Dokumentacja – program obliczeniowy

Adam Wylot

Listopad 2024

Spis treści

1 Ogólny opis programu

1.1 Specyfikacja

- System operacyjny: **Windows**
- Język programowania: **C**
- Standard języka: **C89/C90 (ANSI C)**
- Zgodny kompilator: **gcc (MinGW)**

1.2 Przeznaczenie

Program rozwiązuje zgodne działania arytmetyczne i konwersje systemów liczbowych zawarte w pliku wejściowym. Nagłówki wszystkich operacji, liczby oraz ich wyniki są zapisywane do pliku wyjściowego.
Liczby nie mają stałego ograniczenia co do swojej długości.

1.3 Dostępne działania

Lista dostępnych działań:

- **Dodawanie** *(+)*
- **Odejmowanie** *(-)*
- **Mnożenie** *(*)*
- **Dzielenie** *(/)*
- **Potęgowanie** *(^)*
- **Reszta z dzielenia** *(%)*
- **Konwersja systemu liczbowego**

1.4 Ograniczenia

Głównym ograniczeniem jest ustalony z góry format pliku wejściowego. Program stara się przeczytać nawet źle sformatowany plik wejściowy, ale nie zawsze może być to możliwe *(zapis może być dwuznaczny*).
Kolejnym ograniczeniem jest fakt, że program operuje wyłącznie na **nieujemnych liczbach całkowitych** *(dotyczy się to również wyników operacji np. przy odejmowaniu, więc odjemna > odjemnik)*. Liczby z pliku muszą mieścić się w pamięci operacyjnej komputera.
Posiada ograniczony zbiór operacji. Nie można również kontrolować kolejności wykonywania działań *(np. poprzez użycie nawiasów)*, czy mieszać ich ze sobą. W jednej operacji program przetwarza operację jednego typu, gdzie kolejność wykonywanych w niej działań idzie od góry do dołu.
Ostatnim ograniczeniem jest zakres obsługiwanych systemów liczbowych. Program obsługuje te o podstawach w zakresie **[2, 16]**.

2 Działanie

2.1 Kompilacja

2.1.1 Z poziomu IDE

W celu kompilacji programu należy skorzystać z kompatybilnego środowiska programistycznego *(IDE)*, otworzyć w nim projekt i nacisnąć przycisk odpowiedzialny za kompilacje.

2.1.2 Z poziomu środowiska tekstowego

Żeby uprościć proces do programu został dołączony plik **makefile**.
W celu kompilacji programu należy otworzyć środowisko tekstowe z zainstalowanym i aktywnym **MinGW**.
Następnie przejść do katalogu programu z plikiem makefile i wpisać polecenie ***make***.

2.2 Uruchamianie

Po kompilacji w katalogu **bin** zostanie utworzony plik **filecalc.exe**. Jest to plik wykonywalny programu.
Przez wymóg podania argumentów programu należy uruchamiać go z poziomu środowiska tekstowego poleceniem:

filecalc.exe [-h] ⟨wejście⟩ [wyjście]

Opis opcji:

- -h *(opcjonalny)* — wyświetla instrukcje użytkowania
- wejście — ścieżka do pliku wejściowego
- wyjście *(opcjonalny)* — ścieżka do pliku wyjściowego

Ściezki do plików mogą być zarówno względne o lokalizacje pliku wykonywalnego i absolutne.

2.3 Pliki wejściowe

Program jako wejście przyjmuje plik tekstowy. Plik składa się z wielu operacji szerzej opisanych w sekcji *4.2 Format operacji*. Operacje są oddzielone od siebie **trzema** *(lub większą ilością)* wierszami przerwy.
Ścieżka do pliku wejściowego musi być podana jako parametr wywołania programu.

Zdjęcie 1: Przykładowy plik wejściowy

2.4 Pliki wyjściowe

Plik wyjściowy wygląda identycznie jak plik wejściowy. Jedyną różnicą są wpisane wyniki obliczonych operacji w środkowy *(czyli drugi)* wiersz przerwy oddzielającej.
Jeżeli obliczenie wyniku było niemożliwe pole zostaje puste.
Jeżeli nie podano ścieżki do pliku wyjściowego to plik zostanie utworzony w tym samym miejscu co plik wejściowy. Będzie miał tę samą nazwę z przedrostkiem *"out_"*.
Jeżeli podano nazwę pliku wejściowego i nie ma ona rozszerzenia **.txt* to zostanie ono automatycznie dodane.

