Zadanie 1 (10 pkt. na pracowni, później 5 pkt.). Pobierz ze SKOS swoje rozwiązanie zad. 2 z listy nr 3. Wprowadź w nim poprawki, które zostały wskazane przy jego sprawdzaniu (o ile je pamiętasz). Ten program zapisz pod nazwą old.c.

Skopiuj ten plik i zmodyfikuj go (o ile to potrzebne – być może jest on już właśnie w takim kształcie) tak, by funkcja main zajmowała się wyłącznie sprawdzaniem poprawności argumentów wywołania, a potem kolejno wywoływała dwie funkcje, obliczające i drukujące wynik odpowiednich części zadania. Następnie podziel kod na program właściwy (nazwij go new.c), plik nagłówkowy (module.h) i jego implementację (module_old.c). Zrób to "tak, jak trzeba", w szczególności dołącz (#include) plik nagłówkowy w obu pozostałych plikach.

Napisz nową implementację modułu (nazwij ją module_new.c) tak, by każda z dwóch ww. funkcji zadeklarowanych w nagłówku:

- przydzielała na stercie odpowiednie bufory typu char * (a także tablicę wskaźników na nie), i przepisywała do nich zawartość argv (poza pozycjami 0 i 1);
- modyfikowała zawartość ww. buforów na potrzeby statystyk, jakie ma obliczyć;
- obliczała i drukowała te statystyki;
- zwalniała przydzielone bufory.

Oczywiście możesz definiować kolejne funkcje pomocnicze, które nie muszą być deklarowane w nagłówku, a mogą być wykorzystywane w obu ww. funkcjach. Pamiętaj o sprawdzaniu poprawności przydziału pamięci – jeśli się on nie powiedzie, np. zwolnij dotychczas przydzielone w danej funkcji bufory i zakończ jej działanie, oczywiście nie generując statystyk.

Jeśli Twoja "stara" implementacja już odpowiada powyższej idei (albo np. ma bardzo dużo wad, które trudno szybko naprawić), jako "nową" implementację napisz jej nieznacznie zmodyfikowaną wersję, w której poszczególne słowa z argv nie będą kopiowane do osobnych buforów, tylko do jednego długiego bufora (oczywiście pooddzielane bajtami zerowymi) – wtedy też należy zapamiętać wskaźniki na pozycje początków kolejnych słów w tym buforze. Ta implementacja powinna wywoływać malloc/calloc dokładnie dwa razy (czy też najwyżej dwa razy, uwzględniając ew. niepowodzenie pierwszego z nich) w każdej z dwóch funkcji deklarowanych w nagłówku.

Jako rozwiązanie zadania prześlij 5 plików o podanych wyżej nazwach. Sekwencja poleceń

```
gcc -Wall -Wextra -Werror -xc -std=c11 module_X.c -c
gcc -Wall -Wextra -Werror -xc -std=c11 new.c -c
gcc module_X.o new.o -lm
```

powinna wykonywać się poprawnie niezależnie od tego, czy X zastąpimy przez new czy przez old, i w obu przypadkach generować program o zasadniczo takim samym działaniu. (Plik old.c też powinien kompilować się poprawnie.)

Zadanie 2 (10 pkt.). W tym zadaniu **nie obowiązuje** flaga kompilacji -Werror – dalej będziemy widzieć ostrzeżenia kompilatora i powinnyśmy je w miarę możliwości wyeliminować, ale dla niekompletnych rozwiązań akceptowalne będą te o nieużywanych parametrach funkcji.

W tym zadaniu będziemy reprezentować dodatnie liczby całkowite względem ustalonego zbioru liczb pierwszych – niech będą to 2, 3, 5, 7, 11, 13, 17, 19, czyli kolejno *p*_0, *p*_1, ..., *p*_7. Reprezentacją liczby *p*_0^*a*_0×*p*_1^*a*_1×...×*p*_7^*a*_7 jest oczywiście (ośmio)krotka *a*_0, *a*_1, ..., *a*_7, a inne liczby (w tym 0) nie mają swojej reprezentacji.

Zadeklaruj typ struct, który będzie zawierał tablicę 8 liczb typu unsigned char – wykładników z takiej reprezentacji. Jako samą liczbę będziemy traktować wskaźnik na takiego struct-a (oczywiście zazwyczaj przydzielonego na stercie), który będziemy nazywać penumber (od *prime exponent*).

