Strukturierte Programmierung Übung 10

Auszuarbeiten bis 28.01.20

1. Wortlängenstatistik (10 Punkte)

Gegeben ist ein Text, der in einem *char*-Feld mit der maximalen Länge 512 gespeichert wird. Der Text besteht nur aus Buchstaben (’a ’.. ’z ’ und ’A ’.. ’Z ’) und den Interpunktionszeichen (’.’, ’,’ und ’?’).

Es soll ein C-Programm entwickelt werden, das die Häufigkeit des Vorkommens von Worten bestimmter Länge zählt. Sowohl der Originaltext als auch das Resultat der Analyse nach aufsteigender Wortlänge soll am Bildschirm präsentiert werden.

2. Bepacken eines Rucksacks (10 Punkte)

Entwickeln Sie einen Algorithmus, der das Problem des optimalen Bepackens eines Rucksackes löst. Es geht darum Gegenstände, die durch eine Bezeichnung, die Anzahl der Einheiten und den Wert pro Einheit bestimmt sind, in einen Rucksack einer vorgegebenen Größe zu packen. Optimal bedeutet, dass der Gesamtwert des bepackten Rucksacks maximal ist.

Beispiel:

Rucksack der Größe: 10

Gegenstände

Bezeichnung	Einheiten	Wert pro Einheit
Filzstift	3	1€
Füllfeder	1	5€
Radiergummi	40	0,5€
Kreide	100	0,1€

Optimale Bepackung: 1 x Füllfeder, 3 x Filzstift, 6 x Radiergummi => 11€

Der Algorithmus liefert als Rückgabewerte die optimale Bepackung und auch deren Gesamtwert zurück. Überlegen Sie sich sinnvolle Datenstrukturen!

1 Wortlängenstatistik

Lösungsidee:
Mit einer while Scheife durch die ganze Textlänge iterieren wobei eine Funktion die Buchstaben des nächsten Wortes zählt und dann die Länge dem index hinzufügt. Das wird nur gemacht wenn die Wortlänge nicht 0 ist sonst wird nur 1 dem index hinzugefügt.

Testfälle:
"Gegeben ist ein Text, der in einem char-Feld mit der maximalen Länge 512 gespeichert wird. Der Text besteht nur aus Buchstaben(’a ’..’z ’ und ’A ’..’Z ’) und den Interpunktionszeichen(’.’, ’, ’ und ’ ? ’).";
"";
"Optimale Bepackung: 1 x Füllfeder, 3 x Filzstift, 6 x Radiergummi => 11€";
",.,,.,???,,.,use,, , , of punctuation,,"

C-Program code: Siehe beigelegtes printWordStats.c/.h file.
Tests: Siehe beigelegtes main.c file.

2 Bepacken eines Rucksacks

Lösungsidee:
Mit einem eigenem erstellten Struct Object, das wie siehe Bild bei den Testfällen, die Einträge einer Tabelle gespeichert und hintereinader in ein Array gegeben. Dannach wird in einer while Schleife in jeder Iteration der Index, der den größten ‚Unit‘ Wert im struct hat, bestimmt und bei dem jeweiligen Struct dann die units hinutergezählt, dabei wird noch gezählt wie oft diese Unit hinutergezählt wird. Am Ende der Iteratiobn wird nochmal der Index, der den größten ‚Unit‘ Wert im struct hat, bestimmt und dann mit dem vorherigen Index vergliechen, wenn unterschiedlich werden die schon benutzten Units ausgegeben. Am Ende wird dann nur noch die summe der zusammengezählten Units ausgegeben.

Testfälle:

```c
// test 1 ---------------
{
    struct InventoryChunk items[] = {
        {"Filzstift", 3, 0, 1},
        {"Fuellfeder", 1, 0, 5},
        {"Radiergummi", 40, 0, 0.5},
        {"Kreide", 100, 0, 0.1}
    };

    success = testBackpackPaking("Testcase_1", 11.00, &errCnt, items, STRUCTLEN_4, 10) && success;
}
// test 2 ---------------
{
    struct InventoryChunk items[] = { NULL };
    success = testBackpackPaking("Testcase_2", 0.00, &errCnt, items, STRUCTLEN_0, 10) && success;
}
// test 3 ---------------
{
    struct InventoryChunk items[] = {
        {"Filzstift", 3, 0, 1},
        {"Fuellfeder", 1, 0, 5},
        {"Radiergummi", 40, 0, 0.5},
        {"Kreide", 100, 0, 0.1}
    };

    success = testBackpackPaking("Testcase_3", 38.00, &errCnt, items, STRUCTLEN_4, 300) && success;
}
// test 4 ---------------
{
    struct InventoryChunk items[] = {
        {"Filzstift", 3, 0, 1},
        {"Fuellfeder", 1, 0, 5},
        {"Radiergummi", 40, 0, 0.5},
        {"Kreide", 100, 0, 0.1},
        {"Handy", 3, 0, 50},
        {"Dosenbier", 100, 0, 0.5},
    };

    success = testBackpackPaking("Testcase_4", 238.00, &errCnt, items, STRUCTLEN_6, 300) && success;
}
```

C-Program code: Siehe beigelegtes backpackPaking.c/.h file.
Tests: Siehe beigelegtes main.c file.