Programmiersprachen 1 Übung 3 / Gruppe B / Funktionen und Prozeduren

Aufgabenstellung

Eine einfache Shell ist zu programmieren, die, nachdem der Benutzer sich mit seinem Benutzernamen und Passwort identifiziert hat, eine Kommandozeile zur Verfügung stellt, unter der er einige einfache Befehle aufrufen und sich danach wieder ausloggen kann.

Programmablauf

Nach dem Programmstart erwartet den Benutzer die Aufforderung, seinen Benutzernamen und sein Passwort anzugeben. Das Passwort besteht nur aus Buchstaben (Gross- und Kleinbuchstaben!!). Die Eingabe wird mit RETURN beendet. Für die einzelnen eingegebenen Buchstaben werden - anstatt der normalen Ausgabe - Sterne ('*') auf dem Bildschirm ausgegeben. Gib der Benutzer ein anderes Zeichen ein, wird die Verarbeitung der Eingabe verweigert. Ein <Backspace>, zur Korrektur der eingegebenen Zeichen, wäre eine schöne Ergänzung, wird aber nicht gefordert. Nach Eingabe des Passwortes (Abgeschlossen durch RETURN) wird, wenn seine Eingabe korrekt war (also Benutzername und Passwort), die Kommandozeile zur Verfügung gestellt, ansonsten wird der Benutzer gefragt, ob er die Eingabe wiederholen oder abbrechen möchte.
In der Kommandozeile angekommen, wird diese durch ein Symbol (z.B.: '>' oder '$') am Zeilenanfang gekennzeichnet. Hier können nun die Befehle direkt eingegeben werden: z.B.:

```
> echo Hallo Shell
Hallo Shell
>
```

Wird ein Befehl eingegeben, der in der Shell nicht definiert ist, wird dies mit einer entsprechenden Fehlermeldung quittiert. Werden z.B.: die Parameter eines komplexeren Befehls vergessen, wird der Benutzer auf die korrekte Syntax hingewiesen.
All dies kann sich solange wiederholen, bis der Benutzer mit dem Schlüsselwort 'quit' oder 'logout' die Shell schließt, bzw. ein neues Login vollzieht.

Funktionsweise

Zum Einlesen des **Benutzernamens** bietet sich eine einfache String-Variable an, die mit dem vordefinierten (s. Randbedingungen) Benutzernamen zu vergleichen ist. Für das **Passwort** bietet es sich an, zeichenweise (s. Hinweise) einzulesen. Das Passwort wird ebenfalls im Programm in einem String verwaltet, es müssen also die eingelesenen Zeichen mit diesem String verglichen werden. Hierzu nicht genug: das Passwort soll innerhalb des Programms verschlüsselt verwaltet werden, dazu müßt ihr eine **Verschlüsselungsfunktion** (ROT13 s. Randbedingungen) implementieren, durch die ihr das eingegebene Passwort schickt, damit es mit dem vorgegebenen Passwort verglichen werden kann.
Für die Kommandozeile sind folgende Befehle zu implementieren:

```
echo <EINGABESTRING>   -- Implementiert ein einfaches Echo des Eingabestrings.
sort <EINGABESTRING>   -- Gibt eine, nach Wertigkeit der Zeichen im ASCII-Code,
sortierte Version des
                          Eingabestrings aus (Bubble-Sort, s. Randbedingungen),
                          z.B.:
                          > sort DeAF2
                          2ADFe
                          >
whoami                 -- Gibt den Benutzernamen aus.
code <EINGABESTRING>   -- Gibt den Eingabestring ROT13 codiert aus.
help                   -- Listet alle Befehle auf und erläutert ihre Funktion
quit                   -- Beendet das Programm
logout                 -- Ruft die Login-Funktion auf
passwd <EINGABESTRING>  -- Ändern des Passwortes
```

Randbedingungen

- Es sind **zwei globale Variablen** erlaubt, nämlich der Benutzername und das Passwort.
- Führende **Leerzeichen** und nachfolgende Leerzeichen sind zu überlesen (oder löschen, wie es passt.)
- Zwischen einem Befehl und einem Eingabestring steht nur ein Leerzeichen, stehen dort mehrere, müssen diese ebenfalls überlesen werden.
- Abweichungen von diesen Regeln sind erlaubt (allerdings kaum sinnvoll), müssen jedoch anhand eines

Syntaxdiagramms erklärt werden.

