Widoczne grupy JP 1

Modyfikacje w treści po 14.03 zostały zaznaczone na niebiesko.

Celem zadania jest implementacja struktury danych przechowującej informacje o pacjentach, ich historie chorób oraz opisy chorób.

Operacje na strukturze

Na początku działania programu struktura danych jest pusta. Informacje o modyfikacjach struktury oraz żądania dostępu do opisu chorób są wczytywane ze standardowego wejścia w osobnych liniach. Zakładamy, że nazwiska jednoznacznie identyfikują pacjentów. Choroby numerujemy w kolejności wprowadzania, poczynając od 1.

- *NEW_DISEASE_ENTER_DESCRIPTION name disease description*
  Dodaje informację o chorobie pacjenta o nazwisku *name*. Dalsza część linii stanowi opis przebytej choroby, w tym przypadku jest to *"disease description"*.
- *NEW_DISEASE_COPY_DESCRIPTION name1 name2*
  Dodaje informację o chorobie pacjenta o nazwisku *name1*. Opis nowej choroby jest taki samy jak aktualny opis ostatnio zarejestrowanej choroby pacjenta o nazwisku *name2*.
- *CHANGE_DESCRIPTION name n disease description*
  Aktualizuje opis n-tej choroby pacjenta o nazwisku *name*. Dalsza część linii stanowi nowy opis choroby, w tym przypadku "*disease description"*.
- *PRINT_DESCRIPTION name n*
  Wypisuje na standardowe wyjście opis n-tej choroby pacjenta o nazwisku *name*.
- *DELETE_PATIENT_DATA name*
  Usuwa historię chorób pacjenta o nazwisku *name.* Pacjent z pustą listą chorób pozostaje w strukturze.

Po udanym wykonaniu komendy innej niż *PRINT_DESCRIPTION* należy wypisać linię *OK* na standardowe wyjście. Jeśli podczas wykonywania którejkolwiek operacji próbowalibyśmy odwołać się do danych pacjenta bądź choroby, których nie ma w strukturze, to taką operację należy zignorować. Ponadto w takiej sytuacji należy wypisać linię *IGNORED*.

Licznik referencji

W zadaniu często trzeba będzie przechowywać informacje o różnych przypadkach chorób posiadających w systemie ten sam opis. Kopiowanie napisu i trzymanie go w wielu miejscach pamięci jest wysoce nieefektywnym rozwiązaniem. Zamiast tego lepiej trzymać opis choroby w jednym miejscu i dla każdego przypadku tej choroby pamiętać wskaźnik do opisu.

Problem pojawia się, kiedy trzeba zwolnić pamięć zajmowaną przez opis. Aby wykryć moment, w którym usuwamy ostatni wskaźnik pamiętający położenie napisu w pamięci, można zliczać takie wskaźniki. Najprościej wzbogacić strukturę danych o licznik, który zwiększamy za każdym razem, gdy jej adres jest przekazywany do nowego wskaźnika, oraz zmniejszamy, gdy jeden z takich wskaźników jest usuwany. Kiedy licznik wskaże zero, możemy bezpiecznie zwolnić pamięć.

Szczegóły implementacji

Informacje o zarejestrowanych pacjentach powinny być przechowywane przy użyciu listy. Ponadto historia chorób każdego pacjenta powinna być trzymana na osobnej liście. Polecenie *NEW_DISEASE_COPY_DESCRIPTION* nie powinno alokować pamięci na nowy opis choroby - zamiast tego należy pamiętać odnośnik do już utworzonego opisu. Z drugiej strony polecenia *NEW_DISEASE_ENTER_DESCRIPTION* oraz *CHANGE_DESCRIPTION* powinny każdorazowo zaalokować nową pamięć bez względu na to, jakie opisy znajdują się już w strukturze. Wymagamy, aby pamięć zawierająca opis choroby była zwalniana w momencie, w którym historia chorób żadnego pacjenta już się do niej nie odwołuje. W tym celu należy użyć liczników referencji.

Jeśli dodatkowo program zostanie wywołany z parametrem `-v`, należy dla każdego z poleceń wypisać na standardowe wyjście diagnostyczne (`stderr`) następujący wiersz:

```
DESCRIPTIONS: n
```

gdzie n to liczba opisów chorób przechowywanych w pamięci. Powinna ona zgadzać się z powyższym algorytmem zarządzania pamięcią. Nie dopuszczamy innych parametrów wywołania - w przypadku napotkania innego parametru, program powinien wypisać komunikat *ERROR* i zakończyć działanie z kodem wyjścia 1. Przykład użycia parametrów wywołania programu znajduje sie w zasobach na moodle w pliku `params.c`.

Skrypt testujący

Osobną częścią zadania jest napisanie skryptu `test.sh`. Po wywołaniu `./test.sh prog directory`, skrypt powinien uruchomić program `prog` dla wszystkich plików wejściowych postaci `directory/*.in`, porównać wyniki z odpowiadającymi im plikami `.out`, a następnie wypisać, które testy zakończyły się niepowodzeniem. Wywołanie `./test.sh -v prog directory`, powinno sprawdzić także poprawność programu uruchomionego z opcją `-v`. Wyniki z wyjścia diagnostycznego należy porównać z plikami `.err`.

Ograniczenia

Możesz przyjąć następujące założenia o danych wejściowych:

- Długość żadnego wiersza pliku wejściowego nie przekroczy 10^5.
- Rozmiar pliku wejściowego nie przekroczy 5 MB.
- Każdy wiersz pliku wejściowego kończy się linuksowym znakiem końca linii (kod ASCII 10). Poza nimi w pliku wejściowym występują tylko drukowalne znaki kodu ASCII (patrz isprint()). Żadne dwa białe znaki nie sąsiadują ze sobą.
- Numery chorób będą zawsze liczbami całkowitymi dodatnimi.

