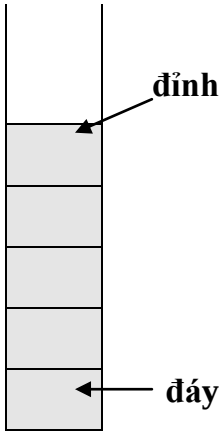
CHƯƠNG III

NGĂN XẾP VÀ HÀNG ĐỢI

I. NGĂN XẾP

1. Định nghĩa

Ngăn xếp (*stack*) là danh sách mà cả hai phép toán thêm vào và loại bỏ được thực hiện ở một đầu theo nguyên tắc *vào sau ra trước* LIFO: *Last In First Out*.



2. Tổ chức ngăn xếp theo danh sách đặc

Ta có thể tổ chức ngăn xếp bằng danh sách đặc như sau.

a. Khai báo

Trong PASCAL	Trong C
Const StackLimit = 100 *{số phần tử tối đa của ngăn xếp}* *Type InfoType* = *<kiểu dữ liệu> ;* *Var stack : array*[1.. *StackLimit*] *of InfoType; {stack}* *sp : integer; {stack pointer}*	*#define StackLimit* 100 *//số phần tử của ngăn xếp* *typedef <kiểu dữ liệu> InfoType;* *InfoType stack*[*StackLimit*+1]*;* *int sp;//stack pointer*

b. Khởi tạo ngăn xếp rỗng

Trong PASCAL	Trong C
Procedure Initialize; *begin* *sp* := 0; *end;*	*void Initialize*() { *sp* = 0; }

c. Thêm phần tử vào ngăn xếp

Điều kiện thực hiện là ngăn xếp chưa đầy (*sp* < *StackLimit*).

Trong PASCAL	Trong C
Procedure Push(*NewElement:InfoType*); *begin* *sp := sp + 1;* *stack*[*sp*] := *NewElement;* *end;*	*void Push*(*InfoType NewElement*) { *stack*[*++sp*] = *NewElement;* }

d. Loại phần tử khỏi ngăn xếp và đưa vào biến

Điều kiện thực hiện là ngăn xếp không rỗng (*sp* > 0).

Trong PASCAL	Trong C
Procedure Pop(*Var Item: InfoType*); *begin* *Item :=stack*[*sp*]; *sp := sp* - 1; *end;*	*void Pop*(*InfoType *x*) { **x = stack*[*sp--*]; } *Ghi chú:* Lệnh gọi thủ tục *Pop* truyền tham số *Item*: *Pop*(*&Item*).

3. Tổ chức ngăn xếp theo danh sách liên kết

a. Khai báo

Trong PASCAL	Trong C
Type *InfoType = <kiểu dữ liệu>;* *NodePointer = ^Node;* *Node = record* *info : InfoType;* *next : NodePointer;* *end;* *var sp : NodePointer; {stack pointer}*	*typedef <kiểu dữ liệu chứa trong nút>* *InfoType;* *struct Node* { *InfoType info;* *struct Node *next;* *};* *typedef struct Node *NodePointer;* *NodePointer sp, p;/ / sp là con trỏ ngăn xếp*

b. Khởi tạo ngăn xếp rỗng

Trong PASCAL	Trong C
Procedure Initialize; *begin* *sp := nil;* *end;*	*void Initialize*() { *sp = NULL;* }

c. Thêm phần tử vào ngăn xếp

Trong PASCAL

```
Procedure Push(NewInfo: InfoType);
var p : NodePointer;
begin
      new(p);
      p^.info := NewInfo;
      p^.next := sp;
      sp := p;
end;
```

Trong C

```
void  Push(InfoType NewInfo)
{
   NodePointer p;
   p = (NodePointer)malloc(sizeof(struct
   Node));
   p->info = NewInfo;
   p->next = sp;
   sp = p;
}
```

d. Lấy phần tử ra khỏi ngăn xếp

Trong PASCAL

```
Procedure   Pop(Var  Item: InfoType);
Var p: NodePointer;
begin
         p := sp;
         sp := sp^.next;
         Item := p^.info;
         dispose(p);
end;
```

Trong C

```
   void Pop(InfoType *Item)
   {
        NodePointer p;
        p = sp;
        sp = p->next;
        *Item = p->info;
        free(p);
   }
```

◊ *Ghi chú*: Gọi thủ tục bằng *Pop*(*&Item*).

4. Ứng dụng

a. Thuật toán chuyển đổi cơ số

Thuật toán này chuyển đổi biểu diễn cơ số 10 của một số nguyên dương *Number* sang cơ số 2 và hiển thị biểu diễn cơ số 2.

(1) Lặp lại các bước sau cho đến khi *Number* = 0:

(a) Tính số dư *Remainder* của phép chia *Number* cho 2

(b) Đặt số dư *Remainder* vào ngăn xếp

(c) Thay *Number* bằng phần nguyên của kết quả phép chia *Number* cho 2

(2) Lặp lại các bước sau cho đến khi ngăn xếp số dư rỗng:

(a) Lấy ra *Remainder* từ ngăn xếp

(b) Hiển thị *Remainder*

Các chương trình sau cài đặt giải thuật chuyển đổi số thập phân sang hệ nhị phân.

◊ Trong PASCAL:

- Sử dụng danh sách đặc

```
Program  BaseTenToBaseTwo;
Const     StackLimit     = 50;
Type   ElementType      = integer;
          StackArray      = array[1..StackLimit] of ElementType;
          StackType    =  record
                                 Top       : 0.. StackLimit;
                                 Element     : StackArray;
                              end;
Var     Number,          {số cần chuyển đổi}
          Remainder : integer;
          Stack :  StackType;
          Response : char;
{*********************************}
Procedure Create(Var  Stack :  StackType);
{tạo stack rỗng}
Begin
     stack.top := 0;
end;
{*********************************}
Function Empty(Stack :  StackType) : boolean;
{trả về true nếu stack rỗng}
Begin
     Empty := (stack.top = 0);
end;
{*********************************}
Procedure  Pop(Var   Stack: StackType; var  item: ElementType);
{lấy item ra từ đỉnh stack}
Begin
     if empty(stack) then halt
     else
          with stack do
               begin
                    item := Element[Top];
                    Top := Top - 1
               end
end;
{*********************************}
Procedure   Push(Var   Stack:   StackType;   item:  ElementType);
{đẩy item vào stack}
Begin
     if stack.top = StackLimit then halt
     else
          with stack do
               begin
                    Top := Top + 1;
                    Element[top] := item;
```

```
            end
    end;
{*********************************}
    BEGIN {chương trình chính}
        repeat
            write('Nhập số : ');
            readln(Number);
            create(stack);
            while Number <> 0 do
                begin
                    Remainder := Number MOD 2;
                    Push( Stack, Remainder);
                    Number := Number DIV 2;
                end;
            write('Biểu diễn cơ số 2 :');
            while not empty(stack) do
                begin
                    Pop( Stack, Remainder);
                    write(remainder:1);
                end;
            writeln; writeln;
            write('Tiếp tục (Y/N):');
            readln(response);
        until not (response in ['Y','y']);
    END. { kết thúc}
```

