

មេរៀនទី១

មូលដ្ឋានគ្រឹះរបស់ Data Structures និង Algorithms

1-1. សេចក្តីផ្តើម

មេរៀននេះណែនាំអោយយើងស្វែងយល់អំពីបញ្ហា Data Structures និង Algorithms និងអំពីតួនាទី ហើយទំនាក់ទំនងរបស់វានៅក្នុងកម្មវិធី និងការស្វែងយល់អំពី Structure Language ដែលបានយកមកប្រើសំរាប់អនុវត្តលើ Structure របស់ Algorithms ។

រាល់ចំណោទបញ្ហាដែលត្រូវដោះស្រាយជាមួយម៉ាស៊ីនចាំបាច់ត្រូវស្វែងយល់អំពី Data Structures និង Algorithms ជាមុនសិនពីព្រោះ Data Structures និង Algorithms ជាមូលដ្ឋានរបស់កម្មវិធី។ យើងមាន Structure ដែលបានបង្កើតឡើងដោយលោក Nicklaus Wirth បានសរសេរ **Data Structures + Algorithms = Programs** មានន័យថា:

ចំណោទបញ្ហា → រកដំណោះស្រាយ និងរៀបចំរបៀបដោះស្រាយ → សរសេរកម្មវិធី។

* **Programs** គឺជាសំនុំមួយសំរាប់ផ្ទុកនូវរាល់គំនិតយោបល់របស់អ្នកសរសេរកម្មវិធីទៅតាមប្រព័ន្ធកំណត់របស់ម៉ាស៊ីនដែលក្រោយពីការអនុវត្តទៅតាមដំណើរការរបស់ម៉ាស៊ីនយើងនឹងទទួលបានលទ្ធផលដែលចង់បាន។

* **Data Structures** គឺជាការរៀបចំទិន្នន័យរបស់អ្នកសរសេរកម្មវិធីទៅតាមកូដរបស់ម៉ាស៊ីន និងរាល់ទិន្នន័យដែលយើងអោយ រួចទាញយកមកប្រើប្រាស់វិញក្នុងគោលបំណងអ្វីមួយ។

* **Algorithms:** គឺជាជំហាននៃការដោះស្រាយបញ្ហាទៅតាមដំណាក់កាលៗ ជា Flowchart និងម៉្យាងទៀត Algorithms គឺជាខ្សែ Statement ដ៏ច្បាស់លាស់ដែលក្រោយពីការកំណត់ជាក់លាក់យើងទទួលបានលទ្ធផលដែលយើងចង់បាន។ យើងមាន Structure ទូទៅសំរាប់ស្វែងយល់អំពីបញ្ហា Algorithms បន្ថែមទៀតដែលត្រូវបានបង្កើតឡើងដោយលោក Kowalski ដែលជាអ្នកបង្កើតកម្មវិធី Logic ត្រូវបានសរសេរដូចខាងក្រោម:

$$\text{Algorithms} = \text{Logic} + \text{Control}$$

• **Logic** គឺជាចំណេះដឹងមូលដ្ឋានប្រើសំរាប់កំណត់អត្ថន័យអោយ Algorithms ហើយផ្នែក Logic ជាអ្នកដឹងថា Algorithms ត្រូវធ្វើ។

• **Control** គឺជាវិធីសាស្ត្រដោះស្រាយលើផ្នែក Logic ហើយវាជាអ្នកដឹងថា Algorithms ត្រូវធ្វើដូចម្តេច។

ឧទាហរណ៍:

ចូរលើកឡើងអំពីលក្ខណៈ Logic និង Control របស់ចំណោទបញ្ហា $n!$ ។

យើងបាន:

-Logic របស់បញ្ហា $n! = 1 \times 2 \times 3 \dots \times n$

-Control របស់ $n!$ លើផ្នែក Logic នេះរួមមាន 3 របៀបគឺ:

-របៀបទី១

គុណបង្រួមបណ្តាលលេខពី 1 ដល់ n ដាក់ចូលក្នុងអថេរណាមួយ។

```
f = 1 ;  
for( i = 1 ; i <= n ; i ++ )  
    f = f * i ;
```

-របៀបទី២

គុណបង្រួមបណ្តាលលេខពី n ដល់ 1 ដាក់ចូលក្នុងអថេរណាមួយ។

```
f = 1 ;  
for ( i = n; i >= 1 ; i - - )  
    f = f * i ;
```

-របៀបទី៣

គុណបង្រួមបណ្តាលលេខសេសជាមួយគ្នាដាក់ចូលក្នុងអថេរណាមួយ និងគុណបង្រួមបណ្តាលលេខគូជាមួយគ្នាដាក់ចូលក្នុងអថេរណាមួយទៀតបន្ទាប់មកគុណលទ្ធផលទាំងពីរជាមួយគ្នាដាក់ក្នុងអថេរណាមួយ។

1-2.បញ្ហាទំនាក់ទំនងមួយចំនួនរបស់ **Data Structure**

•Data Structures គឺសំដែងអំពីដំណើរការនៅក្នុង Memory ដែលអោយឈ្មោះថា Storage Structure ។

•Predefine Data Structure គឺជាទម្រង់ដែលមានស្រាប់នៅក្នុងភាសាកម្មវិធីដូចជាទម្រង់ Array ។ ការជ្រើសរើស Predefine Data Structure សមស្របសំរាប់ដំណោះស្រាយបញ្ហានាំអោយមានដំណើរការ Algorithms មានល្បឿនលឿន និងផ្តល់លទ្ធផលត្រឹមត្រូវ។

• រាល់ចំណោទបញ្ហាដែលមានទ្រង់ទ្រាយតូចមានបរិមាណទិន្នន័យតិច និងរាល់ការធ្វើប្រមាណវិធីរបស់វាគ្រាន់តែប្រើប្រមាណវិធី $+$, $-$, $*$, $/$ និង Logic ដែលអោយឈ្មោះថា បញ្ហាបច្ចេកទេស។

• រាល់ចំណោទបញ្ហាដែលមានទ្រង់ទ្រាយធំ និងមានបរិមាណទិន្នន័យច្រើនហើយ រាល់ការធ្វើប្រមាណវិធីរបស់វាមិនគ្រាន់តែប្រើប្រមាណវិធី $+$, $-$, $*$, $/$ និង Logic ប៉ុណ្ណោះ ទេគឺត្រូវបានអនុវត្តជាមួយប្រមាណវិធី ថ្មីមួយចំនួនទៀតដូចជា បង្កើន, បន្ថែម, កាត់បន្ថយ, លុប, ស្វែងរក និងតំរៀបជាដើមដែលគេអោយឈ្មោះថាចំណោទបញ្ហា **Non Numerical Problem**។

1-3. តួនាទីនិងទំនាក់ទំនងគ្នារវាង Data Structures និង Algorithms

នៅពេលដោះស្រាយជាមួយម៉ាស៊ីនជាដំបូងយើងត្រូវគិតដល់បញ្ហា Data Structures និង Algorithms របស់ចំណោទបញ្ហានោះ។ Algorithms មានតួនាទីដំណើរការរបស់បញ្ហា រីឯ Data Structures មានតួនាទីដំណើរការដោយផ្ទាល់ជាមួយម៉ាស៊ីន ដើម្បីនាំដល់លទ្ធផលដែលយើងចង់បានរបស់ចំណោទបញ្ហា។

ទំនាក់ទំនងគ្នារវាង Data Structures និង Algorithms គឺមានទំនាក់ទំនងគ្នាយ៉ាងជិតស្និទ្ធជាមួយគ្នា មានន័យថានៅពេលយើងនិយាយដល់បញ្ហាទិន្នន័យត្រូវគិតថាទិន្នន័យនោះបានជះឥទ្ធិពលដល់ Algorithms ណាមួយហើយនៅពេលគិតដល់បញ្ហា Algorithms ត្រូវដឹងថា Algorithms នោះបានជះឥទ្ធិពលលើទិន្នន័យណាដើម្បីនាំដល់លទ្ធផលដែលយើងចង់បាន។

ឧទាហរណ៍:

ចូរលើកឡើងពីតួនាទីរបស់ Data Structures និង Algorithms នៅក្នុងចំណោទបញ្ហា

$$ax^2 + bx + c = 0$$

• Algorithms

+ ពិភាក្សាលើមេគុណ

- បើ $a = 0$ នោះ $bx + c = 0$

- បើ $a \neq 0$ នោះ $ax^2 + bx + c = 0$

+ សំដែង $\text{delta} = b * b - 4 * a * c$

+ ពិភាក្សាលើសញ្ញា delta

- បើ $\Delta < 0 \rightarrow$ No root
- បើ $\Delta = 0 \rightarrow x_1 = x_2 = -b/2a$
- បើ $\Delta > 0$ នោះ៖

$$x_1 = \frac{-b + \sqrt{\Delta}}{2a}$$

$$x_2 = \frac{-b - \sqrt{\Delta}}{2a}$$

•Data Structures

- + ការរៀបចំរចនាសម្ព័ន្ធ
- + ការប្រកាសអថេរ៖ a, b, c, x₁, x₂, delta
- + បញ្ចូលអថេរចាំបាច់របស់ចំណោទ a ; b ; c
- + បង្ហាញចេញលទ្ធផលរបស់ចំណោទ

1-4. ភាសាសម្រាប់ Algorithms

នៅពេលដោះស្រាយជាមួយម៉ាស៊ីន យើងចាំបាច់ត្រូវគិតដល់បញ្ហា Algorithms និង Data Structures ហើយនៅពេលសំដែងចេញទម្រង់របស់ Algorithms និង Data Structures យើងមិនត្រូវដាក់ចេញនូវភាសាសម្រាប់ Algorithms និង Data Structures របស់ចំណោទបញ្ហានោះទេ ប៉ុន្តែដើម្បីអោយមានលក្ខណៈងាយស្រួលក្នុងការស្វែងយល់ និងងាយសរសេរ យើងគួរប្រើសរសេរនូវភាសាសម្រាប់ណាមួយ ដែលមានលក្ខណៈជាមូលដ្ឋាន ហើយមានលទ្ធភាពអាចធ្វើការឆ្លុះបញ្ចាំងរាល់ Structure ដែលលើកឡើងដូចជា Pascal, c/c++, Java Programming...។

ដូចនេះ ក្នុងឯកសារនេះយើងប្រើសរសេរភាសា C សំរាប់សំដែងចេញនូវរាល់ Structure របស់ Algorithms និង Data Structures។ យើងមាន Structure ជាមូលដ្ឋានមួយចំនួនរបស់ភាសា C ដូចខាងក្រោម៖

(i)-ទម្រង់ទូទៅរបស់ភាសា C

```
Header file
Globle variable
Function
void main( )
{
    .....
    .....
}
```

(ii)-ការផ្ទេរតម្លៃ

$$V = E ;$$

ដែល (V = Variable, E = Expression)

-អថេរ E ជាអ្នកផ្តល់តម្លៃឲ្យអថេរ V។

-អថេរ E ជា Expression, Variable , Value។

ឧទាហរណ៍:

X = Y ; X = 20 ; X = Y + 10; Y = X;

(iii)-ប្រភេទទិន្នន័យ

ប្រភេទទិន្នន័យ គឺជាប្រភេទទិន្នន័យដែលប្រើសម្រាប់ប្រកាសប្រាប់ពី

លក្ខណៈរបស់វា។ ប្រភេទទិន្នន័យរួមមាន ២ ប្រភេទគឺ៖

(1) Simple data type

int, float , char, long, double

(2) Structure data type

Array , String, Pointer , Enum, Struct , union , and file.

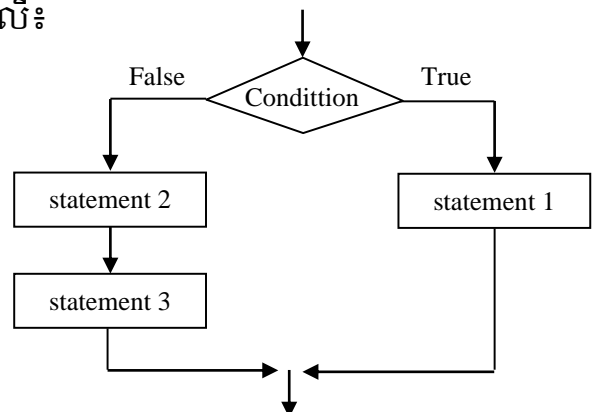
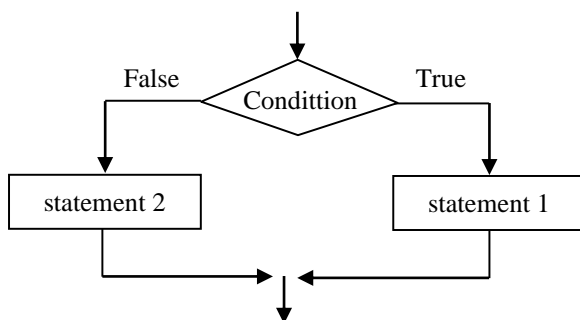
(iv) Statement

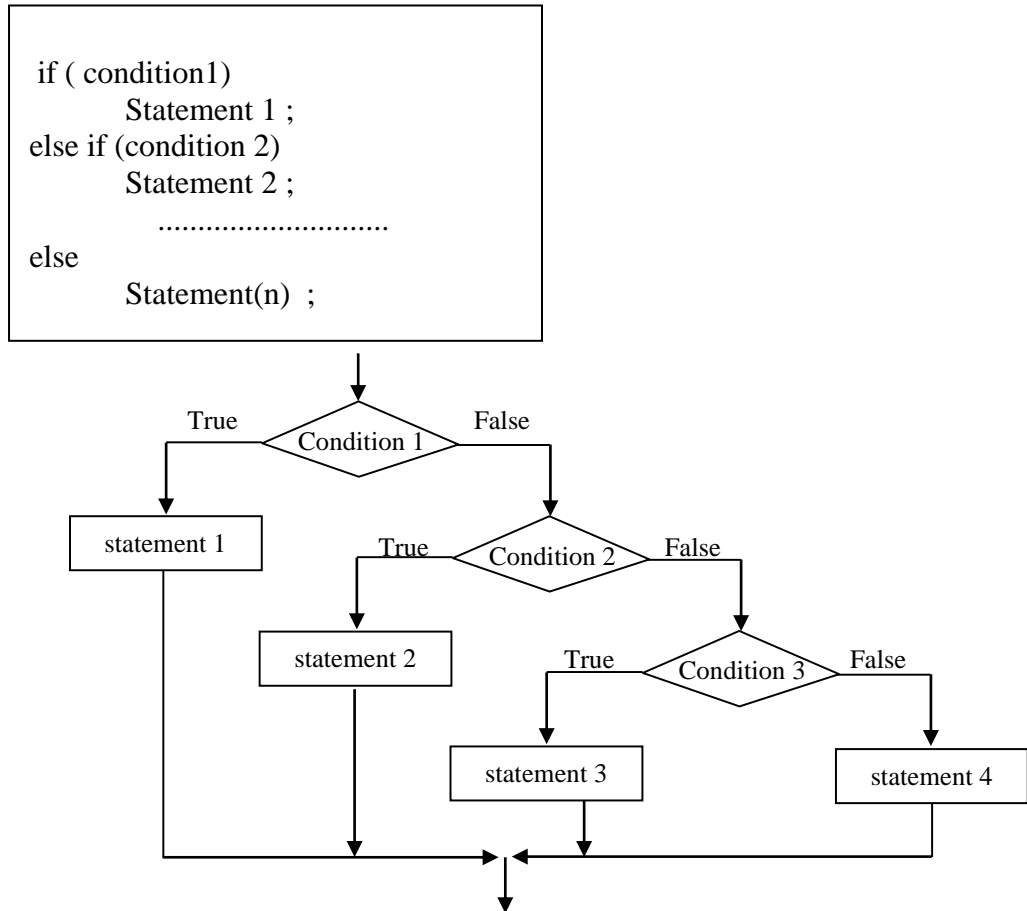
1- Control statement

```
- if ( condition)
    Statement1 ;
else
    statement 2 ; statement 3 ;
```

```
if (condition)
    statement1;
else
    statement2;
```

+Flowchart ដែលតាងឲ្យឃ្លាសរសេរកម្មវិធីខាងលើ៖





-ការប្រើ if នៅក្នុង if មួយទៀត(Nested if (...) statement)

```

if( condition )
    Statement 1;
else
    {
        if( condition )
            Statement 2;
        else
        {
            Statement 3;
            Statement 4;
            Statement 5;
        }
    }
    
```

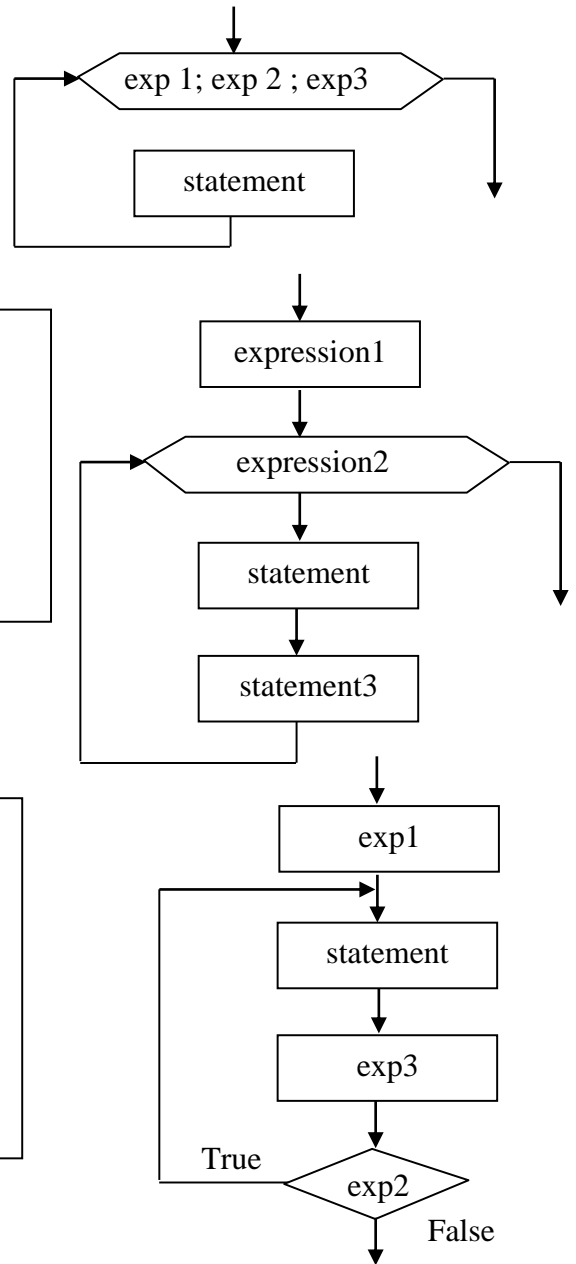
```

- switch(expression)
{
    case const1 : statement1 ; break ;
    case const2 : statement2 ; break;
    .....
    default : statement n ;
}
    
```

2- Loop statement

* For Loop

```
for(exp 1; exp 2 ; exp 3)
    statement ;
```



* While Loop:

```
expression1 ;
while(expression2)
{
    statement ;
    expression3;
}
```

* do / while loop :

```
exp1;
do
{
    statement ;
    exp3 ;
}while(exp2 ) ;
```

ឧទាហរណ៍១៖

កម្មវិធីខាងក្រោមបង្ហាញនូវការផ្ទៀងផ្ទាត់តួអក្សរតាមការជ្រើសរើសយក រួចហើយបង្ហាញសារដែលត្រូវគ្នា។

Program to illustrate the character checking, Accept choice and print appropriate message.

```
# include < stdio.h >
void main( )
{
    char ch;
    printf( "Enter your choice y/n" );
```

```
scanf( "%c", &ch );
if( ch == ' y ' || ch == ' Y ' )
    printf( "Your choice is yes \n" );
else
    printf( "Your choice is no \n" );
} /* main end */
```

ឧទាហរណ៍២៖

កម្មវិធីខាងក្រោមនេះត្រួតពិនិត្យចំនួនបឋម រឺមិនបឋម។

(A prime number is one which is divisible only by 1 or itself)

```
# include < stdio.h >
void main( )
{
    /* កម្មវិធីដើម្បីត្រួតពិនិត្យចំនួនបឋម រឺមិនបឋម */
    int num, i ;
    printf( "Enter a number : " );
    scanf( "%d" , &num );
    i = 2 ;
    while ( i <= num - 1 )
    {
        if( num % i == 0 )
        {
            printf( "Number is not a prime" );
            break;
        }
    }
    if( i == num )
        printf( "Prime number" , num );
} /* main end */
```

ឧទាហរណ៍៣៖

កម្មវិធីខាងក្រោមនេះគណនាផលបូកផលដកនិងផលគុណដោយប្រើ Switch

Statement ។

```
# include < stdio.h >
void main( )
{
    /* Demonstration of switch statement */ (បង្ហាញអំពី switch Statement)
    int a, b, sum, diff, pro, ch ;
    float rat ;
    printf( " Enter value of a and b : \n" );
    scanf( "%d %d" , &a, &b );
    printf( "\n M E N U" );
```



```
printf( "\ n 1 for Sum" );
printf( "\ n 2 for Product" );
printf( "\ n 3 for Ratio" );
printf( "\ n 4 for Difference" );
printf( "\ n Enter your choice 1 / 2 / 3 / 4 : " );
scanf( "%d", &ch );
case 1 :
    sum = a + b ;
    printf( "Sum of a & b = %d ", sum );
    break;
case 2:
    pro = a * b ;
    printf( "Product of a & b = %d " , pro );
    break;
case 3:
    rat = a / b ;
    printf( " Ratio of a & b = %f " , rat );
    break;
case 4:
    diff = a - b ;
    printf( "Diff of a & b = %d " , diff );
    break;
}
```

