

Algorytmy i Struktury Danych

Dynamiczne struktury danych:

STOS, KOLEJKA, LISTA JEDNOKIERUNKOWA

Małgorzata Piekarska

VILO Bydgoszcz

Zmienne dynamiczne – co to jest?

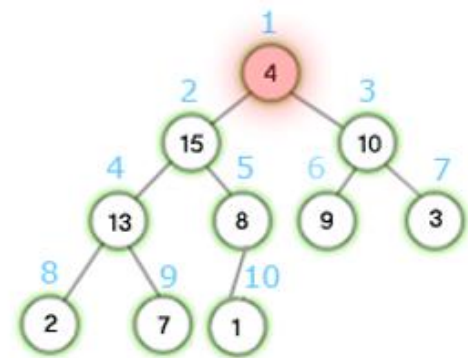
- Zmienna statyczna globalna tworzona jest w trakcie uruchomienia programu i istnieje do jego zakończenia
- Zmienna statyczna lokalna istnieje od momentu wywołania funkcji do momentu jej zakończenia
- **Zmienna dynamiczna** - zmienna utworzona (w dowolnym momencie) za pomocą specjalnej instrukcji przydzielającej tej zmiennej odpowiednią ilość pamięci, a potem za pomocą innej instrukcji usuwana z tej pamięci, kiedy przestaje być potrzebna, zwalniając w ten sposób niepotrzebnie zajmowaną pamięć.

Zmienne dynamiczne – jak?

- Do zmiennej dynamicznej odwołujemy się **przez wskaźnik**
- Wskaźniki nie przechowują bezpośrednio samych danych, lecz **adres** pojedynczej komórki pamięci, gdzie znajduje się zmienna lub gdzie zaczyna się struktura

Zmienne dynamiczne – po co?

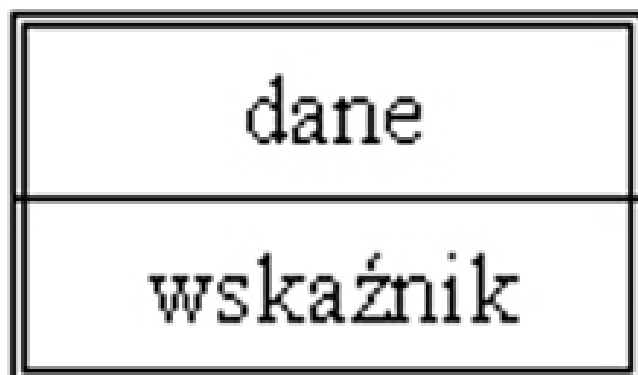
- Lepsze gospodarowanie pamięcią w programie
- Możliwość konstruowania struktur o dynamicznej (zmiennej) wielkości
- Możliwość konstruowania struktur o dowolnej (złożonej) budowie



Dynamiczne struktury danych

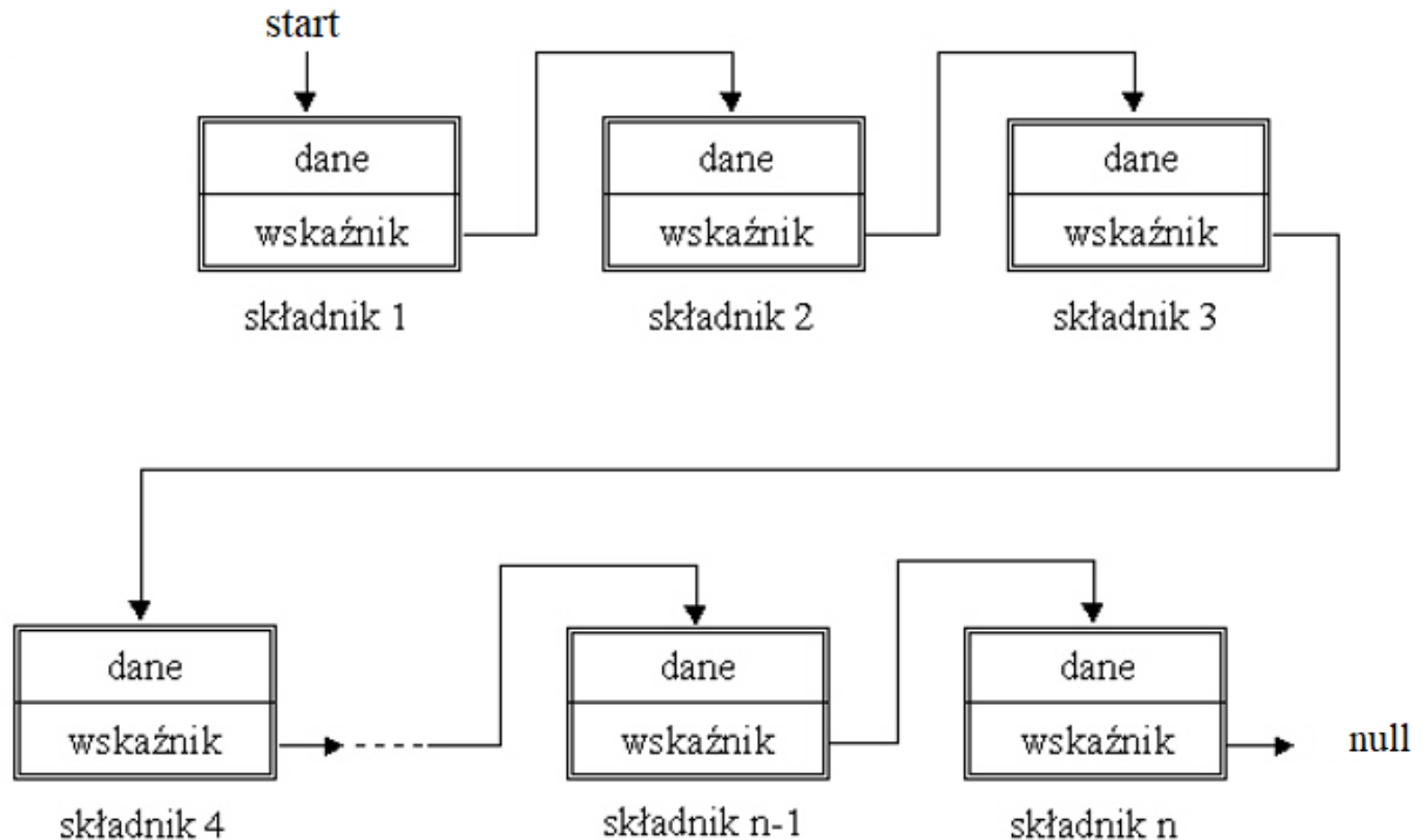
- Szczególne znaczenie ma struktura danych o budowie liniowej, w której każdy obiekt ma swojego poprzednika i swojego następnika.
- **Elementarnym składnikiem** takiej struktury danych jest zmiennej dynamiczna typu rekordowego (**struct**), która zawiera pola dany

Elementarny **skladnik** - struct



```
struct skladnik  
{  
    typ_danych val;  
    skladnik *next;  
};
```

Przykład takiej struktury



STOS

- Struktura, w której wstawianie, usuwanie i jakikolwiek dostęp do elementów jest możliwy tylko i wyłącznie w jednym miejscu, zwanym wierzchołkiem stosu

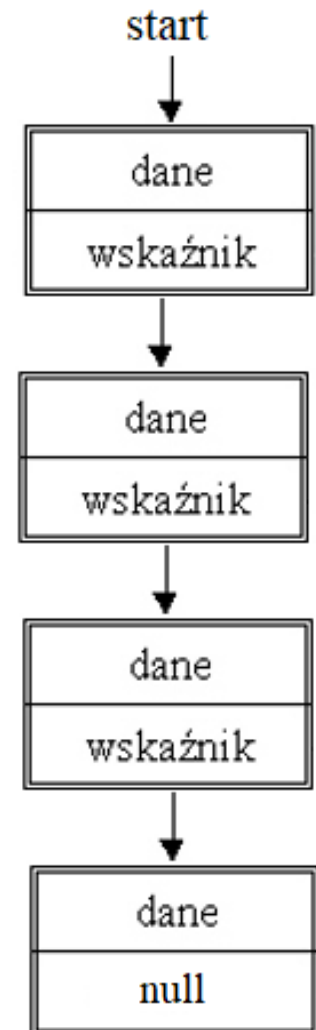


Last In First Out

STOS

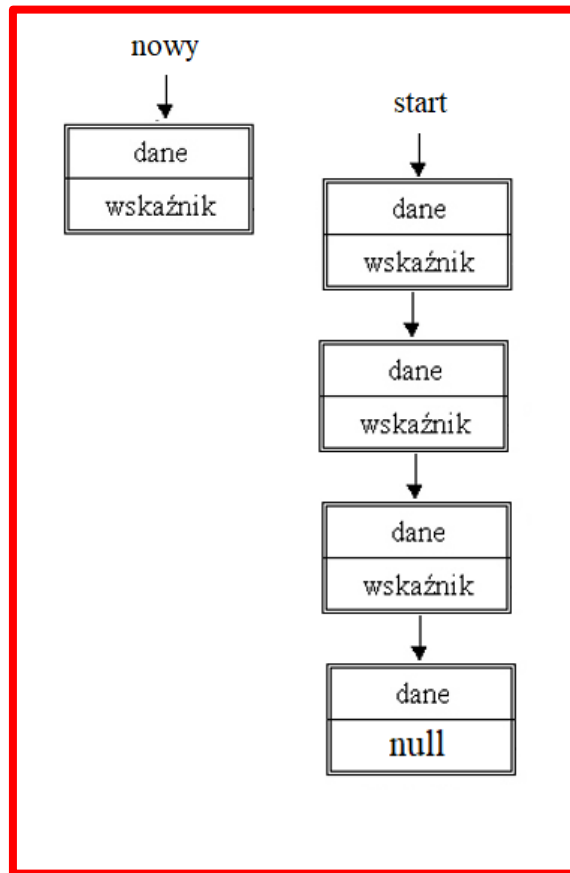
- Struktura, w której wstawianie, usuwanie i jakikolwiek dostęp do elementów jest możliwy tylko i wyłącznie w jednym miejscu, zwanym wierzchołkiem stosu