- Am Anfang des Programm müßt ihr eine **Initialisierungsprozedur** aufrufen, die zuvor den Benutzernamen und das Passwort setzt.
- Folgende Programmteile sind (auf jeden Fall) ebenfalls in eigenen Funktionen und Prozeduren zu implementieren:
  - function ROT13_CODE(s : string) : string; {gibt den ROT13 codierten String zurück}
  - function BUBBLE_SORT(s : string) : string; {gibt den sortierien String zurück}
  - procedure ECHO (s : string); { Gibt den mitgegebenen String in einer eigenen Zeile auf dem Bildschirm aus }
  - function LOGIN : boolean; { führt das gesamte LOGIN durch, also Benutzername Passwort etc.}
  - function MY_SHELL : byte; { Programmteil, der mit der den Kommandos der Shell umgeht. Return-Wert ist Null, wenn Kommando korrekt ausgeführt wurde}
  - procedure Entferne_fuehrende_Leerzeichen (var s : string); {Eliminiert in dem uebergebenen String die Leerzeichen am Anfang des Strings}
  - procedure Entferne_nachfolgende_Leerzeichen (var s : string); {Eliminiert in dem übergebenen String die Leerzeichen am Ende des Strings}
- Weitere Modularisierung bietet sich an. Abweichungen von diesen Prozedur- und Funktionsköpfen sind ebenfalls erlaubt, sofern sie sinnvoll erklärbar sind.
- Alle wichtigen Vorgabewerte sind an zentraler Stelle zugänglich zu machen.

Der ROT13 Algorithmus

- Der Algorithmus wird nur auf Zeichenketten angewandt.
- Die Verschlüsselung betrifft jedes Zeichen einzeln, unabhängig von dessen Umgebung in der Zeichenkette.
- Nur Buchstaben sind betroffen. Ziffern und Sonderzeichen bleiben unberührt.
- Ein Buchstabe wird verschlüsselt, indem auf dessen Nummer im Alphabet die Zahl 13 hinzuaddiert wird. Das Ergebnis ist der Buchstabe, der an der berechneten Stelle im Alphabet steht.
- Sollte die Zahl über das Alphabet hinausschießen, so wird am Anfang des Alphabets wieder angefangen zu zählen. Es wird also ein Buchstabe um 13 Buchstaben ROTiert.
- Verschlüsselung und Entschlüsselung sind identisch.

Der BUBBLE_SORT Algorithmus

- Im Pseudocode:

```
repeat
    GETAUSCHT:=false
    I:=1
    while I < N do
        if S(I) > S(I+1) then
            Austausch S(I) mit S(I+1)
            GETAUSCHT:=true
        end-if
        I:=I+1
    end-while
until GETAUSCHT=false
```

- *N* ist die Länge des Strings; *S* der String;
- Da in dieser Routine ein String sortiert werden soll, muß der String indiziert angesprochen werden, d.h. die einzelnen Elemente des Strings (also die Char-Werte) müssen einzeln angefasst, überprüft und getauscht werden. Diesen indizierten Zugriff erreicht man durch die Verwendung von eckigen Klammern [ ] nach dem String-Namen, die einen Byte-Wert einschliessen. Beispiel:

```
s:='Testwert';
writeln(s[3]);
-> AUSGABE: 's'
```

- Nebenbedingung: Ihr müsst diesen Algorithmus auch erklären können. Hierzu bietet es sich an, sich den String einmal aufzuzeichnen und dann die einzelnen Schritte abzuarbeiten, bis der String sortiert ist ("Schreibtischtest").

Hinweise

- Für die Stringverarbeitung sind folgende Standardprozeduren und Funktionen sinnvoll anwendbar:
  - LENGTH, POS, DELETE, COPY, CONCAT
- Zum Einlesen einzelner Zeichen kann die Funktion READKEY verwendet werden, die kein Zeichen auf dem Bildschirm ausgibt, sondern dieses nur von der Tastatur einließt.
- Um den Unterschied zwischen Gross- und Kleinschreibung zu umgehen, ist die Funktion UPCASE, die für jeweils einen übergebenen Buchstaben sein Gegenstück unter den Grossbuchstaben zurückgibt.
- **Tip:** Implementiert zunächst die einzelnen Prozeduren und Funktionen und stellt sicher, daß diese korrekt funktionieren.
- Ein Beispielprogramm (mehr oder minder eine Minimallösung) und die Help-Prozedur stehen zum DOWNLOAD bereit. Beachtet hierbei, daß das Programm nicht der exacten Aufgabenstellung entspricht!