(v)-អនុគមន៍

អនុគមន៍មានទម្រង់ដូចខាងក្រោម៖

```
Return type Function Name(paramete list....)
{
    Local variable
    statement ;
    .....
}
```

ដែល

* **Return type** រួមមាន ២ប្រភេទគឺ void និង data type។

* **Function Name** គឺជាឈ្មោះរបស់អនុគមន៍។

* **Paramete List** រួមមាន ២ប្រភេទគឺ៖

-**Value** ប៉ារ៉ាម៉ែត្រ គឺជាអថេរដែលមិនប្រែប្រួលតម្លៃមុនពេលហៅអនុគមន៍ និងក្រោយពេលហៅអនុគមន៍។

-**Reference** ប៉ារ៉ាម៉ែត្រ គឺជាអថេរដែលប្រែប្រួលតម្លៃមុនពេលហៅ និងក្រោយពេលហៅអនុគមន៍មកប្រើ។

ឧទាហរណ៍១៖

អនុគមន៍ប្តូរតម្លៃនៃ ២ចំនួនគត់ (ដោយប្រើលក្ខណៈ Value ប៉ារ៉ាម៉ែត្រ និង Reference ប៉ារ៉ាម៉ែត្រ)។

* Value ប៉ារ៉ាម៉ែត្រ

ឧទាហរណ៍

```
void change(int a , int b)
{
    int temp = a ; a = b ; b = temp;
    printf("in value a = %d , b = %d", a , b );
}
```

* Reference ប៉ារ៉ាម៉ែត្រ

ឧទាហរណ៍

```
void change(int * a , int * b)
{
    int temp = * a; *a = *b ; *b = temp ;
    printf("\n value a =%d ; b = %d ", * a , *b) ;
}
```

ឧទាហរណ៍

```
void change (int & a , int & b)
{
    int temp = a ; a = b ; b = temp ;
    printf( "\n value a = %d , b = %d" , a , b) ;
}
```

ឧទាហរណ៍២៖

-Value ប៉ារ៉ាម៉ែត្រ

```
void change(int x , int y) ;    /* x and y ជា Value paramete */
```

ឧទាហរណ៍៣៖

-Reference ប៉ារ៉ាម៉ែត្រ

```
void change(int * x, int * y) ; /* x និង y ជា Return paramete */
```

ឧទាហរណ៍៤៖

```
void change(int & x , int & y) ; /* x និង y គឺជា Reference paramete */
```

**** របៀបសរសេរអនុគមន៍**

យើងមានរបៀបសរសេរអនុគមន៍ ៤បែបដែលត្រូវបានលើកឡើងគឺ៖

១-អនុគមន៍មាន data type និងមានប៉ារ៉ាម៉ែត្រ

២- អនុគមន៍គ្មាន data type និងគ្មានប៉ារ៉ាម៉ែត្រ

៣-អនុគមន៍គ្មាន data type និងមានប៉ារ៉ាម៉ែត្រ

៤-អនុគមន៍គ្មាន data type និងគ្មានប៉ារ៉ាម៉ែត្រ

ឧទាហរណ៍១៖

ចូរសរសេរអនុគមន៍ដើម្បីគណនាផលបូកនៃ ២ ចំនួនគត់ a និង b ។

១-អនុគមន៍មាន data type និងមានប៉ារ៉ាម៉ែត្រ

```
int sum(int a, int b)
{
    int s = a + b ;
    return(s) ; }
```

២- អនុគមន៍មាន data type និងគ្មានប៉ារ៉ាម៉ែត្រ

```
int sum( )
{
    int a, b ;
    printf("input a , b : ");
    scanf( " %d %d ", & a, & b) ;
    int s = a + b;
    return(s) ;
}
```

៣-អនុគមន៍គ្មាន data type និងមានប៉ារ៉ាម៉ែត្រ

```
void sum(int a , int b )
{
    int s = a + b ;
    printf("\n sum = % d ", s) ;
}
```

៤-អនុគមន៍គ្មាន data type និងគ្មានប៉ារ៉ាម៉ែត្រ

```
void sum( )
{
    int a , b ;
    printf("input a , b: ") ;
    scanf("%d %d ", & a, & b);
    int s = a + b ;
    printf( "\n sum = % d ", s) ;
}
```

ឧទាហរណ៍២៖

ចូរសរសេរអនុគមន៍ដើម្បីរកតម្លៃធំបំផុតនៃ ២ ចំនួនគត់ a និង b ។

១-អនុគមន៍មាន data type និងមានប៉ារ៉ាម៉ែត្រ

```
int max(int a, int b)
{
    int Max ;
    if( a > b )
        Max = a;
    else
        Max = b;
    return(Max);
}
```

២- អនុគមន៍មាន data type និងគ្មានប៉ារ៉ាម៉ែត្រ

```
int max( )
{
    int a, b ;
    printf("input a , b : ");
    scanf( " %d %d ", & a, & b );
    int Max;
    if( a > b )
        Max = a;
    else
        Max = b;
    return(Max);
}
```

៣-អនុគមន៍គ្មាន data type និងមានប៉ារ៉ាម៉ែត្រ

```
void max(int a , int b )
{
    int Max;
    if( a > b )
        Max = a;
    else
        Max = b;
    printf("\n Maximum = %d ", Max) ;
}
```

៤-អនុគមន៍គ្មាន data type និងគ្មានប៉ារ៉ាម៉ែត្រ

```
void max( )
{
    int a , b ;
    printf("input a , b: ") ;
    scanf("%d %d ", &a , &b);
    int Max;
    if( a > b )
        Max = a;
```

```

else
    Max = b;
printf("\n Maximum = %d ", Max) ;
}

```

ឧទាហរណ៍៣៖

Illustrate a called by value parameter passing method as technique. To accept any two number and find the sum.

```

#include < stdio.h >
int sumx( int x, int y )
{
    return ( x + y );
}
void main( )
{
    int a, b, sum ;
    printf( "Enter any two number : " );
    scanf( "%d %d" ,&a, &b );
    sum = sumx( a, b );
    printf( "Sum of a and b number is %d " , sum );
}

```

ឧទាហរណ៍៤៖

សរសេរកម្មវិធីដោយប្រើអនុគមន៍ call by reference mechanism ។

```

#include < stdio.h >
void function( int * , int * )
void main( )
{
    int a, b ;
    a = 30 ;
    b = 50 ;
    printf( "a = %d      b = %d before function call \n " , a, b );
    function( &a , &b ) ;
    printf( "a = %d      b = %d after function call \n " , a, b );
    /* Call by reference function */
}

void function( int * x, int * y )
{
    *x = *x + *x ;
    *y = *y + *y ;
}

```

****Automatic Storage Class**

```
main( )
{
    Auto int a, b, c ;
    .....
    .....
}
```

ឧទាហរណ៍១៖

A program to illustrate how automatic variable used in program.

```
# include < stdio.h>
void main( )
{
    auto int num = 200 ;
    functionx( ) ;
    printf( "%d \ n", num ) ;
}
void functionx( )
{
    auto int num = 20 ;
    functiony( ) ;
    printf( "%d \ n", num ) ;
}functiony( )
{
    auto int num = 10 ;
    printf( "%d \ n", num ) ;
}
```

ឧទាហរណ៍២៖

សរសេរកម្មវិធីដើម្បីបំប្លែងពីប្រព័ន្ធគោលដប់ទៅប្រព័ន្ធគោលពីរដោយប្រើអនុគមន៍។

```
# include< stdio.h >
void main( )
{
    auto int dec, bin ;
    int dec_to_bin ( int ) ;
    printf( "Enter a decimal number \ n" ) ;
    scanf( "%d", &dec ) ;
    bin = dec_to_bin ( dec );
    printf( "The decimal number is = %d \ n", dec ) ;
    printf( "The binary number is = %d \ n", bin ) ;
}
/* អនុគមន៍ដើម្បីបំប្លែងពីប្រព័ន្ធគោលដប់ទៅប្រព័ន្ធគោលពីរ */
int dec_to_bin( int d )
{
    auto int b, r, y ;
```

```

b = 0 ;
y = 1 ;
while ( d > 0 )
{
    r = d % 2 ;
    d = d / 2 ;
    b = b + r * y ;
    y = y * 10 ;
}
return( b ) ;
}

```

****External Storage Class**

```

External int  a ;
External float p ;

```

ឧទាហរណ៍១៖

A program to illustrate the properties of external variables.

```

# include < stdio.h >
void main( )
{
    extern int  num = 500 ;
    output( ) ;
}
output( )
{
    printf( "%d", num ) ; / * num declared as external in the main program * /
}

```

ឧទាហរណ៍២៖

ចូរសរសេរកម្មវិធីដើម្បីត្រួតពិនិត្យម៉ាទ្រីសដោយប្រើ External variables។

```

# include< stdio.h >
extern int  a[10][10], b[10][10] ;
extern int  i, j, n ;
void main( )
{
    printf( "Enter order of the matrix \ n" ) ;
    scanf( "%d", &n ) ;
    printf( "\ n Enter matrix elements \ n" ) ;
    for( i = 0 ; i < n ; i ++ )
        for( j = 0 ; j < n ; j ++ )
            scanf( "%d", &a[i][j] ) ;
    printf( "The original matrix is \ n" ) ;
    matprint( a ) ;
    transpose( ) ;
    printf( "The transposed matrix is \ n" ) ;
}

```

```

        matprint( b ) ;
    }
    /* Function to transpose a matrix */ (អនុគមន៍ដើម្បីត្រលប់ម៉ាទ្រីស)
    void transpose( ) ;
    {
        for( i = 0 ; i < n ; i ++ )
            for( j = 0 ; j < n ; j ++ )
                b[i][j] = a[j][i] ;
    }
    /* Function to print a matrix */ (អនុគមន៍ដើម្បីបង្ហាញធាតុរបស់ម៉ាទ្រីស)
    void matprint( int x[ ][ ] )
    {
        for( i = 0 ; i < n ; i ++ )
        {
            for( j = 0 ; j < n ; j ++ )
                printf( "\t %d", x[i][j] ) ;
            printf( "\n" ) ;
        }
    }

```

****Static Storage Class**

<pre> Static int m ; Static float n ; </pre>
--

ឧទាហរណ៍១៖

A program to illustrate the properties of a static variable.

```

#include <stdio.h>
void main( )
{
    int n ;
    for( n = 1 ; n <= 3 ; n ++ )
        staticfun( ) ;
}
void staticfun( )
{
    Static int s = 0 ;    /* s value assigned to 0 at the time of compilation */
    s = s + 10 ;
    printf( "s = %d \n", s ) ;
}

```


ឧទាហរណ៍២៖

ចូរសរសេរកម្មវិធីដើម្បីបង្ហាញស៊េរី Fibonacci ដោយប្រើ static variables។

```
#include <stdio.h>
void main( )
{
    int i, n ;
    int fibo( int ) ;
    printf( "Enter number of elements in the series \ n" ) ;
    scanf( "%d", &n ) ;
    printf( "\ n Fibonacci numbers \ n \ n" ) ;
    for( i = 0 ; i < n ; i ++ )
        printf( "%d \ t", fibo( i ) ) ;
}
/* Function to find Fibonacci numbers */ (អនុគមន៍ដើម្បីរកចំនួន Fibonacci)
int fibo( int k )
{
    static int n1 = 1 ;
    static int n2 = 1 ;
    int n3 ;
    if ( k == 1 )
        n3 = 0 ;
    else if ( k == 2 )
        n3 = 1 ;
    else
        n3 = n1 + n2 ;
    n1 = n2 ;
    n2 = n3 ;
    return ( n3 ) ;
}
```

ឧទាហរណ៍៣៖

Write a program to find the sum of 5 elements static in nature using pointers with function.

```
#include<stdio.h>
void main( )
{
    static int array[5] = { 200, 400, 600, 800, 1000};
    int addnum( int *ptr ); /* function prototype */
    int sum;
    sum = addnum(array);
    printf(“ Sum of all array elements = %d\n”, sum);
}
int addnum(int *ptr)
```

```
{
    int total = 0, index;
    for( index = 0; index < 5; index++)
        total += *(ptr + index);
    return(total);
}
```

****Register Variables**

ឧទាហរណ៍១៖

ចូរសរសេរកម្មវិធីដើម្បីបង្ហាញការនៃចំនួនពី 1 ដល់ 10 ដោយប្រើ register variable ។

```
# include < stdio.h >

void main( )
{
    register int count, sqr ;
    for( count = 1 ; count <= 10 ; count + + )
    {
        sqr = count * count ;
        printf( "\n %d %d", count, sqr ) ;
    }
}
```

ឧទាហរណ៍២៖

ចូរសរសេរកម្មវិធីដើម្បី Register Variable ដែលបានបង្កើតនៅក្នុង CPU ។

```
# include < stdio.h >

void main( )
{
    register int y ;
    printf( "The address of y = %u \n", &y ) ;
}
```

លំហាត់អនុវត្ត

(1)-ចូរសរសេរកម្មវិធី សំរាប់រកតំលៃធំបំផុតនៃបីចំនួនដោយប្រើរបៀបសរសេរអនុគមន៍ អោយបាន ៤របៀប។

(2)-ចូរសរសេរកម្មវិធី សំរាប់ផ្លាស់ប្តូរតំលៃ ២ ចំនួនគត់ដោយប្រើ Value ប៉ារ៉ាម៉ែត្រ និង Reference ប៉ារ៉ាម៉ែត្រព្រមទាំងបង្ហាញនៅលើអេក្រង់ (screen) តាមលំនាំដូចខាងក្រោម៖

1- Value ប៉ារ៉ាម៉ែត្រ

- មុនពេលហៅអនុគមន៍ A = 40 និង B = 50
- ពេលហៅអនុគមន៍ A = 50 និង B = 40
- ក្រោយពេលហៅអនុគមន៍ A = 40 និង B = 50

2- Reference ប៉ារ៉ាម៉ែត្រ

- មុនពេលហៅអនុគមន៍ A = 40 និង B = 50
- ពេលហៅអនុគមន៍ A = 50 និង B = 40
- ក្រោយពេលហៅអនុគមន៍ A = 50 និង B = 40

(3)-Define Function? And discuss the category of Functions.

(4)-Write notes on :

- a)-argument and parameters
- b)-Local variable
- c)-Global variable
- d)-Function prototypes

(5)-Write the Syntax with example of each of the following looping statement in C

- a)-while()
- b)-do while
- c)-for()

(6)-Explain the Case Control Structure used in C with Syntax and example.



មេរៀនទី២

ការធ្វើគម្រោង និងវិភាគ Algorithms

2-1. ការបំប្លែងពីចំណោទបញ្ហាឲ្យទៅជា Module និងដំណោះស្រាយ

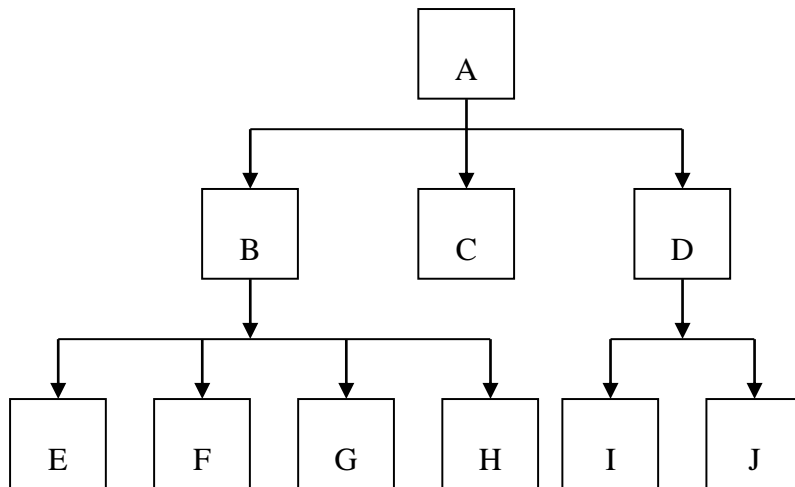
ចំពោះចំណោទបញ្ហាដែលមានទ្រង់ទ្រាយធំ និងមានលក្ខណៈស្មុគស្មាញនោះ Algorithms សម្រាប់ដោះស្រាយក៏មានលក្ខណៈស្មុគស្មាញដែរ ប៉ុន្តែប្រសិនបើចែកបែងចែកវាឲ្យទៅជាបញ្ហាតូចៗដោយប្រើគោលការណ៍ច្បាស់លាស់ នោះបញ្ហាស្មុគស្មាញនឹងមានលក្ខណៈងាយស្រួល។

យើងមានគោលការណ៍ ២យ៉ាងដែលត្រូវបានលើកយកមកប្រើប្រាស់ក្នុងការធ្វើគម្រោងសម្រាប់តាងឲ្យចំណោទបញ្ហាដូចខាងក្រោម៖

(a)-ការរៀបពីលើចុះក្រោម(Top Down Design)

Top Down Design គឺជាវិធានធ្វើគម្រោងតាងភ្ជាប់ជាមួយកម្មវិធី។ គោលការណ៍ Top Down Design ត្រូវបានអនុវត្តតាមលំនាំបែងចែកចំណោទបញ្ហា ដែលមានទ្រង់ទ្រាយធំ និងមានលក្ខណៈស្មុគស្មាញឲ្យទៅជាបញ្ហាតូចៗជាបន្តបន្ទាប់ ដែលតាងដោយ Module ហើយការបែងចែក Module ត្រូវបានអនុវត្តជាបន្តបន្ទាប់រហូតដល់បញ្ហាជាមូលដ្ឋានងាយស្រួលក្នុងការដោះស្រាយ។

យើងមាន Structure ទូទៅរបស់ Top Down Design ដែលតាងដោយ Module ដូចខាងក្រោម៖



ការបែងចែក Module ត្រូវបានកំណត់ដោយ៖

- Module ដំបូងស្ថិតនៅ Level[0] ។
- រាល់ Module ដែលស្ថិតនៅ Level[i + 1] ត្រូវបានបែងចែកពី Module ស្ថិតនៅ Level[i] ជាមួយ $i = 1, 2, 3, \dots$ ។

យើងមានគោលការណ៍អនុវត្តរបស់ចំណោទបញ្ជាត្រូវការធ្វើគម្រោង ត្រូវបានកំណត់ចេញតាមតារាងដូចខាងក្រោម៖

Idea	Top down	P ogram
យុទ្ធសាស្ត្រទូទៅសំរាប់ ដោះស្រាយបញ្ហា ↓ ការបង្កើតផ្នែកជាចំណាត់ថ្នាក់របស់បញ្ហា ↓ ផ្នែកនីមួយៗ ជាពិស្តារ	Module ដំបូង (Level 0) ↓ Sub Module ↓ Module ស្ថិតនៅ Level ចុងក្រោយ	កំណត់សរសេរកម្មវិធី ↓ Sub program ↓ source code

ឧទាហរណ៍៖

ចូរធ្វើគម្រោង Top Down Design របស់ចំណោទបញ្ជាគ្រប់គ្រងលើប្រាក់ឧបត្ថម្ភដល់និស្សិតដែលរៀនពូកែដើម្បីរាយការណ៍ជូនក្រសួង ។

ដើម្បីដោះស្រាយចំណោទបញ្ជានេះជាដំបូងយើងត្រូវគិតដល់បញ្ហាបញ្ចូល និងបញ្ចេញ និងរាល់សំណូមពរ របស់ចំណោទបញ្ជាដែលរួមមាន៖

១-រាល់ព័ត៌មានរបស់និស្សិតត្រូវបានទទួលប្រាក់ឧបត្ថម្ភត្រូវបានរក្សាទុកក្នុង file ណាមួយ។

២-ស្វែងរកបាននូវព័ត៌មានរបស់និស្សិតណាម្នាក់ដែលយើងចង់រកនៅក្នុង List ។

៣-ផ្លាស់ប្តូរនូវរាល់ព័ត៌មានរបស់និស្សិតដូចជា៖ field name , field score , ... ។

៤-បោះពុម្ពនូវរាល់តារាងបញ្ជីឈ្មោះរបស់និស្សិត ដែលត្រូវទទួលបានប្រាក់ឧបត្ថម្ភ។

+ចំពោះចំណោទនេះយើងធ្វើការបែងចែកជាបីភារកិច្ចដូចខាងក្រោម៖

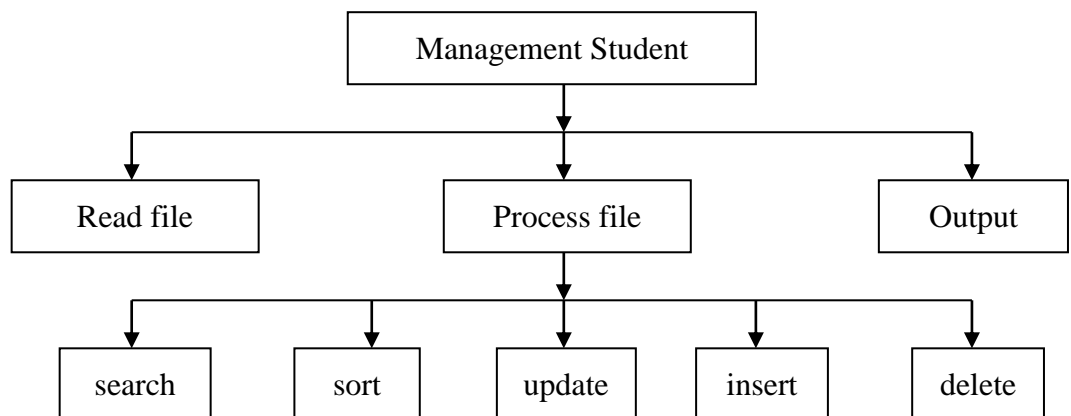
- បញ្ចូលរាល់ព័ត៌មានរបស់និស្សិត និងរក្សាទុកនៅក្នុង file ណាមួយដែលឲ្យឈ្មោះថា ភារកិច្ចអាន file(Read file)។