Zdjęcie 2: Przykładowy plik wyjściowy

3 Obsługa błędów

Obsługa błedów nie była w pełni sprecyzowana w wymaganiach dotyczących projektu, więc używa pewnych interpretacji i założeń. Jedynym wymaganiem było wykonanie jak największej ilości operacji.

Wyszedłem z założenia, że jeżeli nie uda się wczytać choć jednej liczby to wynik operacji nie jest liczony – cała operacja jest pomijana. Gdyby tylko liczba była pomijana, a wynik liczony z reszty liczb to i tak byłby inny niż oczekiwany. Co więcej mógłby spowodować zamieszanie z interpretacją pliku wynikowego czy na pewno był błąd podczas liczenia, czy jednak nie. Narzędzie służące do liczenia *(np. kalkulator)* nie może podawać błędnych lub częściowych wyników – musi być dokładne.
W związku z tym cała operacja musi być poprawna, żeby jej wynik został obliczony i wypisany do pliku wyjściowego.

Wszystkie informacje o błędach wraz z numerem wiersza pliku wejściowego, w którym został napotkany błąd, są wypisywane do **stderr** *(standardowym wyjściu dla błędów)*.

3.1 Błędy w pliku wejściowym

Jako, że plik wejściowy jest podawany przez użytkownika może zawierać błędy *(np. cyfra 8 w liczbie zapisanej w systemie liczbowym o podstawie 5)*. Program próbuje przeczytać plik dalej. Gdy znajdzie błąd to nie kończy swojego działania tylko szuka kolejnej operacji.

Program musi też poprawnie określić typ operacji oraz podstawę systemu liczbowego, który w nim obowiązuje. Gdy w wierszu nagłówkowym operacji znajdzie się błąd to nie będzie w stanie jej przetworzyć. Nie będzie wiedział co zrobić z liczbami.

Liczby muszą być oddzielone od siebie **jednym** *(maksymalnie dwoma)* wierszem przerwy, żeby nie zostały zinterpretowane jako operacje.
Analogicznie nowa operacja musi być poprzedzona **trzema** *(lub większą ilością)* wierszami przerwy. Oczywiście te założenia nie dotyczą pierwszej operacji występującej w pliku wejściowym – ta może znajdować się w pierwszym wierszu pliku.

Przykładowe błędy:

- w pliku wystąpił nieokreślony znak np. $
- cyfra, która nie należy do systemu liczbowego np. $56345\underline{A}34_{(10)}$
- system liczbowy nie jest obsugiwany np.: *** 22**

3.2 Błędy obliczeniowe

Podczas wykonywania działań mogą pojawiać się także niedozwolone operacje tj.:

$$0^0 \qquad \frac{x}{0} \qquad x - y < 0$$

Pierwsze dwie z nich są **symbolami nieoznaczonymi** w matematyce, więc żadne narzędzie liczące nie jest w stanie dokłądnie określić ich wyniku. Tak też jest w tym przypadku.
Ostatnia operacja jest niezgodna z ogarniczeniami programu (sekcja *1.4 Ograniczenia*), więc w pewnym sensie jest dla niego "symbolem nieoznaczonym". Ten obsługuje jedynie nieujemne liczby całkowite. Do tego zaliczają się również wyniki.
W takiej sytuacji również zostanie wypisany błąd i operacja będzie pominięta.

3.3 Kody błędów

Niestety niektórych błędów nie da się obsłużyć, np. *system operacyjny miał problem z zarezerwowaniem potrzebnej pamięci operacyjnej*. Taki błąd jest **błędem krytycznym**. Uniemożliwia prawidłowe, dalsze działanie programu. W takim wypadku skrypt musi się zakończyć.