Last In First Out

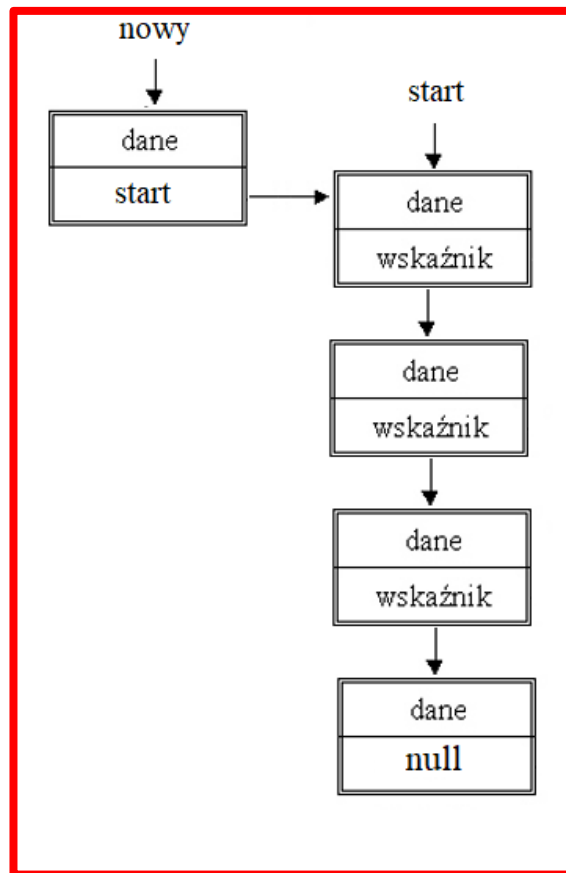


Dodawanie do stosu

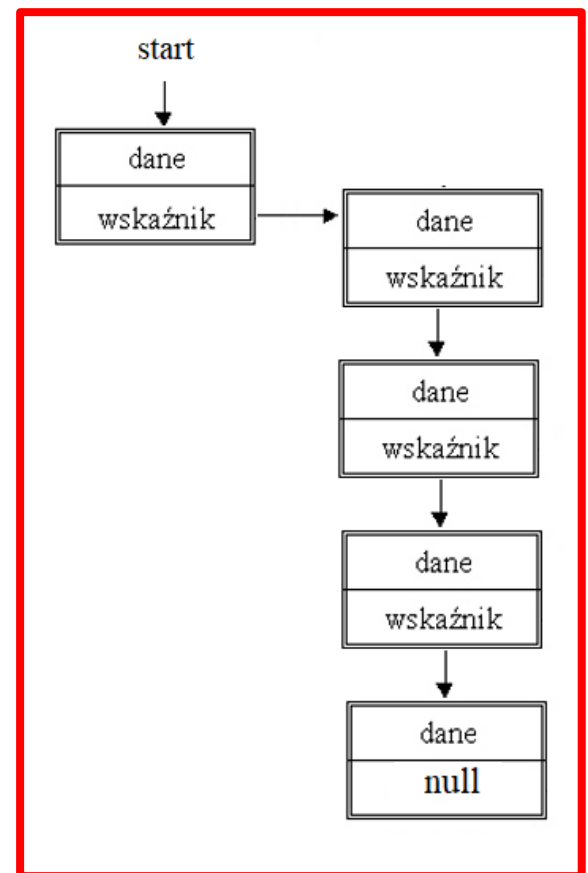
1



2

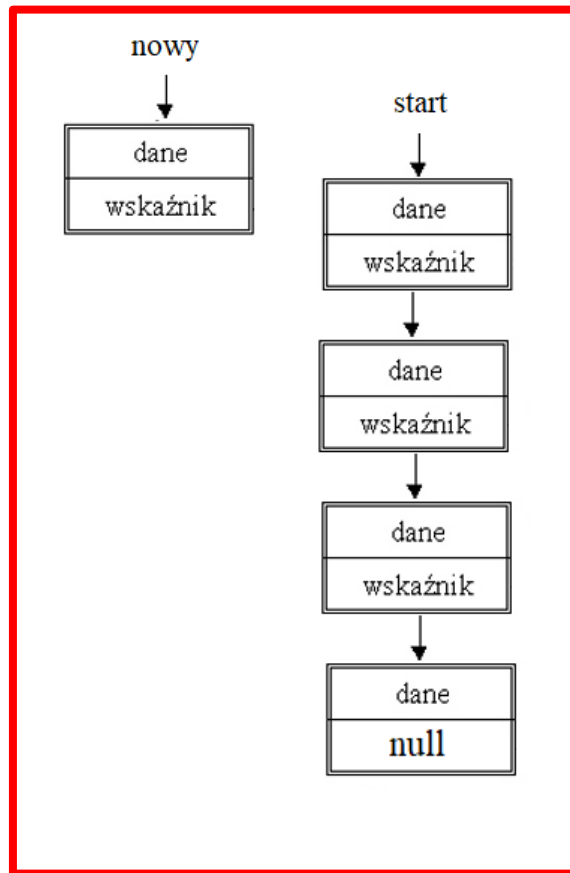


3

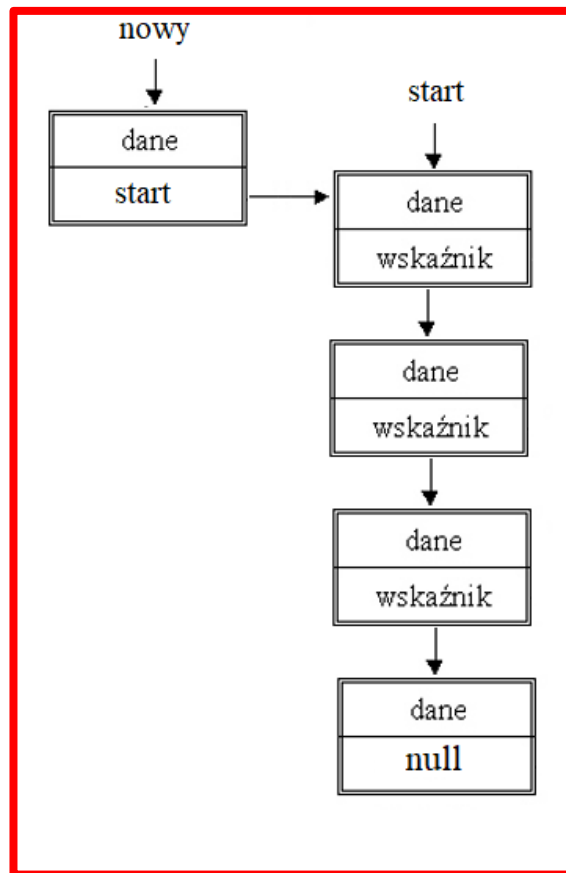


Dodawanie do stosu

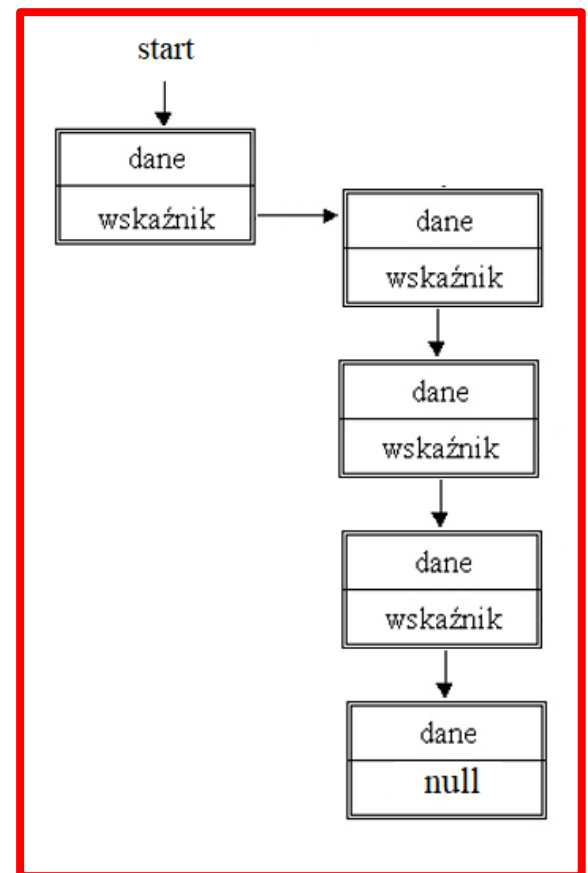
1



2



3



```
skladnik *nowy = new skladnik;  
nowy->val = dane;
```

```
nowy->next = start;
```

```
start = nowy;
```

Dodawanie do stosu

Push (dane)

```
{  
    skladnik *nowy = new skladnik;  
    nowy->val = dane;  
  
    nowy->next = start;  
  
    start = nowy;  
}
```

A co się wydarzy kiedy stos jest pusty?

Push (dane)

```
{  
    skladnik *nowy = new skladnik;  
    nowy->val = dane;  
  
    nowy->next = start;  
  
    start = nowy;  
}
```

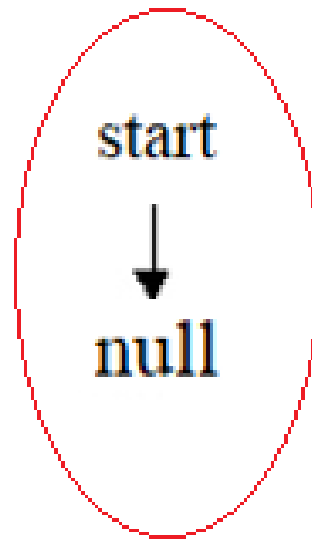
A co się wydarzy kiedy stos jest pusty?

Push (dane)

```
{  
    skladnik *nowy = new skladnik;  
    nowy->val = dane;  
  
    nowy->next = start;  
  
    start = nowy;  
}
```

Czym właściwie jest start kiedy stos jest pusty?

