

Министерство образования Республики Беларусь
Учреждение образования «Белорусский государственный университет
информатики и радиоэлектроники»

Факультет компьютерных систем и сетей

Кафедра информатики

Дисциплина: Операционные среды и системное программирование

ОТЧЁТ
к лабораторной работе №3
на тему

**ОСНОВЫ ПРОГРАММИРОВАНИЯ НА С ПОД UNIX.
ИНСТРУМЕНТАРИЙ ПРОГРАММИСТА В UNIX**

Студент: гр.153502
Миненков Я. А.

Проверил: Гриценко Н. Ю.

Минск 2024

СОДЕРЖАНИЕ

1 Формулировка задачи	3
2 Теоретические сведения.....	4
3 Описание функций программы	5
Список использованных источников.....	7
Приложение А (обязательное) Исходный код программы.....	8

1 ФОРМУЛИРОВКА ЗАДАЧИ

Целью выполнения лабораторной работы является изучение среды программирования и основных инструментов: компилятор/сборщик («коллекция компиляторов») *gcc*, управление обработкой проекта *make* (и язык *makefile*), библиотеки. Практическое использование основных библиотек и системных вызовов: ввод-вывод и работа с файлами, обработка текста, распределение памяти, управление выполнением.

В качестве задачи требуется написать программу на языке программирования C, генерирующую множество строк, заданных регулярным выражением. Регулярные выражения ограничены поддержкой только символьных классов «[...]» и «необязательных» символов.

2 ТЕОРЕТИЧЕСКИЕ СВЕДЕНИЯ

GCC – это набор инструментов, который компилирует код, связывает его с зависимостями библиотек, преобразует этот код в ассемблер и затем подготавливает исполняемые файлы. Он следует стандартной философии дизайна *UNIX*, используя простые инструменты, которые хорошо выполняют отдельные задачи. Набор разработки *GCC* использует эти отдельные инструменты для компиляции программного обеспечения. При запуске *GCC* на файле исходного кода он сначала использует препроцессор для включения заголовочных файлов и удаления комментариев. Затем он токенизирует код, расширяет макросы, обнаруживает любые проблемы времени компиляции, затем подготавливает его к компиляции. Затем он отправляется компилятору, который создает синтаксические деревья объектов программы и контроля потока и использует их для генерации кода на ассемблере. Ассемблер затем преобразует этот код в двоичный исполняемый формат системы. Наконец, связыватель включает ссылки на внешние библиотеки по мере необходимости. Готовый продукт затем может быть выполнен в целевой системе.

Для компиляции и запуска проекта на языке *C* используется утилита *make*.

Makefile интерпретируются и запускаются с помощью инструментов *make* в *Unix/Linux*.

Чаще всего файлы *Makefile* используются для управления зависимостями исходных файлов программ на этапе компиляции и связывания (сборки), то есть для компиляции только тех файлов, которые необходимо скомпилировать, просматривая зависимости друг от друга и даты последних изменений исходных файлов во время компиляции программ.

Чтобы подготовиться к использованию *make*, необходимо создать *make*-файл, который объясняет взаимосвязи между файлами программы и упорядочивает команды. Исполняемый файл программы обычно обновляется из объектных файлов, которые создаются путем компиляции исходных файлов с расширением *.o*.

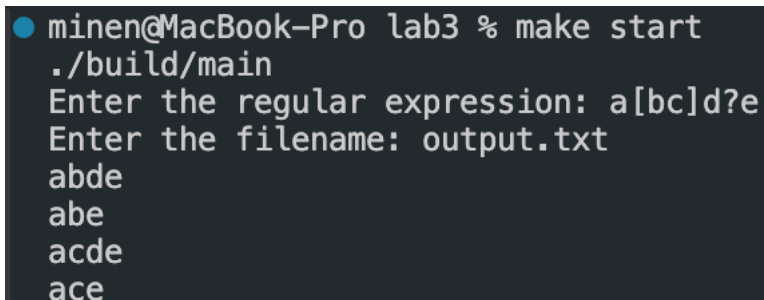
3 ОПИСАНИЕ ФУНКЦИЙ ПРОГРАММЫ

Программа обеспечивает генерацию строк на основе заданного шаблона. Шаблон может содержать следующие специальные символы:

[...] – генерирует все возможные строки, заменяя этот символ на каждый символ в скобках. Например, шаблон *a[bc]* приведет к генерации строк *ab* и *ac*

? – генерирует две строки: одну со следующим символом и одну без него. Шаблон *a?b* приведет к генерации строк *ab* и *b*.

На рисунке 1 демонстрируется выполнение программы.



