Национальный исследовательский университет «Высшая школа экономики»
Факультет компьютерных наук
Программная инженерия

Архитектура вычислительных систем

Отчёт по индивидуальному домашнему заданию№2
Вариант 10

Работу выполнила:
М. В. Царахова
Группа: БПИ-213
Преподаватель:
С. А. Виденин

Москва
2022

Содержание

Постановка задачи

Разработать программу, которая меняет на обратный порядок следования символов каждого слова в ASCII-строке символов. Порядок слов остается неизменным. Слова состоят только из букв. Разделителями слов являются все прочие символы.

1. 7 баллов

Условие: Разработать программу вычисления числа π с точностью не хуже $0,1\%$ посредством дзета-функции Римана. По ссылке Дзета-функция Римана можно прочесть информацию о функции.

Мы знаем, что $\zeta(s) = \frac{1}{1^s} + \frac{1}{2^s} + \frac{1}{3^s} + \ldots$.

Также существуют явные формулы для значений дзета-функции в чётных целых точках. В частности:

$$\zeta(2) = \frac{\pi^2}{6}, \zeta(4) = \frac{\pi^4}{90}, \zeta(6) = \frac{\pi^6}{945}, \zeta(8) = \frac{\pi^8}{9450}, \zeta(10) = \frac{\pi^{10}}{93555}$$

Экспериментально было выяснено, что чем больше аргумент дзета функции, тем более точно можно вычислить число пи, поэтому было принято решение взять аргумент 10. Таким образом $\zeta(10) = \frac{\pi^{10}}{93555}$ и из этого следует что $\pi = \sqrt[10]{93555 * \zeta(10)}$. То есть для того чтобы вычислить число пи мы будем вычислять функцию дзета по её изначальному определению с аргументом 10. Таким образом мы получили вот такую функцию вычисления:

```
double compute() {
    // f(10) = pi ^ 10 / 93555
    // pi ^ 10 = 93555 * f(10)
    // pi = (93555 * f(10)) ^ (0.1)
    double res = 1;
    for (int i = 2; i < 38; ++i) {
        res += (1 / pow(i, 10));
    }
    return pow(res * 93555, 0.1);
}
```

В качестве ввода мы будем вводить число знаков после запятой, то есть точность. Максимальная достигнутая точность это 15 знаков после запятой.

Функции ввода и вывода:

```
void input(FILE *input_, unsigned int *precision) {
    fscanf(input_, "%u", precision);
}

void output(FILE *output_, double out, unsigned int precision) {
    char* format = malloc(20 * sizeof (char));
    format[0] = '\0';
    strcat(format, "%.");
    sprintf(format + 2, "%u", precision);
    strcat(format, "f\n");
    fprintf(output_, format, out);
    free(format);
}
```

В функции вывода мы добавляем в форматную строку количество знаков после запятой.

В функции main реализован приём имён файлов через аргументы, а также можно вводить через консоль если передать "stdin", и выводить в консоль, если передать "stdout". Проверяется количество аргументов командной строки. Также есть проверки на корректность открытия файлов.

```
int main(int argc, char *argv[]) {
    if (argc != 3) {
        printf("Please give input and output file");
        return 0;
    }

    FILE *input_;
    FILE *output_;
```

```
9
10      if (strcmp(argv[1], "stdin") == 0) {
11          input_ = stdin;
12      } else {
13          input_ = fopen(argv[1], "r");
14          if (!input_) {
15              printf("BAD INPUT FILE");
16              return 0;
17          }
18      }
19
20      if (strcmp(argv[2], "stdout") == 0) {
21          output_ = stdout;
22      } else {
23          output_ = fopen(argv[2], "w");
24          if (!output_) {
25              fclose(input_);
26              printf("BAD OUTPUT FILE");
27              return 0;
28          }
29      }
30
31      unsigned int precision;
32      input(input_, &precision);
33      output(output_, compute(), precision);
34
35      fclose(input_);
36      fclose(output_);
37      return 0;
38  }
```

Пройдёмся по всем критериям.

Приведено решение задачи на C.

Ввод данных осуществляется с клавиатуры Вывод данных осуществляется на дисплей. В программе на языке C используются функции с передачей данных через формальные параметры.