Pusty stos

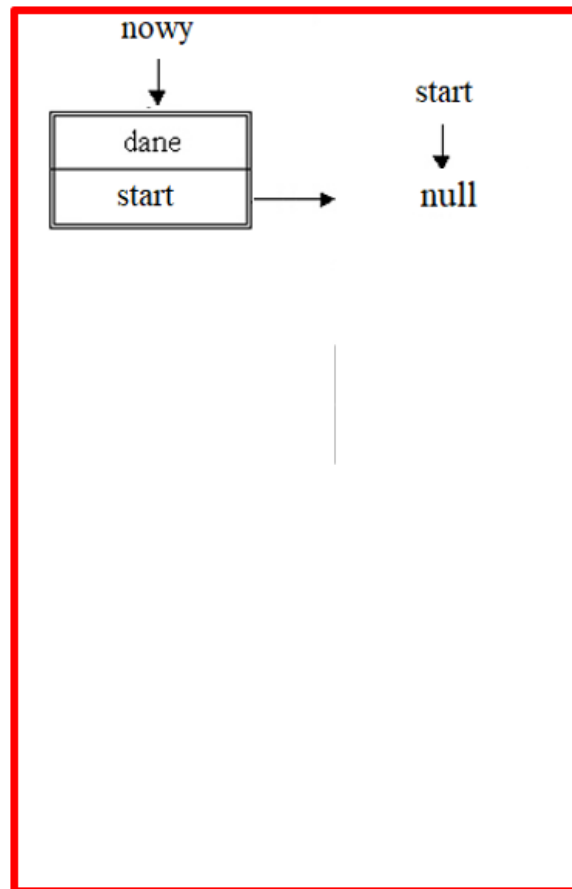


Pusty stos

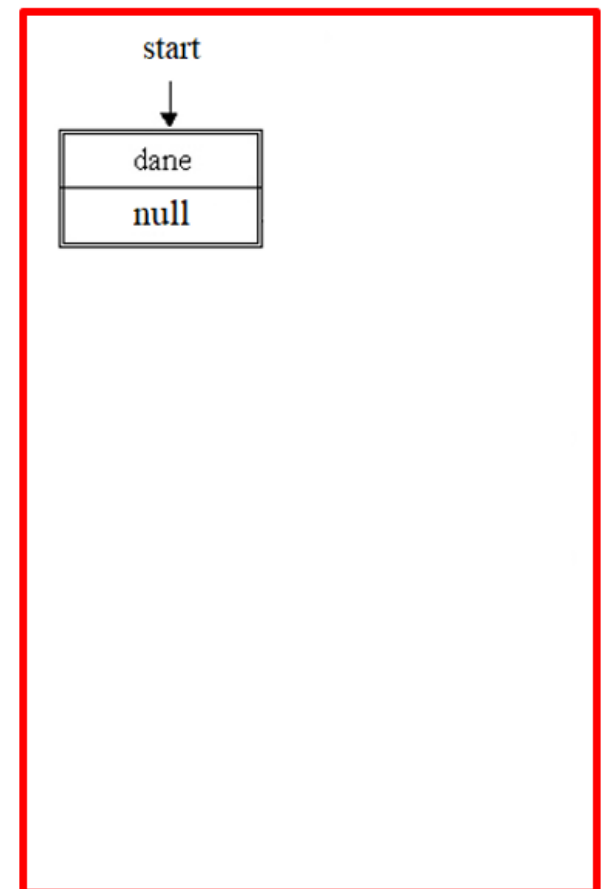
1



2



3



```
skladnik *nowy = new skladnik;  
nowy->val = dane;
```

```
nowy->next = start;
```

```
start = nowy;
```


Sprawdzenie czy stos jest pusty

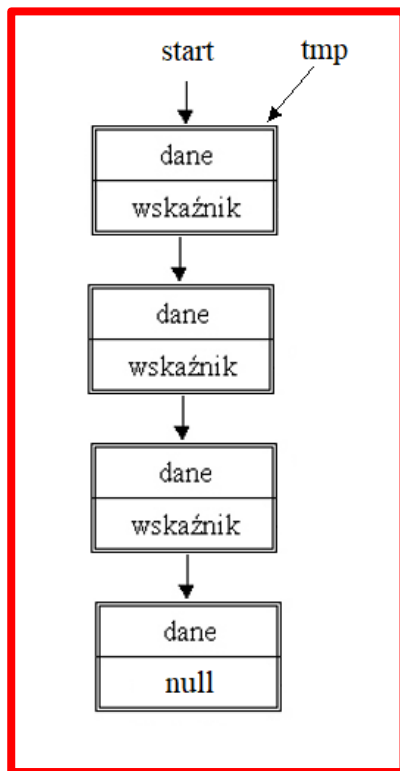
```
Function isEmpty()  
{  
    if start == null  
        return true  
    else  
        return false  
}
```

Sprawdzenie co jest na wierzchu

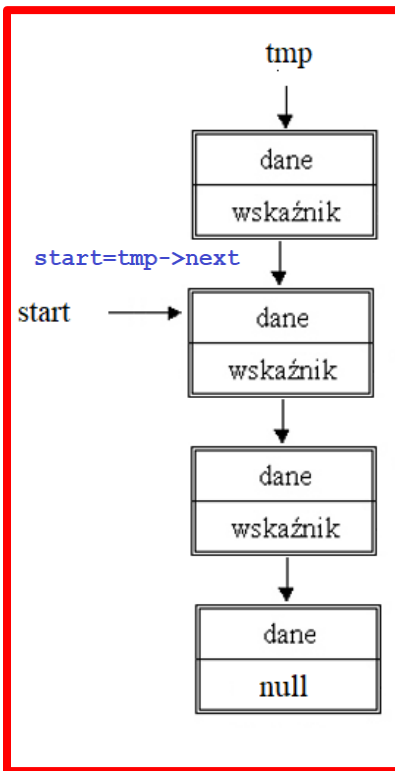
```
Function Top()  
{  
    if start != null  
        return start->val  
    else  
        return null  
}
```

Zdjęcie elementu ze stosu

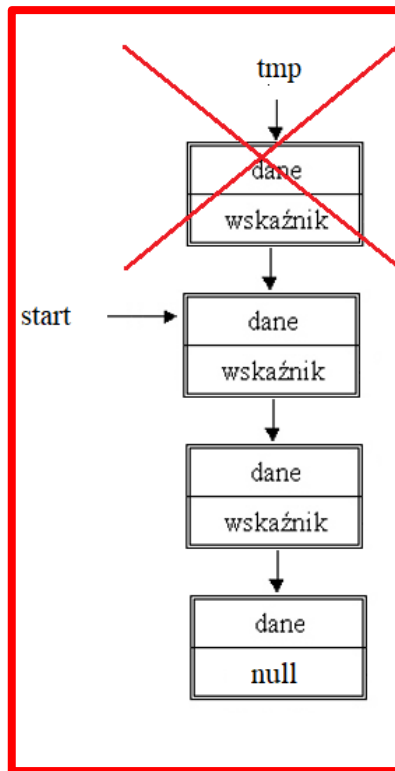
1



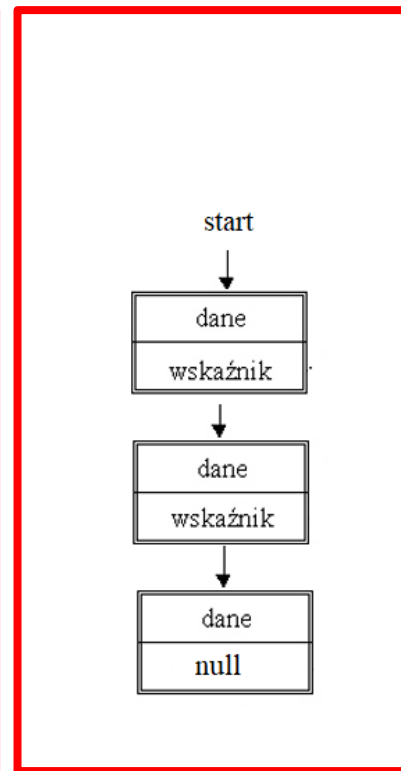
2



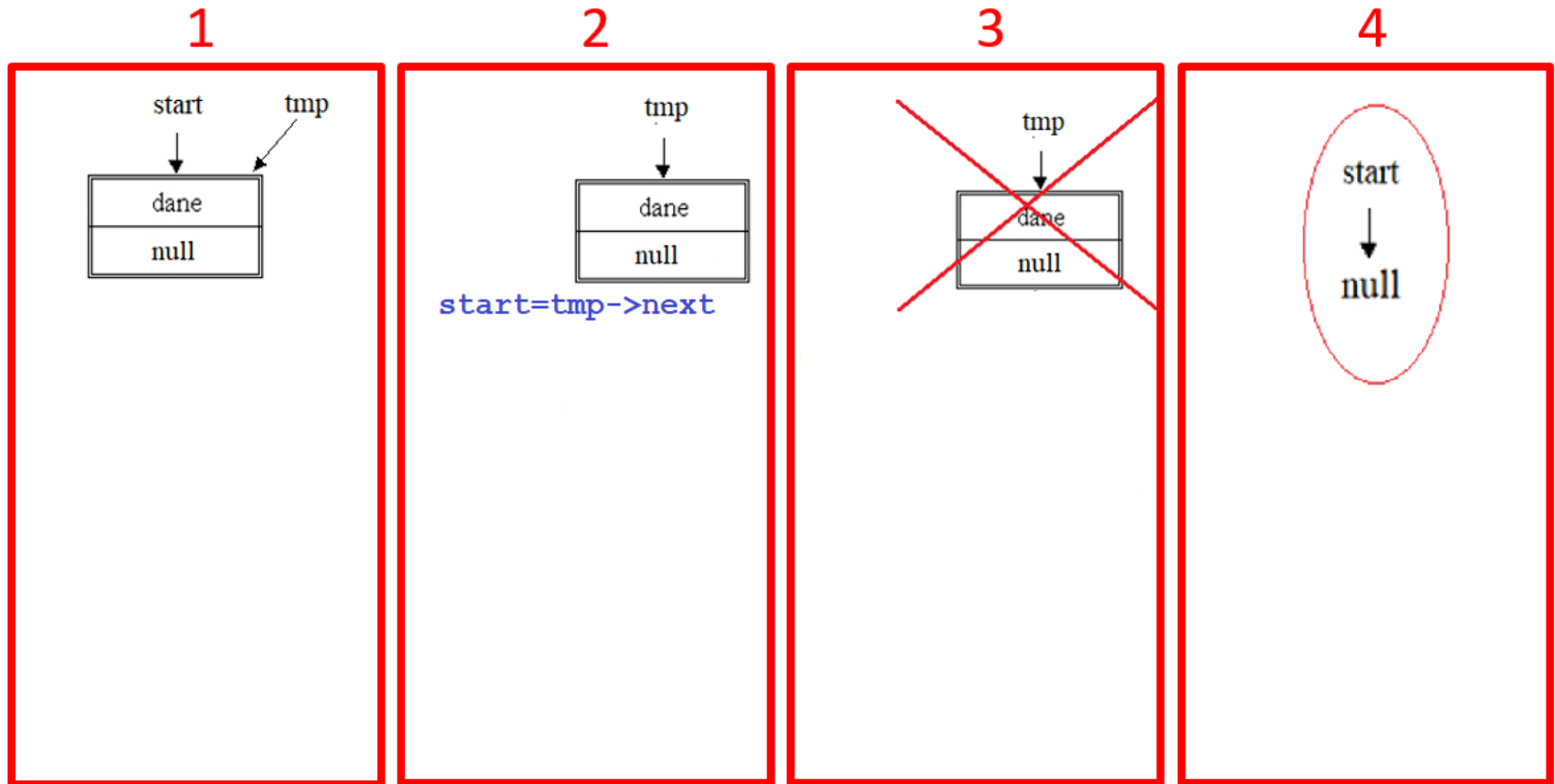
3



4



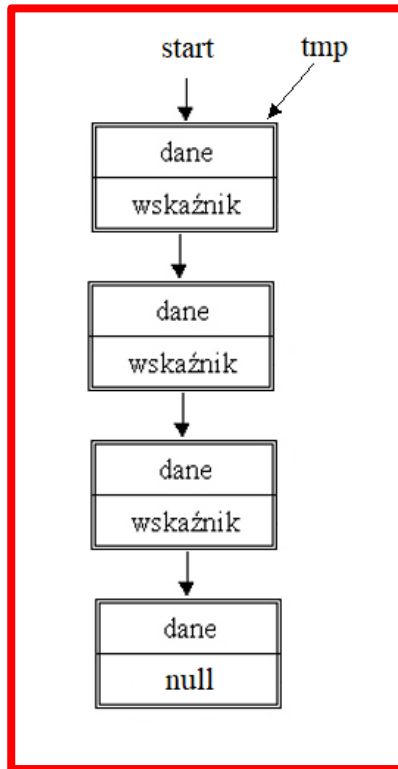
Zdjęcie elementu ze stosu



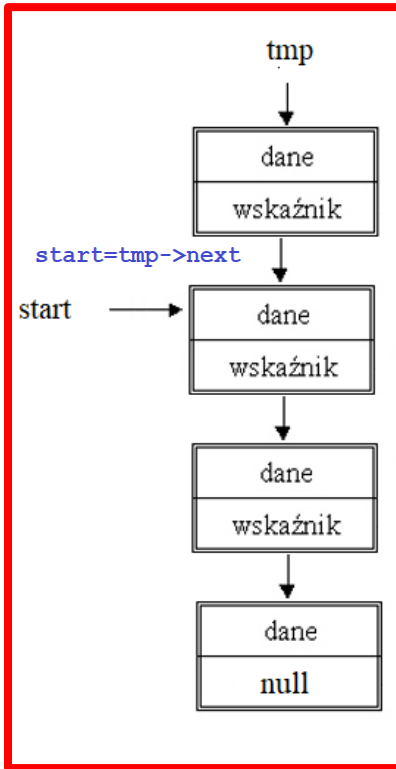
W przypadku kiedy jest to ostatni element ze stosu

