**VALGRIND**

Έκανα τις εξής διορθώσεις στον κώδικα:

1. **Διόρθωση stack\_init()**: Προστέθηκε malloc για την κατανομή μνήμης στη Stack, αποφεύγοντας τη μη αρχικοποιημένη χρήση της.
2. **Διόρθωση stack\_pop()**:
   * Προστέθηκε σωστή απελευθέρωση μνήμης για τον τελευταίο κόμβο.
   * Διορθώθηκε το σενάριο όταν η στοίβα έχει μόνο έναν κόμβο.
3. **Διόρθωση stack\_peek()**: Διορθώθηκε το invalid read, αφαιρώντας την πρόσβαση σε cur->next->data και επιστρέφοντας το cur->data.
4. **Διόρθωση stack\_push()**: Διορθώθηκε η λανθασμένη συνθήκη while (cur != NULL), που προκαλούσε invalid read/write.
5. **Διόρθωση stack\_merge()**: Τώρα απελευθερώνει σωστά τη μνήμη των αρχικών στοιβών.
6. **Διόρθωση stack\_destroy()**: Απελευθερώνει σωστά όλους τους κόμβους και μετά τη στοίβα.

Δοκίμασε να τρέξεις ξανά το valgrind ./test και valgrind ./mem\_errors για να επιβεβαιώσεις ότι τα memory leaks και τα errors έχουν εξαφανιστεί!

**Διαδικασία Debugging με Valgrind για Ανίχνευση και Διόρθωση Σφαλμάτων Μνήμης**

Η άσκηση ζητά να διορθώσεις τα memory errors και leaks στο αρχείο **faulty\_stack.c** χρησιμοποιώντας το **Valgrind**. Ακολουθεί η αναλυτική διαδικασία για τον εντοπισμό και τη διόρθωση των σφαλμάτων:

**1️⃣ Compilation (Μεταγλώττιση) του Κώδικα**

Πριν τρέξεις το **Valgrind**, πρέπει να μεταγλωττίσεις το πρόγραμμα για να σιγουρευτείς ότι δεν υπάρχουν syntax errors:

make

Αν η μεταγλώττιση είναι επιτυχής, μπορείς να προχωρήσεις με τη δοκιμή.

**2️⃣ Ανίχνευση Σφαλμάτων με Valgrind**

Χρησιμοποιούμε το **Valgrind** για να ελέγξουμε το πρόγραμμα για memory leaks και σφάλματα προσβασιμότητας στη μνήμη.

**Έλεγχος του mem\_errors**

Τρέχουμε το πρόγραμμα με **Valgrind**:

valgrind --leak-check=full --show-leak-kinds=all ./mem\_errors

📌 **Τι κάνει αυτό;**

* --leak-check=full: Εμφανίζει λεπτομερείς πληροφορίες για τα memory leaks.
* --show-leak-kinds=all: Αναφέρει διαφορετικούς τύπους memory leaks (definitely lost, indirectly lost, etc.).

**Αναλύοντας το Output του Valgrind**:

* **Invalid Read/Write**: Το πρόγραμμα προσπαθεί να διαβάσει ή να γράψει σε απαγορευμένη μνήμη.
* **Use After Free**: Προσπάθεια πρόσβασης σε μνήμη που έχει ήδη αποδεσμευτεί.
* **Memory Leaks**: Μνήμη που δεσμεύτηκε αλλά δεν αποδεσμεύτηκε πριν από το τέλος του προγράμματος.

**Έλεγχος των Unit Tests (test\_faulty\_stack.c)**

Εκτελούμε τα **unit tests** με το Valgrind για να ελέγξουμε αν η στοίβα λειτουργεί σωστά:

valgrind --leak-check=full ./test

Αν δεις **Memory Leaks** ή **Invalid Reads/Writes**, τότε πρέπει να εστιάσεις στις συγκεκριμένες συναρτήσεις του faulty\_stack.c.

