Universität Augsburg
Institut für Informatik

Übung zur Vorlesung Informatik 1 WiSe 2023/2024
Fakultät für Angewandte Informatik
Institut für Informatik
Prof. Dr. J. Hähner, P. Schalk, V. Schmid, S. Lutz, J. Linne 13.12.2023

Betreutes Programmieren 7

Aufgabe 7 ** *(Mathematische Funktionen)*

Diesmal sollen Sie einige mathematischen Bibliotheksfunktionen aus `math.h` mittels eigener Funktionen oder Makros (teilweise) nachimplementieren, in einer eigenen Übersetzungseinheit zur Verfügung stellen und in einem Hauptprogramm testen. Bei den Implementierungen dürfen dabei natürlich die entsprechenden Bibliotheksfunktionen nicht verwendet werden. Gehen Sie dabei Schritt für Schritt gemäß der nachfolgenden Beschreibung vor.

a) Implementieren Sie eine eigene Header-Datei `my_math.h` mit folgendem Inhalt:

- Definition eines Makros `MY_FABS`, das dieselbe Funktionlität hat wie die Bibliotheksfunktion `fabs`.
- Deklaration einer Funktion `double my_pow(double x, int y)`, die bzgl. der erlaubten Argumente dieselbe Funktionalität hat wie die Bibliotheksfunktion `pow` (diese sieht für `y` sogar den Datentyp `double` vor).

b) Implementieren Sie eine C-Datei `my_math.c` nach folgendem Schema:

- Einbindung der Header-Datei aus der ersten Teilaufgabe.
- Implementierung der Funktion `my_pow`.

  Hinweise:
  - Beim Aufruf `my_pow(a, b)` mit `b > 0` kann dazu in einer Schleife eine Hilfsvariable, beginnend bei `1.0`, `b`-mal mit `a` multipliziert werden.
  - Beim Aufruf `my_pow(a, b)` mit `b == 0` ist der Rückgabewert `1.0`.
  - Beim Aufruf `my_pow(a, b)` mit `b < 0` ist der Rückgabewert `1 / my_pow(a, -b)`.
  - Berücksichtigen Sie für alle Fehlerfälle die folgenden an die Dokumentation von `math.h` unter den http://www2.hs-fulda.de/~klingebiel/c-stdlib/math.htm angelehnten Vorgaben (tatsächlich ist die Dokumentation in `math.h` allgemeiner und weniger exakt, lässt also mehr Raum für undefiniertes Verhalten – Sie sollen es hier aber etwas einfacher haben):
    * Fehlerbehandlungen mit der globalen Variable `errno`: Die Makros `EDOM` und `ERANGE` (die man in `<errno.h>` findet), sind von Null verschiedene ganzzahlige Konstanten, mit denen Fehler im Argument- und Resultatbereich von `my_pow` angezeigt werden.
    * Ein Argumentfehler (*domain error*) liegt vor, wenn ein Argument nicht in dem Bereich liegt, für den `my_pow` definiert ist. Bei einem Argumentfehler erhält `errno` den Wert `EDOM`; der Resultatwert ist `0`.

* Ein Resultatfehler (*range error*) liegt vor, wenn das Resultat von `my_pow` wegen eines Unter- oder Überlaufs nicht als `double` dargestellt werden kann. Bei einem Resultatfehler liefert `my_pow` bei Unterlauf den Wert `(+/-)0.0` bzw. bei Überlauf den Wert `(+/-)HUGE_VAL` und `errno` erhält den Wert `ERANGE`.
  Der Resultatwert bei einem Unterlauf ist dabei `+0.0`, wenn das nicht darstellbare Ergebnis positiv wäre, und `-0.0`, wenn das nicht darstellbare Ergebnis negativ wäre. Analog dazu ist der Resultatwert bei einem Überlauf `+HUGE_VAL`, wenn das nicht darstellbare Ergebnis positiv wäre, und `-HUGE_VAL`, wenn das nicht darstellbare Ergebnis negativ wäre.
* Überlegen Sie, in welchen Fällen es zu einem Unterlauf (also einem Ergebnis `< DBL_MIN`) bzw. einem Überlauf (also einem Ergebnis `> DBL_MAX`) kommen kann und wie Sie einen Unter-/Überlauf feststellen können. Bei welchen Werten von `x` kann ein Unter-/Überlauf auftreten? Wie können Sie währen der schrittweisen Berechnung des Ergebnisses feststellen, ob es bei der nächsten Multiplikation zu einem Unter-/Überlauf kommen würde? Von den Werten welcher Parameter/-Variablen hängt es ab, ob im nächsten Schritt ein Unter-/Überlauflauf auftritt?

Beispiel-Fehlerfälle:

- `my_pow(0, -4)` liefert `0.0` zurück und setzt `errno` auf `EDOM` (Argumentfehler gemäß Definition von `pow`).
- `my_pow(-1.0e-10, 111)` liefert `-0.0` zurück und setzt `errno` auf `ERANGE` (Unterlauf).
- `my_pow(2.0e+20, 222)` liefert `HUGE_VAL` zurück und setzt `errno` auf `ERANGE` (Überlauf).

c) Test mit vorgegebenem Hauptprogramm:

Testen Sie nun Ihre Implementierungen mit dem vorgegebenen Hauptprogramm `bp07.c`, das jede der obigen Funktionen angewendet auf geeignete Benutzereingaben testet. Für die Verarbeitung der Benutzereingaben wird eine Quelldatei `input.c` inklusive passender Header-Datei `input.h` mit vorgegebenen Funktionen zur Verfügung gestellt, die dann im Hauptprogramm aufgerufen werden.

Schließlich kompilieren und verbinden Sie den in dieser Aufgabe vorgegebenen und den von Ihnen implementierten Code zu einem Programm.

Dokumentation der vorgegebenen Funktionen:

- `int flush_buff(void)`: Leert den Eingabestrom. Gibt bei einem Pufferfehler `0`, sonst `1` zurück.
- `double read_double(void)`: Wandelt, falls möglich, eine Benutzereingabe in einen `double`-Wert aus dem Bereich `]DBL_MIN;DBL_MAX[` um und gibt im Erfolgsfall den umgewandelten Wert zurück. Ist komplette Umwandlung der Benutzereingabe nicht möglich, wird der Wert `DBL_MAX` zurückgeliefert. Tritt beim Einlesen ein Pufferfehler auf, wird der Wert `DBL_MIN` zurückgeliefert. Die Funktion `read_double` sorgt bereits dafür, dass nach Beendigung immer ein leerer Puffer vorliegt.
- `void my_pow_tests(void)`: Führt verschiedene Tests mit der implementierten `my_pow`-Funktion durch und gibt zu jedem Test jeweils das Ergebnis und danach den aktuellen Wert von `errno` aus.