Zdjęcie elementu ze stosu

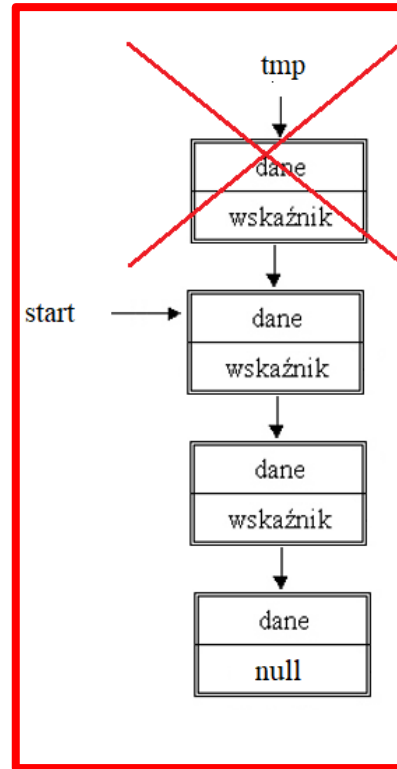
1



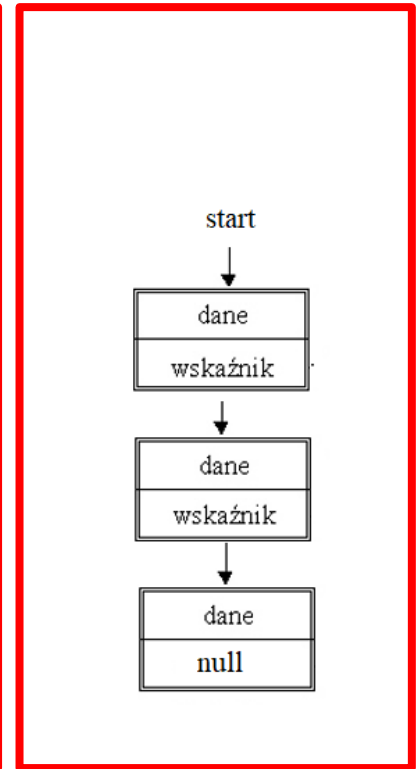
2



3



4



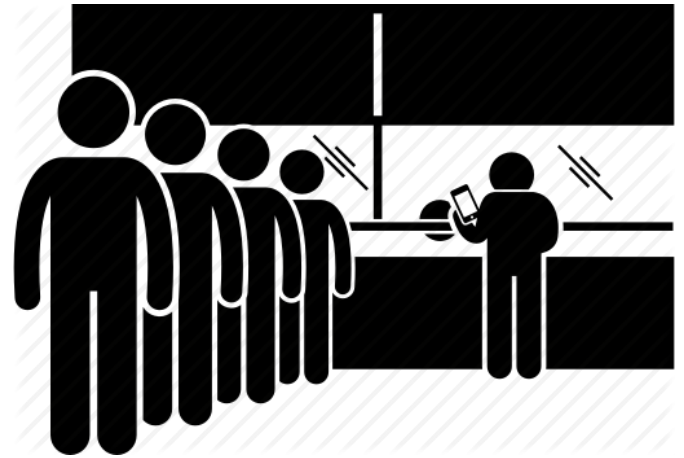
`skladnik *tmp = start` `start = start->next` `delete tmp`

Zdjęcie elementu ze stosu

```
Pop()  
{  
    if start != null  
    {  
        skladnik *tmp = start  
        start = start->next  
        delete tmp  
    }  
}
```

KOLEJKA

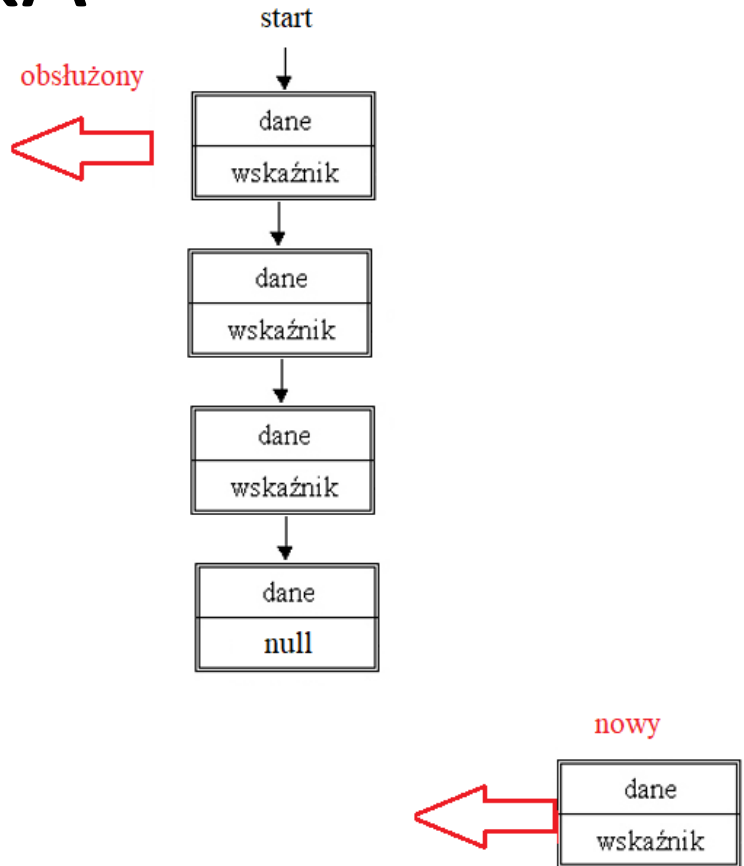
- Struktura danych, w której nowe dane dopisywane są na końcu kolejki, a do dalszego przetwarzania pobierane są z początku kolejki



First In First Out

KOLEJKA

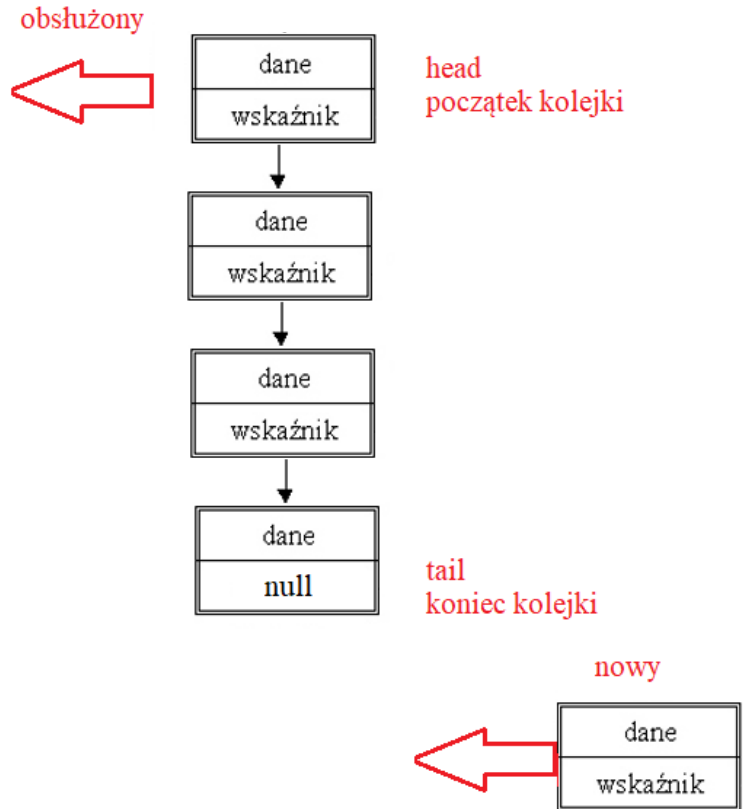
- Struktura danych, w której nowe dane dopisywane są na końcu kolejki, a do dalszego przetwarzania pobierane są z początku kolejki



First In First Out

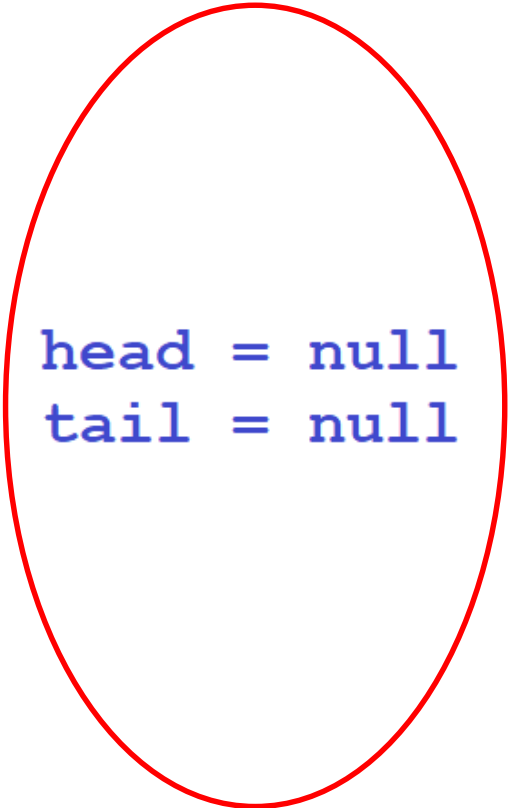
KOLEJKA

- Struktura danych, w której nowe dane dopisywane są na końcu kolejki, a do dalszego przetwarzania pobierane są z początku kolejki