(ii)-អនុវត្តដំណោះស្រាយនូវរាល់សំណួររបស់ចំណោទបញ្ហា ដែលឲ្យឈ្មោះថា ភារកិច្ចដំណើរការ file(Process file)។

(iii)-បង្ហាញចេញនូវរាល់ព័ត៌មានរបស់និស្សិត ដែលត្រូវទទួលបានប្រាក់ឧបត្ថម្ភ ដែលឲ្យឈ្មោះថាភារកិច្ច បញ្ចេញទិន្នន័យ។ យើងឃើញថាចំពោះភារកិច្ចដំណើរការ file នៅមានលក្ខណៈស្មុគស្មាញដែលទាមទារឲ្យយើងបែងចែកជាភារកិច្ចតូចៗបន្តទៀតគឺ៖

- (a) Search គឺសម្រាប់ស្វែងរកនូវព័ត៌មានរបស់និស្សិតនៅក្នុង List។
- (b) Sort គឺសម្រាប់រៀបចំសេរីទិន្នន័យនៅក្នុង List។
- (c) Update គឺសម្រាប់ផ្លាស់ប្តូររាល់ព័ត៌មានរបស់និស្សិតដូចជាfieldName,fieldscore,..។
- (d) Insert គឺសម្រាប់បន្ថែមធាតុចូលទៅក្នុង List។
- (e) Delete គឺសម្រាប់លុបនូវរាល់ព័ត៌មានរបស់និស្សិតណាម្នាក់នៅក្នុង List។

ដូចនេះយើងអាចគូសគំនូសតាង Top Down Design របស់ចំណោទបញ្ហានេះដោយប្រើ Module ដូចខាងក្រោម៖



(b)- Stepwise Refinement

Stepwise Refinement គឺជាវិធានសម្រាប់ធ្វើគម្រោងតាងភ្ជាប់ជាមួយ

Algorithms ហើយវាមានឥទ្ធិពលលើ Top Down Design។ នៅពេលអនុវត្ត Algorithms របស់ Stepwise Refinement រួមមានភាសាពីរដែលបានបង្ហាញចេញគឺភាសាធម្មជាតិ និងភាសាកម្មវិធីឲ្យឈ្មោះថា Pseudo Code រឺ Pseudo Code Language ដោយអនុវត្តជាបណ្តើរៗពី ភាសាធម្មជាតិឆ្លងតាម Pseudo Code និងឈានទៅដល់ភាសាកម្មវិធីដែលយើងបានជ្រើសរើស និងដែលចាប់ផ្តើមពីចំណុចធ្វើអ្វី (what?) ដល់ចំណុចធ្វើដូចម្តេច(How?)។

ឧទាហរណ៍៖

ចូរធ្វើគម្រោង Stepwise Refinement របស់ចំណោទបញ្ហាបង្កើតម៉ាទ្រីស ($a[i][j]$) ដែលមាន n ជួរដេក និង n ជួរឈរដោយបង្ហាញចេញតម្លៃធាតុនីមួយៗរបស់វានៅលើបន្ទាត់ស្របជាមួយអង្កត់ទ្រូងទី១ និងរាល់ធាតុដែលស្ថិតនៅលើអង្កត់ទ្រូងនោះ។

ឧទាហរណ៍ $n = 3 \rightarrow (a[i][j])_{3 \times 3}$

$$\Rightarrow a[i][j] = \begin{pmatrix} 20 & 9 & 40 \\ 7 & 6 & 30 \\ 80 & 4 & 50 \end{pmatrix} \Rightarrow \text{Display}$$

40		
9	30	
20	6	50
7	4	
80		

ចំណោទបញ្ហានេះ Algorithms សម្រាប់ដោះស្រាយរួមមានបីភារកិច្ចគឺ៖

១-កំណត់តម្លៃជួរដេក និងតម្លៃជួរឈរ របស់ម៉ាទ្រីស។

២-កំណត់តម្លៃធាតុនីមួយៗរបស់ម៉ាទ្រីស $a[i][j]$ ។

៣-បង្ហាញចេញនូវធាតុនីមួយៗរបស់ម៉ាទ្រីសនៅលើបន្ទាត់ស្របជាមួយអង្កត់ទ្រូង និងរាល់ធាតុស្ថិតនៅលើអង្កត់ទ្រូងនោះ។ យើងមាន Algorithms របស់ Stepwise Refinement ដើម្បីដោះស្រាយចំណោទបញ្ហានេះត្រូវបានអនុវត្តជាបន្តបន្ទាប់ដូចខាងក្រោម៖

-ចំពោះភារកិច្ចទី១ និងទី២ យើងអាចសរសេរជាមួយភាសាកម្មវិធី (C Language) គឺ

```
scanf(" %d ", &n);
for(i = 0 ; i < n ; i++)
    for(j = 0 ; j < n ; j++)
        scanf(" %d ", &a[i][j]);
```

-ចំពោះភារកិច្ចទី៣ នៅមានលក្ខណៈស្មុគស្មាញទាមទារឲ្យយើងបែងចែកជាពីរភារកិច្ចបន្តទៀតគឺ

៣-១.បង្ហាញចេញនូវធាតុនីមួយៗរបស់វាស្ថិតនៅលើបន្ទាត់ស្របជាមួយអង្កត់ទ្រូងទី១ អាស្រ័យដោយជួរឈរ (column j) ពីទីតាំង $(n - 1)$ ដល់ទីតាំង 0 ។

៣-២.បង្ហាញចេញនូវធាតុនីមួយៗរបស់វាស្ថិតនៅលើបន្ទាត់ស្របអង្កត់ទ្រូងទី១ អាស្រ័យដោយជួរដេក (row i) ពីទីតាំង 0 ដល់ទីតាំង $(n - 1)$ ។

-ចំពោះភារកិច្ច ៣-១ និង ៣-២ យើងអាចសរសេរ:

៣-១. for (j = n - 1 ; j >= 0 ; j --) : ភាសាកម្មវិធី

បង្ហាញចេញនូវធាតុនីមួយៗរបស់វ៉ៃដែលស្ថិតនៅលើបន្ទាត់
ស្របជាមួយអង្កត់ទ្រូងទី ១ អាស្រ័យដោយជួរឈរ j ។

} Pseudo Code
} ភាសាធម្មជាតិ

៣-២. for (i = 1 ; i < n ; i ++)

អាស្រ័យដោយជួរដេក i យើងធ្វើការត្រួតពិនិត្យមើលជាមួយការងារអនុវត្តរបស់ ៣-១ ដូចខាងក្រោម:

-ជាមួយជួរឈរ j = n - 1 នោះបង្ហាញចេញមួយធាតុ ($a[0][2] = 40$) គឺធាតុដែលមាន row = 0 និង column = 2 ។

-ជាមួយជួរឈរ j = n - 2 នោះបង្ហាញចេញ ២ ធាតុ ($a[0][1] = 9$; $a[1][2] = 30$) គឺធាតុមាន row = 0, column = j និង row = 1 , column = j + 1 ។

-ជាមួយជួរឈរ j = n - 3 នោះបង្ហាញចេញ ៣ធាតុ ($a[0][0] = 20$, $a[1][1] = 6$, $a[2][2] = 50$) គឺធាតុដែលមាន row = 0, column = j, row = 1, column = j + 1 និង row = 2, column = j + 1

.....

ដូចនេះយើងឃើញថា លំនាំអនុវត្តន៍ ៣-១ គឺចំនួនធាតុដែលបានបង្ហាញចេញនៅលើបន្ទាត់ស្របនីមួយៗរបស់វ៉ៃស្មើនឹង $n - j$ និងធាតុដែលបានបង្ហាញចេញ $a[i][j + i]$ ជាមួយ $0 \leq i < n - j$ ។

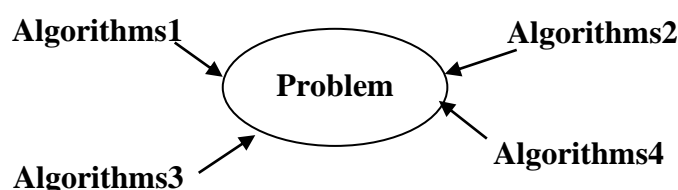
ដូចនេះភារកិច្ច ៣-១ អាចសរសេរតាមភាសា C ដូចខាងក្រោម:

```
for(j = n-1 ; j >= 0 ; j -- )
{
    for( i = 0 ; i < n - j ; i ++ )
        printf("%4d ", a [i] [ j + i] );
    printf(" \n ");
}
```

ដូចគ្នាផងដែរចំពោះភារកិច្ច ៣-២ យើងអាចសរសេរតាមភាសា C ដូចខាងក្រោម:

```
for ( i = 1; i < n ; i ++ )
{
    for(j = 0; j < n-i ; j ++ )
        printf(" % 5d" , a [ i + j ] [j]);
    printf(" \n " );
}
```


2-2: ការវិភាគ Algorithms(Algorithms Analysis)



នៅក្នុងចំណោមបញ្ហាមួយតែងតែមានមធ្យោបាយសម្រាប់ដោះស្រាយច្រើន មានន័យថាយើងមិនត្រូវកំណត់មធ្យោបាយតែមួយសម្រាប់ធ្វើការដោះស្រាយបញ្ហានោះទេ។

តើយើងត្រូវធ្វើយ៉ាងណា ដើម្បីឲ្យជ្រើសរើសបាននូវ Algorithms មួយដែលមានលក្ខណៈល្អ រឺអន់ថយ? យើងមានកត្តាជាច្រើនសម្រាប់ធ្វើការវាយតម្លៃលើ Algorithms ដើម្បីកំណត់ចេញនូវលក្ខណៈល្អ រឺអន់ថយរបស់វា ប៉ុន្តែនៅក្នុងឯកសារនេះយើងជ្រើសរើសនូវមធ្យោបាយមួយសម្រាប់ធ្វើការវាយតម្លៃលើ Algorithms តាមរយៈពេលវេលាអនុវត្តរឺកម្រិតស្មុគស្មាញដោយប្រើគោលការណ៍កំណត់ចេញនូវតម្លៃប្រហាក់ប្រហែល ដែលតាងដោយនិមិត្តសញ្ញា $O()$ (Big.oh)។ និមិត្តសញ្ញា $O()$ ត្រូវបានតាងដើម្បីធ្វើការវាស់ពេលវេលាអនុវត្តន៍របស់ Algorithms តាមរយៈអនុគមន៍មួយចំនួន ដែលអាស្រ័យដោយទំហំទិន្នន័យកំណត់ដោយអថេរ n ហើយអនុគមន៍ តែងត្រូវបានគេប្រើប្រាស់រួមមាន៖

$$\log_2(n), n, n\log_2(n), n^2, n^3, 2^n$$

យើងមានតារាងតម្លៃត្រូវគ្នារបស់អនុគមន៍ទាំងនេះដូចខាងក្រោម៖

$\log_2(n)$	n	$n\log_2(n)$	n^2	n^3	2^n
0	1	0	1	1	2
1	2	2	4	8	4
2	4	8	16	64	16
3	8	24	64	512	256
4	16	64	256	4096	65536
5	32	160	1024	32768	429496

ឧបមាថា c ជាចំនួនថេរ, n ជាទំហំទិន្នន័យហើយ $f(n)$ ជាអនុគមន៍នៃពេលវេលាអនុវត្តរបស់ Algorithms ។

យើងមានទម្រង់ $c.f(n) +$ អង្គមានកំរិតយឺតជាង

ដូចនេះគេបានពេលវេលាអនុគមន៍ ឬ កម្រិតស្មុគស្មាញរបស់វាគឺ

$$O(f(n))$$

ឧទាហរណ៍១៖

ឧបមាថា យើងមានអនុគមន៍ដូចខាងក្រោម៖

$$T(n) = \begin{cases} 1 & \text{if } n = 1 \\ 2T(n/2) + 4 & \text{if } n > 1 \end{cases}$$

ដូចនេះគេបានពេលវេលាអនុវត្ត ឬកម្រិតស្មុគស្មាញរបស់វាគឺ $O(n^3)$

ឧទាហរណ៍២៖

ចូរកំណត់ $O()$ របស់អនុគមន៍មួយចំនួនខាងក្រោម៖

- (i)- $5n^3+4n^2+4$
- (ii)- $30n+96$
- (iii)- $6n^3+8n+9$
- (iv)- 2^n+4n+7
- (v)- $4n^2+2n+8$

យើងបាន៖

- (i)- $O(n^3)$ (ii)- $O(n)$
- (iii)- $O(n^3)$ (iv)- $O(2^n)$ (v)- $O(n^2)$

យើងឃើញថាពេលវេលាអនុវត្ត រឺកម្រិតស្មុគស្មាញរបស់ Algorithms ទី២ គឺ $O(n)$ មានលក្ខណៈប្រសើរជាងគេ រីឯពេលវេលាអនុវត្ត ឬកម្រិតស្មុគស្មាញរបស់ Algorithms ទី៤គឺ $O(2^n)$ មានលក្ខណៈអន់ថយជាងគេ។

ឧទាហរណ៍៣៖

ដើម្បីគណនាផលបូក $\text{sum} = 1! + 2! + 3! + \dots + n!$ គេមាន Algorithms ពីរក្នុងការដោះស្រាយ។ តើ Algorithms ណាមួយដែលមានលក្ខណៈប្រសើរជាងគេ?

+ Algorithms ទីមួយ

```
long sum (int n)
{
    int i , j ;
    long s , p; s = 0 ;
    for (i = 1; i <= n ; i ++ )
    {
        p = 1;
        for (j = 1; j <= i ; j ++ )
            p = p * j ;
        s = s + p ;
    }
    return (s);
}
```

ដូចនេះ Statement ដែលអនុវត្តច្រើនជាងគេគឺ៖

```
for (i = 1 ; i <= n ; i ++ )
    for (j = 1 ; j < i ; j ++ )
        p = p * j ;
```

- បើ $i = 1$ statement អនុវត្ត: 1
- បើ $i = 2$ statement អនុវត្ត: 2
- បើ $i = n$ statement អនុវត្ត: n



ដូចនេះកម្រិតស្មុគស្មាញរបស់ Algorithms គឺ $O(n^2)$

+ Algorithms ទីពីរ

```
log sum (int n)
{
    int i ; long s = 0 , p = 1 ;
    for (i = 1 ; i <= n ; i ++ )
    {
        p = p * i ;
        s = s + p ;
    }
    return (s) ;
}
```

ដូចនេះ Statement ដែលអនុវត្តច្រើនជាងគេគឺ៖

```
for (i = 1 ; i <= n ; i ++ )
    p = p * i
```

- ពេលវេលាអនុវត្តនៃ $T(n) = n + 1$

ដូចនេះកម្រិតស្មុគស្មាញនៃ Algorithms គឺ $T(n) = O(n)$

យើងសង្កេតឃើញថា Algorithms ទី២ មានលក្ខណៈប្រសើរជាង Algorithms ទី១ ។

ឧទាហរណ៍៤៖

ឧបមាថាយើងមាន Algorithms របស់ចំណោទបញ្ហា

$$s(n) = 1 + 2 + 3 + \dots + n$$

ត្រូវបានសរសេរដូចខាងក្រោម៖

```
1- sum = 0;
2- for (i = 1 ; i <= n ; i ++ )
3-     sum = sum + i;
4- printf( "sum = %d ", sum ) ;
```

(ក)-ចូរកំណត់អនុគមន៍នៃវេលាអនុវត្ត $T(n)$ របស់ Algorithms ។

(ខ)-ចូរកំណត់ពេលវេលាអនុវត្ត ឬកម្រិតស្មុគស្មាញ $O()$ របស់ Algorithms នេះ។
យើងបាន

(ក)-កំណត់ $T(n)$

- statement₁ មានពេលវេលាអនុវត្ត 1
- statement2 មានពេលវេលាអនុវត្ត $n + 1$
- statement3 មានពេលវេលាអនុវត្ត n
- statement4 មានពេលវេលាអនុវត្ត 1

ដូចនេះពេលវេលាអនុវត្តសរុបរបស់ Algorithms នេះ ឬអនុវត្តពេលវេលារបស់វាគឺ

$$T(n) = 2n + 3$$

(ខ)-កំណត់ $O()$

ពេលវេលាអនុវត្ត ឬកម្រិតស្មុគស្មាញរបស់ $T(n) = 2n + 3$ គឺ $O(n)$

ឧទាហរណ៍៥៖

ឧបមាថាយើងមាន Algorithms របស់ចំណោទបញ្ជាគណនាផលគុណម៉ាទ្រីសដែលមាន n ជួរដេក និង n ជួរឈរ ត្រូវបានសរសេរដូចខាងក្រោម៖

- 1- for($i = 0 ; i < n ; i++$)
- 2- for($j = 0 ; j < n ; j++$)
- 3- scanf("% d %d ", & a [i] [j], & b [i] [j]);
- 4- for($i = 0 ; i < n ; i++$)
- 5- for($j = 0 ; j < n ; j++$)
- 6- C [i] [j] = 0 ;
- 7- for($k = 0 ; k < n ; k++$)
- 8- C [i] [j] = C [i] [j] + a [i] [k] * b [k] [j] ;
- 9- printf(" % d ", c [i] [j]);

យើងបាន

(ក)- កំណត់អនុគមន៍នៃពេលវេលាអនុវត្ត $T(n)$

- statement₁ មានពេលវេលាអនុវត្ត $(n+1)$ ដង

- statement2 មានពេលវេលាអនុវត្ត $n(n+1)$ ដង
- statement3 មានពេលវេលាអនុវត្ត n^2 ដង
- statement4 មានពេលវេលាអនុវត្ត $(n+1)$ ដង
- statement5 មានពេលវេលាអនុវត្ត $n(n+1)$ ដង
- statement6 មានពេលវេលាអនុវត្ត n^2 ដង
- statement7 មានពេលវេលាអនុវត្ត $n^2(n+1)$ ដង
- statement8 មានពេលវេលាអនុវត្ត n^3 ដង
- statement9 មានពេលវេលាអនុវត្ត n^2 ដង

ដូចនេះ ពេលវេលាអនុវត្តសរុប ឬអនុគមន៍ពេលវេលារបស់ Algorithms នេះគឺ៖

$$T(n) = n^3 + 4n^2 + 5n + 4$$

(ខ)-កំណត់ពេលវេលាអនុវត្ត ឬកម្រិតស្មុគស្មាញរបស់ Algorithms នេះគឺ $O(n^3)$

* យើងមានរបៀបកំណត់ $O()$ ម្យ៉ាងទៀតដោយប្រើគោលការណ៍នៃវិធានបូក និងវិធានគុណ

១-វិធានគុណ

ឧបមាយើងមានដំណាក់កាលអនុវត្ត Algorithms រៀងគ្នា $T_1(n) = O(f(n))$ និង $T_2(n) = O(g(n))$ ។

ដូចនេះពេលវេលាអនុវត្ត ឬកម្រិតស្មុគស្មាញរបស់វាគឺ

$$T(n) = O(f(n) + g(n))$$

២-វិធានបូក

ឧបមាយើងមានដំណាក់កាលអនុវត្ត Algorithms រៀងគ្នា $T_1(n) = O(f(n))$ និង $T_2(n) = O(g(n))$ ។

ដូចនេះពេលវេលាអនុវត្ត ឬកម្រិតស្មុគស្មាញរបស់វាគឺ

$$T(n) = O(\max(f(n), g(n)))$$

ប្រសិនបើ $f(n) \geq g(n)$

ឧទាហរណ៍៦

យើងមាន Statement ដូចខាងក្រោម៖

1- for(i = 1; i <= n ; i ++)

2- for(j = 1 ; j <= n ; j ++)

3- x = 2 * i ;

4- y = X * j ;

5- printf(" x = %d , y = %d ", x , y);

ចូរកំណត់ $O()$ របស់ Statement ខាងលើនេះដោយប្រើគោលការណ៍នៃវិធានបូក និងវិធានគុណ។

យើងបាន

កំណត់ $O()$ ដោយប្រើវិធានបូក និងវិធានគុណ

- Statement₁ មានពេលវេលាអនុវត្តជា ~~$O(n+1)$~~ $O(n)$
- Statement₂ មានពេលវេលាអនុវត្តជា ~~$O(n+1)*n$~~ $O(n^2)$
- Statement₃ មានពេលវេលាអនុវត្តជា ~~$O(1*n)$~~ $O(n)$
- Statement₄ មានពេលវេលាអនុវត្តជា ~~$O(1*n)$~~ $O(n)$
- Statement₅ មានពេលវេលាអនុវត្តជា $O(1)$

ដូចនេះពេលវេលាអនុវត្ត ឬកម្រិតស្មុគស្មាញរបស់វាគឺ

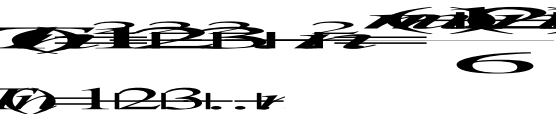
$$O(n^2)$$

លំហាត់អនុវត្ត

១-ចូរបង្ហាញពីលក្ខណៈសំខាន់ៗនៃការធ្វើ Top Down Design ក្នុងការដោះស្រាយបញ្ហាណាមួយ។

២-ចូរបង្ហាញពីលក្ខណៈសំខាន់ៗនៃការធ្វើ Stepwise Refinement ក្នុងការដោះស្រាយបញ្ហាណាមួយ។

