Отчет по лабораторной работе № 25 по курсу

«Практикум программирования»

Студент группы М8О-109Б-22 Концебалов Олег Сергеевич

Контакты: telegram @baronpipistron

Работа выполнена: 11.04.2023

Преподаватель: каф.806 Сысоев Максим Алексеевич

Отчет сдан «11» апреля 2023 г., итоговая оценка ____

Подпись преподавателя _____

1. Тема: Автоматизация сборки программ модульной структуры на языке C++ с использованием утилиты Make

2. Цель работы: Ознакомиться и разобраться с утилитой Make, ее устройством, функциями, научиться писать Makefiles

3. Задание: Описать, что делает уже готовый Makefile: какая цель что делает, что подставляется во флаги компилятора и т.д.

4. Оборудование (студента):

Процессор AMD Ryzen 5 5600H with Radeon Graphics 3.30 GHz, ОП 16,0 Гб, SSD 512 Гб. Монитор 1920x1080 144 Hz

5. Программное обеспечение (студента):

Операционная система семейства Linux, наименование Ubuntu, версия 18.10

Интерпретатор команд: bash, версия 4.4.19

Система программирования – версия --, редактор текстов Emacs, версия 25.2.2

Утилиты операционной системы –

Прикладные системы и программы –

Местонахождение и имена файлов программ и данных на домашнем компьютере –

6. Идея, метод, алгоритм решения задачи *(в формах: словесной, псевдокода, графической [блок-схема, диаграмма, рисунок, таблица] или формальные спецификации с пред- и постусловиями)*

Беру наш готовый Makefile и вместе с гуглом разбираюсь в нем, что за что отвечает и как писать подобный Makefiles.

7. Сценарий выполнения работы *(план работы, первоначальный текст программы в черновике [можно на отдельном листе] и тесты, либо соображения по тестированию)*

1. Более подробно читаю про Makefiles
2. Беру наш Makefile и описываю его

8. Распечатка протокола *(подклеить листинг окончательного варианта программы с тестовыми примерами, подписанный преподавателем)*

1) *Готовый Makefile:*

```
CXX      := g++
CXX_FLAGS := -std=c++17 -Wall -fsanitize=address

BIN     := bin
SRC     := src
INCLUDE := include
LIB     := lib
TESTS   := tests
LIBRARIES   := -lstdc++ -lm
EXECUTABLE  := main
RUN_TEST    := test

build: $(BIN)/$(EXECUTABLE)

run: clean build
	@echo "Executing..."
	$(BIN)/$(EXECUTABLE)

$(BIN)/$(EXECUTABLE): $(SRC)/*.cpp | $(BIN)
	@echo "Building..."
	$(CXX) $(CXX_FLAGS) -I $(INCLUDE) -L $(LIB) $^ -o $@ $(LIBRARIES)

$(BIN)/$(RUN_TEST): $(SRC)/*.cpp $(TESTS)/*.cpp $(TESTS)/*.hpp |$(BIN)
	@echo "Building tests..."
	$(CXX) $(CXX_FLAGS) -I $(INCLUDE) -L $(LIB) $^ -o $@ $(LIBRARIES)

$(BIN):
	mkdir $@
```

```
clean:
	@echo "Clearing..."
	-rm -r $(BIN)

_test: $(BIN)/$(RUN_TEST)

test: clean _test
	@echo "Run tests..."
	$(BIN)/$(RUN_TEST)
```

2) Описание Makefile'а:

Что такое Makefile:

Makefile — это набор инструкция для программы make, которая позволяет собирать программный проект буквально в одно действие. То есть он позволяет одновременно компилировать и выполнять несколько программ сразу, а не поочередно каждую, как это делается руками

Makefile из примера:

CXX

Определяет компилятор C++, который мы используем для сборки нашего проекта. В нашем случае используется компилятор g++ (CXX := g++)

CXX_FLAGS

Определяет набор флагов компилятора. Мы выбираем стандарт языка (C++17), включаем все предупреждения комилятора, что поможет нам найти проблемы в коде (-Wall), отлавливаем ошибки памяти, например утечки (fsanitize=adsress)

Потом идут переменные, которые определяют следующие директории:

- BIN := bin — директория, в которой содержатся бинарные файлы. Бинарные файлы — это файлы, которые содержат машинный код, уже понятный самому компьютеру. В нашем случае — это исполняемые файлы и файлы библиотек, которые создадутся в результате компиляции нашей программы.
- SRC := src — директория, в которой содержатся исходные коды проекта. В ней находятся файлы с расширениями .cpp
- INCLUDE := include — директория, в которой содержатся подключаемые заголовочные файлы. Они содержат объявления классов, структур, констант, функций и , как правило, имеют расширение .hpp
- LIB := lib — директория, в которой содержатся все используемые библиотеки. В ней могут быть размещены как стандартные библиотека, так и кастомные, созданные разработчиками
- TESTS := tests — директория, в которой содержатся тестовые файлы