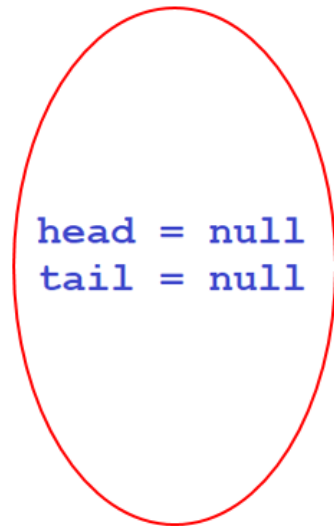
First In First Out

Pusta kolejka

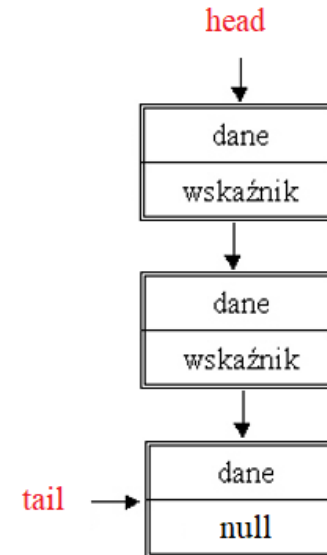


```
head = null  
tail = null
```

Dodawanie do kolejki



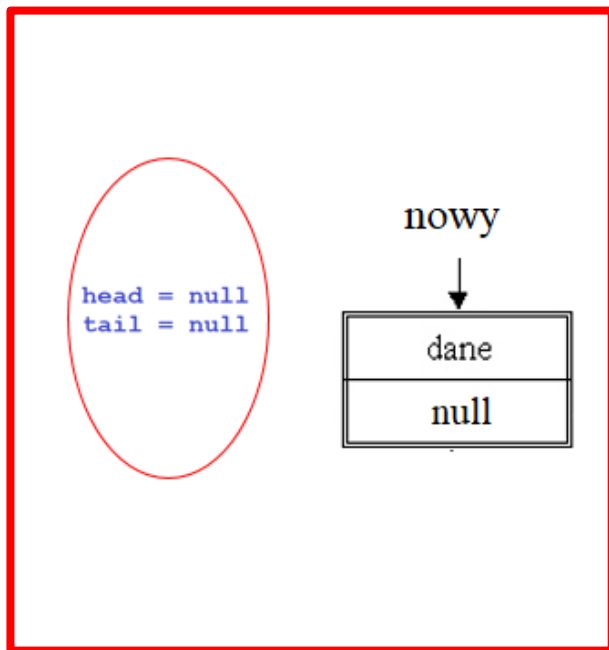
Dodawanie do **pustej** kolejki
wymaga modyfikacji
wskaźnika **head** i wskaźnika **tail**



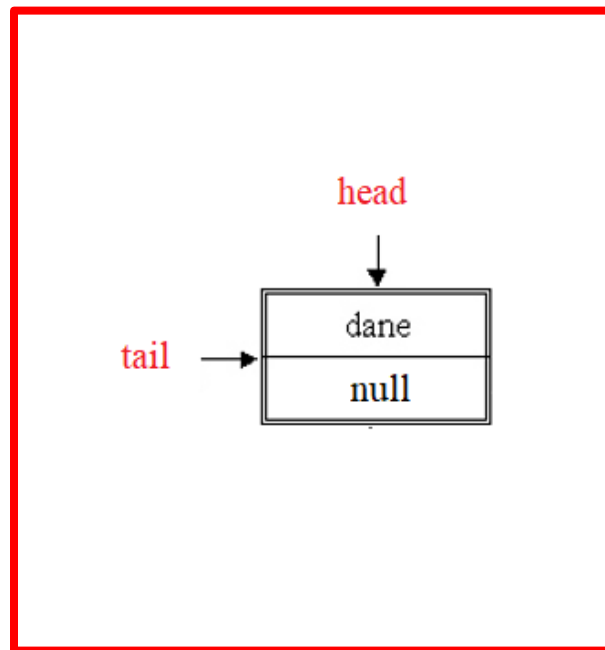
Dodawanie do **niepustej** kolejki
wymaga modyfikacji jedynie
wskaźnika **tail**

Dodawanie do pustej kolejki

1

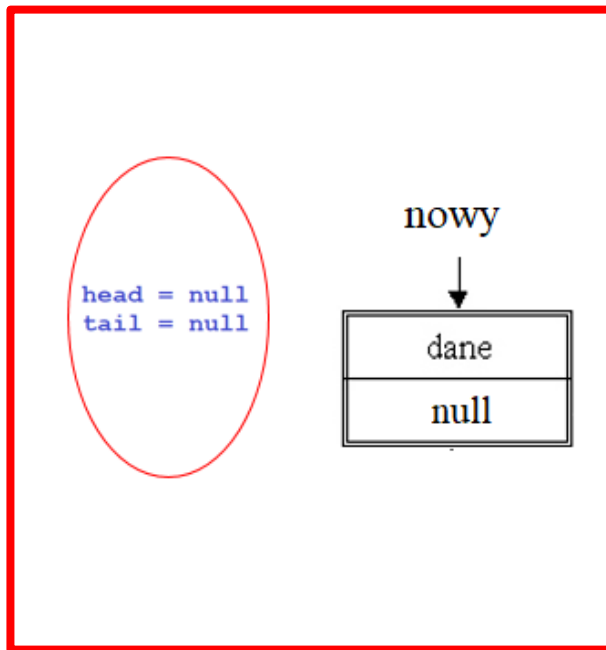


2



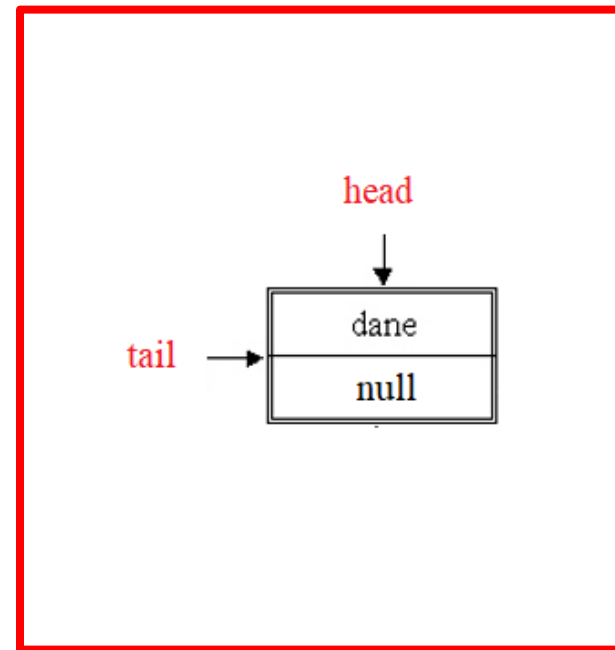
Dodawanie do pustej kolejki

1



```
skladnik *nowy = new skladnik  
nowy->val = dane  
nowy->next = null
```

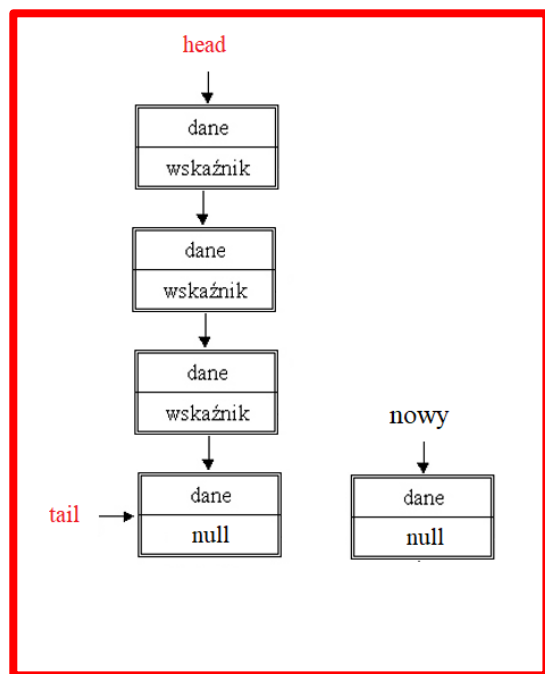
2



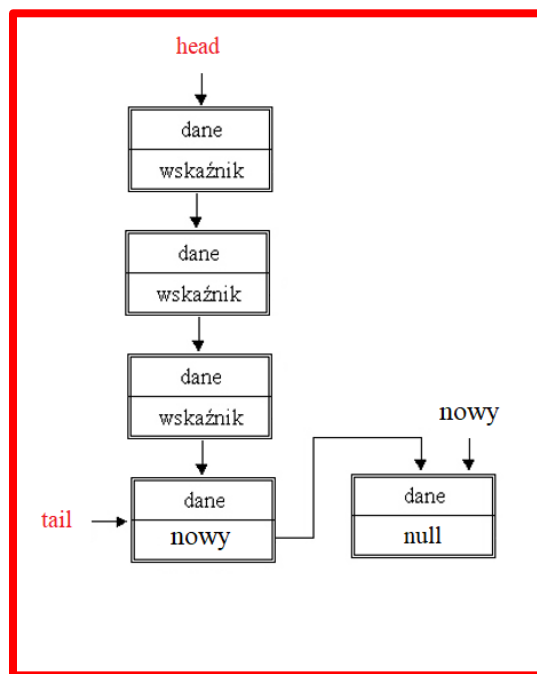
```
head = nowy  
tail = nowy
```

Dodawanie do niepustej kolejki

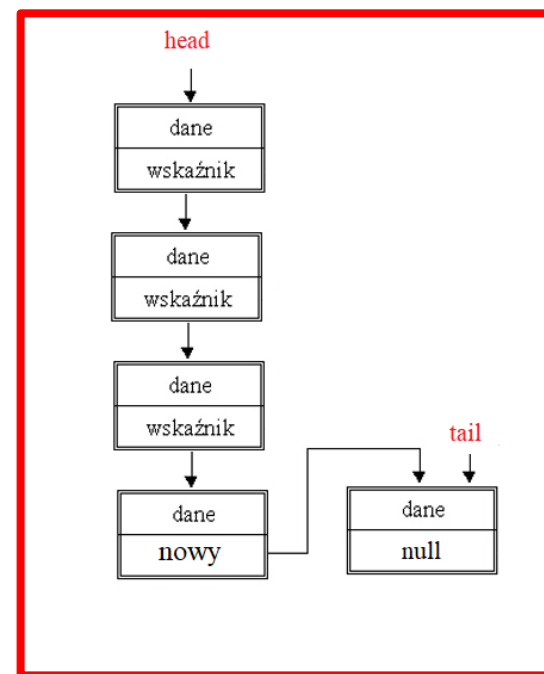
1



2



3



```
skladnik *nowy = new skladnik  
nowy->val = dane  
nowy->next = null
```

```
tail->next = nowy
```

```
tail = nowy
```

Dodawanie do kolejki

```
Push(dane)
{
    skladnik *nowy = new skladnik
    nowy->val = dane
    nowy->next = null
    if (head==null)
    {
        head = nowy
        tail = nowy
    }
    else
    {
        tail->next = nowy
        tail = nowy
    }
}
```

