Отчет по домашнему заданию 2

1. Оценить работоспособность программы shell-sort, используемую в качестве домашнего задания.

Скомпилируем и запустим программу с некоторым набором входных параметров:

```
stas@stas-Aspire-A715-42G:~$ gcc shell-sort.c -o sort
stas@stas-Aspire-A715-42G:~$ ./sort 3 2 1
Output: 0 1 2 N
stas@stas-Aspire-A715-42G:~$
```

Повторим с некоторым другим набором:

```
stas@stas-Aspire-A715-42G:~$ ./sort 1
Output: 0 N
stas@stas-Aspire-A715-42G:~$ ./sort 10 20 30
Output: 0 10 20 N
stas@stas-Aspire-A715-42G:~$
```

Заметим, что выводится число 0 вместо некоторых введённых чисел. Можно сделать вывод, что

на данный момент программа работает некорректно.

2. Проверить корректность программы на следующем

наборе данных, вводимых в

командной строке: 2 1 5 -12 3 -9. Все ли нормально?

```
./sort 2 1 5 -12 3 -9

Output: -12 -9 1 2 3 5 N
```

Массив отсортирован, никакие элементы не были заменены на другие.

Вывод: на данном наборе чисел все нормально.

3. Рассмотреть другие тестовые наборы, чтобы определить,

возможно ли некорректное поведение.

Вдруг это прикол, и все выполняется корректно!?

Как уже было показано выше, констатировали, что это не прикол

4. Провести отладку и коррекцию программы, если некорректное поведение обнаружено.

Некорректное поведение: обнаружено. Приступаем к коррекции

Выполним следующие команды:

```
gcc -g shell-sort.c -o sort
gdb sort
break main                          ставим брейкпоинт на функции main
run 5 4 3 2 1
```

- **Убеждаемся в нормальном выделении памяти для массива:**

```
Breakpoint 1, main (argc=6, argv=0x7fffffffe048) at shell-sort.c:32
32        a = (int *)malloc((argc - 1) * sizeof(int));
(gdb) print argc
$1 = 6
(gdb)
```

- **Как видим, запись чисел также корректна**

```
(gdb) next
33        for (i = 0; i < argc - 1; i++) {
(gdb) next
34          a[i] = atoi(argv[i + 1]);
(gdb) next
33        for (i = 0; i < argc - 1; i++) {
(gdb) next
34          a[i] = atoi(argv[i + 1]);
(gdb) next
33        for (i = 0; i < argc - 1; i++) {
(gdb) next
34          a[i] = atoi(argv[i + 1]);
(gdb) next
33        for (i = 0; i < argc - 1; i++) {
(gdb) next
34          a[i] = atoi(argv[i + 1]);
(gdb) next
33        for (i = 0; i < argc - 1; i++) {
(gdb) next
34          a[i] = atoi(argv[i + 1]);
(gdb) next
33        for (i = 0; i < argc - 1; i++) {
(gdb) next
36        shell_sort(a, argc);
(gdb) print a[0]
$2 = 5
(gdb) print a[1]
$3 = 4
(gdb) print a[2]
$4 = 3
(gdb) print a[3]
$5 = 2
(gdb) print a[4]
$6 = 1
(gdb)
```

Заметим, что при этом второй аргумент в данном случае равен 6

(т.к. argc также учитывает имя функции в качестве первого значения argv[])

После нескольких итераций цикла вследствие этого получаем обращение к

несуществующему элементу массива:

```
(gdb) next
17        int v = a[i];
1: a[0] = 1
2: a[1] = 4
3: a[2] = 3
4: a[3] = 2
5: a[4] = 5
6: h = 4
7: i = 5
8: j = 0
```

(переменные были добавлены с помощью команды disp)

Исправим вызов функции, передавая туда значение размера argc - 1

Посмотрим на результат:

```
/sort 12 11 10 9 8 7 6 5 4 3 2 1 0

Output: 0 1 2 3 4 5 6 7 8 9 10 11 12 N
```

Методом пристального взгляда и повторным проходом по каждой строке в

gdb установлено, что более код ошибок не имеет

Откорректированная программа:

```
/* shell-sort.c - Сортировка Шелла */

#include <stdio.h>
#include <stdlib.h>

static void shell_sort(int a[], int size) {
  int i, j;
  int h = 1;

  do {
    h = h * 3 + 1;
  } while (h <= size);

  do {
    h /= 3;
    for (i = h; i < size; i++) {
      int v = a[i];
      for (j = i; j >= h && a[j - h] > v; j -= h) {
        a[j] = a[j - h];
      }
      if (i != j) {
        a[j] = v;
      }
    }
  } while (h != 1);
}

int main(int argc, char *argv[]) {
  int *a;
  int i;

  a = (int *)malloc((argc - 1) * sizeof(int));
  for (i = 0; i < argc - 1; i++) {
    a[i] = atoi(argv[i + 1]);
  }
  shell_sort(a, argc - 1);
  printf("Output: ");
```

```
  for (i = 0; i < argc - 1; i++) {
    printf("%d ", a[i]);
  }
  printf("N\n");
  free(a);
  return 0;
}
```

Почему все работало на предложенном в задании массиве

Проверив для разных размеров массива содержимое ячейки a[size] пришел к выводу, что иногда там находится 0, а иногда - очень большое число, которое не "проскакивает" в массив во время сортировки. Таким образом, наблюдается неопределенное поведение