Zadanie 1 (10 pkt. na pracowni, później 5 pkt.). Dziś Mikołajki, więc zadaniem będzie implementacja nieskomplikowanego typu złożonego do przechowywania danych o beneficjentach prezentów mikołajkowych. W tym celu stwórz 2 pliki: `beneficjent.h` oraz `beneficjent.c`. W pliku `beneficjent.h` umieść definicję rekordu `Osoba` o następujących polach:

- `imie` typu `char*`
- `wiek` typu `uint8_t` (typ całkowitoliczbowy o dokładnie 8 bitach, bez znaku, zadeklarowany w `stdint.h`)
- `zaslugi` typu `unsigned int`
- `szelmostwa` typu `unsigned int`

oraz deklaracje funkcji:

- `struct Osoba* nowa_osoba(char* imie, uint8_t wiek)` – przydziela pamięć na nowy obiekt (użyj funkcji `malloc` bądź `calloc`) i zwraca do niej wskaźnik (a jeśli się to nie uda, to `NULL`)
- `void usun_osobe(struct Osoba* osoba)` – zwalnia pamięć przydzieloną na dany obiekt (użyj funkcji `free`, nie zapomnij zwolnić też pamięci zajmowanej przez pole `imie`)
- `void zmien_imie(struct Osoba* osoba, char* imie)` – zmienia wartość pola `imie` (warto użyć funkcji `realloc` oraz `strcpy`)
- `void zmien_wiek(struct Osoba* osoba, uint8_t wiek)` – zmienia wartość pola `wiek`
- `void dodaj_zasluge(struct Osoba* osoba)` – dodaje 1 do wartości pola `zaslugi`
- `void dodaj_szelmostwo(struct Osoba* osoba)` – dodaje 1 do wartości pola `szelmostwa`
- `void ustaw_statystyki(struct Osoba* osoba, unsigned int zaslugi, unsigned int szelmostwa)` – ustawia wartości pól `zaslugi` oraz `szelmostwa`
- `int czy_dostanie_prezent(struct Osoba* osoba)` – zwraca 1, gdy dana osoba powinna zostać beneficjentem (czyli jeśli ma więcej zasług niż szelmostw lub jest poniżej czwartego roku życia), 0 w przeciwnym przypadku
- `void wypisz_info(struct Osoba* osoba)` – wypisuje na standardowe wyjście w czytelnym formacie wszystkie informacje o danej osobie
- `int komparator(const struct Osoba* os1, const struct Osoba* os2)` – w zależności od tego, czy `os1` czy `os2` jest "wcześniej w kolejce" do prezentu, zwraca odpowiednio -1 i 1 (lub 0, jeśli jest "remis"); jako kryterium przyjmij wartość różnicy `zaslugi-szelmostwa`, a w drugiej kolejności – wiek (słowa kluczowe `const` można pominąć, zwłaszcza gdyby z jakiegoś powodu skutkowały błędami kompilacji, ale mogą okazać się konieczne, jeśli w jeśli w zad. 2 będziemy chcieć skorzystać z bibliotecznego `qsort`)

Definicje powyższych funkcji umieść w pliku `beneficjent.c`. Zadbaj o to, aby program kontrolował poprawność wywołań (i np. nie ustawiał pól w pamięci, której nie udało się poprawnie przydzielić).

W funkcji `main` programu (zaimplementowanej w osobnym pliku `prog.c`) zaimplementuj w miarę kompletny (i niekoniecznie interaktywny) test ww. funkcji.

Uwaga: przypisywanie imienia nie może wyglądać tak: `osoba->imie = imie;` – wtedy przypisujemy tylko wskaźnik (który może np. prowadzić do sekcji `.rodata`, a nie na stertę). Należy zaalokować dodatkową pamięć i skopiować imię do nowego bufora.

Obserwacja: to nieprzypadkowo wygląda jak programowanie klasy i jej metod w paradygmacie obiektowym.

Zadanie 2 (10 pkt.).

Skorzystaj modułu zaimplementowanego w zadaniu 1 i napisz program, który pozwala na interaktywne manipulowanie listą osób.

