Wstęp do programowania

Stosy i kolejki

Stosy

- Stosy elementów ze zbioru A, to struktury danych, które umożliwiają wykonanie następujących operacji i funkcji {var x:typA; s:stack of typA}:
  - Empty(s) czy stos s jest pusty?
  - Init(s) utwórz pusty stos s
  - Push(s,x) włóż na stos s wartość x
  - Pop(s,x) zdejmij ze stosu s element i przypisz go x
  - ==========
  - Top(s,x) podaj wartość x na szczycie stosu s
  - Full(s) czy stos s jest pełny?
  - Clear(s) wyczyść stos s

Kolejki

- Kolejki elementów ze zbioru A, to struktury danych, które umożliwiają wykonanie następujących operacji i funkcji {var x:typA; k:kolejka of typA}:
  - Pusta(k) czy kolejka k jest pusta?
  - TwórzPustą(k) utwórz pustą kolejkę k
  - Wstaw(k,x) włóż do kolejki k wartość x
  - Pobierz(k,x) usuń z kolejki k element i przypisz go x
  - ==========
  - Pierwszy(k,x) podaj wartość x na początku kolejki k
  - Pełna(k) czy kolejka k jest pełna?
  - Usuń(k) wyczyść kolejkę k

Implementacja tablicowa stosu

```
Type stackA = record
       T  :array[1..n] of typA;
       top:Integer
     end;
```

- Zakładamy, że stos ma co najwyżej n elementów typu A przechowywanych w tablicy T oraz że `top` jest indeksem pierwszego wolnego miejsca w tablicy na kolejny element.

Implementacja tablicowa stosu

```
function Empty(const S:stackA):Boolean;
begin
  Empty:= S.top=1
end;

procedure Init(var S:stackA);
begin
  S.top:=1
end;
```

Implementacja tablicowa stosu

```
procedure Push(var S:stackA; x:typA);
begin
   S.T[S.top]:=x;
   Inc(S.top)
end;

procedure Pop(var S:stackA; var x:typA);
begin
   Dec[S.top];
   x:=S.T[S.top]
end;
```

Implementacja tablicowa stosu

```
procedure Top(const S:stackA; var x:typA);
begin
  x:=S.T[S.top-1]
end;

function Full(const S:stackA):Boolean;
begin  Full:=S.top>n  end;

procedure Clear(var S:stackA);
begin  S.top:=1   end; {to samo co Init}
```

Dwa stosy w jednej tablicy

- Często stosujemy implementację dwóch stosów w jednej tablicy: można je oddzielnie przechowywać z obu końców (procedury wstawiania i usuwania są dualne). Procedura `Pełny` wymaga porównania obu wskaźników wierzchołka stosu.

Implementacja listowa stosu

```
type typA=Integer;
stackA=listaA;
listaA=^record
         w:typA;
      nast:listaA
end;
```

...czyli zwykła lista elementów typu typA.

Implementacja listowa stosu

```
function Empty(const S:stackA):Boolean;
begin
  Empty:= S=nil
end;

procedure Init(var S:stackA);
begin
  S:=nil
end;
```

Implementacja listowa stosu

```
procedure Push(var S:stackA; x:typA);
var pom:stackA;
begin
 new(pom);
 pom^.nast:=S;
 pom^.w:=x;
 S:=pom
end;
{Dodajemy nowy element na początku
  listy}
```

Implementacja listowa stosu

```
procedure Pop(var S:stackA; var x:typA);
var pom:stackA;
begin
 pom:=S;
 x:=S^.w;
 S:=S^.nast;
 dispose(pom)
end;
{Usuwamy pierwszy element listy}
```

Implementacja listowa stosu

```
procedure Top(var S:stackA; x:typA);
begin
  x:=S^.w
end;

function Full(const S:stackA):Boolean;
begin
  Full:=MaxAvail < sizeof(stackA^)
end;
```

Implementacja listowa stosu

```
procedure Clear(var S:stackA);
var dummy:typA;
 begin
   while not Empty(S) do
     Pop(S,dummy)
 end;
```

