**МИНОБРНАУКИ РОССИИ**

**Санкт-Петербургский государственный**

**электротехнический университет**

**«ЛЭТИ» им. В.И. Ульянова (Ленина)**

**Кафедра МО ЭВМ**

отчет

**по лабораторной работе №2**

**по дисциплине «Программирование»**

Тема: СБОРКА ПРОЕКТОВ В ЯЗЫКЕ СИ

|  |  |  |
| --- | --- | --- |
| Студент гр. 3341 |  | Мальцев К.Л. |
| Преподаватель |  | Глазунов С.А. |

Санкт-Петербург

2023

## Цель работы

Целью данной работы является изучение процесса сборки программ, написанных на языке Си на примере использования make-файлов.

Для достижения поставленной цели требуется решить следующие задачи:

1) Изучить как происходит процесс компиляции и линковки с использованием компилятора gcc.

2) Изучить структуру и правила составления make-файлов.

3) Написать make-файл для сборки заданной программы.

## Задание

Вариант 4

В текущей директории создайте проект с make-файлом. Главная цель должна приводить к сборке проекта. Файл, который реализует главную функцию, должен называться menu.c; исполняемый файл - menu. Определение каждой функции должно быть расположено в отдельном файле, название файлов указано в скобках около описания каждой функции.

Реализуйте функцию-меню, на вход которой подается одно из значений 0, 1, 2, 3 и массив целых чисел размера не больше 100. Числа разделены пробелами. Строка заканчивается символом перевода строки.

В зависимости от значения, функция должна выводить следующее:

0 : индекс первого чётного элемента. (index\_first\_even.c)

1 : индекс последнего нечётного элемента. (index\_last\_odd.c)

2 : Найти сумму модулей элементов массива, расположенных от первого чётного элемента и до последнего нечётного, включая первый и не включая последний. (sum\_between\_even\_odd.c)

3 : Найти сумму модулей элементов массива, расположенных до первого чётного элемента (не включая элемент) и после последнего нечётного (включая элемент). (sum\_before\_even\_and\_after\_odd.c)

иначе необходимо вывести строку "Данные некорректны".

## Основные теоретические положения

Для сборки программ, состоящих из нескольких файлов часто используются make-файлы. Make-файл - это текстовый файл, в котором определенным образом описаны правила сборки приложения. Для работы с make-файлами используется утилита make, которая позволяет не компилировать файлы дважды, если это не требуется. Например в тех случаях, когда в вашем большом проекте изменился только один файл, утилита скомпилирует только его и не станет перекомпилировать все остальные. Давайте рассмотрим основные принципы работы make.

При запуске утилита пытается найти файл с заданным по умолчанию именем Makefile в текущем каталоге и выполнить содержащиеся в нем инструкции. Возможно явно указать какой make-файл использовать с помощью ключа -f: make -f MyMakefile

Базовые части make-файла выглядят обычно следующим образом:

цель: зависимости

[tab] команда (таких строк может быть несколько)

По-умолчанию, основной целью считается первая описанная цель в файле. С нее и начинается обработка файла утилитой make. Целью в make-файле является файл, который получается в результате выполнения команд. Также целью может быть название действия, которое будет выполнено (без зависимостей), например:

clean:

[tab] rm \*.o

Зависимости - это файлы, которые make проверяет на наличие и дату изменений. Зависимости необходимы для получения цели: утилита make проверяет, были ли зависимости обновлены с последнего запуска make, и, если зависимость стала новее, обновляет цель. Таким образом, обновляются только “устаревшие” цели и нет необходимости каждый раз пересобирать весь проект.

## Выполнение работы

В файле menu.c:

Подключается стандартная библиотека *<stdio.h>*:

*#include <stdio.h>*

Подключаются остальная библиотеки:

*#include "index\_first\_even.h"*

*#include "index\_last\_odd.h"*

*#include "sum\_between\_even\_odd.h"*

*#include "sum\_before\_even\_and\_after\_odd.h"*

Создаётся макрос *#define ARR\_SIZE 100*

Определяются следующие функции:

1. *void get\_input(char\*, int\*, int[]);*
2. *void allocator(char, int[], int);*

Создается переменная *type* (тип *char*)

Задаётся переменная *array* (массив типа *int*, размера *ARR\_SIZE*), где будет храниться, введенный массив чисел.

