ЧЕРНОВИК

1. Работа с текстовыми файлами

Файл представляет собой последовательность байт. В общем случае любой файл является *двоичным* (или *бинарным*). Но если файл содержит только печатные символы (т.е. символы которые можно вводить с клавиатуры), его называют *текстовым*.

Текстовый файл состоит из строк, каждая из которых заканчивается маркером конца строки. Благодаря этим маркерам вы видите текст файла разделенным на строки, если открываете его в текстовом редакторе. В конце файла иногда располагается маркер конца файла.

Вы не можете видеть маркеры конца строк, потому что они являются «сигналами» для программы (в нашем случае – текстового редактора) о переходе на новую строку. В разных операционных системах используются разные маркеры конца строк: Windows – 0x0D, 0x0A; Linux – 0x0A; Mac OS – 0x0D.

Работа с файлами в языке Си выполняется с помощью *файловых указателей*, которые имеют тип **`FILE`**, определяемый в заголовочном файле stdio.h.

Функция fopen открывает файл. Она получает два аргумента – строку с именем файла и строку с режимом доступа к файлу. Имя файла может быть как абсолютным, так и относительным. fopen возвращает файловый указатель, с помощью которого далее можно осуществлять доступ к файлу. Если вызов функции fopen прошёл неудачно, то она возвратит NULL.

```
#include <stdio.h>

FILE* fopen(const char* filename, const char* mode);
```

Некоторые режимы открытия файла.

Режим (mode)	Описание
"r"	Чтение. Файл должен существовать.
"w"	Запись. Если файл с таким именем не существует, он будет создан, в противном случае его содержимое будет потеряно.
"a"	Запись в конец файла. Файл создаётся, если не существовал.

Функция fopen может открывать файл в текстовом или бинарном режиме. По умолчанию используется текстовый режим. Если необходимо открыть файл в бинарном режиме, то в конец строки добавляется буква b, например "rb", "wb", "ab".

Функция fclose закрывает файл. Она получает один аргумент – файловую переменную.

```
#include <stdio.h>

int fclose(FILE* f);
```

Функция fscanf аналогична по смыслу функции scanf, но в отличие от нее осуществляет форматированный ввод из файла.

```
#include <stdio.h>

int fscanf(FILE* f, const char *str, ...);
```

Функция fscanf возвращает количество удачно считанных данных или признак конца файла EOF.

Допустим, есть текстовый файл, который содержит целые числа. Необходимо найти максимальное среди этих чисел.

```
#include <stdio.h>

int main(void)
{
    FILE *f;
    int num, max;

    f = fopen("test.txt", "r");
    if (f == NULL)
    {
        printf("I/O error\n");

        return -1;
    }

    if (fscanf(f, "%d", &max) == 1)
    {
        while (fscanf(f, "%d", &num) == 1)
            if (num > max)
                max = num;

        printf("max is %d\n", max);
    }
    else
        printf("There are not enough data.\n");

    fclose(f);

    return 0;
}
```

При запуске программы автоматически создаются три файловые переменные с именами stdin, stdout и stderr. Переменная stdin связана с клавиатурой, а переменные stdout и stderr – с экраном. Эти переменные можно использовать в любом месте, где можно использовать переменную типа **FILE***.

```
...

float a, b;

if (fscanf(stdin, "%f%f", &a, &b) != 2)
    fprintf(stderr, "I/O error\n");
else

...
```

2. Параметры командной строки

Материал данного раздела частично приводится по книге С.В. Шапошниковой «Особенности языка Си».

Часто данные передаются в программу из командной строки при ее запуске. Например,

```
# gcc.exe –std=c99 -Wall -Werror -o args.exe args.c
```

Здесь запускается программа gcc.exe, которая из командной строки получает шесть аргументов: -std=c99, -Wall, -Werror, -o, args.exe и args.c.

Если программа написана на языке Си, то после ее запуска управление передается в функцию main, которая и получает аргументы командной строки:

```
int main(int argc, char** argv);
```

Данный вариант функции main получает два параметра:

- целое число (argc), обозначающее количество аргументов (элементов, разделенных пробелами) в командной строке при вызове (следует иметь в виду, что само имя программы также учитывается),
- указатель на массив строк (argv), где каждая строка - это отдельный аргумент из командной строки.

Для приведенного выше примера значение argc равно 7, а массив строк argv определяется как {«gcc.exe», «-std=c99», «-Wall», «-Werror», «-o», «args.exe», «args.c», NULL}.

То, что в программу передаются данные, не означает, что функция main должна их обрабатывать. Если функция main определена без параметров (`int main(void)`), то получить доступ к аргументам командной строки невозможно, но при этом вы можете указывать их при запуске.

Изучите вывод программы, приведенной ниже, запуская ее с различными аргументами командной строки.

```
#include <stdio.h>

int main(int argc, char **argv)
{
    printf("argc = %d\n", argc);

    for (int i = 0; i < argc; i++)
        puts(argv[i]);

    return 0;
}
```

Замечание
Имена argc и argv не являются обязательными (т.е. вы может использовать любые), но лучше придерживаться именно этих имен, чтобы ваши программы были более понятны не только вам, но и другим программистам.

