

Zadanie 4: Kolejki

Opened: Tuesday, 28 November 2023, 10:05 AM

Due: Wednesday, 20 December 2023, 4:05 PM

Uwaga: Od tego zadania programy należy pisać w języku C++ (standard C++17). W praktyce, do rozwiązania tego zadania wystarczy zbiór instrukcji z języka C i `vector` z C++ (ale można używać więcej).

Twoim zadaniem jest efektywnie zasymulować zdarzenia występujące w dniu pracy urzędu. W urzędzie jest m okienek, ponumerowanych od 0 do $m-1$. W kolejce do każdego okienka ustawiają się interesanci. Gdy urzędnik jest wolny, obsługuje pierwszego interesanta z kolejki, po czym ten opuszcza urząd i już tego dnia nie wraca. Poza tym w urzędzie zdarzają się różne inne sytuacje, opisane przez poniższe funkcje. Twoim zadaniem jest zaimplementować te funkcje.

`void otwarcie_urzedu(int m)` – Ta funkcja zostanie wywołana tylko raz, jako pierwsza. Wywołanie tej funkcji informuje Twoją bibliotekę, że w urzędzie jest m okienek (oraz jedno okienko specjalne – o nim później). W urzędzie nie ma jeszcze żadnego interesanta.

`interesant *nowy_interesant(int k)` – Do urzędu wchodzi nowy interesant i od razu ustawia się na końcu kolejki przy okienku o numerze k . Funkcja powinna stworzyć nowy element typu `interesant` i dać w wyniku wskaźnik na ten element.

`int numerek(interessant* i)` – zwraca numerok interesanta i . Każdy interesant po przyjsciu do urzędu od razu dostaje numerok. Numerki zaczynają się od 0 i są kolejnymi liczbami całkowitymi.

`interesant *obsluz(int k)` – Urzędnik obsługujący okienko o numerze k próbuje obsłużyć kolejnego interesanta. Jeśli kolejka do okienka jest niepusta, pierwszy interesant w kolejce zostaje obsłużony i wychodzi z urzędu; wówczas wynikiem funkcji powinien być wskaźnik na tego interesanta. W przeciwnym przypadku nikt nie zostaje obsłużony, a wynikiem funkcji powinno być `NULL`.

`void zmiana_okienka(interessant *i, int k)` – Interesant i orientuje się, że stoi w kolejce do niewłaściwego okienka. Opuszcza swoją kolejkę i ustawia się na końcu kolejki do okienka k , którą obecnie uważa za właściwą. Możesz założyć, że tuż przed wywołaniem tej funkcji interesant i był ustawiony w jakiejś kolejce i nie była to kolejka do okienka o numerze k .

`void zamkniecie_okienka(int k1, int k2)` – Urzędnik obsługujący okienko numer $k1$ udaje się na przerwę. Wszyscy interesanci stojący w kolejce do tego okienka zostają skierowani do okienka numer $k2$ i ustawiają się na końcu kolejki przy okienku $k2$ w tej samej kolejności, w jakiej stali w kolejce do okienka $k1$. Nie wiemy, jak długo potrwa przerwa urzędnika; może się zdarzyć, że tego samego dnia w kolejce do okienka $k1$ ustawią się interesanci i że zostaną oni obsłużeni.

`std::vector<interesant*> fast_track(interessant *i1, interessant *i2)` – Urzędnik otwiera na chwilę okienko specjalne, w którym można szybko obsłużyć wszystkie sprawy. Pewna grupa interesantów stojących dotychczas kolejno w jednej kolejce orientuje się, co się dzieje, i natychmiast przechodzi do okienka specjalnego, gdzie zostaje od razu obsłużona w tej samej kolejności, w jakiej stali w swojej dotychczasowej kolejce, po czym opuszcza urząd, a okienko specjalne się zamyka. Grupa interesantów jest podana za pomocą pierwszego ($i1$) i ostatniego ($i2$) interesanta w grupie; jeśli $i1=i2$, to obsłużony zostaje tylko interesant $i1$. Wynikiem funkcji powinien być `vector` interesantów obsłużonych kolejno w okienku specjalnym, w kolejności obsłużenia.