Задаётся переменная *size* (тип *int*), изначально равная 0. Данная переменная является счётчиком, который будет отображать сколько чисел было введено в массив.

Далле отрабатывает функция *get\_input(&type, &size, array)*, которая записывает входные значения в *type* и *array*, находит размер входного массива.

Далее отрабатывает функция *allocator(type, array, size)*, которая в зависимости от значения переменной *type* выводит ответ на одну из подзадач или выводит строку “Данные некорректны”.

Функции:

1. *void get\_input(char\* type, int\* size, int array[])*

Получает на вход указатель переменную *type* (тип *char*), указатель на переменную size (тип *int*), массив array (тип *int*). С помощью *scanf()* записывается входное значение в *type*. Далее в цикле при помощи проверки каждого входного символа вводится массив *array* и изменяется значение *size*. *while (getchar() != '\n') { scanf("%d", &array[(\*size)++]); }*

1. *void allocator(char type, int array[], int size)*

Получает на вход переменную *type* (тип *char*), массив *array* (тип *int*), переменную *size* (тип *int*). С помощью оператора множественного выбора *switch()* сравнивает *type* со списком константных выражений {0, 1, 2, 3}

Если *type* = 0, то выводится результат функции *index\_first\_even(array, size)*

Если *type* = 1, то выводится результат функции *index\_last\_odd(array, size)*

Если *type* = 2, то выводится результат функции *sum\_between\_even\_odd(array, size)*

Если *type* = 3, то выводится результат функции *sum\_before\_even\_and\_after\_odd(array, size)*

Если *type* не равен ни одному из перечисленных выше значений, то выводится строка "Данные некорректны"

Все остальные функции записаны в одноименных файлах формата \*.c. В начале каждого файла есть подключение библиотеки *<stdlib.h>*, т.к. во всех описанных функциях используется *abs()*.

Примечание: в файлах *sum\_before\_even\_and\_after\_odd.c* и *sum\_between\_even\_odd.c* также подключены библиотеки *"index\_first\_even.h"* и *"index\_last\_odd.h"*.

Функции:

1. *int index\_first\_even(int array[], int size)*

Получает на вход массив *array* (тип *int*), переменную *size* (тип *int*). Возвращает индекс первого чётного элемента. Индекс находится путём прохода по массиву циклом: *for (int i = 0; i < size; i++){…}*. Внутри тела цикла выполняется проверка на чётность очередного *array[i]*: *if(abs(array[i]) % 2 == 0)*. В данной задаче в массиве могут оказаться отрицательные числа, поэтому для корректной проверки на чётность берётся остаток от деления модуля *array[i]* на 2. Если условие выполнилось, то выполняется *return i* и выход из функции. (В задаче гарантировано, что в массиве есть хотя бы один чётный и нечётный элемент, поэтому данное условие гарантировано будет выполнено)

1. *int index\_last\_odd(int array[], int size)*

Получает на вход массив *array* (тип *int*), переменную *size* (тип *int*). Возвращает индекс последнего нечётного элемента. Работает аналогично функции *int* *index\_first\_even(int array[], int size)*. Отличие заключается в том, что цикл запускается с конца *for (int i = size-1; i >= 0; i--){…}*, и в теле цикла заменено условие проверки на чётность на проверку на нечётность: *if (abs(array[i]) % 2 != 0)*.

1. *int sum\_between\_even\_odd(int array[], int size)*

Получает на вход массив *array* (тип *int*), переменную *size* (тип *int*). Возвращает сумму модулей элементов массива, расположенных от первого чётного элемента и до последнего нечётного, включая первый и не включая последний. Создаются две переменные типа *int*: *ife* = *index\_first\_even(array, size)* и *ilo* = *index\_last\_odd(array, size)*. В них хранится первое вхождение чётного элемента и последнее вхождение нечётного элемента, соответственно. Создаётся переменная *sm* (тип *int*), равная 0. В ней будет храниться искомая сумма. Далее запускается цикл: *for (int i = ife; i < ilo; i++){…}*, в теле которого изменяется переменная *sm*: *sm += abs(array[i])*. После цикла функция возвращает значение переменной *sm*.

