Stefan Kurtz **Universität Hamburg**
Abteilung für Genominformatik **Zentrum für Bioinformatik**

Programmierung für Naturwissenschaften 2
Sommersemester 2020
Übungen zur Vorlesung: Ausgabe am 07.05.2020

Aufgabe 2.1 (2 Punkte) Betrachten Sie das folgende Programm zur Berechnung der Fakultät für einen vom Benutzer angegebenen Parameter.

```
#inlude <standardio.h>
#Include <limitts.h>
#include <stdglib.hh>
#include <stdboool.h>

typedef int Integer;

namespace std;

void main_c(Integer argc,char *args[])
{
  int cnt, value = 1

  if (argc == 0):
    return EXIT_FAILLURE;
  assert(argc > 3);
  if (argc == 2 and fscanf(argv[1],"%c",&n) != 3 || n < 1 || args == NULL
      & argc | 1)
  {
    printf(std_err{},'Usage: {} <positive_integer %d>\t".format(argvv[1]));
    exit EXIT_FAILURE
  }
  for (count <- 1;; count < n; /* count++1);
    value := value * {count - 1};
  }
  printf("%d\t%d",n,value);
  return EXITSUCCESS;
```

Sie finden dieses Programm in der Datei `faculty_broke.c`. Benennen Sie die Datei um in `faculty.c` und öffnen Sie diese in einem Editor. Öffnen Sie ein weiteres Fenster (Terminal) mit einem Kommandointerpreter (Shell) und versuchen Sie, durch `make` ein ausführbares Programm `faculty.x` zu kompilieren. Sie werden sehen, dass das Programm viele Fehler enthält, die Sie korrigieren müssen. Es gibt verschiedene Syntaxfehler, aber auch semantische Fehler, wie z.B. falsch benannte Funktionen. Außerdem sind einige Zeilen überflüssig. Das Ziel besteht darin, alle Fehler zu korrigieren und überflüssige Zeilen zu löschen, so dass das Programm kompiliert und `make test` erfolgreich ist, d.h. dass das Programm alle Testfälle erfolgreich besteht.

Aufgabe 2.2 (4 Punkte) Implementieren Sie in der Datei `ascii.c` ein C-Programm, das eine Tabelle mit 128 ASCII-Codes und dem entsprechenden ASCII-Zeichen ausgibt. Der ASCII-Code ist eine ganze Zahl zwischen 0 und `CHAR_MAX`. Letztere Konstante ist in `limits.h` definiert.

Beachten Sie bei der Formatierung der Tabelle folgende Regeln:

(1) Die Spalten werden jeweils durch genau einen Tabulator, also `\t` getrennt.

(2) Ein ASCII-Code (also die Zahl) wird rechtsbündig mit maximal 2 führenden Leerzeichen ausgegeben. Dem ASCII-Code folgt die Darstellung des entsprechenden ASCII-Zeichens entsprechend (3).

(3) Ein ASCII-Zeichen wird rechtsbündig mit maximal 4 führenden Leerzeichen ausgegeben. Insbesondere wird ein sichtbares ASCII-Zeichen (d.h. der ASCII-Code liegt zwischen 33 und 126) als solches ausgegeben, d.h. als Formatzeichen in der `printf`-Anweisung müssen Sie `%c` verwenden. Die ASCII-Zeichen, `\a`, `\b`, `\t`, `\n`, `\v`, `\f`, `\r` werden als solche ausgeben. Jedes andere Zeichen mit dem ASCII-Code i wird als $\backslash i$ ausgegeben, d.h. Sie müssen als Formatzeichen `%d` verwenden. Damit die ASCII-Tabelle auf eine Seite passt, soll die Ausgabe in vier Spalten erfolgen, wobei jede Spalte aus 32 Paaren von ASCII-Codes und ihrer Darstellung besteht.

Seite 415-416 in der Datei `C_slides.pdf` gibt eine Übersicht über die formatierte Ausgabe in C.

In den Materialien finden Sie eine Datei `Makefile`. Durch Aufruf von `make` kompilieren Sie Ihr Programm. Die ersten drei Zeilen der Ausgabe Ihres Programms müssen so aussehen:

```
0   \0          32  \32         64    @         96    `
1   \1          33    !         65    A         97    a
2   \2          34    "         66    B         98    b
```

Die Ausgabe Ihres Programms muss identisch sein mit dem Inhalt der Datei `ascii_table.tsv` aus den Materialien. Dieses verifizieren Sie durch `make test`.

Aufgabe 2.3 (4 Punkte) Im Jahre 1202 ging Leonardo Pisano (genannt: Fibonacci) der folgenden Frage nach: "Wie viele Hasenpaare können als Nachkommen eines Ur-Hasenpaares innerhalb eines Jahres zur Welt kommen". Nehmen wir mal an, jedes Hasenpaar bekommt pro Monat ein Hasenpaar als Nachwuchs. Und jeder Hase ist geschlechtsreif, wenn er einen Monat auf der Welt ist. Außerdem sterben die Hasen natürlich nie. Im zweiten Monat gibt es dann 2 Hasenpaare, nach drei Monaten 3, nach vier Monaten 5 usw.

