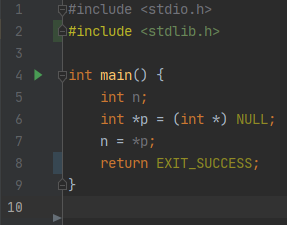
Code wurde auf dem HTWG Container ausgeführt

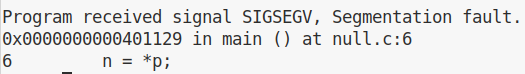
**14.1**

Das Programm wird mit clang -g -Wall -Wpedantic -Wextra -std=gnu17 erfolgreich kompiliert. Wenn man es allerdings ausführt wird ein Segmentation Fault Error geworfen. Was nicht verwunderlich ist, da man ja einen „leeren“ pointer dereferenziert. Die Dereferenziert Variable kann nun mal mit einer „leeren“ Adresse nicht viel anfangen.



**14.2**

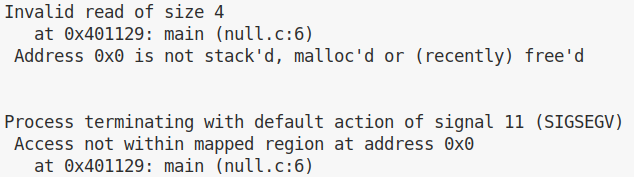
Wie zu erwarten zeigt der Debugger, den gleichen Fehler nur mit etwas mehr Information. Er gibt z.B. die Stelle im Code an, wodurch der Fehler zum Vorschein kommt. Man bemerke allerdings, dass das nicht die Stelle ist wo dem pointer NULL zugewiesen wird, sondern erst die Stelle, wo es dereferenziert wird. Auch gibt er eine Adresse, wahrscheinlich die physikalische, im Speicher, wo das Programm wahrscheinlich untergebracht wurde, an.



**14.3**

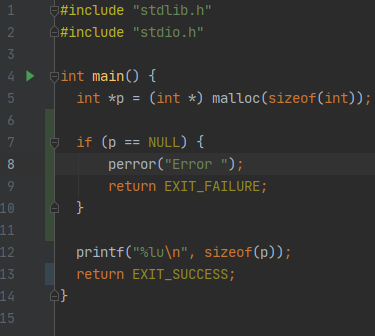
Auch Valgrind gibt logischer weise auch einen SIGSSEGV oder Segmentation fault zurück. Allerdings sagt er, dass es daran liegt, dass es eine invalid read of Size 4 gibt. Dies wird wahrscheinlich aus dem Grund hervorgerufen, da wir einen INT pointer haben, die normalerweise eine 4 Byte Größe haben, aber eine NULL Byte Größe zurückkommt. Auch sagt er das im HEAP nichts allokiert wurde, was normalerweise der Fall sein sollte, wenn der Pointer richtig erstellt würde.

Die Option –leak-check=yes schaltet den detailierten Memory leak detector ein. Dieser verlangsamt das Programm und führt dazu das mehr Speicher verbraucht wird, aber er gibt auch mehr Fehler und Meldung bezügl. Speicherfehler und Speicherleaks aus.



