Wykład 7

Abstrakcyjne typy danych – kolejka priorytetowa

Definicja kolejki priorytetowej:

Kolejka priorytetowa to struktura danych zawierająca elementy z kluczami, która pozwala na przeprowadzanie dwóch podstawowych operacji: wstawiania nowego elementu i usuwania elementu o największej wartości (R.Sedgewick,Algorytmy w C++)

Zastosowania kolejki priorytetowej (wg R.Sedgewick,Algorytmy w C++)

- Systemy symulacyjne, w których dane kolejki mogą odpowiadać czasom wystąpienia zdarzeń przeznaczonych do chronologicznego przetwarzania
- Planowanie zadań w systemach komputerowych – dane kolejki mogą oznaczać priorytety wskazujące, którzy użytkownicy mają być obsługiwania w pierwszej kolejności
- Obliczenia numeryczne, gdzie klucze mogą być wartościami błędów obliczeniowych, oznaczającymi, że największy powinien zostać obsłużony jako pierwszy

Kolejka priorytetowa

Etap 1 - Opis ADT

Nazwa typu: Kolejka elementów

Własności typu: Potrafi przechować ciąg elementów – usuwa zawsze największy element

Dostępne działania: Inicjalizacja kolejki priorytetowej
Określenie, czy kolejka priorytetowa jest pusta
Dodanie elementu do kolejki priorytetowej,
Usuwanie z kolejki priorytetowej największego elementu

Etap 2 - Budowa interfejsu

void Inicjalizacja(kolejka_p& Kolejka_P);
{ *działanie:* inicjuje kolejkę priorytetową
warunki wstępne: Kolejka_P jest pustą kolejką priorytetową
warunki końcowe: kolejka priorytetowa zostaje zainicjowana jako pusta}

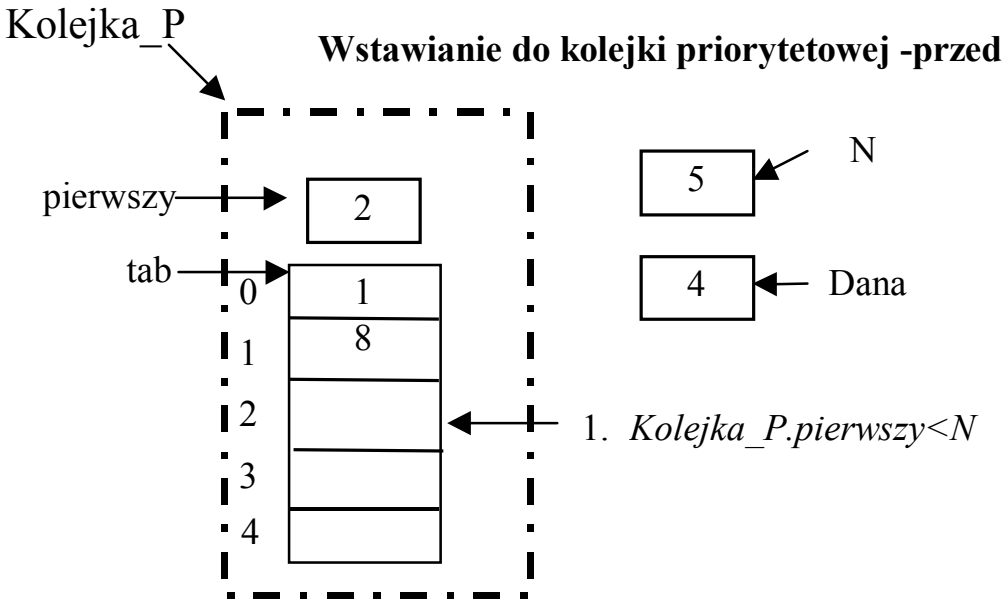
inline int Pusty(kolejka_p Kolejka_P);
{*działanie*: określa, czy kolejka priorytetowa jest pusta; typ **inline**, bo często wywoływana
warunki wstępne: *Kolejka_P* jest zainicjowana,
warunki końcowe: funkcja zwraca 1, jeśli kolejka priorytetowa jest pusta, jeśli nie- 0}

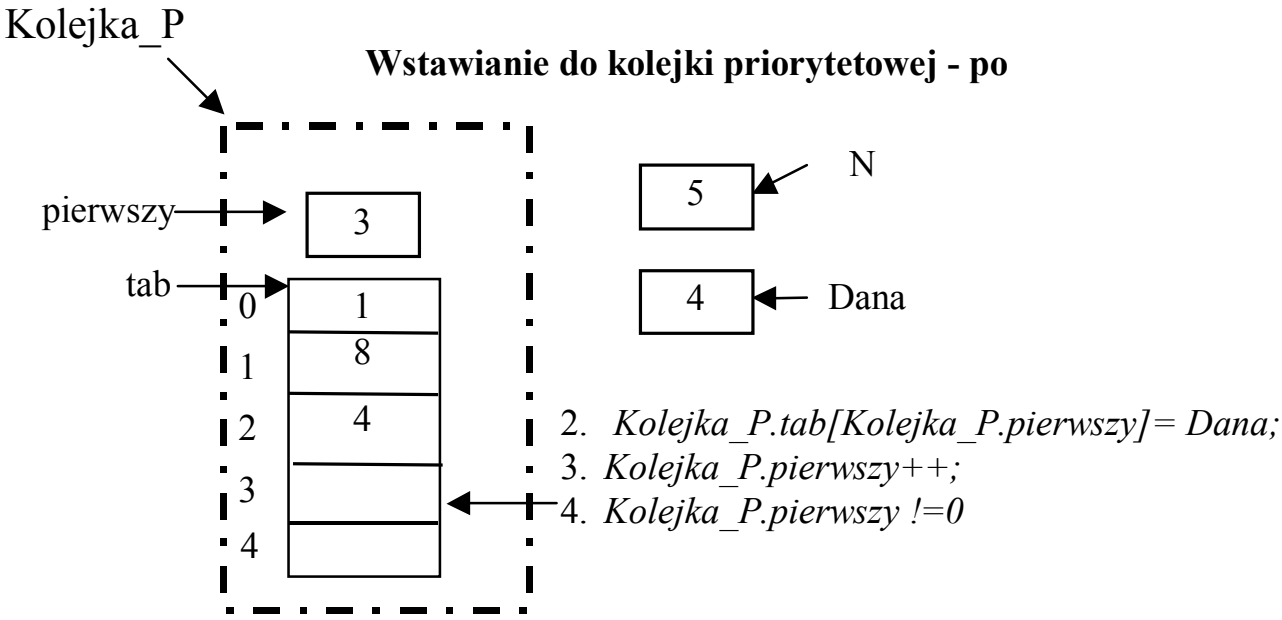
int Wstaw(kolejka_p& Kolejka_P, dane Dana);
{ *działanie:* dodaje element w dowolny sposób do kolejki priorytetowej
warunki początkowe: Dana jest daną do wstawienia do zainicjowanej kolejki priorytetową *Kolejka_P*
warunki końcowe: jeśli jest to możliwe, funkcja dodaje daną *Dana* do kolejki priorytetową i zwraca 1, w przeciwnym wypadku 0 }

int Usun(kolejka_p& Kolejka_P);
{ *działanie:* usuwa największy element wstawiony do kolejki priorytetowej,
warunki początkowe: Kolejka_P jest niepustą kolejką priorytetową
warunki końcowe: usuwa element największy z kolejki priorytetowej i zwraca dane przez **return.** Po usuwaniu kolejka może być pusta i musi być zainicjowana}

