Technische Universität München
Fakultät für Informatik
Rechnerarchitektur-Praktikum
SS 2015

DICHTEMESSUNG

ANWENDERDOKUMENTATION

Bearbeitet von:

Niklas Rosenstein

12.07.2015

INHALTSVERZEICHNIS

1. Aufgabe des Programms
2. Systemanforderungen
3. Erzeugen und Testen des Programms
4. Manuelle Ausführung des Programms
5. Struktur der Eingabedatei
6. Schnittstelle der Dichteberechnung

1. Aufgabe des Programms

Im Rahmen des Rechnerarchitektur Praktikums an der Technischen Universität München, wurde ein Programm zu Berechnung von Dichtewerten anhand einer Liste von Gewicht-Volumen Wertepaaren erstellt. Das Programm sortiert anschließend die Ergebnisliste und berechnet den Durchschnittswert, sowie die Differenz zwischen dem kleinsten und größten Ergebniswert zum Durchschnittswert. Bei diesem letzten Abschnitt werden die 10 kleinsten und größten Werte vernachlässigt, was dazu führt, dass das Programm mindestens 21 Eingabewerte verlangt.

2. Systemanforderungen

Das Programm ist für die 32-Bit Linux Plattform optimiert und es ist nicht garantiert, dass es auf anderen Plattformen ohne Änderungen lauffähig gemacht werden kann. Auf 64-Bit Linux Systemen ist es notwendig das Paket **`libc6-dev-i386`** vor der Kompilierung zu installieren.

3. Erzeugen und Testen des Programms

Mit dem Kommando **`make build`** wird aus dem Quellcode ein ausführbares Programm an dem Pfad **`build/main`** erstellt. Im nachfolgenden ist die zu erwartende Ausgabe bei einer erfolgreichen Kompilierung zu finden.

```
$ make build
mkdir -p build
gcc -m32 -c -o build/read.o read.c
gcc -m32 -o build/main build/calc.o build/read.o
```

Mit dem Befehl **`make test`** wird dem Anwender ermöglicht, einen Testlauf mit einer der mitgelieferten Beispiel-Eingabedateien durchzuführen. Der Anwender wird dabei vorerst in einen Auswahlbildschirm geleitet, zu welchem er nach dem Testlauf wieder zurückkehrt. Hier muss eine Zahl eingegeben werden, mit welcher entschieden wird, welcher Testfall in das Programm gespeist wird.
Nach Auswahl des Testfalls wird das Programm gestartet und nach kurzer Zeit werden die Ergebnisse auf dem Bildschirm ausgegeben. Für den Testfall “random015.txt” ist eine Fehlermeldung zu erwarten, da mindestens 21 Wertepaare erwartet werden.

In der Ausgabe des Programms sind im ersten Block die Ergebnisse der Dichteberechnung und im zweiten Block der Durschnitts-, Differenz-Maximal- und Differenz-Minimalwert zu finden.

4. Manuelle Ausführung des Programms

Nachdem die Schritte in (3.) befolgt wurden liegt das Programm an dem Pfad “build/main”. Es kann nach belieben umbenannt und verschoben werden. Die folgenden Kommandozeilenargumente werden akzeptiert:

`-v`	Dieses Argument kann kein oder mehrmals gegeben werden. Mit jedem mal wird das “Verbosity Level” um eines erhöht und damit die Menge an zusätzlichen Information bei der Ausgabe.
`-s`	Wenn dieses Argument gegeben ist, werden die Ergebnisse in die Konsole ausgegeben. Normalerweise sollte man diese Option immer übergeben.
`file`	Der Dateiname der Eingabedatei. Dieser Parameter ist optional. Sollte kein Dateiname gegeben sein, wird von stdin gelesen.

Beispiel:

```
$ build/main -vv -s ~/metal_measurements.txt
```

5. Beschreibung des Programmablaufs

Die Eingabedatei für das Programm muss ein striktes ASCII-Format einhalten. Auf jeder Zeile wird ein Wertepaar erwartet, wobei der erste Wert das Gewicht und der zweite das Volumen repräsentiert. Das genau Format einer Zeile lautet

```
%f mm %f mm
```

Leere Zeilen werden nicht unterstützt. Wenn eine ungültige Zeile gefunden wird, wird der Lesevorgang abgebrochen und das Programm fährt fort wie normal.

6. Ansätze zur Weiterentwicklung

Die Prozedur der Dichteberechnung kann in andere Programme mittels der in Assembler implementierten `calc()` Funktion eingebunden werden. Die Signatur lautet:

```
/// ------------------------------------------------------------
/// Computes the density for each pair of weight and volume values
/// in \p data1 and \p data2. The \p results1 array will be sorted,
/// so the outcoming values will not be in original order. (Yes,
/// this makes the function not really useful :-)
///
/// @param[in] nlines - A pointer to the number of entries in
///   \p data1, \p data2 and \p results1. A minimum value of
///   21 (twenty-one) is expected, otherwise the function will
///   lead to undefined behaviour.
/// @param[in] data1 - Pointer to an array of weight-values. The
///   number of values are specified by \p nlines.
/// @param[in] data2 - Pointer to an array of volume-values. The
///   number of values are specified by \p nlines.
/// @param[out] results1 - Pointer to an array of \p nlines values.
///   The calculated density values will be stored in this array.
/// @param[out] results2 - Pointer to an array of 3 (three) values.
///   The average, delta max and delta min value will be stored
///   in this array.
/// ------------------------------------------------------------
void calc(int* nlines, float* data1, float* data2,
          float* results1, float* results2);
```

Auf Plattformen, die Funktionsnamen dekorieren, müssen eventuell compilerspezifische Änderungen vorgenommen werden, damit der Linker mit dem Symbol calc verlinkt, und nicht mit einem dekoriertem Derivat wie etwa **`_calc`**. In GCC beispielsweise lässt sich dies durch das überschreiben des Assemblerfunktionsnamen erreichen:

```
void calc(int* nlines, float* data1, float* data2,
          float* results1, float* results2) asm ("calc");
```

Beispiel:

```
int nlines = …;
float* data1 = malloc(sizeof(float) * nlines);
float* data2 = malloc(sizeof(float) * nlines);
float* results1 = malloc(sizeof(float) * nlines);
float results2[3];
// Fill data1, data2
calc(&nlines, data1, data2, results1, results2);
```