List-based FIFO queue (AiSD 2021)

Zadanie polega na implementacji kolejki FIFO opartej o listę jedno albo dwukierunkową. Jest to ono rozwinięciem zadania XOR linked list. i można w nim zastosować zaimplementowaną tam listę, można również wykorzystać szablon listy dostępny z biblioteki STL <list>.

Poza funkcjonalnościami (ADD_BEG, ADD_END, DEL_BEG, DEL_END, PRINT_FORWARD, PRINT_BACKWARD) z zadania XOR linked list, lista udostępnia dodatkowo:

SIZE - liczba elementów listy.

Kolejka jest strukturą danych wykorzystującą listę i udostępniającą następujące funkcjonalności powiązane z następującymi komendami:

PUSH N - wepchnij element N na koniec kolejki.
POP - wyciągnij pierwszy element z kolejki.
PRINT_QUEUE - drukuj kolejkę od pierwszego elementu do ostatnio wepchniętego.
COUNT - liczba elementów kolejki ≤ SIZE.
GARBAGE_SOFT - wpisz wartość 0 do wszystkich elementów listy które nie należą do kolejki.
GARBAGE_HARD - usuń wszystkie elementy listy które nie należą do kolejki.

Kolejka FIIO rozbudowuje listę XOR linked list o wskaźniki na początek (FRONT) i koniec kolejki (BACK). Po wepchnięciu elementu do kolejki wskaźnik na koniec kolejki (BACK) należy przesunąć na kolejny element listy. Po wyjęciu z kolejki wskaźnik na początek (FRONT) również należy przesunąć na kolejny element listy. Widać, że podczas wpychania i wyjmowania z listy kolejka będzie "podążać wzdłuż listy". Aby uniknąć wyczerpania pamięci i móc wykorzystać elementy listy przez kolejkę więcej niż raz, należy albo pracować na liście cyklicznej albo po dotarciu na koniec listy rozpocząć przeglądanie od początku. Należy również obsłużyć sytuacje wyjątkowe czyli: wepchnięcie elementu w wypadku pustej listy (brak elementów) albo pełnej listy (liczba elementów kolejki równa liczbie elementów listy) oraz wyciągnięcie elementu z pustej kolejki.

Kiedy lista jest pusta (nie posiada elementów) wskaźniki oczywiście wskazują na nullptr. Kiedy wpychamy element (PUSH N) do kolejki a lista jest pusta należy przed tą operacją dodać jeden element do listy. Tak samo należy powiększyć listę, kiedy znajdują się w niej już jakieś elementy ale wszystkie są zajęte przez kolejkę. Dodawany element listy powinien być przed elementem na który wskazuje wskaźnik na koniec kolejki (BACK).

Co jakiś czas może okazać się przydatne usunięcie elementów listy, które w danym momencie nie przechowują żadnych elementów kolejki. Taka sytuacja może mieć miejsce kiedy przez dłuższy czas jedynie wpychaliśmy elementy do kolejki a później większość z nich wyciągnęliśmy. Obrazuje to często spotykaną sytuację związaną z przeciążeniem systemów obsługujących kolejkowane zadania np. system STOS. Normalnie kolejka w STOS'ie z zadaniami studentów jest pusta albo zawiera kilka zadań ale pod koniec deadline'ów potrafi się bardzo mocno rozrosnąć co prowadzi do przeciążenia a w rezultacie może owocować upadkiem systemu. Komenda GARBAGE_HARD realizuje właśnie funkcjonalność zmniejszania rozmiaru listy.

The task is to implement FIFO queue using doubly or singly linked list. It is an extension of XOR linked list assignment XOR linked list and the list implemented for that assignment can be reused; but also an STL <list> list can be used.

In addition to the functions of the XOR list (ADD_BEG, ADD_END, DEL_BEG, DEL_END, PRINT_FORWARD, PRINT_BACKWARD) from the XOR list assignment XOR linked list the list implements:

SIZE - liczba elementów listy.

The queue is a data structure that uses list and provides functions corresponding to the following commands:

PUSH N - add element N to the end of the queue.
POP - remove the first element from the queue.
PRINT_QUEUE - print the queue from the first element to the lastly added.
COUNT - the number of elements in queue ≤ SIZE.
GARBAGE_SOFT - change the values of all elements of the list that do not belong to the queue to 0.
GARBAGE_HARD - remove all elements from the list that does not belong to the queue.

The FIFO queue extends the list to the pointers to the beginning (FRONT) and the end (BACK) of the queue. After inserting an element to the queue the pointer to the end of the queue (BACK) has to be changed so it will point to the next element of the list. After removing an element from the queue (FRONT) pointer has to start pointing at the element after the previously pointer by (FRONT). Notice, that by both removing and adding the element the queue has to “follow the list”. To avoid memory loss and to be able to reuse one can use a cyclical list or after reaching the end of the queue start from the beginning. Also, one has to handle exceptions: inserting an element to an empty queue (no elements), adding elements to full list (the number of elements of the queue equal to the number of elements of the list), or trying to remove elements from an empty queue.

