1 Debug

1.1 Баг первый

```
bool TreeInsert(TNodePtr* root, const int x) {
    ...
    while (way != NULL) { // error
        ...
    }
    ...
}
```

```
bool TreeInsert(TNodePtr* root, const int x) {
    ...
    while (*war != NULL) {
        ...
    }
    ...
}
```

Листинг 1: tree.c

1.1.1

Выполним следующие команды:

```
$ make
$ ulimit -c unlimited
$ ./a.out < input-1.txt
$ gdb a.out core
...
Core was generated by `./a.out'.
Program terminated with signal SIGSEGV, Segmentation fault.
#0  0x000000000040083b in TreeInsert ()
```

Из всего этого можно понять, что ошибка где-то в *TreeInsert*, но где именно не понятно :c

1.1.2

Добавим флаг $-g$ в $CFLAGS$ в $Makefile$'е и выполним следующие команды:

```
$ make
$ ./a.out < input-1.txt
$ gdb a.out core
...
Core was generated by `./a.out'.
Program terminated with signal SIGSEGV, Segmentation fault.
#0  0x000000000040083b in TreeInsert (root=root@entry=0x7ffd0d2f1e80, x=1)
    at tree.c:8
8                   if (x < (*way)->Key) {
```

Уже лучше. Можно вывести стектрейс, значения локальных переменных и аргументов функции:

```
(gdb) bt
#0  0x000000000040083b in TreeInsert (root=root@entry=0x7ffd0d2f1e80, x=1)
    at tree.c:8
#1  0x000000000040062b in main () at main.c:31
(gdb) info locals
way = 0x7ffd0d2f1e80
(gdb) info args
root = 0x7ffd0d2f1e80
x = 1
```

1.1.3

Хм, странно, но значение *way* вывелось нормально, но всё же проделаем всё по плану. Заменим в *Makefile*'е в $CFLAGS$ $-O2$ на $-O0$ и выполним следующие команды:

```
...
(gdb) p way
$1 = (TNodePtr *) 0x7ffcac37c9a0
(gdb) p *way
$2 = (TNodePtr) 0x0
```

1.1.4

Из всего вышесказанного можно понять, что ошибка где-то в $TreeInsert$. Ну и то, что, возможно, почему-то при разыменовании указателя на указатель в if получается нулевой указатель.

Условием выхода из *while*'а была проверка указателя на $TNodePtr$ на $NULL$, а нужна проверка $TNodePtr$ на $NULL$.

1.2 Баг второй

```
bool TreeErase(TNodePtr* root, const int x) {
    ...
    ld->Left = ld->Right->Left;
    ...
}
```

```
bool TreeErase(TNodePtr* root, const int x) {
    ...
    ld->Left = ld->Left->Right;
    ...
}
```

Листинг 2: tree.c

Выполним следующие команды:

```
$ ./a.out < input-2.txt
$ gdb a.out core
...
Core was generated by `./a.out'.
Program terminated with signal SIGSEGV, Segmentation fault.
#0  0x00000000004009bb in TreeErase (root=0x7ffff314d860, x=2) at tree.c:43
43                      ld->Left = ld->Right->Left;
(gdb) ld->Right
$1 = (TNodePtr) 0x0
```

Здесь ошибка в том, что мы "подвешиваем" не то поддерево вместо удаляемой вершины. И так получилось, что в данном случае у *ld* вообще нет правого поддерева.

1.3 Баг третий

```
TNodePtr CreateNode(int key) {
    ...
    TNodePtr node = malloc(sizeof(TNodePtr));
    ...
}
```

```
TNodePtr CreateNode(int key) {
    ...
    TNodePtr node = malloc(sizeof(struct TNode));
    ...
}
```