- Sử dụng danh sách liên kết

```
    Program  BaseTenToBaseTwo;
    Type   ElementType = integer;
        StackType  = ^StackNode;
        StackNode = Record
                      Data :   ElementType;
                      Next :    StackType;
                    End;
    Var     Number,          {số cần chuyển đổi}
            Remainder,  {số dư của phép chia Number cho 2}
                  : integer;
            Stack :  StackType;  {ngăn xếp của số dư}
            Response : char;  {trả lời của người dùng}
{*********************************}
    Procedure Create(Var  Stack :  StackType);
    {tạo stack rỗng}
    Begin
        stack := nil;
    end;
{*********************************}
```

```
  Function Empty(Stack :  StackType) : boolean;
  {trả về true nếu stack rỗng}
  Begin
       Empty := (stack = nil);
  end;
{*********************************}
  Procedure  Pop(Var   Stack: StackType; var  item: ElementType);
  {lấy item ra từ đỉnh stack}
  var tempPtr : StackType; {con trỏ tạm thời}
  Begin
       if  empty(stack) then halt
       else
            begin
                 item := stack^.data;
                 tempPtr := Stack;
                 stack := stack^.next;
                 dispose(tempPtr);
            end
  end;
{*********************************}
  Procedure  Push(Var  Stack:  StackType;  item:  ElementType);
  {đẩy item vào stack}
  var tempPtr : StackType; {con trỏ tạm thời}
  Begin
    new(tempPtr);
    tempPtr^.data := item;
    tempPtr^.next := stack;
    stack := tempPtr;
  end;
{*********************************}
  BEGIN {chương trình chính}
       repeat
            write('Nhập số : ');
            readln(Number);
            create(stack);
            while Number <> 0 do
                 begin
                      Remainder := Number MOD 2;
                      Push( Stack, Remainder);
                      Number := Number DIV 2;
                 end;
            write('Biểu diễn cơ số 2 :');
            while not empty(stack) do
                 begin
                      Pop( Stack, Remainder);
                      write(remainder:1);
```

```
            end;
        writeln; writeln;
        write('Tiếp tục :');
        readln(response);
    until not (response in ['Y','y']);
END. { kết thúc}
```

◊ Trong C:

```
main()
{
while (Number !=0)
   {
      Remainder = Number % 2;
      Push(Remainder);
      Number = Number/2;
   }
while (sp != NULL) //hoặc while (sp > 0) đối với danh sách đặc
   {
      Pop(&Remainder);
      printf("%d",Remainder);
   }
}
```

b. Ký pháp nghịch đảo Ba lan

Trong đa số ngôn ngữ lập trình các biểu thức số học được viết theo *ký pháp trung tố* (*infix notation*) như

$$A * (B + C)$$

Nhiều bộ dịch trước hết chuyển ký pháp trên sang *ký pháp hậu tố* (*postfix notation*), sau đó mới tạo ra các chỉ thị máy để đánh giá biểu thức hậu tố này. Các biểu thức hậu tố dễ đánh giá cơ học hơn biểu thức trung tố.

Trong những năm đầu 1950, Nhà toán học Balan *Jan Lukasiewicz* đã đưa ra cách viết các biểu thức toán học không cần dấu ngoặc, gọi là *ký pháp nghịch đảo Balan* (*Reverse Polish Notation* - RPN) hay *ký pháp tiền tố* (*prefix notation*).

◊ *Ví dụ*. Biểu thức

$$2 * (3 + 4)$$

viết thành

$$2\ 3\ 4\ +\ *$$

Biểu thức

$$(1 + 5) * (8 – (4 – 1))$$

viết thành

1 5 + 8 4 1 – – *

• *Thuật toán đánh giá biểu thức RPN*

(1) Khởi động ngăn xếp rỗng của các toán hạng.

(2) Lặp lại các bước sau cho đến khi gặp dấu kết thúc biểu thức được đọc:

(a) Đọc phần tử (toán hạng, toán tử)

(b) Nếu là toán hạng, đẩy vào ngăn xếp.

Nếu là toán tử, thực hiện các bước sau:

(i) Lấy ra ngăn xếp 2 giá trị (nếu không lấy được hai toán hạng thì có lỗi biểu thức, kết thúc thuật toán).

(ii) Áp dụng toán tử trên vào hai giá trị vừa lấy.

(iii) Đẩy kết quả vào ngăn xếp.

(3) Khi gặp dấu kết thúc biểu thức, giá trị biểu thức là giá trị duy nhất ở đỉnh ngăn xếp.

◊ *Ví dụ*

Biểu thức	Ngăn xếp	Chú thích
2 4 * 9 5 + – ↑	|2| ←đỉnh	Đẩy 2 vào ngăn xếp
4 * 9 5 + – ↑	|4| ←đỉnh |2|	Đẩy 4 vào ngăn xếp
* 9 5 + – ↑	|8| ←đỉnh	Lấy 4 và 2 từ ngăn xếp, nhân chúng với nhau và đẩy kết quả là 8 vào ngăn xếp
9 5 + – ↑	|9| ←đỉnh |8|	Đẩy 9 vào ngăn xếp
5 + – ↑	|5| ←đỉnh |9| |8|	Đẩy 5 vào ngăn xếp
+ – ↑	|14| ←đỉnh |8|	Lấy 5 và 9 từ ngăn xếp, cộng chúng với nhau, rồi đẩy kết quả là 14 vào ngăn xếp.
– ↑	|– 6| ←đỉnh	Lấy 14 và 8 từ ngăn xếp, trừ chúng với nhau, rồi đẩy kết quả là -6 vào ngăn xếp.

Kết thúc	|– 6| ←đỉnh	Giá trị biểu thức nằm ở đỉnh ngăn xếp.

Chương trình sau cài đặt giải thuật đánh giá biểu thức RPN.