Oto lista kodów wyjściowych i ich opis, które może zwrócić program po zakończeniu:

Kod	**Opis**
0	Sukces – program nie napotkał błędów krytycznych
1	Nieokreślony błąd
100	Błąd alokowania pamięci
110	Program nie otrzymał wymaganych argumentów
120	Podany plik wejściowy był pusty

4 Implementacja

4.1 Realizacja

Każde opsiane działanie nazywam operacją. Plik zawiera się więc z n operacji. W każdej operacji są liczby. Dokładniejszy opis operacji znajduje się w sekcji *4.2 Format operacji*.
Każda liczba jest zapisywana cyfra po cyfrze w pamięci zaczynając od tej najmniej znaczącej. Jest zapisywana w tablicy, gdzie każdy element ma **jeden bajt** *(dynamiczna tablica typu char)*. Na jednym bajcie zapisane są po dwie cyfry, ponieważ program obsługuje systemy liczbowe z zakresu $[2, 16]$. Z tego wynika, że do zapisu największej możliwej cyfry, czyli $F_{(16)} = 15_{(10)}$ potrzebne są *4 bity*.
Program czyta liczby z pliku wejściowego znak po znaku alokując pamięc dynamicznie. Gdy zabraknie na nie miejsca powiększa zarezerwowaną przestrzeń **dwukrotnie**.
Gdy napotka jakiś nieznany znak lub cyfrę nienależącą do danego systemu liczbowego to odpowiednio reaguje i obsługuje ten błąd (dokładniejszy opis w sekcji *3.1 Błędy w pliku wejściowym*).
Program czyta liczby i wynokuje działania dopóki nie trafi na kolejną operację.

4.2 Format operacji

Każda operacja rozpoczyna się od wiersza, który ją opisuje. Jest on rózny w zależności czy dana operacja odpowiada za działanie arytmetyczne, czy konwersje systemu liczbowego, co będzie opisane w podsekcjach.
Następnie pod nagłówkiem operacji znajdują się kolejno zapisane liczby.
W jednym wierszu może znajdywać się maksymalnie jedna liczba oraz każda liczba musi być oddzielona **jednym** *(co najwyżej dwoma)* pustym wierszem.

4.2.1 Operacja arytmetyczna

Każda operacja arytmetyczna jest opisana przez znak opisujący jakie działanie ma być wykonywane oraz podstawa systemu liczbowego z zakresu **[2, 16]** jaki obowiązuje w danej operacji. Są oddzielone od siebie pojedynczą **spacją**. Dostępne działania oraz symbole używane do ich opisania są zawarte w sekcji *1.3 Dostępne działania*.

Przykładowy wiersz nagłówkowy operacji:

*** 16**

Oznacza on, że jest to operacja **mnożenia w systemie heksadecymalnym**.

4.2.2 Konwersja systemu liczbowego

Każda operacja konwersji systemu liczbowego jest opisana przez przez dwie liczby z zakresu **[2, 16]** określające podstawy systemów liczbowych.
Pierwszy z nich określa to w jakim systemie są zapisane liczby należące do danej operacji. Drugi określa na jaki system ma się odbyć konwersja liczb.

Przykładowy wiersz nagłówkowy operacji:

5	**16**

Oznacza on, że jest to operacja konwersji liczb zapisanych w systemie liczbowym o **podstawie 5** do liczb zapisanych w postaci **heksadecymalnej**.

4.3 Struktury w programie

4.3.1 długa liczba (number_t)

Jest to najbardziej podstawowa struktura w programie. Są w niej zawarte wszystkie potrzebne informacje o **długiej liczbie**.

```
typedef struct {
    unsigned char* repr;
    size_t repr_size;
    size_t length;
    short base;

} * number_t;
```

Zdjęcie 3: Struktura number_t

Opis pól:

- repr – tablica, która przetrzymuje cyfry liczby
- repr_size – rozmiar zaalokowanej pamięci na tablicę repr
- length – długość liczby w ilości cyfr
- base – podstawa systemu liczbowego, w którym zapisana jest liczba

4.3.2 operacja (operation_t)

Zawiera potrzebne informacje o aktualnie przetwarzanej operacji.

```
typedef struct {
    number_t (*func)(number_t, number_t);
    oper_type_t type;
    short base;
    short conv_base;

} * operation_t;
```