Ввод данных в программу может быть как с клавиатуры, так и из файлов. Имена файлов задаются с использованием аргументов командной строки. Командная строка проверяется на корректность числа аргументов. В программе присутствует проверка на корректное открытие файлов. При наличии ошибок выводятся соответствующие сообщения. Внутри функций на C используются локальные переменные.

Была получена ассемблерная программа с помощью команды

```
1    gcc -O0 -S -masm=intel -fno-asynchronous-unwind-tables -fcf-protection=none
     main.c
```

-O0 убирает любую оптимизацию, -S останавливает компилятор после порождения ассемблерного кода, -fno-asynchronous-unwind-tables отвечает за удаление макросов, -fcf-protection=none - отключает механизмы защиты безопасности.

Чтобы минимизировать обращение к стеку, заменяя его обращением к регистрам, ко всем объявлениям переменных было приписано слово register (например register double out). После все строки ассемблерной программы были прокомментированы. Потом ассемблерный файл был разделен на две части - main.s и func.s.

С помощью следующей команды был получен исполняемый файл из программы на си

```
1    gcc main.c -o c_out
```

А с помощью этой команды был получен исполняемый файл из ассемблерной программы

```
1    gcc main.s func.s -o asm_out
```

2. Код на ассемблере

Функция main - main.s
Функции input, output, compute - func.s
Исходный код на ассемблере - old.s

3. Сопоставление программ на ассемблере

Количество строк в old.s - 316.
Количество строк в func.s - 124.
Количество строк в main.s - 118.
Всего количество строк в func.s и main.s - 242
Размер объектного файла old.o - 4064
Размер объектного файла main.o - 2208
Размер объектного файла func.o - 1864
Размер бинарного файла old - 17304
Размер бинарного файла new - 17104
Сумма размеров новых объектных файлов на 10 байт больше чем в старом объектном файле. Размер нового бинарного файла на 200 байт меньше, чем размер старого бинарного файла.

4. Тестирование

1. Неправильное количество аргументов

   ./c_out: "Please give input and output file"

   ./asm_out: "Please give input and output file"

2. В качестве input_file_name передается несуществующий файл

   ./c_out bad.txt stdout: "BAD INPUT FILE"

   ./asm_out bad.txt stdout: "BAD INPUT FILE"

3. Используется stdin и stdout

   ./c_out stdin stdout:

   Ввод: 1 Вывод: 3.1

   ./asm_out stdin stdout:

   Ввод: 1 Вывод: 3.1

4. В качестве input_file_name передается 1.txt = "4"

   ./c_out tests/1.txt tests_out/c1.txt

   ./asm_out tests/1.txt tests_out/a1.txt

   Результат: 3.1416

5. В качестве input_file_name передается 2.txt = "6"

   ./c_out tests/2.txt tests_out/c2.txt

   ./asm_out tests/2.txt tests_out/a2.txt

   Результат: 3.141593

6. В качестве input_file_name передается 3.txt = "3"

   ./c_out tests/3.txt tests_out/c3.txt

   ./asm_out tests/3.txt tests_out/a3.txt

   Результат: 3.142

7. В качестве input_file_name передается 4.txt = "10"

   ./c_out tests/4.txt tests_out/c4.txt

   ./asm_out tests/4.txt tests_out/a4.txt

   Результат: 3.1415926536

8. В качестве input_file_name передается 5.txt = "15"

   ./c_out tests/5.txt tests_out/c5.txt

   ./asm_out tests/5.txt tests_out/a5.txt

   Результат: 3.141592653589794

9. В качестве input_file_name передается 6.txt = "20"

   ./c_out tests/6.txt tests_out/c6.txt

   ./asm_out tests/6.txt tests_out/a6.txt

   Результат: 3.14159265358979356009

10. В качестве input_file_name передается 7.txt = "30"

    ./c_out tests/7.txt tests_out/c7.txt

    ./asm_out tests/7.txt tests_out/a7.txt

    Результат: 3.14159265358979356008717331860 7

    Все эти результаты соответствуют числу пи 3.14 15 92 65 35 89 79 32 38 46, с точностью до 15 знака, что даже более чем то, что требовалось от нас в задании.

Вывод - программы идентичны, тесты полностью соответствуют запрашеваемой логике. Эти результаты верные, показывают что программа корректно работает со всеми типами ввода и вывода.