1.7. РЕАЛІЗАЦІЯ СТЕКІВ ТА ЧЕРГ З ВИКОРИСТАННЯМ МАСИВІВ

Будемо розглядати стеки та черги як структури даних, кількість елементів яких заздалегідь не визначена, тобто їх кількість може суттєво змінюватись під час роботи програми. Враховуючи це, для реалізації цих структур будемо використовувати динамічні масиви.

Стек на базі динамічного масиву

Довжину динамічного масиву можна змінювати під час виконання програми, але ці зміни стосуються виключно додавання певної кількості елементів у кінець або видалення певної кількості елементів з кінця

масиву. Таким чином, треба організовувати роботу так, щоб додавання та видалення елементів відбувалося з кінця масиву. Алгоритм додавання елементів у стек буде відбуватися так:

На вхід масиву з ім'ям `M` подається черговий елемент, кількість елементів динамічного масиву збільшується на `1`, номер цього елемента `N` (не забувати, що рахунок елементів у динамічному масиві з `0`), елемент записується у комірку `M[N]`. Таким чином отримуємо масив, у якому в «голові» знаходиться перший елемент, а у «хвості» – останній.

При необхідності забрати елемент зі стеку треба тільки прочитати останній елемент (його номер можна визначити за допомогою вбудованої функції `High(M)`), та зменшити кількість елементів масиву на одиницю. Номер останнього елемента – це кількість елементів стеку мінус одиниця. Динамічний масив дозволяє раціонально використовувати оперативну пам'ять. Нижче наведено фрагмент програми для обробки стеку на базі динамічного масиву.

```
type
   Person=record //опис структури елемента стеку
     Name: string[20];
     Year: byte;
   end;
   Group=array of Person; {динамічний масив елементів
     стеку}
var
   P: Person;
   DinArray: Group;
   I,N: byte;
   Ft: textfile;
procedure ExtractStack(var DinArray: Group;
                       var P: Person);
{Вибір чергового елемента зі стеку, зменшення розміру
масиву, розташування вибраного елемента у змінній P}
var
   N: integer;
begin
   N:=High(DinArray);
   if N=-1 then
     begin //стек порожній
       writeln('Stack is empty');
       Exit;
     end;
```

```
   P.Name:=DinArray[N].Name;  {присвоєння останнього
елемента змінній P}
   P.Year:=DinArray[N].Year;
   SetLength(DinArray,N);  {зменшення розміру масиву
на 1}
end;  {procedure ExtractStack}

procedure AddStack(var DinArray: Group; P: Person);
{Додавання чергового елемента P у стек, збільшення
розміру масиву на 1}
var
   N: integer;
begin
   N:=High(DinArray)+1;  {кількість елементів у стеку}
   Inc(N);
   SetLength(G,N);  {збільшення кількості елементів
у масиві}
   G[N-1].Name:=P.Name;  {запис нового елемента у
добавлену комірку массиву}
   G[N-1].Year:=P.Year;
end;  {procedure AddStack}
```

Черга на базі динамічного масиву

Порівняно із попереднім випадком задача дещо ускладнюється. Проблема полягає у тому, що заявки, які надходять, записуються у «хвіст» черги, а на виконання вибираються з «голови». Друга проблема полягає у тому, що черга – це динамічний потік заявок, які неперервно просуваються від «хвоста» до «голови». Для організації черги можна застосувати два підходи.

Перший: збільшувати розмір масиву на одиницю (+1) при надходженні чергової заявки і записувати її у «хвіст» масиву, вибір заявки буде відбуватися з «голови». Але проблема полягає в тому, що при такому підході розмір масиву необмежено росте разом із кількістю заявок, які надходять. А при виборі заявок на виконання з «голови» черги комірки масиву звільняються і надалі не використовуються (зменшити кількість елементів динамічного масиву можна тільки з його «хвоста»). Це призводить до нераціонального використання оперативної пам`яті комп'ютера.

Другий підхід: ініціалізувати динамічний масив із кількістю елементів, яка достатня для зберігання поточної кількості заявок на будь-який час роботи системи. Масив уявляємо собі «круговим» (кругова черга).

