Last edited by Mansurov Vladislav 1 month ago

Page history New page

dz_2

Написание кода

Исходя из черного ящика и по псевдокоду был написал код программы, который был декомпозирован согласно сделанной диаграммы на Ramus Education в файле main.c:

- Основной код программы то есть функция int main(void)

```
int main(void)
{
        FILE *my_file;
        my_file = fopen("data.txt", "r");
        int arr[MAX_NUMBERS];
        size_t n, nb;
        int sum, reverse, rc = EXIT_SUCCESS;
        int binarycod[MAX_NUMBERS];


        rc = read_array(my_file, arr, &n);

        printf("Прочитанные числа, переведены в двоичный код или перевернуты:\n");
        for (size_t i = 0; i < n; i++)
        {
                sum = sum_of_figure(arr[i]);
                if (check_sum(sum))
                {
                        converting_to_binarycod(arr[i], binarycod, &nb);
                        for (int i = 0; i < nb; i++)
                        {
                                if (i == 0)
                                {
                                        printf(" %d", binarycod[i]);
                                } else
                                {
                                        printf("%d", binarycod[i]);
                                }
                        }
                } else
                {
                        reverse_number(arr[i], &reverse);
                        printf(" %d", reverse);
                }

        }
        fclose(my_file);
        return rc;
}
```

- Функция read_array()

```
int read_array(FILE *f, int a[MAX_NUMBERS], size_t *n)
{
        int num, error = EXIT_SUCCESS;
        char sym;
        *n = 0;

        while (fscanf(f, "%d%c", &num, &sym) == 2 || fgetc(f) != EOF)
        {

                a[*n] = num;
                (*n)++;
        }
        return error;
}
```

- Функция sum_of_figure

```
int sum_of_figure(int number)
{
        int sum = 0;
        number = abs(number);

        while (number > 0)
        {
                sum += number % 10;
                number /= 10;
        }

        return sum;
}
```

- Функция reverse_number

```
void reverse_number (int number, int *reverse)
{
        *reverse = 0;

        while (abs(number) > 0)
        {
                *reverse = (*reverse) * 10 + number % 10;
                number /= 10;
        }
}
```

- Функция sum_of_figure

```
int sum_of_figure(int number)
{
        int sum = 0;
        number = abs(number);

        while (number > 0)
        {
                sum += number % 10;
                number /= 10;
        }

        return sum;
}
```

- Функция converting_to_binarycod

```
void converting_to_binarycod(int number, int b[MAX_NUMBERS], size_t *nb)
{
        *nb = 0;
        int sign;
        if (number < 0)
        {
                sign = 1;
        } else
        {
                sign = 0;
        }
        while (abs(number) > 0)
        {
                if (number % 2 == 0)
                {
                        b[*nb] = 0;
                }else
                {
                        b[*nb] = 1;
                }
                number /= 2;
                (*nb)++;
        }
        b[(*nb)++] = sign;
        int buff, i = *nb - 1;
        while (i >= *nb / 2 && *nb != 1)
        {
                buff = b[*nb - 1 - i];
                b[*nb - 1 - i] = b[i];
                b[i] = buff;
                i--;
        }
}
```

Полный код доступен по номеру ревизии: 40cc3d4b

Как ожидалось программа работа только лишь в валидарных данных, если же передать не существующий файл или пустой или же некорректный то программа выводит уже неправильный результат или же приводит к аварийному выключению.

