Теория

Форматированный ввод вывод в строку

Считывание и печать даты в формате: YYYY-MM-DD. Например "2008-10-5".

```
char str1[20];
char str2[20] = "2010-5-2";
sprintf(str1, "%d-%2d-%2d", 2008, 10, 5);
sscanf(str2, "%d-%d-%d", &year, &month, &day);
```

1. **Конвертация переменных в строки и обратно:** Объявить и инициализировать переменную типа float. Конвертировать это число в строку, используя функцию sprintf().
2. **Форматированное считывание из строки:** Создать строку представляющую некоторое время и равную "14:54:25.734". Считать количество часов, минут, секунд и миллисекунд в переменные соответствующего типа.

Работа с файлами

- **Запись в файл:** Создаём и открываем файл "myfile.txt"на запись ("w"означает "write"). Проверяем получилось ли открыть файл, если нет, то пишем сообщение об ошибке и выходим. В дальнейших примерах эта проверка будет опускаться. Если получилось открыть, то записываем в файл строку с помощью fprintf().

```
FILE* fp = fopen("myfile.txt", "w");
if (fp == NULL)
{
    printf("Error!\n");
    exit(1);
}
fprintf(fp,"What 1 programmer do in one month, 2 programmers can do in %d months\n", 2);
fclose(fp);
```

- **Чтение из файла:** Открываем файл на чтение ("r"означает "read"). И считываем 3 целых числа.

```
FILE* fp = fopen("numbers.txt", "r");
int x, y, z;
fscanf(fp,"%d%d%d", &x, &y, &z);
fclose(fp);
```

- **Бинарное чтение/запись:** Записываем содержимое массива arr в файл "arr_binary.data". Считываем числа из файла в новый массив.

```
FILE* fp1 = fopen("arr_binary.data", "w");
int arr[5] = {55, 66, 77, 88, 99};
fwrite(arr, sizeof(int), 5, fp1);
fclose(fp1);
// ...
FILE* fp2 = fopen("arr_binary.data", "r");
int another_arr[5];
fread(arr, sizeof(int), 5, fp2);
fclose(fp2);
```

- **Режимы открытия файла:**

r	открыть существующий файл для чтения
w	создать новый файл и открыть его для записи
a	открыть для записи в конец файла
r+	открыть для чтения/записи, с начала файла
w+	создать новый файл и открыть его для чтения/записи
a+	открыть для чтения/записи в конец файла

Задачи

Файлы

Для работы с файлами вам нужно знать в какой директории находится ваш исполняемый файл. Для этого создайте новую директорию в папку `/home-local/Student/workspace/`, а в этой директории создайте исходный файл формата .c. Его можно открыть с помощью geany или nano. Если вы хотите компилировать с помощью gcc, то используйте опцию -o, чтобы задать имя исполняемого файла:

```
gcc -std=c99 -o myexe ./source_file.c
```

1. **Создать файл:** Напишите программу, которая будет создавать файл, используя функцию fopen().
2. **Запись в файл:** Записать в файл по имени `hello.txt` фразу "Hello files".
3. **Запись массива чисел:** Записать в файл по имени `numbers.txt` все числа от 0 до 100000, которые делятся на 17.
4. **Input/output:** Создать файл `input.txt` следующего формата: сначала идёт число n, а затем n целых чисел, например:

   ```
   7
   9 3 5 10 43 52 5
   ```

   Ограничений на число n нет, поэтому для выделения памяти под массив нужно использовать malloc(). Считать эти числа, найти их среднее и записать все числа большие, чем среднее, в файл по имени `output.txt`.
5. **Little or big endian:** Число 6242983 в шестнадцатеричной системе счисления имеет вид 0x005f42a7. Запишите это число в файл в бинарном виде, используя функцию fwrite(). Проверьте, что записалось в файл. Чтобы посмотреть файл в шестнадцатеричном виде можно использовать утилиту xxd:

   ```
   xxd <имя вашего файла>
   ```

   Определить какой порядок байт используется в вашей системе: прямой(big endian) или обратный(little endian).