1. *int sum\_before\_even\_and\_after\_odd(int array[], int size)*

Получает на вход массив *array* (тип *int*), переменную *size* (тип *int*). Возвращает сумму модулей элементов массива, расположенных до первого чётного элемента (не включая элемент) и после последнего нечётного (включая элемент). Работает аналогично функции *int* *sum\_between\_even\_odd(int array[], int size)*. Отличие заключается в том, что вместо одного цикла *for (int i = ife; i < ilo; i++){…}* используется 2 цикла *for (int i = 0; i < ife; i++){…}* и *for (int i = ilo; i < size; i++){…}*. В телах обоих циклов изменяется переменная *sm*: *sm += abs(array[i])*. После циклов функция возвращает значение переменной *sm*.

Также для решения задачи требовалось описать заголовочные файлы. Все они выполнены по шаблону:

#ifndef <ИМЯ ЗАГОЛОВОЧНОГО ФАЙЛА>\_H

#define <ИМЯ ЗАГОЛОВОЧНОГО ФАЙЛА>\_H

<объявление функции, определенной в соответствующем файле \*.c>

#endif

Пояснение: конструкция #ifndef … #define … … #endif нужна для того, чтобы при подключении заголовочных файлов не произошло «бесконечного подключения», её также можно было бы заменить на конструкцию #pragma once.

Для сборки проекта был написан Makefile.

Используется переменная *TARGET* с значением *menu*. Она описывает итоговой результат выполнения make-файла – исполняемый файл *menu*.

Используется переменная *CC* с значением *gcc*. (Используемое семейство компиляторов)

Используется переменная *CFLAGS* с значением *-Wall -std=gnu99* (флаг компиляции, где *-Wall* нужен для того, чтобы в терминале показывались все предупреждения (warnings all); *-std=gnu99 –* версия компилятора)

Используется переменная *SRC,* в которую записаны имена файлов шаблона \*.c. Для развертки шаблона используется *wildcard*.

Используется переменная *OBJ*, в которую записаны имена файлов шаблона \*.c с изменным разрешением на .o. Для данной замены использовалась конструкция *OBJ = $(patsubst %.c, %.o, $(SRC))*

Основная цель make-файла – all описана так: *all : $(TARGET)*

Для получения цели *$(TARGET)* нужны файлы-объектники. Использовалась запись:

*$(TARGET) : $(OBJ)*

*[tab]$(CC) $(CFLAGS) $^ -o $@*.

(С помощью семейства компиляторов *$(CC)* и флагов *$(CFLAGS)* получить из бинарных файлов *$(OBJ)* исполняемый файл *$(TARGET)*)

$^ - Названия всех предварительных условий с пробелами между ними

$@ - имя файла целевого объекта правила.

Для получения объектных файлов использовалась запись:

*%.o : %.c*

*[tab]$(CC) $(CFLAGS) -c $< -o $@*

(C помощью семейства компиляторов *$(CC)* и флагов *$(CFLAGS)* получить соответствующие объектники из файлов формата \*.c)

$< - название первого обязательного условия.

Прописана команда clean, удаляющая исполняемый файл и объектники.

*clean :*

*[tab]rm $(TARGET) \*.o*

Разработанный программный код см. в приложении А.

## Тестирование

Результаты тестирования представлены в табл. 1.

Таблица 1 – Результаты тестирования

|  |  |  |  |
| --- | --- | --- | --- |
| № п/п | Входные данные | Выходные данные | Комментарии |
|  | 0 5 -31 -12 4 18 | 2 | — |
|  | 1 9 17 -23 4 11 6 | 4 | — |
|  | 2 37 42 15 8 -45 9 | 110 | — |
|  | 3 -5 9 4 133 200 15 76 | 105 | — |

## Выводы

В результате выполнения данной работы были достигнуты следующие выводы:

1) Был изучен процесс компиляции и линковки программ на языке Си с использованием компилятора gcc. Были рассмотрены основные этапы этого процесса и принципы работы компилятора.

