Dynamiczne struktury danych

Kolejka
Stos

Ewa Figielska

Dynamiczne strukury danych

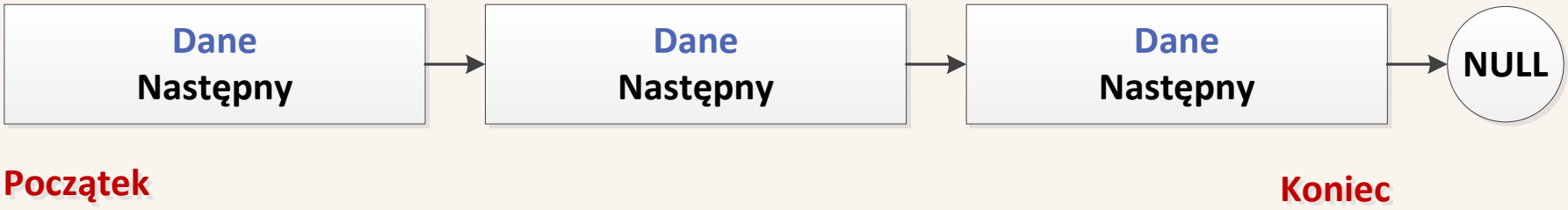
- Zawierają powiązane ze sobą elementy;
- Każdy element przechowuje dane;
- Zarządzanie pamięcią odbywa się w sposób dynamiczny:
  - nowy element jest tworzony i dołączany do struktury, wtedy gdy zachodzi potrzeba przechowania/pamiętania danych;
  - istniejący element jest usuwany ze struktury danych (i pamięci komputera), kiedy przechowywanie danych nie jest już potrzebne – magazynowane dane zostają przekazane na zewnątrz;
  - elementy przechowujące dane mogą być rozmieszczone w różnych miejscach w pamięci komputera (nie jeden za drugim) – powiązanie elementów w jedną całość jest rezalizowane przez odwołania (wskaźniki lub referencje), tzn. każdy element posiada odwołanie do innego elementu.

- Użycie struktury dynamicznej oszczędza pamięć komputera, zajęte jest tylko tyle miejsca w pamięci, ile jest potrzebne na przechowanie danych.
- Budowa i sposób obsługi różnych struktur dynamicznych mogą dobrze pasować do różnych systuacji ze świata rzeczywistego.

Kolejka

Kolejka jest to struktura danych składająca się z powiązanych ze sobą elementów służących do przechowywania danych, elementy są dodawane i usuwane do/z kolejki dynamicznie w trakcie pracy programu.

- **Nowy element jest dołączany na końcu kolejki.**
- **Element jest usuwany (wraz z pobraniem danych) z początku kolejki.**
- **Każdy element przechowuje odwołanie do elementu następnego.**

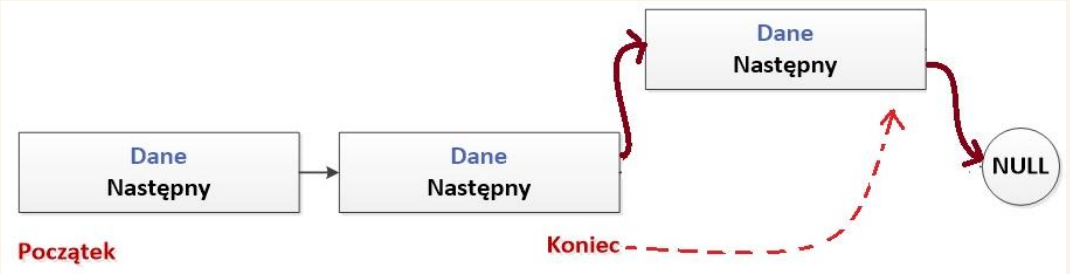


Następny - odwołanie do następnego elementu - może mieć postać wskaźnika lub referencji (w zależności od języka programowania)

Kolejka
pseudokod

```
class Element {
   Dane dane; //zmienna dane reprezentuje przechowywane dane
   Element nastepny; //odwołanie do następnego elementu
}
class Kolejka {
  Element poczatek := NULL ; //odwołanie do początku (wartość początkowa NULL)
  Element koniec := NULL;   //odwołanie do końca (wartość początkowa = NULL)

  boolean Dolacz(Dane dane) {//tworzy nowy element, wpisuje do niego dane i dołącza do kolejki
    Element nowy := new Element;
    if (nowy = NULL) return 0; //utworzenie nowego elementu nie powiodło się
    nowy.dane := dane; //wpisanie danych do nowego elementu
    nowy.nastepny := NULL; //za nowym elementem nic nie ma
    if (poczatek = NULL) poczatek := nowy; //jeżeli nowy jest pierwszsym elementem kolejki
    else  koniec.nastepny := nowy; //wstawienie nowego elementu na koniec kolejki
    koniec := nowy; //nowy element staje się końcem kolejki
    return 1;
  }
```



