Imperative Programmierung
–Aufgabenblatt 06 (41 + 3 Punkte)–

Hinweise

Die Lösungen der Aufgaben sind als PDF-Dokument bzw. C-Quelltext mit Hilfe des Versionskontrollsystems Subversion (SVN) abzugeben. Platzieren Sie das PDF-Dokument mit ihren Antworten im Ordner `a06` innerhalb Ihres Gruppenverzeichnisses[1]. Platzieren Sie die C-Quelltexte im Unterordner `a06/src`. C-Quelltexte müssen fehlerfrei mit den Optionen `-pedantic -Wall -Werror -std=c99` kompiliert werden können. Schreiben Sie in die abgegebenen Dateien die Namen und Matrikelnummern aller Gruppenmitglieder. Verspätete Abgaben oder Abgaben ohne Matrikelnummer werden nicht gewertet! Plagiate jeglicher Art führen zur Meldung beim Prüfungsausschuss und zum nicht bestehen des Moduls.

Ausgabe: 05.12.2024
Abgabe: 05.01.2025 bis 23:59 Uhr

Aufgabe 1 - Polynome (7 Punkte)

Für eine natürliche Zahl n ist $f(x) = a_n x^n + a_{n-1}x^{n-1} + \ldots + a_1 x + a_0$ mit $a_n \neq 0$ ein Polynom n-ten Grades. Ein solches Polynom lässt sich über die Koeffizienten a_0 bis a_n eindeutig beschreiben.

(a) Lesen Sie in der Funktion `main` den Grad n und die entsprechenden ganzzahligen Koeffizienten eines Polynoms n-ten Grades ein und speichern Sie diese in einem Array entsprechender Länge. Implementieren Sie die Funktion `print_polynomial`, die aus den Koeffizienten eines Polynoms n-ten Grades eine formschöne Ausgabe nach dem folgenden Muster erzeugt: $a_n * x\hat{}n + a_{n-1} * x\hat{}(n-1) + \ldots + a_1 * x + a_0$.
Bsp.: `f(x) = + 3 * x^2 + 4 * x - 3` für ein Polynom zweiten Grades. Geben Sie das Eingelesene Polynom anschließend mit Hilfe der Funktion `print_polynomial` aus. Beachten Sie korrekte Ausgabe negativer Koeffizienten. (4 Punkte)

(b) Implementieren Sie die Funktion `evaluate`, die den Funktionswert eines Polynoms an einer Stelle berechnet und das Ergebnis zurückliefert. Beispiel: Für das Polynom $f(x) = 3x^2 + 4x - 3$ soll an der Stelle $x = 1.0$ das Ergebnis 4.0 berechnet werden.
Hinweis: Verwenden sie keine Hilfsfunktionen aus `<math.h>`. (3 Punkte)

Aufgabe 2 - Reihensummen (8 Punkte)

Es sollen in der Datei `reihensumme.c` Funktionen mit vorgegebenen Formeln berechnet werden. Nutzen Sie hierzu Fließkommazahlen doppelter Genauigkeit. Verwenden Sie innerhalb der Funktionen keine Funktionen aus `<math.h>` (z.B. `abs` oder `pow`!)!

(a) Schreiben Sie eine Funktion `func1`, die die Reihensumme der unten angegebenen Formel bis zu einem vorgegebenen n berechnet! (3 Punkte)

$$\sum_{k=2}^{n} \frac{2k+1}{k^2(k+1)^2}$$

(b) Schreiben Sie eine Funktion `ln`, die nach der unten angegebenen Formel für x den Funktionswert $\ln(x)$ berechnet. Brechen Sie die Berechnung ab, sobald das aktuelle Reihenglied im Betrag den Wert EPS unterschreitet (Verwenden Sie die symbolische Konstante `#define EPS 0.0001`)! (3 Punkte)

$$\ln(x) = \sum_{n=1}^{\infty} \frac{(-1)^{n+1}}{n}(x-1)^n$$

(c) In Aufgabe b) wurde eine Formel zur näherungsweisen Berechnung des natürlichen Logarithmus angegeben, die beispielsweise für x-Werte zwischen 1.0 und 2.0 gültig ist. In (2 Punkte)

[1] Ihr Gruppenverzeichnis ist unter `https://svn.informatik.uni-rostock.de/lehre/ip2024/groups/<group>/` mit <group> $\in \{01, ..., 57\}$ zu finden.