When the list is empty (no elements in it) the pointers point to nullptr. When adding an element (PUSH N) to the queue and the list is empty, then before executing the operation one element has to be added to the list. Similarly, the list has to be extended when the list is not empty, but all the elements are occupied by the queue. Inserted element should be inserted before the element pointed by pointer to the end of the queue (BACK).

From time to time it might be useful to remove the elements that do not contain elements from the queue. Such a situation may occur if there were many insertions and after the insertions there was a sequence of pops. The situation is quite common in queuing systems. Consider for example STOS system. Normally the queues are empty or contains a few tasks. But near the deadlines they may grow considerably, which can overload the system and crash it. The operation to shrink the size of the list is implemented by GARBAGE_HARD.

Wejście

Pewna liczba komend uruchamiających określone funkcjonalności na liście albo na kolejce.

Wyjście

Wyniki działania odpowiednich komend na kolejce albo bezpośrednio na liście charakteryzującej się pewnym stanem. Początkowo lista jak i kolejka jest pusta a późniejszy jej stan zależy od kolejno wywoływanych komend. Niektóre komendy nie generują żadnego wyjścia np. (ADD_BEG N, PUSH N) ale mają wpływ na stan listy i pośrednio na stan kolejki oraz na kolejne komendy które wyświetlają pewne informacje np. (POP, PRINT_QUEUE).

Input

Some number of commands for the list or for the queue.

Output

The results of the operations of commands on the queue or directly on the list characterized by some state. Initially the list and the queue are empty, later their state depends on the commands. Some commands do not generate any output, but influence the state of the list and indirectly the queue and for the consecutive commands which print some in formations.

Przykład/Example

Wejście1/Input1

PUSH 1

PRINT_FORWARD

PRINT_QUEUE

SIZE

COUNT

PUSH 2

PRINT_FORWARD

PRINT_QUEUE

SIZE

COUNT

PUSH 3

PRINT_FORWARD

PRINT_QUEUE

SIZE

COUNT

POP

PRINT_FORWARD

PRINT_QUEUE

SIZE

COUNT

PUSH 4

PRINT_FORWARD

PRINT_QUEUE

SIZE

COUNT

POP

PRINT_FORWARD

PRINT_QUEUE

SIZE

COUNT

POP

PRINT_FORWARD

PRINT_QUEUE

SIZE

COUNT

POP

PRINT_FORWARD

PRINT_QUEUE

SIZE

COUNT

POP

PRINT_FORWARD

PRINT_QUEUE

SIZE

COUNT

Wyjście1/Output1

1

1

1

1

2 1

1 2

2

2

3 2 1

1 2 3

3

3

1

3 2 1

2 3

3

2

3 2 4

2 3 4

3

3

2

3 2 4

3 4

3

2

3

3 2 4

4

3

1

4

3 2 4

NULL

3

0

NULL

3 2 4

NULL

3

0

Wejście2/Input2

ADD_BEG 1

ADD_BEG 2

ADD_BEG 3

ADD_BEG 4

ADD_BEG 5

PRINT_FORWARD

PRINT_QUEUE

PUSH 10

PRINT_FORWARD

PRINT_QUEUE

PUSH 11

PRINT_FORWARD

PRINT_QUEUE

PUSH 12

PRINT_FORWARD

PRINT_QUEUE

PUSH 13

PRINT_FORWARD

PRINT_QUEUE

PUSH 14

PRINT_FORWARD

PRINT_QUEUE

PUSH 15

PRINT_FORWARD

PRINT_QUEUE

POP

POP

POP

PRINT_FORWARD

PRINT_QUEUE

POP

POP

POP

PRINT_FORWARD

PRINT_QUEUE

Wyjście2/Output2

5 4 3 2 1

NULL

5 4 3 2 10

10

5 4 3 11 10

10 11

5 4 12 11 10

10 11 12

5 13 12 11 10

10 11 12 13

14 13 12 11 10

10 11 12 13 14

15 14 13 12 11 10

10 11 12 13 14 15

10

11

12

15 14 13 12 11 10

13 14 15

13

14

15

15 14 13 12 11 10

NULL

Wejście3/Input3

```
ADD_END 42
PRINT_FORWARD
GARBAGE_SOFT
PRINT_FORWARD
PUSH 10
PRINT_FORWARD
GARBAGE_SOFT
PRINT_FORWARD
ADD_END 6
ADD_END 5
ADD_END 4
ADD_END 3
ADD_END 2
ADD_END 1
ADD_BEG 8
ADD_BEG 9
PRINT_FORWARD
PUSH 11
PUSH 12
PUSH 13
PUSH 14
PRINT_FORWARD
GARBAGE_SOFT
PRINT_FORWARD
```

