Kalcoolator

Dokumentacja programu wykonującego działania na liczbach z systemów liczbowych o podstawach od 2 do 16.

10.01.2023

Wojciech Matejuk
Wydział Matematyki i Nauk Informacyjnych
Politechnika Warszawska

Spis treści

1. Opis projektu

Program umożliwia wykonywanie działań dodawania, mnożenia, dzielenia, potęgowania oraz zamiany podstawy systemów liczbowych dla liczb o długości do 80 znaków, zapisanych w systemach od 2 do 16. Napisany jest w języku C.

Jako argument program przyjmuje ścieżkę (względną lub nie) do pliku wejściowego, który musi być plikiem tekstowym. Najlepiej gdy jest on zakończony na *„_in.txt”* Każde działanie (poza zamianą systemu) w pliku wejściowym powinno być zapisane w następujący sposób (// - pominąć):

<znak działania> <podstawa systemu liczbowego>

<pierwsza liczba w działaniu>

<druga liczba w działaniu>

//miejsce na odpowiedz

<znak działania>(itd...)

Zamiana systemu musi mieć w pliku *in.txt* postać:

<podstawa systemu wczytywanej liczby> <podstawa systemu do którego chcemy przekonwertować liczbę>

<liczba>

//miejsce na odpowiedz

itd. wg powyższego schematu.

Wynikiem działania programu jest plik zakończony na "*_out.txt*", zawierający treść pliku wejściowego uzupełnioną o wyniki działań. Jeśli plik wejściowy zakończony jest na „_in.txt”, końcówka zostaje podmieniona. Przykładowe pliki wejściowe i wyjściowe znajdują się w folderze *examples*.

2. Opis rozwiązania

2.1 Zmienne globalne

Program na początku definiuje dwie stałe zmienne:

const int *SIZE*

Maksymalną długość liczby w programie + 1. Ma wartość 81.

const char* *nums*

Ciąg znaków „0123456789ABCDEF”. Miejsce znaku w tablicy odpowiada jego wartości, więc aby pobrać znak o wartości k wystarczy użyć nums[k], gdzie 0 <= k <= 15 i k jest liczbą całkowitą.

2.2 Funkcje pomocnicze:

Podczas wykonywania działań program korzysta z dodatkowych funkcji, które nie są wywoływanie bezpośrednio w funkcji main. Są to:

void *turn*(char* *str*)

Funkcja odwraca tablicę znaków (tak jak funkcja strrev(), z której nie korzystam w tej implementacji).

Np.:

$$AF58B \rightarrow B85FA$$

void *swap*(char *a*, char** *b*)**

Zamienia wskaźniki dwóch tablic znaków.

int *val*(char *a*)

Zwraca wartość znaku w systemie dziesiątkowym. Zwraca -1 gdy a nie może być zinterpretowana jako cyfra. Np.:

Dla a = ‘A’ zwróci 10.

Dla a = ‘5’ zwróci 5.

Dla a = ‘F’ zwróci 15.

Dla a = ‘K’ zwróci -1.

int *is_greater*(char* greater, char* less)

Jeśli długości tablic greater i less się różnią, to większa jest dłuższa tablica.

Jeśli tablice są tej samej długości, sprawdza po kolei znaki od najbardziej znaczącego aż trafi na różniące się. Jeśli liczba greater jest większa zwraca 1, jeśli nie – zwraca 0.

Np.:

Dla greater = „AAA156F”, less = „AAA385F” zwróci 0.

Dla greater = „ABC123”, less = „ABC” zwróci 1.

int *are_equal*(char* *a*, char* *b*)

Porównuje długości tablic a jeśli są równe to sprawdza czy wszystkie znaki są równe. Zwraca 1 jeśli tablice są takie same, 0 jeśli się różnią.

int *change_to_int*(char* *a*, int *base*)

Zaczynając od najmniej znaczącej cyfry liczby ‘a’ pobiera ich wartości funkcją val i przemnaża je przez bazę systemu liczby ‘a’ podniesioną do potęgi o takiej wartości, na jakim miejscu stoi znak w liczbie ‘a’ licząc od prawej strony od 0. Uzyskane liczby sumuje w zmiennej lokalnej out.

Zwraca out.

Np. dla a = „12A”, base = 13 tworzy out = 0 i sumuje:

$$out = val(A) * 13^0 + val(2) * 13^1 + val(1) * 13^2$$

Czyli

$$out = 10 * 13^0 + 2 * 13^1 + 1 * 13^2$$
$$out = 205$$

char* *change_from_int*(int a, int base, char* out)

Dopóki liczba *a* jest większa od 0 oblicza resztę z dzielenia tej liczby przez podstawę systemu liczbowego w jakim chcemy zapisać liczbę początkową i odejmuję tę resztę od a. Zapisuje w zmiennej i liczbę takich odejmowań. Do tablicy out zapisuje na i-tym miejscu znak nums[reszta].