Etap 3 - Implementacja za pomocą tablicy nieposortowanej

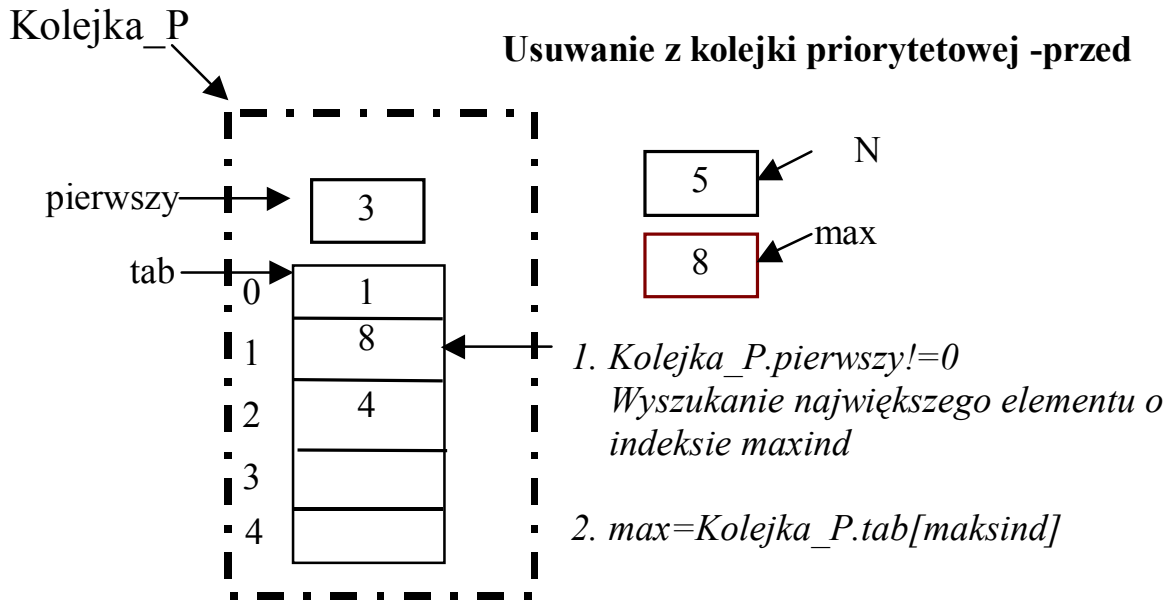
- wstawianie

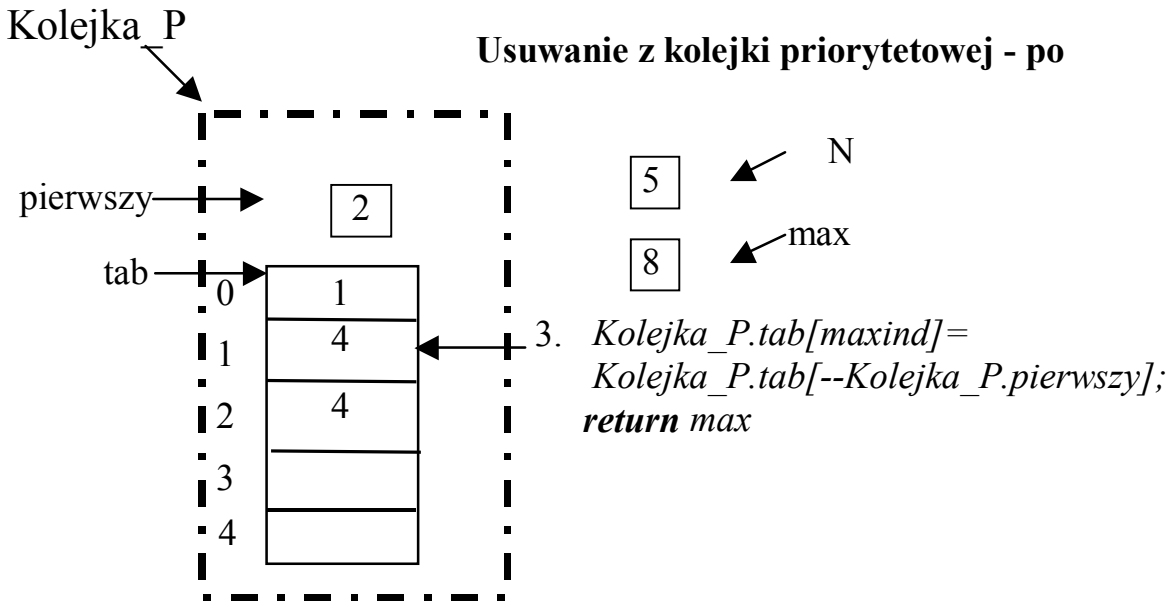





```
int Wstaw_p(kolejka_p& Kolejka_P, dane Dana)
 { if (Kolejka_P.pierwszy==N) return 0;        //nie można wstawić do stosu, gdy jest pełen
   Kolejka_P.tab[Kolejka_P.pierwszy++]= Dana;
   return 1;
  }
```

- Usuwanie z kolejki priorytetowej






```
dane Usun_max(kolejka_p& Kolejka_P)
 { int maxind=0;
   for (int i=1; i< Kolejka_P.pierwszy; i++)    //wyszukanie największego elementu
     if (Kolejka_P.tab[i] > Kolejka_P.tab[maxind])
          maxind = i;
   dane max = Kolejka_P.tab[maxind];           //pobranie największego elementu
                          //zapisanie w miejscu pobranego elementu ostatniego z ciągu
   Kolejka_P.tab[maxind] = Kolejka_P.tab[--Kolejka_P.pierwszy];
   return max;                                 //zwrócenie największego elementu
  }
```