◊ Trong PASCAL:

```
Program  InfixToRPN;
CONST
  MaxExpression  = 80 {chiều dài tối đa của biểu thức}
  StackLimit     = 50 {kích thước ngăn xếp}
  EndMark        = ';'      {dấu kết thúc biểu thức trung tố}
TYPE
  ElementType  = integer;
  Expression     = String;
  Stackarray= ARRAY[1 .. StackLimit] Of  ElementType;
  CharacterSet =  SET Of char;
  StackType  = RECORD
                    Top: 0 .. StackLimit;
                    Element: Stackarray
                END;
VAR
rpn : Expression;         {biểu thức RPN}
OperatorSet : CharacterSet; {tập hợp toán tử}
NumericSet : CharacterSet;  {tập hợp chữ số}
Response : char;
{**********thủ tục tạo ngăn xếp rỗng**************}
PROCEDURE Create(Var Stack : StackType);
begin
     Stack.Top := 0;
end;
{**********hàm ngăn xếp rỗng**************}
FUNCTION  Empty(Stack : StackType): boolean;
begin
     Empty := (Stack.Top := 0)
end;
{**********thủ tục lấy Item từ đỉnh ngăn xếp không rỗng**************}
PROCEDURE  Pop(Var Stack : StackType; var Item: ElementType);
begin
  with Stack do
     begin
          Item := Element[Top];
          Top := Top – 1;
     end;
end;
{**********thủ tục đẩy Item vào đỉnh ngăn xếp không đầy**************}
PROCEDURE  Push(Var Stack : StackType; Item: ElementType);
```

```
begin
  with Stack do
     begin
          Top := Top + 1;
          Element[Top] := Item;
     end;
end;
{*******thủ tục tính biểu thức dạng RPN*********}
PROCEDURE EvaluateRPN(rpn :expression; OperatorSet: CharacterSet);
  var   Stack: StackType;          {ngăn xếp toán hạng}
     numchar : Expression;
  i, numlen, num1, num2, num, number, code: integer; {* numlen là độ dài số đang đọc *}
  Token: char;              {ký tự trong biểu thức}
  TopSymbol: char;
  error : boolean;          {có lỗi trong biểu thức}
  Procedure StringToNum;
  Begin
       Val(numchar, number, code);
       numchar := '';  numlen := 0;
  End;
  Procedure Compute;
  Begin
     Pop(Stack, num1);
     Pop(Stack, num2);
     case Token of
        "+": num := num2 + num1;
        "-": num := num2 - num1;
        "*": num := num2 * num1;
        "/": num := num2 / num1;
     end
     Push(Stack, num);
  End;
begin {EvaluateRPN}
  write('Biểu thức RPN là:');
  Create(Stack);
  error := false; i := 1;
  Token := rpn[1];
  numchar := '';  numlen := 0;
  while (Token <> EndMark) and not error do
     begin {while ...}
        if  (rpn[i] = ' ') and (numlen > 0) then
           begin
              StringToNum;
              Push(Stack, number);
              i := i + 1;
```

```
            end;
         {bỏ qua dấu trống}
         while rpn[i] = ' '  do
            i := i+1;
         Token := rpn[i];
         if  Token IN OperatorSet then
            begin
               if  numlen > 0 then
                  begin
                     StringToNum;
                     Push(Stack, num);
                  End;
               Compute;
            end
         else
            begin
               numlen := numlen+1; numchar[numlen] := Token;
            end;
         i := i+1;
      end;{ while (Token <> EndMark)}
   if  empty(Stack) or (Stack.Top > 1)
      writeln('lỗi biểu thức')
   else
      writeln('kết quả là : ', Stack.Element[Top]);
 end;{EvaluateRPN}
BEGIN {chương trình chính}
   {khởi động tập các toán tử số học}
   OperatorSet := ['+', '-', '*', '/'];
   NumericSet := ['0','1','2','3','4','5','6','7','8','9'];
   writeln('Đưa ', EndMark, 'vào cuối mỗi biểu thức RPN');
   writeln;
   repeat
      write('Biểu thức RPN ?');
      i := 0;
      repeat
            i := i+1;
            readln(rpn[i])
      until rpn[i] = EndMark;
      readln;
      EvaluateRPN(rpn, OperatorSet);
      writeln;
      write('Tiếp tục (Y  hay N) ?');
      readln(response)
   until not (response IN ['Y', 'y'])
END.
```

◊ Trong C:

```
#define MaxExpression  80  //chiều dài tối đa của biểu thức
#define StackLimit     50   //kích thước ngăn xếp
#define EndMark        ';'       //dấu kết thúc biểu thức RPN
typedef  int ElementType ;
typedef  Expression    string[MaxExpression];
struct    StackType
   {
      int Top;
      int Element[StackLimit];
   };
Expression rpn;         //biểu thức RPN
char Response;
void Create(struct StackType *);//thủ tục tạo ngăn xếp rỗng
int Empty(struct StackType);// hàm ngăn xếp rỗng
void Push(struct StackType *, ElementType);// thủ tục đẩy Item vào đỉnh ngăn xếp
void Pop(struct StackType *, ElementType *);// thủ tục lấy Item từ đỉnh ngăn xếp
int Compute(char , int , int );// hàm tính toán
void StringToNum(Expression *, int *, int *); đổi dãy thành số
int LaKySo(char);// hàm kiểm tra chữ số
int LaToanTu(char);// hàm kiểm tra toán tử
void EvaluateRPN(Expression);
int main()
   {
      int i;
      printf("\nĐưa vào cuối biểu thức RPN  dấu  %c \n",  EndMark);
      do
      {
         printf("Biểu thức RPN ?");
         i = -1;
         do
            {
               i++;
               rpn[i] = getchar();
            }
         while (rpn[i] == EndMark);
         getch();
         EvaluateRPN(rpn);
         printf("\n");
         printf("Tiếp tục (Y  hay N) ?");
         response = getchar();
      }
      while  ((response == "Y") || response == "y"));
   }
void Create(struct StackType *Stack)
    {
```

```
        Stack->Top = -1;
    }
int Empty(struct StackType Stack)
  {
        if  (Stack.Top == -1)
            return 1;
        else
            return 0;
  }
void Pop(struct StackType *Stack, ElementType *Item)
  {
    *Item = Stack->Element[Stack->Top--];
  }
void Push(struct StackType *Stack, ElementType Item)
  {
    Stack->Element[++(Stack->Top)]=Item;
  }
int Compute(char operator, int num1, int num2)
  {
     switch (operator)
     {
        case '+':
            return (num2 + num1); break;
        case '–':
            return (num2 – num1); break;
        case '*':
            return (num2 * num1); break;
        case '/':
            return (num2 / num1); break;
     }
  }
void StringToNum(Expression *numchar, int *number, int *numlen)
  {
     *number = atoi(*numchar);
     *numchar = "";
     *numlen = 0;
  }
int LaToanTu(char x)
{
  if  ((x =="+") || (x =="–") ||(x =="*") ||(x =="/"))
     return 1;
  else
     return 0;
}
int LaKySo(char x)
{
```

```
    if ((x >= "0") && (x <="9"))
        return 1;
    else
        return 0;
}
int  EvaluateRPN(Expression rpn)
{
    StackType Stack;          //ngăn xếp toán hạng
    Expression numchar; //dãy chữ số đang đọc
    int    i, numlen, num1, num2, num, number; // numlen là độ dài số đang đọc
    char Token;               //ký tự trong biểu thức
    Create(&Stack);
    i = 0;
    Token = rpn[0];
    numchar = "";
    numlen = 0;
    while (Token != EndMark)
        {
            if ((rpn[i] == " ") && (numlen > 0))
                {
                    StringToNum( &numchar, &number, &numlen);
                    Push(&Stack, number);
                    i++ ;
                }
            //bỏ qua dấu trống
            while (rpn[i] == " ")
                i++;
            Token = rpn[i];
            if (LaToanTu(Token))
                {
                    if (numlen > 0)
                        {
                            StringToNum(&numchar, &number, &numlen);
                            Push(&Stack, number);
                        }
                    Pop(&Stack, &num1);
                    Pop(&Stack, &num2);
                    number = Compute(Token, num1, num2);
                    Push(&Stack, number);
                }
            else
                if (LaKySo(Token))
                    {
                        numlen = numlen+1; numchar[numlen] = Token;
                    }
                else
```

```
                { printf("lỗi biểu thức"); exit(1);}
            i++ ;
        }  // while (Token != EndMark)
        if  (empty(Stack) || (Stack.Top > 0))
            printf("lỗi biểu thức");
        else
            printf("kết quả là : %d ", Stack->Element[Top]);
} //EvaluateRPN
```