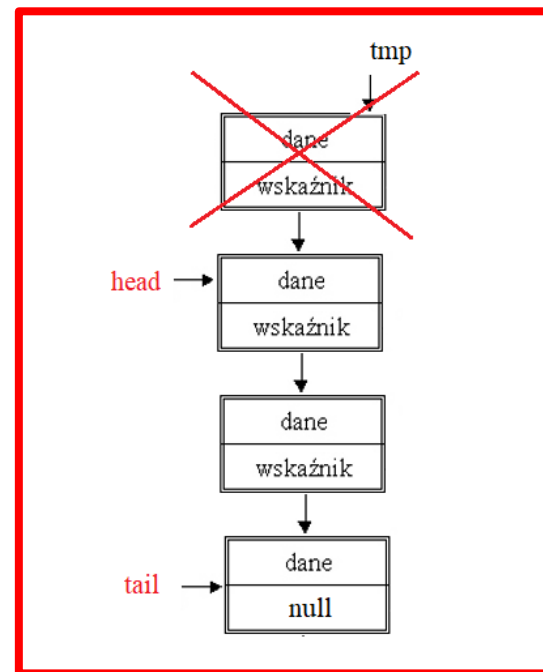
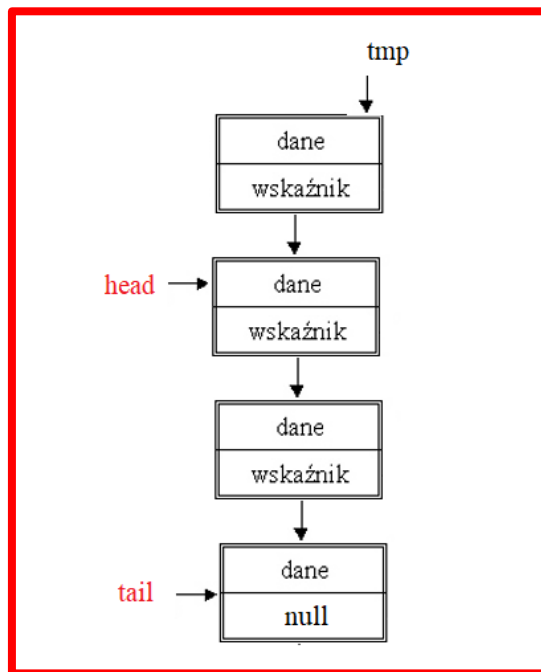
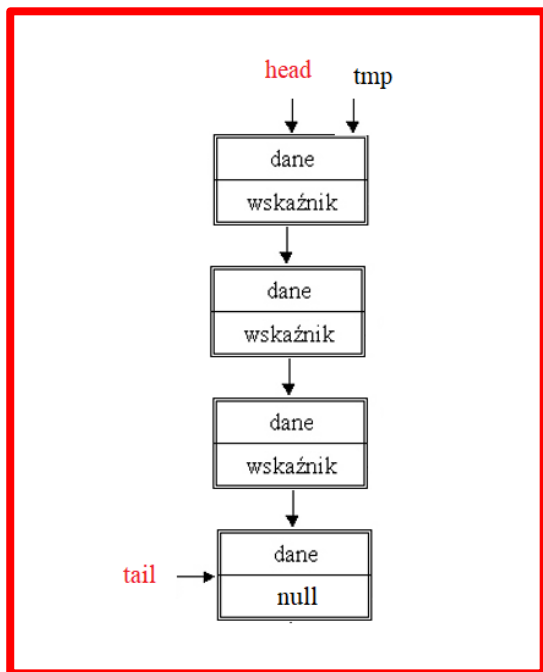
Sprawdzenie czy kolejka jest pusta

```
Function isEmpty()  
{  
    if head == null  
        return true  
    else  
        return false  
}
```


Sprawdzenie co jest na wierzchu

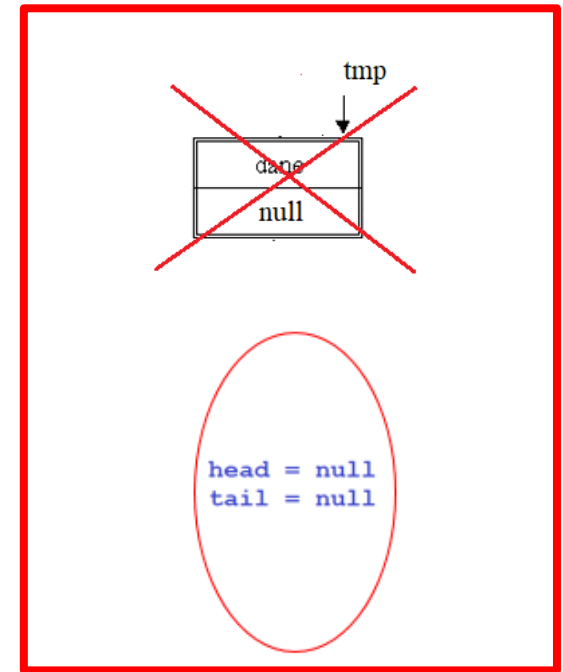
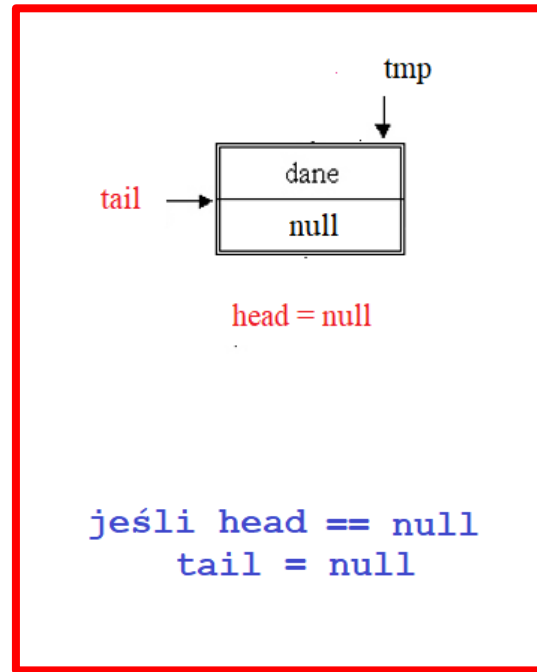
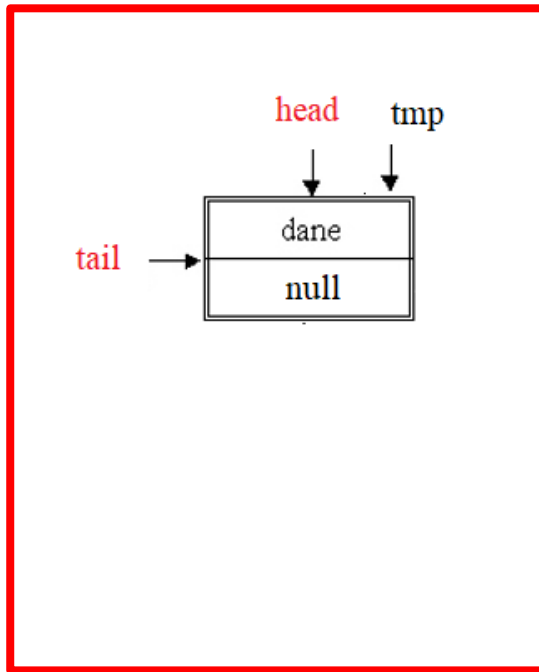
```
Function Front()  
{  
    if head != null  
        return head->val  
    else  
        return null  
}
```

Zdjęcie elementu z kolejki



`head = head->next`

Zdjęcie elementu z kolejki



`head = head->next`

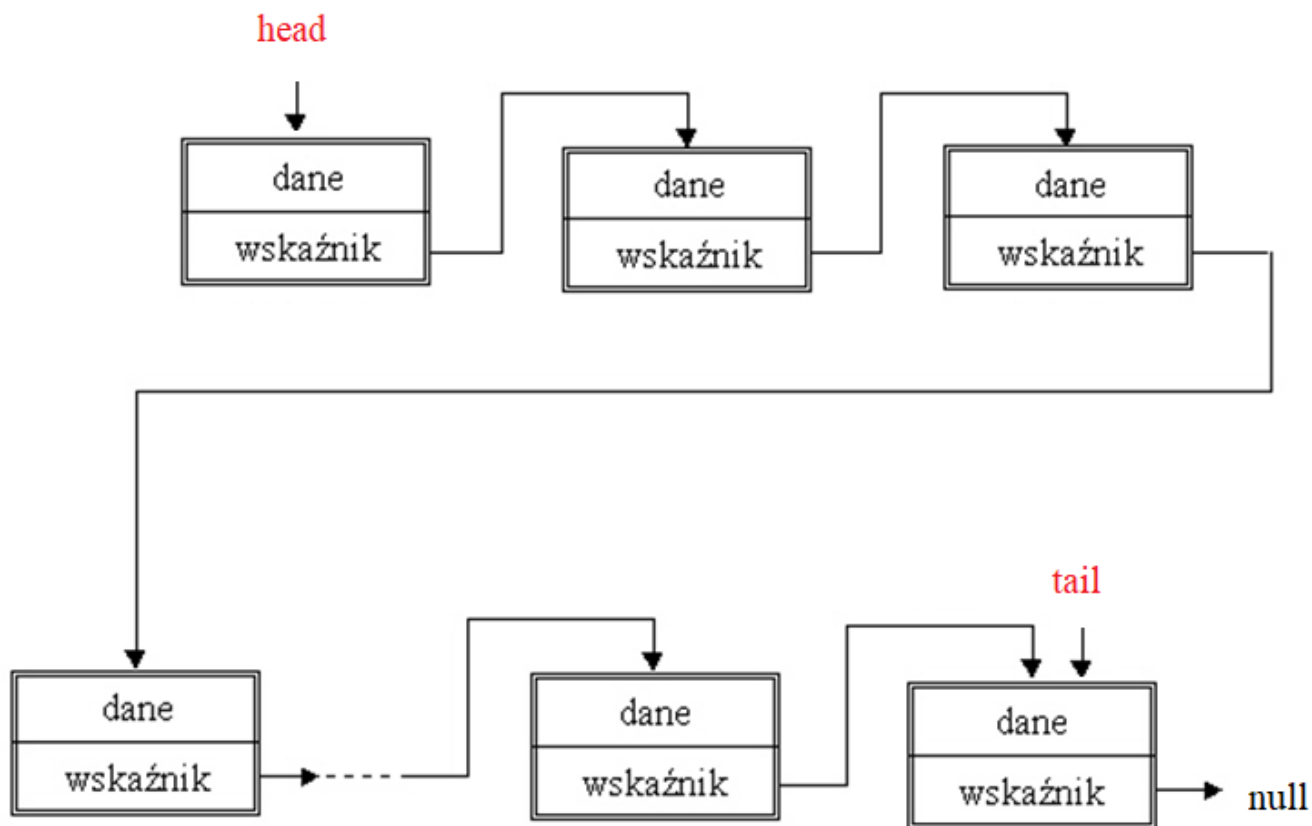
w przypadku kiedy jest to ostatni element w kolejce

