Dokumentation der Implementation eines Filters in C

1 Einleitung

In dieser Dokumentation wird die Implementation einer Aufgabe in der Programmiersprache C beschrieben. Inhalt dieser Aufgabe ist es, ein Filterprogramm zu programmieren bei welchem es möglich ist, einen Dateinamen als Parameter in dem Programm einzulesen, diese Datei zu öffnen. Letztlich soll durch die Übergabe von Parametern es ermöglicht werden, die **Wörter**, **Zeichen** und **Zeilen** der vorliegenden Zeichenkette zu zählen und das Ergebnis in die Standard-Ausgabe zu schreiben. Die Funktionalität dieses Filters ist mit der des Werkzeugs `wc` in UNIX-Systemen vergleichbar.

Zur Kompilierung und Testen des Quelltextes wurde der Kompilierer **gcc** in der Version 9.2.0 aus MinGW verwendet. Für die Erstellung der Dokumentation wurde **TeXworks** verwendet.

1.1 Angeforderte Parameter

- -c zähle alle Zeichen
- -w zähle alle Wörter
- -l zähle alle Zeilen
- -i Angabe des zu lesenden Dateipfads

2 Struktur des Programms

2.1 Erkennen der Parameter und Flags

Zunächst gilt es zu erkennen, welche Operationen von dem Programm durchgeführt werden sollen. Hierzu wird eine Struktur verwendet, welcher die Intentionen des Benutzers als Flags zwischenspeichert.

```
struct mode {
    int characters;
    int words;
    int lines;
};
```

Die verwendete Struktur `mode` wird global definiert und besitzt drei Attribute des Typs `int`, welche jeweils angeben, ob der Nutzer **Zeichen**, **Wörter** oder **Zeilen** zählen möchte. Diese Attribute können unabhgängig voneinander auf 1 (true) gesetzt sein und werden daher verwendet um die gesetzten Flags zu erfassen.

```
struct mode flags;
flags.characters = 0;
flags.words = 0;
flags.lines = 0;
```

Die Struktur wird zu Beginn der Startfunktion `main` mit der Bezeichnung *flags* deklariert und mit den Standardwerten initialisier. Somit existiert nun eine Instanz der Struktur mode, in welcher die Intentionen des Benutzers im Verlauf des Programmes gespeichert und gelesen werden können. Zu Beginn der `main` Funktion wird ebenso eine Variable des Typs int mit der Bezeichnung option deklariert, in welcher das gerade erkannte Programmargument zwischengespeichert wird.

```
while((option = getopt(argc, argv, "cwli")) != -1 ) {
        switch (option) {
            case 'c':
            flags.characters = 1;
            break;
            case 'w':
            flags.words = 1;
            break;
            case 'l':
            flags.lines = 1;
            break;
            case 'i':
```

```
13              if (access(optarg, F_OK) == 0 )
14                  file_name = optarg;
15          }
16  }
```

In der Bedingung der *while* Schleife wird der Variable *option* der Rückgabewert von `getopt` zugewiesen. Mit Hilfe der *while* Schleife wird bei jedem Durchlauf der Schleife ein Argument erkannt. In dem Kontext dieses Programmes prüfen wir in der *switch-case* Abfrage ob 'c', 'w' oder 'l' als Argumente erkannt wurden. Sollten sie erkannt worden sein, so wird das jeweilige Attribut der Struktur *intention* auf wahr gesetzt.

Die `getopt` Funktion der Bibliothek unistd.h erwartet drei Parameter: die Anzahl der Argumente des Programms, ein Array welches die Argumente beinhaltet und eine Zeichenkette welche angibt, welche Argumente zugelassen werden.

Die *while* Schleife ist erst beendet, wenn der Rückgabewert der `getopt` Funktion -1 beträgt, welches bedeutet, dass keine weiteren Parameter mehr vorliegen.

2.2 Lesen der Eingabe

```
1  char* file_name = "";
2  FILE* file = stdin;
```

Zu Beginn der `main` Funktion wird eine Zeigervariable mit der Bezeichnung file_name deklariert und mit einer leeren Zeichenkette initialisiert. In dieser Variable kann im Verlauf des Programms der Name der Datei zwischengespeichert werden, welche gelesen werden soll. Des Weiteren wird eine Zeigervariable mit dem Typ FILE deklariert, welche standardmäßig auf den Standard-Input zeigt.

```
1  if (file_name == "")
2    if (access(argv[argc-1], F_OK) == 0 )
3                file_name = argv[argc-1];
```