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

//1. interfejs ADT kolejki priorytetowej
 typedef int dane;                    // dane umieszczone w kolejce priorytetowej
 const long N=11;
 struct stos
  { int pierwszy;
    dane tab[N];
  };
 typedef stos kolejka_p;              //nowa nazwa wynikająca z przedefiniowanego
                                      //zastosowania danych i niektorych funkcji stosu
        //prototypy funkcji kolejki_prorytetowej
 void Inicjalizacja(kolejka_p& Kolejka_P);
 inline int Pusty(kolejka_p Kolejka_P);
 int Wstaw_p(kolejka_p& Kolejka_P, dane Dana);
 dane Usun_max(kolejka_p& Kolejka_P);

 //2. funkcje we/wy dla danych umieszczonych w kolejce p.

 void Pokaz_dane (dane Dana);
 dane Dane(char* menu);

 //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_p",
                "2 : Usuwanie z kolejki_p najwiekszego elementu",
                  "3 : Wydruk kolejki_p nierosnaco wraz z jej usuwaniem",
                  " >Esc Koniec programu"};
//funkcje klienta korzystajace z kolejki priorytetowej
void Wstaw_do_kol_p(kolejka_p& Kolejka_P);
void Wyswietl_usun_z_kol_p(kolejka_p& Kolejka_P);
```

```
void main(void)
 { kolejka_p Kolejka_P;
   char Wybor;

   clrscr();
   Inicjalizacja(Kolejka_P);
   do
   { Wybor= Menu(POZ, Tab_menu);
    switch (Wybor)
     { case '1' : Wstaw_do_kol_p(Kolejka_P);
                  break;
       case '2' : if (Pusty(Kolejka_P))
                    Komunikat("\nKolejka_p pusta\n");
                  else (Usun_max(Kolejka_P));
                  break;
       case '3' : if (Pusty(Kolejka_P))
                    Komunikat("\nKolejka_p pusta\n") ;
                  else Wyswietl_usun_z_kol_p(Kolejka_P);
                  break;
      }
   } while (Wybor !=Esc );
  }

//***funkcje klienta korzystające z kolejki priorytetowej**
 void Wstaw_do_kol_p(kolejka_p& Kolejka_P)
  { dane Dana= Dane("Podaj dane do kolejki p: ");
     if (Wstaw_p(Kolejka_P, Dana)==0)
         Komunikat("Brak pamieci\n");}

 void Wyswietl_usun_z_kol_p(kolejka_p& Kolejka_P)
    {dane d;
     while (!Pusty(Kolejka_P))
      { d=Usun_max(Kolejka_P);
        Pokaz_dane(d);
      }}
```

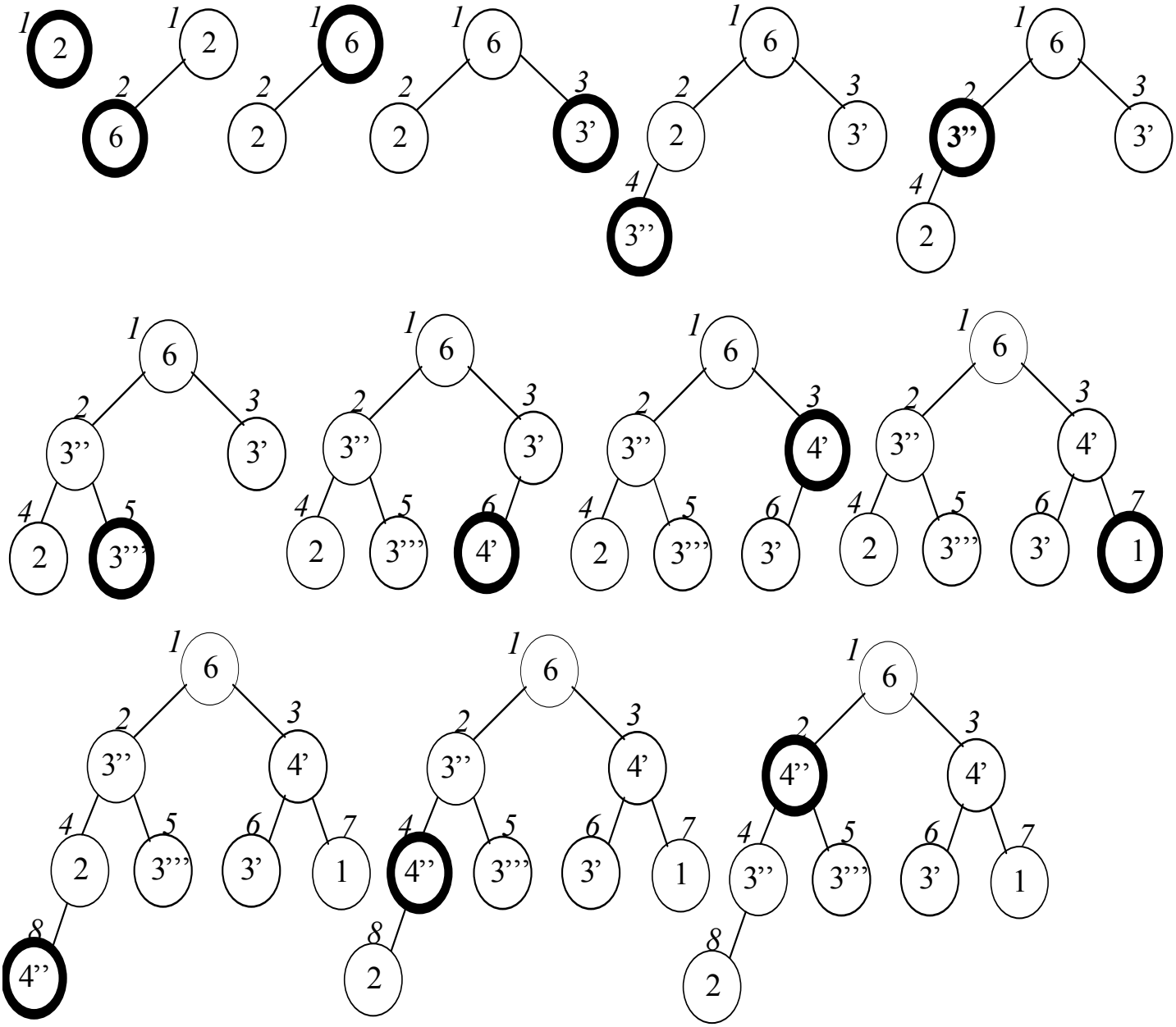
```
//*********funkcje interfejsu ADT kolejki p************
 void Inicjalizacja(kolejka_p& Kolejka_P)
 {  Kolejka_P.pierwszy = 0; }