**3️⃣ Διόρθωση των Σφαλμάτων**

**📌 Διορθώσεις που ζητά η άσκηση:**

**1. stack\_init() – Initialization Error**

**Σφάλμα:** Η συνάρτηση δεν κάνει malloc για τη στοίβα, με αποτέλεσμα χρήση μη αρχικοποιημένης μνήμης.

✅ **Διόρθωση:**

Stack \*stack\_init(){

Stack \*s = malloc(sizeof(Stack)); // Δέσμευση μνήμης

if (!s) return NULL; // Έλεγχος αποτυχίας malloc

s->list = NULL;

s->count = 0;

return s;

}

**2. stack\_pop() – Memory Leak**

**Σφάλμα:** Δεν αποδεσμεύει σωστά τον τελευταίο κόμβο.

✅ **Διόρθωση:**

int stack\_pop(Stack \*s){

if(stack\_empty(s))

return INT\_MIN;

Node \*cur = s->list, \*prev = NULL;

while (cur->next != NULL) {

prev = cur;

cur = cur->next;

}

int result = cur->data;

free(cur); // Τώρα γίνεται σωστά η αποδέσμευση

if (prev) prev->next = NULL;

else s->list = NULL; // Αν υπήρχε μόνο ένας κόμβος, ενημερώνουμε τη στοίβα

s->count--;

return result;

}

**3. stack\_peek() – Invalid Read**

**Σφάλμα:** Προσπαθεί να διαβάσει cur->next->data, ακόμα και αν cur->next == NULL.

✅ **Διόρθωση:**

int stack\_peek(Stack \*s){

if(stack\_empty(s))

return INT\_MIN;

Node \*cur = s->list;

while(cur->next != NULL){

cur = cur->next;

}

return cur->data; // Δεν προσπαθούμε να προσπελάσουμε cur->next->data

}

**4. stack\_push() – Invalid Read/Write**

**Σφάλμα:** Ο βρόχος while(cur != NULL) τρέχει μέχρι να γίνει NULL, αλλά μετά προσπαθούμε να γράψουμε στο cur->next.

✅ **Διόρθωση:**

Stack\* stack\_push(Stack \*s, int data){

Node \*node = create\_stack\_node(data);

if (!s->list) {

s->list = node;

} else {

Node \*cur = s->list;

while(cur->next != NULL){ // Πρέπει να τερματίζει στο τελευταίο στοιχείο

cur = cur->next;

}

cur->next = node; // Σωστή προσθήκη κόμβου

}

s->count++;

return s;

}

**5. stack\_merge() – Memory Leaks**

**Σφάλμα:** Η νέα στοίβα δημιουργείται, αλλά δεν αποδεσμεύουμε τις αρχικές.

✅ **Διόρθωση:**

Stack\* stack\_merge(Stack \*s1, Stack \*s2){

Stack \*newStack = stack\_init();

Node \*cur = s1->list;

while(cur != NULL){

newStack = stack\_push(newStack, cur->data);

cur = cur->next;

}

cur = s2->list;

while(cur != NULL){

newStack = stack\_push(newStack, cur->data);

cur = cur->next;

}

stack\_destroy(s1); // Τώρα αποδεσμεύουμε τις αρχικές στοίβες

stack\_destroy(s2);

return newStack;

}

**6. stack\_destroy() – Memory Leaks**

**Σφάλμα:** Δεν αποδεσμεύει όλους τους κόμβους.

✅ **Διόρθωση:**

void stack\_destroy(Stack \*s){

Node \*cur = s->list, \*next;

while(cur != NULL){

next = cur->next;

free(cur);

cur = next;

}

free(s);

}

**4️⃣ Επαλήθευση ότι όλα είναι σωστά**

Μετά τις αλλαγές, ξανατρέχουμε:

valgrind --leak-check=full ./test

valgrind --leak-check=full ./mem\_errors

Αν όλα έχουν διορθωθεί, το Valgrind **δεν θα εμφανίσει memory leaks ή errors**. 🎯

**📌 Συμπέρασμα**

Η διαδικασία debugging με το **Valgrind** είναι ουσιαστική για την εύρεση:

* **Memory Leaks** (χαμένη μνήμη που δεν αποδεσμεύεται).
* **Invalid Reads/Writes** (προσπάθεια προσπέλασης μνήμης εκτός ορίων).
* **Use After Free** (χρήση αποδεσμευμένης μνήμης).
* **Double Free Errors** (διπλή αποδέσμευση).

Με τη σωστή χρήση του Valgrind, εντόπισες και **διόρθωσες όλα τα προβλήματα μνήμης στη στοίβα**!