Później odwraca tablicę out funkcją *turn* i zwraca out.

Np. dla a = 129, base = 14:

$$129\ mod\ 14 = 3, \frac{129-3}{14} = 9 \rightarrow out[0] = nums[3], a = 9$$
$$9\ mod\ 14 = 9, \frac{9-9}{14} = 0 \rightarrow out[1] = nums[9], a = 0$$
$$turn(out) \rightarrow out = 93$$

2.3 Funkcje główne

Funkcje wywoływane w funkcji *main*(). Zwracają -1 gdy wystąpi błąd powodowany przez nieprawidłowe dane wejściowe i 0 gdy operacja zostanie wykonana prawidłowo. Wynik jest zapisywany w tablicy podanej jako ostatni argument podczas wywoływania tych funkcji.

int *format*(char* *in*, int *base*)

Funkcja usuwa niepotrzebne zera na początku liczb pobieranych z pliku wejściowego. Zwraca -1 gdy liczba posiada niepoprawne cyfry. Np. (tablica in)

$000025F06 \rightarrow 25F06$, zwraca 0

$000000 \rightarrow 0$, zwraca 0

$TONIELICZBA \rightarrow$ <pusta tablica>, zwraca -1

int *add*(char* *a*, char* *b*, int *base*, char* *out*)

Używa zmiennej *car* (carry), na początku równej 0.

Tak jak w dodawaniu pisemnym, dodaje *car* i wartości cyfr *a* i *b*, zaczynając od najmniej znaczących. Reszta dzielenia takiej sumy, przechowywanej w zmiennej num, przez *base* jest zapisywana na kolejnym miejscu tablicy out. Iloraz takiej sumy i *base* zaokrąglony w dół jest zapisywany do zmiennej *car*.

Jeśli któraś liczba ma więcej cyfr to na następnych miejscach w tablicy out zapisywana jest suma kolejnych, bardziej znaczących, cyfr dłuższej i zmiennej car.

Jeśli po zsumowaniu wszystkich znaków *car* jest większe niż 0 to na ostatnim miejscu tablicy *out* zapisywany jest znak o wartości car. Następnie tablica *out* jest odwracana za pomocą funkcji *turn*. Np. dla a = „FF", b = „3", base = 16:

$$num = \big(car + val(F) + val(3)\big)\ mod\ 16 = (0 + 15 + 3)\ mod\ 16 = 2$$

$$car = \frac{car + val(F) + val(3)}{16} = \frac{0 + 15 + 3}{16} = 1 (zaokrąglamy\ w\ dół)$$

$$out[0] = nums[0]$$

$$num = \big(car + val(F)\big)\ mod\ 16 = (1 + 15)\ mod\ 16 = 0$$

$$car = \frac{car + val(F)}{16} = \frac{1 + 15}{16} = 1$$

$$out[1] = nums[0]$$

$$out[2] = nums[car] = nums[1]$$

$$turn(out) \rightarrow out = 102$$

int *change*(char* *a*, int *from_base*, int *to_base*, char* *out*)

Tworzy zmienną *i* = 0. Po kolei dla każdego znaku w liczbie *a* od końca:

Mnoży wartość tego znaku przez bazę *from_base* podniesioną do *i*-tej potęgi, konwertuje te wartości za pomocą *change_from_int* do bazy *to_base* i sumuje tak otrzymane liczby w bazie *to_base* za pomocą funkcji *add*. Zwiększa *i* o jeden.

Później, otrzymany tak wynik zapisuje do tablicy *out*.

Np. dla a = AB, from_base=13, to_base=16:

$$num = val(B) * 13^0 = 11 = B_{16}$$

$$sum = B_{16}$$

$$num = val(A) * 13^1 = 10 * 13 = 130 = 82_{16}$$

$$sum = 82_{16} + B_{16} = 8D_{16}$$

$$out = "8D"$$

int *multiply*(char* *a*, char* *b*, int *base*, char* *out*)

Tak jak w mnożeniu pisemnym, dla każdego znaku liczby *b* mnoży jego wartość przez liczbę *a* i zapisuje wynik w zmiennej *comp*, na której początku znajduje się tyle zer ile znaków liczby *b* już mnożyliśmy. Na kolejnym miejscu *comp* zapisywana jest reszta z dzielenia iloczynu przez *base*. Jeśli iloczyn jest większy od *base* to zapisuje iloraz tego iloczynu i *base* zaokrąglony w dół jako *car* i dodaje go do następnego iloczynu lub zapisuje na koniec *comp*. Potem *comp* jest odwracane funkcją *turn* i sumowane za pomocą funkcji *add*.

