Programmieren C: Zeichenweises File-I/O: Klammernprüfung

Schreib ein Programm, das einen File liest und prüft, ob die Klammern **( )**, **[ ]** und **{ }** schön *paarweise zusammenpassen* und *richtig geschachtelt* sind.
Es sollen 3 Fehlerfälle erkannt und gemeldet werden:

- Eine schließende Klammer *passt nicht* zur letzten öffnenden Klammer die noch nicht geschlossen wurde.
- Es werden *mehr Klammern geschlossen* als geöffnet wurden.
- Der File ist zu Ende *bevor* alle geöffneten Klammern geschlossen wurden.

Lösungsidee:

- Bei jeder öffnenden Klammer merkt man sich die dazugehörige schließende Klammer als nächstes Element in einem *Array*.
- Bei einer schließenden Klammer, die zum letzten Element im Array passt, geht man wieder auf das vorige Element im Array zurück.
- Passt eine schließende Klammer nicht zum letzten Element im Array, sind die Klammern falsch geschachtelt.
- Kommt eine schließende Klammer, obwohl keine gültigen Elemente mehr im Array sind, ist die Klammer zu viel.
- Enthält das Array am File-Ende noch gültige Elemente, fehlen schließende Klammern.

Hinweise:

- Du sollst den File ***zeichenweise*** lesen.
- Prüfe alle deine I/O-Operationen auf ***Fehler*** und gib *ordentliche Fehlermeldungen* aus!
- Du sollst für die Fallunterscheidung der zu bearbeitenden Zeichen einen **`switch`**-***Befehl*** verwenden.
- Es reicht, wenn dein Programm alle Klammern prüft, ohne sich darum zu kümmern, in welchem Zusammenhang die Klammern auftreten.

  Mit anderen Worten: Eine Unterscheidung zwischen "gültigen" Klammern in einem Source-File und solchen, die in Strings oder Kommentaren versteckt sind und daher bei der Schachtelung nicht mitzählen, ist ***nicht gefordert*** (das wäre viel zu kompliziert!).
- Du darfst zur Buchführung über die gerade offenen Klammern ein Array ***fixer*** Größe verwenden. Wenn zu viele geschachtelte Klammern geöffnet werden, darfst du mit einer Fehlermeldung abbrechen.
- Deine Meldungen sollen auch angeben, ***welche*** Art von Klammer zuviel bzw. zuwenig ist, oder welche Art von schließender Klammer du erwartet hättest und welche wirklich kam.
- Du darfst nach dem ersten Klammernfehler in einem File ***geordnet aufhören***. Es ist nicht gefordert, zu erkennen, ob da jetzt eine Klammer eingefügt, geändert oder übersprungen werden muss, damit die Schachtelung der restlichen noch offenen Klammern wieder zusammenpasst.

- **Tipp:** Die Programmstruktur wird vielleicht etwas einfacher zu realisieren, wenn du den Code nach dem **`fopen`** und vor dem **`fclose`** in eine Funktion packst (dann kannst du nämlich im Fehlerfall einfach ein **`return`** machen, sonst wird der Ausstieg aus dem **`switch`** und dem **`while`** umständlich).

Zusatzaufgaben:

- Kannst du in der Fehlermeldung bei falschen Klammern auch die *Zeilen- und Spaltennummer ausgeben*? Welches Zeichen musst du dazu erkennen und gesondert behandeln?
- Kannst du dein Programm so erweitern, dass es mit *keinem, einem oder mehreren Filenamen* aufgerufen werden kann?

  Bei *keinem* soll es den zu prüfenden Text von **`stdin`** lesen, bei *mehreren* soll es die einzelnen Files der Reihe nach und unabhängig voneinander lesen und prüfen (d.h. bei jedem File wieder mit leerer Klammernschachtelung beginnen). Meldungen über falsche Klammern sollen auch den Filenamen enthalten (die Zeilennummer wird dabei pro File gezählt).

  Bei einem Fehler in einem File soll das Programm nicht aufhören, sondern mit dem *nächsten File* weitermachen.

Alternative Lösungsidee, rekursiv:

Eine (etwas längere und minimal ineffizientere, aber logisch überschaubarere) Lösung beruht auf einer *rekursiven Funktion* mit *einem Aufruf pro Klammern-Ebene* (dafür gibt es *kein Array* und damit keine Grenze für die Klammern-Schachtelung mehr):

- Die Funktion hat einen **`char`**-*Parameter*, nämlich beim rekursiven Aufruf auf Grund einer neuen Klammernebene die erwartete *schließende Klammer*. Der Funktionsaufruf soll den *Input bis zu einschließlich dieser Klammer* verarbeiten.

  Beim *äußersten Aufruf* aus dem *Hauptprogramm* wird **`'\0'`** übergeben. Das zeigt an, dass *keine schließende Klammer* kommen darf.
- Die Funktion zeigt Erfolg oder Fehler durch ihren *Returnwert* (**`true`** / **`false`**) an.
- Die Funktion enthält eine **`while`**-Schleife mit **`switch`** zum *zeichenweisen Lesen*:
  - Trifft man auf eine *öffnende Klammer*, erfolgt ein *rekursiver Aufruf* mit der entsprechenden schließenden Klammer. Ist dieser erfolgreich, liest man weiter, endet er mit Fehler, kehrt man sofort mit Fehler zurück.
  - Trifft man auf eine *schließende Klammer*, gibt es drei Fälle:
    - Die schließende Klammer ist *gleich dem Parameter* (der gesuchten Klammer): Dann kehrt die Funktion sofort mit *Erfolg* zurück.
    - Der Parameter ist **`'\0'`**: Dann dürfte gar *keine schließende Klammer* kommen, weil keine Klammer offen ist. Fehlermeldung und Rückkehr mit Misserfolg.
    - Sonst: Die Klammer *passt nicht* zur öffnenden Klammer. Ebenfalls Fehlermeldung und Rückkehr mit Misserfolg.
- Trifft die Schleife auf das *File-Ende*, muss man ebenfalls den Parameter prüfen: Ist er **`'\0'`**, so ist alles in Ordnung: Rückkehr mit *Erfolg*. Sonst *fehlt eine schließende Klammer*: Fehlermeldung und Rückkehr mit Misserfolg.
- Für *File, Filename und aktuelle Position* darfst du *globale Variablen* verwenden.

Alternative Idee statt **`switch`***:*

- Mach ***zwei Strings*** mit den *drei öffnenden Klammern*
  und mit den *dazupassenden schließenden Klammern*.
- *Suche* das aktuelle Zeichen in diesen Strings (nimm dafür eine *String-Funktion*!).
- Für *öffnene Klammern* lässt sich aus dem Such-Ergebnis *mittels Pointer-Rechnung*
  die *dazupassende schließende Klammer* ermitteln (*ohne* **`if`**):
  Beide stehen an derselben Position im jeweiligen String.