`math.h` steht die Funktion `log(x)` zur Verfügung. Testen Sie in der `main`-Funktion ihre Funktion im Intervall von 1.0 bis 2.0 mit einer Schrittweite von 0.1 und geben Sie zum Vergleich das Ergebnis von `log(x)` an!

Aufgabe 3 - Matrizen (9 Punkte)

Matrizen lassen sich in C zum Beispiel durch zweidimensionale Arrays repräsentieren. Lösen Sie die folgenden Aufgaben mithilfe von zweidimensionalen Arrays.

(a) Implementieren Sie die Funktion `transpose`, die als Übergabeparameter eine mxn- und eine nxm-Matrix, sowie nötige Werte zu Anzahl der Zeilen und Spalten akzeptiert. Die mxn-Matrix soll daraufhin transponiert und in der zweiten übergebenen Matrix gespeichert werden. (5 Punkte)

(b) Implementieren Sie die Funktion `max_row_sum`, die die Zeilennummer mit der maximalen Zeilensumme (Zeilensumme ist Summe der Zahlen einer Zeile) von einer übergebenen Matrix als Rückgabewert liefert. (4 Punkte)

Aufgabe 4 - Spiegelzahlen mit Rekursion (8 Punkte)

Die Spiegelzahl einer natürlichen Zahl n erhält man, indem man die Ziffern von n in umgekehrter Reihenfolge aufschreibt. Beispielsweise ist 12589 die Spiegelzahl von 98521. Implementieren Sie die Lösungen der folgenden Aufgaben in der Datei `spiegel.c`. Arbeiten Sie hierbei rein mathematisch, d.h. nehmen Sie keine Konvertierung der Zahlen in Zeichenketten oder Arrays vor.

(a) Implementieren Sie die Funktion `count_digits`, die für eine Ganzzahl die Anzahl ihrer Dezimalziffern bestimmt. Lösen Sie das Problem mittels Iteration. (2 Punkte)

(b) Implementieren Sie die Funktion `count_digitsR`, die die Anzahl der Dezimalziffern mittels Rekursion bestimmt. (2 Punkte)

(c) Implementieren Sie die Funktion `reverse`, die für eine übergebene Zahl die Spiegelzahl berechnet. Lösen Sie das Problem mittels Iteration. (2 Punkte)

(d) Implementieren Sie die Funktion `reverseR`, die die Spiegelzahl mittels Rekursion berechnet. Hinweis: Es ist zulässig in der Funktion weitere Funktionen aufzurufen! (2 Punkte)

Auf der nächsten Seite geht es weiter!

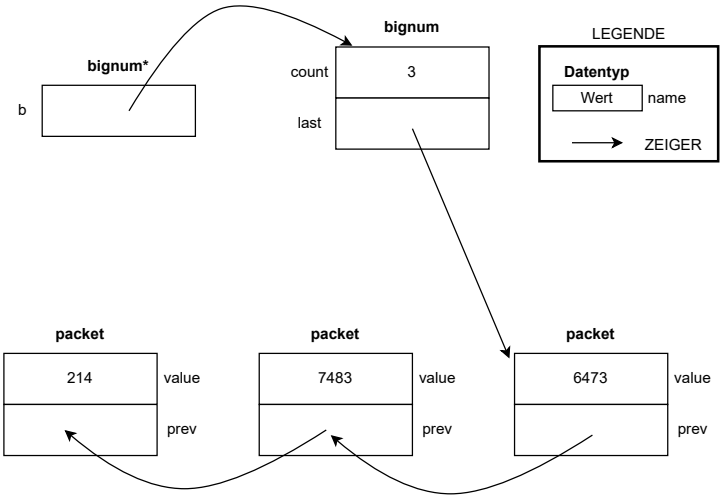
Aufgabe 5 - Eigene Datentypen und Dynamischer Speicher (9 Punkte)

Die in C vorhandenen Datentypen für ganze Zahlen, wie z.B. `int`, arbeiten mit einer festen Speichergröße (z.B. 32 Bit) und können daher auch nur Zahlen bis zu einer maximalen Größe repräsentieren. Um *positive ganze Zahlen* beliebiger Größe zu repräsentieren, soll mit Hilfe von `typedef` und Strukturen ein eigener Datentyp `bignum` definiert werden.