 inline int Pusty(kolejka_p Kolejka_P)
 { return Kolejka_P.pierwszy==0; }

 int Wstaw_p(kolejka_p& Kolejka_P, dane Dana)
  { if (Kolejka_P.pierwszy==N) return 0;
    Kolejka_P.tab[Kolejka_P.pierwszy++]= Dana; //dodanie elementu na końcu ciągu
    return 1;
   }

 dane Usun_max(kolejka_p& Kolejka_P)
   { int maxind=0;
     for (int i=1; i< Kolejka_P.pierwszy; i++)    //wyszukanie największego elementu
       if (Kolejka_P.tab[i] > Kolejka_P.tab[maxind])
            maxind = i;
     dane max = Kolejka_P.tab[maxind];           //pobranie największego elementu
                                  //zapisanie w miejscu pobranego elementu ostatniego z ciągu
     Kolejka_P.tab[maxind] = Kolejka_P.tab[--Kolejka_P.pierwszy];
     return max;                                 //zwrócenie największego elementu
    }

//*********funkcje ogólnego 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(); }

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

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

Etap 3 - Implementacja za pomocą tablicy - kopca

Przykład: Należy wstawić dany ciąg wejściowy do tablicy kopca, czyli ustawić elementy w ciągu, aby wartość każdego elementu o numerze *„i”* (element zwany *„ojcem”*) była nie mniejsza niż wartość elementów o numerach: *„2 * i”* (element *„lewy”*) oraz *„2 * i + 1”* (element *„prawy”*).

Numery elementów	*1*	*2*	*3*	*4*	*5*	*6*	*7*	*8*
wartości elementów	2	6	3	3	3	4	1	4

Zadanie 1: Utwórz kopiec metodą wstępującą

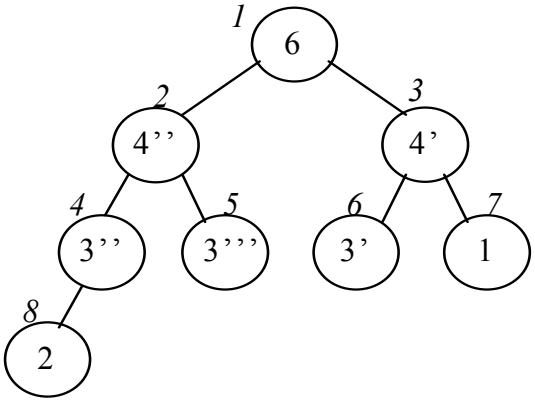


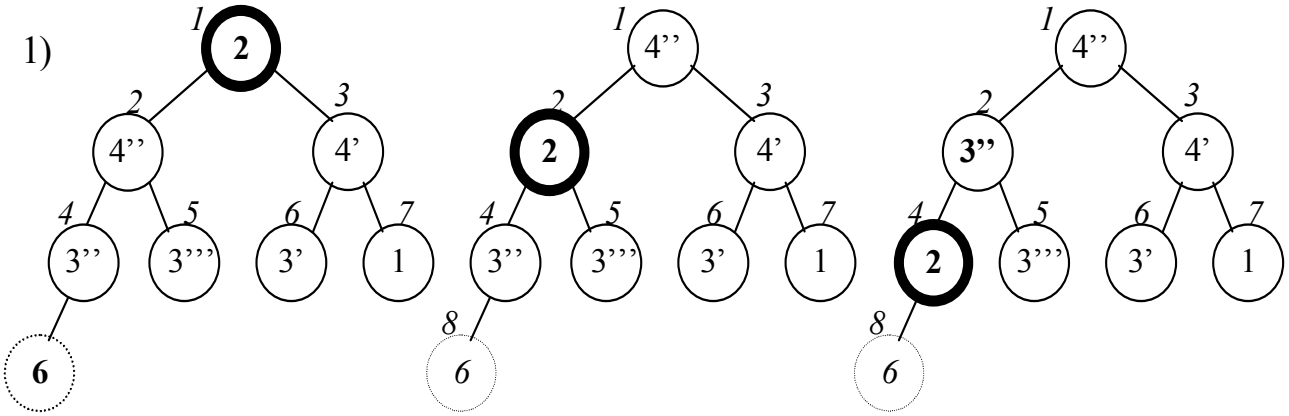
Uwaga 1: Ciąg przekształcony w kopiec. W ciągu typu *„kopiec”* element o największej wartości znajduje się na pierwszej pozycji:

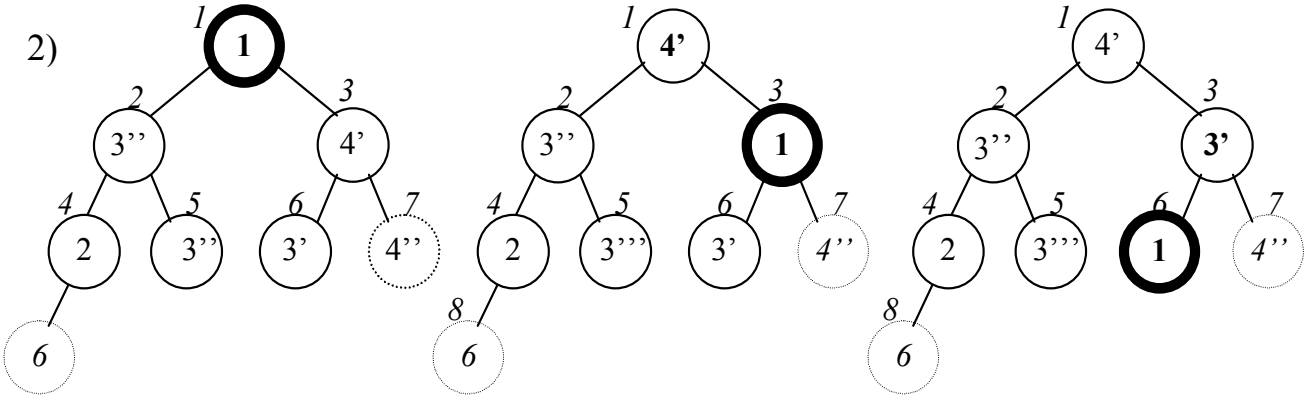
Numery elementów	*1*	*2*	*3*	*4*	*5*	*6*	*7*	*8*