Wyjście3/Output3

42

0

10

10

9 8 10 6 5 4 3 2 1

12 11 10 6 5 4 3 14 13

12 11 10 0 0 0 0 14 13

Wejście4/Input4

PUSH 10

PUSH 11

PUSH 12

PRINT_BACKWARD

PRINT_QUEUE

ADD_BEG 1

ADD_END 6

ADD_BEG 2

ADD_BEG 3

ADD_END 5

ADD_END 4

PRINT_BACKWARD

PRINT_QUEUE

PUSH 13

PUSH 14

PUSH 15

PUSH 16

PUSH 17

PRINT_BACKWARD

PRINT_QUEUE

DEL_END

PRINT_BACKWARD

PRINT_QUEUE

DEL_END

PRINT_BACKWARD

PRINT_QUEUE

DEL_BEG

PRINT_BACKWARD

PRINT_QUEUE

DEL_BEG

PRINT_BACKWARD

PRINT_QUEUE

Wyjście4/Output4

10 11 12

10 11 12

4 5 6 10 11 12 1 2 3

10 11 12

16 17 6 10 11 12 13 14 15

10 11 12 13 14 15 16 17

17 6 10 11 12 13 14 15

10 11 12 13 14 15 17

6 10 11 12 13 14 15

10 11 12 13 14 15

6 10 11 12 13 14

10 11 12 13 14

6 10 11 12 13

10 11 12 13

Wejście5/Input5

ADD_END 42

PRINT_FORWARD

GARBAGE_HARD

PRINT_FORWARD

PUSH 10

PRINT_FORWARD

GARBAGE_HARD

PRINT_FORWARD

ADD_END 6

ADD_END 5

ADD_END 4

ADD_END 3

ADD_END 2

ADD_END 1

ADD_BEG 8

ADD_BEG 9

PRINT_FORWARD

PUSH 11

PUSH 12

PUSH 13

PUSH 14

PRINT_FORWARD

GARBAGE_HARD

PRINT_FORWARD

Wyjście5/Output5

42

NULL

10

10

9 8 10 6 5 4 3 2 1

12 11 10 6 5 4 3 14 13

12 11 10 14 13

Wejście6/Input6

PUSH 5

PUSH 1

PUSH 2

PUSH 3

POP

PUSH 4

PRINT_FORWARD

PRINT_QUEUE

PUSH 5

PRINT_FORWARD

PRINT_QUEUE

POP

PUSH 6

PRINT_FORWARD

PRINT_QUEUE

POP

PUSH 7

PUSH 8

PRINT_FORWARD

PRINT_QUEUE

POP

PUSH 8

PUSH 10

PUSH 11

PRINT_FORWARD

PRINT_QUEUE

POP

POP

POP

POP

POP

POP

POP

POP

POP

PRINT_FORWARD

PRINT_QUEUE

Wyjście6/Output6

5

3 2 1 4

1 2 3 4

3 2 1 5 4

1 2 3 4 5

1

3 2 6 5 4

2 3 4 5 6

2

3 8 7 6 5 4

3 4 5 6 7 8

3

11 10 8 8 7 6 5 4

4 5 6 7 8 8 10 11

4

5

6

7

8

8

10

11

NULL

11 10 8 8 7 6 5 4

NULL

Wejście7/Input7

PUSH 5

PUSH 6

PUSH 1

PUSH 2

PUSH 3

PUSH 4

POP

POP

PUSH 7

PUSH 8

PRINT_FORWARD

PRINT_QUEUE

ADD_BEG 5

ADD_END 6

PRINT_FORWARD

PRINT_QUEUE

POP

POP

POP

POP

PRINT_FORWARD

PRINT_QUEUE

ADD_BEG 1

ADD_BEG 2

ADD_END 4

ADD_END 3

PRINT_FORWARD

PRINT_QUEUE

GARBAGE_HARD

PRINT_FORWARD

PRINT_QUEUE

Wyjście7/Output7

6

4 3 2 1 8 7

1 2 3 4 7 8

5 4 3 2 1 8 7 6

1 2 3 4 5 6 7 8

1

2

3

4

5 4 3 2 1 8 7 6

5 6 7 8

2 1 5 4 3 2 1 8 7 6 4 3

5 1 2 3 4 6 7 8

2 1 5 8 7 6 4 3

5 1 2 3 4 6 7 8

Wejście8/Input8

PUSH 1

PUSH 2

POP

COUNT

SIZE

PRINT_FORWARD

PRINT_QUEUE

PUSH 3

PUSH 4

POP

COUNT

SIZE

PRINT_FORWARD

PRINT_QUEUE

PUSH 5

POP

POP

COUNT

SIZE

PRINT_FORWARD

PRINT_QUEUE

PUSH 6

PUSH 7

PUSH 8

COUNT

SIZE

PRINT_FORWARD

PRINT_QUEUE

ADD_BEG 9

COUNT

SIZE

ADD_END 10

COUNT

SIZE

PRINT_FORWARD

PRINT_QUEUE

DEL_BEG

COUNT

SIZE

PRINT_FORWARD

PRINT_QUEUE

DEL_END

COUNT

SIZE

PRINT_FORWARD

PRINT_QUEUE

Wyjście8/Output8

1

1

2

2 1

2

2

2

3

2 4 3

3 4

3

4

1

3

5 4 3

5

4

4

5 8 7 6

5 6 7 8

5

5

6

6

9 5 8 7 6 10

5 9 10 6 7 8

5

5

5 8 7 6 10

5 10 6 7 8

4

4

5 8 7 6

5 6 7 8