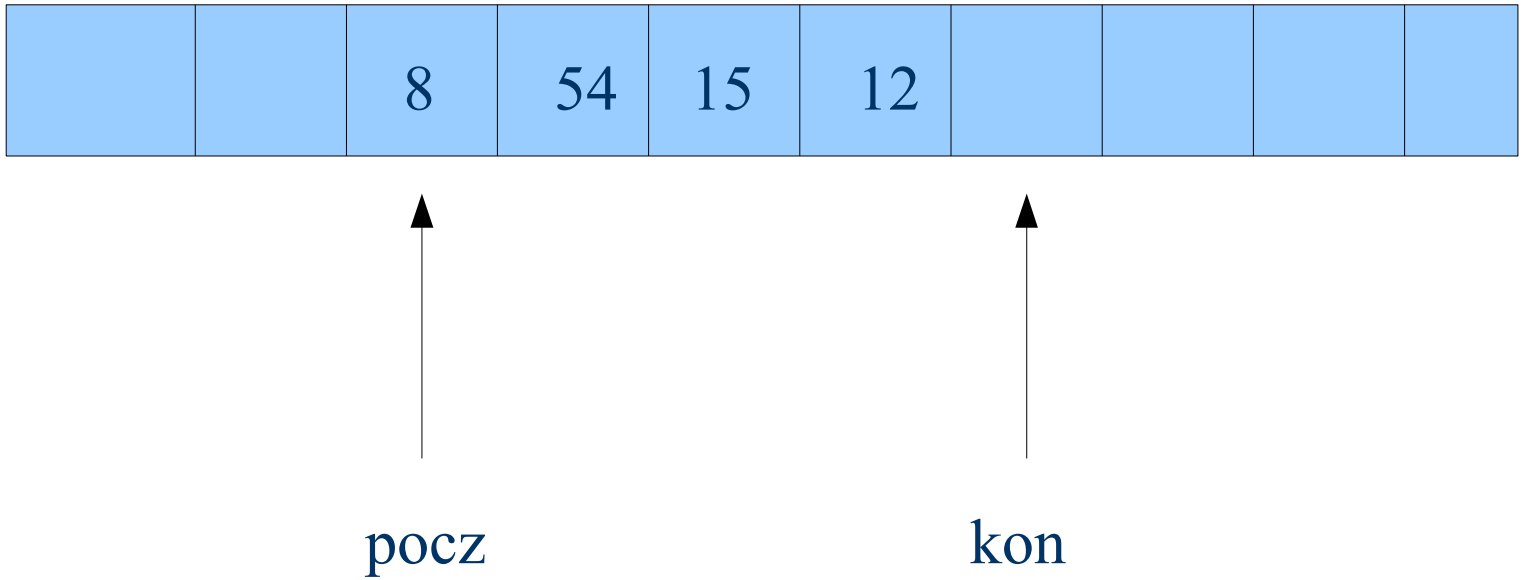
Implementacja tablicowa kolejki

```
Type kolejkaA = record
      T  :array[0..n] of typA;
      pocz,kon:Integer
     end;
```

- Zakładamy, że kolejka ma co najwyżej n elementów typu A przechowywanych w tablicy T. Indeksy pocz i kon odpowiadają odpowiednio pierwszemu elementowi w kolejce oraz pierwszemu wolnemu miejscu w tablicy na kolejny element.

Implementacja tablicowa kolejki

- Kolejka w tablicy



Implementacja tablicowa kolejki

- Użyteczna funkcja:

```
function Next(j:Integer):Integer;
begin
 if j=n then Next:=0 else
             Next:=j+1
end;

{lub j:=j mod n + 1, co jest może i
  zgrabniejsze, ale droższe niż „if”}
```

Implementacja tablicowa kolejki

```
function Pusta(const k:kolejkaA):Boolean;
begin
   Empty:= k.pocz=k.kon
end;

procedure TwórzPustą(var k:kolejkaA);
begin
   k.pocz:=0; k.kon:=0;
end;
```

Implementacja tablicowa kolejki

```
procedure Wstaw(var k:kolejkaA; x:typA);
begin
   k.T[k.kon]:=x;
   k.kon:=Next(k.kon)
end;

procedure Pobierz(var k:kolejkaA; var
                                  x:typA);
begin
   x:=k.T[k.pocz];
   k.pocz:=Next(k.pocz)
end;
```

