

Programmierübungen

Wintersemester 2007/2008

2. Übungsblatt

6. November 2007

Abgabe bis Freitag, 16. November 23:59 Uhr

Die Abgabe Ihrer Bearbeitung können Sie im eClaus-System durchführen. Erarbeiten Sie Lösungsideen zu den Aufgaben möglichst in Kleingruppen. Es wird jedoch von Ihnen erwartet, dass jeder Teilnehmer eine eigene Lösung abgibt. Sollten kopierte Quelltexte abgegeben werden, so werden grundsätzlich alle Kopien mit 0 Punkten bewertet. In den Vortragsfolien der Programmierübungen oder im Skript zur Einführung in die Informatik abgedruckte Quelltexte können verwendet werden, müssen aber der Programmierrichtlinie entsprechend formatiert und kommentiert werden.

Beachten Sie die Programmierrichtlinie und kommentieren Sie Ihren Quelltext! Dokumentieren Sie unbedingt Ihre Lösungsidee in den Quelltext-Kommentaren.

<http://www.iste.uni-stuttgart.de/ps/Lehre/WS0708/inf-prokurs>

Aufgabe 2.1: Quersumme

(3 Punkte)

Implementieren Sie zwei Varianten des nachfolgenden Programms:

1. Verwenden Sie eine **while**-Schleife,
2. Verwenden Sie eine rekursive Funktion.

Das Programm gibt den Text „Eingabezahl: “ aus. Der Benutzer gibt eine ganze Zahl ein. Das Programm berechnet die Quersumme q der Zahl und gibt diese aus in dem Text „Quersumme: q “.

Die Quersumme einer Zahl ist die Summe ihrer Ziffern bei Darstellung im Dezimalsystem.

Nennen Sie die Programme „Cross_Sum_1“ und „Cross_Sum_2“.

Beispiele:

```
Eingabezahl: 1134
Quersumme: 9
```

```
Eingabezahl: -56736234
Quersumme: 36
```

Aufgabe 2.2: Palindrome

(6 Punkte)

Ein Palindrom ist ein Wort oder Satz, das identisch ist unabhängig davon, ob es von vorne nach hinten oder von hinten nach vorne gelesen wird. Z. B. „neben“, „anna“, ...

Erstellen Sie ein **package** `Palindromes`, das drei Funktionen enthält: `Is_Palindrome_For`, `Is_Palindrome_While`, `Is_Palindrome_Recursive`. Alle diese Funktionen haben folgende Spezifikation (der Name muss jeweils entsprechend ersetzt werden):

```
function Is_Palindrome
  (Word : in String)
  return Boolean;
```

Die Funktionen sollen jeweils überprüfen ob die als aktueller Parameter übergebene Zeichenkette ein Palindrom ist. Wurde ein Palindrom übergeben, so soll die Funktion True zurückgeben, andernfalls False. Erstellen Sie entsprechende Kopf-Kommentare, die diese Funktionalität beschreiben.

Implementieren Sie die Funktionen, verwenden Sie jeweils eine **for**-Schleife, eine **while**-Schleife bzw. einen rekursiven Algorithmus.

Testen Sie Ihre Implementierung.

Aufgabe 2.3: Kleinstes gemeinsames Vielfaches (4 Punkte)

Implementieren Sie ein Programm, das das kleinste gemeinsame Vielfache zweier ganzer Zahlen berechnet. Nennen Sie das Programm „LCM“.

Das Programm liest vom Benutzer zwei positive ganze Zahlen ein. Das Programm berechnet das kleinste gemeinsame Vielfache der Zahlen und gibt dieses aus.

Beispiele:

```
erste Zahl: 15
zweite Zahl: 25
kgV: 75
```

```
erste Zahl: 13
zweite Zahl: 12
kgV: 156
```

Aufgabe 2.4: Permutationen (7 Punkte)

Menschen haben die Fähigkeit in längeren Texten Wörter lesen zu können, an denen nur die Anzahl der Buchstaben sowie der erste und letzte Buchstabe korrekt geschrieben sind. Die inneren Buchstaben können vertauscht sein.

Beispiel: Die inneren Buchstaben können vertauscht sein.

Schreiben Sie ein Programm „Permute“ mit folgendem Ablauf:

- Das Programm gibt den Text „Gib ein Wort ein: “ aus.
- Der Benutzer gibt ein Wort ein.
- Das Programm listet alle möglichen Permutationen des Worts auf, wobei das erste und das letzte Zeichen des Worts jeweils unverändert bleiben.

Beispiele:

```
Gib ein Wort ein: Handy
Handy
Hadny
Hnady
Hnday
Hdany
Hdnay
```

Sollten zwei Permutationen des Worts gleich aussehen, weil gleiche Buchstaben vertauscht werden, so soll Ihr Programm trotzdem beide Resultate ausgeben:

```
Gib ein Wort ein: Leere
Leere
Leree
Leere
Leree
Lreee
Lreee
```

Hinweise:

- Planen Sie zunächst Ihr Vorgehen, bevor Sie beginnen Quelltext zu schreiben.
- Verwenden Sie das Verfahren von den Folien 243 und 244 aus dem Skript zur Vorlesung „Einführung in die Informatik“ um alle möglichen Permutationen einer Index-Folge aufzulisten.
- Beachten Sie zu dem Verfahren: der Schritt „Sortiere $P(i+1)$ bis $P(n)$ “ kann realisiert werden als „kehre die Reihenfolge der Elemente $P(i+1)$ bis $P(n)$ um“.
- Finden Sie eine geeignete Modellierung für Permutationen, z. B. **type** `Permutation is array (Positive range <>) of Positive;` wobei eine bestimmte Permutation P die Bedeutung hat, dass das i . Zeichen des ausgegebenen Worts von der Position $P(i)$ des Eingabeworts stammt.
- Teilen Sie Ihre Implementierung in überschaubare Teile auf, z. B. eine Prozedur `Next_Permutation`, die mit dem Verfahren aus dem Skript die nächste Permutation bestimmt und eine Funktion, die aus Eingabewort und Permutation das permutierte Wort berechnet. Selbstverständlich können Sie auch weitere Unterprogramme verwenden. Fassen Sie die Unterprogramme in einem Paket zusammen (z. B. **package** `Permutations`).

Aufgabe 2.5: Permutierte Texte

(bis zu 2 Zusatzpunkte)

Bitte beachten: Die Bearbeitung dieser Aufgabe ist freiwillig. Es können bis zu 2 Punkte erworben werden, jedoch kann die Anzahl erworbener Punkte für das ganze Aufgabenblatt nicht über 20 Punkte gesteigert werden.

Schreiben Sie ein Programm mit folgender Funktionalität: Das Programm liest eine Textdatei ein, wählt für jedes Wort des Texts eine bestimmte Permutation aus und gibt den resultierenden Text wieder aus. Versuchen Sie die Auswahl der Permutation so zu wählen, dass besonders „schöne“ Schreibweisen herauskommen.

Ein Pseudo-Zufallszahlengenerator wird in dem vordefinierten generischen Paket `Ada.Numerics.Discrete_Random` angeboten. Betrachten Sie die Beispiele im Ada Reference Manual (A.5.2). Links zu PDF und HTML-Versionen finden sich auf der Webseite zu den Programmierübungen.