Zestaw 2

Kolejka

Algorytmy i strukury danych I

Kolejka

> ***Kolejka*** *(ang.* queue*) to następna podstawowa struktura danych, która implementuje zbiory dynamiczne. Elementy są usuwane z kolejki w kolejności od* najwcześniej *dodanego (strategia first-in, first-out -* ***FIFO****).*

Proszę przeczytać artykuł o implementacji bufora cyklicznego.

Interfejs

```
class Queue {
  Queue();              // Konstruktor domyślny
  void push(int x);     // Wstawia element x do kolejki (także enqueue)
  int pop();            // Usuwa element z kolejki i zwraca jego wartość (dequeue)
  int size();           // Zwraca liczbę elementów w kolejce
  bool empty();         // Sprawdza czy kolejka jest pusta
};
```

Uwagi

Operacje mają mieć złożoność $O(1)$. Złożoność obliczeniowa programów powinna być *optymalna* dla danej implementacji.

W przypadku wystąpienia błędu **niedomiaru** lub **przepełnienia** operacje powinny wyrzucać wyjątek std::out_of_range.

Dla prostoty, cała implementacja klasy może znajdować się w jednym pliku nagłówkowym. Taka konstrukcja jest konieczna w przypadku szablonów klas. Pliki źródłowe muszą mieć podaną nazwę, a programy wykonywalne muszą mieć *rozszerzenie* `.x`.

Zadanie 1. Implementacja tablicowa kolejki (`ArrayQueue.hpp`)

Napisać klasę `Queue`, która implementuje kolejkę w oparciu o bufor cykliczny. Dodać konstruktor, który jako argument przyjmuje rozmiar tablicy. Nie używać klasy `std::vector`.

Zadanie 2. Implementacja wskaźnikowa kolejki (`LinkedQueue.hpp`)

Napisać implementację wskaźnikową kolejki.

Zadanie 3. Kolejka (`Queue.cpp`)

Program `Queue.x` ma wczytać ze standardowego wejścia dane wg poniższego formatu wygenerowane przez program opisany w zadaniu **Generator** z Zestawu 1. Wynik działania odpowiednich operacji na kolejce wypisać na standardowe wyjście. Kolejka przechowuje elementy typu `int`.

Format danych: W pierwszej linii podana jest liczba $n \leq 10^6$ wskazującą na liczbę operacji do wykonania oraz n linii poleceń. Operacje mogą być następującego typu:

- `A x` - wstaw do kolejki liczbę $0 \leq x \leq 10^6$
- `D` - pobierz element z kolejki i go wypisz, jeśli kolejka jest pusta wypisz "EMPTY"
- `S` - wypisz liczbę elementów w kolejce

> ***Uwaga:*** *Programy* ***muszą*** *wczytywać dane wejściowe ze standardowego wejścia i wypisać rezultat na standardowe wyjście.*

Zadanie 4. Sortowanie pozycyjne (`Radix.cpp`)

Zaimplementuj algorytm sortowania pozycyjnego przy użyciu (tablicy) dziesięciu *kolejek*. Należy użyć własnej implementacji *kolejki*. Napisać funkcję `void radix(std::vector<int>& v)`, która sortuje w kolejności rosnącej wektor `v`. Założyć, że liczby są nieujemne i mniejsze od 10^9. Funkcja `main` wczytuje dane do wektora `std::vector<int>` (patrz wskazówki w *Materiałach*), wywołuje funkcję `radix` i wypisuje elementy posortowanego wektora przy użyciu pętli *for-each*:

```
for(const auto& i : v)
    std::cout << i << std::endl;
```

Pytania

- Opisz trzy sposoby obsługi cykliczności bufora.
- Omów przykłady zastosowania **kolejki**?
- Co oznaczają akronimy *LIFO* i *FIFO*?

Uwagi

- Na platformę Pegaz należy wysłać spakowany katalog w formacie `.tar.gz` lub `zip`.
- Katalog musi się nazywać `Zestaw03` i zawierać tylko pliki źródłowe i `Makefile`.
- Pliki źródłowe muszą mieć podaną nazwę, a programy wykonywalne muszą mieć *rozszerzenie* `.x`.
- Wywołanie komendy `make` w tym katalogu powinno kompilować wszystkie programy i tylko kompilować.
- Kompilacja musi przebiegać bez błędów ani ostrzeżeń.
- Należy używać własnych implementacji typów danych w programach.
- Programy nie powinny wypisywać niczego ponad to co opisano w instrukcji. Proszę dokładnie czytać opis formatu danych wejściowych i wyjściowych.
- Implementacje klas mogą znajdować się w pliku nagłówkowym. Taka konstrukcja jest konieczna w przypadku szablonów klas.

Dodatkowe punkty

Dodatkowe punkty (po 1 pkt) można zdobyć za:

- Napisanie szablonu klas
- Wykorzystanie referencji do r-wartości, semantyki przenoszenia, uniwersalnych referencji, doskonałego przekazywanie.
- Napisanie testera

Andrzej Görlich
a.goerlich@outlook.com