wartości elementów	6	4	4	3	3	3	1	2

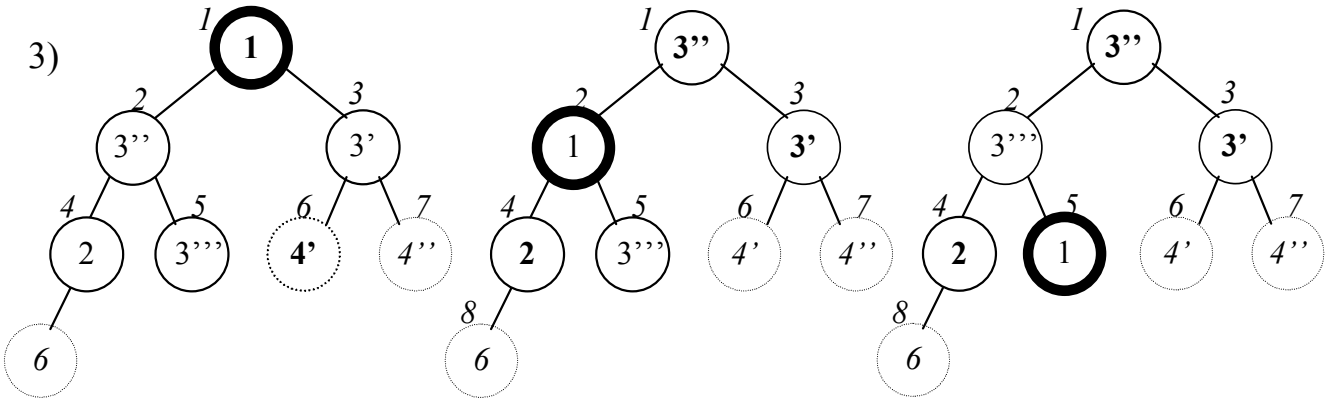
Zadanie 2: Należy kolejno usuwać kolejne elementy największe z kolejki priorytetowej za pomocą **przesiewania zstępującego** czyli zamiany największego elementu z początku kopca z elementem ostatnim w kopcu i odbudowy kopca z pozostałych elementów. Następnie można ten ostatni element usunąć kolejki priorytetowej. Proces ten należy powtarzać *„N"* razy, gdzie *„N"* jest liczbą elementów ciągu.



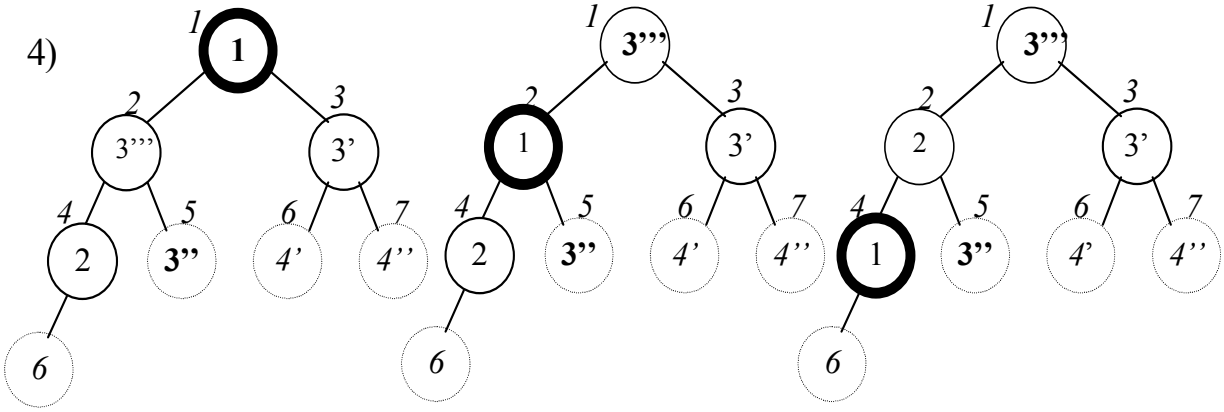




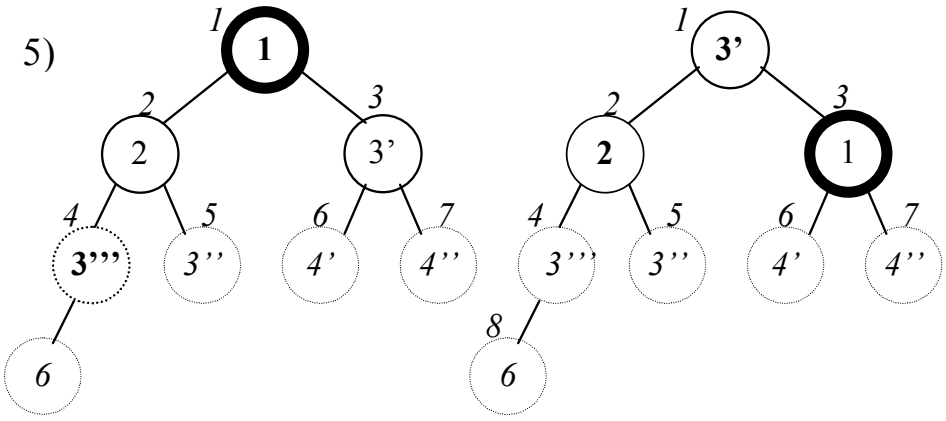
3)


1
3''
3'
2
3'''
4'
4''
6
3''
1
3'
2
3'''
4'
4''
6
3''
3'''
3'
2
1
4'
4''
6

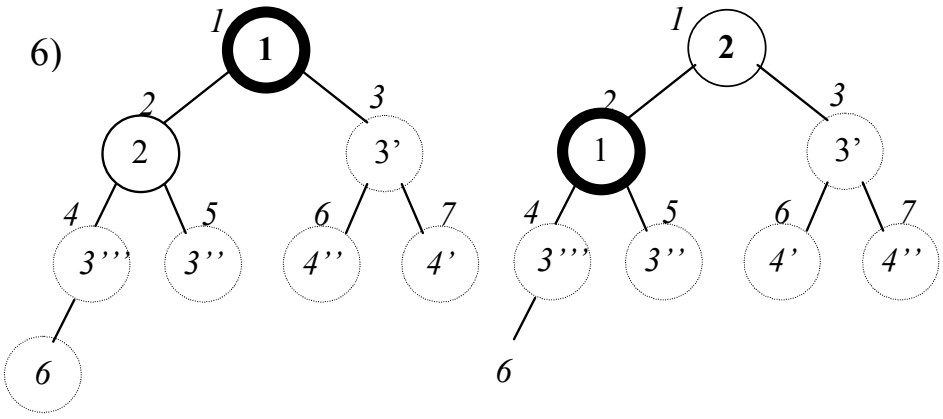
4)

