Министерство науки и высшего образования Российской Федерации  
Федеральное государственное бюджетное образовательное учреждение   
высшего образования   
«Владимирский государственный университет   
имени Александра Григорьевича и Николая Григорьевича Столетовых»  
(ВлГУ)

Институт прикладной математики, физики и информатики

Кафедра физики и прикладной математики

ЛАБОРАТОРНАЯ РАБОТА №6  
по дисциплине  
«Основы программирования»  
на тему:  
« Многомодульное программирование»

Выполнил:  
ст. гр. ПМИ-123

Рушев А.М.  
  
  
Принял:  
ст. преподаватель   
каф. ФиПМ  
Шишкина М.В.

Владимир,2024

**Цель работы**

Изучение способов построения и компиляции многомодульных проектов.

**Постановка задачи**

Создать заголовочный файл, разместить в нём заголовки функций.

В отдельном модуле с расширением \*.срр описать следующие функции:

* заполнения целочисленного массива.
* сортировки одномерного целочисленного массива (любым способом).
* вывод элементов массива на экран.
* записи элементов массива в файл.

Вынести объявления этих функций в заголовочный файл.

В main функции реализовать следующий сценарий:

Объявить два одномерных целочисленных массива, количество элементов массива должно быть различным.

Для каждого массива вызвать следующие функции в указанном порядке: заполнения, сортировки, вывода на экран, записи элементов в файл.

Реализовать сортировку слиянием элементов массива. (желательно написать для этого функцию, получающую в качестве входных данных два указателя на файл).

Вывести на экран данные из третьего файла.

Реализовать раздельную и совместную компиляцию.

**Теоретическая часть**

Модуль — файл с исходным кодом и расширением \*.cpp.

Многомодульное программирование позволяет разбить задачу на подзадачи. При этом, открывается возможность совместной работы над проектом. В ходе которой каждый разработчик решает свою задачу. Даёт возможности использования ранее написанного кода, для этого достаточно подключить соответствующим модуль к проекту.

В одном из модулей проекта обязательно должна содержаться main-функция, являющаяся точкой входа в программу.

При работе над проектом рекомендуется в модуле с main- функцией реализовывать только сценарий работы – вызов соответствующих функций.

Описание функций и классов, группировать в модулях с учетом их предназначения. Вынося при этом заголовки описаний в заголовочные файлы.

Заголовочным файлом называют файл с кодом на C++ и расширением \*.h. В заголовочные файлы выносят объявления функций, пользовательских типов данных и некоторые директивы.

Получение из исходного кода исполняемого файла происходит в несколько этапов: препроцессинг, трансляция (компиляция), сборка (линковка).

Препроцессинг включает в себя следующие действия:

- текстовое включение файлов —. #include

- макроподстановки.—. #define

- обработка директив условной компиляции — #if, #ifdef, #elif, #else, #endif.

Назначение компиляции - перевод программы, написанной на языке высокого уровня в набор команд близких к машинным.

Таким образом, после компиляции файл с исходным кодом программы преобразуется в файл, называемый объектным модулем имеющий расширение \*.obj.

Компоновка — это последний этап процесса получения исполняемого файла, состоящий из связывания воедино всех объектных файлов проекта.

Возможно возникновение ошибок связывания, например, если функция была объявлена, но не определена, ошибка обнаружится только на этом этапе.

В ходе трансляции устанавливается связь между операндами и адресами память, в которых хранятся, например, результаты вычислений. Компоновщик отвечает за то, чтобы конкретному операнду соответствовала определённая область памяти.

К компонуемой программе добавляются коды библиотечных функций (обеспечивающих выполнение конкретных действий – вычисления, вывод на экран), а также код, обеспечивающий размещение программы в памяти, её корректное начало и завершение.

Преобразованная компоновщиком программа называется загрузочным или исполняемым модулем.

В языке С++ не существует специальных языковых конструкций, описывающих программную структуру многомодульного проекта.

Возможна реализация одного из двух режимов компиляции: раздельная и совместная.

Раздельная компиляция.