Zdjęcie elementu z kolejki

```
Pop ()
{
    if (head==null)
        return

    skladnik *tmp = head
    head = head->next
    if head == null
        tail = null
    delete tmp
}
```

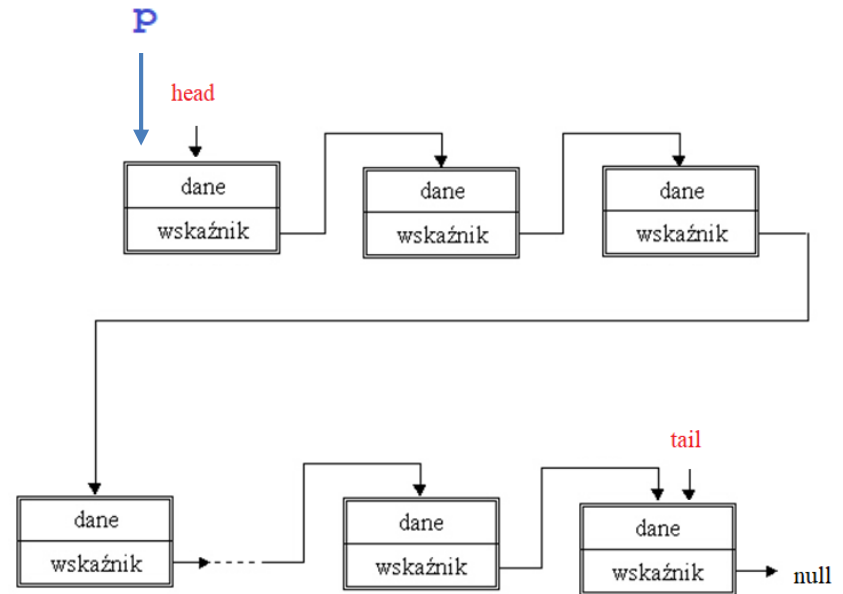
Lista jednokierunkowa



Dodaj y wszędzie za wartością x

- Jak znaleźć x?

```
skladnik *p = head
while (p != null)
{
    if (p->val == x)
    {
        p = p->next
    }
    p = p->next
}
```

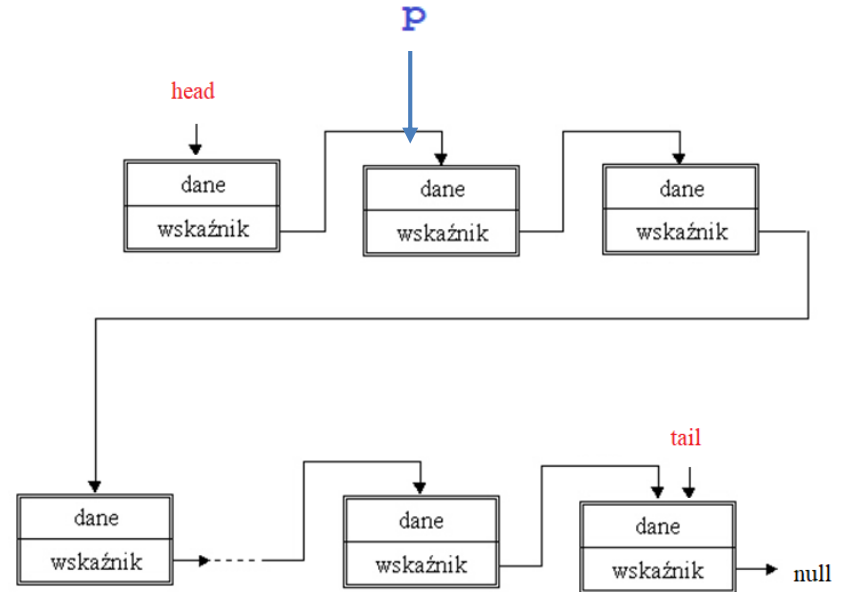


Dodaj y wszędzie za wartością x

- Jak znaleźć x?

```
skladnik *p = head
while (p != null)
{
    if (p->val == x)
    {

    }
    p = p->next
}
```

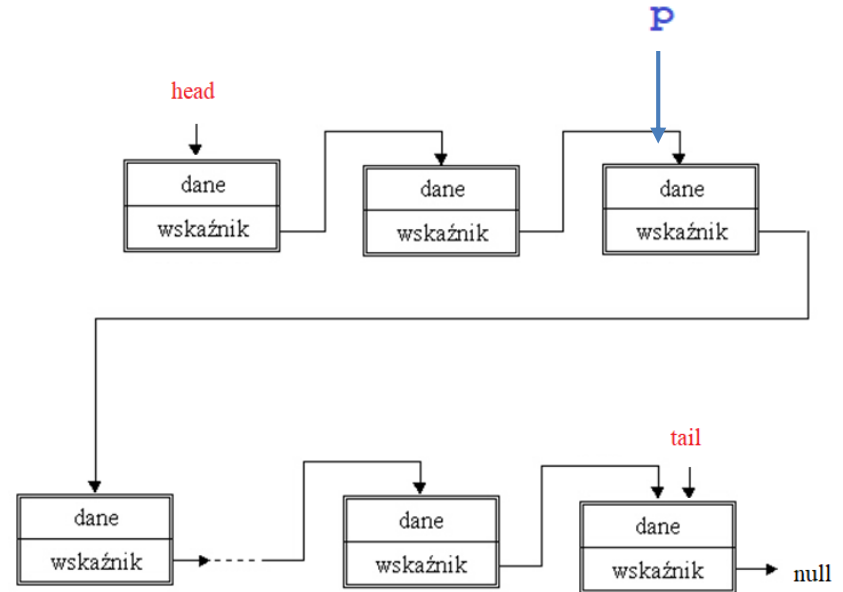


Dodaj y wszędzie za wartością x

- Jak znaleźć x?

```
skladnik *p = head
while (p != null)
{
    if (p->val == x)
    {

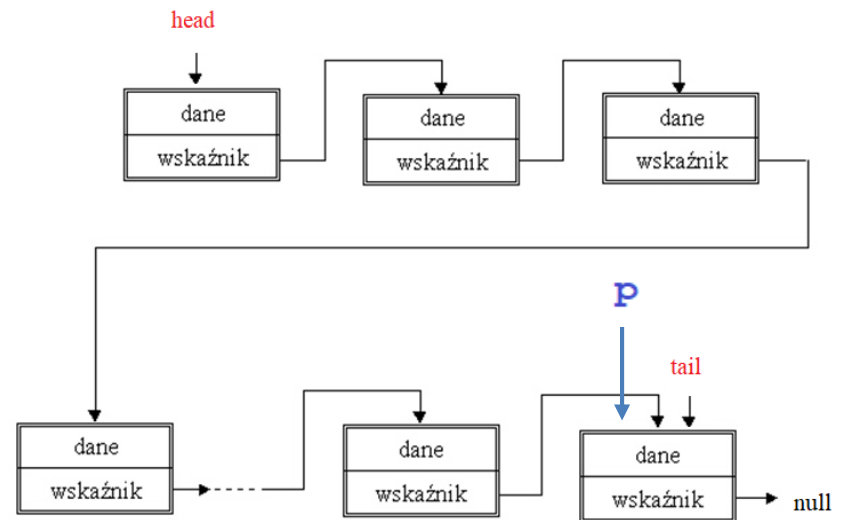
    }
    p = p->next
}
```



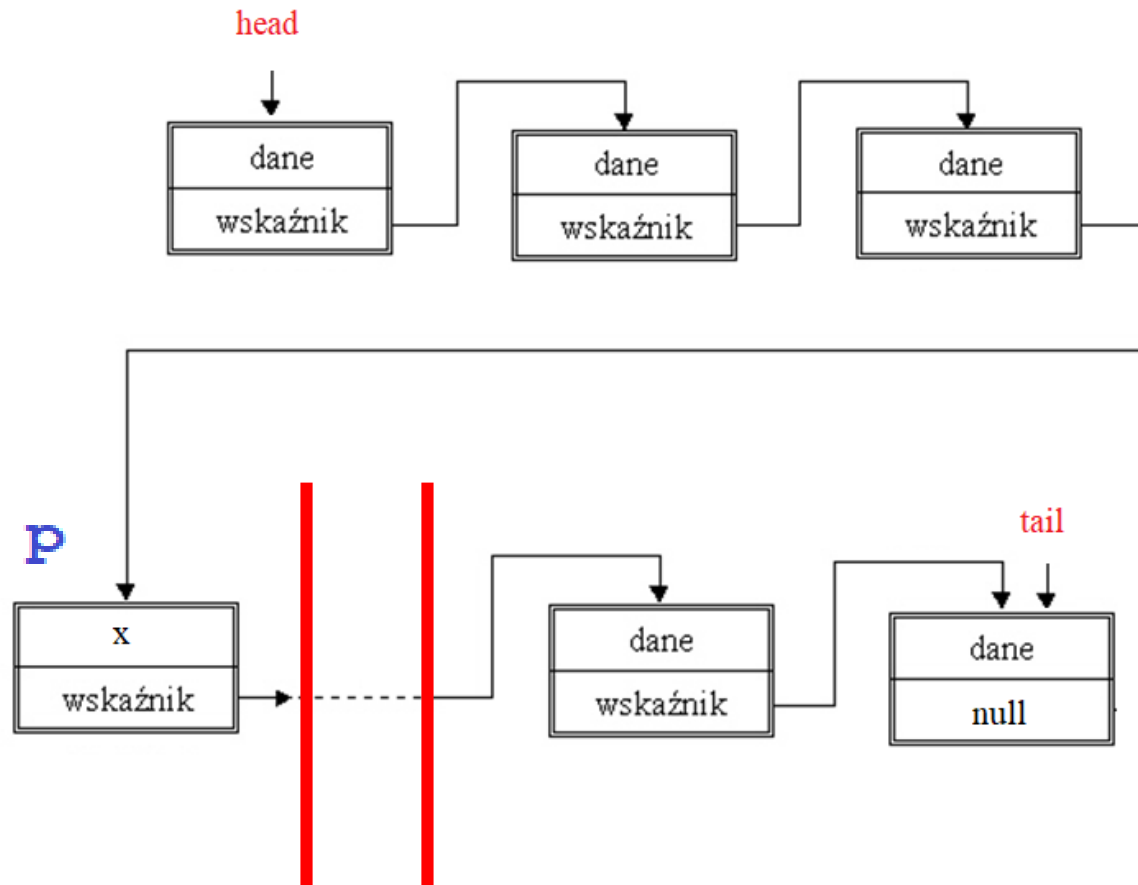
Dodaj y wszędzie za wartością x

- Jak znaleźć x?

```
skladnik *p = head
while (p != null)
{
    if (p->val == x)
    {
    }
    p = p->next
}
```