៣-ចូរកំណត់កម្រិតស្មុគស្មាញរបស់ Statement ដូចខាងក្រោម៖

ក- 

ខ- (a)-

```
sum = 0 ;
for(i = 1 ; i <= n ; i ++ )
    sum = sum + i ;
```

(b)-

```
sum = 0 ;
for( i = 1 ; i <= n ; i ++ )
    sum = sum + j ;
```

(c)-

```
sum = 0 ;
for(i = 1 ; i <= n ; i ++ )
    for( j = 1 ; j <= i * i ; j ++ )
```

(d)-

```
sum = 0 ;
for( i = 1 ; i <= n ; i ++ )
    for(j = i ; j <= n * n ; j ++ )
        sum = sum + j ;
```

(e)-

```
sum = 0 ;
for(i = 1) ; i <= n ; i ++ )
    for ( j = n ; j >= i ; j --)
```

គ-ចូររកកម្រិតស្មុគស្មាញរបស់ Algorithms ដូចខាងក្រោម៖

(a)-

```
int sum(int a [ j ] , int n )
{
    int s, s1, i , j , k;
    s = 0 ;
    for(i = 0 ; i < n ; i ++ )
        for(j = 0 ; j < n ; j ++ )
            s1 = 0 ;
        for(k = 0 ; k <= j ; k ++ )
            s1 = s1 + a [ k ] ;
```

```

        if(s > s1)
            s = s1 ;
    return (s) ;
}

(b)-
void s(int n)
{
    while(n > 0)
    {
        printf( "% d", n% 2 ) ;
        n = n / 2 ;
    }
}

(c)-
for( i = 1; i <= n; i+ +)
    for( j = 1; j <= 2*i + 1; j+ +)
        s = s + i*j;

(d)-
void sort( float a[], long n )
{
    float temp;
    long i, j;
    for( i = 0; i < n - 1; i+ +)
        for( j = i + 1; j < n; j+ +)
            if(a[i]>a[j])
            {
                temp = a[i];
                a[i] = a[j];
                a[j] = temp;
            }
}

```

៤-ចូររកពេលវេលាអនុវត្ត និងកម្រិតស្មុគស្មាញ $O()$ របស់ Statement ដូចខាងក្រោម៖

- 1-for(i = 0; i < n ; i + +)
- 2- for(j = 0; j < n; j+ +)
- 3- scanf(“%d%d”, &a[i][j], &b[i][j]);



មេរៀនទី៣

Recursive និង Recursive Algorithms

3-1: និយមន័យ

Recursive គឺជាទ្រង់ទ្រាយទាំងឡាយណាដែលបានកើតឡើងក្រោមទ្រង់ទ្រាយរបស់ខ្លួន ឬជាទ្រង់ទ្រាយដែលមានខ្លួនវានៅក្នុងផងដែរ។

ឧទាហរណ៍៖

រូបភាពរបស់អ្នកព័ត៌មាននៅលើកញ្ចក់ទូរទស្សន៍ ដែលបានកើតចេញជាបន្តបន្ទាប់ បួនដប់រូបភាពគឺជា Recursive។

យើងមាននិយមន័យ Recursive របស់ចំណោទបញ្ហាមួយចំនួននៅក្នុងគណិតវិទ្យាដូចខាងក្រោម៖

១-ចំណោទបញ្ហា $n!$

+ បើ $n = 0$ នៅ $n! = 1$

+ បើ $n > 1$ នៅ $n! = n(n-1)!$

២-ចំណោទបញ្ហា $S(n) = 1 + 2 + 3 + \dots + n$

+ បើ $n = 1$ នៅ $S(n) = 1$

+ ផ្ទុយមកវិញ ($n > 1$) នៅ $S(n) = n + S(n-1)$

៣-ចំណោទបញ្ហាខ្សែលេខ Fibonacci(n)

+ បើ $n \leq 2$ នៅ $Fib(n) = 1$

+ ផ្ទុយមកវិញ ($n > 2$) នៅ $Fib(n) = Fib(n-1) + Fib(n-2)$

* **Recursive Algorithms** គឺជា Algorithms ដែលមានលំនាំដោះស្រាយមានន័យថា បើយើងចង់ដោះស្រាយបញ្ហា f គេត្រូវបន្តការដោះស្រាយបញ្ហា f ដោយបញ្ហា f មានលំនាំដោះស្រាយដូចបញ្ហា f ដែរ ប៉ុន្តែបញ្ហា f តូចជាងបញ្ហា f ហើយលំនាំដោះស្រាយនេះត្រូវបានអនុវត្តជាបន្តបន្ទាប់រហូតដល់បញ្ហាដែលអាចផ្តល់តម្លៃដោយផ្ទាល់ទើបឈប់។ ម៉្យាងទៀត Recursive Algorithms គឺជាអនុគមន៍ដែលបានហៅខ្លួនឯងមកប្រើនៅក្នុងខ្លួនឯង។

3-2: បញ្ហា Recursive

តើបញ្ហា Recursive មានលំនាំដំណើរការបែបណា? (បញ្ហា Recursive ជាបញ្ហាបែបណា) ដើម្បីស្វែងយល់អំពីបញ្ហានេះ យើងមានឧទាហរណ៍របស់ចំណោទបញ្ហាគណនាផលបូក n តួចំនួនគត់ $s(n) = 1 + 2 + 3 + \dots + n$ ត្រូវបង្ហាញចេញដូចខាងក្រោម៖

ចំពោះចំណោទ $s(n)$ ឧបមាយើងមានតម្លៃ $n = 9$ នោះផលបូក 9 តួចំនួនគត់

$$s(9) = 1 + 2 + 3 + 4 + 5 + 6 + 7 + 8 + 9 = 45$$

បើយើងបន្តអោយតម្លៃ $n = 10$ នោះផលបូក 10 តួចំនួនគត់

$$s(10) = 1 + 2 + 3 + \dots + 10 = 45 + 10 = 55$$

$$s(10) = 5(9) + 10$$

តាមលំនាំនេះយើងបានដំណាក់កាលវិភាគដូចមានខាងក្រោម៖

- ចង់ដោះស្រាយបញ្ហា $s(10)$ ជាដំបូងយើងត្រូវដោះស្រាយ $s(9)$ បន្ទាប់មកដោះស្រាយបញ្ហា

$$s(10) + s(9) \text{ បន្ទាប់មកដោះស្រាយបញ្ហា } s(10) = 10 + s(9)$$

- យើងបាន

-ចង់ដោះស្រាយ $s(n)$ ជាដំបូងត្រូវដោះស្រាយ $s(n-1)$ បន្ទាប់មកដោះស្រាយ

$$s(n) = n + s(n-1)$$

-ចង់ដោះស្រាយ $s(n-1)$ ជាដំបូងត្រូវដោះស្រាយ $s(n-2)$ បន្ទាប់មកដោះស្រាយ

$$s(n-1) = (n-1) + s(n-2)$$

.....

-ចង់ដោះស្រាយ $s(2)$ ជាដំបូងត្រូវដោះស្រាយ $s(1)$ បន្ទាប់មកដោះស្រាយ

$$s(2) = 2 + s(1)$$

-ជាចុងក្រោយយើងមាន $s(1) = 1$

- * ដំណាក់កាលអនុវត្តបញ្ហាស

-ដឹង $s(1) \Rightarrow$ ដោះស្រាយបាន $s(2) = s(1) + 2$

-ដឹង $s(2) \Rightarrow$ ដោះស្រាយបាន $s(3) = 3 + s(2)$

.....

-ដឹង $s(n-1) \Rightarrow$ ដោះស្រាយបាន $s(n) = n + s(n-1)$

យើងឃើញថា ចំណោទបញ្ហា Recursive គឺជាបញ្ហាត្រូវបានអនុវត្តតាម ២ ដំណាក់កាលគឺ

១-ដំណាក់កាលវិភាគ គឺជាដំណាក់កាលវិភាគចេញពីចំណោទបញ្ហាដើមទៅរក

ចំណោទបញ្ហាដែលមានទម្រង់រួម (មានលំនាំដោះស្រាយដូចបញ្ហាដើម និងមានទ្រង់ទ្រាយ

តូចជាងបញ្ហាដើម) ជាបន្តបន្ទាប់ហើយដំណាក់កាលវិភាគត្រូវបានបញ្ចប់នៅពេលវិភាគដល់បញ្ហាទម្រង់រួមមានលក្ខណៈផ្តល់តម្លៃដោយផ្ទាល់។

២-ដំណាក់កាលបញ្ជ្រាស គឺជាដំណាក់កាលដោះស្រាយដែលមានទម្រង់រួមដោយបំផុត(បញ្ហាដែលផ្តល់តម្លៃដោយផ្ទាល់) រហូតដល់បញ្ហាស្មុគស្មាញបំផុត(បញ្ហាដើម)។

3-3: អនុគមន៍ Recursive(Recursive Function / Recursive Algorithms)

យើងមានចំណោទបញ្ហាមួយចំនួនដែលមានលក្ខណៈ Recursive ត្រូវបានសរសេរជាអនុគមន៍ Recursive ដូចខាងក្រោម៖

១-ចំណោទបញ្ហា $s(n) = 1 + 2 + 3 + \dots + n$

```
int sum( int n )
{
    if(n == 1 )
        return(1) ;
    else
        return(n + sum(n-1)) ;
}
```

២-ចំណោទបញ្ហាខ្សែលេខ Fibonacci(n)

```
int Fibonacci(int n)
{
    if( n == 1 || n ==2)
        return(1);
    else
        return(Fibonacci(n-1) + Fibonacci(n-2) ) ;
}
```

៣-ចំណោទបញ្ហា $n!$

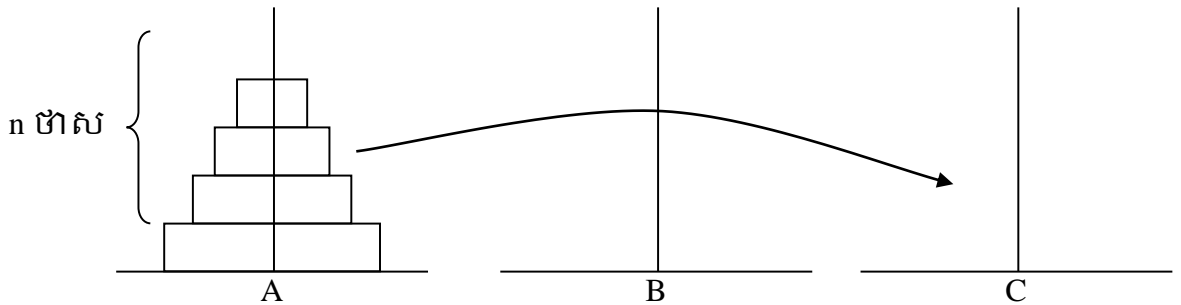
```
int factorial(int n)
{
    if(n == 0)
        return(1);
    else
        return(n * factorial(n - 1)) ;
}
```

៤-ចំណោទបញ្ហា a^n ($a > 0, a \neq 1$)

```
int calculate(int a , int n)
{
    if(n ==0)
        return(1);
    else
        return(a + calculate(a , n - 1)) ;
}
```

៥-ចំណោទបញ្ហា Tower of Game

ចំណោទនេះមានលក្ខណៈជាល្បែងដោយអនុវត្តការផ្លាស់ប្តូរ n ថាសពីបង្គោលមួយទៅកាន់បង្គោលមួយ ទៀត។



ចំណោទបញ្ហាមានលំនាំប្រើបង្គោលបី គឺ (A, B, C) ដោយមាន n ថាសនៅលើបង្គោល A ត្រូវបានតំរៀបតាមទំហំតូចចុះជាបន្តបន្ទាប់ដើម្បីអោយមានរាងជាកំពូលប្រាសាទ និងត្រូវអនុវត្តការផ្លាស់ប្តូរ n ថាសពី បង្គោល A ទៅកាន់បង្គោល C។

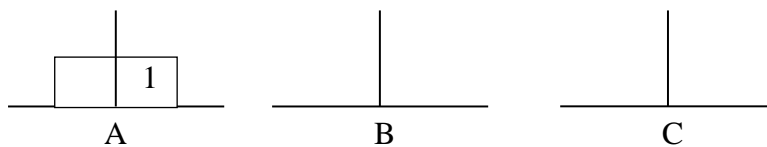
ដោយត្រូវគោរពលក្ខខណ្ឌរបស់ចំណោទមួយចំនួនដូចខាងក្រោម៖

-រាល់លើកនៃការផ្លាស់ប្តូរថាសអនុញ្ញាតឲ្យយកតែ ១ថាស។

-រាល់លើកនៃការផ្លាស់ប្តូរថាសមិនអនុញ្ញាតឲ្យយកថាសធំជាងដាក់ពីលើថាសតូច។

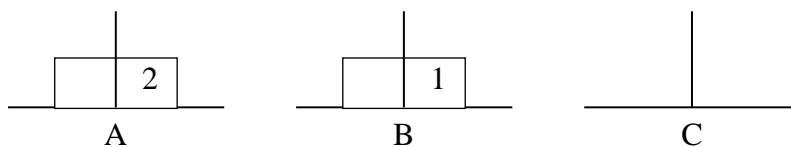
-អនុញ្ញាតឲ្យយកបង្គោល ១ ធ្វើជាបង្គោលសម្រាប់ផ្ញើតើបញ្ហា Tower of Game មានលក្ខណៈ Recursive បែបណា? យើងមានរបៀបអនុវត្តការផ្លាស់ប្តូរថាសជាមួយតម្លៃ n មួយចំនួនដូចខាងក្រោម៖

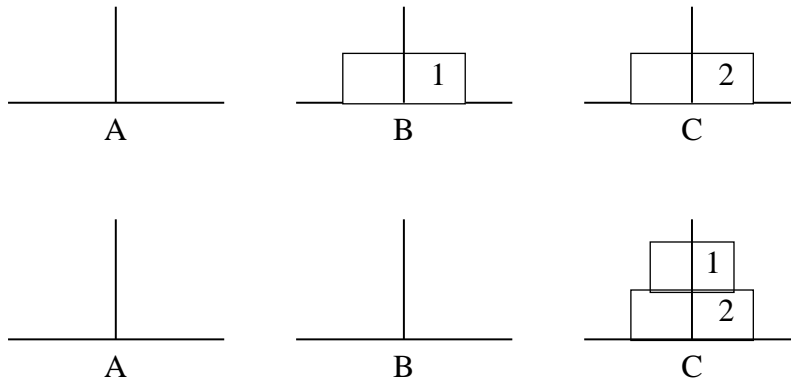
* ករណី $n=1$



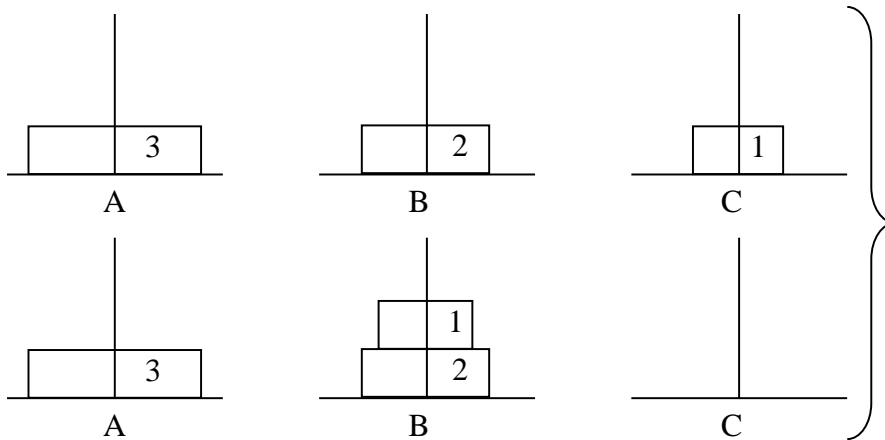
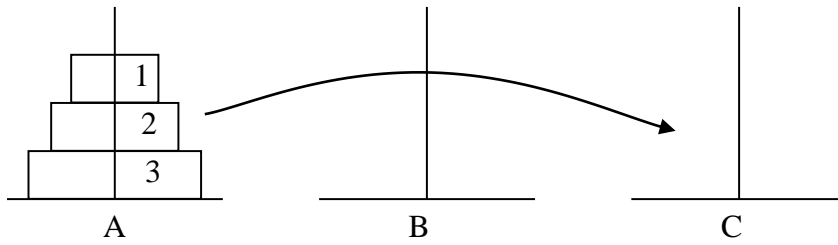
ផ្លាស់ប្តូរថាសដោយផ្ទាល់ពីបង្គោល A ទៅបង្គោល C

* ករណី $n=2$

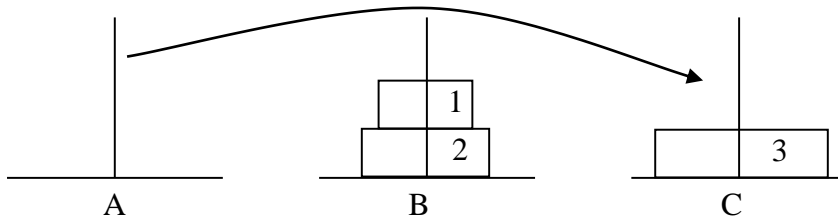




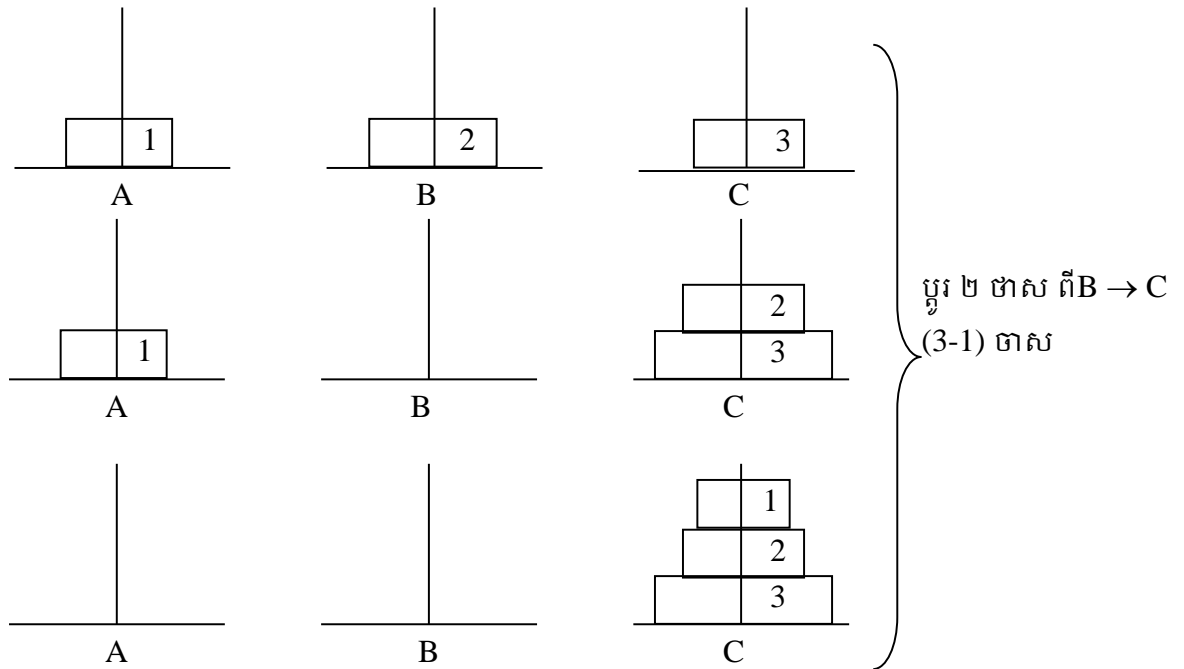
* ករណី $n = 3$



ប្តូរ ២ ថាសពី A ទៅ B
(3-1) ថាស



$A \rightarrow C$ ប្តូរថាស ទី៣ ពី
 $A \rightarrow C$



• យើងឃើញថាតាមលំនាំអនុវត្តនៃការផ្លាស់ប្តូរថាសខាងលើជាមួយតម្លៃត្រូវគ្នារបស់ n ថាស

មួយចំនួនគឺ

-ប្រសិនបើយើងចង់ប្តូរ n ថាសពីបង្គោល A ទៅបង្គោល C នោះអនុវត្តតាមដំណាក់កាលដូចខាងក្រោម៖

- + ប្តូរ $(n-1)$ ថាសពីបង្គោល A ទៅបង្គោល B
- + ប្តូរថាសទី n ពីបង្គោល A ទៅបង្គោល C
- + ប្តូរ $(n-1)$ ថាសពីបង្គោល B ទៅបង្គោល C
- ប្រសិនបើប្តូរ $(n-1)$ ថាសពី A ទៅ B នោះអនុវត្តតាម៖
- + ប្តូរ $(n-2)$ ថាសពី A ទៅ C
- + ប្តូរថាសទី $(n-1)$ ពី A ទៅ B
- + ប្តូរ $(n-2)$ ថាសពី C ទៅ B

ដូចនេះចំពោះចំណោទ Tower of Game គឺជាចំណោទបញ្ហា Recursive ដែលមានលក្ខណៈ

Recursive ដូចខាងក្រោម៖

+ ជាមួយ $n = 1$ ថាសគឺជាអនុវត្តន៍ការផ្លាស់ប្តូរថាសដោយផ្ទាល់ពីបង្គោល A ទៅបង្គោល C។

+ជាមួយ $n > 1$ ថាស គឺអនុគមន៍ការផ្លាស់ប្តូរថាសតាមបីដំណាក់កាលដូចខាងក្រោម៖

- ប្តូរ $(n-1)$ ថាសពីបង្គោល A ទៅ B ។
- ប្តូរថាសទី n ពីបង្គោល A ទៅ B ។
- ប្តូរ $(n-1)$ ថាសពីបង្គោល B ទៅ C ។