Die Idee besteht darin, die Stellen einer großen Zahl in kleinere Pakete fester Länge aufzuteilen. In einem Paket sollen so maximal 4 Dezimalziffern der großen Zahl in einem `int` gespeichert werden. Diese kann also alle Zahlen zwischen 0 und 9999 (inklusive) enthalten. Dies entspricht einer Darstellung der Zahl zur Basis 10000. Beispiel: die Zahl 21474836473 würde durch die drei "Pakete" $214(\times 10000^2) + 7483(\times 10000^1) + 6473(\times 10000^0)$ des Typs `int` repräsentiert werden.

Um das einfache dynamische Wachstum (oder Schrumpfen) der Anzahl der Stellen zu gewährleisten, werden die Pakete in einer *einfach verketteten Liste* verwaltet. Der Datentyp `bignum` verfügt dabei einen Eintrag über die Anzahl der Stellen `int count` sowie einen Zeiger auf die letzten (niedrigsten) vier Stellen der Zahl `packet* last`. Jedes `packet` verfügt über den Zeiger `prev`, über den man die Stellen-Pakete einer `bignum`-Zahl rückwärts durchwandern kann. Die unten stehende Abbildung gibt einen schematischen Überblick für den Fall einer Variablen `b` vom Typ Zeiger auf `bignum`, welche die Zahl 21474836473 repräsentiert.

Nutzen Sie für die folgenden Aufgaben die vorgegebene Datei `bignum_template.c` von Stud.IP!



(a) Definieren Sie mit Hilfe von `typedef` den Datentyp `bignum`, der die *Anzahl der Pakete* (`count`) und den *Zeiger auf das letzte* (`last`) `packet` aufnehmen kann. Verwenden Sie für ein einzelnes Paket (`packet`) die im Template vorgegebene Definition. Achten Sie bei den folgenden Aufgaben darauf, dass `count` nach einer Berechnung stets den richtigen Wert enthält. (1 Punkt)

(b) Implementieren Sie die Funktion `void print(bignum b)`, die eine `bignum` mittels `printf` auf der Standardausgabe darstellt. Dabei sollen die einzelnen Pakete ausgehend von `last` ausgegeben werden (Hinweis: die Darstellung ist dann nicht als Dezimaldarstellung interpretierbar!). In Klammern soll hinter der Zahl die Anzahl der Pakete ausgegeben werden. (2 Punkte)

(c) Implementieren Sie die Funktion `bignum* input(int count, int packets[])`, die aus einem Array von `int`-Zahlen eine neue `bignum` erzeugt. Nehmen Sie an, dass die Einträge des Felds `packets[]` stets maximal vierstellige Zahlen beinhalten und sortiert von der höchstwertigsten Stelle (in `packets[0]`) zur niedrigwertigsten Stelle (in `packets[count - 1]`) enthalten sind. (2 Punkte)

(d) Implementieren Sie die Funktion `bignum* add(bignum a, bignum b)`, die zwei `bignum`s addiert und eine entsprechende neue `bignum` zurück gibt. Achten Sie dabei darauf, den Überlauf, der z.B. bei der Addition der Pakete 9999 und 1 auftritt, richtig zu behandeln und eine entsprechend größere `bignum` als Ergebnis zurück zu geben. Im vorigen Beispiel also 10000, wobei für die erste Stelle ein neues Paket notwendig wird. Zur Vereinfachung darf die resultierende `bignum` mit umgekehrter Reihenfolge der Pakete (vom höchsten zum niedrigstelligsten Paket) konstruiert werden. (4 Punkte)

Zusatz: Implementieren Sie eine Funktion `void reverse(bignum* num)`, die die Reihenfolge der Pakete in einer `bignum` umkehrt. Diese kann verwendet werden um das Ergebnis nach dem Aufrufen von `sum` zu korrigieren. (3 Zusatzpunkte).