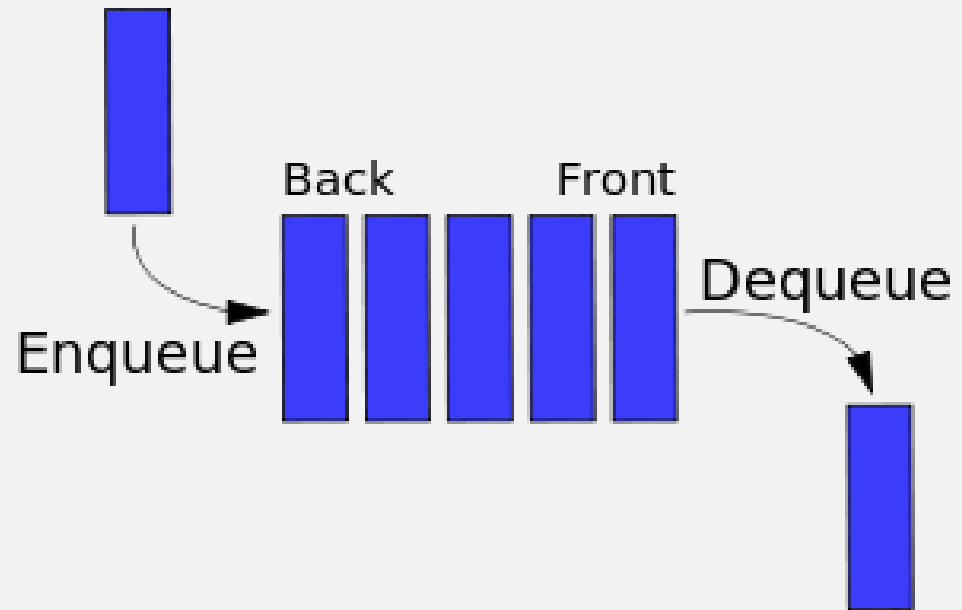


<QUEUE>

KOLEJKA FIFO



- Ostatni element jest ściągany jako ostatni.
- Operacje podstawowe:
wstaw (enqueue),
usuń (dequeue).

KONSTRUKTOR

```
#include "stdafx.h"
#include <queue>

int main()
{
    std::queue<double> kolejkaFIFO;

    std::deque<int> kolejka2(17, 97);
    std::queue<int> kolejkaFIFO2(kolejka2);

    return 0;
}
```

- Standardowe typy danych.
- Struktury.
- Klasy.
- Kopiowanie istniejącej double-ended queue.

PUSH, POP, EMPTY

```
int main()
{
    std::queue<double> kolejkaFIFO;

    for (int i = 0; i < 10; i++)
    {
        kolejkaFIFO.push(i + 1);
    }

    while (!kolejkaFIFO.empty())
    {
        kolejkaFIFO.pop();
    }

    return 0;
}
```

- push(value) – dodaj element na koniec
- pop() – usuń pierwszy element
- empty() – (bool) czy pusty

FRONT, BACK

```
int main()
{
    std::queue<double> kolejkaFIFO;

    for (int i = 0; i < 10; i++)
    {
        kolejkaFIFO.push(i + 1);
        std::cout << "Dodawanie elemntu: " << kolejkaFIFO.back() << std::endl;
    }
    std::cout << "\n" << std::endl;

    while (!kolejkaFIFO.empty())
    {
        std::cout << "Sciaganie elemntu: " << kolejkaFIFO.front() << std::endl;
        kolejkaFIFO.pop();
    }

    return 0;
}
```

FRONT, BACK

```
C:\WINDOWS\system32\cmd.exe
Dodawanie elemntu: 1
Dodawanie elemntu: 2
Dodawanie elemntu: 3
Dodawanie elemntu: 4
Dodawanie elemntu: 5
Dodawanie elemntu: 6
Dodawanie elemntu: 7
Dodawanie elemntu: 8
Dodawanie elemntu: 9
Dodawanie elemntu: 10

Sciaganie elemntu: 1
Sciaganie elemntu: 2
Sciaganie elemntu: 3
Sciaganie elemntu: 4
Sciaganie elemntu: 5
Sciaganie elemntu: 6
Sciaganie elemntu: 7
Sciaganie elemntu: 8
Sciaganie elemntu: 9
Sciaganie elemntu: 10
Press any key to continue . . .
```

- `front()` – dostęp do pierwszego elementu
- `back()` – dostęp do ostatniego elementu

SIZE

```
int main()
{
    std::queue<int> kolejkaFIFO;

    for (int i = 0; i < 10; i++)
    {
        kolejkaFIFO.push(i + 1);
    }

    std::cout << "Ilosc elementow w kolejce: " << kolejkaFIFO.size() << std::endl;

    while (!kolejkaFIFO.empty())
    {
        kolejkaFIFO.pop();
    }

    std::cout << "Ilosc elementow w kolejce (po usunieciu): " << kolejkaFIFO.size() << std::endl;

    return 0;
}
```

SIZE

```
C:\WINDOWS\system32\cmd.exe
Ilosc elementow w kolejce: 10
Ilosc elementow w kolejce (po usunieciu): 0
Press any key to continue . . . _
```