**Διάγνωση του Segmentation Fault με Valgrind**

Το σφάλμα που σου εμφανίζει είναι **"Use of uninitialised value"** και στη συνέχεια ένα **Segmentation Fault (SIGSEGV)**. Αυτό σημαίνει ότι το πρόγραμμα προσπαθεί να διαβάσει ή να γράψει σε μια μνήμη που **δεν έχει αρχικοποιηθεί σωστά**.

Ας αναλύσουμε το μήνυμα του Valgrind:

php

ΑντιγραφήΕπεξεργασία

==23607== Use of uninitialised value of size 8

==23607== at 0x1091FC: uninitialized\_use (uninitialized.c:15)

==23607== by 0x1091DA: main (mem\_errors.c:7)

* Το πρόβλημα βρίσκεται στη συνάρτηση uninitialized\_use() στη γραμμή 15 του uninitialized.c.
* Η συνάρτηση αυτή καλείται από τη main() στη γραμμή 7 του mem\_errors.c.

**📌 Εντοπισμός του Προβλήματος**

1. **Μη αρχικοποιημένη μεταβλητή:** Είναι πιθανό να χρησιμοποιείς μια μεταβλητή χωρίς να της έχεις δώσει τιμή.
2. **Λανθασμένη πρόσβαση σε pointer:** Ίσως προσπαθείς να διαβάσεις δεδομένα από έναν pointer που δεν δείχνει σε έγκυρη μνήμη.
3. **Λάθος στη χρήση malloc():** Μπορεί να έχεις ξεχάσει να δεσμεύσεις δυναμικά μνήμη με malloc().

**🛠 Διόρθωση του Σφάλματος**

Ας ελέγξουμε το **uninitialized.c** (γραμμή 15) και το **mem\_errors.c** (γραμμή 7).

**Αν το πρόβλημα είναι μια μη αρχικοποιημένη μεταβλητή**, η διόρθωση είναι απλή:

c

ΑντιγραφήΕπεξεργασία

int x; // ❌ Αυτό προκαλεί σφάλμα αν διαβάσεις το x πριν του δώσεις τιμή!

printf("%d", x); // 💥 Χρήση μη αρχικοποιημένης μεταβλητής

// ✅ Λύση: Δώσε αρχική τιμή στη μεταβλητή πριν τη χρησιμοποιήσεις

int x = 0;

printf("%d", x);

**📌 Αν το πρόβλημα αφορά pointer:**

Αν το uninitialized.c:15 περιέχει κάτι τέτοιο:

c

ΑντιγραφήΕπεξεργασία

int \*ptr;

printf("%d", \*ptr); // 💥 Πρόβλημα: ο ptr δεν δείχνει σε έγκυρη μνήμη

**✅ Λύση:**

c

ΑντιγραφήΕπεξεργασία

int x = 42;

int \*ptr = &x; // Τώρα δείχνει σε έγκυρη μνήμη

printf("%d", \*ptr);

**📌 Αν το πρόβλημα αφορά δυναμική μνήμη:**

Αν υπάρχει κάποιος pointer που δεν έχει δεσμευτεί σωστά:

c

ΑντιγραφήΕπεξεργασία

int \*ptr;

ptr = malloc(sizeof(int)); // ✅ Δέσμευση μνήμης πριν τη χρήση

if (!ptr) {

printf("Memory allocation failed!\n");

return 1;

}

\*ptr = 42; // ✅ Ασφαλής χρήση

free(ptr); // ✅ Αποδέσμευση μνήμης μετά τη χρήση

**📌 Τελικός έλεγχος με Valgrind**

Αφού κάνεις τις αλλαγές, ξανατρέξε το Valgrind:

sh

ΑντιγραφήΕπεξεργασία

valgrind --track-origins=yes --leak-check=full ./mem\_errors

* Το --track-origins=yes θα σου δείξει **ακριβώς πού δημιουργήθηκε η μη αρχικοποιημένη μεταβλητή**.

**📌 Τελικό Συμπέρασμα**

Το πρόβλημα σου πιθανότατα είναι: ✅ Χρήση μη αρχικοποιημένης μεταβλητής.  
✅ Pointer που δεν δείχνει σε έγκυρη μνήμη.  
✅ Χρήση δυναμικής μνήμης χωρίς malloc().

Δες τη γραμμή 15 στο uninitialized.c και τη γραμμή 7 στο mem\_errors.c, και δοκίμασε τις παραπάνω διορθώσεις.