• *Thuật toán chuyển từ dạng trung tố sang dạng RPN*

(1) Khởi động ngăn xếp rỗng của các toán tử và dấu ngoặc.

(2) Lặp lại các bước sau cho đến khi gặp dấu kết thúc biểu thức được đọc hoặc gặp lỗi:

(a) Đọc phần tử (toán hạng, toán tử, dấu ngoặc "(", ")")

(b) Nếu phần tử là

(i) dấu ngoặc trái "(" : đẩy nó vào ngăn xếp.

(ii) dấu ngoặc phải ")" : lấy ra và hiển thị các phần tử của ngăn xếp cho đến khi dấu ngoặc trái "(" được đọc. (nếu không gặp dấu ngoặc trái mà ngăn xếp rỗng thì biểu thức có lỗi).

(iii) Toán tử : Lặp lại quá trình sau cho đến khi đưa được Toán tử vào ngăn xếp:

+ Nếu ngăn xếp rỗng hay toán tử được ưu tiên hơn toán tử hay phần tử ở đỉnh ngăn xếp, thì đẩy toán tử vào ngăn xếp.

+ Ngược lại lấy phần tử ra khỏi ngăn xếp và hiển thị nó (Ghi chú: dấu ngoặc trái được xem là có độ ưu tiên thấp hơn các toán tử).

(iv) Toán hạng: hiển thị nó.

(3) Khi gặp dấu kết thúc biểu thức trung tố, lấy ra và hiển thị các phần tử của ngăn xếp cho đến lúc ngăn xếp rỗng.

◊ *Ví dụ:* Chuyển biểu thức 2 * (3 + 4) về dạng RPN.

Biểu thức	Ngăn xếp	Chú thích	Biểu thức RPN
2 * (3 + 4) ↑	⌊⌋ rỗng	hiển thị 2 →	2
* (3 + 4) ↑	⌊*⌋ ←đỉnh	Đẩy * vào ngăn xếp	2
(3 + 4)	⌊(⌋ ←đỉnh	Đẩy "(" vào ngăn xếp	2

↑	| * |			
3 + 4) ↑	| (| | * |	←đỉnh	hiển thị 3 →	2 3
+ 4) ↑	| + | | (| | * |	←đỉnh	Đẩy + vào ngăn xếp	2 3
4) ↑	| + | | (| | * |	←đỉnh	hiển thị 4 →	2 3 4
) ↑	| + | | (| | * |		lấy ra và hiển thị các phần tử ngăn xếp cho đến khi gặp dấu "("	2 3 4 +
	| * |		lấy ra và hiển thị hết các phần tử ngăn xếp	2 3 4 + *

Như vậy dạng RPN của biểu thức 2 * (3 + 4) là 2 3 4 + *

Chương trình sau cài đặt giải thuật chuyển đổi biểu thức trung tố sang dạng RPN.

◊ Trong PASCAL:

```
Program  InfixToPRN;
CONST
  MaxExpression  = 80 {chiều dài tối đa của biểu thức}
  StackLimit     = 50 {kích thước ngăn xếp}
  EndMark        = ';'     {dấu kết thúc biểu thức trung tố}
TYPE
  ElementType  = char;
  Expression   = String;
  Stackarray = ARRAY[1 .. StackLimit] Of  ElementType;
  CharacterSet =  SET Of char;
  StackType  = RECORD
                  Top: 0 .. StackLimit;
                  Element: Stackarray
               END;
VAR
  infix : Expression;          {biểu thức trung tố}
  OperatorSet : CharacterSet;  {tập hợp toán tử}
  i : integer;
  Response : char;
{**********thủ tục tạo ngăn xếp rỗng**************}
PROCEDURE Create(Var Stack : StackType);
```

```
begin
  Stack.Top := 0
end;
{**********hàm ngăn xếp rỗng**************}
FUNCTION  Empty(Stack : StackType): boolean;
begin
  Empty := (Stack.Top := 0)
end;
{**********thủ tục lấy Item từ đỉnh ngăn xếp không rỗng**************}
PROCEDURE  Pop(Var Stack : StackType; var Item: ElementType);
begin
  with Stack do
     begin
        Item := Element[Top];
        Top := Top - 1
     end
end;
{**********thủ tục đẩy Item vào đỉnh ngăn xếp không đầy**************}
PROCEDURE  Push(Var Stack : StackType; Item: ElementType);
begin
  with Stack do
     begin
        Top := Top + 1;
        Element[Top] := Item;
     end;
end;
{********thủ tục chuyển biểu thức trung tố infix sang dạng RPN*********}
PROCEDURE ConvertToRPN(infix : expression; OperatorSet: CharacterSet);
var Stack: StackType;          {ngăn xếp toán tử}
  i: integer;
  Token: char;              {ký tự trong biểu thức}
  TopSymbol: char;
  error : boolean;          {có lỗi trong biểu thức}
  DonePopping: boolean; {việc lấy ra từ ngăn xếp hoàn tất}
  {*** Hàm tìm độ ưu tiên của toán tử hay dấu ***}
     FUNCTION Priority(Operator : char) : integer;
     begin
        case Operator of
           '('       : Priority := 0;
           '+', '-'  : Priority := 1;
           '*', '/'  : Priority := 2;
        end
     end;
begin {ConvertToRPN}
  write('Biểu thức RPN là:');
  Create(Stack);
```

```
error := false;
i := 1;              {bắt đầu chuyển qua dạng RPN}
Token := infix[1];
while (Token <> EndMark) and not error do
   begin {while ...}
      {bỏ qua dấu trống}
      while infix[i] = ' '  do
            i := i+1;
      Token := infix[i];
      if Token = '(' then          {dấu ngoặc trái}
            Push(Stack, Token)
      else
         if  Token = ')'  then        {dấu ngoặc phải}
            begin { if  Token = ')'}
               DonePopping := false;
               repeat
                     if empty(stack) then
                        error := true
                     else
                        begin{else}
                           Pop(stack, TopSymbol);
                           if TopSymbol <> '('  then
                              write(TopSymbol : 2)
                           else
                              DonePopping := true
                        end{else}
               until DonePopping or error
            end { if  Token = ')'}
         else
            if  Token  IN  OperatorSet  THEN  {toán tử số học}
               begin { if  Token  IN  OperatorSet...}
                  DonePopping := false;
                  while not empty(stack) and not DonePopping do
                     begin { while not empty(stack)...}
                        Pop(Stack, TopSymbol}
                        if Priority(Token) <= Priority(TopSymbol)
                           then  write(TopSymbol : 2)
                        else
                           begin
                              push(stack, TopSymbol);
                              DonePopping := true;
                           end {else}
                     end { while not empty(stack) ...}
                  push(stack, Token)
               end { if  Token  IN  OperatorSet ...}
            else
```

```
                    if  Token <> EndMark then {toán hạng}
                        write(Token : 2);
                i := i+1;
            end;{ while (Token <> EndMark)}
        while not empty(stack) and not error do
            begin
                Pop(stack, TopSymbol);
                if TopSymbol <> '('  then
                        write(TopSymbol : 2)
                else
                        error := true
            end; {while...}
        if error then
                writeln('<< có lỗi trong biểu thức trung tố >>')
        else
                writeln;
      end;{convertToRPN}
    BEGIN {chương trình chính}
        {khởi động tập các toán tử số học}
        OperatorSet := ['+', '-', '*', '/'];
        writeln('Đưa ', EndMark, 'vào cuối mỗi biểu thức trung tố');
        writeln;
        repeat
            write('Biểu thức trung tố ?');
            i := 0;
            repeat
                i := i+1;
                readln(infix[i])
            until infix[i] = EndMark;
            readln;
            ConvertToRPN(infix, OperatorSet);
            writeln;
            write('Tiếp tục (Y  hay N) ?');
            readln(response)
        until not (response IN ['Y', 'y']
    END.
```