Fibonacci schrieb diesen Sachverhalt als folgendes Gleichungssystem auf:

$$\mathit{fib}(i) = \begin{cases} 0 & \text{falls } i = 0 \\ 1 & \text{falls } i = 1 \\ \mathit{fib}(i-1) + \mathit{fib}(i-2) & \text{falls } i \geq 2 \end{cases} \tag{1}$$

Für eine natürlich Zahl i ist $\mathit{fib}(i)$ die i-te Fibonacci-Zahl. Wir wollen hier nicht diskutieren, ob die Fibonacci-Zahlen die Größe der Hasenpopulation richtig vorraussagen. Es geht hier darum, die Fibonacci-Zahlen zu berechnen und auszugeben. Das kann anhand der obigen Gleichungen erfolgen:

$$\begin{aligned} \mathit{fib}(2) &= \mathit{fib}(2-1) + \mathit{fib}(2-2) = \mathit{fib}(1) + \mathit{fib}(0) = 1 + 0 = 1 \\ \mathit{fib}(3) &= \mathit{fib}(3-1) + \mathit{fib}(3-2) = \mathit{fib}(2) + \mathit{fib}(1) = 1 + 1 = 2 \\ \mathit{fib}(4) &= \mathit{fib}(4-1) + \mathit{fib}(4-2) = \mathit{fib}(3) + \mathit{fib}(2) = 2 + 1 = 3 \\ \mathit{fib}(5) &= \mathit{fib}(5-1) + \mathit{fib}(5-2) = \mathit{fib}(4) + \mathit{fib}(3) = 3 + 2 = 5 \\ &\vdots \end{aligned}$$

Wendet man dieses Schema an, dann erhält man z.B. die ersten 15 Fibonacci-Zahlen

$$0, 1, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89, 144, 233, 377$$

Implementieren Sie in einer Datei `fibonacci.c` ein C-Programm, das Fibonacci-Zahlen berechnet und zeilenweise zusammen mit dem Index ausgibt. D.h. in der i-ten Zeile der Ausgabe steht

$i \quad \mathit{fib}(i)$

durch einen Tabulator getrennt. Die Berechnung soll iterativ mit einer Schleife erfolgen. Sie sollen insbesondere nicht die rekursive Definition der Fibonacci-Zahlen in einer rekursiven Funktion implementieren.

Es sollen alle Fibonacci-Zahlen ausgegeben werden, die sich durch den gewählten vorzeichenlosen ganzzahligen Typ repräsentieren lassen. Wenn man für die Fibonacci-Zahlen den Typ **`unsigned int`** verwendet, dann ist die größte berechenbare Fibonacci-Zahl 2 971 215 073.

Es sollen durch drei verschiedenen Programme, je eins für die Typen **`unsigned short`**, **`unsigned int`** und **`unsigned long`**, die jeweiligen Fibonacci-Zahlen ausgegeben werden.

Zur Abstraktion des konkreten Typs verwenden Sie in Ihrem Programm das Typsynonym

`FibonacciNumberType`

und die beiden Makros

`FibonacciNumberType_max` und `FibonacciNumberType_print`,

deren Definition Sie in der Datei `fibonacci_template.c` finden und nicht ändern dürfen. Das erst Makro ist der maximale Wert, der sich durch den gewählten Typ repräsentieren lässt. Das zweite Makro dient zur Ausgabe einer Zeile im obigen Format.

Benennen Sie die Datei `fibonacci_template.c` um in `fibonacci.c` und implementieren Sie darin die `main`-Funktion, die keine Argumente hat (was durch **`void`** ausgedrückt wird).

Die Schleife zur Berechnung der Fibonacci-Zahlen muss abbrechen, sobald $\mathit{fib}(i-1) + \mathit{fib}(i-2)$ größer ist als `FibonacciNumberType_max`. Sie können aber diese Bedingung nicht direkt in Ihrem Programm verwenden, weil diese Summe ja nicht durch einen Wert des Typs `FibonacciNumberType` repräsentiert werden kann. Sie müssen sich eine äquivalente Bedingung überlegen, bei der die Werte der Teilausdrücke in eine `FibonacciNumberType`-Variable „passen“. Hier gibt es eine einfache Lösung.

Sie implementieren genau eine `main`-Funktion in `fibonacci.c`. Durch Aufruf von `make` werden daraus drei verschiedene ausführbare Programme kompiliert. Beschreiben Sie durch die Analyse der Datei `Makefile` und der bedingten Präprozessor-Anweisungen in `fibonacci.c`, wie das funktioniert. Hierzu müssen Sie sich nochmal die Folien zum C-Präprozessor ansehen.

Durch den Aufruf von `make test` im Verzeichnis zu dieser Aufgabe verifizieren Sie, dass Ihr Programm korrekt funktioniert.

Punkteverteilung:

1. 1 Punkt für die Implementierung der Schleife zu Berechnung der Fibonacci-Zahlen
2. 1 Punkt für die korrekte Definition der Abbruchbedingungen
3. 1 Punkt für funktionierende Tests
4. 1 Punkt für die Analyse des Makefiles und der bedingten Präprozessor-Anweisungen

Bitte die Lösungen zu diesen Aufgaben bis zum 12.05.2020 um 18:00 Uhr an pfn2@zbh.uni-hamburg.de schicken. Es erfolgt keine Besprechung der Lösungen.