



## Zadanie 2

**Otwarto:** poniedziałek, 14 października 2024, 00:00

**Wymagane do:** niedziela, 3 listopada 2024, 23:59

## Kolejki napisów

### Wstęp

Biblioteka standardowa języka C++ udostępnia bardzo przydatne kontenery, np. `unordered_map` i `deque`, których nie ma w bibliotece C.

Często potrzebujemy łączyć kod C++ z kodem spadkowym w C. Celem tego zadania jest napisanie w C++ modułu obsługującego kolejki napisów, tak aby można ich używać w C. Moduł składa się z pliku nagłówkowego (z rozszerzeniem `.h`) i pliku z implementacją (z rozszerzeniem `.cpp`).

Rozwiązując to zadanie, studenci powinni poznać:

- kolejne kontenery z STL,
- sposób łączenia kodu w C++ z kodem w C,
- metody inicjowania obiektów globalnych w C++ i wynikające stąd problemy,
- preprocesor, elementy kompilowania warunkowego.

### Polecenie

Moduł `strqueue` powinien udostępniać następujące funkcje.

```
unsigned long strqueue_new();
```

Tworzy nową, pustą kolejkę napisów i zwraca jej identyfikator.

```
void strqueue_delete(unsigned long id);
```

Jeżeli istnieje kolejka napisów o identyfikatorze `id`, usuwa ją, a w przeciwnym przypadku nic nie robi.

```
size_t strqueue_size(unsigned long id);
```

Jeżeli istnieje kolejka napisów o identyfikatorze `id`, zwraca liczbę jej elementów, a w przeciwnym przypadku zwraca 0.

```
void strqueue_insert_at(unsigned long id, size_t position, const char* str);
```

Jeżeli istnieje kolejka napisów o identyfikatorze `id` oraz `str != NULL`, wstawia napis `str` przed pozycją `position` lub na koniec kolejki, jeżeli wartość `position` jest większa lub równa liczbie napisów w kolejce. Jeżeli kolejka nie istnieje lub `str == NULL`, to nic nie robi. Pozycje w kolejce numerowane są od zera.

```
void strqueue_remove_at(unsigned long id, size_t position);
```

Jeżeli istnieje kolejka napisów o identyfikatorze `id` i wartość `position` jest mniejsza niż liczba napisów w kolejce, usuwa napis na pozycji `position`, a w przeciwnym przypadku nic nie robi.

```
const char* strqueue_get_at(unsigned long id, size_t position);
```

Jeżeli istnieje kolejka napisów o identyfikatorze `id` i wartość `position` jest mniejsza niż liczba napisów w kolejce, zwraca wskaźnik do napisu na pozycji `position`, a w przeciwnym przypadku zwraca wartość `NULL`.

```
void strqueue_clear(unsigned long id);
```

Jeżeli istnieje kolejka napisów o identyfikatorze `id`, usuwa z niej wszystkie napisy, a w przeciwnym przypadku nic nie robi.

?

```
int strqueue_comp(unsigned long id1, unsigned long id2);
```

Porównuje leksykograficznie kolejki napisów o identyfikatorach `id1` i `id2`, zwracając

- `-1`, gdy `kolejka(id1) < kolejka(id2)`,
- `0`, gdy `kolejka(id1) == kolejka(id2)`,
- `1`, gdy `kolejka(id1) > kolejka(id2)`.

Jeżeli kolejka napisów o którymś z identyfikatorów nie istnieje, to jest traktowana jak kolejka pusta.

## Wymagania formalne

Oczekiwane rozwiązanie powinno korzystać z kontenerów i metod udostępnianych przez standardową bibliotekę C++. Nie należy definiować własnych struktur, unii lub klas.

Kolejka napisów powinna przechowywać kopie napisów, a nie wartości przekazanych jej wskaźników.

Powinna być też możliwość używania wyżej opisanych funkcji w języku C++. Przy kompilowaniu pliku nagłówkowego modułu w C++ deklaracje funkcji powinny znaleźć się w przestrzeni nazw `cxx`.