◊ Trong C:

```
#define  MaxExpression  80  //chiều dài tối đa của biểu thức
#define  StackLimit     50   //kích thước ngăn xếp
#define  EndMark        ';'        //dấu kết thúc biểu thức trung tố
typedef  int ElementType ;
typedef  Expression    string[MaxExpression];
struct   StackType
    {
```

```
        int Top;
        int Element[StackLimit];
    };
Expression infix;           //biểu thức trung tố
int i ;
char  Response ;
void Create(struct StackType *);//thủ tục tạo ngăn xếp rỗng
int Empty(struct StackType);// hàm ngăn xếp rỗng
void Push(struct StackType *, ElementType);// thủ tục đẩy Item vào đỉnh ngăn xếp
void Pop(struct StackType *, ElementType *);// thủ tục lấy Item từ đỉnh ngăn xếp
void StringToNum(StackType *, Expression *, int *);//đổi dãy thành số
int LaKySo(char);// hàm kiểm tra chữ số
int LaToanTu(char);// hàm kiểm tra toán tử
int  Priority(char ); // hàm độ ưu tiên toán tử và dấu "("
void ConvertToRPN(Expression infix);
int main()
{
        printf("Đưa dấu %c  vào cuối mỗi biểu thức trung tố", EndMark);
        printf("\n");
        do
        {
            printf("Biểu thức trung tố ?");
            i = 0;
            do
            {
                i++;
                infix[i] = getchar();
            }
            while (infix[i] != EndMark)
            getch();
            ConvertToRPN(infix);
            printf("\n");
            printf("Tiếp tục (Y  hay N) ?");
            response = getchar();
        }
        while  ((response == "Y") || (response == "y"))
}
//  thủ tục
void Create(struct StackType *Stack)
     {
          Stack->Top = -1;
     }
int Empty(struct StackType Stack)
    {
          if  (Stack.Top == -1)
              return 1;
```

```
        else
            return 0;
    }
void Pop(struct StackType *Stack, ElementType *Item)
    {
      *Item = Stack->Element[Stack->Top--];
    }
void Push(struct StackType *Stack, ElementType Item)
    {
     Stack->Element[++(Stack->Top)]=Item;
    }
int LaToanTu(char x)
{
    if  ((x ==”+”) || (x ==”–”) ||(x ==”*”) ||(x ==”/”))
        return 1;
    else
        return 0;
}
int LaKySo(char x)
{
    if  ((x >= “0”) && (x <=”9”))
        return 1;
    else
        return 0;
}
int  Priority(char Operator)
    {
        switch Operator
        {
            case  ‘(‘ :
                return 0; break;
            case  ‘+‘   :
                return 1; break;
            case  ‘-‘ :
                return 1; break;
            case  ‘*‘   :
                return 2; break;
            case  ‘/‘ :
                return 2; break;
        }
    }
void ConvertToRPN(Expression infix) //  chuyển biểu thức trung tố  sang dạng RPN
{
     StackType Stack;          //ngăn xếp toán tử
     int  i, error;
     char Token ;              //ký tự trong biểu thức
```

```
char TopSymbol;
int  DonePopping;   //việc lấy ra từ ngăn xếp hoàn tất
printf("Biểu thức RPN là:");
Create(&Stack);
i = 0;              //bắt đầu chuyển qua dạng RPN
Token = infix[1];
error = 0;
while ((Token != EndMark) && !error)
 {
       //bỏ qua dấu trống
       while (infix[i] == " ")
            i++;
       Token = infix[i];
       if (Token == "("          //dấu ngoặc trái
            Push(&Stack, Token);
       else
          if  (Token == ")")      //dấu ngoặc phải
            {
               DonePopping = 0;
               do
                  {
                     if (empty(Stack))
                        error = 1;
                     else
                        {
                           Pop(&Stack, &TopSymbol);
                           if (TopSymbol != "(")
                              printf(" %d", TopSymbol);
                           else
                              DonePopping = 1;
                        }
                  }
               while (!DonePopping && !error)
            }
         else
            if  (LaToanTu(Token))
               {
                  DonePopping = 0;
                  while (!empty(stack) && !DonePopping)
                     {
                        Pop(&Stack, &TopSymbol);
                        if (Priority(Token) <= Priority(TopSymbol))
                           printf(" %c", TopSymbol);
                        else
                           {
                              Push(&Stack, TopSymbol);
```

```
                                DonePopping = 1;
                            }
                        } // while (!empty(stack) && !DonePopping)
                    Push(&Stack, Token)
                } // if  LaToanTu(Token)
            else
                if  (Token != EndMark) //toán hạng
                    printf(" %d", Token);
        i++;
    } // while (Token != EndMark)
    while (!empty(Stack) && !error)
    {
        Pop(&Stack, TopSymbol);
        if (TopSymbol != "(")
            printf(" %d", TopSymbol);
        else
            error = 1;
    }
    if (error)
        printf("<< có lỗi trong biểu thức trung tố >>");
    else
        printf("\n");
}
```