Чаще всего в программу при запуске передаются имена файлов и «опции» (ключи), которые влияют на процесс выполнения программы. Рассмотрим пример программы, которая

получает имя обрабатываемого файла через аргументы командной строки.

```
#include <stdio.h>

int main(int argc, char** argv)
{
    FILE *f;
    int num;

    if (argc != 2)
    {
        fprintf(stderr, "num_reader.exe file-name\n");

        return -1;
    }

    f = fopen(argv[1], "r");
    if (f == NULL)
    {
        fprintf(stderr, "I/O error\n");

        return -2;
    }

    while (fscanf(f, "%d", &num) == 1)
        printf("%d\n", num);

    fclose(f);

    return 0;
}
```

3. Утилита gcov

Утилита *gcov* используется (в том числе) для анализа покрытия кода тестами. Информация, которая получается в результате работы этой утилиты, позволяет ответить на следующие вопросы:

- как часто выполняется каждая строка кода;
- какие строки кода не выполняются;
- какой процент строк кода покрыт тестами (для каждого файла);
- какой процент ветвлений покрыт тестами (для каждого файла).

gcov обрабатывает программы, которые были получены только с помощью компилятора *gcc*. При сборке программы желательно выключить оптимизацию с помощью ключа «-O0» и добавить следующие ключи «-fprofile-arcs» и «-ftest-coverage».

Два последних ключа вынуждают компилятор добавить в исполняемый файл программы дополнительный код, который во время ее выполнения собирает статистическую информацию и сохраняется ее в служебные файлы. *gcov* использует эти файлы для создания «аннотированного» листинга исходного кода программы, который содержит информацию о частоте выполнения каждой строки.

Рассмотрим шаги использования утилиты *gcov* на примере исследования программы из Приложения А.

Шаг 1. Компиляция программы

```
# c99 -Wall -Werror -pedantic -O0 -fprofile-arcs -ftest-coverage sq.c -o sq.exe
```

Шаг 2. Запуск программы
Например, так

```
# sq.exe
Enter a, b, c: 1 2 1
x1 = x2 = -1.000000
```

Шаг 3. Запуск утилиты gcov

Например, так

```
# gcov sq.c
File 'sq.c'
Lines executed:69.23% of 13
Creating 'sq.c.gcov'

File 'c:/mingw/include/stdio.h'
Lines executed:100.00% of 5
Creating 'stdio.h.gcov'
```

Утилита gcov выдаст на экран указанную информацию и создаст «аннотированный» листинг sq.c.gcov (см. Приложение Б). Возможно, что «аннотированный» листинг будет включать в себя несколько файлов.

Изучим содержание «аннотированного» листинга. Он начинается «преамбулой», каждая строка которой имеет вид:

```
-:0:<tag>:<value>
```

Из «преамбулы» нас будет интересовать количество запусков программы:

```
        -:    0:Runs:1
```

У остальных строк «аннотированного» листинга формат следующий:

```
<количество выполнений строки>:<номер строки>:<строка исходного кода>
```

Для строк, которые не содержат код, количество выполнений обозначается символом «-», а для строк, которые не выполнялись ни разу, - «#####».

Повторяя шаги 2 и 3 для разных входных данных и анализируя «аннотированный» листинг, можно добиться полного покрытия кода тестами.

Замечание 1
Следует помнить, что даже 100% покрытие кода тестами не означает, что в программе нет ошибок!

Замечание 2
Ключ «-a» дополнит «аннотированный» листинг информацией о выполнении всех базовых блоков каждой строки программы, а не только основного.

Замечание 3
Ключ «-b» дополнит «аннотированный» листинг информацией статистикой о выполнении условных операторов.

Приложение А

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

int main(void)
{
    float a, b, c, d;

    printf("Enter a, b, c: ");
    if (scanf("%f%f%f", &a, &b, &c) == 3)
    {
        if (a != 0.0)
        {
            d = b * b - 4 * a * c;

            if (d < 0.0)
            {
                printf("There are no real roots\n");
            }
            else if (d > 0)
            {
                printf("x1 = %f, x2 = %f\n", (-b - sqrt(d)) / (2 * a),
                                             (-b + sqrt(d)) / (2 * a));
            }
            else
            {
                printf("x1 = x2 = %f\n", -b / (2 * a));
            }
        }
        else
        {
            printf("Equation is not square\n");
        }
    }
    else
    {
        printf("I/O error\n");
    }

    return 0;
}
```

Приложение Б

```
        -:    0:Source:sq.c
        -:    0:Graph:sq.gcno
        -:    0:Data:sq.gcda
        -:    0:Runs:1
        -:    0:Programs:1
        -:    1:#include <stdio.h>
        -:    2:#include <math.h>
        -:    3:
        1:    4:int main(void)
        -:    5:{
        -:    6:    float a, b, c, d;
        -:    7:
        1:    8:    printf("Enter a, b, c: ");
        1:    9:    if (scanf("%f%f%f", &a, &b, &c) == 3)
        -:   10:    {
        1:   11:        if (a != 0.0)
        -:   12:        {
        1:   13:            d = b * b - 4 * a * c;
        -:   14:
        1:   15:            if (d < 0.0)
        -:   16:            {
    #####:   17:                printf("There are no real roots\n");
        -:   18:            }
        1:   19:            else if (d > 0)
        -:   20:            {
    #####:   21:                printf("x1 = %f, x2 = %f\n", (-b - sqrt(d)) / (2 * a),
    #####:   22:                                             (-b + sqrt(d)) / (2 * a));
        -:   23:            }
        -:   24:            else
        -:   25:            {
        1:   26:                printf("x1 = x2 = %f\n", -b / (2 * a));
        -:   27:            }
        -:   28:        }
        -:   29:        else
        -:   30:        {
    #####:   31:            printf("Equation is not square\n");
        -:   32:        }
        -:   33:    }
        -:   34:    else
        -:   35:    {
    #####:   36:        printf("I/O error\n");
        -:   37:    }
        -:   38:
        1:   39:    return 0;
        -:   40:}
```