Należy ukryć przed światem zewnętrznym wszystkie zmienne globalne i funkcje pomocnicze nienależące do wyspecyfikowanego interfejsu modułu.

Funkcje powinny wypisywać na standardowe wyjście diagnostyczne informacje o parametrach wywołania i wyniku wykonania. Szczegółowy format komunikatów diagnostycznych pokazują poniżej zamieszczone przykłady użycia. Zachowanie niezmienników i spójność danych można sprawdzać za pomocą asercji. Kompilowanie z parametrem `-DNDEBUG` powinno wyłączać wypisywanie informacji diagnostycznych i asercje. Obsługa standardowego wyjścia diagnostycznego powinna być realizowana z użyciem strumienia C++, czyli biblioteki `iostream`.

Nie należy nadużywać kompilowania warunkowego. Fragmenty tekstu źródłowego realizujące wyspecyfikowane operacje na kolejkach nie powinny zależeć od sposobu kompilowania (parametr `-DNDEBUG` lub jego brak) – inaczej posiadanie wersji diagnostycznej nie miałoby sensu.

Rozwiązanie powinno zawierać pliki `strqueue.h` i `strqueue.cpp`, które należy umieścić w Moodle.

Rozwiązanie będzie kompilowane i testowane na maszynie students.

## Ocenianie rozwiązania

### Ocena automatyczna

Za testy automatyczne zostanie przyznana ocena z przedziału od 0 do 6 punktów. Za błędną nazwę pliku zostanie odjęty 1 punkt. Za ostrzeżenia wypisywane przez kompilator zostanie odjęty 1 punkt. Nie ma punktów ułamkowych.

### Ocena jakości kodu

Ocena jakości kodu jest z przedziału od 0 do 4 punktów. Nie ma punktów ułamkowych. Odejmujemy punkty za:

- brzydki [styl](#) (niepoprawne wstawianie spacji, wcięć, odstępów, brak komentarzy, magiczne stałe itd.);
- dłubanie własnych klas, struktur lub algorytmów zamiast użycia STL-owych;
- zły dobór typu lub kontenera, brak nazw typów, niewiele mówiące nazwy typów;
- rozwlekłą lub nieelegancką strukturę programu, rozpatrywanie zbyt wielu warunków brzegowych, powtarzanie kodu, nieefektywne korzystanie z klasy `string`, np. `if (str != "")` zamiast `if (!str.empty())`, przechowywanie liczb jako napisów;
- korzystanie z wejścia-wyjścia dostarczanego przez bibliotekę C zamiast ze strumieni;
- wprowadzanie sztucznych ograniczeń na rozmiar danych;
- nieusuwanie lub nieefektywne usuwanie niepotrzebnych już danych;
- nieskuteczną obsługę (lub jej brak) problemu „static initialization order fiasco” (czytanka „Inicjowanie obiektów globalnych”), o ile nie zostanie to wykryte przez testy automatyczne; inicjowanie strumienia diagnostycznego w każdej funkcji (przez powielenie kodu inicjującego);
- niepoprawne pliki nagłówkowe, czyli brak `include guard` `#ifndef #define #endif` lub `#ifdef __cplusplus`;
- użycie `#if` zamiast `if constexpr` lub nieużycie `if constexpr`;
- nieukrycie przed światem zewnętrznym wszystkich zmiennych globalnych i funkcji pomocniczych nienależących do wyspecyfikowanego interfejsu modułu;
- uzależnienie fragmentów tekstu źródłowego realizujące wyspecyfikowane operacje na kolejkach od sposobu

?

kompilowania (parametr `-DNDEBUG` lub jego brak);

- braki w wypisywanych informacjach diagnostycznych; wypisywanie informacji diagnostycznych w złych miejscach, np. informacja o wywołaniu funkcji powinna być wypisana na początku, zanim funkcja zacznie coś robić, a informacja o wykonaniu dopiero wtedy, gdy struktura danych została poprawnie zmodyfikowana lub zbadana;
- użycie `typedef` zamiast `using`;
- błędy w stosowaniu przestrzeni nazw;
- wielokrotne wyszukiwanie tego samego klucza w mapie;
- inne znalezione i niewymienione w powyższych kryteriach błędy niewykryte przez testy automatyczne.