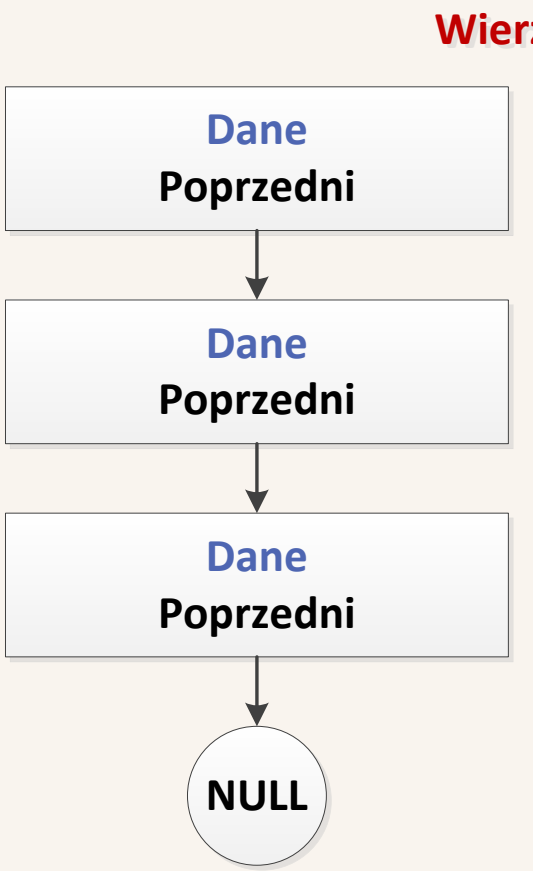

```
  boolean Usun(Dane dane) {//przekazuje dane przez argument i usuwa element je przechowujący
    if (poczatek = NULL) return 0;
    dane := poczatek.dane; //przekazanie danych na zewnątrz
    poczatek := poczatek.nastepny; //aktualizacja początku po usunięciu elementu
    if (poczatek = NULL) koniec := NULL;
    return 1;
  }
}
```

Uwaga: Implementacja w C++ będzie wymagała wstawienia znaku referencji oraz zwolnienia pamięci zajmowanej przez usuwany początek.

Stos

Stos jest to struktura danych składająca się z powiązanych ze sobą elementów służących do przechowywania danych. Elementy są kładzione na stos i pobierane ze stosu dynamicznie w trakcie pracy programu.

- **Nowy element jest kładziony na wierzchołek stosu.**
- **Element jest usuwany (wraz z pobraniem danych) z wierzchołka stosu.**
- **Każdy element posiada odwołanie do elementu poprzedniego.**



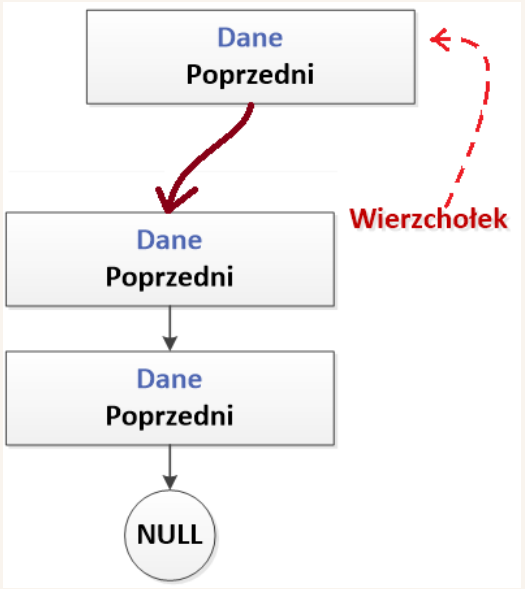
Poprzedni - odwołanie do poprzedniego elementu - może mieć postać wskaźnika lub referencji (w zależności od języka programowania)

Stos
pseudokod

```
class Element {
   Dane dane;
   Element poprzedni; //odwołanie do poprzedniego elementu
}

class Stos {
  Element wierzcholek := NULL; //odwołanie do wierzchołka stosu, wartość początkowa =  NULL

  boolean Dolacz(Dane dane) {//tworzy nowy element, wpisuje do niego dane  i kładzie go na stosie
    Element nowy := new Element; //utworzenie nowego elementu
    if (nowy = NULL) return 0; //nie udało się utworzyć nowego elementu
    nowy.dane := dane; //zapamiętanie danych w nowym elemencie
    if (wierzcholek = NULL) nowy.poprzedni := NULL; //stos był pusty
    else  nowy.poprzedni := wierzcholek; //położenie nowego elementu na stos
    wierzcholek := nowy; //nowy element staje się wierzchołkiem stosu
    return 1;
  }
```




```
  boolean Usun(Dane dane) {//przekazuje dane przez argument i usuwa element je przechowujący
    if (wierzcholek = NULL) return 0;
    dane := wierzcholek.dane; //pobranie danych
    wierzcholek := wierzcholek.poprzedni; //aktualizacja wierzchołka
    return 1;
  }
}
```

Uwaga: Implementacja w C++ będzie wymagała wstawienia znaku referencji i zwolnienia pamięci zajmowanej przez usuwany wierzchołek.

Przykład wykorzystania kolejki i stosu. Pseudokod.

```
main() {
   Dane dane;
   Kolejka kolejka := new Kolejka;
   kolejka.Dolacz(new Dane(klient3));
   kolejka.Dolacz(new Dane(klient8));
   kolejka.Dolacz(new Dane(klient5));
   while(kolejka.Usun(dane)) {
      dane.drukuj();
   } //Drukuje: klient3, klient8, klient5

   Stos stos := new Stos;
   stos.Dolacz(new Dane(dok6));
   stos.Dolacz(new Dane(dok8));
   stos.Dolacz(new Dane(dok4));
   while (stos.Usun(dane)) {
      dane.drukuj();
   } //Drukuje: dok4 dok8, dok6
}
```

```
//Zakładamy, że klasa zawierająca i
obsługująca dane posiada metodę drukuj
class Dane {
   TypDanych d;
public:
   Dane(Typdanych d) {...}
   void drukuj() {...}
}
```