Заповнення масиву відбувається з «хвоста», вибір елементів – з «голови». Кількість елементів обирається такою, що до заповнення всіх комірок масиву відбувається вибір елементів з черги і частина комірок в «голові» звільнюється. Тепер, якщо всі елементи масиву заповнені заявками і надходить нова – вона записується у першу комірку масиву, заявка з якої вже вибрана раніш і рахується вільною. Таким чином, запис у масив відбувається у комірки по годинникової стрілці, а вибір елементів – проти годинникової стрілки. У більшості випадків поточна довжина черги не дуже велика, тому при такому підході можна обійтися масивом невеликої довжини. Якщо трапиться критичний випадок, можна передбачити дещо збільшення елементів динамічного масиву.

Нижче наведено коди основних процедур для реалізації роботи з чергою на базі «кругового» динамічного масиву довжиною 10 елементів. Виведення стану черги виконується у візуальний компонент на формі – Memo.

```
const
   K: integer=0; {кількість елементів черги}
   F: integer=0; {номер елемента, з якого починається
      черга}
   R: integer=0; {номер елемента, яким закінчується
      черга}
type
   MasDin=array of integer; {масив елементів черги,
      рахуємо як "круговий"}
var
   M: MasDin;
   Numb: integer;
procedure Insert(P: integer; var F,R: integer;
                 var M: MasDin);
{Вставка елемента в чергу:
P – елемент, що вставляється;
F,R – перший і останній номери елементів масиву черги}
begin
   Numb:=High(M); {максимальний номер елемента масиву
      черги}
   if Numb=-1 then
      begin {якщо масив черги порожній – ініціалі-
зувати 10 елементів (початок роботи програми)}
         SetLength(M,10);
         Numb:=High(M); //M=9
      end;
```

```
   M[R]:=P; //запис заявки у масив (у чергу)
   Inc(R);
   Inc(K); //кількість заявок у черзі
end; //procedure Insert

procedure InsertToLine;
{Додавання поточного елемента в чергу, якщо вона
не переповнена}
var
   Temp,I,J: integer;
   St: string;
begin
   if K=10 then
      begin
         MainForm.Memo1.Lines.Add('--Черга  перепов-
нена!!!---');
         Exit;
      end;
   if R=10 then R:=0; {поточний номер комірки, куди
писати заявку}
   Temp:=Random(101);
   if Temp=0 then Temp:=1;
   Insert(Temp,F,R,M); //додавання елемента
   I:=F; St:=''; J:=1;
   while J <= K do
      begin {виведення поточного стану черги на
екран}
         St:=St+IntToStr(M[I])+' ';
         Inc(J);
         Inc(I);
         if I=10 then I:=0;
      end;
   MainForm.Memo1.Lines.Add(St);
end; //procedure InsertToLine;

procedure Extract(var P: integer; var F,R: integer;
                  var M: MasDin);
{Вилучення елемента з "голови" черги а заповнення
його елементом, який можна рахувати як порожній,
наприклад, -200 (для контролю коректності роботи
```

```
програми), якщо програма працює правильно, цей елемент
не повинен виводитись на екран}
begin
   P:=M[F];
   M[F]:=200;
   Dec(K);
   Inc(F);
end; //procedure Extract

procedure ExtractFromLine;
{Вилучення елемента із "голови" черги та заповнення
його елементом, який можна ражувати як порожній –
200 (для контролю коректності роботи програми), якщо
програма працює правильно, цей елемент не повинен
виводитись на екран}
var
   Temp,I,J: integer;
   St: string;
begin
   if K=0 then
      begin
         St:='----Черга порожня !!!----';
         MainForm.Memo1.Lines.Add(St);
         Exit;
      end;
   St:='';
   Extract(Temp,F,R,M); //забрати елемент із черги
   if F=10 then F:=0;
   I:=F; J:=1;
   while J<=K do
      begin //виведення поточного стану черги на екран
         St:=St+IntToStr(M[I])+' ';
         Inc(J);
         Inc(I);
         if I=10 then I:=0;
      end;
   if St<>'' then MainForm.Memo1.Lines.Add(St)
   else MainForm.Memo1.Lines.Add('Із черги забрано
останній елемент');
   if F>9 then F:=0;
end; //procedure ExtractFromLine;
```