3-4: ~~ការវិភាគ~~ ធ្វើគំរោង Recursive Algorithms

ដើម្បីធ្វើការវិភាគលើ Recursive Algorithms យើងត្រូវអនុវត្តតាមដំណាក់កាលដូចខាងក្រោម៖

1-សរសេរអនុគមន៍ Recursive ។

2-គូស Recursive Trees ឬគូស Structure Trees តាមលំនាំដំណើរ Recursive របស់ចំណោទបញ្ហាដែលតាង Node ។

3-វិភាគការទៅលើចាប់យក Memory គឺសមាមាត្រជាមួយ Level ខ្ពស់បំផុតរបស់ Recursive Trees ។

4-វិភាគពេលវេលាអនុវត្ត ឬកម្រិតស្មុគស្មាញ គឺសមាមាត្រជាមួយចំនួន Node ដែលបានដំណើរការ Recursive នៅលើ Recursive Trees ។

ឧទាហរណ៍៖

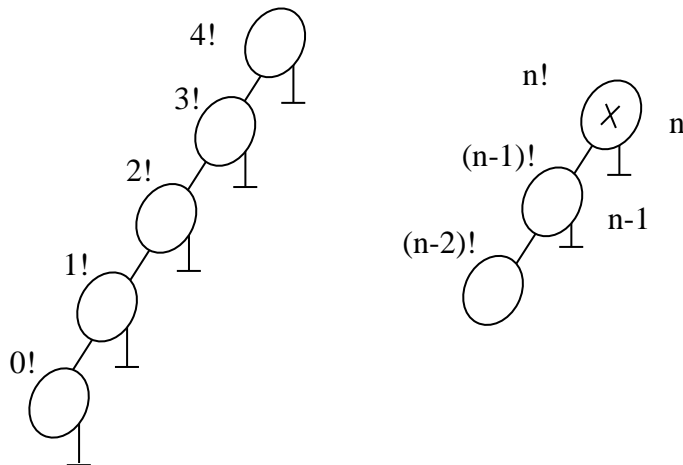
ចូរវិភាគ Recursive Algorithms របស់បញ្ហា $n!$ ដើម្បីធ្វើការវិភាគ Recursive Algorithms របស់ $n!$ យើងត្រូវអនុវត្តតាមដំណាក់កាលដូចខាងក្រោម៖

១-សរសេរអនុគមន៍ Recursive

```
int factorial(int n)
{
    if(n == 0)
        return(1);
    else
        return(n * factorial (n-1));
}
```

២-គូស Recursive Trees

$n = 4$



៣-វិភាគចាប់យក Memory គឺជាសមាមាត្រជាមួយ Level ខ្ពស់បំផុតរបស់ Recursive Trees នៃចំណោទបញ្ហា $n!$ ដោយ $n = 4$ យើងមាន 5 Levels \rightarrow ការចាប់យក Memory របស់ $n!$ ស្មើ $(n + 1) \cdot 1$

៤-វិភាគពេលវេលាអនុវត្ត គឺជាសមាមាត្រជាមួយចំនួន Node ដែលបានដំណើរការ Recursive នៅលើ Recursive Trees ដោយ $n = 4$ យើងមានដំណើរការ Recursive ៥ដង ដូចនេះពេលវេលាអនុវត្ត Recursive របស់ $n!$ ស្មើ $n + 1$ គឺ $O(n) \cdot 1$

ឧទាហរណ៍១៖

ចូរសរសេរកម្មវិធីដើម្បីរក n តួនៃស៊េរី Fibonacci ដោយប្រើអនុគមន៍ Recursion ដូចខាងក្រោម៖

```
#include< stdio.h >
void main( )
{
    int n, i;
    clrscr( );
    printf("Input term number");
    scanf("%d", &n);
    printf("Fibonacci series upto %d term is \n", n);
    for(i = 1; i <= n; i++)
        printf("%d", fibo(i));
}

int fibo(int n)
{
    int f;
```



```

        if(n == 1 || n == 2)
            return(1);
        else
        {
            f = fibo(n - 1) + fibo(n - 2);
            return(f);
        }
    }
}

```

នៅពេល **RUN** កម្មវិធីបង្ហាញដូចខាងក្រោម៖

Input term number 15

Fibonacci series upto 15 term is

1 1 2 3 5 8 13 21 34 55 89 144 233 377 610

ឧទាហរណ៍២៖

ចូរសរសេរកម្មវិធីដើម្បីគណនា n Factorial ដោយប្រើអនុគមន៍ Recursive ដូចខាងក្រោម៖

```

#include< stdio.h >
int factorial( int m );
void main( )
{
    int n, fact;
    printf("\nGive element number");
    scanf("%d", &n);
    fact = factorial(n);
    printf("\nFactorial of %d is %d", n, fact);
}

/* Recursion code for factorial */
int factorial(m)
int m;
{
    int f;
    if(m == 0)
        return(1);
    else
        f = m * factorial(m - 1);
    return(f);
}

```

ឧទាហរណ៍៣៖

សរសេរកម្មវិធីដើម្បីបង្ហាញសញ្ញាផ្កាយជាតារាងត្រីកោណដោយប្រើអនុគមន៍

Recursive ដូចខាងក្រោម៖

```
#include< stdio.h >
#include< conio.h >
void main( )
{
    int n, j;
    clrscr( );
    printf("\nInput number of rows");
    scanf("%d", &n);
    display(n);
}
void display(int m)
{
    if(m < 0)
        return;
    else
    {
        printf("\n");
        write(m);
        display(m - 1);
    }
}
void write(int p)
{
    if(p <= 0)
        return;
    else
    {
        printf("*");
        printf("\n");
    }
}
```

នៅពេល RUN កម្មវិធីបង្ហាញដូចខាងក្រោម៖

```
Input number of rows 10
*****
*****
*****
*****
*****
*****
*****
*****
*****
*****
```

ឧទាហរណ៍៤៖

ចូរសរសេរកម្មវិធីដើម្បីប្តូរពីអក្សរតូចទៅអក្សរធំ និងពីអក្សរធំទៅតូចដោយប្រើ

Recursive ដូចខាងក្រោម៖

```
# include< stdio.h >
# include< stdlib.h >
# include< ctype.h >
void main(int argc, char *argv[])
{
    int (*convcase[2])(int) = {toupper, tolower};
    int func;
    int result = EXIT_SUCCESS;
    int ch;
    if(argc > 0)
    {
        if(toupper((unsigned char)argv[0][0]) == 'U')
        {
            func = 0;
        }
        else
        {
            func = 1;
        }
        while((ch = getchar( )) != EOF)
        {
            ch = (*convcase[func])((unsigned char)ch);
            putchar(ch);
        }
    }
    else
    {
        fprintf(stderr, "Unknown name. Can't decide what to do.\n");
        result = EXIT_FAILURE;
    }
}
```

ឧទាហរណ៍៥៖

ចូរសរសេរកម្មវិធីដើម្បីរក PGCD នៃពីរចំនួនដោយប្រើអនុគមន៍ Recursive

ដូចខាងក្រោម៖

```
#include< stdio.h >
void main( )
{
    int a, b, pgcd;
    int hcf( );

    printf("Type in any 2 numbers whose PGCD is to be found\n");
    scanf("%d %d", &a, &b);
    pgcd = hcf( a, b );
    printf("PGCD of %4d and %4d is %4d\n", a, b, pgcd);
}
int hcf( p, q )
{
    int p, q;
    int r, factor ;

    r = p - (p/q * q);
    if( r == 0 )
        return(q);
    else
        return hcf(q, r);
}
```

នៅពេល RUN កម្មវិធីបង្ហាញដូចខាងក្រោម៖

```
Type in any 2 numbers whose PGCD is to be found
4      8

PGCD of    4 and 8 is    4
```

នៅពេល RUN កម្មវិធីបង្ហាញដូចខាងក្រោម៖

```
Type in any 2 numbers whose PGCD is to be found
13    45

PGCD of    13 and 45 is    1
```

ឧទាហរណ៍៦៖

ចូរសរសេរកម្មវិធីដើម្បីរកតម្លៃធំបំផុត និងតូចបំផុតដោយប្រើអនុគមន៍ Recursive។

```
# include< stdio.h >

void main( )
{
    int i, n, m, maxim, minim, a[50];
    int max( );
    int min( );

    printf("Finding the maximum and minimum recursively\n\n");
    printf("Number of elements in vector?");
    scanf("%d", &n);
    printf("%d", n);

    printf("\n\nVector elements?\n");
    for( i = 0; i < n; i ++ )
        scanf("%d", &a[i]);
    for( i = 0; i < n; i ++ )
        printf("%5d", a[i]);
    printf("\n\n");
    i = 0;
    m = n - 1;
    maxim = max( i, m, a );
    printf("\nMaximum of %d elements is %d\n", n, maxim);
    printf("\n");
    i = 0;
    m = n - 1;
    minim = min( i, m, a );
    printf("\nMiniimum of %d elements is %d\n", n, minim);
    printf("\n");
}
```

/* អនុគមន៍ដើម្បីរកតម្លៃធំបំផុត*/

```
int max( j, n, a )
{
    int j, n, a[50];
    int m1, m2;
    if( n == 0 )
        return(a[j]);
    else
    {
        m1 = max( j, n/2, a );
        m2 = max( j, n/2 + 1; n/2, a );
        if( m1 < m2 )
            return(m2);
        else
            return(m1);
    }
}
```

/* អនុគមន៍ដើម្បីរកតម្លៃតូចបំផុត*/

```
int min( j, n, a )
{
    int j, n, a[50];
    int m1, m2;
    if( n == 0 )
        return(a[ j ]);
    else
    {
        m1 = min( j, n/2, a );
        m2 = min( j, n/2 + 1; n/2, a );
        if( m1 < m2 )
            return(m1);
        else
            return(m2);
    }
}
```

នៅពេល **RUN** កម្មវិធីបង្ហាញដូចខាងក្រោម៖

Finding the maximum and minimum in vector recursively

Number of elements in vector ? 8

Vector elements ?

12 34 56 78 54 -12 0 89

Maximum of 8 elements is 89

Minimum of 8 elements is -12

លំហាត់អនុវត្ត

I- ចូរធ្វើគម្រោង និងសរសេរ Recursive Algorithms

១-គណនា $Sum = 1.2 + 2.3 + \dots + n(n+1)$

២-Sum = $a^1 + a^2 + \dots a^n$

៣-Sum = $1^2 + 2^2 + 3^2 + \dots + n^2$

៤-គណនាផលគុណនៃពីរចំនួនគត់វិជ្ជមាន។

៥-គណនាផលបូកគ្រប់ធាតុរបស់ Array ដែលមាន n ធាតុ

$a[0] + a[1] + a[2] + \dots + a[n-1]$

៦-ត្រលប់នៃចំនួនគត់វិជ្ជមានមួយ (167 → 761)

៧-គណនាអាយុរបស់នរណាម្នាក់។ បើគេស្គាល់ Birth , Year និង Current year។

៨-គណនាអនុគមន៍ Fibonacci (Rabbits Problem) ។

៩-ចូរធ្វើគម្រោង និងសរសេរ Recursive Algorithms និងរកកម្រិតស្មុគស្មាញរបស់ Recursive Algorithms បំប្លែងចំនួនគត់វិជ្ជមានមួយពីប្រព័ន្ធគោល ១០ ទៅប្រព័ន្ធគោល ២។

១០-ចូរធ្វើគម្រោង និងសរសេរ Recursive Algorithms ដើម្បីគណនាតម្លៃ PGCD នៃ ២ ចំនួនគត់វិជ្ជមាន។

១១-To find the sum of two nonnegative integers recursive $sum = a + b$

```
int sum (int a , int b)
{
    if(a == 0) return (b)
    else return (sum ( - a , + b ) ) ;
}
```

១២-Recurring (n) number of character.

```
hello !! → !! olleh
void rev (int)
{
    char c;
    if (n == 1)
    {
        c = getchar ( ) ;
        c = getchar ( ) ;
        putchar (c) ;
    }
    else
    {
        c = getchar ( ) ;
        c = getchar ( ) ;
```



```

        rev( - n );
        putchar( c );
    }
}

```

II-គេហោយ Algorithms ដូចខាងក្រោម៖

```

long sum( long n )
{
    long sum = 0;
    for( int i = 1; i <= n; i++ )
        sum = sum + (i - 1)*pow(i,i);
    return( sum );
}

```

ក-ចូរសរសេរទម្រង់ផលបូកដែលបានមកពីលទ្ធផលរបស់ Algorithm ខាងលើ។

ខ-ចូរសរសេរជា Recursive Algorithm សមមូលនឹង Algorithm ខាងលើ។

គ-ចូររកកម្រិតស្មុគស្មាញរបស់ Recursive Algorithm សំនួរ ខ ខាងលើ។

ឃ-ចូរវិភាគពីដំណើរការ របស់ Recursive Algorithm ខាងលើក្នុងករណី $n = 4$ ។

III-

ក-ចូរសរសេរ Recursive Algorithm ដើម្បីគណនា

$$\text{Sum} = 1*2 + 2*3 + 3*4 + \dots + n*(n+1)$$

ខ-ចូរវិភាគពីដំណើរការអនុវត្តន៍របស់ Recursive Algorithm ខាងលើក្នុងករណី $n=5$ ។

IV-គេហោយ Algorithms ដូចខាងក្រោម៖

```

int sum( long n )
{
    int sum = 0;
    for( int i = 1; i <= n; i++ )
        sum = sum + (3*i - 1);
    return( sum );
}

```

ក-ចូរសរសេរទម្រង់ផលបូកដែលបានមកពីលទ្ធផលរបស់ Algorithm ខាងលើ។

ខ-ចូរសរសេរជា Recursive Algorithm សមមូលនឹង Algorithm ខាងលើ។

គ-ចូររកកម្រិតស្មុគស្មាញរបស់ Recursive Algorithm សំនួរ ខ ខាងលើ។

V-ចូរសរសេរ Recursive Algorithm ដើម្បី

ក-រក PGCD នៃពីរចំនួនគត់។

ខ-គណនាផលបូក $\text{sum} = n^1 + n^2 + n^3 + \dots + n^n$ ។



មេរៀនទី៤

Array , Stack , and Queue

4-1: Array

4-1-1: និយមន័យ

Array គឺជាសំណុំធាតុ ដែលមានឈ្មោះរួម មានប្រភេទទិន្នន័យដូចគ្នា និង មាន Index កំណត់ទុកជាមុន ហើយរាល់ធាតុនីមួយៗរបស់ Array n ធាតុត្រូវបានផ្ទុកជាបន្តបន្ទាប់តាមលក្ខណៈវិច័យទីតាំង Index = 0 រហូតដល់ទីតាំងចុងក្រោយរបស់ Array

(index = n - 1) ។

ឧទាហរណ៍១

int a [20] ;

a_0	a_1	a_2	a_3	...	a_{18}	a_{19}
[0]	[1]					[19]

ឧទាហរណ៍២

float b[3][2] ;

b ₀₀	b ₀₁	b ₁₀	b ₁₁	b ₂₀	b ₂₁
[0]	[1]	[2]	[3]	[4]	[5]

4-1-2:របៀបផ្ទុកធាតុរបស់ Array

Array គឺជា Structure ដែលអាចគណនា Address បាន និងមានរបៀបផ្ទុកជាបន្តបន្ទាប់តាមលក្ខណៈវិច័យទីតាំង គឺធាតុដំបូង a[0] ផ្ទុកត្រង់ទីតាំងដំបូង (index=0) ធាតុ a[1] ផ្ទុកត្រង់ទីតាំងទី 1 , ... ធាតុ a[i] ផ្ទុកត្រង់ទី i ...និងធាតុ a[n-1] ផ្ទុកត្រង់ទីតាំង n-1

(index = n - 1) ។

ឧបមាយើងមាន Array មួយវិមាត្រ (វិច័យទី) a[i] ដែលមាន n ធាតុហើយរាល់ធាតុនីមួយៗត្រូវការ Memory ទំហំ C word នោះ Array a[i] ត្រូវការ Memory location (C x n) words ជាបន្តបន្ទាប់ និងមានរបៀបផ្ទុកធាតុនីមួយៗរបស់វាដូចខាងក្រោម៖

a_0	a_1	a_2	a_3	a_4	a_5	a_i	a_{n-1}
Lo: address ដំបូង					c word				

ឧបមាថាយើងមាន Array ពីរវិមាត្រដែលជាម៉ាទ្រីស $a[i][j]$ ដែលមាន 3 ជួរដេក និង 4 ជួរឈរ នោះយើងមានរបៀបផ្ទុកធាតុនីមួយៗនៃម៉ាទ្រីស $a[i][j]$ តាមលំនាំដូចខាងក្រោម៖

$$i \rightarrow \begin{pmatrix} 0 & 1 & 2 & 3 \\ a_{00} & a_{01} & a_{02} & a_{03} \\ a_{10} & a_{11} & a_{12} & a_{13} \\ a_{20} & a_{21} & a_{22} & a_{23} \end{pmatrix} j \downarrow$$

១-ផ្ទុកតាមជួរដេកជាចំបង (Row major order)

a_{00}	a_{01}	a_{02}	a_{03}	a_{10}	a_{12}	a_{13}	a_{20}	a_{21}	a_{22}	a_{23}
----------	----------	----------	----------	----------	----------	----------	----------	----------	----------	----------

២-ផ្ទុកតាមជួរឈរជាចំបង (Column major order)

a_{00}	a_{10}	a_{21}	a_{01}	a_{11}	a_{21}	a_{02}	a_{12}	a_{22}	a_{03}	a_{13}	a_{23}
----------	----------	----------	----------	----------	----------	----------	----------	----------	----------	----------	----------

ដូចនេះយើងមានរបៀបផ្ទុកធាតុនីមួយៗនៃ Array n វិមាត្រ

$$A([s_0][s_1] \dots [s_{n-1}]) \text{ ជាមួយ } Lb_i \leq s_i \leq Ub_i$$

ដែល

Lb_i គឺជា Lower bound

s_i គឺជា index of array

Ub_i គឺជា upper bound

$$i = 0, 1, 2, \dots, n-1$$

ហើយធាតុនីមួយៗរបស់វាចាប់យក Memory c word និងមាន Address ដំបូងជា Lo គឺមានលំនាំផ្ទុក២របៀបដូចខាងក្រោម៖

១-របៀបផ្ទុកតាមជួរដេកជាចំបង (Row major order) គឺផ្ទុកអស់ពីជួរដេកមួយចូលដល់ជួរដេកមួយទៀត ដោយអនុវត្តការផ្ទុកជាបន្តបន្ទាប់ពីជួរដេកដំបូងរហូតដល់ជួរដេកចុងក្រោយនៃ Array។

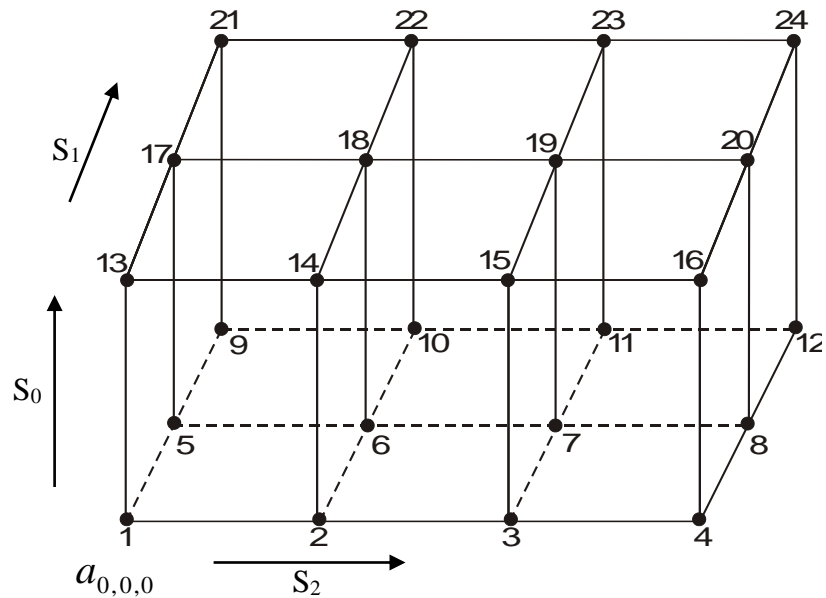
២-របៀបផ្ទុកតាមជួរឈរជាចំបង (Column major order) គឺផ្ទុកអស់ពីជួរឈរមួយចូលដល់ជួរឈរមួយទៀត ដោយអនុវត្តការផ្ទុកជាបន្តបន្ទាប់ពីជួរឈរដំបូងរហូតដល់ជួរឈរចុងក្រោយនៃ Array។

ឧទាហរណ៍៖

ចូរសរសេររបៀបផ្ទុកធាតុនីមួយៗនៃ Array បីវិមាត្រ $a[s_0][s_1][s_2]$

ជាមួយ ~~$a_{0,0,0}$~~ ហើយធាតុនីមួយៗរបស់វាចាប់យក

Memory 1 word និងមាន Address ដំបូងស្មើ 100។



១-របៀបផ្ទុកតាមជួរដេកជាចំបង (Row major order)

a_{000} a_{001} a_{002} a_{003}

a_{010} a_{011} a_{012} a_{013}

a_{020} a_{021} a_{022} a_{023}

a_{100} a_{101} a_{102} a_{103}

២-របៀបផ្ទុកតាមជួរឈរជាចំបង (Column major order)

a_{000} a_{001} a_{010} a_{110}

a_{020} a_{120} a_{001} a_{101}

a_{011} a_{111} a_{021} a_{121}

a_{002} a_{102} a_{012} a_{112}

a_{022} a_{122} a_{003} a_{103}

a_{013} a_{113} a_{023} a_{123}

4-1-3:របៀបគណនា Address របស់ Array

(i)-Array មួយវិមាត្រ (One dimensional Array)