Аргументы командной строки

- **Аргументы ls:** Программы могут принимать аргументы. Простейший пример – утилита ls. Если запустить ls без аргументов, то она просто напечатает содержимое текущей директории. Однако этим возможности утилиты не ограничиваются. Запустите утилиту ls с разными параметрами (это лучше делать в директории содержащей хотя бы несколько различных файлов и папок):

  1. `ls -m`
  2. `ls -a`
  3. `ls -l`
  4. `ls -R`
  5. `ls -l -R`
  6. `ls -lR`
  7. `ls -l <путь до другой директории>`
  8. `ls --help`

- **argc:** Следующая простейшая программа печатает количество аргументов командной строки. Скомпилируйте эту программу и протестируйте её, запуская с разным количеством аргументов.

  ```
  int main(int argc, char** argv)
  {
          // argc - количество аргументов командной строки
          // argv - массив массивов символов ( или массив строк ) - сами аргументы
          printf("%d\n", argc);
  }
  ```

- **Bogosort:** В папке `~/workspace/seminar10_file/programms/bogosort` лежит исходный код известной (но не очень практичной) сортировки Bogosort. Алгоритм этой сортировки состоит из двух шагов:

- Проверяем не отсортирован ли массив
- Случайно перемешиваем массив и переходим к 1-му шагу

Скомпилируйте программу с помощью команды `gcc ./bogosort.c -o bogosort`. Обратите внимание, что был использован аргумент командной строки `-o` компилятора gcc, задающий имя результирующего исполняемого файла. Если его не использовать, то исполняемый файл будет иметь имя по умолчанию `a.out`. Запустите исполняемый файл `bogosort`. Массив, который нужно отсортировать задаётся через аргументы командной строки.

```
./bogosort 5 4 3 2 1      // Сортируем массив {5, 4, 3, 2, 1}
```

Попробуйте отсортировать массив из 10-ти элементов.

- **Сравнение скорости сортировок:** В папке `~/workspace/seminar10_file/programms/sorting` содержатся исходные коды нескольких сортировок: сортировки пузырьком (bubblesort.c), быстрой сортировки (самописная – quicksort.c, с использование qsort() – qsort.c) и цифровой сортировки (radixsort.c). Вам предлагается сравнить скорость работы этих сортировок.
  - Чтобы сравнить эти сортировки, нужны числа для сортировки. Скомпилируйте файл `generate_numbers.c` с помощью команды `gcc -std=c99 -o generate ./generate_numbers.c`. Появится исполняемый файл `generate`. Если вызвать эту программу следующим образом: `./generate 100`, то программа сгенерирует 100 случайных чисел и запишет их в файл `numbers.txt`.
  - Скомпилируйте файлы bubblesort.c, qsort.c, radixsort.c, countsort.c в исполняемые файлы bubblesort, qsort, radixsort, countsort соответственно. Каждая из этих программ считывает числа из файла `numbers.txt` сортирует их соответствующим образом и записывает отсортированный массив в файл `output.txt`.
  - Сгенерируйте 1000 случайных чисел и протестируйте скорость работы различных сортировок на этих числах. Для этого можно использовать утилиту `time`:

    ```
    time ./bubblesort
    ```

    Сгенерируйте 10^5 случайных чисел и протестируйте скорость работы различных сортировок на этих числах. То же самое для 10^7 чисел.
  - У программы `generate` есть ещё один параметр командной строки, который задаёт максимальное значение сгенерированных чисел. Сгенерируйте 10^7 чисел от 0 до 9 и протестируйте скорость работы различных сортировок.

Посимвольное считывание из файла

Пример программы, которая находит количество цифр в файле:

```
FILE * f = fopen("input.txt", "r");
int c, number_of_digits = 0;

while ((c = fgetc(f)) != EOF)
{
    if (c >= '0' && c <= '9')
        number_of_digits += 1;
}
fclose(f);
```

```
FILE * f = fopen("input.txt", "r");
int c, number_of_digits = 0;

while ((c = fgetc(f)) != EOF)
{
    if (c >= '0' && c <= '9')
        number_of_digits += 1;
}
fclose(f);
```

- Написать программу, которая находит количество символов, слов и строк в файле. Слово это любая последовательность символов, разделённая пробелом, символом табуляции('\t') или символом перевода строки('\n'). Рекомендуется написать отдельные функции под каждую из этих подзадач.
- Изменить программу из предыдущей задачи так, чтобы она принимала название файла через аргумент командной строки. Протестировать работу программы на книгах из папки `~/workspace/seminar10_file/programms/books/`.