір варіанту

Задана літера це перша літера прізвища студента *(записаного латинськими літерами)*

Приклад коду

Наведений зразок коду реалізовує завдання з виконання умови розміщення першими слів, що починаються на літеру ‘К’.

Літера для прикладу	K
Декларація константи в класі	`static const char FIRST_CH = 'K';`

Лістинг

```
// compile with: /clr

using namespace System;
//using namespace System::Collections;
using namespace System::Collections::Generic;
using namespace System::IO;
using namespace System::Linq;
using namespace System::Text;
using namespace System::Text::RegularExpressions;
using namespace System::Threading::Tasks;

namespace ACMHW22_2{
	public ref class ACMHW22_2{
		static const char FIRST_CH = 'K';

		const int MAX_BUFFER_SIZE = 8192;

	public: void scan(String ^ str, List<int> ^ list){
			if (str == nullptr || list == nullptr){
				return;
			}
			Regex ^ token_re = gcnew Regex("[a-z]+", RegexOptions::IgnoreCase);
			for (Match ^ match = token_re->Match(str); match->Success; match =
match->NextMatch()){
				list->Add(match->Index);
			}
		}

	public: static void copyStream(Stream ^ input, Stream ^ output, int start, int
maxExpectedEnd){
			int maxExpectedBytesRead = maxExpectedEnd - start;
			array<Byte> ^ buffer = gcnew array<Byte>(maxExpectedBytesRead);
			for (int bytesRead; (bytesRead = input->Read(buffer, start,
maxExpectedBytesRead)) != 0/* != -1*/ && bytesRead <= maxExpectedBytesRead;){ // !
				output->Write(buffer, 0, bytesRead);
			}
		}

	public: static String ^ toString(int value){
			return String::Format("{0}", value) + "\n";
		}

	public: void printListIndexes(List<int> ^ list) {
			if (list == nullptr) {
				return;
```

```
                    }

                    copyStream(
                           gcnew MemoryStream(
                               Encoding
                               ::ASCII
                               ->GetBytes(
                                   String
                                   ::Concat(
                                       list
                                       ->ConvertAll(
                                           gcnew Converter<int, String^>(
                                               toString
                                           )
                                       )
                                   )
                               )
                           ),
                           (gcnew StreamWriter(Console::OpenStandardOutput()))-
>BaseStream,
                           0,
                           MAX_BUFFER_SIZE);
             }

      public: void print(String ^ str, List<int> ^ list) {
                    if (list == nullptr) {
                           return;
                    }

                    for each(int value in list){
                           String ^ word = str->Substring(value);
                           Match ^ match = (gcnew Regex("[a-z]+",
RegexOptions::IgnoreCase))->Match(word);
                           if (match->Success) {
                                  Console::WriteLine(match->Value);
                           }
                    }
             }

      ref class ClassCompareFunction1 : IComparer<int>{
             private: String ^ str;

             public: ClassCompareFunction1(String ^ text) {
                           this->str = text;
                    }

                    int strcmp__withoutCase(int str1BaseIndex, int str2BaseIndex) {
                           for (int str1Index = str1BaseIndex, str2Index = str2BaseIndex;
str1Index < str->Length && str2Index < str->Length; ++str1Index, ++str2Index) {
                                  wchar_t str1_tolower = Char::ToLower(str[str1Index]);
                                  wchar_t str2_tolower = Char::ToLower(str[str2Index]);

                                  if (str1_tolower != str2_tolower)
                                  {
                                         return str1_tolower < str2_tolower ? -1 : 1;
                                  }
                           }

                           return 0;
                    }

                    int strcmp_K__withoutCase(int str1BaseIndex, int str2BaseIndex) {
                           wchar_t chr1_toupper = Char::ToUpper(str[str1BaseIndex]);
                           wchar_t chr2_toupper = Char::ToUpper(str[str2BaseIndex]);
                           if (chr1_toupper == FIRST_CH && chr2_toupper != FIRST_CH) {
```

```
                                return -1;
                        }
                        else if (chr1_toupper != FIRST_CH && chr2_toupper == FIRST_CH)
{
                                return 1;
                        }