-Address នៃធាតុ $a[i]$ ត្រូវបានគណនាតាមរូបមន្តដូចខាងក្រោម៖

$$\text{Loc}(a[i]) = L_0 + C \cdot i$$

ដែល

- $\text{Loc}(a[i])$ ជា Address នៃធាតុរបស់ $a[i]$
- L_0 ជា Address ដើម (Address នៃធាតុ $a[0]$)
- C ជាទំហំនៃធាតុនីមួយៗរបស់ Array ដែលចាប់យក (គិតជា word)
- i ជា Index

ឧទាហរណ៍១៖

float $a[20]$; 4 bytes = 2 words

int $a[50]$; 2 bytes = 1 words

ឧទាហរណ៍២៖

គេមាន Array មួយវិមាត្រ float $a[45]$; ហើយ Address ដើមមានតម្លៃស្មើ 182 ។

ចូរគណនា Address នៃធាតុទី 18។

គណនា Address នៃធាតុ $a[18]$

តាមរូបមន្ត $\text{Loc}(a[i]) = L_0 + C \cdot i$

ដូចនេះ $\text{Loc}(a[18]) = 182 + 2 \cdot 18 = 218$

* float = 4 bytes , int = 2bytes ហើយ 1 word = 2 bytes \rightarrow float = 2 words

(ii)-Array ពីរវិមាត្រ (Two dimensional Array)

- Array នៃធាតុ $a[i][j]$ ត្រូវបានគណនាតាមរូបមន្តដូចខាងក្រោម៖

$$\text{Loc}(a[i][j]) = L_0 + C \cdot (n \cdot i + j) \quad (\text{តាមជួរដេក})$$

$$\text{Loc}(a[i][j]) = L_0 + C \cdot (m \cdot j + i) \quad (\text{តាមជួរឈរ})$$

ដែល - m គឺជាជួរដេក

- n គឺជាជួរឈរ

ឧទាហរណ៍៖

គេមាន Array ពីរវិមាត្រ $\text{int } a[20][15]$; បើ Address ដើមមានតម្លៃ 246 ។
ចូរគណនា Address នៃធាតុ $a[7][9]$ តាមជួរដេក និងជួរឈរ។
យើងបាន

-តាមជួរដេក

$$\text{ដូចនេះ } \text{Loca}[1][9] = 246 + 1 * (15 * 7 + 9) = 360$$

- តាមជួរឈរ

$$\text{ដូចនេះ } \text{Loca}[1][9] = 246 + 1 * (20 * 9 + 7) = 433$$

(iii)-Array បីវិមាត្រ (Three dimensional Array)

-Address នៃធាតុ $a[i][j][k]$ ត្រូវបានគណនាតាមរូបមន្តដូចខាងក្រោម៖

$$\text{Loc } a[i][j][k] = \text{Lo} + p_1 * (i - Lb_1) * c_1 + p_2 * (j - Lb_2) * c_2 + p_3 * (k - Lb_3) * c_3$$

(iv)-Array n វិមាត្រ (n dimensional array)

$$A([s_0][s_1] \dots [s_{n-1}]) \text{ ជាមួយ } Lb_i \leq s_i \leq Ub_i$$

ដែល

Lb_i គឺជាគោលក្រោម

Ub_i គឺជាគោលលើ

s_i គឺជា Index របស់ Array ($i = 0, 1, 2, \dots, n-1$)

ហើយរាល់ធាតុនីមួយៗរបស់វាចាប់យក C word និងមាន Address ដំបូងជា Lo ។

គេមានរូបមន្តសម្រាប់គណនា Address របស់ធាតុនីមួយៗនៃ Array នេះតាមលំនាំ ២ របៀបដូចខាងក្រោម៖

១-របៀបគណនា Address តាមជួរដេក

$a[s_0][s_1] \dots [s_{n-1}]$ តាមជួរដេកជាចំបង (Row major order)

$$\text{Loc } (a[s_0][s_1] \dots [s_{n-1}]) = Lo + \sum_{i=0}^{n-1} [p_i * (s_i - Lb_i) * c_i]$$

ក្នុងនោះ៖ 

ឧទាហរណ៍៖

គេមាន Array ពីរវិមាត្រ $a[s_0][s_1]$ ជាមួយ ~~$0 \leq s_0 \leq 7$~~ ហើយរាល់ធាតុនីមួយៗរបស់វាចាប់យក 1 word និងមាន Address ដំបូងស្មើ 100។

(ក)-ចូរសរសេររបៀបផ្ទុកធាតុនីមួយៗនៃ Array នេះតាមជួរដេក។

(ខ)-ចូរគណនា Address នៃធាតុ $a[2][5]$ និង $a[3][4]$ តាមជួរដេក។

ចម្លើយ

យើងបាន

(ក)-របៀបផ្ទុកធាតុនីមួយៗនៃ Array តាមជួរដេកគឺ

$a_{00} a_{01} a_{02} a_{03} a_{04} a_{05} a_{06}$

$a_{10} a_{11} a_{12} a_{13} a_{14} a_{15} a_{16}$

$a_{20} a_{21} a_{22} a_{23} a_{24} a_{25} a_{26}$

$a_{30} a_{31} a_{32} a_{33} a_{34} a_{35} a_{36}$

$a_{40} a_{41} a_{42} a_{43} a_{44} a_{45} a_{46}$

(ខ)-គណនា Address នៃធាតុ $a[2][5]$ និង $a[3][4]$ តាមជួរដេក

ដោយ



ដែល

$$P_1 = 1$$

$$P_2 = 1$$

$$P_3 = 1$$

និង $P_1 = 1$

$$\Rightarrow \text{Loc}(a[2][5]) = 100 + [7 * (2 - 0) * 1 + 1 * (5 - 0) * 1] = 119$$

$$\Rightarrow \text{Loc}(a[3][4]) = 100 + [7 * (3 - 0) * 1 + 1 * (4 - 0) * 1] = 125$$

២-របៀបគណនា Address តាមជួរឈរ

$a[s_0][s_1] \dots [s_{n-1}]$ តាមជួរឈរជាចំបង (Column major order)

$$Loc(a[s_0][s_1] \dots [s_{n-1}]) = Lo + \sum_{i=n-1}^0 [p_i * (s_i - Lb_i) * c]$$

ក្នុងនោះ:

$$p_i = \prod_{j=i+1}^{n-1} c_j$$

ឧទាហរណ៍១៖

Write a program to store 10 salesmen's amount in an array and find out total sale and best sales amount.

/* Storing 10 sales amount in an array and find out total sale and Best sale amount */

```
#include <stdio.h>
void main( )
{
    const int n = 10 ;
    int sale_amt [100] ;
    int tot_amt ;
    int best = 0 ;

    /* storing values in an array */(ផ្ទុកតំលៃនៅក្នុង Array)
    for( i = 0 ; i < n ; i ++ )
    {
        printf( " Enter sales amount : \n" ) ;
        scanf( "%d", &sale_amt[ i ] ) ;
    }

    /* Calculate the total sale and find out best amount */
    tot_sale = 0 ;
    best = 0 ;
    for( i = 0 ; i < n ; i ++ )
    {
        printf( " \n sale amount  %d = %d " , i, sale_amt [i] ) ;
        tot_amt = tot_amt + sale_amt[i] ;
        if( sale_amt[i] > best )
            best = sale_amt[i] ;
    }

    /* Printing Total and Best Sale amount */
    printf( "\n Total sale amount = %d " , tot_amt ) ;
    printf( "\n Best sale amount = %d " , best ) ;
}
```


ឧទាហរណ៍២៖

ចូរសរសេរកម្មវិធីដើម្បីបញ្ចូល N ចំនួន រួចតំរៀបតាមលំដាប់ពីតូចទៅធំ។

```
/* Sorting an array */
# include <stdio.h>
# include <conio.h>
# include <iostream.h>
void main( )
{
    int num[100], i ,j, temp, n ;
    printf( " Enter the number of observations : " ) ;
    scanf( " %d " , &n ) ;
    /* Entering value of Observation */
    printf( " Enter the observations = \n ) ;
    for( i = 0 ; i < n ; i ++ )
    {
        scanf( " %d " , &num[i] ) ;
    }
    /* Sorting observations in ascending order */
    for( i = 0 ; i < n ; i ++ )
    {
        for( j = i + 1 ; j < n ; j ++ )
        {
            if( num[i] > num[j] )
            {
                temp =num[i] ;
                num[i] = num[j] ;
                num[j] = temp ;
            }
        }
    }
    /* Printing sorted array */
    printf( " Observations in Ascending order \n ) ;
    for( i = 0 ; i < n ; i ++ )
        printf( " %d \n " , num[i] ) ;
}
```

ឧទាហរណ៍៣៖

ចូរសរសេរកម្មវិធីដើម្បីស្វែងរក Item នៅក្នុង Array [Linear search] ។

```
# include < stdio.h>
# include < iostream.h>
# include < conio.h>
# include < stdlib.h>
# include < iomanip.h>
void main( )
{
    /* Program for linear search for given number */
```

```

int a[100], i , n, sc, key, pos ;
printf( " Enter the array limit : \n" ) ;
scanf( " %d " , &n ) ;

/* Enter array values */(បញ្ចូលធាតុរបស់ Array)

printf( " Enter the elements values \n " ) ;
for( i = 0 ; i < n ; i ++ )
    scanf( " %d " , &a[i] ) ;
printf( " Enter the key value for searching : \n " ) ;
scanf( " %d " , &key ) ;

/* Linear search */
sc = 0 ;
for( i = 0 ; i < n ; i ++ )
{
    if( a[i] == key )
    {
        sc = 1 ;
        pos = i ;
    }
}
if( sc == 1 )
    printf( " \n The element is found in location : %d " , pos + 1 ) ;
else
    printf( " \n Given value is not found " ) ;
}

```

ឧទាហរណ៍៤៖

ចូរសរសេរកម្មវិធីដើម្បីកំណត់លំដាប់នៃម៉ាទ្រីស និងគណនាផលបូកធាតុទាំងអស់របស់ម៉ាទ្រីស។

```

# include <stdio.h >
# include < conio.h >
# include < iostream.h >
# include < stdlib.h >
# include < iomanip.h >
void main( )
{
    /* Array declaration */(ប្រកាស Array)

    int a[10][10] ;
    int i, j, sum = 0, m, n ;
    printf(" Enter the order of matrix \n " ) ;
    printf(" Enter Row Range: \n " ) ;
}

```

```
scanf( " %d " , &m ) ;
printf(" Enter Col Range: \n " ) ;
scanf( " %d " , &n ) ;
/* Storing values in an array */(តំរៀបធាតុរូបសំ Array)
printf(" Enter Elements values \n" ) ;
for( i = 0 ; i < m ; i++ )
{
    for( j = 0 ; j < n ; j++ )
        scanf( " %d " , &a[ i ][ j ] ) ;
}
/* Printing matrix and finding sum of elements */
printf(" Printing given matrix \n " ) ;
for( i = 0 ; i < m ; i++ )
{
    for( j = 0 ; j < n ; j++ )
    {
        printf( " %d " , &a[ i ][ j ] ) ;
        sum = sum + a[ i ][ j ] ;
    }
    printf( " \n " ) ;
}
printf( " Sum of all elements = %d " , sum ) ;
}
```

ឧទាហរណ៍៥

ចូរសរសេរកម្មវិធីដើម្បីកំណត់តម្លៃ **Pointer** ដោយប្រើប្រមាណវិធី **&** និង *****។

```
#include<iostream.h>
#include<conio.h>
void main( )
{
    int *x, y;      /* x is pointer to integer variable */
    clrscr( );
    y = 10;
    x = &y;
    printf(" Address of y = %d\n", &y);
    printf(" Value of y = %d\n", y);
    printf(" Address of y = %d\n", x);
    printf(" Value of y = %d\n", *x);
}
```

នៅពេល **RUN** កម្មវិធីបង្ហាញដូចខាងក្រោម៖

Address of y = 65555

Value of y = 10

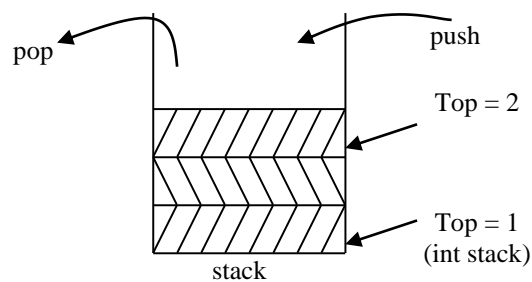
Address of y = 65555

Value of y = 10

4-2: Stack

4-2-1: និយមន័យ

Stack គឺជាទម្រង់ List ពិសេសមួយដែលរាល់ប្រមាណវិធីបន្ថែមធាតុ (push) និងរាល់ប្រមាណវិធីដកយកធាតុ (pop) របស់ stack ត្រូវបានអនុវត្តនៅ ផ្នែកម្ខាងនៃកំពូល Stack ត្រូវបានហៅថា Top។ ម៉្យាងទៀត Stack ត្រូវបានហៅថាទម្រង់ List ដែលមានលក្ខណៈ LIFO (Last In First Out) មានន័យថាធាតុចូលក្រោយត្រូវបានយកចេញមុន។ យើងមានទម្រង់ទូទៅរបស់ Stack ដូចខាងក្រោម៖



LIFO(Last Int First Out)

* ការផ្ទុក Stack ជា Array

គេអាចផ្ទុក Stack លើ Array មួយវិមាត្រដែលមាន n ផ្នែកនៃ Memory បន្តបន្ទាប់គ្នា បើ Address នៃផ្នែក Top របស់ Stack តាងដោយ Top នោះគេបានដូចរូបរាងក្រោម៖

s_0	s_1	s_2	s_3	s_4	...	s_{n-2}	s_{n-1}
					...	s	

* ការបង្កើត Data Structures សម្រាប់ Stack

```
# define MaxStack ...
typedef int element type ;
struct stack type
{
    int Top ;
    element type s[ MaxStack ] ;
} ;
```

+ បើ st ជា Stack នោះត្រូវប្រកាស Stack type st ;

4-2-2: ការប្រើប្រាស់លើ Stack

-Stack ត្រូវបានប្រើសម្រាប់ដោះស្រាយបញ្ហាដែលមានលក្ខណៈ LIFO ។

-Stack ត្រូវបានគេប្រើប្រាស់សម្រាប់ការងារ Compiler របស់ម៉ាស៊ីន ។

-Stack ត្រូវបានគេប្រើសម្រាប់ដោះស្រាយបញ្ហាតាមបែប Recursive

មានន័យថាវាអាចជំនួសឱ្យការងារ Recursive ។

- Stack ត្រូវបានគេប្រើសម្រាប់អនុវត្តការងារហៅអនុគមន៍នៅក្នុងកម្មវិធី ។

4-2-3: ប្រមាណវិធីលើ Stack

យើងមានប្រមាណវិធីមួយចំនួនរបស់ Stack ដូចខាងក្រោម៖

* របៀបបង្កើត Data Structures

Stack ត្រូវបានប្រកាសជាទម្រង់ struct ដែលរួមមាន 2 fields គឺ

-field Top ជាអថេរសម្រាប់ចង្អុលទៅកាន់ធាតុស្ថិតនៅកំពូល Stack ។

-field Node សម្រាប់ផ្ទុកទិន្នន័យរបស់ធាតុនីមួយៗនៅក្នុង Stack ។

ទម្រង់ List របស់ Stack ត្រូវបានអនុវត្តជាមួយ Array រឺ Pointer ប៉ុន្តែក្នុងមេរៀននេះ យើងប្រើ Array ។

```
# define MaxStack...
struct stack
{
    int Top ;
    data type Node[ MaxStack] ;
};
struct stack st, a [ 100 ] , * pt ;
```

* ប្រមាណវិធី Initialize

សម្រាប់បង្កើត Stack ទំនេរគឺ

Top = -1

* ប្រមាណវិធី Full Stack

សម្រាប់ត្រួតពិនិត្យមើល Stack ពេញគឺ

Top = Maxstack -1

*ប្រមាណវិធី push

សម្រាប់បន្ថែមធាតុ Item ណាមួយចូលទៅក្នុង Stack ត្រង់ទីតាំង Top ដោយអនុវត្តតាមពីរដំណាក់កាលគឺ៖

-ប្រសិន Full Stack នោះបង្ហាញប្រាប់ Stack Overflow

-ប្រសិន Not fullstack នោះ៖

- បង្កើត Top មួយទីតាំងគឺ

$$\text{Top} = \text{Top} + 1$$

- ផ្ទេរតម្លៃ Item ចូល Stack ត្រង់ទីតាំង Top

$$\text{Node}[\text{Top}] = \text{Item};$$

ឧទាហរណ៍១៖

គេមានប្រមាណវិធីលើ Stack ជាមួយទំហំរបស់វាស្មើ ៥ ដូចខាងក្រោម៖

- Initialize(stack)

[0]	[1]	[2]	[3]	[4]

↑
Top = -1

- push(stack) , ' A ')

A				
---	--	--	--	--

↑
Top = 0

- push(stack , ' B ') ; push(stack , ' M ') ;

A	B	M		
---	---	---	--	--

↑
Top

* ប្រមាណវិធី pop

សម្រាប់ដកធាតុចេញពី Stack ដោយអនុវត្តតាមពីរដំណាក់កាលគឺ

+ប្រសិន Stack ទំនេរនោះ Stack Underflow

+ផ្ទុយមកវិញនោះអនុវត្ត

- ដកធាតុត្រង់ទីតាំង Top

$$\text{Element} = \text{Node}[\text{Top}] ;$$

- បន្ថយ Top មួយទីតាំង

$$\text{Top} = \text{Top} - 1;$$

ឧទាហរណ៍២៖

គេមានទម្រង់ Stack ដូចខាងក្រោម៖

40	60	20	45	
[0]	[1]	[2]	[3]	[4]

Top = 3

• pop(stack)

40	60	20	45	

Top = 2

គេអាចសរសេរ Algorithms នៃប្រមាណវិធីរបស់ Stack ដោយប្រើ C Language ដូចខាងក្រោម៖

1-Algorithms សម្រាប់បង្កើត Stack ទំនេរ។

```
void initialize(struct stack * ps)
{
    ps → Top = -1
}
```

2-Algorithms សម្រាប់បន្ថែមធាតុ Item ណាមួយចូល Stack។

```
void push (struct stack * ps , datatype Item)
{
    if (ps → Top == MaxStack -1)
        printf ("\n stack overflow) ;
    else
    {
        ps → Top = ps → Top + 1 ;
        ps → Node[ ps → Top] = Item;
    }
}
```

3- Algorithms សម្រាប់ដកយកធាតុ Item ណាមួយចេញពី Stack ។

```
datatype pop (struct stack * ps)
{
    datatype Element ;
    if (ps → Top == -1)
    {
        print f("\n stack underflow") ;
        return(0);
    }
    else
    {
        Element = ps → Node [ ps → Top ] ;
        ps → Top = ps → Top - 1 ;
        return ( Element ) ;
    }
}
```

4-clear stack

```
void clear_stack( stacktype * st )
{
    st → Top = -1
}
```

5- Algorithms សម្រាប់ត្រួតពិនិត្យមើល Stack ទំនេរ

```
int empty_stack(stacktype st )
{
    return ( st-> Top == -1 );
}
```

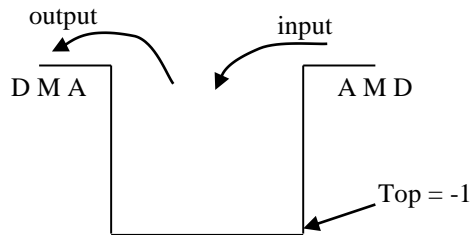
6- Algorithms សម្រាប់ត្រួតពិនិត្យមើល Stack ពេញ។

```
int full_stack(stacktype st )
{
    return ( st.Top == MaxStack ) ;
}
```


4-2-4:អនុវត្តលើStack

ឧទាហរណ៍១៖

ចូរសរសេរ Algorithms សម្រាប់បង្ហាញចេញនូវតម្លៃបញ្ជីសរសេររបស់តួអក្សរដែលបានវាយបញ្ចូលពី keyboard ដោយប្រើលក្ខណៈ Stack



- បញ្ចូលតួអក្សរពី keyboard ជាបន្តបន្ទាប់រហូតដល់តម្លៃកំណត់ណាមួយ។
- បង្កើត Stack ទំនើប : Initialize(stack)។
- បញ្ចូលធាតុនីមួយៗទៅក្នុង Stack : push(stack, item)។
- ដកយកធាតុនីមួយៗចេញពី Stack : pop(stack)។
- បង្ហាញចេញធាតុនីមួយៗរបស់ Stack នៅលើអេក្រង់ (screen)។