Do rekordu `Osoba` dodaj pole `indeks` typu `unsigned int`. To pole ustawiaj niezależnie od żądań użytkownika, zadbaj o to, aby każda osoba miała unikalny indeks; możesz użyć zmiennej nielokalnej, ale postaraj się, żeby była ona możliwie "mało nielokalna". Nie powinno być więcej różnic między kodami modułów w obu zadaniach.

Napisz program, który w nieskończonej pętli wczytuje ze standardowego wejścia polecenia z następującej listy i je realizuje:

- `0` – wyświetl krótką instrukcję obsługi
- `1 N` – wczytaj dane o `N` osobach, które muszą wtedy być podane w `N` kolejnych liniach, zawierających na początku wiek, zasługi i szelmostwa (liczby nieujemne), a potem imię
- `2` – posortuj dane używając funkcji `komparator`; możesz użyć `qsort` ze `stdlib.h` albo napisać któreś z prostych sortowań znanych z WDI
- `3` – wypisz dane
- `4 i w` – zmień wiek osoby o indeksie `i` na wartość `w`
- `5 i n` – zmień imię osoby o indeksie `i` na wartość `n`
- `6 i` – dodaj zasługę osobie o indeksie `i`
- `7 i` – dodaj szelmostwo osobie o indeksie `i`
- `8 i z s` – ustaw statystyki osobie o indeksie `i` na `z` (`zaslugi`) i `s` (`szelmostwa`)
- `9 i` – wypisz informacje o osobie o indeksie `i`

Do rekordów z danymi o osobach należy odwoływać się wyłącznie przy użyciu funkcji z `beneficjent.h` (znowu może się to kojarzyć z programowaniem obiektowym).

Wczytane dane zapamiętaj w tablicy – może to być wręcz tablica wskaźników na `struct Osoba`, ale powinno też się udać z tablicą bezpośrednio zawierającą `struct`-y, jeśli tak będzie Ci łatwiej.

Zarówno powyższe polecenia, jak i informacje o kolejnych osobach w poleceniu `1` wczytuj z osobnych wierszy – użyj do tego funkcji `fgets`, załóż jakąś maksymalną długość wiersza i zdefiniuj ją makrem `#define` w pliku programu (nie module – ma ona dotyczyć tylko wczytywanych danych, nie wpływać na implementację "ogólnego przeznaczenia" operacji na typie `struct Osoba`). Wiersze możesz parsować ręcznie albo np. przy użyciu `strtoul` czy `sscanf`, ale (zwłaszcza w tym ostatnim przypadku) pamiętaj, że imię może zawierać spacje.

W programie zadbaj o odpowiednie komunikaty, np. w przypadku podania niepoprawnego polecenia, które oczywiście nie powinno być wykonywane. Pliki źródłowe nazwij jak w zad. 1.

Zadanie 3 - Napisz program który zsumuje *n* 50-cyfrowych liczb, obliczy i wypisze pierwsze dziewięć cyfr z ich sumy.

Wejście

Na standardowym wejściu znajdziesz liczbę *n* (n<=100), a następnie *n* 50 cyfrowych liczb.

n

i_1

i_2

i_3

...

i_n

Wyjście

Na standardowym wyjściu należy wypisać pierwszych 9 cyfr, z sumy *n* elementów.

Przykłady

Przykład A

Wejście

```
1
37107287533902102798797998220837590246510135740250
```

Wyjście

```
371072875
```

Przykład B

Wejście

```
2
3710728753390210279879799822083759024651013574025046376937677490009712648124896970078050417018260538
```

Wyjście

```
834842252
```

Przykład C

Wejście

```
5
3710728753390210279879799822083759024651013574025046376937677490009712648124896970078050417018260538743249861995247410594742333095130581237266173096299194221336357416157252243056330181107240615490825023067588207539346171171980310421047513778063246676
```

Wyjście

```
272819012
```

Uwagi

Te liczby nie zmieszczą się w żadnym standardowym typie, zastosuj własny typ i zaimplementuj na nim operacje dodawania.