Twój program będzie miał do dyspozycji 64 MB pamięci. Przed zakończeniem należy zwolnić całą zaalokowaną pamięć. Złożoność pamięciowa rozwiązania powinna być proporcjonalna do rozmiaru pliku wejściowego.

Wymagamy, aby złożoność obliczeniowa operacji *NEW_DISEASE_ENTER_DESCRIPTION* oraz *NEW_DISEASE_COPY_DESCRIPTION* była proporcjonalna do liczby pacjentów zarejestrowanych w strukturze. W przypadku pozostałych operacji - proporcjonalna do sumy liczby pacjentów oraz długości historii chorób interesującego nas pacjenta.

Podział na pliki

Twoje rozwiązanie powinno zawierać następujące pliki:

- `structure.h`
  Plik nagłówkowy biblioteki wykonującej operacje na strukturze danych.
- `structure.c`
  Implementacja biblioteki wykonującej operacje na strukturze danych.
- `parse.h`
  Plik nagłówkowy biblioteki wczytującej i parsującej polecenia.
- `parse.c`
  Implementacja biblioteki wczytującej i parsującej polecenia.
- `hospital.c`
  Główny plik programu, w którym wczytujemy wejście i wywołujemy funkcje z pliku `structure.h`. Plik ten nie powinien znać typów danych, użytych do implementacji struktury danych.
- `test.sh`
  Patrz sekcja "skrypt testujący".
- `Makefile`
  W wyniku wywołania polecenia `make` powinien zostać wytworzony program wykonywalny `hospital`.

Dodatkowo w wyniku wywołania polecenia `make debug` powinien zostać wytworzony plik `hospital.dbg`, który powinien zawierać symbole do debugowania (opcja `-g` kompilacji) tak, aby ułatwiało to śledzenie wycieków pamięci za pomocą programu `valgrind`. Jeśli któryś z plików źródłowych ulegnie zmianie, ponowne wpisanie `make` lub `make debug` powinno na nowo stworzyć odpowiedni plik wykonywalny.

Zachęcamy, by `Makefile` działał w następujący sposób:

1. osobno kompilował każdy plik `.c` i osobno linkował,
2. przy zmianie w pliku źródłowym powinny być wykonane tylko te polecenia kompilacji, które są potrzebne,
3. `make clean` powoduje usunięcie wszystkich plików wykonywalnych i dodatkowych plików kompilacji.

Jednak w tym zadaniu nie będziemy stosować kar punktowych za brak spełnienia tych trzech warunków.

Punktacja

Za w pełni poprawne rozwiązanie zadania implementujące wszystkie funkcjonalności można zdobyć maksymalnie 20 punktów. Możliwe są punkty karne za poniższe uchybienia:

- Za wycieki pamięci można stracić co najwyżej 6 punktów.
- Brak obsługi parametrów wywołania lub błędne wyniki na wyjściu diagnostycznym grożą utratą 4 punktów.
- Za niezgodną ze specyfikacją strukturę plików w rozwiązaniu można stracić co najwyżej 4 punkty.
- Za niepoprawną obsługę poleceń, które należało zignorować, można stracić co najwyżej 3 punkty.
- Za błędy stylu kodowania można stracić co najwyżej 3 punkty.
- Za błędy w skrypcie testującym można stracić maksymalnie 2 punkty.
- Programy o złożoności pamięciowej gorszej od oczekiwanej są narażone na utratę do 8 punktów.
- Programy o złożoności obliczeniowej gorszej od oczekiwanej są narażone na utratę do 4 punktów

Rozwiązania należy implementować **samodzielnie** pod rygorem niezaliczenia przedmiotu.

Przykład

Dla danych wejściowych:

```
1. NEW_DISEASE_ENTER_DESCRIPTION Kowalski Bardzo ciężka choroba
2. NEW_DISEASE_ENTER_DESCRIPTION Nowak delikatne przeziębienie...
3. NEW_DISEASE_COPY_DESCRIPTION Kowalski nowak
4. NEW_DISEASE_COPY_DESCRIPTION Kowalski Nowak
5. PRINT_DESCRIPTION Kowalski 3
6. PRINT_DESCRIPTION Kowalski 2
7. CHANGE_DESCRIPTION Kowalski 2 przeziębienie z powikłaniami !
8. NEW_DISEASE_COPY_DESCRIPTION van_Beethoven Kowalski
9. NEW_DISEASE_COPY_DESCRIPTION van_Beethoven Nowak
```

10. DELETE_PATIENT_DATA Kowalski
11. NEW_DISEASE_COPY_DESCRIPTION Nowak Kowalski
12. PRINT_DESCRIPTION van_Beethoven 1

poprawnym wynikiem jest:	zaś poprawny wynik na wyjście diagnostyczne to:
1. OK	1. DESCRIPTIONS: 1
2. OK	2. DESCRIPTIONS: 2
3. IGNORED	3. DESCRIPTIONS: 2
4. OK	4. DESCRIPTIONS: 2
5. IGNORED	5. DESCRIPTIONS: 2
6. delikatne przeziębienie...	6. DESCRIPTIONS: 2
7. OK	7. DESCRIPTIONS: 3
8. OK	8. DESCRIPTIONS: 3
9. OK	9. DESCRIPTIONS: 3
10. OK	10. DESCRIPTIONS: 2
11. IGNORED	11. DESCRIPTIONS: 2
12. przeziębienie z powikłaniami !	12. DESCRIPTIONS: 2

Dostępne od:	Monday, 14 March 2016, 13:15
Termin oddania:	Sunday, 3 April 2016, 23:55

Prześlij plik