Wykład 6_2

Abstrakcyjne typy danych – kolejki. Implementacja za pomocą tablicy i rekurencyjnej struktury danych czyli listy wiązanej

Etap 1 - Opis ADT

Nazwa typu: Kolejka elementów
Własności typu: Potrafi przechować ciąg elementów o dowolnym rozmiarze
Dostępne działania: Inicjalizacja kolejki
Określenie, czy kolejka jest pusta
Dodanie elementu do kolejki,
Usuwanie z kolejki,

```
typedef  int  dane;                    //typ informacji umieszczanej w kolejce

typedef struct ELEMENT* stos;          //typ wskazania na element kolejki
struct ELEMENT                         //typ  elementu kolejki
   { dane Dane;
     stos Nastepny;
   };
struct kolejka                         //typ przechowujący dane kolejki
 { stos Poczatek, Koniec;
};
```

Etap 2 - Budowa interfejsu

```
void Inicjalizacja(kolejka& Kolejka);
```
{ *działanie:* inicjuje kolejkę
warunki wstępne: Kolejka.Poczatek i *Kolejka.Koniec* wskazują na pustą kolejkę
warunki końcowe: kolejka zostaje zainicjowana jako pusta}

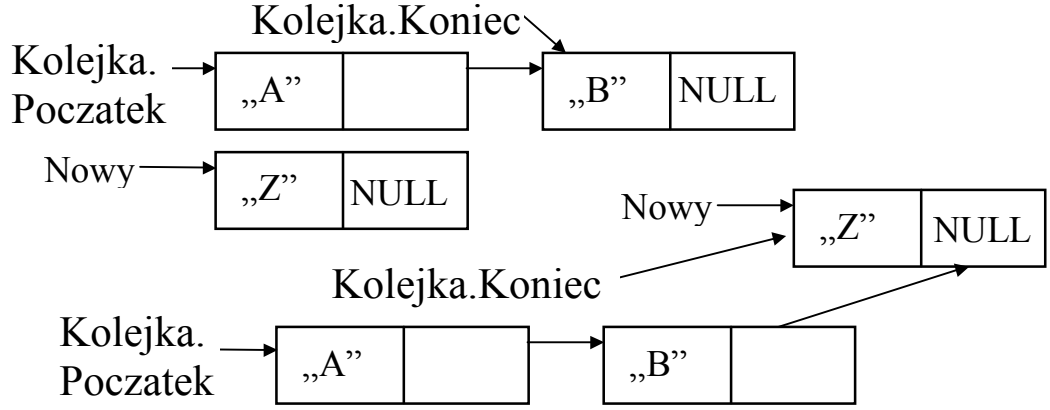
```
inline int Pusty(kolejka& Kolejka);
```
{*działanie*: określa, czy kolejka jest pusta; typ **inline**, bo często wywoływana
warunki wstępne: *Kolejka* jest zainicjowana,
warunki końcowe: funkcja zwraca 1, jeśli kolejka jest pusta, jeśli nie- 0}

```
int Wstaw(kolejka& Kolejka, dane Dana);
```
{ *działanie:* dodaje element na koniec ciągu, zwany końcem kolejki
warunki początkowe: Dana jest daną do wstawienia na koniec zainicjowanej kolejki wskazanym przez *Kolejka.Koniec*
warunki końcowe: jeśli jest to możliwe, funkcja dodaje daną *Dana* na koniec kolejki i zwraca 1, w przeciwnym wypadku 0 }

```
int Usun(kolejka& Kolejka);
```
{ *działanie:* usuwa element wstawiony do kolejki na początku ciągu,
warunki początkowe: Kolejka.Poczatek jest niepustą kolejką
warunki końcowe: usuwa element na poczatku kolejki i zwraca dane przez **return**. *Kolejka.Koniec* jest równy *Kolejka.Poczatek*, gdy kolejka jest pusta }

