



МИНОБРНАУКИ РОССИИ
Федеральное государственное бюджетное образовательное учреждение
высшего образования
«МИРЭА – Российский технологический университет»
РТУ МИРЭА

Институт информационных технологий
Кафедра вычислительной техники

Отчет по лабораторной работе № 4
по дисциплине
«Инструментальные средства разработки вычислительных
систем»

Тема работы:
«Создание консольного приложения»

Выполнил: студент группы ИВБО-02-19

К. Ю. Денисов

Принял:

И. Р. Сон

Москва 2022

Цель работы

Ознакомится с инструментальными средствами программирования на C/C++.

Задание

В ходе выполнения данной лабораторной работы необходимо написать Makefile, который будет содержать цели для компиляции файла языка C.

Ход работы

Утилита make

Утилита make, входящая в состав практически всех Unix-подобных операционных систем — это традиционное средство, применяемое для сборки программных проектов. Она является универсальной программой для решения широкого круга задач, где одни файлы должны автоматически обновляться при изменении других файлов.

При запуске программа make читает файл с описанием проекта (make-файл) и, интерпретируя его содержимое, предпринимает необходимые действия. Файл с описанием проекта представляет собой текстовый файл, где описаны отношения между файлами проекта, и действия, которые необходимо выполнить для его сборки.

Основным make-файла являются правила (rules). В общем виде правило выглядит так:

```
< цель_1 > ... < цель_n > : < зависимость_1 > ... < зависимость_n >  
    < команда_1 >  
    < команда_2 >  
    ...  
    < команда_n >
```

Цель (target) — это некий желаемый результат, способ достижения которого описан в правиле.

Реализация make-файла

Для выполнения данной практической работы был создан шаблон проекта, содержащего несколько файлов с исходным кодом на языке C и несколько заголовочных файлов.

Листинг 1 — Содержание файлов, используемых в проекте

```
-----
2 ./editor.c
#include "textline.h"
4 -----
./textline.c
6 #include "textline.h"
-----
8 ./st_mode.c
#include <sys/stat.h>
10 #include <limits.h>

12 int main1(int argc, char **argv){
return 0;
14 }
-----
16 ./main.c
#include "main.h"
18 #include "stdio.h"

20 int main (){
printf("%s", "\nThis program was built using GNU make utility\n\n");
22 return 0;
}
```

С целью создать универсальный Makefile были использованы автоматические переменные, описанные в данной таблице.

Таблица 1 — Автоматические переменные make

Имя автоматической переменной	Значение
<code>\$@</code>	Имя цели обрабатываемого правила
<code>\$<</code>	Имя первой зависимости обрабатываемого правила
<code>\$^</code>	Список всех зависимостей обрабатываемого правила

Также с целью удобства построение списка объектных файлов, построение зависимостей от заголовочных файлов и установка целей, было также автоматизировано. Данная методика состоит из двух шагов:

- Получить список всех файлов с исходным текстом программы (всех

файлов с расширением *".cpp"*). Для этого можно использовать функцию *wildcard*.

- Преобразовать список исходных файлов в список объектных файлов (заменить расширение *".cpp"* на расширение *".o"*). Для этого можно воспользоваться функцией *patsubst*.

Перечисление зависимостей "вручную" требует довольно кропотливой работы. Недостаточно просто открыть файл с исходным текстом и перечислить имена всех заголовочных файлов, подключаемых с помощью *#include*. Дело в том, что одни заголовочные файлы могут, в свою очередь, включать в себя другие заголовочные файлы, так что придется отслеживать всю "цепочку" зависимостей.