ចូរសរសេរកម្មវិធីសម្រាប់អនុវត្តលើការងារ Algorithms ខាងលើ។

```
#define MaxStack 100
struct stack
{
    int Top , char Node[ MaxStack] ;
};
void initialize (struct stack * ps)
{
    ps → Top = -1)
}

void push (struct stack * ps , char item)/*អនុគមន៍ដើម្បីបញ្ចូលធាតុចេញពី Stack*/
{
    if (ps → Top == MaxStack -1)
        print f( "\n stack overflow ") ;
    else {
        ps → Top = ps →Top + 1 ;
        ps → Node [ ps → Top] = item ;
    }
}

char pop (struct stack * ps) /*អនុគមន៍ដើម្បីដកយកធាតុចេញពី Stack*/
{
    char Element ;
```

```

        if (ps → Top == -1 )
        {
            print f ("\n stack underflow ");
            return(0);
        }
    else
    {
        Element = ps → Node [ ps → Top ] ;

        ps → Top = ps → Top -1 ;

        return (Element) ;

    }

}

void main ( )
{
    struct stack st; char ch; char a [ MaxStack] ;
    int pos, i ;
    do
    {
        initialize ( & st)
        print f("\n input character : " ) ;
        pos = 0 ;
    } while (a [ pos + + ] = getchar ( ) != ' s ' ;
    a [ - - pos] = '\0' ;
    for (i = 0 ; i < pos ; i + + )
        push (& st, a [ i ] );
    print f( "\n Reversed character : ");
    while (st. Top != 1)
        print f("%4c ") pop (&st) );
    do {
        print f("\n Do you want to continue ( Y / N): ");
        ch = getch ( ) ;
    } while (ch == ' y ' || ch == ' Y ');
    getch( ) ;
}

```

ឧទាហរណ៍២៖

អនុគមន៍ដើម្បីបញ្ចូល integer item

```
void push( int item, int *top, int s[ ] )
{
    if( *top == STACK_SIZE - 1 )
    {
        printf( " Stack overflow \n " ) ;
        return ;
    }
    s[ ++ ( *top ) ] = item ; /* Increment top and then insert an item */
}
```

ឧទាហរណ៍៣៖

អនុគមន៍សម្រាប់បញ្ចូល character item

```
void push( char item, int *top, char s[ ] )
{
    if( *top == STACK_SIZE - 1 )
    {
        printf( "Stack overflow \n " ) ;
        return ;
    }
    s[ ++ ( *top ) ] = item ; /* Insert an item on the stack */
}
```

ឧទាហរណ៍៤៖

អនុគមន៍សម្រាប់លុប integer item

```
int pop( int *top, int s[ ] )
{
    int item ;
    if( *top == -1 )
    {
        return ( 0 ) ; /* Indicates empty stack */
    }
    item = s[ ( *top ) - - ] ; /* Access the item and delete */
    return( item ) ; /* Send the item deleted to the calling function */
}
```

ឧទាហរណ៍៥៖

អនុគមន៍សម្រាប់លុប character item ។

```
char pop( int *top, char s[ ] )
{
    char item ;
    if( *top == -1 )
    {
        return( 0 ) ;    /* Indicates empty stack */
    }
    item = s[ ( *top ) - - ] ;    /* Access the item and delete */
    return( item ) ;    /* Send the item deleted to the calling function */
}
```

ឧទាហរណ៍៦៖

ចូរសរសេរអនុគមន៍មួយសម្រាប់បង្ហាញតម្លៃរបស់ Stack ។

C function to display the contents of the stack

```
void display( int top, int s[ ] )
{
    int i ;
    if( top == -1 )
    {
        printf( "Stack is empty" ) ;
        return ;
    }
    printf( "Contents of the stack \n" ) ;
    for( i = 0 ; i <= top ; i ++ )
    {
        printf( "%d \n" , s[ i ] ) ;
    }
}
```

ឧទាហរណ៍៧៖

ចូរសរសេរកម្មវិធីមួយដើម្បីអនុវត្តលើ Stack ដោយប្រើ Array ។

C Program to implement the stack using Array

```
# include <stdio.h >
# include < process.h >
# define STACK_SIZE 5
void main( )
{
    int    top ;           /* Points to top of the stack */
    int    s[ 10 ] ;      /* Holds the stack items */
    int    item ;          /* Item to be inserted or deleted item */
    int    choice ;        /* User choice for push, pop, and display */
    top = -1 ;             /* Stack is empty to start with */
    for( ; ; )
    {
        printf( "1: Push 2: Pop \n " ) ;
        printf( "2: Display 4: Exit \n ) ;
        printf( "Enter the choice \n " ) ;
        scanf( "%d", &choice ) ;
        switch( choice )
        {
            case 1:
                printf( "Enter the item to be inserted \n" ) ;
                scanf( "%d", &item ) ;
                push( item, &top, s ) ;
                break ;
            case 2:
                item = pop( &top, s ) ;
                if( item == 0 )
                    printf( " Stack is empty \n " ) ;
                else
                    printf( " Item deleted = %d\n " , item ) ;
                break ;
            case 3:
                display( top, s ) ;
                break ;
            default:
                exit( 0 ) ;
        }
    }
}
```

ឧទាហរណ៍៨៖

ចូរសរសេរកម្មវិធីមួយសម្រាប់សម្តែងប្រតិបត្តិការលើ Stack ដោយប្រើ Structure ។

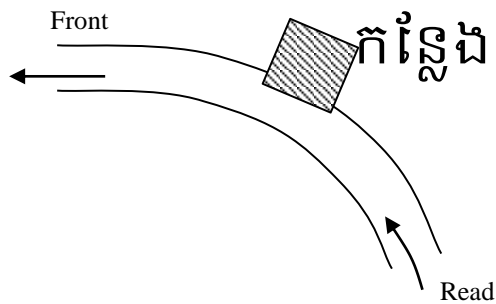
C program to simulate the stack operations using structures

```
# include < stdio.h >
# include < process.h >
# define STACK_SIZE 5
struct stack
{
    int item[ STACK_SIZE ] ;
    int top ;
};
typedef struct stack STACK ;
void main( )
{
    int item ;          /* Item to be inserted */
    int choice ;        /* Push, pop, display or quit */
    STACK s ;          /* To store items */
    s.top = -1 ;        /* Stack is empty initially */
    for( ; ; )
    {
        printf( "1: Push 2: Pop\n" ) ;
        printf( " 3: Display 4: Exit\n" ) ;
        printf( " Enter the choice\n" ) ;
        scanf( " %d ", &choice ) ;
        switch( choice )
        {
            case 1 :
                printf( " Enter the item to be inserted\n" ) ;
                scanf( "%d", &item ) ;
                push( item, &s ) ;
                break ;
            case 2 :
                item = pop( &s ) ;
                if( item != 0 )
                {
                    printf( Item deleted = %d \n ", item ) ;
                }
                Break ;
            case 3 :
                display( s ) ;
                break ;
            default : exit( 0 ) ;
        }
    }
}
```

4-3: Queue

4-3-1: និយមន័យ

Queue គឺជាទម្រង់ List ពិសេសមួយដែលរាល់ការធ្វើប្រមាណវិធីរបស់វាត្រូវបានអនុវត្តតាមជួរមុខ និងជួរក្រោយ ហើយរាល់ការបន្ថែមធាតុចូលក្នុង Queue (Insert) ត្រូវបានអនុវត្តនៅជួរក្រោយ (Rear) និងរាល់ការដកយកធាតុចេញពី Queue (Remove) ត្រូវបានអនុវត្តនៅជួរមុខ (front)។ ម៉្យាងទៀត Queue ត្រូវបានហៅថាទម្រង់ List ដែលមានលក្ខណៈ FIFO (first in first out) មានន័យថាធាតុចូលមុនត្រូវបានយកចេញមុន។ យើងមានទម្រង់ទូទៅរបស់ Queue មានដូចខាងក្រោម៖



4-3-2: ការប្រើប្រាស់លើ Queue

- Queue គឺត្រូវបានប្រើសម្រាប់ដោះស្រាយបញ្ហាដែលមានលក្ខណៈ FIFO (first in first out)។
- Queue គឺត្រូវបានប្រើសម្រាប់ដំណើរការអនុវត្តនៅក្នុងប្រព័ន្ធធនាគារ។
- Queue គឺត្រូវបានប្រើនៅក្នុងដំណើរការរបស់ Printer ដែលបានភ្ជាប់ប្រព័ន្ធនetwork។

4-3-3: ប្រមាណវិធីលើ Queue

យើងមានប្រមាណវិធីមួយចំនួនរបស់ Queue ត្រូវបានអនុវត្តតាមគោលការណ៍ដំណើរការបីយ៉ាងដូចខាងក្រោម៖

១-គោលការណ៍ដំណើរការទី១

យើងមានប្រមាណវិធីមួយចំនួនរបស់ Queue ត្រូវបានអនុវត្តតាមគោលការណ៍ដំណើរការទី១ ដូចខាងក្រោម៖

-របៀបបង្កើត Data Structures

```
# define MaxQueue...
struct Queue
{
    int Rear ;
    int front ;
    datatype Node[ MaxQueue] ;
};
```

-ប្រមាណវិធី Initialize

សម្រាប់បង្កើត Queue ទំនេរគឺ

Rear = -1 និង front = 0

-ប្រមាណវិធី Full Queue

សម្រាប់ត្រួតពិនិត្យមើល Queue ពេញគឺ

Rear = [MaxQueue]-1

-ប្រមាណវិធី Insert

សម្រាប់បន្ថែមធាតុ Item ណាមួយចូលក្នុង Queue ដោយអនុវត្ត

* ប្រសិន Not Full Queue

+ បង្កើត Rear មួយទីតាំងគឺ

Rear = Rear +1

+ ផ្ទេរតម្លៃ Item ចូល Queue ត្រង់ទីតាំង Rear

Node[Rear] = item

* ប្រសិន Full Queue នោះបង្ហាញប្រាប់ Queue Overflow ។

-ប្រមាណវិធី Empty Queue

សម្រាប់ត្រួតពិនិត្យមើល Queue ទំនេរគឺ

Front > Rear

-ប្រមាណវិធី Remove

សម្រាប់ដកយកធាតុចេញពី Queue ត្រង់ទីតាំង Front ដោយអនុវត្ត

* ប្រសិន Empty Queue នោះបង្ហាញប្រាប់ Queue Underflow

* ប្រសិន Not Empty Queue នោះ

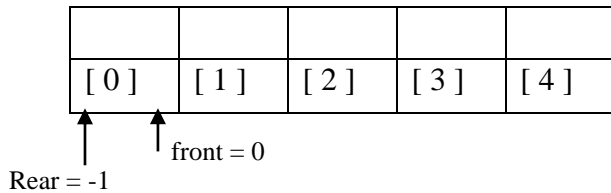
+ ដកយកធាតុត្រង់ទីតាំង Front ។

+ បង្កើន Front មួយទីតាំង។

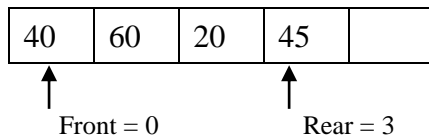
ឧទាហរណ៍:

យើងមានទម្រង់ Queue ដូចខាងក្រោម៖

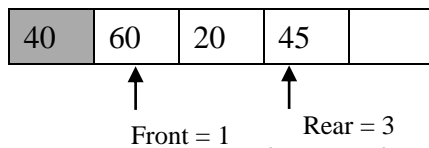
• Initialize(Queue)



• Insert : 40 , 30 , 20 , 45



• Remove(Queue)



២-គោលការណ៍ដំណើរការទី២

លំនាំដំណើរការទី២ មានលក្ខណៈខុសពីគោលការណ៍ដំណើរការទី១ត្រង់ពេលអនុវត្តប្រមាណវិធី Remove និងអថេរ Front របស់គោលការណ៍ដំណើរការទី២ ជានិច្ចកាលស្មើសូន្យ។

* របៀបបង្កើត Data Structures

ប្រើអថេរ Rear សម្រាប់ចង្អុលនៅកាន់ធាតុស្ថិតនៅជួរក្រោយ និងអថេរ Node សម្រាប់ផ្ទុកទិន្នន័យនីមួយៗរបស់ Queue ។

```
# define MaxQueue...

struct Queue
{
    int Rear ;
    datatype Node[ MaxQueue ] ;
};
```

* ប្រមាណវិធី Initialize

សម្រាប់បង្កើត Queue ទំនេរគឺ $Rear = -1$

			...		
[0]	[1]	[2]			[Max Queue - 1]

$Rear = -1$

void Initialize (struct Queue * pq)

```
{
    pq → Rear = -1 ;
}
```

* ប្រមាណវិធី Insert

សម្រាប់បន្ថែមធាតុ item ចូលក្នុង Queue ដោយអនុវត្តតាមពីរជំណាក់កាល

+ ប្រសិន Full Queue នោះបង្ហាញប្រាប់ Queue Overflow ។

+ ផ្ទុយមកវិញ (Not Full Queue) នោះ៖

- បង្កើត Rear មួយទីតាំងគឺ

$Rear = Rear + 1$

- បន្ថែមធាតុ Item ត្រង់ទីតាំង Rear គឺ

$Node[Rear] = Item$

ឧទាហរណ៍៖

យើងមានទម្រង់ Queue ដូចខាងក្រោម

[0]	[1]	[2]	[3]

$Rear = -1$

Insert(Queue)

20			
----	--	--	--

$Rear = 0$

20	40		
----	----	--	--

$Rear = 1$

20	40	30	
----	----	----	--

$Rear = 2$

/*អនុគមន៍សម្រាប់បន្ថែមធាតុចូលទៅក្នុង Queue*/

```
void insert (struct Queue * pq , datatype item)
{
    if (pq → Rear == Max Queue -1)
        print f("\n Queue overflow");
    else
    {
        pq → Rear = pq → Rear + 1 ;
        pq → Node [ pq → Rear ] = item;
    }
}
```

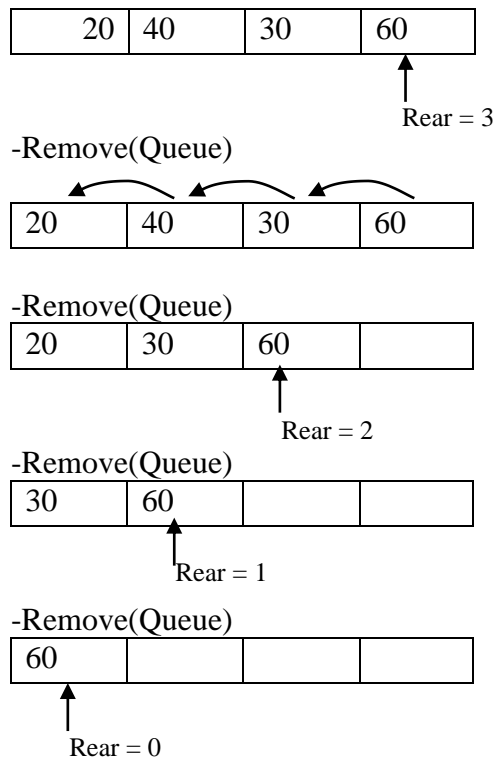
*ប្រមាណវិធី Remove

សម្រាប់ដកធាតុចេញពីជួរមុខរបស់ Queue ត្រង់ទីតាំងដំបូង (Node[0]) និងធ្វើការរំកិលធាតុនីមួយៗរបស់ Queue ទៅកាន់ទីតាំងមុខបន្ទាប់មួយទីតាំងនិងឱ្យអថេរ

Rear = Rear -1 ។

ឧទាហរណ៍៖

យើងមានទម្រង់ Queue ដូចខាងក្រោម៖



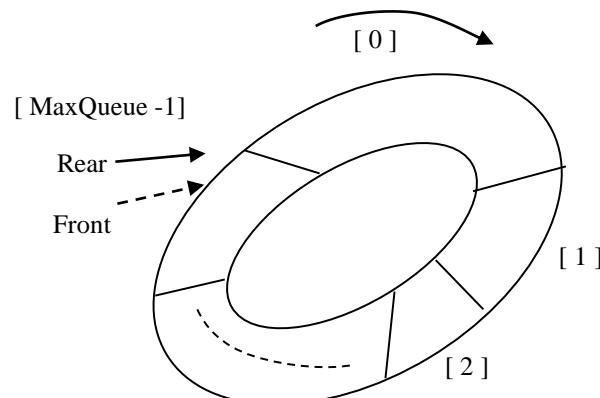
/*អនុគមន៍សម្រាប់ដកយកធាតុចេញពី Queue*/

```
datatype Remove (struct Queue * pq )
{
    if (pq → Rear == -1 )
    {
        ... ;
    }
    else
    {
        datatype Element = Node [0] ;
        for (int i = 0 ; i < pq → Rear -1 ; i ++ )
            pq → Node [ i ] = pq → Node [ i + 1 ] ;
        pq → Rear = pq → Rear -1 ;
    }
}
```

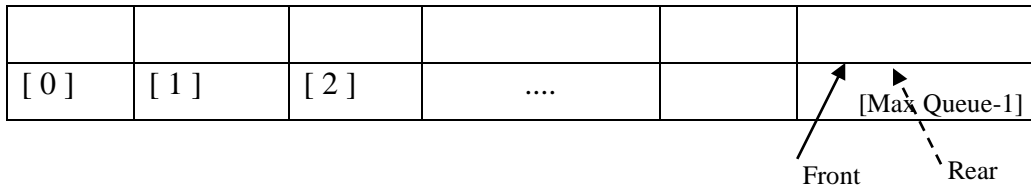
៣-គោលការណ៍ដំណើរការទី៣

លំនាំដំណើរការទី៣របស់ Queue ត្រូវបានគេនិយមប្រើជាងគោលការណ៍ដំណើរការទី១ និងគោលការណ៍ដំណើរការទី២ ដោយគោលការណ៍នេះមានលក្ខណៈដំណើរការប្រសើរជាង គឺវាអាចដោះស្រាយបាននូវគុណវិបត្តិរបស់គោលការណ៍ដំណើរការទី១ និងគោលការណ៍ដំណើរការទី២របស់ Queue ។

គោលការណ៍ដំណើរការទី៣របស់ Queue ត្រូវបានគេប្រើ Structure Queue ដែលមានលក្ខណៈជា Circle Array ដោយចាត់ទុកធាតុស្ថិតនៅទីតាំងដំបូងនៃ Array (Node[0]) គឺនៅបន្ទាប់ពីធាតុស្ថិតនៅទីតាំងចុងក្រោយនៃ Array (Node[MaxQueue – 1]) ។



ទម្រង់ Circle Array របស់គោលការណ៍ដំណើរការទី៣ នៅពេលដំណើរការអនុវត្តត្រូវគូសតាមលំនាំដូចខាងក្រោម៖



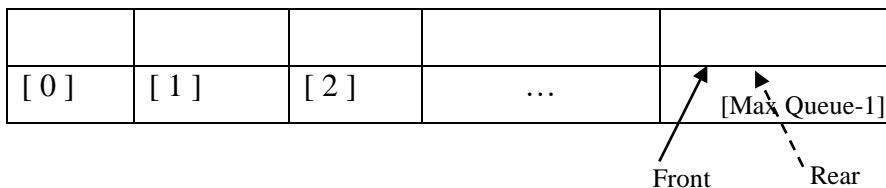
ហើយរាល់ទីតាំងរបស់វាដែលមានលំនាំទំហំ MaxQueue អាចប្រើប្រាស់បានតែ ចំនួន MaxQueue -1 ប៉ុណ្ណោះដោយទុកមួយទីតាំងអោយនៅទំនេរសម្រាប់ដំណើរការរបស់ វា។ ក្នុងលំនាំនេះអថេរ Front អាចធំជាងអថេរ Rear ដោយអថេរ Front និង Rear ចាប់ផ្តើម ដំណើរការដំបូងពីទីតាំង MaxQueue -1 ។

យើងមានប្រមាណវិធីមួយចំនួនរបស់ Queue តាមគោលការណ៍ដំណើរការទី៣ដូច ខាងក្រោម៖

* ប្រមាណវិធី Initialize

សម្រាប់បង្កើត Queue ទំនេរគឺ

$$\text{Rear} = \text{front} = \text{MaxQueue} - 1$$



* ប្រមាណវិធី Insert

សម្រាប់បន្ថែមធាតុ Item ណាមួយចូល Queue ដោយអនុវត្ត

+ ប្រសិន Rear = MaxQueue -1 នោះអោយអថេរ

$$\text{Rear} = 0$$

+ ផ្ទុយមកវិញ Rear # MaxQueue -1 នោះអោយអថេរ

$$\text{Rear} = \text{Rear} + 1$$

+ ប្រសិន Rear = Front នោះ

Queue Overflow

+ ផ្ទុយមកវិញ Rear = Front នោះបន្ថែមធាតុ item ចូល Queue ត្រង់ទីតាំង Rear

$$\text{Node}[\text{Rear}] = \text{Item}$$

* ប្រមាណ Remove

សម្រាប់ដកឯកតាចេញពី Queue ដោយអនុវត្ត

+ ប្រសិន Front # MaxQueue -1 នោះអោយអថេរ Front = 0

- ផ្ទុយមកវិញ

Front = Front +1

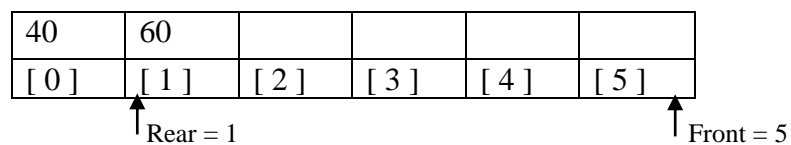
- ដកឯកតាចេញត្រង់ទីតាំង Front

Element = Node[Front]

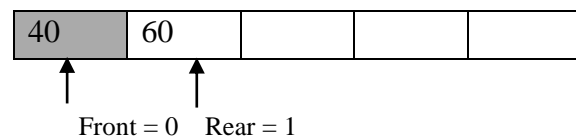
+ ផ្ទុយមកវិញនោះ Queue Underflow

ឧទាហរណ៍៖

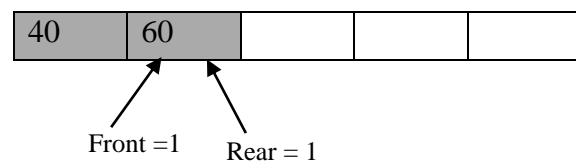
យើងមានទម្រង់ Queue ដូចខាងក្រោម៖



-Remove(Queue)



-Remove(Queue)