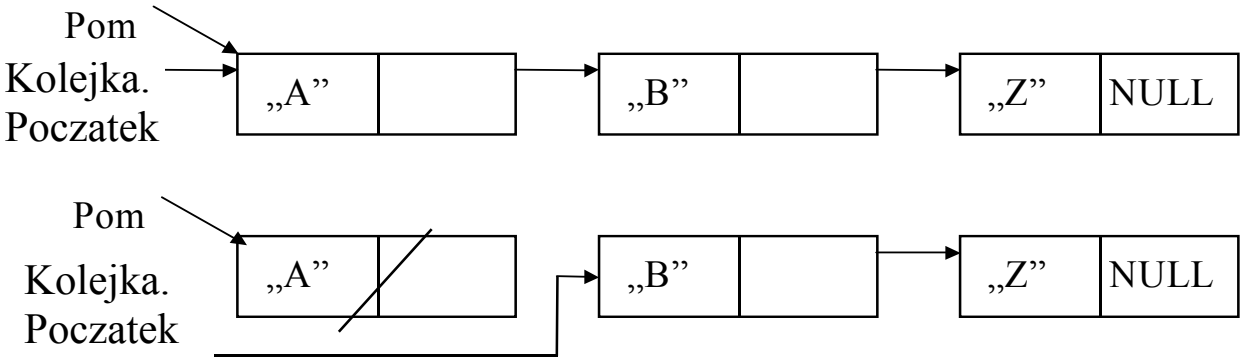
Etap 3. Implementacja kolejki jako listy wiązanej

- wstawianie elementów zawsze na końcu struktury




```
int Wstaw(kolejka& Kolejka, dane Dana)
 { stos Nowy;
   Nowy = new ELEMENT;
   if (Nowy!=NULL)   Nowy->Dane=Dana;
   else              return 0;
   if (Kolejka.Poczatek ==NULL)  Kolejka.Poczatek=Nowy;
   else                          Kolejka.Koniec->Nastepny= Nowy;
   Kolejka.Koniec= Nowy;
   Kolejka.Koniec->Nastepny=NULL;
   return 1;
  }
```