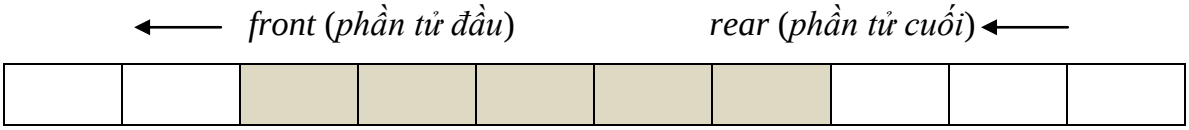
II. HÀNG ĐỢI

1. Định nghĩa

Hàng đợi (*queue*) là danh sách mà phép toán thêm vào thực hiện ở đầu này và phép loại bỏ thực hiện ở đầu kia. Hàng đợi tuân thủ nguyên tắc *Vào trước Ra trước* hay còn gọi là *Nguyên tắc FIFO* (*First In First Out*).

2. Tổ chức hàng đợi theo danh sách đặc

Sử dụng danh sách đặc để cài đặt hàng đợi có thể minh họa như sau:



a. Khai báo

Trong PASCAL	Trong C
Const ElemNo = <số phần tử tối đa của hàng đợi >; *Type InfoType = <kiểu dữ liệu>;* *Element = record* *info : InfoType;* *...* *end;* *Var queue : array*[1.. *ElemNo*] *of Element; {queue}* *front, rear : integer; { con trỏ trước, con trỏ sau của hàng đợi }*	*#define ElemNo <số phần tử tối đa của hàng đợi>* *typedef < kiểu dữ liệu phần tử> InfoType;* *struct Element {* *InfoType info;* *...* *};* *struct Element queue*[*ElemNo*+1]; *int front, rear; // con trỏ trước, con trỏ sau của hàng đợi*

• *Khắc phục hàng đợi bị tràn*

Hàng đợi bị *đầy* là tất cả các ô đã được sử dụng.

Hàng đợi bị *tràn* là còn ô trống nhưng không thêm phần tử mới được vì *rear* đã tịnh tiến đến cuối hàng đợi. Để khắc phục hàng bị tràn ta sử dụng các quy tắc sau:

◊ *Quy tắc di chuyển tịnh tiến:*

Nếu con trỏ sau *rear* = *ElemNo*, thì tịnh tiến hàng đợi lên trước.

◊ *Quy tắc di chuyển vòng:*

Nếu con trỏ sau *rear* = *ElemNo* và còn ô trống, thì thêm phần tử mới vào ô thứ nhất.

b. Khởi tạo

Trong PASCAL

```
Procedure Initialize;
begin
    front := 0;
    rear := 0
end;
```

Trong C

```
void Initialize()
{
    front = 0;
    rear = 0;
}
```

c. Thêm phần tử vào hàng đợi

Hàm *Push* trả về vị trí của phần tử mới hoặc trả về 0 nếu hàng đợi bị đầy.

- *Phương pháp di chuyển tịnh tiến*

Ta luôn có *front* ≤ *rear*, số phần tử của hàng đợi là

$$rear - front + 1$$

Hàng đợi đầy khi *front* = 1 và *rear* = *ElemNo*.

Trong PASCAL

```
function Push(NewElement: Element) integer;
var i : integer;
begin
    if rear - front + 1 = ElemNo then
      Push := 0 {hàng đầy }
   else
      begin
        if front = 0 then
          begin
            front := 1;
            rear := 0;
          end;
        if rear = ElemNo then {dồn
           hàng lên đầu}
          begin {dời hàng lên (front-1)
                 vị trí}
            for i := front to rear do
            queue[i-front+1]:=queue[i];
            rear := ElemNo - front +1;
            front := 1;
          end;
        rear := rear + 1;
        queue[rear] := NewElement;
        Push := 1;
      end;
end;
```

Trong C

```
int Push(Element NewElement)
{
   int i;
   if (rear - front + 1 == ElemNo)
      return 0; //hàng đầy
   else
      {
        if (front == 0)
          {
            front = 1;
            rear = 0;
          }
        if (rear == ElemNo) //dồn hàng
        lên đầu
          { //dời hàng lên (front -1) vị trí
            for (i= front; i <= rear; i++)
              queue[i-front+1]= queue[i];
            rear = ElemNo - front + 1;
            front = 1;
          }
        rear = rear + 1;
        queue[rear] = NewElement;
        return 1;
      }
}
```

• *Phương pháp di chuyển vòng*

Với phương pháp này *front* có thể nhỏ hơn, lớn hơn hoặc bằng *rear*.

Hàng đợi đầy khi

$$rear - front + 1 = 0$$

hoặc khi

$$rear - front + 1 = ElemNo.$$

Trong PASCAL

```
Procedure Push(NewElement: Element);
var i : integer;
begin
   if (rear - front + 1 = 0) or (rear - front + 1 = ElemNo) then
      Push := 0 {hàng đầy }
   else
      begin
         if front = 0 then
            begin
               front := 1;
                rear := 0
            end;
         if rear = ElemNo then {quay vòng}
            rear := 0;
         rear := rear + 1;
         queue[rear] := NewElement;
      end;
end;
```

Trong C

```
void Push(Element NewElement)
{
   int i;
   if ((rear - front + 1 == 0) || (rear - front + 1 == ElemNo)) then
      return 0; //hàng đầy
   else
   {
      if (front == 0)
      {
         front = 1;
         rear = 0;
      }
      if (rear == ElemNo)// quay vòng
         rear = 0;
      rear = rear + 1;
      queue[rear] = NewElement;
   }
}
```

d. Lấy phần tử ra khỏi hàng đợi

Thủ tục *Pop* có 1 tham biến: *ElementDel* chứa phần tử bị xoá.