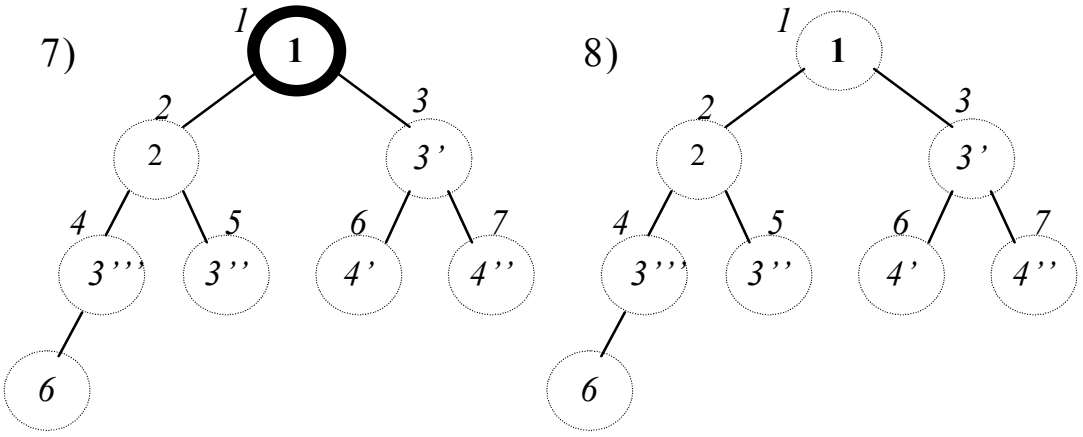

1
3'''
3'
2
3''
4'
4''
6
3'''
1
3'
2
3''
4'
4''
6
3'''
2
3'
1
3''
4'
4''
6

5)


1
2
3'
3'''
3''
4'
4''
6
3'
2
1
3'''
3''
4'
4''
8
6

6)


1
2
3'
3'''
3''
4''
4'
6
2
1
3'
3'''
3''
4'
4''
6



Uwaga 2:

Ciąg usuwanych elementów posiada własności „kopca" , gdzie element o numerze *i* jest mniejszy lub równy elementom o numerach *2*i* oraz *2*i+1.*

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

//1. interfejs ADT stosu
 typedef int dane;                                  // dane umieszczone stosie
 const long N=11;
 struct stos
  { int pierwszy;
    dane tab[N];
  };
 typedef stos kolejka_p;                  //nowa nazwa wynikająca z przedefiniowanego
                                          //zastosowania danych i niektórych funkcji stosu
        //prototypy funkcji kolejki_prorytetowej
 void Inicjalizacja(kolejka_p& Kolejka_P);
 inline int Pusty(kolejka_p Kolejka_P);
 int Wstaw_p(kolejka_p& Kolejka_P, dane Dana);
 dane Usun_max(kolejka_p& Kolejka_P);

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

 //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_p",
                        "2 : Usuwanie z kolejki_p najwiekszego elementu",
                        "3 : Wydruk kolejki_p nierosnaco wraz z jej usuwaniem",
                         " >Esc Koniec programu"};

//funkcje klienta korzystajace z kolejki priorytetowej
void Wstaw_do_kol_p(kolejka_p& Kolejka_P);
void Wyswietl_usun_z_kol_p(kolejka_p& Kolejka_P);
```

```
void main(void)
 { kolejka_p Kolejka_P;
   char Wybor;

   clrscr();
   Inicjalizacja(Kolejka_P);
   do
   { Wybor= Menu(POZ, Tab_menu);
    switch (Wybor)
    {case '1' : Wstaw_do_kol_p(Kolejka_P);
                break;
     case '2' :  if (Pusty(Kolejka_P))
                    Komunikat("\nKolejka_p pusta\n");
                else (Usun_max(Kolejka_P));
                    break;
     case '3' :  if (Pusty(Kolejka_P))
                        Komunikat("\nKolejka_p pusta\n") ;
                 else Wyswietl_usun_z_kol_p(Kolejka_P);
                 break;
      }
   } while (Wybor !=Esc );
  }
```

/*funkcje do przesiewania stogu w górę i w dół, pomocnicze dla kolejki priorytetowej szybkie znajdowanie elementu największego*/

```
inline void zamien(dane  &a, dane  &b)
   { dane pom=a;
     a=b;
     b=pom;
   }
```

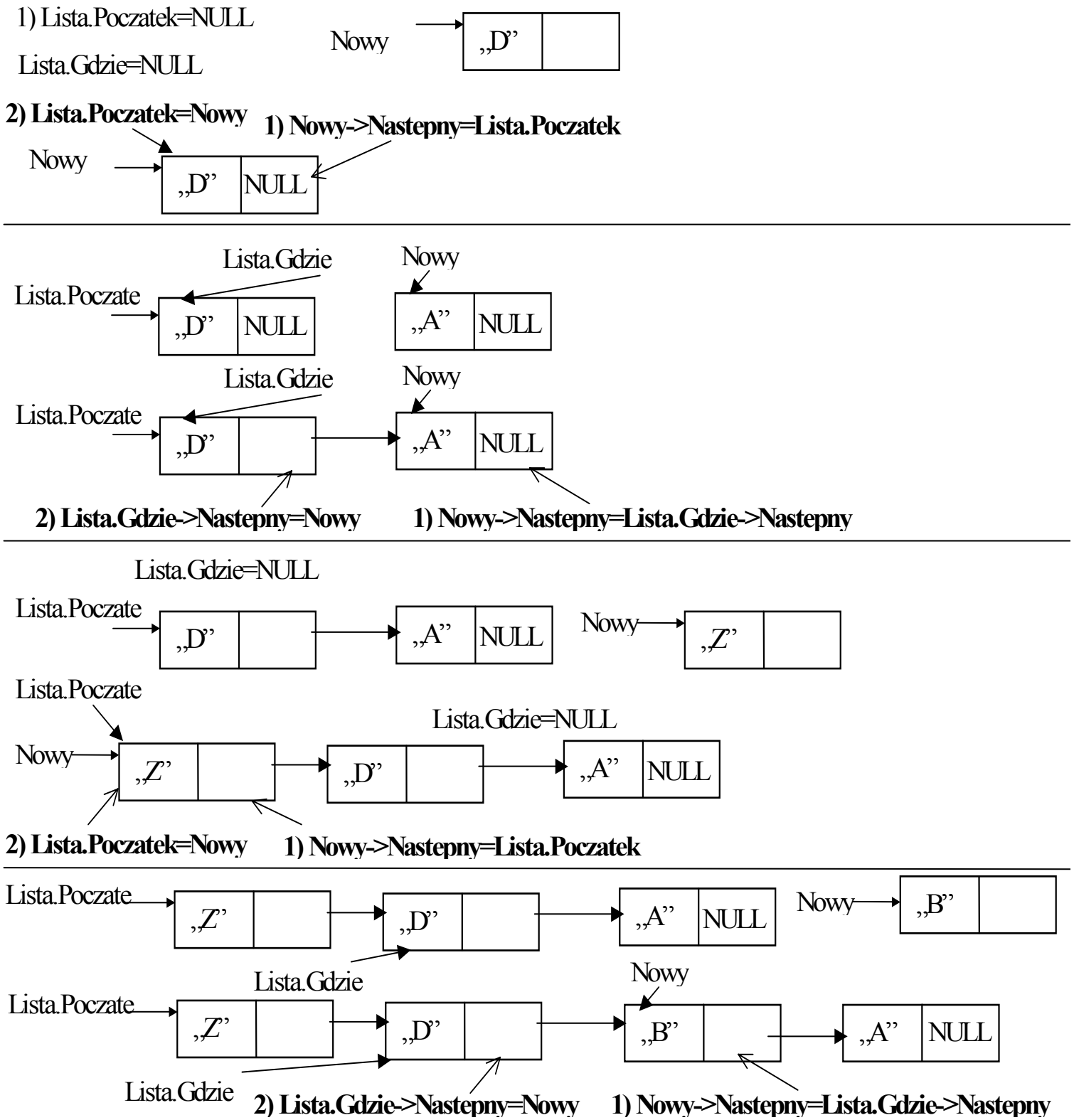
//prostszy zapis funkcji do „przesiewania zstępującego kopca”