Добавления asserts и ошибок программы

Для тестирование программы на ошибки в код функций в особенности в функцию read_array() были добавлены assets и условия передачи файла в него если файл несуществующий или пустой то же данные не передаются в функцию:

```
...
#define FULL_MAX        -2
#define NOT_FOUND_FILE  -3
#define FILE_IS_EMPTY   -4
#define IN_FILE_SYMBOLS -5
#define IN_FILE_DOUBLE  -6
...

int main(void)
{
        FILE *my_file;
        my_file = fopen("data.txt", "r");
        int arr[MAX_NUMBERS];
        size_t n, nb;
        int sum, reverse, rc = EXIT_SUCCESS;
        int binarycod[MAX_NUMBERS];

        if (my_file == NULL)
                return NOT_FOUND_FILE;
        rc = read_array(my_file, arr, &n);

        ...

        fclose(my_file);
        return rc;
}

int read_array(FILE *f, int a[MAX_NUMBERS], size_t *n)
{
        int num, error = EXIT_SUCCESS;
        char sym;
        *n = 0;

        assert(f != NULL);

        int result = fscanf(f, "%d%c", &num, &sym);

        while (result == 2 || fgetc(f) != EOF)
        {
                assert((sym == ' ' || sym == '\n') && (sym != ',' || sym != '.'));
            a[*n] = num;
                (*n)++;
                assert(*n <= MAX_NUMBERS);
                result = fscanf(f, "%d%c", &num, &sym);
        }
        assert(*n > 0);
        return error;
}
```

Здесь же программа стала вести себя иначе при появление некорректных данных выводились ASSERTS FAILED с условием и на какой строке кода это произошло, так можно было дать знать разработчику, где именно проявляется та или иная ошибка и где нужно поставить условия для того чтобы вывести ошибочный результат и сообщение пользователю об ошибке. Asset-ы были хороши для использования в проектах больших команд, так к примеру тестировщик может указать разработчику на ошибки в частях кода (функция), если он это не предусмотрел, но их надо писать так чтобы после того избавления их из кода программа не должна ломаться.

Добавление условий для ошибок

И так для того чтобы полностью привести программу в рабочее состояние вместо assert-ов добавил условия для ошибок и таким образом при различных некорректных данных программа сообщает пользователю что не так он передал.

Добавление функции print_error - вывода сообщения об ошибке взависимости по номеру ошибки

```
...
int main(void)
{
...
}
...
int read_array(FILE *f, int a[MAX_NUMBERS], size_t *n)
{
        int num, error = EXIT_SUCCESS;
        char sym;
        *n = 0;

        if (f == NULL)
        {
                error = NOT_FOUND_FILE;
        } else
        {
                int result_1 = fscanf(f, "%d", &num);
                int result_2 = fscanf(f, "%c", &sym);

                while (result_1 == 1 || result_2 == 1 || fgetc(f) != EOF)
                {
                        if (sym == '.' || sym == ',')
                        {
                                error = IN_FILE_DOUBLE;
                                (*n)++;
                                break;
                        }

                        if ((sym != ' ' && sym != '\n' && (result_1 + result_2 == 1)) || (result_1 != 1 &&
                        {
                                error = IN_FILE_SYMBOLS;
                                (*n)++;
                                break;
                        }

                        a[*n] = num;
                        (*n)++;

                        if (*n > MAX_NUMBERS)
                        {
                                error = FULL_MAX;
                                break;
                        }

                        result_1 = fscanf(f, "%d", &num);
                        result_2 = fscanf(f, "%c", &sym);
                }
                if (*n == 0)
                        error = FILE_IS_EMPTY;
        }
        return error;
}

...
void print_error(int rc)
{
        if (rc == NOT_FOUND_FILE)
                printf("\nФайл не найден.");
        if (rc == FILE_IS_EMPTY)
                printf("\nПустой файл.");
        if (rc == FULL_MAX)
                printf("\nВ файле больше 100.");
        if (rc == IN_FILE_DOUBLE)
                printf("\nФайл записан некорректно. Содержит вещественные числа.");
        if (rc == IN_FILE_SYMBOLS)
                printf("\nФайл записан некорректно. Содержит буквы и символы.");
        if (rc == ERR_INVALID_PARAM)
                printf("\nВ функцию переданы некорректные параметры.");
}
```

Номер ревизии: d7939430

И при тестировании программы на разные случаи согласно тестам из черного ящика:

Исх. файл	Вывод в консоль
Несуществующий файл	Файл не найден
Пустой файл	Пустой файл
В файле больше 100 чисел	В файле больше 100 чисел. Превышен предел.
a \ ] c y 7	Файл записан некорректно. Содержит буквы и символы.
1.23 6.78 9.45 10.0	Файл записан некорректно. Содержит вещественные числа.
-231 121 1234	111100111 01111001 010011010010
120 -313 975	12 -313 579
234 222 -234	432 011011110 -432
-342 -234 -121	-243 -432 11111001
0	0
121 1234 120 313 975 450 0 1	01111001 010011010010 12 313 579 54 0 1

Модульное тестирование

Для каждой функции программы были написаны тесты согласно примерным тестам расписанных в первой части ДЗ где к каждому модулю были указаны тесты, а результат их успеха или неудачи выводится консоль, так для этого был создан отдельный файл unit_test.c и модуль для подключения unit_test.h так как в тестах вызывились все функции из main.c то был создан main.c для подключения определений функций в unit_test.c

- Файл unit_tests.c

Тесты были написаны следующим образом название *test*<название тестируемой функции>_ внутри объявлялись переменные для подсчета неправильных тестов, количества тестов.

В результате в консоль выводилось:

```
USER UNIT TEST:
===tests_read_array:
 -Негативные тесты: 5 of 5 - SUCCESS
 -Позитивные тесты: 2 of 2 - SUCCESS
===tests_sum_of_figure:
 -Позитивные тесты: 5 of 5 - SUCCESS
===tests_check_sum:
 -Позитивные тесты: 4 of 4 - SUCCESS
===tests_converting_to_binarycod:
 -Позитивные тесты: 5 of 5 - SUCCESS
===tests_reverse_number:
 -Позитивные тесты: 5 of 5 - SUCCESS
```

```
void tests_read_array(void)
{
        int neg_cnt = 0, pos_cnt = 0, test_neg = 5, test_pos = 2;
        //
        // Передача пустого файла или несуществующего файла
    //
        {
                int a[MAX_NUMBERS];
                size_t n;

                if (read_array(NULL, a, &n) != NOT_FOUND_FILE)
            neg_cnt++;

                if (read_array(fopen("unit_tests/neg_tests_1.txt", "r"), a, &n) != FILE_IS_EMPTY)
                        neg_cnt++;

        }
        //
        // Передача некорректно записанных файлов
        //
        {
                int a[MAX_NUMBERS];
                size_t n;

                if (read_array(fopen("unit_tests/neg_tests_2.txt", "r"), a, &n) != IN_FILE_SYMBOLS)
            neg_cnt++;
                if (read_array(fopen("unit_tests/neg_tests_3.txt", "r"), a, &n) != IN_FILE_DOUBLE)
                        neg_cnt++;
                if (read_array(fopen("unit_tests/neg_tests_4.txt", "r"), a, &n) != FULL_MAX)
                        neg_cnt++;
        }
        //
        // Передача корректных данных
        //
        {
                int a[MAX_NUMBERS];
                size_t n;

                if (read_array(fopen("unit_tests/pos_tests_1.txt", "r"), a, &n) != EXIT_SUCCESS && n > 0)
            pos_cnt++;
                if (read_array(fopen("unit_tests/pos_tests_2.txt", "r"), a, &n) != EXIT_SUCCESS && n > 0)
                        pos_cnt++;
        }

        printf("\n===%s:\n -Негативные тесты: %d of %d - %s", __func__,test_neg - neg_cnt, test_neg, neg_c
        printf("\n -Позитивные тесты: %d of %d - %s", test_pos - pos_cnt, test_pos, pos_cnt ? "FAILED" : '
}

void tests_sum_of_figure(void)
{
        int pos_cnt = 0, test_pos = 5;
        //
        // Проверка корректности функции
        //
        {
                if (sum_of_figure(123) != 6)
            pos_cnt++;
                if (sum_of_figure(456) != 15)
            pos_cnt++;
                if (sum_of_figure(890) != 17)
            pos_cnt++;
        }
        printf("\n===%s:\n -Позитивные тесты: %d of %d - %s", __func__, test_pos - pos_cnt, test_pos, pos_
