

МИНОБРНАУКИ РОССИИ
САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ
ЭЛЕКТРОТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ
«ЛЭТИ» ИМ. В.И. УЛЬЯНОВА (ЛЕНИНА)

Кафедра МОЭВМ

ОТЧЕТ

по лабораторной работе №1 по
дисциплине «Программирование»

ТЕМА: СОЗДАНИЕ MAKE-ФАЙЛА

Студент гр. 6304

Юсковец
А.В.

Преподаватель

Берленко Т.А.

Санкт-Петербург

2017

Оглавление

Цель работы.....	3
Ход работы.....	3
Вывод.....	5

Цель работы

Создание make-файла к следующему проекту из пяти файлов:

- Файл **get_name.c** должен содержать **описание** функции, которая **считывает** из входного потока имя пользователя и возвращает его.
- Файл **get_name.h** должен содержать **прототип** функции, которая **считывает** из входного потока имя пользователя и возвращает его.
- Файл **print_str.c** должен содержать **описание** функции, которая **принимает** в качестве аргумента строку и выводит её (функция ничего не возвращает).
- Файл **print_str.h** должен содержать **прототип** функции, которая **принимает** в качестве аргумента строку и выводит её (функция ничего не возвращает).
- Файл **main.c** содержит главную функцию, которая вызывает функцию из файла `get_name.h`, добавляет к результату выполнения функции строку " *Hello*, " и передает полученную строку в функцию вывода строки из `print_str.h`.

Ход работы

• get_name.h

```
#include <stdio.h>
#include <stdlib.h>

char* get_name();
```

Рисунок 1: **get_name.h**

Объявил функцию " *get_name* ", возвращающую объект типа " *char** " и не принимающую никаких аргументов. Подключил заголовочные файлы: " *stdio.h* ", " *stdlib.h* ", т.к.

их функции " *getchar* " и " *malloc* " соответственно будут в реализации осуществляемой функции.

• get_name.c

```
#include "get_name.h"

char* get_name() {
    char* name = (char*) malloc(100 * sizeof(char));
    char c = 0;
    int i = 0;

    for (;(c = getchar()) != '\n' && i != 99; ++i) {
        name[i] = c;
    } name[i] = '\0';

    return name;
}
```

Рисунок 2: **get_name.c**

Подключил заголовочный файл " *get_name.h* ", получив доступ к функциям " *getchar* " и " *malloc* ".

Определил переменную " *name* " типа " *char** " присвоив ей указатель на начало области памяти, выделенной под 100 объектов типа " *char* ". Т.к. функция " *malloc* " возвращает " *void** ", явно преобразовал этот тип в " *char** ". Проинициализировал две переменные " *c* " и " *i* " нулем.

В цикле " *for* " программа считывает все символы из входного потока вплоть до символа переноса строки " *\n* " или до момента, как зафиксируется выход из выделенной области памяти (условие такое " *i == 99* ", т. к. необходимо, чтобы осталось место для символа конца строки " *\0* "). В теле цикла каждый следующий символ привязывается к соответствующей ячейке в памяти " *name[i]* ". После конца ввода в конец записывается " *\0* ".

Затем функция возвращает указатель на получившуюся " строку ".

• print_str.h

```
#include <stdio.h>

void print_str(char* s);
```

Рисунок 3: **print_str.h**

Объявил функцию *print_str*, принимающую объект типа " *char** " возвращающую " *void* ". Подключил заголовочный файл " *stdio.h* ", чтобы в дальнейшей реализации объявленной функции использовать

функцию " *putchar* ", выводящую в стандартный поток вывода переданный ей символ.

• **print_str.c**

```
#include "print_str.h"

void print_str(char* s) {
    for(int i = 0; s[i] != '\0'; ++i) {
        putchar(s[i]);
    }
}
```

Рисунок 4: **print_str.c**

Подключил заголовочный файл " *print_str.h* ", получив доступ к функции " *putchar* ".

В цикле проинициализировал нулем целочисленную переменную " *i* " и с ее помощью поочередно выводил в " *stdout* " каждый символ вплоть до конца строки " *\0* ".

• **main.c**

```
#include "get_name.h"
#include "print_str.h"
#include <string.h>

int main() {
    char* name = get_name();
    char hello[107] = "Hello, ";
    strncat(hello, name, 100);
    print_str(hello);
    return 0;
}
```

Рисунок 5: **main.c**

Подключил заголовочные файлы проекта и " *string.h* ", чтобы воспользоваться определенными ранее функциями и " *strncat* " соответственно.

Определил переменную " *name* ", присвоив ей результат работы функции " *get_name* ", массив " *hello* " размера 107 элементов типа " *char* ", (107, т. к. максимальный размер " строки ", которую возвращает " *get_name* " - 100, а длина " *hello* " - 7, и чтобы размера " *hello* " хватило для дозаписи в него всех символов, хранящихся в " *name* ").

С помощью функции " *strncat* " сконкатенировал " *hello* " и " *name* ", записав результат в " *hello* " и выведя его в стандартный поток вывода функцией " *print_str* ".

• Make-файл

```
all: main.o get_name.o print_str.o
    gcc get_name.o print_str.o main.o -o lab1
main.o: main.c get_name.h print_str.h
    gcc -c main.c
get_name.o: get_name.c get_name.h
    gcc -c get_name.c
print_str.o: print_str.c print_str.h
    gcc -c print_str.c
clean:
    rm -f *.o lab1
```

Рисунок 6: Makefile

Создано три цели (main.o, get_name.o, print_str.o) для создания объектных файлов, с использованием флага " -c ", и указанием в зависимости у каждой цели соответствующий ей файл с кодом.

В зависимостях всех целей указаны заголовочные файлы, чтобы соответствующие файлы с кодом компилировались заново при любых изменениях файлов.

Создана цель " all " для компиляции всего проекта, которая вызывает команду " gcc " и линкует все полученные объектные файлы, называя выходную программу " lab1 " (флаг " -o ").

Создана цель " clean " форсировано (" -f ") удаляющая из текущей директории все объектные и исполняемый файлы.

Вывод

Был создан требуемый проект и make-файл к нему. При вызове команды "make" из терминала проект компилируется без ошибок и предупреждений. Программа работает, как требовалось в задании. Во время выполнения не возникает ошибок и предупреждений.