Все модули компилируются независимо друг от друга, генерируется столько \*.obj файлов, сколько было модулей с исходным кодом.

Во время компоновки собираются все obj файлы в один exe файл. Для успешной компоновки необходимо, чтобы во всех компонуемых файлах содержалось объявление всех используемых функций (как пользовательских, так и стандартных) и определение внешних переменных (extern). Иначе возникнет ошибка компоновки.

Для этого необходимо подключение через директиву #include заголовочных файлов во все модули, где планируется вызов объявленных в них функций.

Совместная компиляция.

При совместной компиляции на вход компилятору подаётся один исходный файл, содержащий в себе все исходные модули, подключённые через директиву #include. Далее полученный obj файл обрабатывается компоновщиком, который в свою очередь собирает exe-файл.

Модули с исходным кодом, подключенные с помощью директивы #include, в модуль, содержащий main – функцию, из проекта необходимо исключить (remove). Иначе возникнет ошибка.

**Практическая часть**

Листинг кода приложения:

AllFunctions.h – заголовочный файл функций

#pragma once

void InputMass(int\*, int);

void SortMass(int\*, int);

void PrintScreenMass(int\*, int);

void WriteToFile(int\*, int, char\*);

void scriptmas(int\*, int\*, int, int);

File.h – заголовочный файл для работы с файлами

#include <cstdio>

#pragma once

void writetofile(FILE\*, FILE\*);

int mergeFiles(const char\*, const char\*, const char\*);

void writemasToFile(int\*, int, const char\*);

void printfile(const char\*);

File.cpp

#include "file.h"

#include<iostream>

#define \_CRT\_SECURE\_NO\_WARNINGS

using namespace std;

void writetofile(FILE\* file, FILE\* outputFile) {

int num;

while (fscanf\_s(file, "%d", &num) != EOF) {

fprintf(outputFile, "%d ", num);

}

}

int mergeFiles(const char\* fileName1, const char\* fileName2, const char\* outputFileName)

{

FILE\* file1;

FILE\* file2;

FILE\* outputFile;

fopen\_s(&file1, fileName1, "r");

fopen\_s(&file2, fileName2, "r");

fopen\_s(&outputFile, outputFileName, "w");

if (file1 == NULL || file2 == NULL || outputFile == NULL)

{

cout << "Error opening files." << endl;

return 0;

}

int num1, num2;

if (fscanf\_s(file1, "%d", &num1) == EOF)

{

writetofile(file2, outputFile);

return 0;

}

else if (fscanf\_s(file2, "%d", &num2) == EOF)

{

writetofile(file1, outputFile);

return 0;

}

else

{

while (true)

{

if (num1 <= num2)

{

fprintf(outputFile, "%d ", num1);

if (fscanf\_s(file1, "%d", &num1) == EOF)

{

fprintf(outputFile, "%d ", num2);

break;

}

}

else

{

fprintf(outputFile, "%d ", num2);

if (fscanf\_s(file2, "%d", &num2) == EOF)

{

fprintf(outputFile, "%d ", num1);

break;

}

}

}

}

writetofile(file1, outputFile);

writetofile(file2, outputFile);

fclose(file1);

fclose(file2);

fclose(outputFile);

return 0;

}

void writemasToFile(int mas[], int size, const char\* filename) {

FILE\* file;

fopen\_s(&file, filename, "w");

if (file != NULL) {

for (int i = 0; i < size; i++) {

fprintf(file, "%d ", mas[i]);

}

fclose(file);

}

else {

cout << "Error" << endl;

}

}

void printfile(const char\* filename) {

FILE\* file;

errno\_t err = fopen\_s(&file, filename, "r");

if (err != 0) {

cout << "Failed to open file" << endl;

return;

}

int num;

while (fscanf\_s(file, "%d", &num) != EOF) {

cout << num << " ";

}

fclose(file);

}

AllFunctions.cpp

#include <iostream>

#include "AllFunctions.h"

using namespace std;

extern int dir = 0;

static void InputMass(int\* arr, int n) {

for (int i = 0; i < n; i++) {

arr[i] = 1 + rand() % 50;