- `size()` – zwraca długość/rozmiar kolejki

SWAP

```
int main()
{
    std::queue<int> kolejka1;
    std::queue<int> kolejka2;

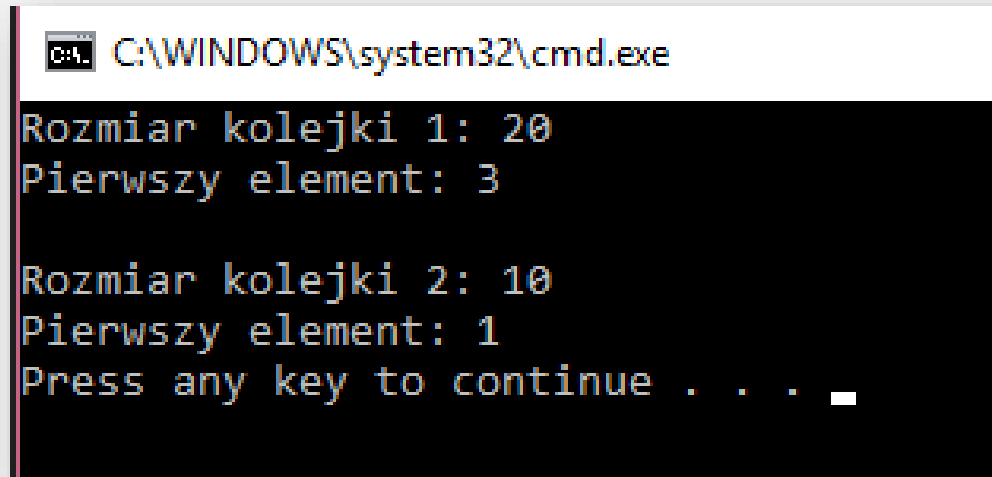
    for (int i = 0; i < 10; i++)
    {
        kolejka1.push(i + 1);
        kolejka2.push(i + 3);
        kolejka2.push(i + 7);
    }

    kolejka1.swap(kolejka2);

    std::cout << "Rozmiar kolejki 1: " << kolejka1.size() << "\nPierwszy element: " << kolejka1.front() << std::endl;
    std::cout << "\nRozmiar kolejki 2: " << kolejka2.size() << "\nPierwszy element: " << kolejka2.front() << std::endl;

    return 0;
}
```

SWAP



```
C:\WINDOWS\system32\cmd.exe
Rozmiar kolejki 1: 20
Pierwszy element: 3

Rozmiar kolejki 2: 10
Pierwszy element: 1
Press any key to continue . . .
```

- `swap()` – zamienia dwie kolejki ze sobą (typy muszą być zgodne)

KOLEJKA PRIORYTETOWA

```
int main()
{
    std::queue<int> kolejka;
    std::priority_queue<int> kolejkaPriorytetowa;

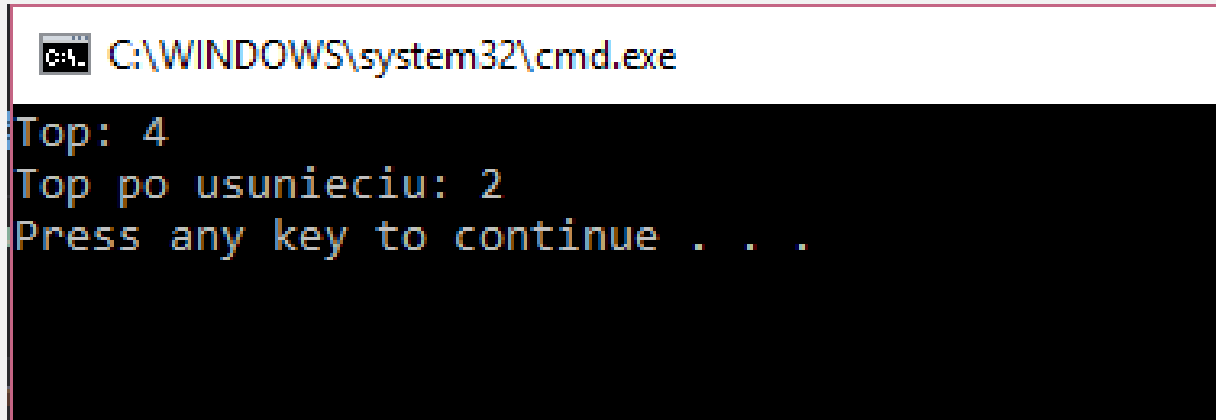
    kolejkaPriorytetowa.push(2);
    kolejkaPriorytetowa.push(4);
    kolejkaPriorytetowa.push(1);

    std::cout << "Top: " << kolejkaPriorytetowa.top() << std::endl;

    kolejkaPriorytetowa.pop();
    std::cout << "Top po usunieciu: " << kolejkaPriorytetowa.top() << std::endl;
    return 0;
}
```

- Elementy są ułożone zgodnie z priorytetem.
- Jeśli chcemy używać własnego typu danych musimy zdefiniować własną metodę porównującą.

KOLEJKA PRIORYTETOWA



```
C:\WINDOWS\system32\cmd.exe
Top: 4
Top po usunieciu: 2
Press any key to continue . . .
```

- Zamiast `front()` i `back()` pojawia się `top()`, który służy do odczytu elementu o największym priorytecie.