- usuwanie elementów zawsze na początku struktury. W przypadku, gdy po usunięciu kolejka staje się pusta – *Kolejka.*Koniec jest równy *Kolejka.*Poczatek




```
dane Usun(kolejka& Kolejka)
  {stos Pom;
   dane d;
   Pom = Kolejka.Poczatek;
   Kolejka.Poczatek = Kolejka.Poczatek->Nastepny;
   d= Pom->Dane;                                    //((*Pom).Dane)
   delete Pom;
   if (Kolejka.Poczatek==NULL) Kolejka.Koniec=NULL;
   return d; }
```

```
#include <conio.h>
#include <stdlib.h>
#include <string.h>
#include <stdio.h>

        //1. interfejs ADT kolejki
typedef int dane;                    // dane umieszczone w kolejce
typedef struct ELEMENT* stos;   //nazwa wskaznika na element stosu
struct ELEMENT
 {   dane Dane;
     stos Nastepny;
 };
struct kolejka
 {    stos Poczatek, Koniec;
 };
                //funkcje ADT kolejki
void Inicjalizacja(kolejka& Kolejka);
inline int Pusty(kolejka Kolejka);
int Wstaw(kolejka& Kolejka, dane Dana);
dane Usun(kolejka& Kolejka);

       //2. funkcje we/wy dla danych umieszczonych w kolejce
void Pokaz_dane (dane Dana);
dane Dane();

       //3. funkcje ogólnego przeznaczenia
 void Komunikat(char*);
 char Menu(const int ile, char *Polecenia[]);

     //4. elementy programu
const int Esc=27;
const int POZ=4;
char * Tab_menu[POZ] = {  "1 : Wstawianie do kolejki- na koniec",
                          "2 : Usuwanie z kolejki-na poczatku",
                          "3 : Wydruk kolejki wraz z jej usuwaniem",
                          " >Esc Koniec programu"};

            //funkcje klienta korzystajace z kolejki
void Wstaw_do_kolejki(kolejka& Kolejka);
void Wyswietl_usun_z_kolejki(kolejka& Kolejka);
```

```
void main(void)
 { kolejka Kolejka;
   char Wybor;

   clrscr();
   Inicjalizacja(Kolejka);
   do
   { Wybor= Menu(POZ, Tab_menu);
      switch (Wybor)
         { case '1' :  Wstaw_do_kolejki(Kolejka);
                       break;
           case '2' :  if (Pusty(Kolejka))
                              Komunikat("\nKolejka pusty\n");
                       else (Usun(Kolejka));
                       break;
           case '3' :  if (Pusty(Kolejka))
                              Komunikat("\nKolejka jest pusta\n") ;
                       else Wyswietl_usun_z_kolejki(Kolejka);
                       break;
         }
   } while (Wybor !=Esc );
  }

//**************funkcje klienta korzystające ze stosu***********
 void Wstaw_do_kolejki(kolejka& Kolejka)
  {  dane Dana;
     Dana= Dane();
     if (Wstaw(Kolejka, Dana)==0)
       Komunikat("Brak pamieci\n");
  }

 void Wyswietl_usun_z_kolejki(kolejka& Kolejka)
    { dane d;
      while (!Pusty(Kolejka))
        {  d=Usun(Kolejka);
           Pokaz_dane(d);
        }
    }
```

```
//*********funkcje interfejsu ADT kolejki***********

 void Inicjalizacja(kolejka& Kolejka)
    { Kolejka.Poczatek=Kolejka.Koniec = NULL;
    }

 inline int Pusty(kolejka Kolejka)
    {  return Kolejka.Poczatek==NULL;
    }

 int Wstaw(kolejka& Kolejka, dane Dana)
  {
    stos Nowy;

    Nowy = new ELEMENT;
    if (Nowy!=NULL)
      Nowy->Dane=Dana;
    else return 0;
    if (Kolejka.Poczatek ==NULL)     Kolejka.Poczatek=Nowy;
    else                             Kolejka.Koniec->Nastepny= Nowy;
                                     Kolejka.Koniec= Nowy;
    Kolejka.Koniec->Nastepny=NULL;
    return 1;
   }

 dane Usun(kolejka& Kolejka)
   {
    stos Pom;
    dane d;

    Pom = Kolejka.Poczatek;
    Kolejka.Poczatek = Kolejka.Poczatek->Nastepny;
    d= Pom->Dane;                                    //((*Pom).Dane)
    delete Pom;
    if (Kolejka.Poczatek==NULL) Kolejka.Koniec=NULL;
    return d;
   }
```

```
//*******funkcje we/wy dla danych umieszczonych w kolejce*********
 dane Dane()
   { int a;
      do
       { fflush(stdin);
         printf("\n\nPodaj dane typu int: ");
       } while (scanf("%d",&a)!=1);
     return a;
    }

  void Pokaz_dane(dane Dana)
    {   printf("\nNumer: %d\n", Dana);
        printf("Nacisnij dowolny klawisz...\n");
        getch();
    }

//*********funkcje ogolnego przeznaczenia************
char Menu(const int ile, char *Polecenia[])
    {  clrscr();
       for (int i=0; i<ile;i++)
             printf("\n%s",Polecenia[i]);
       return getch();
    }

  void Komunikat(char* s)
     {   printf(s);
         getch();
     }
```

Etap 3. Implementacja stosu za pomocą tablicy

zainicjowanie kolejki: N=8 Koniec=0, Poczatek=N
dodanie: tab[Koniec], Koniec=(Koniec+1) **mod** N
usuwanie: Poczatek=Poczatek **mod** N **, return** tab[Poczatek], Poczatek++

A) Kolejka pusta zainicjowana: N=8 ↓Koniec=0, ↓Poczatek=N

↓Koniec=0, ↓Poczatek=N

0	1	2	3	4	5	6	7

B) Kolejka zawiera trzy elementy: ↓Koniec=3, ↓Poczatek=0 (dodano: 0(a, 1(b), 2(c)))

↓Poczatek ↓Koniec

0	1	2	3	4	5	6	7
a	b	c					

C) Kolejka zawiera trzy elementy: ↓Poczatek=3, ↓Koniec=5 (usunięto: 0(a), 1(b), 2(c) i dodano: 3(d), 4(e)))

↓Poczatek ↓Koniec

0	1	2	3	4	5	6	7
			d	e			

D) Kolejka zawiera pięć elementów: ↓Koniec=2, ↓Poczatek=5 (usunięto 3(d), 4(e)) i wstawiono: 5(a), 6(b), 7(c), 0(d), 1(e) **)**