Ponadto piętnujemy:

- przekazywanie funkcjom dużych argumentów przez wartość – takie obiekty przekazujemy przez stałą referencję, czyli `const &`; na razie tylko wskazujemy te błędy i nie odejmujemy za nie punktów, bo są to zagadnienia pojawiające się w kolejnych zadaniach, w których już będziemy za to karać.

## Przykłady użycia

Przykłady użycia znajdują się w plikach `strqueue_test_1.c` i `strqueue_test_2.cpp`. Przykłady informacji diagnostycznych wypisywanych przez `strqueue_test_1_dbg`, `strqueue_test_2_dbg_a` i `strqueue_test_2_dbg_b` znajdują się w plikach `strqueue_test_1.log` i `strqueue_test_2.log`.

Kompilowanie przykładów:

```
g++ -c -Wall -Wextra -O2 -std=c++17 strqueue.cpp -o strqueue_dbg.o
g++ -c -Wall -Wextra -O2 -std=c++17 -DNDEBUG strqueue.cpp -o strqueue_nodbg.o
gcc -c -Wall -Wextra -O2 -std=c17 strqueue_test_1.c -o strqueue_test_1.o
g++ -c -Wall -Wextra -O2 -std=c++17 strqueue_test_2.cpp -o strqueue_test_2.o
g++ strqueue_test_1.o strqueue_dbg.o -o strqueue_test_1_dbg
g++ strqueue_test_1.o strqueue_nodbg.o -o strqueue_test_1_nodbg
g++ strqueue_test_2.o strqueue_dbg.o -o strqueue_test_2_dbg_a
g++ strqueue_dbg.o strqueue_test_2.o -o strqueue_test_2_dbg_b
g++ strqueue_test_2.o strqueue_nodbg.o -o strqueue_test_2_nodbg
```

[strqueue\\_test\\_1.c](#)

16 października 2024, 11:47

[strqueue\\_test\\_1.log](#)

10 października 2024, 01:51

[strqueue\\_test\\_2.cpp](#)

10 października 2024, 01:51



[strqueue\\_test\\_2.log](#)

10 października 2024, 01:51

[zadanie2\\_testy.zip](#)

4 listopada 2024, 19:19

## Status przesłanego zadania

Grupa	zadanie 2 grupa 4 zespół 5
Status przesłanego zadania	Przesłane do oceny
Stan oceniania	Ocenione
Pozostały czas	Zadanie zostało przesłane 20 min. 47 sek. przed terminem
Ostatnio modyfikowane	niedziela, 3 listopada 2024, 23:38
Przesyłane pliki	<div><div> <a href="#">strqueue.cpp</a> 3 listopada 2024, 23:38</div><div> <a href="#">strqueue.h</a> 1 listopada 2024, 18:48</div></div>
Komentarz do przesłanego zadania	<a href="#">▶ Komentarze (0)</a>

## Informacja zwrotna

Ocena	4,00 / 4,00
Ocenione dnia	poniedziałek, 9 grudnia 2024, 13:08
Ocenione przez	JP    Jakub Pawlewicz

### Komentarz zwrotny




Należało uprościć funkcje w których występowały długie specyfikacje typów, np. `get_queues` można było napisać tak:

```
auto &get_queues(void)...
```

Skontaktuj się z nami



Obserwuj nas

 Skontaktuj się z pomocą techniczną

Jesteś zalogowany(a) jako [Krzysztof Hałubek](#) (Wyloguj)

[Podsumowanie zasad przechowywania danych](#)

[Pobierz aplikację mobilną](#)



[Pobierz aplikację mobilną](#)

Motyw został opracowany przez

conecti.me

Moodle, 4.1.16 (Build: 20250210) | [moodle@mimuw.edu.pl](mailto:moodle@mimuw.edu.pl)