}

}

static void SortMass(int\* arr, int n) {

for (int i = 0; i < n; i++) {

for (int j = 0; j < n - 1; j++) {

if (arr[j] > arr[j + 1]) {

int temp = arr[j];

arr[j] = arr[j + 1];

arr[j + 1] = temp;

}

}

}

}

static void PrintScreenMass(int\* arr, int n) {

for (int i = 0; i < n; i++) {

cout << arr[i] << "\t";

}

cout << endl;

}

void scriptmas(int\* mas1, int\* mas2, int size1, int size2) {

static int count = 0;

InputMass(mas1, size1);

SortMass(mas1, size1);

PrintScreenMass(mas1, size1);

InputMass(mas2, size2);

SortMass(mas2, size2);

PrintScreenMass(mas2, size2);

cout << endl << "Количество вызовов функции: " << ++count << endl;

dir++;

}

ConsoleApplication6.cpp

#include <iostream>

#include "AllFunctions.h";

#include "File.h";

using namespace std;

int main()

{

setlocale(LC\_ALL, "rus");

cout << "Лабораторная работа №6" << endl;

const int size1 = 10;

const int size2 = 7;

int\* arr6\_1 = new int[size1];

int\* arr6\_2 = new int[size2];

scriptmas(arr6\_1, arr6\_2, size1, size2);

writemasToFile(arr6\_1, size1, "Text1.txt");

writemasToFile(arr6\_2, size2, "Text2.txt");

mergeFiles("Text1.txt", "Text2.txt", "Text3.txt");

cout << endl;

printfile("Text3.txt");

cout << endl << endl;

scriptmas(arr6\_1, arr6\_2, size1, size2);

system("pause");

return 0;

}

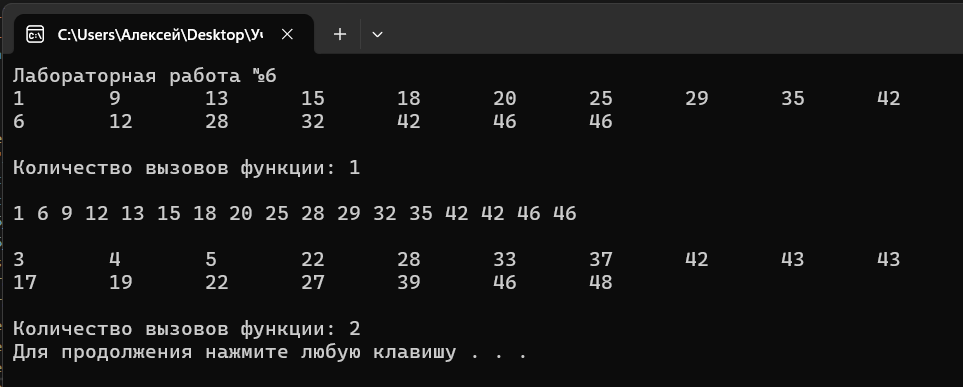


Рисунок 1 – Результат выполнения кода программы

**Вывод:**

Плюсы многомодульного программирования:

1. Улучшение структуры программы: разделение программы на модули позволяет лучше организовать код, что упрощает его понимание, поддержку и масштабирование.

2. Увеличение повторного использования кода: модули могут быть использованы повторно в разных частях программы или в других проектах.

3. Улучшение совместной работы: каждый модуль может разрабатываться и тестироваться независимо от других, что упрощает совместное программирование.

4. Увеличение безопасности: использование модулей помогает ограничить доступ к приватной информации и уменьшить возможность ошибок в коде.

Минусы многомодульного программирования:

1. Увеличение сложности кода: большое количество модулей может привести к усложнению структуры кода и его пониманию, особенно для новых разработчиков.

2. Проблемы с зависимостями: неправильное управление зависимостями между модулями может привести к ошибкам в работе программы.

3. Проблемы с тестированием: тестирование модулей независимо друг от друга может быть сложным и требовать дополнительных усилий.

4. Дополнительные затраты на разработку: создание многих модулей может потребовать дополнительных ресурсов и времени на их разработку и поддержание.