

Wstęp do programowania, potok imperatywny (Info, I rok) 16/17, laboratorium

Moja strona domowa ► Informatyka ► Informatyczne studia I stopnia ► I rok ►
WPI.LAB.INFO.I.16/17 ► Zadanie 3 ► Zadanie 3: Liliczby



NAWIGACJA

Moja strona domowa

■ Strona główna

Strony

Bieżący

przedmiot

WPI.LAB.INFO

.I.16/17

Uczestnicy

Odznaki

Główne

składowe


Zadanie 0

(treningowe
)


Zadanie 1

Zadanie 2

Zadanie 3

 **Zadanie
3:**

Liliczby

 Zadanie
3:

poprawn
ość

Zadanie 4
(poprawkow
e)

Moje kursy

Zadanie 3: Liliczby

Wprowadzenie

Liliczba (ang. *nunumber*) to rekurencyjna reprezentacja nieujemnej liczby całkowitej za pomocą ciągu cyfr, które również są liliczbami. Reprezentacja ta, sformułowana w nieco inny ale równoważny sposób, była rozważana przez Donalda Knutha.

Wartością liliczby jest suma potęg dwójki o wykładnikach będących wartościami cyfr tej liliczby. Powiemy, że liliczba jest znormalizowana, jeśli jej cyfry są znormalizowanymi liliczbami i są uporządkowane rosnąco według wartości.

Tekstowym zapisem liliczby jest słowo języka z poniższą gramatyką w rozszerzonej notacji BNF:

```
<liliczba> ::= { "Y" <cyfra> } "Z"  
<cyfra> ::= <liliczba>
```

Można zauważyć, że jest to język słów powstałych przez dopisanie symbolu **Z** na koniec wyrażenia nawiasowego, w którym **Y** pełni rolę nawiasu otwierającego a **Z** to nawias zamykający.

Wszystkie liliczby, których zapisem są poniższe słowa

```
YZYZYZYZYZYZYZYZYZ  
YZYZYZYZYZYZYZYZYZ  
YZYZYZYZYZYZYZYZYZ  
YYYZZYZYZYZYZYZYZ  
YYYZZYZYZYZYZYZYZ  
YYYZZYZYZYZYZYZYZ  
YYYZZYZYZYZYZYZYZ  
YYZZYYZZYYZZYYZZ  
YYZZYYZZYYZZYZ  
YYZZYYZZYYZZYZ
```

mają wartość 10. Ostatnia z nich jest znormalizowana.

Administracja
kursem

Inne przykłady znormalizowanych liliżeb są w poniższej tabelce:

[illegible]

Wartość	Liliczba
1000	YYZYYZZZYYZYZZZZZZYYZZYYZZZZYYZYZZYYZZZZYY
1009	YZYYYYZZZZYYZYZZZZZZYYZZYYZZZZYYZYZZYYZZZZY
1024	YYYZZYYZYZZZZZ
10000	YYYYZZZZYYZYZZZZZZYYZYZZZZZZYYZZYYZYZZZZYYZ
65536	YYYYYZZZZZZ
200791	YZYYZZYYZZZZYYZZZZZZYYZZYYZZZZYYZZYYZZYYZZZ
2^{65536}	YYYYYYZZZZZZZ
2^{65537}	YYZYYYYYZZZZZZZ
$2^{(2^{65536})}$	YYYYYYYZZZZZZZZ
$2^{(2^{(2^{65536})})}$	YYYYYYYZZZZZZZZZ
$5+2^{(2^{(2^{65536})})}$	YZYYZZZZYYZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZ

Zwracamy uwagę, że pozycyjny zapis binarny trzech ostatnich wymienionych wartości miałby więcej bitów, niż jest atomów na Ziemi.

Polecenie

Napisz program, który wczyta dwie, nie koniecznie znormalizowane, liliczbę zapisane na wejściu i wypisze ich iloczyn jako znormalizowaną liliczbę.

Postać danych

Na wejściu programu są dwa wiersze. W każdym z nich jest zapisana jedna liliczba. Oprócz liter `Y` i `Z` nie ma tam żadnych innych znaków, nawet spacji.

Postać wyniku

Program wypisuje jeden, prawidłowo zakończony za pomocą `'\n'`, wiersz z zapisem znormalizowanej liliczbę.

Przykłady

- Dla danych przykład1.in wynikiem programu powinno być przykład1.out ($3 * 6 = 18$).
- Dla danych przykład2.in wynikiem programu powinno być przykład2.out ($12 * 12 = 144$).
- Dla danych przykład3.in wynikiem programu powinno być przykład3.out ($1009 * 199 = 200791$).

Uwagi i wskazówki

- Wolno założyć, że dane są poprawne.
- Program nie powinien nakładać żadnych ograniczeń na rozmiar danych i

- Oczekujemy rozwiązania o koszcie wielomianowym względem rozmiaru danych.
- Przyjmujemy, że wynik funkcji `main()` inny niż 0 informuje o błędzie wykonania programu.
- Do treści zadania dołączone są pliki `.in` z danymi przykładowymi i pliki `.out` z wynikami wzorcowymi.
- Poprawność wyniku można sprawdzić, przekierowując na wejście programu zawartość pliku z przykładowymi danymi i porównując rezultat, za pomocą programu `diff`, z plikiem zawierającym wynik wzorcowy, np.:

```
< przyklad.in ./liliczby | diff - przyklad.out
```

Wynik uznajemy za poprawny tylko, jeśli jest identyczny z wynikiem wzorcowym.