Процесс построения зависимостей можно автоматизировать, если воспользоваться специальной опцией компилятора GCC. Для совместной работы с *make* компилятор GCC имеет несколько опций:

Таблица 2 — Опции GCC для генерации списка зависимостей

Ключ компиляции	Назначение
<i>-M</i>	Для каждого файла с исходным текстом препроцессор будет выдавать на стандартный вывод список зависимостей в виде правила для программы <i>make</i> . В список зависимостей попадает сам исходный файл, а также все файлы, включаемые с помощью директив <i>#include <имя_файла></i> и <i>#include "имя_файла"</i> . После запуска препроцессора компилятор останавливает работу, и генерации объектных файлов не происходит.
<i>-MM</i>	Аналогичен ключу <i>-M</i> , но в список зависимостей попадает только сам исходный файл, и файлы, включаемые с помощью директивы <i>#include "имя_файла"</i>
<i>-MD</i>	Аналогичен ключу <i>-M</i> , но список зависимостей выдается не на стандартный вывод, а записывается в отдельный файл зависимостей. Имя этого файла формируется из имени исходного файла путем замены его расширения на <i>".d"</i> . Например, файл зависимостей для файла <i>main.cpp</i> будет называться <i>main.d</i> . В отличие от ключа <i>-M</i> , компиляция проходит обычным образом, а не прерывается после фазы запуска препроцессора.
<i>-MMD</i>	Аналогичен ключу <i>-MD</i> , но в список зависимостей попадает только сам исходный файл, и файлы, включаемые с помощью директивы <i>#include "имя_файла"</i>

Для реализации настоящего *make*-файла оптимальной оказывается опция *-MD*, в случае использования которой компиляция проходит в обычном режиме, но кроме этого, *gcc* создаст для каждого исходного файла файл с

расширением ".d", который будет содержать список зависимостей.

С помощью директивы `#include <имя_файла>` в make-файле можно будет подключить сгенерированные файлы зависимостей.

Приведем содержание созданного Makefile.

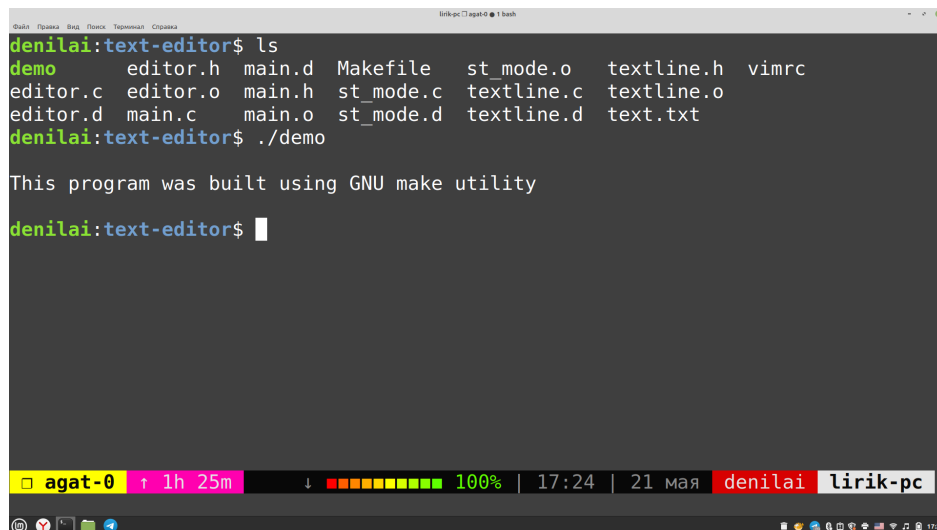
Листинг 2 — Makefile

```
demo: $(patsubst %.c, %.o, $(wildcard *.c))
2 gcc $^ -o $@

%.o: %.c
4 gcc -c -MD $<

6 include $(wildcard *.d)
```

В ходе выполнения команды *make* будет создан исполняемый файл *demo*. Запустив который, можно будет увидеть строку «This program was built using GNU make utility.», что говорит о том, что компиляция и линковка прошла успешно (см. рисунок 1).



```
denilai:text-editor$ ls
demo editor.h main.d Makefile st_mode.o textline.h vimrc
editor.c editor.o main.h st_mode.c textline.c textline.o
editor.d main.c main.o st_mode.d textline.d text.txt
denilai:text-editor$ ./demo

This program was built using GNU make utility

denilai:text-editor$
```

Рисунок 1 — Демонстрация работы программы

Вывод

В ходе настоящей лабораторной работы был создан шаблон консольного приложения, сборка которого производилась с помощью утилиты GNU Make 4.2.1. При написании make-файла были использованы инструменты по автоматизации создания списка зависимостей и целей.