Tak otrzymana suma jest zapisywana w tablicy out.

Np. dla a=32, b=12, base=4:

$$i = 0$$
$$num = \big(car + val(2) * val(2)\big)\ mod\ 4 = (0 + 2 * 2)\ mod\ 4 = 0$$
$$car = \frac{car + val(2) * val(2)}{4} = \frac{4}{4} = 1$$
$$comp[0] = nums[0]$$
$$num = \big(car + val(2) * val(3)\big) mod 4 = (1 + 2 * 3) mod 4 = 3$$
$$car = \frac{\big(car + val(2) * val(3)\big)}{4} = \frac{1 + 2 * 3}{4} = 1\ (zaokrąglenie\ w\ dół)$$
$$comp[1] = nums[3]$$
$$car = 0 \rightarrow\ comp[2] = nums[car] = nums[1]$$
$$turn(comp) \rightarrow comp = 130$$
$$sum = sum + comp = 0 + 130 = 130$$

$$i = 1 \rightarrow comp[0] = 0$$
$$num = \big(car + val(1) * val(2)\big)\ mod\ 4 = (0 + 1 * 2)\ mod\ 4 = 2$$
$$car = \frac{car + val(1) * val(2)}{4} = \frac{2}{4} = 0$$
$$comp[1] = nums[2]$$
$$num = \big(car + val(1) * val(3)\big) mod 4 = (0 + 1 * 3) mod 4 = 3$$
$$car = \frac{\big(car + val(1) * val(3)\big)}{4} = \frac{0 + 1 * 3}{4} = 0\ (zaokrąglenie\ w\ dół)$$
$$comp[2] = nums[3]$$
$$car = 0 \rightarrow\ NOP$$
$$turn(comp) \rightarrow comp = 320$$
$$sum = sum + comp = 130 + 320 = 1110$$
$$out = 1110$$

int *power*(char* *a*, char* *b*, int *base*, char* *out*)

Jeśli *b* jest 0 zapisze do *out* „1”, jeśli jest równa 1, zapisze a do *out*.

Jeśli jest większa od 1 to stworzy zmienną *i* równą 2, temp1 = a i temp2=a i dopóki *i**2 jest mniejsze równe *b* będzie mnożyć *temp1* przez *temp2* za pomocą funkcji *multiply,* zapisywać wynik w temp1 i temp2 i mnożyć *i* razy 2. Potem dopóki *i* będzie dalej mnożyć ale zapisując w zmiennej temp1 wynik, a w temp2 liczbę a i zwiększać i o jeden. Na koniec wynik zapisywany jest w tablicy out.

Np. dla a=2, b=6, base =10:

$$temp1 = temp1 * temp2 = 2 * 2 = 4, i = 2$$
$$temp2 = 4$$
$$temp1 = temp1 * temp2 = 4 * 4 = 16, i = 4$$
$$temp2 = a = 2, bo\ i * 2 > 6$$
$$temp1 = temp1 * temp2 = 16 * 2 = 32$$
$$temp1 = temp1 * temp2 = 32 * 2 = 64$$
$$out = 64$$

int *divide*(char* *a*, char* *b*, int *base*, char* *out*)

Najpierw sprawdza do której potęgi można podnieść liczbę *b* zanim będzie większa od *a*. Wykorzystuje funkcje *power* Następnie sprawdza ile razy można do tak otrzymanej potęgi liczby *b* dodać coraz mniejsze potęgi tej liczby zanim otrzyma się liczbę większą od *a*. Jeśli dodaje potęgę b^{k}, to do wyniku *out* dodaje b^{k-1}. Tak otrzymany wynik jest zaokrąglonym w dół ilorazem *a* i *b*.

Np. dla a=2, b=22, base=10:

$$2^5 = 32 > 22$$
$$2^4 = 16 < 22$$
$$2^4 + 2^3 = 16 + 8 = 24 > 22$$
$$2^4 + 2^2 = 16 + 4 = 20 < 22$$
$$2^4 + 2^2 + 2^2 = 16 + 4 + 4 = 24 > 22$$
$$2^4 + 2^2 + 2^1 = 16 + 4 + 2 = 22 = 22 = b$$
$$out = 2^{4-1} + 2^{(2-1)} + 2^{(1-1)} = 8 + 2 + 1 = 11$$

int *modulo*(char* *a*, char* *b*, int *base*, char* *mid*)

Najpierw znajduje największą wielokrotność liczby *b*, mniejszą od *a* i zapisuje ją jako tablicę *prev*. Potem pomiędzy nią a najmniejszą wielokrotnością *b*, większą od *a*, wyszukuje binarnie liczbę *a*. Odbywa się to dodając wartość środkową do liczby *prev*. Ta wartość zapisywana jest jako tablica *mid*.