Zaimplementuj następujące funkcje:

- penumber new_penumber(unsigned long long n) tworzącą reprezentację n,

- void del_penumber(penumber n) zwalniającą pamięć zajętą przez n,
- void print_penumber(penumber n) drukującą reprezentację n w postaci np. 2^5x11, w szczególności z pominięciem wykładników zerowych (jeśli wszystkie są zerowe, to należy wydrukować pojedynczą cyfrę 1) i bez złamania wiersza na końcu,
- unsigned long long to_ull(penumber n) obliczającą wartość reprezentowaną w n,
- int divides(penumber n, penumber m) sprawdzającą, czy n dzieli m,
- penumber pnprod(penumber n, penumber m), penumber pnlcm(penumber n, penumber m), penumber pngcd(penumber n, penumber m), penumber pndiv(penumber n, penumber m) tworzące penumber będący odpowiednio iloczynem, najmniejszą wspólną wielokrotnością, największym wspólnym dzielnikiem i ilorazem (n przez m) argumentów.

Jeśli wywołanie funkcji nie powiodło się (przez błędne argumenty albo nieudany przydział pamięci), powinna ona zwracać NULL (jeśli jej typ wynikowy to penumber) lub 0 (to_ull); dla print_penumber i divides użytkownik musi sam pamiętać o przekazaniu poprawnych argumentów. Dla to_ull sprawdzaj (przy użyciu stałych z limits.h), czy wynik obliczeń mieści się w zadanym typie.

Deklaracje umieść w pliku nagłówkowym primrepr.h, implementację – w module primrepr.c (oba te pliki stanowią rozwiązanie zadania). Pobierz plik main.c dostępny w SKOSie tam, gdzie będziesz przesyłać rozwiązanie tego zadania. Polecenie gcc -Wall -Wextra -xc -std=c11 main.c primrepr.c -lm powinno wykonywać się poprawnie, przy czym funkcje, dla których masz tylko trywialną, tymczasową implementację, oczywiście mogą powodować ww. ostrzeżenia.

Zadanie 3 (10 pkt.).

Na potrzeby zadania wyobraźmy sobie, że jesteśmy w urzędzie. Przed naszymi oczami znajduje się k stanowisk obsługi petenta numerowanych od 1 do k. Przed i-tym stanowiskiem znajduje się m_i krzeseł - poczekalnia.

Mogą się zdarzyć następujące sytuacje:

- do urzędu może przyjść petent (o podanym imieniu) ze sprawą obsługiwaną w okienku o numerze i - taka osoba staje na końcu kolejki do stanowiska i
- stanowisko o numerze i może zaprosić kolejnego petenta - a pierwsza osoba z kolejki do tego stanowiska podchodzi z gwarancją obsłużenia jej sprawy,
- urząd może zakończyć dzień pracy - wszyscy idą do domu i nikt więcej nie jest już obsługiwany.

Możesz założyć, że stanowisko nie zaprosi kolejnego petenta, gdy nie ma nikogo w kolejce. Gdy do urzędu wchodzi petent, a jego poczekalnia jest pełna, poddaje się i wychodzi.

Twoim zadaniem jest wypisać imiona obsłużonych petentów w kolejności w jakiej podchodzili do stanowisk.

W tym zadaniu kluczowe są limity:

- stanowisk jest nie więcej niż 2^{16}
- długość imienia pojedynczej osoby to nie więcej niż 2^{18} (imię tworzą małe litery alfabetu angielskiego)
- maksymalna długość pojedynczej kolejki to 2^{19}
- sumaryczna liczba miejsc w kolejkach jest nie większa niż łączna liczba liter w imionach osób znajdujących się w kolejkach w dowolnym momencie. Obie wartości są mniejsze od 2^{20}

Uwaga Limit pamięci wynosi 14 MB.

Wskazówka Długości imion petentów nie są znane. Warto jest mieć jeden odpowiednio duży bufor służący tylko do wczytywania imion i alokować pamięć na imiona po poznaniu ich długości.

Wejście:

W pierwszej linii wejścia znajduje się liczba k. W drugiej k liczb: m_1, m_2, ... m_k. Następnie następują opisy zdarzeń w następującej formie:

- N imie nr_stanowiska (nowy petent)
- Z nr_stanowiska (stanowisko zaprasza kolejnego petenta)
- K (koniec dnia pracy urzędu, pojawia się raz, na końcu wejścia)

Wyjście:

Wyjście powinno składać się z tylu wierszy, ilu petentów zostało obsłużonych. W każdym wierszu powinno znaleźć się imię obsłużonego petenta.

Przykłady:

Przykład A

Wejście

```
1
2
N ala 1
N bartek 1
```

N cyryl 1
Z 1
Z 1
N damian 1
Z 1
K

Wyjście

ala
bartek
damian

Przykład B

Wejście

2
1 1
K

Wyjście

Przykład C

Wejście

2
4 6
N ala 1
N bartek 2
N cyprian 2
Z 2
Z 1
N damian 2
Z 2
K

Wyjście

bartek
ala
cyprian

Wyjaśnienie: pierwsze zaprasza stanowisko drugie, stąd pierwszą osobą na wyjściu jest *bartek*, pierwszy w kolejce do stanowiska drugiego.