• *Di chuyển tịnh tiến*

Trong PASCAL

```
Procedure Pop(var ElementDel : Element);
begin
   if front <> 0 then
      begin
         ElementDel := queue[front];
         front := front + 1;
         if front > rear then
            begin
               front := 0;
               rear := 0
            end;
```

Trong C

```
void Pop(Element *ElementDel)
{
   if (front != 0)
      {
         ElementDel = queue[front];
         front = front + 1;
         if (front > rear)
            {
               front = 0;
               rear = 0;
            }
```

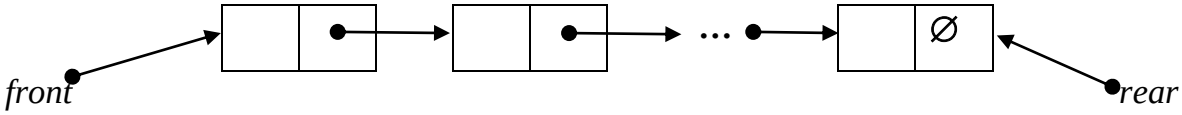
end; *end;*	} }

• *Di chuyển vòng*

Trong PASCAL	Trong C
Procedure Pop(var ElementDel : Element); *begin* *if front <> 0 then* *begin* *ElementDel := queue[front];* *if front = rear then* *begin* *front := 0;* *rear := 0* *end* *else* *begin* *front := front + 1;* *if front > ElemNo then* *front := 1;* *end;* *end;* *end;*	*void Pop(Element *ElementDel)* { *if (front != 0)* { **ElementDel = queue[front];* *if (front == rear)* { *front = 0;* *rear = 0;* } *else* { *front = front + 1;* *if (front > ElemNo)* *front = 1;* } } }

3. Tổ chức hàng đợi theo danh sách liên kết

Sử dụng danh sách liên kết để cài đặt hàng đợi có thể minh họa như sau:



a. Khai báo

Trong PASCAL	Trong C
Type InfoType = <kiểu dữ liệu>; *NodePointer = ^Node;* *Node = record* *info : InfoType;* *next : NodePointer;* *end;* *Var front, rear : NodePointer; {queue pointer}*	*typedef <kiểu dữ liệu> InfoType;* *struct Node {* *InfoType info;* *struct Node *next;* *};* *typedef struct Node *NodePointer;* *NodePointer front, rear;// queue pointer*

b. Khởi tạo hàng đợi rỗng

Trong PASCAL

```
Procedure Initialize;
begin
    front := nil;
    rear := nil
end;
```

Trong C

```
void Initialize()
{
    front = NULL;
    rear  =  NULL;
}
```

c. Thêm phần tử vào hàng đợi

Trong PASCAL

```
Procedure  Push(NewInfo: InfoType);
var p : NodePointer;
begin
        new(p);
        p^.info := NewInfo;
        p^.next := nil;
        if  rear = nil then
            front := p
        else
            rear^.next := p;
        rear := p;
end;
```

Trong C

```
void  Push(InfoType  NewInfo)
{
   NodePointer p;
   p=(NodePointer)malloc(sizeof(struct
      Node));
   p->info = NewInfo;
   p->next = NULL;
   if  (rear == NULL)
      front = p;
   else
      rear->next = p;
   rear = p;
}
```

d. Lấy phần tử ra khỏi hàng đợi

Trong PASCAL

```
Procedure  Pop(var InfoDel : InfoType);
   var p : NodePointer;
   begin
      if  front <>  nil then
         begin
            p := front;
            front := p^.next;
            InfoDel := p^.info;
            if  front = nil then
               rear := nil;
            dispose(p)
         end;
   end;
```

Trong C

```
   void  Pop(InfoType *InfoDel)
   {
      NodePointer p;
      if  (front != NULL)
      {
         p = front;
         front = p->next;
         *InfoDel = p->info;
         if  (front == NULL)
            rear = NULL;
         free(p);
      }
   }
```

◊ *Ghi chú*. Gọi thủ tục *Pop*(*&InfoDel*)

4. Ứng dụng. ***Thuật toán đảo ngược số nguyên dương***

Thuật toán này đảo ngược số nguyên dương *Number*.

(*i*) Lặp lại các bước sau cho đến khi *Number* = 0:

a. Tính số dư *Remainder* của phép chia *Number* cho 10

b. Đẩy số dư *Remainder* vào hàng đợi

c. Thay *Number* bằng phần nguyên của kết quả phép chia *Number* cho 10

(*ii*) Lặp lại các bước sau cho đến khi hàng đợi số dư rỗng:

a. Lấy ra *Remainder* từ hàng đợi

b. Hiển thị *Remainder*

Các chương trình sau cài đặt giải thuật đảo ngược số nguyên dương.

Trong PASCAL

```
Type nodeptr = ^node;
node = record
            v integer ;/*dinh ke*/
            next  nodeptr;
        end;
Var
   Number, c : integer;
   front, rear : nodeptr;
Procedure  Push(n: integer);
var p : nodeptr;
begin
    new(p);
    p^.v := n;
    p^.next := nil;
    if  (rear = nil) then
       front := p
    else
        rear^.next := p;
    rear := p;
end;
procedure  Pop(var n: integer);
var p : nodeptr;
begin
    p := front;
    front := p^.next;
    n := p^.v;
    if  (front = nil) then
       rear := nil;
    dispose(p);
end;
```

Trong C

```
struct  node   {
        int  v;/*dinh ke*/
        struct node  *next;
        };
typedef  struct node *nodeptr;
int   Number, c;
nodeptr front, rear;
void Pop(int *);
void Push(int );
main()
{
  printf("nhap so = ");
  scanf("%d", &Number);
  front = NULL; rear = NULL;
  while (Number > 0)
   {
      c = Number % 10;
      Push(c);
      Number = Number / 10;
   }
   while (front != NULL)
   {
      Pop(&c);
      printf("%d",c);
   }
  getch();
}
void  Push(int n)
{
    nodeptr p;
```

```
BEGIN
  write("nhap so = ");
  read(Number);
  front := nil; rear := nil;
  while (Number > 0) do
   begin
       c := Number MOD 10;
       Push(c);
       Number := Number DIV 10;
   end;
   while (front <> nil) do
   begin
       Pop(c);
       write(c);
   end;
  readln;
END.
```

```
    p = (nodeptr)malloc(sizeof(struct node));
    p->v = n;
    p->next = NULL;
    if  (rear == NULL)
       front = p;
    else
        rear->next = p;
    rear = p;
}
void Pop(int *n)
{
    nodeptr p;
    p = front;
    front = p->next;
    *n = p->v;
    if  (front == NULL)
       rear = NULL;
    free(p);
}
```