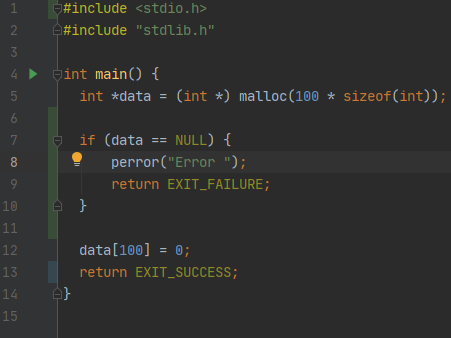
**14.4**

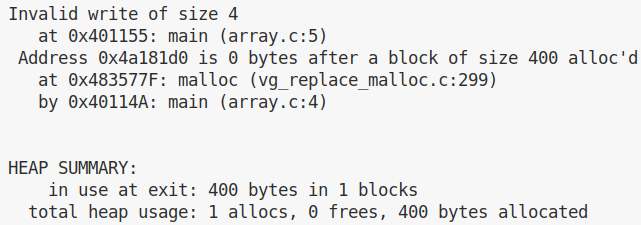
Wenn das Programm ausgeführt wird, wird kein Fehler geworfen. Auch mit GDB kann kein Fehler gefunden werden, da der Code an sich richtig ist. Wenn man dann allerdings mit Valgrind mit der leak-check=yes Flag drüber geht, kann man natürlich feststellen das 4 Bytes verloren gehen. Diese 4 Bytes kommen von den nicht befreiten INT pointer, den wir hier wieder benutzen. Warum die anderen beiden „verfahren“ keinen Fehler werfen kommt daher, dass sobald das Programm beendet ist, der Speicher eh wieder freigestellt wird. Es ist in diesem Fall also nur eine Code Unschönheit.



**14.5**

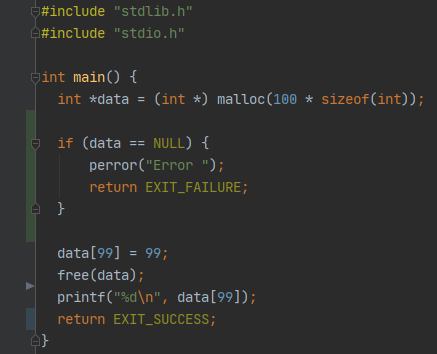
Wie vorher lässt sich das Programm normal per Command Line sowie unter GDB ohne Fehler ausführen. Wenn man nun wieder Valgrind mit der gleicher Flag ausführt bekommt man 2 Fehler. Zum einer wieder der gleiche wie oben. Also das 400 Bytes verloren gehen, da kein Free gemacht wird. Dieser war zu erwarten. Zum anderen wird ein invalid write of 4 Bytes geworfen. Dies dürfte dadurch versucht sein da wird data [100] = 0 machen. Denn unser Array ist zwar 100 Elemente groß doch der höchste Index dürfte 99 sein, da Arrays bei 0 starten.

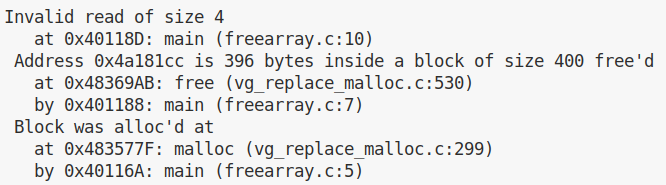




**14.6**

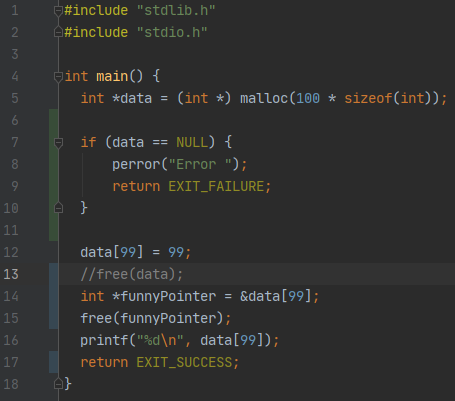
Wie vorher lässt sich das Programm normal per Command Line sowie unter GDB ohne Fehler ausführen. Was eigentlich unterwartet war denn die Werte des Arrays sollten eigentlich nicht mehr vorhanden sein da sie vorher gefreet werden. Dies spiegelt sich auch in dem Valgrind Ergebnis wider. Diesmal haben wir durch den Free immerhin keine Bytes lost allerdings haben wir einen invalid read. Den wie vorher schon gesagt wird vor dem Print der Array gefreet, allerdings wird trotzdem der richtige Wert geprintet.





**14.7**

In unserem Fall haben wir wie im Buch beschrieben einen Pointer gefreed der auf eine bestimmte Stelle von unserem Array zeigt. Kompiliert wird das Ganze noch, allerdings sobald man das Programm ausführt wird ein invalid pointer bei free () geworfen. Laut Manpage verhält sich free () ungewiss, wenn man fehlerhafte Werte übergibt, daher ist es gut, dass das Programm durch sowas aborted wird.



**14.8**

Die Aufgabe wurde so verstanden, dass ein Programm geschrieben werden soll, dass einen Paramter übergeben bekommt um so viel ein Array reallociert wird.