↓Koniec ↓Poczatek

0	1	2	3	4	5	6	7
d	e				a	b	c

E) **Kolejka pełna: ↓Koniec=5, ↓Poczatek=5** (po wstawieniu na koniec następujących elementów: 2(f), 3(g), 4(h))

↓Poczatek
↓Koniec

0	1	2	3	4	5	6	7
d	e	f	g	h	a	b	c

F) **K**olejka pusta, stan nieokreślony: ↓Koniec=5, ↓Poczatek=5 (po usunięciu elementów: 5(a), 6(b), 7(c), 0(d), 1(e), 2(f), 3(g), 4(h)). Należy więc po usunięciu ostatniego elementu zainicjować kolejkę tak, aby Poczatek =N, Koniec=0, czyli wprowadzić kolejkę **w stan A**)

↓Poczatek
↓Koniec

0	1	2	3	4	5	6	7

```
#include <conio.h>
#include <stdlib.h>
#include <string.h>
#include <stdio.h>

        //1. interfejs ADT kolejki
 typedef int dane;                  // dane umieszczone stosie
 const long N=5;
 struct kolejka                     //elementy typu kolejki
  { int Poczatek;
    int Koniec;
    dane tab[N];
  };
        //funkcje interfejsu mają ten sam nagłówek jak dla listy wiązanej
 void Inicjalizacja(kolejka& Kolejka);
 inline int Pusty(kolejka Kolejka);
 int Wstaw(kolejka& Kolejka, dane Dana);
 dane Usun(kolejka& Kolejka);

//Funkcje we/wy, ogólnego przeznaczenia, klienckie oraz program są
//identyczne dla obu implementacji

        //2. funkcje we/wy dla danych umieszczonych w kolejce
 void Pokaz_dane (dane Dana);
 dane Dane();

       //3. funkcje ogolnego przeznaczenia
 void Komunikat(char*);
 char Menu(const int ile, char *Polecenia[]);

      //4. elementy programu
const int Esc=27;
const int POZ=4;
 char * Tab_menu[POZ] = { "1 : Wstawianie do kolejki- na koniec",
                          "2 : Usuwanie z kolejki-na poczatku",
                        "3 : Wydruk kolejki wraz z jej usuwaniem",
                          " >Esc Koniec programu"};
             //funkcje klienta korzystajace z kolejki
void Wstaw_do_kolejki(kolejka& Kolejka);
void Wyswietl_usun_z_kolejki(kolejka& Kolejka);
```

```
void main(void)
 { kolejka Kolejka;
   char Wybor;

   clrscr();
   Inicjalizacja( Kolejka);
   do
    {   Wybor= Menu(POZ, Tab_menu);
        switch (Wybor)
           { case '1' :   Wstaw_do_kolejki(Kolejka); break;
             case '2' :   if (Pusty(Kolejka))    Komunikat("\nStos pusty\n");
                          else (Usun(Kolejka));
                             break;
             case '3' :   if (Pusty(Kolejka)) Komunikat("\nStos pusty\n") ;
                          else Wyswietl_usun_z_kolejki(Kolejka);
                          break;
    }
   } while (Wybor !=Esc );
  }
//*********funkcje interfejsu ADT kolejki***********
 void Inicjalizacja(kolejka& Kolejka)
 {  Kolejka.Poczatek = N;
    Kolejka.Koniec=0; }

 inline int Pusty(kolejka Kolejka)
 { return Kolejka.Poczatek==N && Kolejka.Koniec==0; }

 int Wstaw(kolejka& Kolejka, dane Dana)
  { if (Kolejka.Poczatek != Kolejka.Koniec)
         {  Kolejka.tab[Kolejka.Koniec++] = Dana;
            Kolejka.Koniec %= N;
            if  (Kolejka.Poczatek==N)   Kolejka.Poczatek=0;
            return 1; }
    else   return 0;
  }

 dane Usun(kolejka& Kolejka)
   {  dane d;
      Kolejka.Poczatek %= N;
      d= Kolejka.tab[Kolejka.Poczatek++];
      if (Kolejka.Poczatek%N==Kolejka.Koniec)
              Inicjalizacja(Kolejka);
    return d; }
```