Листинг 3: node.c

Выполним следующие команды:

```
$ valgrind ./a.out < input-1.txt
...
==60831== Command: ./a.out
==60831==
==60831== Invalid write of size 8
==60831==    at 0x4007DC: CreateNode (node.c:11)
==60831==    by 0x400888: TreeInsert (tree.c:16)
==60831==    by 0x400759: main (main.c:31)
==60831==  Address 0x5205090 is 8 bytes after a block of size 8 alloc'd
==60831==    at 0x4C2DB8F: malloc (in /usr/lib/valgrind/vgpreload_memcheck-amd64-
==60831==    by 0x4007BC: CreateNode (node.c:6)
==60831==    by 0x400888: TreeInsert (tree.c:16)
==60831==    by 0x400759: main (main.c:31)
...
```

По тому, что вывел *valgrind*, можно понять, что теоретически ошибка где-то в *CreateNode* и это как-то связано с записью в память.

И, действительно, в функции *CreateNode* мы выделяем 8 байт, а не 24 байта на структуру *TNode*.

1.4 Баг четвёртый

```
bool TreeErase(TNodePtr* root, const int x) {
    ...
    TNodePtr nodeToDelete = ld;
    DestroyNode(nodeToDelete);
    cur->Right = ld->Right;
    ...
}
```

```
bool TreeErase(TNodePtr* root, const int x) {
    ...
    TNodePtr nodeToDelete = ld;
    cur->Right = ld->Right;
    DestroyNode(nodeToDelete);
    ...
}
```

Листинг 4: tree.c

Выполним следующую команду:

```
$ valgrind ./a.out < input-3.txt
...
==61210== Command: ./a.out
==61210==
==61210== Invalid read of size 8
==61210==    at 0x4009F9: TreeErase (tree.c:49)
==61210==    by 0x400774: main (main.c:35)
==61210==  Address 0x52051b0 is 16 bytes inside a block of size 24 free'd
==61210==    at 0x4C2EDEB: free (in /usr/lib/valgrind/vgpreload_memcheck-amd64-lir
==61210==    by 0x400811: DestroyNode (node.c:16)
==61210==    by 0x4009F4: TreeErase (tree.c:48)
==61210==    by 0x400774: main (main.c:35)
==61210==  Block was alloc'd at
==61210==    at 0x4C2DB8F: malloc (in /usr/lib/valgrind/vgpreload_memcheck-amd64-
==61210==    by 0x4007BC: CreateNode (node.c:6)
==61210==    by 0x400888: TreeInsert (tree.c:16)
==61210==    by 0x400759: main (main.c:31)
...
```

Теперь какая-то ошибка с чтением из памяти в функции *TreeErase*. И, действительно, мы пытаемся обратиться по указателю, но память, на которую он указывает, ранее освободили.

1.5 Баг пятый

```
bool TreeErase(TNodePtr* root, const int x) {
    ...
    *way = (cur->Left != NULL) ? cur->Left : cur->Right;
    ...
}
```

```
bool TreeErase(TNodePtr* root, const int x) {
    ...
    TNodePtr nodeToDelete = *way;
    *way = (cur->Left != NULL) ? cur->Left : cur->Right;
    DestroyNode(nodeToDelete);
    ...
}
```

Листинг 5: tree.c

Выполним следующую команду:

```
$ valgrind ./a.out < input-4.txt
...
==61505== HEAP SUMMARY:
==61505==     in use at exit: 24 bytes in 1 blocks
==61505==   total heap usage: 5 allocs, 4 frees, 5,192 bytes allocated
==61505==
==61505== LEAK SUMMARY:
==61505==    definitely lost: 24 bytes in 1 blocks
==61505==    indirectly lost: 0 bytes in 0 blocks
==61505==      possibly lost: 0 bytes in 0 blocks
==61505==    still reachable: 0 bytes in 0 blocks
==61505==         suppressed: 0 bytes in 0 blocks
==61505== Rerun with --leak-check=full to see details of leaked memory
...
```

Одно ясно - где-то утечка памяти. Посмотрев на входные данные, функцию *TreeErase* и немного поразмыслив, можно прийти к мысле, что ошибка где-то в "переподвешивании". И, действительно, заменив значения по *TNodePtr** на новый *TNodePtr*, память по старому *TNodePtr* нигде не освобождается.