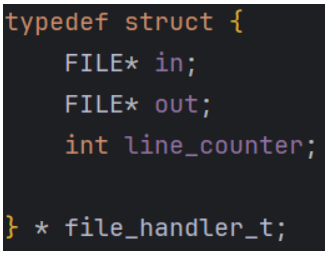
Zdjęcie 4: Struktura operation_t

Opis pól:

- (*func)() – wskaźnik do funkcji, która ma przetwarzać działanie obowiązujące w danej operacji
- type – typ operacji
- base – podstawa systemu liczbowego, w którym zapisane są liczby w operacji
- conv_base – jeżeli jest to operacja arytmetyczna to jego wartość wynosi zero; natomiast jeżeli jest to konwersja to zawiera podstawę docelowego systemu liczbowego

4.3.3 obsługa plików (file_handler_t)

Zawiera strumienie do pliku wejściowego i pliku wyjściowego. Obsługuje wszystkie operacje odczytu i zapisu. Zapewnia integralność przepływu danych przez program.



Zdjęcie 5: Struktura file_handler_t

Opis pól:

- in – strumień pliku wejściowego
- out – strumień pliku wyjściowego
- line_counter – licznik linii w pliku wejściowym

4.4 Użyte Algorytmy

Program nie używa żadnych skomplikowanych algorytmów. Wszystkie operacje arytmetyczne na liczbach oprócz potęgowania są napisane bez wzorca. **Dodawanie**, **odejmowanie**, **mnożenie** i **dzielenie** operają się na *"liczeniu pisemnym pod kreską"* jakie wykonywało się w szkole.
Rozważałem również użycia algorytmu Karacuby do wykonywania operacji mnożenia, ale jest on niewydajny dla małych liczb oraz jest to algorytm rekurencyjny. W tym programie założyłem, że większą wartością jest oszczędność pamięci operacyjnej niż szybkość działania.

Do potęgowania użyłem iteracyjnej wersji **algorytmu szybkiego potęgowania**, który korzysta z postaci binarnej wykładnika. Czyta wykładnik cyfra po cyfrze zaczynając od najmniej znaczącej. Wynik operacji początkowo jest ustawiany na wartość *1*.
Następnie czyta wszystkie cyfry wykładnika, zaczynając od najmniej znaczącej. Przy każdej iteracji **mnoży wynik przez samego siebi**e, a następnie sprawdza cyfrę wykładnika w postaci binarnej:

- **jeśli 1 – pomnóż wynik przez podstawę**

Po przejściu przez wszystkie cyfry otrzymamy wynik potęgowania.

Do liczenia reszty z dzielenia *modulo* użyłem klasycznego liczenia.
Liczbę dzielę całkowicie przez dzielnik, a następnie wynik tej operacji przez dzielnik mnożę. W ten sposób otrzymam najbliższą, mniejszą niż dzielna wielokrotność dzielnika.
Żeby otrzymać resztę z dzielenia należy odjąć od dzielnej otrzymany wynik.

Do konwersji systemu liczbowego używam: *konwersji z dowolnego systemu na dziesiętny* oraz *z dziesiętnego na dowolny*. System dziesiętny jest więc systemem przejściowym każdej operacji.
Do tych operacji używam najbardziej popularnych algorytmów tj. mnożenie cyfr przeż potęge podstawy systemu oraz zapisywanie reszty z dzielenia w odwrotnej kolejności. Są to bardzo znane i powszechnie używane algorytmy, więc pominę opis ich działania.

5 Podsumowanie

Zadanie nie sprawiło mi większych problemów. Oczywiście w między czasie pojawiły się jakieś bardzo drobne błędy, których szukałem niekrótko *(np. napisanie znaku + zamiast – przy liczbie)*, ale towarzyszą one przy każdym trochę większym programie.
Najbardziej skomplikowanym procesem było napisanie algorytmu dzielenia, ponieważ ten jest tym najbardziej skomplikowanym jeżeli chodzi o *liczenie pisemne "pod kreską"*. Na szczęście nie było to najcięższe zadanie z jakim miałem do czynienia.