підготував старший викладач каф. ЕОМ Козак Назар Богданович

Домашнє завдання №20

Замінивши використання ***std::vector*** на ***boost::container::vector***, а ***std::map*** на ***boost::container::map***, виконати домашнє завдання №19 повторно.

**** коментар:*** *це завдання тотожне до завдань №19, №21 та №22; таким чином можна порівняти різні засоби програмування; далі наводиться приклад повністю виконаного завдання; для компіляції і запуску* **** рекомендується використати*** ***https://www.onlinegdb.com/online_c++_compiler*** *; для компіляції на ПК слід скачати з* https://www.boost.org/users/history/version_1_72_0.html та розархівувати *набір бібліотек Boost, після чого налаштувати Visual Studio як показано в цій відео-демонстрації* https://www.youtube.com/watch?v=SWCCFGX6c0g .

Вибір варіанту

Варіант завдання відповідає варіанту домашнього завдання №19

Приклад коду

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

Літера для прикладу	K
Макровизначення	`#define FIRST_CH 'K'`

Лістинг

```
#define _CRT_SECURE_NO_WARNINGS
#include <ctype.h>

#include <cctype> // #include <locale>

#include <fstream>
#include <sstream>
#include <iostream>

#include <regex>

#include <boost/container/vector.hpp>
#include <boost/container/map.hpp>
#include <algorithm>

#define FIRST_CH 'K'

void scan(std::string* str, boost::container::vector<int>* list) {
	if (!str || !list) {
		return;
	}

	std::regex token_re("[a-z]+", std::regex::icase);
	for (std::sregex_token_iterator iterator = std::sregex_token_iterator(str-
>begin(), str->end(), token_re, 0);
		iterator != std::sregex_token_iterator(); ++iterator) {
		list->push_back((int)(iterator->first - str->begin())); // !
```

```
	}
}

void printListIndexes(boost::container::vector<int>* list) {
	if (!list) {
		return;
	}

	std::copy(list->begin(), list->end(), std::ostream_iterator<int>(std::cout,
"\n")); // \r\n
}

void print(std::string* str, boost::container::vector<int>* list) {
	if (!str || !list) {
		return;
	}

	for (int value : *list) {
		std::string word = str->substr(value);
		std::smatch match;

		if (std::regex_search(word, match, std::regex("[a-z]+",
std::regex::icase))) {
			std::cout << match[0] << std::endl;
		}
	}
}

class ClassCompareFunction1 {
private:
	std::string* str;
public:
	ClassCompareFunction1(std::string* text) : str(text) {}
private:
	int strcmp__withoutCase(int str1BaseIndex, int str2BaseIndex) {
		for (int str1Index = str1BaseIndex, str2Index = str2BaseIndex; str1Index <
(int)str->size() && str2Index < (int)str->size(); ++str1Index, ++str2Index) {
			int str1_tolower = std::tolower((*str)[str1Index]);
			int str2_tolower = std::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) {
		int chr1_toupper = std::toupper((*str)[str1BaseIndex]);
		int chr2_toupper = std::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);
	}
```

```
	bool compareFunction(int arg1, int arg2) {
		//return str->substr(arg1) < str->substr(arg2); // with case sensitive
		return strcmp__withoutCase(arg1, arg2) < 0;
	}

public:
	bool compareFunction1(int arg1, int arg2) {
		return strcmp_K__withoutCase(arg1, arg2) < 0; // !
	}

	bool operator()(int arg1, int arg2) {
		return compareFunction1(arg1, arg2);
	}
};

void sort(std::string* str, boost::container::vector<int>* data) {
	std::sort(data->begin(), data->end(), ClassCompareFunction1(str));
}

boost::container::map<int, std::string>* getMapList(std::string* str,
boost::container::vector<int>* list) {
	if (!str || !list) {
		return nullptr;
	}

	boost::container::map<int, std::string>* mapList = new boost::container::map<int,
std::string>();

	for (unsigned int index = 0; index < list->size(); ++index) {
		std::string word = str->substr((*list)[index]);
		std::smatch match;

		if (std::regex_search(word, match, std::regex("[a-z]+",
std::regex::icase))) {
			mapList->insert(std::pair<int, std::string>(index, match[0]));
		}
	}

	return mapList;
}

std::string toString(const std::pair< int, std::string >& data) {
	std::ostringstream str;
	str << "{ " << data.first << ", " << data.second << " }\n";
	return str.str();
}

void printMapList(boost::container::map<int, std::string>* mapList) {
	if (!mapList) {
		return;
	}

	std::transform(
		mapList->begin(),
		mapList->end(),
		//std::ostream_iterator< std::string >(std::cout, "\n"),
		std::ostream_iterator< std::string >(std::cout),
		toString);
}

int main() {
	boost::container::vector<int>* list = new boost::container::vector<int>();
	std::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

      scan(&text, list);
      sort(&text, list);
      boost::container::map<int, std::string>* mapList = getMapList(&text, list);

      std::cout << "Indexes:" << std::endl;
      printListIndexes(list);

      std::cout << std::endl << "Values:" << std::endl;
      print(&text, list);

      std::cout << std::endl << "Values(by map):" << std::endl;
      printMapList(mapList);

#ifdef __linux__
      std::cout << "Press any key to continue . . . " << std::endl;
      (void)getchar();
#elif defined(_WIN32)
      system("pause");
#else
#endif

      delete list;
      delete mapList;

      return 0;
}
```