- Program, który zarezerwował pamięć funkcjami `malloc`, `realloc` itp. ma obowiązek zwolnić ją funkcją `free`. Jeśli tego nie zrobi, występuje zjawisko wycieku pamięci, które uznajemy za błąd. W wykryciu tego i innych błędów może pomóc program `valgrind`. By z niego skorzystać, kompilujemy swój program z dodatkową opcją `-g`, np. poleceniem:

```
gcc -std=c89 -pedantic -Wall -Wextra -Werror -g liliczby.c -o liliczby
```

Spowoduje to dołączenie do programu wykonywalnego informacji pomagających w lokalizacji błędu. Tak skompilowany program uruchamiamy pod kontrolą `valgrind` poleceniem:

```
valgrind --leak-check=full ./liliczby
```

Opcja `--leak-check=full` wskazuje, że chcemy, między innymi, wykryć wycieki pamięci i znaleźć ich źródło.

Na zakończenie wykonania programu przez `valgrind`, na wyjście diagnostyczne wypisywany jest raport. Jeżeli nie wykryto błędów, może on mieć postać np.

```

==46974== Memcheck, a memory error detector
==46974== Copyright (C) 2002-2015, and GNU GPL'd, by Julian
Seward et al.
==46974== Using Valgrind-3.11.0 and LibVEX; rerun with -h fo
r copyright info
==46974== Command: ./liliczby
==46974==
==46974==
==46974== HEAP SUMMARY:
==46974==      in use at exit: 0 bytes in 0 blocks
==46974==    total heap usage: 97 allocs, 97 frees, 9,712 byt
es allocated
==46974==
==46974== All heap blocks were freed -- no leaks are possibl
e
==46974==
==46974== For counts of detected and suppressed errors, reru
n with: -v
==46974== ERROR SUMMARY: 0 errors from 0 contexts (suppresse
d: 0 from 0)

```

Wartość inna niż 0 po `ERROR SUMMARY` informuje, że wykryto błąd. W raporcie będą też wskazówki pomagające w lokalizacji błędu.

- Rozwiązania do testów będą kompilowane poleceniem:

```
gcc -std=c89 -pedantic -Wall -Wextra -Werror -g nazwa.c -o n
azwa
```

Wszystkie wymienione opcje kompilatora są obowiązkowe i nie wolno dodawać do nich żadnych innych.

- Podczas testów rozwiązania będą uruchamiane pod kontrolą programu `valgrind`. Jeżeli wykryje on błąd, np. wyciek pamięci, to przyjmujemy, że program testu nie przeszedł nawet, jeśli jego wynik będzie prawidłowy.
- Tekstowy zapis liliczby można przekształcić na wyrażenie arytmetyczne, zastępując kończące `Z` przez `-0`, każdą parę znaków `YZ` przez `Y0Z`, parę `ZY` przez `Z+Y` a następnie każdy znak `Y` przez `2^` i każdy znak `Z` przez `)`.

Za pomocą programu `sed` zastępującego wzorce w tekście oraz kalkulatora wyrażeń arytmetycznych `bc` możemy poznać dziesiętny zapis wartości liliczby. Np. polecenie:

```
echo YZZZYYYYZZZZYYZZYYYYZZZZ | sed 's:Z$:-0:g; s:YZ:Y0Z:g; s
:ZY:Z+Y:g; s:Y:2^(:g; s:Z:):g' | bc
```

wypisze:

```
50
```

- Rozwiązanie zadania wymaga zastosowania rekursji i dynamicznych struktur danych.
- Iloczyn dwóch liczb można wyznaczyć, korzystając ze wzoru:

$$\left(\sum_{i=0}^m 2^{a_i} \right) \left(\sum_{j=0}^n 2^{b_j} \right) = \sum_{i=0}^m \sum_{j=0}^n 2^{a_i+b_j}$$

Będzie do tego potrzebne dodawanie.


Obliczenie sumy znormalizowanych liczb wymaga scalenia uporządkowanych rosnąco list cyfr. W przypadku, gdy ta sama cyfra występuje w obu dodawanych liczbach, realizujemy przeniesienie.

By scalić listy cyfr potrzebujemy porównania. Cyfry liczb porównujemy w kolejności od najbardziej znaczących.

Pracę nad rozwiązaniem proponujemy zacząć od programu, który czyta i wypisuje liczbę. Następnie zmieniamy go, kolejno, w program czytający dwie liczby i wypisujący wynik ich porównania, ich sumę i na koniec ich iloczyn.

 przykład1.in
 przykład1.out
 przykład2.in
 przykład2.out
 przykład3.in
 przykład3.out

Status przesłanego zadania

Numer próby	To jest próba nr 1.
Status przesłanego zadania	Przesłane do oceny
Stan oceniania	Nie ocenione
Termin oddania	środa, 11 styczeń 2017, 10:00
Pozostały czas	Zadanie zostało złożone 17 godz. 50 min. przed terminem
Ostatnio modyfikowane	wtorek, 10 styczeń 2017, 16:09
Przesyłane pliki	 PIOTR_SZUBERSKI_ZADANIE3_LILICZBY.c
Komentarz do przesłanego zadania	► Komentarze (0)

Jesteś zalogowany(a) jako Piotr Szuberski (Wyloguj)
WPI.LAB.INFO.I.16/17

Moodle, wersja 3.1.1 | moodle@mimuw.edu.pl