Np. dla a=27, b=8, base=10:

$$prev = 24$$
$$tail = 0, head = 8, mid = 4$$
$$prev + mid = 24 + 4 = 28 > 27$$
$$tail = 0, head = 4, mid = 2$$
$$prev + mid = 24 + 2 = 26 < 27$$
$$tail = 2, head = 4, mid = 3$$
$$prev + mid = 24 + 3 = 27$$
$$mid = 3$$

2.4 Funkcja *main*(int *argc*, int *argv*[])

Przyjmuje argument będący ścieżką do wejściowego pliku tekstowego. Otwiera go oraz otwiera lub tworzy plik tekstowy którego nazwa jest taka jak pliku wejściowego, ale ze zmienionym ostatnim członem *„_in.txt”* na *„_out.txt*”. Jeśli plik wejściowy nie posiada nazwy zakończonej na *„_in.txt”*, to po prostu *„_out.txt”* dodawane jest na koniec nazwy tego pliku. Tworzy zmienne *str1*, *str2*, *out*, *base_str* i *base*.

Dopóki nie trafi na znak EOF, odczytuje z pliku wejściowego po jednym ciągu znaków, oddzielonym spacją lub innym znakiem białym. Wstawia do pliku wyjściowego ten znak oraz spacje. Jeśli tym co odczyta jest znak +, *, /, ^ lub %, odczytuje dalej bazę i zapisuje ją do *base* oraz liczby na których ma zostać wykonana operacja i zapisuje je do *str1* i *str2*.

Na bieżąco wpisuje te dane do pliku wyjściowego. Wywołuje pożądaną operację:

\+ - *add*

/ - *divide*

* - *multiply*

^ - *power*

% - *modulo*

Jako argument *out* podaje lokalną tablicę *out*.

Zapisuje zawartość *out* w pliku wyjściowym pod opisem operacji.

Wraca do początku pętli.

Jeśli funkcja odczyta z pliku wejściowego na początku opisu liczbę, to znaczy że zadaną operacją jest zmiana systemów liczbowych – liczba ta oznacza bazę początkową . Pobiera więc bazę końcową oraz liczbę którą należy przetransferować. Wywołuje funkcje *change* w której modyfikuje tablicę *out*. Zapisuje ją do pliku wyjściowego pod opisem operacji.

Wraca do początku pętli.

Każdą liczbę formatuje wykreślając niepotrzebne zera z początku liczby wejściowej za pomocą funkcji format.

Jeśli któraś z funkcji głównych zwróci -1 to zamiast wyniku do pliku wyjściowego wpisany zostanie komunikat o nieprawidłowych danych.

Na koniec funkcja *main*() zamyka pliki wejściowy i wyjściowy i zwraca 0.

Kod źródłowy znajduje się w folderze src, w pliku mw_count.c

Pliki wykonywalne znajdują się w folderze exe i mają nazwy:

mw_count.o – plik wykonywalny dla systemów Unix

mw_count.exe – plik wykonywalny dla systemów Windows.

3. Podsumowanie

Najbardziej wymagającym problemem w powyższym projekcie było znalezienie sposobu na szybkie dzielenie oraz znajdowanie modulo. Dzięki sprytnemu podejściu do dzielenia stworzyłem algorytm działający zadowalająco szybko. Do obliczania modulo zaimplementowałem wyszukiwanie binarne, które wydaje się najszybszym sposobem wykonania tej operacji, biorąc pod uwagę brak operacji odejmowania w programie

Podczas tworzenia projektu nauczyłem się operowania tablicami, używania wskaźników, operowania pamięcią, modyfikowania plików tekstowych, przyjmowania argumentu przy wywoływaniu programu, składni języka C oraz przećwiczyłem myślenie algorytmiczne i umiejętność rozwiązywania problemów.

Program może być wykorzystywany do wykonywania obliczeń w różnych systemach liczbowych jak również do zamiany takich systemów dla poszczególnych liczb. Spełnia więc wszystkie zamierzone funkcje. Jest również zabezpieczony na wypadek błędnych danych i nielegalnych operacji.