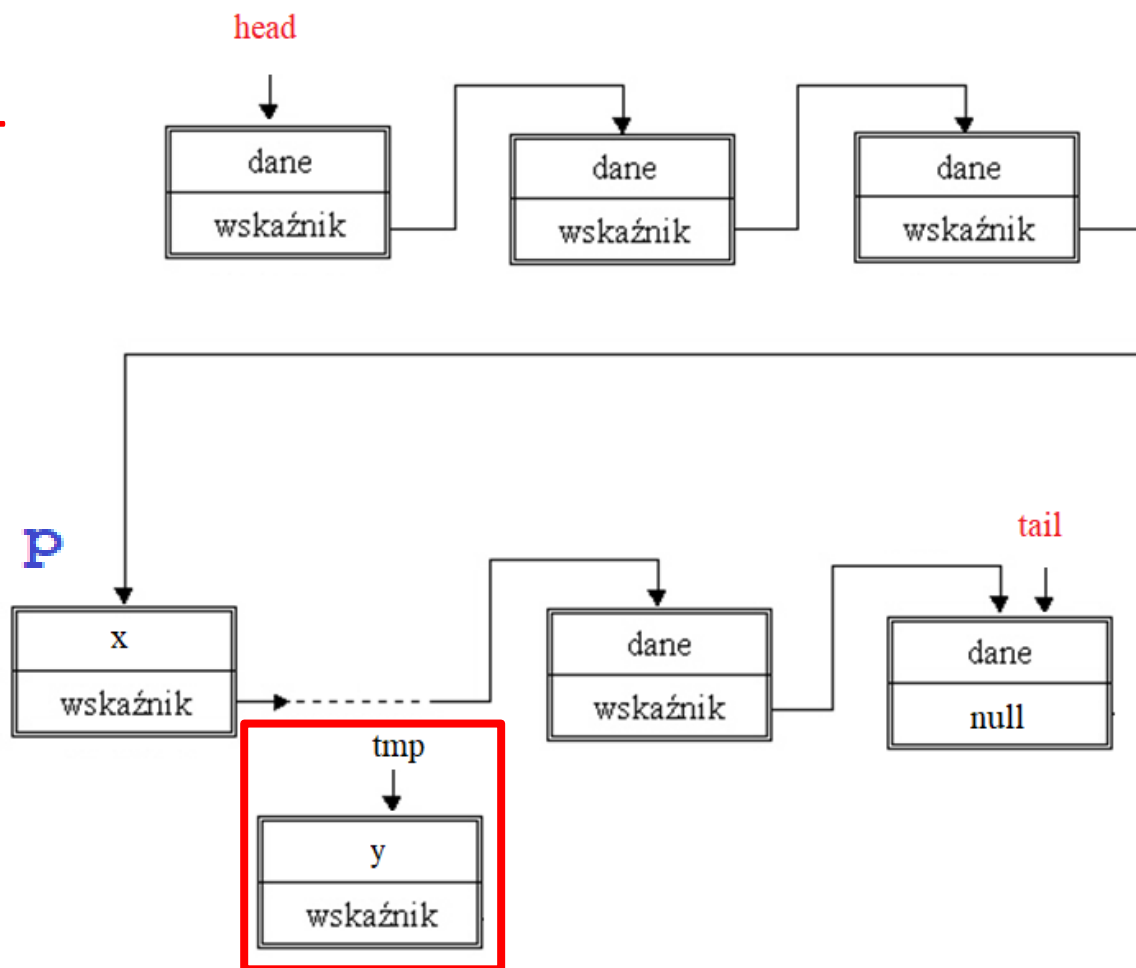


„Rozcinamy” listę wstawiając nowy



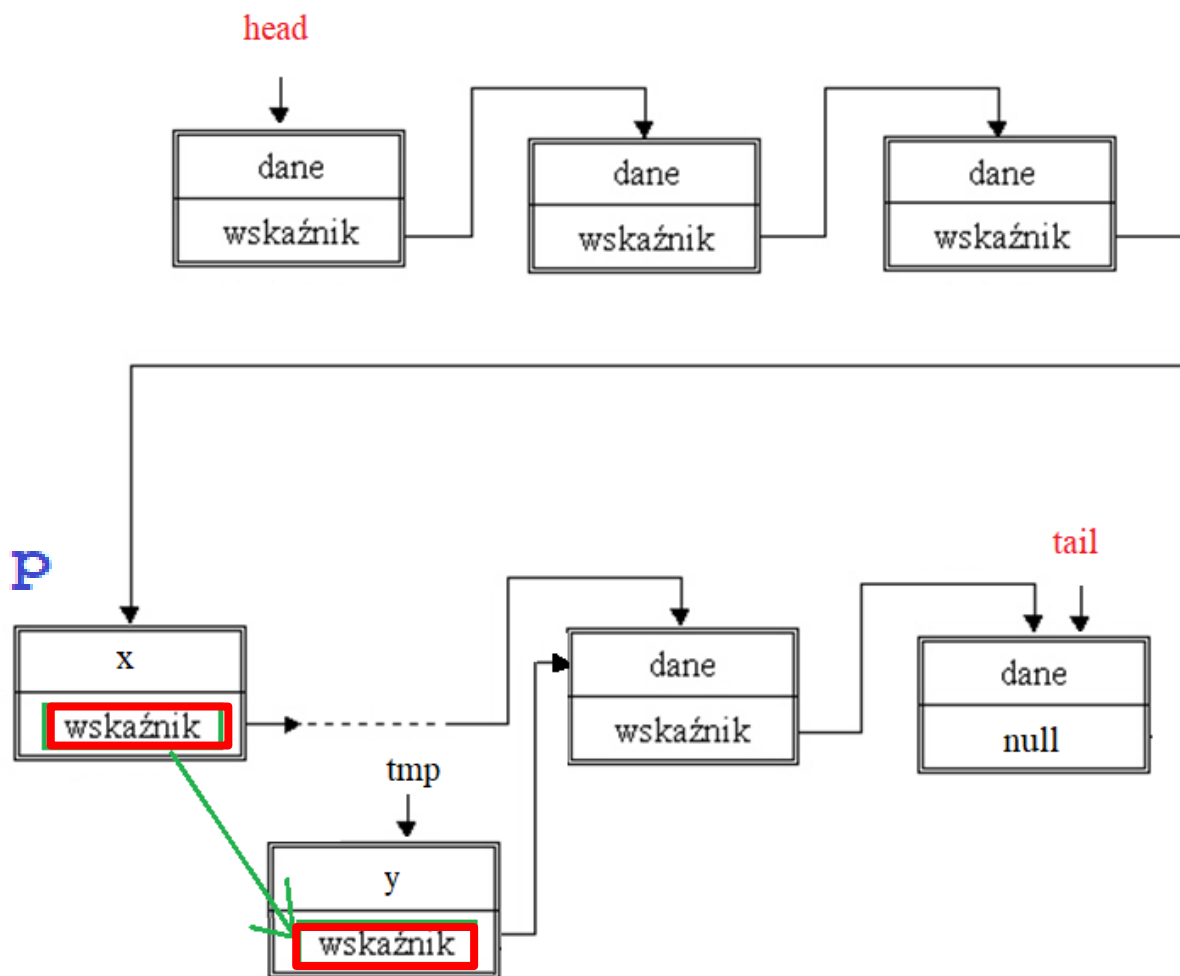
Dodaj y wszędzie za wartością x

Krok 1

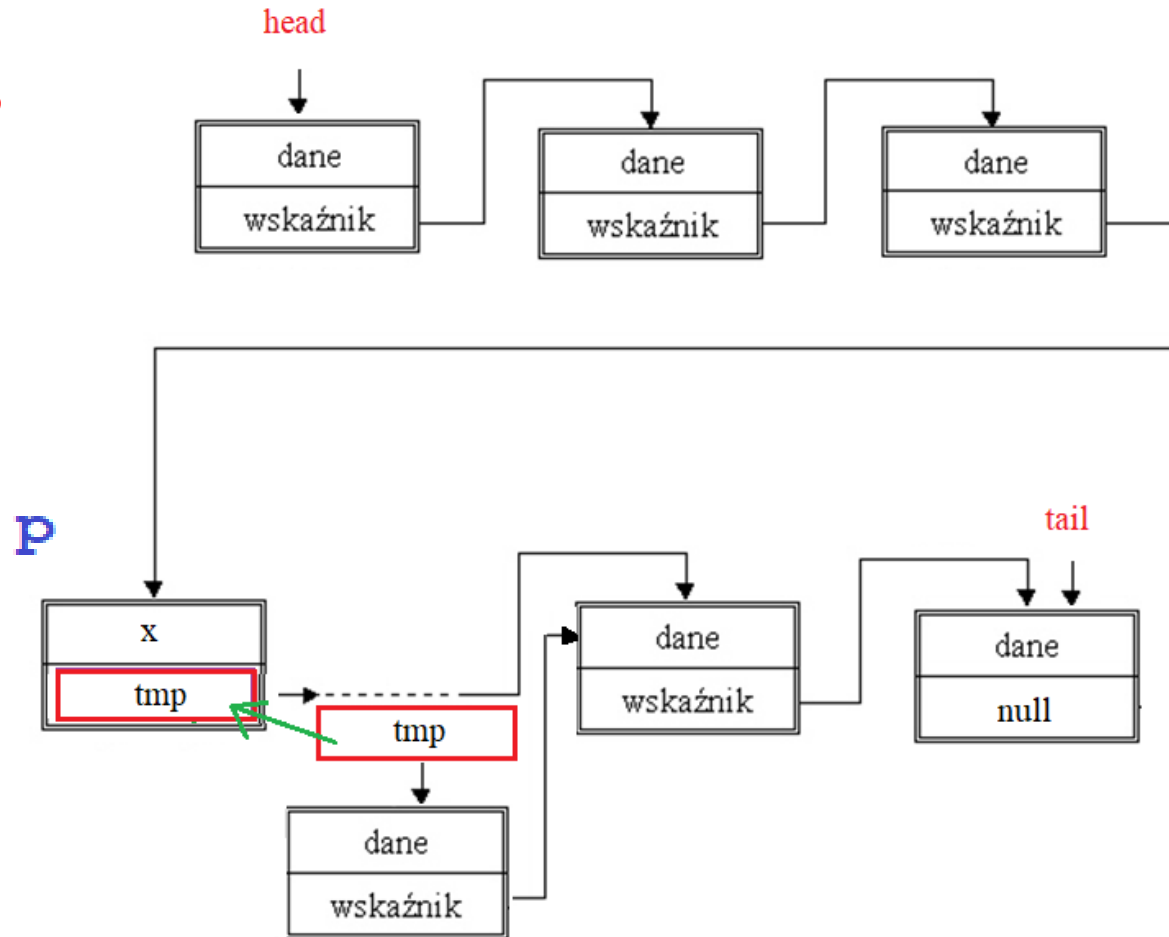


Dodaj y wszędzie za wartością x

Krok 2



Krok 3



Dodaj y wszędzie za wartością x

```
skladnik *p = head
while (p != null)
{
    if (p->val == x)
    {
        skladnik *tmp = new skladnik
        tmp->val = y
        tmp->next = p->next
        p->next = tmp
        p = p->next
    }
    p = p->next
}
```