2) Была изучена структура и правила составления make-файлов. Были рассмотрены основные элементы make-файла, такие как цели, зависимости и команды.

3) Был написан make-файл для сборки заданной программы. В make-файле были указаны цели сборки, зависимости между файлами и команды, необходимые для компиляции и линковки программы.

# Приложение А Исходный код программы

Название файла: Makefile

TARGET = menu

CC = gcc

CFLAGS = -Wall -std=gnu99

all : $(TARGET)

SRC = $(wildcard \*.c)

OBJ = $(patsubst %.c, %.o, $(SRC))

$(TARGET) : $(OBJ)

$(CC) $(CFLAGS) $^ -o $@

%.o : %.c

$(CC) $(CFLAGS) -c $< -o $@

clean :

rm $(TARGET) \*.o

Название файла: menu.c

#include <stdio.h>

#include "index\_first\_even.h"

#include "index\_last\_odd.h"

#include "sum\_between\_even\_odd.h"

#include "sum\_before\_even\_and\_after\_odd.h"

#define ARR\_SIZE 100

void get\_input(char\*, int\*, int[]);

void allocator(char, int[], int);

int main() {

char type;

int array[ARR\_SIZE];

int size = 0;

get\_input(&type, &size, array);

allocator(type, array, size);

return 0;

}

void get\_input(char\* type, int\* size, int array[]) {

scanf("%c", type);

while (getchar() != '\n') {

scanf("%d", &array[(\*size)++]);

}

}

void allocator(char type, int array[], int size) {

switch(type) {

case '0':

printf("%d\n", index\_first\_even(array, size));

break;

case '1':

printf("%d\n", index\_last\_odd(array, size));

break;

case '2':

printf("%d\n", sum\_between\_even\_odd(array, size));

break;

case '3':

printf("%d\n", sum\_before\_even\_and\_after\_odd(array, size));

break;

default:

printf("Данные некорректны\n");

}

}

Название файла: index\_first\_even.c

#include <stdlib.h>

int index\_first\_even(int array[], int size) {

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

if (abs(array[i]) % 2 == 0) return i;

}

return -1;

}

Название файла: index\_last\_odd.c

#include <stdlib.h>

int index\_last\_odd(int array[], int size) {

for (int i = size-1; i >= 0; i--) {

if (abs(array[i]) % 2 != 0) return i;

}

return -1;

}

Название файла: sum\_before\_even\_and\_after\_odd.c

#include <stdlib.h>

#include "index\_first\_even.h"

#include "index\_last\_odd.h"

int sum\_before\_even\_and\_after\_odd(int array[], int size) {

int sum = 0;

for (int i = 0; i < index\_first\_even(array, size); i++) {

sum += abs(array[i]);

}

for (int i = index\_last\_odd(array, size); i < size; i++) {

sum += abs(array[i]);

}

return sum;

}

Название файла: sum\_between\_even\_odd.c

#include <stdlib.h>

#include "index\_first\_even.h"

#include "index\_last\_odd.h"

int sum\_between\_even\_odd(int array[], int size) {

int sum = 0;

for (int i = index\_first\_even(array, size); i < index\_last\_odd(array, size); i++) {

sum += abs(array[i]);

}

return sum;

}

Название файла: index\_first\_even.h

#ifndef INDEX\_FIRST\_EVEN\_H

#define INDEX\_FIRST\_EVEN\_H

int index\_first\_even(int[], int);

#endif

Название файла: index\_last\_odd.h

#ifndef INDEX\_LAST\_ODD\_H

#define INDEX\_LAST\_ODD\_H

int index\_last\_odd(int[], int );

#endif

Название файла: sum\_before\_even\_and\_after\_odd.h

#ifndef SUM\_BEFORE\_EVEN\_AND\_AFTER\_ODD\_H

#define SUM\_BEFORE\_EVEN\_AND\_AFTER\_ODD\_H

int sum\_before\_even\_and\_after\_odd(int[], int);

#endif

Название файла: sum\_between\_even\_odd.h

#ifndef SUM\_BETWEEN\_EVEN\_ODD\_H

#define SUM\_BETWEEN\_EVEN\_ODD\_H

int sum\_between\_even\_odd(int[], int);

#endif