Implementacja tablicowa kolejki

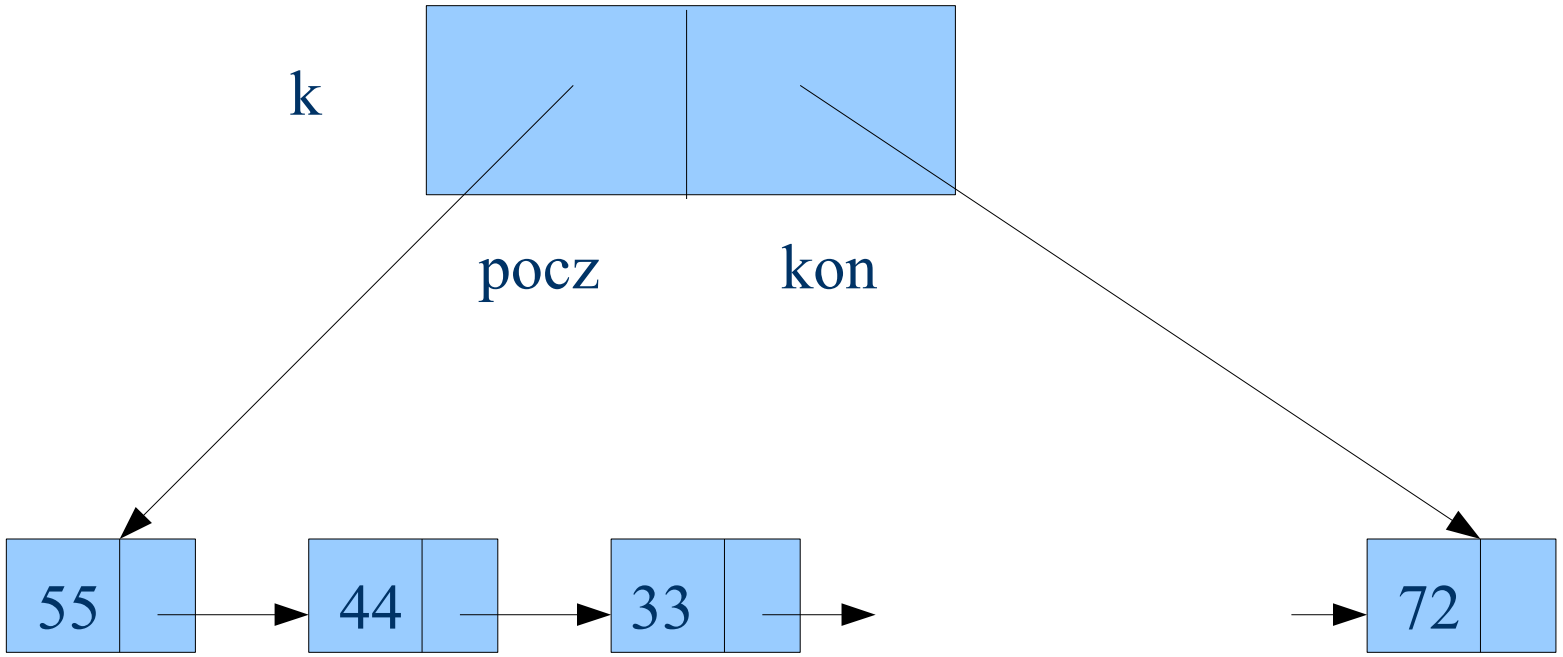
```
procedure Pierwszy(const k:kolejkaA;
                                  var x:typA);
begin
  x:=k.T[k.pocz]
end;

function Pełna(const k:kolejkaA):Boolean;
begin  Pełna := k.pocz=Next(k.kon)  end;

procedure Usuń(var k:kolejkaA);
begin  k.kon:=k.pocz end;
```

Implementacja listowa kolejki

```
type kolejkaA=record pocz,kon:listaA;
```



Implementacja listowa kolejki

```
function Pusta(const k:kolejkaA):Boolean;
begin
   Empty:= k.pocz=nil
end;

procedure TwórzPustą(var k:kolejkaA);
begin
   k.pocz:=nil;
end;
```

Implementacja listowa kolejki

```
procedure Wstaw(var k:kolejkaA; x:typA);
begin
  if k.pocz=nil then begin
    new(k.pocz);
    k.pocz^.w:=x;
    k.kon:=k.pocz
  end
  else begin
    WstawZa(x,k.kon);
    k.kon:=k.kon^.nast
  end
end;
```

Implementacja listowa kolejki

```
procedure Pobierz(var k:kolejkaA;
                                  var x:typA);


var pom:listaA;
begin
  x:=k.pocz^.w;
  pom:=k.pocz;
  if k.pocz=k.kon then k.pocz:=nil
                  else k.pocz:=k.pocz^.nast;
  dispose(pom)
end;
```

Implementacja listowa kolejki

```
procedure Pierwszy(const k:kolejkaA;
                                  var x:typA);
begin
   x:=k.pocz^.w
end;

function Pełna(const k:kolejkaA):Boolean;
begin
 Pełna := MaxAvail < `sizeof(listaA^)
end;
```

Implementacja listowa kolejki

```
procedure Clear(var k:kolejkaA);
var dowol:typA;
begin
 while not Pusta(k) do Pobierz(k,dowol)
end;
```

Implementacja listowa kolejki

- Można kolejkę ładnie zaimplementować w liście cyklicznej. Wtedy wskaźnik do listy jest ostatnim elementem kolejki. Wstawiamy za niego, modyfikując go, a pierwszy element jest zawsze następnikiem tego wskaźnika.