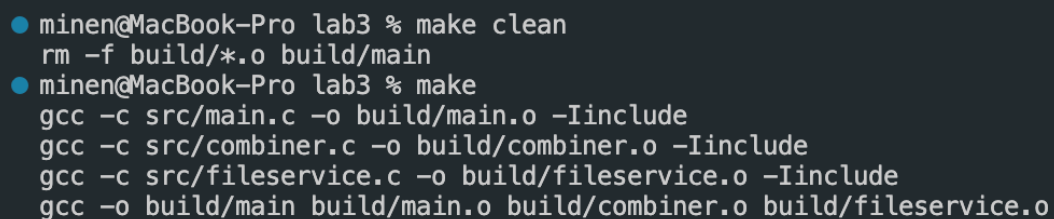
```
minen@MacBook-Pro lab3 % make start
./build/main
Enter the regular expression: a[bc]d?e
Enter the filename: output.txt
abde
abe
acde
ace
```

Рисунок 1 – Выполнение программы

Результаты выполнения программы записываются в указанный файл.

Для автоматизации процесса сборки был создан *Makefile*, который используется утилитой *make*. В нем описаны правила, которые определяют, как из исходного кода получить исполняемый файл. *Makefile* включает в себя три цели: *all*, *clean*, *start*. *All* определяет цель по умолчанию. *Clean* определяет цель *clean*, которая удаляет все сгенерированные файлы. Цель *start* запускает собранную программу.

Пример очистки и сборки проекта с помощью *make* (рисунок 2).



```
minen@MacBook-Pro lab3 % make clean
rm -f build/*.o build/main
minen@MacBook-Pro lab3 % make
gcc -c src/main.c -o build/main.o -Iinclude
gcc -c src/combiner.c -o build/combiner.o -Iinclude
gcc -c src/fileservice.c -o build/fileservice.o -Iinclude
gcc -o build/main build/main.o build/combiner.o build/fileservice.o
```

Рисунок 2 – Очистка и сборка проекта

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

[1] GNU Compiler Collection [Электронный ресурс]. – Режим доступа: <https://www.incredibuild.com/integrations/gcc>. – Дата доступа: 03.03.2024.

[2] Makefile and make [Электронный ресурс]. – Режим доступа: <https://medium.com/@ayogun/what-is-makefile-and-make-how-do-we-use-it-3828f2ee8cb>. – Дата доступа: 03.03.2024.

ПРИЛОЖЕНИЕ А

(обязательное)

Исходный код программы

Файл combiner.c

```
#include "combiner.h"
#include <stdio.h>
#include <string.h>
#include <stdlib.h>

char* generate_string(char* pattern, char* prefix) {
    if (pattern[0] == '\\0') {
        return strdup(prefix);
    } else if (pattern[0] == '[') {
        char* end = strchr(pattern, ']');
        char* result = malloc(256);
        result[0] = '\\0';
        for (char* ch = pattern + 1; ch != end; ++ch) {
            char new_prefix[256];
            strcpy(new_prefix, prefix);
            strncat(new_prefix, ch, 1);
            char* generated_string = generate_string(end + 1,
new_prefix);

            strcat(result, generated_string);
            strcat(result, "\\n");
            free(generated_string);
        }
        return result;
    } else if (pattern[1] == '?') {
        char new_prefix[256];
        strcpy(new_prefix, prefix);
        strncat(new_prefix, pattern, 1);
        char* result1 = generate_string(pattern + 2, new_prefix);
        char* result2 = generate_string(pattern + 2, prefix);
        char* result = malloc(strlen(result1) + strlen(result2) + 2);
        strcpy(result, result1);
        strcat(result, "\\n");
        strcat(result, result2);
        free(result1);
        free(result2);
        return result;
    } else {
        char new_prefix[256];
        strcpy(new_prefix, prefix);
        strncat(new_prefix, pattern, 1);
        return generate_string(pattern + 1, new_prefix);
    }
}

void print_string(char* str) {
    printf("%s\\n", str);
}
```

```
}
```

Файл fileservice.c

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include "fileservice.h"

void writeToFile(const char* filename, const char* content) {
    char outputFilePath[100];
    snprintf(outputFilePath, sizeof(outputFilePath), "output/%s",
filename);

    FILE* file = fopen(outputFilePath, "w");
    if (file == NULL) {
        printf("Failed to open file.\n");
        return;
    }
    fprintf(file, "%s", content);
    fclose(file);
}
```

Файл main.c

```
#include <stdio.h>
#include <stdlib.h>
#include "combiner.h"
#include "fileservice.h"

int main() {
    char regex[100];
    char filename[100];
    printf("Enter the regular expression: ");
    scanf("%s", regex);
    printf("Enter the filename: ");
    scanf("%s", filename);
    char *result = generate_string(regex, "");
    writeToFile(filename, result);
    return 0;
}
```