Es gilt den Dateinamen zu erkennen, auch wenn das Argument -i nicht an den Programmaufruf übergeben wurde. Hierzu wird die Funktion access der Bibliothek unistd.h verwendet. Diese Funktion erwartet zwei Parameter: den Dateipfad und den zu überprüfenden Sachverhalt. Im Kontext dieses Programms prüft diese Funktion nun, ob das letzte Argument welches übergeben wurde eine valide Datei ist. Sollte es sich um eine valide Datei handeln, so wird dessen Name in der Variable file_name zwischengespeichert.

```
1  if (flags.characters || flags.words || flags.lines) {
2      if(file_name != "")
3          file = fopen(file_name, "r");
```

Es wird überprüft ob mindestens eine der Flags gesetzt wurde (Zeichen, Wörter oder Zeilen müssen gezählt werden), ansonsten wird die Hilfeseite angezeigt. Wenn ein valider Dateiname angegeben wurde, wird dem Dateizeiger die Datei zugewiesen, welche sich an dem Pfad von file_name befindet. Diese Seite wird anschließend mit Schreibrechten geöffnet.

```
1  fclose(file);
```

Anschließend muss die Datei noch geschlossen werden, sobald das Zählen beendet wurde.

```
1  else {
2    display_usage();
3    free(buffer);
4    return 0;
5  }
```

Ist weder eine Datei übergeben worden, noch eine Eingabe über Standard-Input geschehen, so gilt es die Hilfeseite anzuzeigen.

3 Implementierung der Zählung

```
1  int characters = 0, words = 1, lines = 0;
```

Die Zähler für **Wörter**, **Zeichen** und **Zeilen** werden am Anfang der Main-Funktion als Variablen des Typs int deklariert und initialisiert. Der Zähler für Wörter wird mit dem Wert 1 initialisiert, da Wörter immer mit einem Leerzeichen getrennt werden und somit immer ein Wort mehr als die Anzahl der Trennungszeichen vorliegen muss.

```
int c = getc(file);
```

Es wird eine Variable c mit dem Typ int deklariert in welcher das zu prüfende Zeichen gespeichert wird. Ihr wird bei der Initialisierung das ersten Zeichen der Datei zugewiesen.

```
while(c != EOF) {
    characters++;
     if (c == '\n') lines++;
     else if (c == ' ') words++;
     c = getc(file);
 }
```

Jedes Zeichen der Datei wird iteriert, bis das Ende der Datei (EOF) erreicht wurde. Bei jedem Durchlauf wird die Anzahl der Zeichen erhöht. Der Zeilenzähler wird erhöht, sobald ein Newline-Zeichen gefunden wurde, und der Wortzähler wird erhöht, sobald ein Leerzeichen gefunden wurde. Zuletzt wird der Variable c das nächste Zeichen der Datei zugewiesen.

```
if(flags.characters) printf("%d characters\n", characters);
if(flags.words) printf("%d words\n", words);
if(flags.lines) printf("%d lines\n", lines);
printf("%d lines\n", count_lines(buffer, strlen(buffer)));
```

Zuletzt wird überprüft, ob in der Struktur *flags* die Attribute *characters* (Zeichen), *words*(Wörter) oder *lines*(Zeilen) auf wahr gesetzt sind. Sollten sie auf wahr gesetzt sein, so wird die dazugehörige Funktion aufgerufen.

4 Auswertung der Ergebnisse

Für die Erprobung der einzelnen Funktionen dieses Programms wurde mit der in der Aufgabe vorliegenden Textdatei *7zara10.txt* gearbeitet. Wird das vorliegende Programm unter Windows (mit den vorher angeführten Werkzeugen) kompiliert und ausgeführt, so erfolgt die folgende Ausgabe:

```
$ a.exe -c -w -l -i 7zara10.txt
523940 characters
81058 words
4612 lines
```

Der selbe Programmaufruf funktioniert ebenso wenn der Parameter -i ausgelassen wird:

```
$ a.exe -c -w -l 7zara10.txt
523940 characters
81058 words
4612 lines
```

Des Weiteren kann das Programm die Zeichen, Wörter und Zeilen zählen, wenn aus dem Standard-Input gelesen wird:

```
$ cat 7zara10.txt | a.exe -c -w -l
523940 characters
81058 words
4612 lines
```

Bei *unterschiedlichen* Implementationen des Programms dieser Aufgabe kann es zu unterschiedlichen Ergebnissen bei der *selben* Eingabedatei kommen. Dies liegt darin begründet, dass unterschiedliche Implementationen verschiedene Anforderungen an die Zählung stellen.