BÀI TẬP

1. Viết chương trình nhập vào một số nguyên dương và sử dụng ngăn xếp chuyển đổi thành số nhị phân.

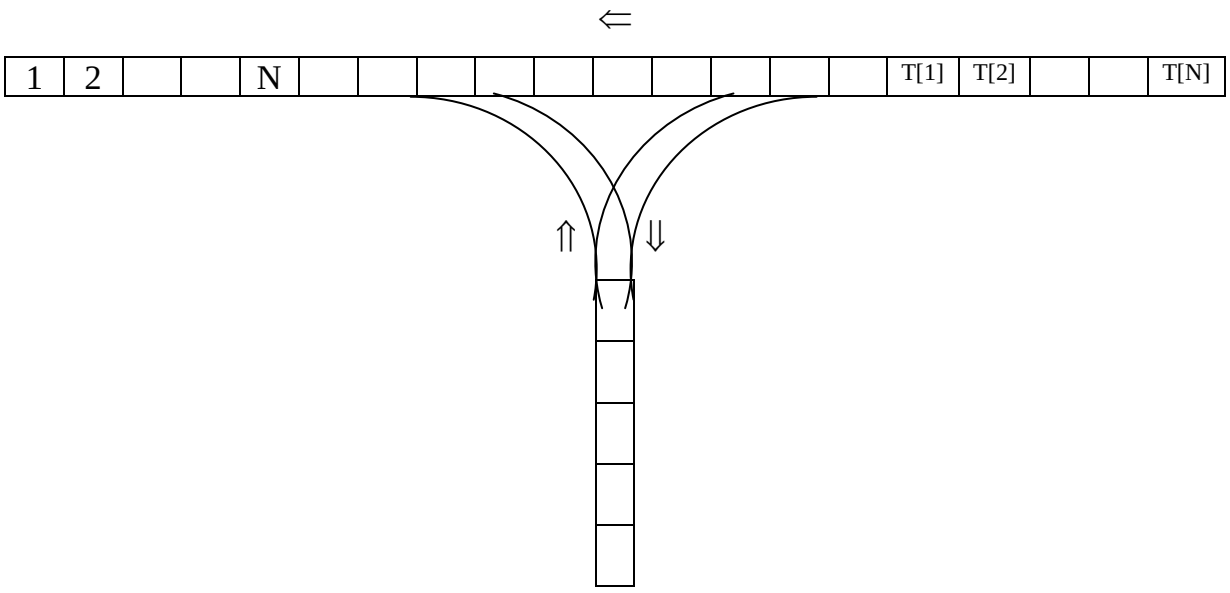
2. Viết chương trình nhập vào một số nguyên dương và sử dụng ngăn xếp tìm các thừa số nguyên tố (theo thứ tự giảm dần) của nó.

3. Viết chương trình nhập vào một số nguyên dương và sử dụng hàng đợi hiển thị số đảo ngược của nó.

4. Viết chương trình sử dụng ngăn xếp chuyển đổi biểu thức trung tố về biểu thức nghịch đảo Ba Lan.

5. Viết chương trình sử dụng ngăn xếp đánh giá biểu thức nghịch đảo Ba Lan.

6. Cho một đoàn tàu N toa được đánh số từ 1 đến N. Tuy nhiên thứ tự sau trước của các toa đã bị xáo trộn. Để dồn toa theo thứ tự số toa người ta dùng một đường ray phụ (xem hình).



Các toa tàu có thứ tự xáo trộn ở bên phải dược dồn sang trái từng toa và sắp xếp lại theo thứ tự 1, 2, 3, …, N. Các toa chỉ được di chuyển theo chiều mũi tên. Đường ray phụ chỉ chứa được tối đa M toa ($M \leq N$).

Hãy viết chương trình cài đặt thuật toán dồn toa với dữ liệu đầu vào:

Vị trí các toa bên phải cho bởi mảng T:

T[i] là số của toa đứng ở vị trí thứ i, i = 1, 2, …, N

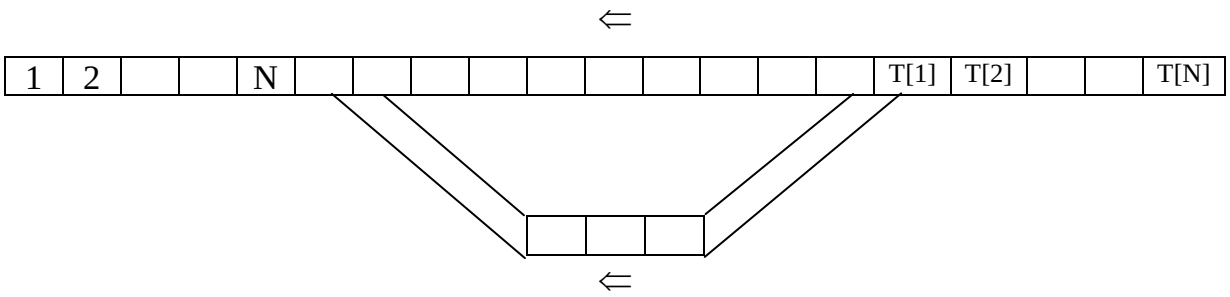
Kết quả đầu ra là :

(1) Kết luận dồn toa được kèm theo cách di chuyển từng toa hoặc

(2) Kết luận không dồn toa được.

7. Một câu lạc bộ sắp xếp các cặp nhảy theo nguyên tắc sau: người nam đến đầu tiên ghép với người nữ đến đầu tiên, người nam đến thứ hai ghép với người nữ đến thứ hai, … Viết chương trình cài đặt hàng đợi những người khiêu vũ (nhập tên và giới tính), sau đó xuất ra màn hình từng cặp khiêu vũ và những người còn dư.

8. Cho một đoàn tàu N toa được đánh số từ 1 đến N. Tuy nhiên thứ tự sau trước của các toa đã bị xáo trộn. Để dồn toa theo thứ tự số toa người ta dùng một đường ray phụ (xem hình).



Các toa tàu có thứ tự xáo trộn ở bên phải dược dồn sang trái từng toa và sắp xếp lại theo thứ tự 1, 2, 3, …, N. Các toa chỉ được di chuyển theo chiều mũi tên. Đường ray phụ chỉ chứa được tối đa M toa ($M \leq N$).

Hãy viết thuật toán dồn toa với dữ liệu đầu vào:

Vị trí các toa bên phải cho bởi mảng T:

T[i] là số của toa đứng ở vị trí thứ i, i = 1, 2, …, N

Kết quả đầu ra là :

(1) Kết luận dồn toa được kèm theo cách di chuyển từng toa hoặc

(2) Kết luận không dồn toa được.

9. Viết chương trình cài đặt hàng đợi có ưu tiên giúp theo dõi trình tự xử lý đơn khiếu nại. Đơn khiếu nại nào có độ ưu tiên cao hơn sẽ được giải quyết trước. Nếu các đơn có cùng độ ưu tiên thì đơn nào đến trước sẽ giải quyết trước. Chương trình có các chức năng sau:

i) Nhập đơn: tên đơn và độ ưu tiên.

ii) Giải quyết đơn: in ra màn hình tên đơn giải quyết.

iii) Xem đơn: in ra màn hình các đơn chưa xử lý.