                        else if (chr1_toupper == FIRST_CH && chr2_toupper == FIRST_CH)
{
                                return strcmp__withoutCase(str1BaseIndex + 1,
str2BaseIndex + 1);
                        }

                        return strcmp__withoutCase(str1BaseIndex, str2BaseIndex);
                }

        public: int compareFunction(int arg1, int arg2) {
                        //return str->Substring(arg1)->CompareTo(str-
>Substring(arg2)); // with case sensitive
                        return String::Compare(str->Substring(arg1), str-
>Substring(arg2), StringComparison::OrdinalIgnoreCase);
                }

        public: int compareFunction1(int arg1, int arg2) {
                        return strcmp_K__withoutCase(arg1, arg2);
                }

        public: virtual int Compare(int arg1, int arg2) {
                        return compareFunction1((int)arg1, (int)arg2);
                }
        };

        public: void sort(String ^ str, List<int> ^ data){
                //IComparer<int> ^ comparer = gcnew ClassCompareFunction1(str);
                data->Sort(gcnew ClassCompareFunction1(str));
        }

        Dictionary<int, String^> ^ getMapList(String ^ str, List<int> ^ list){
                if (str == nullptr || list == nullptr){
                        return nullptr;
                }

                Dictionary<int, String^> ^ mapList = gcnew Dictionary<int,
String^>();

                for (int index = 0; index < list->Count; ++index){
                        String ^ word = str->Substring(list[index]);
                        Match ^ match = (gcnew Regex("[a-z]+",
RegexOptions::IgnoreCase))->Match(word);
                        if (match->Success){
                                mapList->Add(index, match->Value);
                        }
                }

                return mapList;
        }

        public: static String ^ toString(KeyValuePair<int, String^> data){
                return "{ " + String::Format("{0}", data.Key) + ", " +
String::Format("{0}", data.Value) + " }\n";
        }

        public: void printMapList(Dictionary<int, String^> ^ mapList) {

                        if (mapList == nullptr) {
                                return;
```

```
                                    }


        Array::ForEach(Enumerable::ToArray(Enumerable::Select(mapList, gcnew
Func<KeyValuePair<int, String^>, String^>(toString))), gcnew
Action<String^>(Console::Write)); // Console::Out->Write
                }

public: void printMapList__use_cliext(Dictionary<int, String^> ^ mapList);

                static void Main(array<System::String ^> ^args){
                        ACMHW22_2 ^ acmhw22_2 = gcnew ACMHW22_2();

                        List<int> ^ list = gcnew List<int>();
                        String ^ text =
                                "Sir, in my heart there was a kind of fighting " +
                                "That would not let me sleep. Methought I lay " +
                                "Worse than the mutines in the bilboes. Rashly- " +
                                "And prais'd be rashness for it-let us know " +
                                "Our indiscretion sometimes serves us well ... "
                                ; // — Hamlet, Act 5, Scene 2, 4–8

                        acmhw22_2->scan(text, list);
                        acmhw22_2->sort(text, list);
                        Dictionary<int, String^> ^ mapList = acmhw22_2->getMapList(text,
list);

                        Console::WriteLine("Indexes:");
                        acmhw22_2->printListIndexes(list);
                        Console::WriteLine();
                        Console::WriteLine("Values:");
                        acmhw22_2->print(text, list);
                        Console::WriteLine();
                        Console::WriteLine("Values(by map):");
                        acmhw22_2->printMapList(mapList);
                        Console::WriteLine("Press any key to continue . . . ");
                        Console::ReadKey();

                }
        };
};

//#include <functional>
#include <cliext/adapter>
#include <cliext/algorithm>
#include <cliext/map>

// cliext use example
void ACMHW22_2:: ACMHW22_2::printMapList__use_cliext(Dictionary<int, String^> ^ mapList)
{
        if (mapList == nullptr) {
                return;
        }
    cliext::collection_adapter<System::Collections::Generic::IDictionary<int, String^>>
mapListAdapter(mapList);
        for each (KeyValuePair<int, String^> ^kvp in mapListAdapter){
                Console::WriteLine("{{ {0}, {1} }}", kvp->Key, kvp->Value);
        }
}

int main(array<System::String ^> ^args){
        ACMHW22_2:: ACMHW22_2::Main(args);
    return 0;
}
```