- LIBRARES := -lstdc++ -lm — директория, в которой содержится список библиотек, которые должны бать связаны с бинарным файлом. Они компилируются отдельно от основного кода и связываются с ним во время сборки
- EXECUTABLE := main — имя исполняемого файла
- RUN_TEST := test — макрос, который используется для запуска тестовых файлов. Вызывает функцию, содержащую набор тестов и выводит их результат. Если тест не пройден, то выводит сообщение об ошибке

build: $(BIN)/$(EXECUTABLE)

В этом таргете мы собираем исполняемый файл из исходников, даем ему имя EXECUTABLE, а после помещаем его в директорию бинарных файлов BIN. Благодаря этому компилятор точно знает, куда поместить готовый исполняемый файл после сборки

run: clean build

В этом таргете мы удаляем временные файлы проекта, потом компилируем все исходники и создаем исполняемый файл. Т.е. на выходе получим запущенную программу, собранную из исходников проекта. Ну и печатаем Executing..., т.е. Выполнение...

$(BIN)/$(EXECUTABLE): $(SRC)/*.cpp | $(BIN)

В таргете указываем то, что надо собрать, т.е. в нашем случае путь к исполняемому файлу из BIN и EXECUTABLE.

В зависимости указываем все исходники проекта, которые записаны в директории SRC и имеют расширение .cpp

Так же в зависимости есть | $(BIN). Как я понял эта штука говорит создать BIN, если его еще нет

Печатаем Building…, т.е. Сборка...

$(CXX) $(CXX_FLAGS) -I $(INCLUDE) -L $(LIB) $^ -o $@ $(LIBRARIES)

Решил рассмотреть эту строчку отдельно, т.к. встречается в нескольких местах.

По сути, здесь пишем консольную команду в терминале линукса, т.е. она имеет вид

g++ -std=c++17 -Wall -fsanitize=address -I include -L lib *подтягиваем все зависимости* -o *для всех целей и библиотек*

Т.е. команда описывает как должны компилироваться все исходные файлы, создаваться объектные, получение исходных и их сохранение в директорию BIN

Здесь есть два новых для меня флага -I, который указывает компилятору путь к директории с заголовочными файлами, и -L, который указывает на процессе линковки путь к директориям, в которых указаны используемые библиотеки

$(BIN)/$(RUN_TEST): $(SRC)/*.cpp $(TESTS)/*.cpp $(TESTS)/*.hpp |$(BIN)

Делаем все тоже самое, что и в ***$(BIN)/$(EXECUTABLE): $(SRC)/*.cpp | $(BIN)***, но путь формируем из BIN и RUN_TESTS, а в зависимости указываем не только все исходные файлы, но еще и исходники тестовых файлов с расширением .cpp и их заголовочные файлы с расширением .hpp.

Так же в зависимости печатаем Building tests..., т.е. Сборка тестов...

И есть строка рассмотренная выше

$(BIN):

Говорит создать директорию BIN, если она еще не создана

В зависимости пишем, чтобы директория создалась для всех целей ($@)

Гарантирует, что все скомпилированные файлы поместятся в отдельную папку и не засорят рабочую область

clean:

Здесь очищаем директорию BIN

С помощью -rm удаляем файлы, с помощью символа – перед rm избегаем вывода сообщения об ошибке, если файл не существует, выполнение программы продолжится

Это полезно чтобы очистить директорию BIN перед новой компиляцией и избежать конфликтов между старыми и новыми файлами

И печатает Cleaning…, т.е. Очистка...

_test: $(BIN)/$(RUN_TEST)

Здесь компилируем запускаем тестовые файлы, путь формируем из BIN и RUN_TESTS

Удобно для автоматизации тестирования программы

test: clean _test

Как понял эта цель зависит от выполнения ***clean*** и ***_test.*** Когда они выполнятся выведем Run tests… и будем запускать тестовые файлы, пути к которым формируем из BIN и RUN_TESTS

Таким образом здесь мы автоматизируем тестирование

9. Дневник отладки *(дата и время сеансов отладки и основные события [ошибки в сценарии и программе, нестандартные ситуации] и краткие комментарии к ним. В дневнике отладки приводятся сведения об использовании других ЭВМ, существенном участии преподавателя и других лиц в написании и отладке программы)*

№	*Лаб. или дом*	*Дата*	*Время*	*Событие*	*Действие по исправлению*	*Примечания*

Отсутствуют

10. Замечания автора *(по существу работы)*

Замечания отсутствуют

11. Вывод

Довольно интересная лаба. Познакомился с новым для себя Makefile’ом, узнал про него много нового, пока разбирался что за что отвечает. Оказалось, что это довольно

удобная вещь для автоматизации, не придется запускать каждый код отдельно. Но было бы хорошо написать его самому, для большего понимания. Все-таки теория хорошо, но практика всегда гораздо лучше. Буду стараться юзать Makefiles для своих новых проектов и лаб

Работа на 9/10

Подпись студента _____