```
void przywroc_kopiec_w_dol(dane t[], long p, long ojciec)
 {long maks,ll,pp;
     while (ojciec < p)
       {   ll = 2*ojciec;                    //lewy nastepca ojca
           pp=ll+1;                          //prawy nastepca ojca
           if (ll <= p && t[ll] >  t[ojciec]) maks=ll;
           else maks=ojciec;
           if (pp <= p && t[maks] < t[pp]) maks=pp;
           if (maks!=ojciec)
              { zamien(t[maks],t[ojciec]);
                ojciec=maks;}
           else break;
   }
}
```

```
//funkcja, która „przesiewa kopiec wstępująco”

void przywroc_kopiec_w_gore(dane t[], long p, long ojciec)
 {  while (ojciec>0 && t[ojciec/2] < t[ojciec])
      { zamien(t[ojciec/2], t[ojciec]);
        ojciec=ojciec/2;}
 }
//*************************************************************
//**funkcje interfejsu ADT kolejki p opartej na funkcjach stogu*

void Inicjalizacja(kolejka_p& Kolejka_P)
 {  Kolejka_P.pierwszy = 0; }

 inline int Pusty(kolejka_p Kolejka_P)
 { return Kolejka_P.pierwszy==0; }

 int Wstaw_p(kolejka_p& Kolejka_P, dane Dana)
  { if (Kolejka_P.pierwszy==N) return 0;
                                    //wstawianie jako do stosu, czyli na koniec kopca
    Kolejka_P.tab[Kolejka_P.pierwszy++]= Dana;
    przywroc_kopiec_w_gore(Kolejka_P.tab, 0, Kolejka_P.pierwszy-1);
                                  //wstawienie największego elementu na początek kopca
    return 1;
   }

 dane Usun_max(kolejka_p& Kolejka_P)
   {
     zamien(Kolejka_P.tab[0], K Kolejka_P.tab[--Kolejka_P.pierwszy]);
                                                  //największy element na koniec stogu
     przywroc_kopiec_w_dol(Kolejka_P.tab, Kolejka_P.pierwszy-1, 0);
                                                  //odtworzenie stogu bez ostatniego elementu
     dane max= Kolejka_P.tab[Kolejka_P.pierwszy];
                                                  //pobranie największego elementu ze stogu
     return max;
   }
```

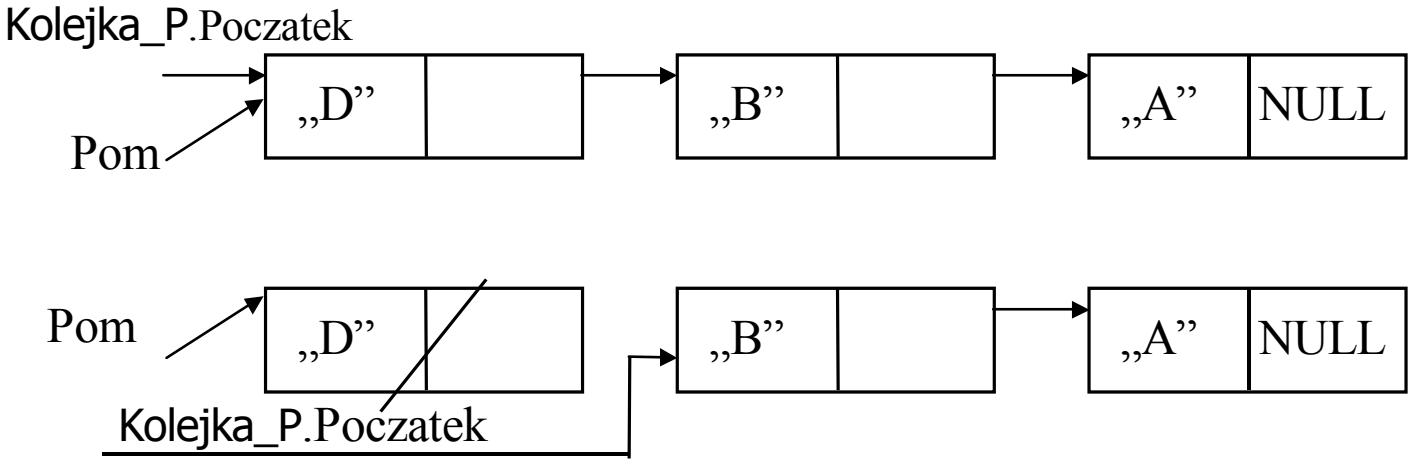
Etap 3. Implementacja kolejki priorytetowej za pomocą sortowanej listy wiązanej

- szukanie i wstawianie do listy w przed większym lub równym elementem lub na końcu listy, gdy nie znaleziono elementu równego lub większego (wynik funkcji Szukaj jest dowolny)