`void naczelnik(int k)` – Raz po raz ze swojego gabinetu wygląda naczelnik urzędu i z nudów przestawia osoby ustawione w kolejce. Dokładniej, za każdym razem wybiera on jedno okienko numer k i nakazuje wszystkim interesantom ustawionym w kolejce do tego okienka odwrócić kolejność ich ustawienia w kolejce. Mimo że taka operacja jest w gruncie rzeczy niezrozumiała, interesanci wykonują polecenia naczelnika niezwłocznie z obawy, żeby nie wpadł na jeszcze dziwniejszy pomysł. Jeśli w kolejce do tego okienka znajdował się co najwyżej jeden interesant, operacja naczelnika nie odnosi żadnego skutku.

`std::vector<interesant *> zamkniecie_urzedu()` – Ta funkcja zostanie wywołana raz, na koniec interakcji z biblioteką. Oznacza koniec dnia pracy urzędu. Wszyscy pozostali interesanci zostają szybko obsłużeni i urząd na ten dzień się zamyka. Wynikiem funkcji powinien być `vector` wszystkich interesantów, którzy do tego momentu zostali w urzędzie, w kolejności: najpierw wszyscy stojący w kolejce do okienka 0 (w kolejności obsłużenia), potem wszyscy stojący w kolejce do okienka 1 (w kolejności obsłużenia) itd.

Twoja biblioteka nie zwalnia pamięci po żadnym interesancie. Jest to w odpowiedzialności użytkownika biblioteki. Użytkownik zwalnia pamięć dopiero wtedy, gdy interesant opuszcza urząd. **Uwaga:** Użytkownik będzie zwalniał pamięć po interesantach za pomocą funkcji `free`, więc Twoja biblioteka musi alokować pamięć za pomocą funkcji `malloc`.

Deklaracje podanych funkcji znajdują się w pliku [kol.h](#). Twoim zadaniem jest uzupełnić w pliku [kol.h](#) definicję typu `struct interesant` (nie zmieniając nic więcej w tym pliku) oraz zaimplementować podane funkcje w pliku [kol.cpp](#).

Komenda kompilacji:

```
g++ @opcjeCpp main.cpp kol.cpp -o main.e
```

Różnice w pliku [opcjeCpp](#) w stosunku do pliku [opcje](#) z C są następujące: usunięte zostały opcje kompilacji: `-wvla` (która sprawiała, że użycie tablic zmiennej długości jest uznawane za usterkę), `-Wjump-misses-init` (opcja właściwa dla C) i `-std=c17`, a w zamian została dodana opcja kompilacji `-std=c++17`.

Aby Twoje rozwiązanie uzyskało maksymalną punktację, koszt czasowy każdej funkcji musi być proporcjonalny do rozmiaru parametrów i wyjścia. Wyjątkiem są funkcje `otwarcie_urzedu` i `zamkniecie_urzedu`, które mogą dodatkowo używać czasu $O(m)$.

Twoje rozwiązanie zostanie także uruchomione za pomocą narzędzia `valgrind`, które pozwala sprawdzać m.in., czy program nie miał wycieków pamięci (nieco więcej o tym narzędziu dowiesz się w scenariuszu za tydzień). Zakładamy, że pamięć po interesantach, którzy wyszli z urzędu, zostanie zwolniona przez użytkownika. W przypadku wykrycia wycieków pamięci za pomocą komendy:

```
valgrind --tool=memcheck --leak-check=yes ./main.e
```

możesz stracić od 1 do 2 punktów za zadanie.

W rozwiązaniu zadania możesz posiłkować się materiałami z wykładu.

Przykład

W załączniku: [main.cpp](#).



[kol.h](#)

17 September 2023, 1:54 PM



[main.cpp](#)

17 September 2023, 1:54 PM





[opcjeCpp](#)


17 September 2023, 1:54 PM

Submission status

Attempt number	This is attempt 1.
Submission status	Submitted for grading
Grading status	Graded
Time remaining	Assignment was submitted 20 hours 42 mins early
Last modified	Tuesday, 19 December 2023, 7:22 PM

File submissions		
	 kol.cpp	19 December 2023, 7:21 PM
	 kol.h	19 December 2023, 7:21 PM
Submission comments	▶ Comments (0)	

Feedback

Grade	5.00 / 5.00
Graded on	Wednesday, 20 December 2023, 10:18 AM
Graded by	 Łukasz Bożyk

Contact us



Follow us

 Contact site support

You are logged in as Witold Formański (Log out)

[Data retention summary](#)

[Get the mobile app](#)

[Get the mobile app](#)

This theme was developed by

conecti.me

Moodle, 4.1.10 (Build: 20240422) | moodle@mimuw.edu.pl