1.2.2

Les différences d'affichage sont probablement due à un effet de bord (possiblement due aux limites de mémoire de l'implémentation statique).

1.2.3

Après l'execution sur `testfile2.txt`, on constate qu'il y a bien un effet de bord, car le programme crash avec l'erreur suivante : `Aborted (core dumped)`.

1.2.4

Pour corriger le problème, on peut insérer le code suivant à la ligne `37` :

```
/* 37. */         if(overflow(theStack)) return 1;
```

De manière plus "propre", on peut créer une variable qui est actualisée et vérifiée dans la condition à chaque tour de boucle puis vérifier à la fin si la boucle a été interrompue à cause du dépassement mémoire à venir :

```
/* 34. */     bool overflowed;
/* 35. */     for (int i=0; i<n && (overflowed = overflow(theStack)); ++i) {
/* 36. */         int v;
/* 37. */         fscanf(input, "%d", &v);
/* 38. */         theStack = push(theStack, v);
/* 39. */     }
/* 40. */     if(overflowed) return 1;
```

2.1.2

Le debugger m'indique que l'erreur de segmentation (`SIGSEGV`) arrive lors d'un `push`.

2.1.3

Le problème peut être corrigé en changeant la ligne `31` par :

```
/* 31. */     Element new = malloc(sizeof(struct s_element));
```

Aussi Valgrind indiquait aussi qu'il restait de la mémoire alloué après l'execution. Ceci est dû à l'implémentation dynamique qui nos force à allouer un espace mémoire pour notre pile. Pour corriger ce problème, on rajoute donc dans le main :

```
/* 01. */ #include <stdlib.h>

...
```

```
/* 50. */     free(theStack);
/* 51. */
/* 52. */     return 0;

...
```

A noter que, par prudence, puisque l'on ouvre un fichier, il faut le fermer après utilisation. On rajoute donc dans le main :

```
/* 41. */     fclose(input);
```