```
int Szukaj(lista& Lista,dane Dana)
  {Lista.Gdzie = NULL;
   if (Pusty(Lista)) return 0;
   stos Nast = Lista.Poczatek;
   while ((Nast->Nastepny !=NULL) && Nast->Dane>Dana)
       { Lista.Gdzie= Nast;
         Nast = Nast->Nastepny; }
   if (Nast->Dane> Dana)              //wstawiany element jest najmniejszy,
      Lista.Gdzie= Nast;              //wiec jest wstawiany na końcu
   if (Nast->Dane==Dana) return 2;
   else return 1;
  }

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

- usuwanie elementów zawsze na początku kolejki priorytetowej – posortowanej listy wiązanej



/*interfejs ADT kolejki priorytetowej jako uporządkowanej listy wiązanej */

```
dane Usun_max(kolejka_p& Kolejka_P) //jak ze stosu-zdejmuje się największy element
  { stos Pom;                       //listy posortowanej nierosnąco
    dane d;
    Pom= Kolejka_P.Poczatek;        //zapamiętanie pierwszego elementu do usunięcia
    Kolejka_P.Poczatek= Kolejka_P.Poczatek->Nastepny;
                                    //odłączenie pierwszego elementu od listy
    d=Pom->Dane;                        //((*Pom).Dane)
    delete Pom;                     //usunięcie pierwszego elementu z pamięci
    return d;
  }

int Wstaw_p(kolejka_p& Kolejka_P, dane Dana)      //powstaje lista posortowana nierosnąco
 { Szukaj(Kolejka_P, Dana);
   int a = Wstaw(Kolejka_P, Dana);|
   return a;
 }


void Inicjalizacja(kolejka_p& Kolejka_P)
 { Kolejka_P.Poczatek = NULL;}

 inline int Pusty(kolejka_p Kolejka_P)
 { return Kolejka_P.Poczatek==NULL;  }
```

```
#include <conio.h>
#include <stdio.h>
//1. interfejs ADT kolejki priorytetowej jako uporządkowanej listy
//wiązanej
 typedef int dane;                          // dane umieszczone liście
 typedef struct ELEMENT* stos;              // nazwa wskaźnika na element stosu

 struct ELEMENT                             //element listy
  {   dane Dane;
      stos Nastepny;
  };
 struct lista                               //typ listy wiązana uporządkowana
  { stos Poczatek;
    stos Gdzie;
  };
 typedef lista kolejka_p;                   //nowa nazwa wynikająca z przedefiniowanego
                                            //zastosowania danych i niektórych funkcji listy
       //prototypy funkcji kolejki_prorytetowej
 void Inicjalizacja(kolejka_p& Kolejka_p);
 inline int Pusty(kolejka_p Kolejka_p);
 int Wstaw_p(kolejka_p& Kolejka_P, dane Dana);
 dane Usun_max(kolejka_p& Kolejka_P);

 //2. funkcje we/wy dla danych umieszczonych na liście
 void Pokaz_dane (dane Dana);
 dane Dane(char* s);

 //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_p",
                         "2 : Usuwanie z kolejki_p najwiekszego elementu",
                         "3 : Wydruk kolejki_p nierosnaco wraz z jej usuwaniem",
                        " >Esc Koniec programu"};
```

```
//funkcje klienta korzystajace z kolejki priorytetowej
void Wstaw_do_kol_p(kolejka_p& Kolejka_P);
void Wyswietl_usun_z_kol_p(kolejka_p& Kolejka_P);

 void main(void)
 { kolejka_p Kolejka_P;
   char Wybor;

   clrscr();
   Inicjalizacja(Kolejka_P);
   do
   { Wybor= Menu(POZ, Tab_menu);
    switch (Wybor)
    {case '1' : Wstaw_do_kol_p(Kolejka_P);
                break;
     case '2' : if (Pusty(Kolejka_P))
                     Komunikat("\nKolejka_p pusta\n");
                else (Usun_max(Kolejka_P));
                   break;
     case '3' : if (Pusty(Kolejka_P))
                     Komunikat("\nKolejka_p pusta\n") ;
                else Wyswietl_usun_z_kol_p(Kolejka_P);
               break;
       }
   } while (Wybor !=Esc );
  }
/*pomocnicze funkcje dla kolejki priorytetowej – jako uporządkowanej listy wiazanej*/
int Szukaj(lista& Lista,dane Dana)
   {Lista.Gdzie = NULL;
    if (Pusty(Lista)) return 0;
    stos Nast = Lista.Poczatek;
    while ((Nast->Nastepny !=NULL) && Nast->Dane>Dana)
     { Lista.Gdzie= Nast;
       Nast = Nast->Nastepny; }
    if (Nast->Dane> Dana)   //wstawiany element jest najmniejszy,
        Lista.Gdzie= Nast;  //wiec jest wstawiany na koncu
    if (Nast->Dane==Dana) return 2;
    else return 1;
   }

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

/*interfejs ADT kolejki priorytetowej jako uporządkowanej listy wiązanej */

```
void Inicjalizacja(kolejka_p& Kolejka_P)
 { Kolejka_P.Poczatek = NULL;}

inline int Pusty(kolejka_p Kolejka_P)
{ return Kolejka_P.Poczatek==NULL;    }


int Wstaw_p(kolejka_p& Kolejka_P, dane Dana)     //powstaje lista posortowana nierosnąco
 { Szukaj(Kolejka_P, Dana);
   int a = Wstaw(Kolejka_P,Dana);
   return a;
 }

dane Usun_max(kolejka_p& Kolejka_P)           //jak ze stosu-zdejmuje się największy element
  { stos Pom;                                 //listy posortowanej nierosnąco
    dane d;
    Pom= Kolejka_P.Poczatek;
    Kolejka_P.Poczatek= Kolejka_P.Poczatek->Nastepny;
    d=Pom->Dane;
    delete Pom;
    return d;
  }
```

//****funkcje klienta korzystające z kolejki priorytetowej****

```
void Wstaw_do_kol_p(kolejka_p& Kolejka_P)
 {   dane Dana= Dane("Podaj dane: ");
     { if (!Wstaw_p(Kolejka_P, Dana))
       Komunikat("\nBrak pamieci");}
  }

void Wyswietl_usun_z_kol_p(kolejka_p& Kolejka_P)
   {dane d;
    while (!Pusty(Kolejka_P))
     {  d=Usun_max(Kolejka_P);
        Pokaz_dane(d); }
   }
```

//*********funkcje ogólnego 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(); }
```

//funkcje we/wy dla danych umieszczonych na kolejce priorytetowej

```
dane Dane(char* s)
  { int a;
    do
     { fflush(stdin);
       printf("\n\n%s",s);
     } while (scanf("%d",&a)!=1);
   return a;}

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