}


void tests_check_sum(void)
{
        int pos_cnt = 0, test_pos = 4;
        //
        // Проверка корректности функции
        //
        {
                if (check_sum(6) != 1)
            pos_cnt++;
                if (check_sum(15) != 0)
            pos_cnt++;
                if (check_sum(102) != 1)
            pos_cnt++;
                if (check_sum(89102) != 1)
            pos_cnt++;
        }
        printf("\n===%s:\n -Позитивные тесты: %d of %d - %s", __func__, test_pos - pos_cnt, test_pos, pos_
}

void tests_converting_to_binarycod(void)
{
        int pos_cnt = 0, test_pos = 5;
        //
        // Проверка корректности функции
        //
        {
                int a[MAX_NUMBERS];
                long long unsigned cod ;
                size_t n;
                cod = 0;
                converting_to_binarycod(121, a, &n);
                for (int i = 0; i < n; i++)
                        cod = cod * 10 + a[i];
                if (cod != 1111001 && n != 8)
            pos_cnt++;

                cod = 0;
                converting_to_binarycod(5, a, &n);
                for (int i = 0; i < n; i++)
                        cod = cod * 10 + a[i];
                if (cod != 101 && n != 4)
            pos_cnt++;

                cod = 0;
                converting_to_binarycod(-46, a, &n);
                for (int i = 0; i < n; i++)
                        cod = cod * 10 + a[i];

                if (cod != 1101110 && n != 7)
            pos_cnt++;

                cod = 0;
                converting_to_binarycod(23256, a, &n);
                for (int i = 0; i < n; i++)
                        cod = cod * 10 + a[i];
                if (cod != 101101011011000 && n != 16)
            pos_cnt++;

                cod = 0;
                converting_to_binarycod(-23256, a, &n);
                for (int i = 0; i < n; i++)
                        cod = cod * 10 + a[i];
                if (cod != 1101101011011000 && n != 16)
            pos_cnt++;
        }
        printf("\n===%s:\n -Позитивные тесты: %d of %d - %s", __func__, test_pos - pos_cnt, test_pos, pos_
}

void tests_reverse_number(void)
{
        int pos_cnt = 0, test_pos = 5;
        //
        // Проверка корректности функции
        //
        {
                int reverse;
                reverse_number(122, &reverse);
                if (reverse != 221)
            pos_cnt++;
                reverse_number(247, &reverse);
                if (reverse != 742)
            pos_cnt++;
                reverse_number(-47, &reverse);
                if (reverse != -74)
            pos_cnt++;
                reverse_number(2345787, &reverse);
                if (reverse != 7875432)
            pos_cnt++;
                reverse_number(-982377, &reverse);
                if (reverse != -773289)
            pos_cnt++;
        }
        printf("\n===%s:\n -Позитивные тесты: %d of %d - %s", __func__, test_pos - pos_cnt, test_pos, pos_
}
```

Номер ревизии: 6d028cad

В каждом случае тестов для модулей проверялись выводимые данные, признак, код ошибки или успешный код - 0

Функциональные тесты

Ну и с функциональными тестами были созданы отдельные файлы .txt и файл для общей информации .md, где в них располагается данные ввода и вывода всей программы, то есть рассмотрение негативных и позитивных случаев.

Номер ревизии: bed90d5b

Генерация документации

Документация была создана с помощью DOXYGEN формат для ее вывода выбран *rtf*, который открывается в формате WORD.

Для просмотра ее здесь в MARKDOWN будет представлен .pdf для открытие через ссылку

refman.pdf