ឧទាហរណ៍១៖

អនុគមន៍ដើម្បីត្រួតពិនិត្យ Queue ពេញរឺមិនពេញ។

int qfull(int r)

{

/* returns true if queue is full otherwise return false */

return(r == QUEUE_SIZE - 1) ? 1 : 0 ;

}

ឧទាហរណ៍២៖

អនុគមន៍សម្រាប់បញ្ចូល item នៅខាងក្រោយនៃ Queue។

```
void insert_rear( int item, int q[ ], int *r )
{
    if( qfull( *r ) )          /* Is queue full ? */
    {
        printf( " Queue overflow \n " );
        return ;
    }
    /* Queue is not full */
    Q[ ++ ( *r ) ] = item ;      /* Update rear pointer and insert an item */
}
```

ឧទាហរណ៍៣៖

អនុគមន៍ដើម្បីត្រួតពិនិត្យ Queue ទំទេរ។

```
int qempty( int f, int r )
{
    /* returns true if queue is empty otherwise return false */
    return( f > r ) ? 1 : 0 ;
}
```

ឧទាហរណ៍៤៖

អនុគមន៍ដើម្បីលុប item ចេញពីមុខនៃ Queue។

```
void delete_front( int q[ ], int *f, int *r )
{
    if( qempty( *f, *r ) )
    {
        printf( " Queue underflow \n " );
        return ;
    }
    printf( " The element deleted is %d \n ", q[ ( *f ) ++ ] );
    if( *f > *r )
    {
        *f = 0 ;
        *r = -1 ;
    }
}
```

ឧទាហរណ៍៥៖

អនុគមន៍សម្រាប់បង្ហាញតម្លៃនៃ Queue ។

Function to display the contents of queue

```
void display( int q[ ], int f, int r )
{
    int i ;
    if( qempty( f, r )
    {
        printf( " Queue is empty \n " ) ;
        return ;
    }
    printf( " Contents of queue is \n " ) ;
    for( i = f ; i <= r ; i ++ )
        printf( " %d \n ", q[ i ] ) ;
}
```

ឧទាហរណ៍៦៖

ចូរសរសេរកម្មវិធីជាភាសា C ដើម្បីអនុវត្តនូវ Queue ធម្មតា។

```
# include < stdio.h >
# include < process.h >
# define QLTEUE_SIZE 5
void main( )
{
    int choice, item, f, r, q[ 10 ] ;
    /* Queue is empty */
    f = 0 ;          /* Front end of Queue */
    r = -1 ;         /* Rear end of queue */
    for( ; ; )
    {
        printf( " 1: Insert 2: Delete \n " ) ;
        printf( " 3: Display 4: Exit \n " ) ;
        printf( " Enter the choice \n " ) ;
        scanf( " %d ", &choice ) ;
        switch( choice )
        {
            case 1 :
                printf( " Enter the item to be inserted \n" ) ;
                scanf( " %d ", &item ) ;
                insert_rear( item, q, &r ) ;
                break ;
            case 2:
```



```

        delete_front( q, &f, &r ) ;
        break ;
    case 3:
        display(q, f, r ) ;
        break ;
    default :
        exit( 0 ) ;
    }
}
}

```

ឧទាហរណ៍៧៖

ចូរសរសេរកម្មវិធីជាភាសា C ដើម្បីអនុវត្តនូវ Double-end Queue ។

C program to implement double-end queue

```

#include < stdio.h >
#include < process.h >
#define QUEUE_SIZE 5
/* Include function to check for overflow */
/* Include function to check for underflow */
/* Include function to insert an item at the front end */
/* Include function to insert an item at the rear end */
/* Include function to delete an item at the front end */
/* Include function to delete an item at the rear end */
/* Include function to display the contents of queue */
void main( )
{
    int choice, item, f, r, q[10] ;
    f = 0 ;
    r = -1 ;
    for( ; ; )
    {
        printf( " 1: Insert_front 2: Insert_rear\n" ) ;
        printf( " 3: Delete_front 4: Delete_rear\n" ) ;
        printf( " 5: Display 6: Exit\n" ) ;
        printf( " Enter the choice\n" ) ;
        scanf( " %d " , &choice ) ;
        switch( choice )
        {
            case 1:
                printf( " Enter the item to be inserted\n" ) ;
                scanf( "%d " , & item ) ;
                insert_front( item, q, &f, &r ) ;
                break ;
            case 2:
                printf( " Enter the item to be inserted\n" ) ;
                scanf( " %d " , & item ) ;
                insert_rear( item, q, &r ) ;

```

```

        break ;
    case 3:
        delete_front( q, &f, &r );
        break;
    case 4:
        delete_rear( q, &f, &r );
        break;
    case 5:
        display( q, f, r );
        break ;
    default :
        exit( 0 );
    }
}
}

```

ឧទាហរណ៍៨៖

ចូរសរសេរកម្មវិធីជាភាសា C ដើម្បីអនុវត្តនូវ Circular Queue ។

C program to implement circular queue

```

#include < stdio.h >
#include < process.h >
#define QUEUE_SIZE 5
/* function to check for overflow shown in example 1.*/
/* function to check for underflow shown in example 2. */
/* function to insert an item at the rear and shown in example 3. */
/* function to delete an item from the front end shown in example 4. */
/* function to display the contents of the queue shown in example 5 */
void main( )
{
    int choice, item, f, r, count, q[20];
    f = 0;
    r = -1;
    count = 0; /* queue is empty */
    for( ; )
    {
        printf( " 1: Insert 2: Delete\n " );
        printf( " 3: Display 4: Exit\n " );
        printf( " Enter the choice\n " );
        scanf( " %d " , &choice);
        switch( choice )
        {
            case 1:
                printf( " Enter the item to be inserted\n " );
                scanf( " %d " , &item );
                insert_rear( item, q, &r, &count );
                break;
            case 2:

```

```

delete_front( q, &f, &count );
break;
case 3:
display( q, f, count );
break;
default:
exit( 0 );
}
}
}

```

ឧទាហរណ៍៩៖

ចូរសរសេរកម្មវិធីជាភាសា C ដើម្បីអនុវត្តនូវ Circular Queue ជាមួយ Item ជាតួអក្សរ។

C program to implement circular queue with items as strings

```

#include < stdio.h >
#include < process.h >
#include < alloc.h >
#include < string.h >
#define QUEUE_SIZE 5

/* Function to check for underflow */(អនុគមន៍សម្រាប់ត្រួតពិនិត្យមើលQueue)
int qempty( int count )
{
    return( count == 0 )? 1 : 0;
}

/* Function to check for overflow */(អនុគមន៍សម្រាប់ត្រួតពិនិត្យ Queue ពេញ)
int qfull( int count )
{
    return( count == QUEUE_SIZE ) ? 1: 0;
}

/* Function to insert an item at the rear end */(អនុគមន៍សម្រាប់បន្ថែមធាតុ)
void insert_rear( char item[ ], char q[ ][30], int *r, int * count )
{
    if( qfull( *count ))
    {
        printf( " Overflow of queue\n " );
        return;
    }
    *r = ( * r + 1 )% QUEUE_SIZE;
    strcpy( q[ *r], item );
    *count += 1;
}

/* Function to delete an item from the front end */(អនុគមន៍សម្រាប់លុបធាតុ)
void delete_front( char q[ ][30], int *f, int *count )
{

```

```

if(qempty( *count ) )
{
    printf( " Underflow of queue\n " );
    return;
}
printf( " The delete element is %s\n " , q[ *f ] );
*f = ( *f + 1 )% QUEUE_SIZE;
*count = count -1;
}

/* Function to display the contents of the queue */(អនុគមន៍សម្រាប់បង្ហាញធាតុ)
void display( char q[ ] [ 30 ], int f, int count )
{
    int i, j;
    if( qempty(count))
    {
        printf( " Q is empty\n " );
        return;
    }
    printf( " Contents of queue is \n " );
    i = f ;
    for( j = 1; j <= count ; j ++ )
    {
        printf( " %s\n " , q[ i ] );
        i = ( i + 1 )%QUEUE_SIZE;
    }
    printf( " \n " );
}

void main( )
{
    int choice, f, r, count ;
    char item[20],q[20][30];
    f = 0;
    r = -1;
    count = 0; /* queue is empty */(Queueទំនេរ)
    for( ; ; )
    {
        printf( " 1: Insert 2: Delete\n " );
        printf( " 3: Display 4: Exit\n " );
        printf( " Enter the choice\n " );
        scanf( " %d " , &choice );
        switch( choice )
        {
            case 1:
                printf( " Enter the item to be inserted\n " );
                scanf( " %s " , &item);
                insert_rear( item, q, &r, &count );
                break;

```

```

        case 2:
            delete_front(q, &f, &count );
            break;
        case 3:
            display(q, f, count );
            break;
        default:
            exit( 0 );
    }
}
}

```

កម្មវិធីជាភាសា C ដើម្បីអនុវត្តនូវ Stack និង Queue

Sample C programs to represent the Stack and Queue Implementation

ឧទាហរណ៍១៖

/* Stack Implementation using arrays to insert element in the Stack */

```

#include< stdio.h >
#include< conio.h >
#define Max 5
int Staff[Max], top = -1;
void display( )
{
    if( ( top == -1 || ( top == 0 ) )
        {
            printf( " \n Stack is full\n " );
        }
    else
    {
        printf( " \n Stack elements are \n " );
        for( int i = top - 1; i >= 0; i - - )
            printf( " %5d " ,Staff[ i ] );
    }
}

void push( ) (អនុវត្តន៍សម្រាប់បន្ថែមធាតុចូល)
{
    int ele;
    char ch;
    if( top == -1 )
        top = 0 ;
    do
    {
        if( top >= 5 )

```

```

        printf( "\n STACK IS FULL " );
        break;
    }
    else
    {
        clrscr( );
        printf( " \n ENTER THE ELEMENT TO BE INSERTED \n " );
        scanf( " %d" , &ele);
        Staff[ top + + ] = ele;
        display( );
    }
    printf( " \n DO U WANT 2 ADD MORE ELEMENTS:? \n );
    scanf( " %c " , &ch );
} while( ( ch == 'y' ) || ( ch == 'Y' ));
}

```

void pop() (អនុគមន៍សម្រាប់ដកយកធាតុចេញ)

```

{
    if( ( top == -1 ) || ( top == 0 )
    {
        printf( " \n stack is underflow \n " );
    }
    else
    {
        printf( " \n %d is delete from stack \n " , Staff[ -- top ] );
        display( );
    }
}

```

ឧទាហរណ៍២៖

កម្មវិធីជាភាសា C ដើម្បីអនុវត្តនូវ Stack ដោយប្រើ push និង pop ។

/* Stack Implementation using push and pop */

```
# include< stdio.h >
```

```
void main( )
```

```

{
    clrscr( );
    char c;
    int choice;
    do
    {
        clrscr( );
        printf(" Enter the choice\n" );
        printf("1->push\n");
        printf("2->pop\n");
        scanf( " %d ",&choice);
        if( choice == 1 )

```

```

        push( );
    else if( choice == 2 )
        pop( );
    else
        printf( " \n invalid choice" );
    printf( " \n Do you want to continue:? ");
    scanf(" %c",&c);
}while((c == 'y')||(c == 'Y' ))
}

```

ឧទាហរណ៍៣៖

កម្មវិធីជាភាសា C ដើម្បីអនុវត្តនូវ Stack ដោយប្រើ Array ។

```

/* Stack Implementation using Array */
#include< stdio.h >
#include< conio.h >
#define Max 5
int a[ Max ], top = -1;
void push( )
{
    int ele;
    char ch;
    if( top == -1 )
        top = 0;
    do
    {
        if( top >= 5 )
        {
            printf( " Stack if full " );
            break;
        }
        else
        {
            clrscr( );
            printf( " Enter the elements to be inserted\n");
            scanf(" %d",&ele);
            a[top+ ] = ele;
        }
        printf(" Do you want to add more elements:?\n");
        scanf(" %c" ,&ch);
        printf(" %c",ch);
    }while( ( ch == 'Y' )||( ch == 'y' );
}
void pop( )
{
    if( top == -1)
    {
        printf(" Stack is underflow\n" );
    }
}

```

```

        else
        {
            for( int i = top - 1; i >= 0; i - - )
            }
    }
void main( )
{
    clrscr( );
    char c;
    int choice;
    do
    {
        clrscr( );
        printf( " Enter your choice\n" );
        printf( " 1->push");
        printf(" 2->pop\n");
        scanf("%d", &choice);
        if( choice == 1 )
            push( );
        else if( choice == 2 )
            pop( );
        else
            printf( " \n invalid choice " );
        printf( " Do you want to continue\n ");
        scanf( "%c",&c );
    }while( ( c == 'y' )||(c == 'Y' );
}

```

ឧទាហរណ៍៤៖

កម្មវិធីជាភាសា C ដើម្បីអនុវត្តនូវ Queue ដោយប្រើ Array ។

/* Queue Implementation using Array */

```

#include< stdio.h >
#include< conio.h >
#define Max 5
int Queue[Max], front = -1, rear = 0;
void Display( )
{
    int i;
    if( front + 1 == rear )
        printf(" Empty Queue\n ");
    else
    {
        printf(" Elements of Queue\n ");
        for( i = front + 1; i< rear; i + + )
            printf( " %d\t ", Queue[ i ] );
        printf( " \n " );
    }
}

```


void Insert_element() (អនុគមន៍សម្រាប់បន្ថែមធាតុ)

```
{
    int ele;
    if( rear == Max )
        printf( " Queue is Full " );
    else
    {
        printf( " Enter Element to be inserted\n" );
        scanf( " %d",&ele );
        Queue[ rear + + ] = ele ;
        Display( );
    }
}
```

void Delete_element() (អនុគមន៍សម្រាប់លុបធាតុ)

```
{
    if( front + 1 == rear )
        printf(" Empty Queue\n " );
    else
    {
        printf(" %d is Deleted from Queue\n " , Queue[+ +front]);
        Display( );
    }
}
```

void main()

```
{
    clrscr( );
    char c;
    int choice;
    do
    {
        printf( " Select your choice\n" );
        printf(" 1->Insert\n " );
        printf( " 2->Delete\n" );
        scanf( "%d" , &choice );
        if( choice == 1 )
            Insert_element( );
        else if( choice == 2 )
            Delete_element( );
        else
            printf( " Invalid choice" );
        printf( " \n Do you want to continue\n " );
        scanf( "%c",&c);
    }while((c == 'y')||(c == 'Y' ));
}
```

ឧទាហរណ៍៥៖

កម្មវិធីជាភាសា C ដើម្បីអនុវត្តនូវ Stack ដោយប្រើ Pointer។

/* Stack Implementation using Pointers */

```
#include< stdio.h >
#include< conio.h >
#include< alloc.h >
#define NewNode (Node * )malloc(sizeof(Node))
typedef struct node
{
    int item;
    struct node *next;
}Node;
Node *Push(Node *);
Node *Pop(Node *);
void Display(Node *);
void main( )
{
    int ch;
    Node *start = NULL;
    clrscr( );
    do
    {
        printf( " Select choice\n " );
        scanf( " %d " ,&ch );
        switch(ch)
        {
            case 1:
                start = Push(start);
                break;
            case 2:
                start = Pop(start);
                break;
            default:
                printf( " Invalid Choice\n " );
                break;
        }
        Display(start);
    }while(ch!= 0);
}
Node *Push(Node *s)
{
    Node *tmp = NULL;
    tmp = NewNode;
    int item;
    tmp->next = NULL;
    printf( " Enter the Item\n " );
```

```

scanf( " %d ",&tmp->item);
if( s == NULL)
    s = tmp;
else
{
    tmp->next = s;
    s = tmp;
}
return( s );
}
Node * Pop( Node *s )
{
    Node *tmp;
    if( s != NULL )
    {
        printf( " Element Deleted: %d\n ", s->item);
        tmp = s;
        s = s->next;
        free( tmp );
    }
    else
        printf( " Stack Underflow\n " );
    return( s );
}
void Display(Node *s)
{
    Node *tmp;
    tmp = s;
    if( tmp!= NULL )
    {
        while(tmp!= NULL )
        {
            printf(" %d",tmp->item );
            tmp = tmp -> next;
        }
        printf(" Null\n");
    }
    else
        printf(" Stack Underflow\n");
}

```

ឧទាហរណ៍៦៖

កម្មវិធីជាភាសា C ដើម្បីអនុវត្តនូវ Queue ដោយប្រើ Pointer។

```

/* Queue Implementation using Pointers */
# include< stdio.h >
# include< conio.h >
# include< alloc.h >
# define NewNode(Node *)malloc(sizeof(Node))

```

```
typedef struct node
{
    int item;

    node *next;
}Node;

Node *Insert( Node * f, Node *r );
Node *Delete( Node *f, Node *r );
void Display( Node *f, Node *r );
void main( )
{
    Node *front = NULL, *rear = NULL;
    int ch, flag = 0;
    do
    {
        printf(" Enter Your Choice\n" );
        scanf("%d",&ch);
        switch(ch)
        {
            case 1:
            {
                rear = Insert(front, rear);
                if(flag == 0)
                {
                    front = rear;
                    flag = 1;
                }
                break;
            }
            case 2:
            {
                front = Delete(front,rear);
                if( front == NULL)
                {
                    rear = NULL;
                    flag = 0;
                }
                break;
            }
            default:
            {
                printf(" Invalid Choice\n ");
                break;
            }
        }
    }while(ch!= 0);
}
```

```

Node *Insert(Node *f, Node *r)
{
    Node *tmp;
    tmp = NewNode;
    scanf("%d",&tmp -> item);
    tmp -> next = NULL;
    if(r == NULL)
    {
        r = tmp;
        f = tmp;
    }
    else
    {
        tmp -> next = r;
        r = tmp;
        Display(r,f);
    }
    return( r );
}

Node *Delete( Node *f , Node *r )
{
    Node *ptr;
    if(f == NULL)
        printf("Empty Queue\n");
    else if(f == r)
    {
        printf("Element Deleted: %d\n",f -> item);
        free( r );
        f = r = NULL;
    }
    else
    {
        ptr = r;
        while(ptr -> next -> next != NULL)
            ptr = ptr -> next;
    }
}

```

```
ptr -> next = NULL;
printf("Element Deleted: %d\n",f -> item);
free( f );
f = ptr;
}
Display( r , f );
return( f );
}
void Display( Node *r , Node *f )
{
    Node * ptr = r;
    if( ptr == NULL )
        printf("Queue Empty\n");
    else
    {
        while( ptr != f )
        {
            printf( " %d -> " , ptr -> item );
            ptr = ptr -> next;
        }
        printf(" %d -> " ,ptr -> item);
        printf(" \n ");
    }
}
```

លំហាត់អនុវត្ត

Array

១-គេមាន Array ពីរវិមាត្រ float a[4][3];

ក-ចូរសរសេរពីរបៀបផ្ទុកនៃធាតុរបស់ Array ខាងលើនេះតាមជួរដេក។

ខ-ចូរគណនា Address នៃធាតុ a[3][1] តាមជួរដេក។

(បើគេដឹងថា Address ដើមស្មើ 490)

២-គេមាន Array ពីរវិមាត្រ float a[3][5];

ក-ចូរសរសេរពីរបៀបផ្ទុកនៃធាតុរបស់ Array ខាងលើនេះតាមជួរដេក។

ខ-ចូរគណនា Address នៃធាតុ a[2][3] តាមជួរដេក។

(បើគេដឹងថា Address ដើមស្មើ 118)

៣-គេមាន Array ពីរវិមាត្រ float a[6][3];

ក-ចូរសរសេរពីរបៀបផ្ទុកនៃធាតុរបស់ Array ខាងលើនេះតាមជួរឈរ។

ខ-ចូរគណនា Address នៃធាតុ a[4][2] តាមជួរឈរ។

(បើគេដឹងថា Address ដើមស្មើ 88)

៤-គេមាន Array ពីរវិមាត្រ float a[3][6];

ក-ចូរសរសេរពីរបៀបផ្ទុកនៃធាតុរបស់ Array ខាងលើនេះតាមជួរដេក។

ខ-ចូរគណនា Address នៃធាតុ a[1][4] តាមជួរដេក។

(បើគេដឹងថា Address ដើមស្មើ 96)

៥-គេមាន Array បីវិមាត្រ: float a[2][3][3]; បើគេដឹងថា Address ដំបូងស្មើ 120។

ក-ចូរសរសេរពីរបៀបផ្ទុកនៃធាតុនីមួយៗតាមជួរដេក និងជួរឈរ។

ខ- ចូរគណនា Address នៃធាតុ a[1][2][1] តាមជួរដេក។

Stack and Queue

១-គេសន្មតថា st ជា stack មួយផ្ទុកចំនួនគត់ហើយ i និង j ជាអញ្ញាតដែលមានប្រភេទទិន្នន័យដូចជាធាតុរបស់ Stack ដែរ។ ចូរគូររូបបញ្ជាក់ពីដំណើរការចំពោះការធ្វើប្រមាណវិធីជាបន្តបន្ទាប់ដូចខាងក្រោម៖

- (a)- clear stack (st)
- (b)- push (st , 15)
- (c)- i = pop (st)
- (d)- push (st , i)
- (e)- push (st , i)

(f)- $j = \text{pop}(st)$

២-សន្មតថា x, y ជាអញ្ញាតពីរដែលមានប្រភេទទិន្នន័យដូចគ្នាគ្នាបស់ Stack st ដែរ។

ប្រសិនបើ x និង y ត្រូវបានគេកំណត់តម្លៃឲ្យរួចជាស្រេចហើយនោះ តើវាមានការប្រែប្រួលយ៉ាងណាទៅពេលដែលគេអនុវត្តប្រមាណវិធីជាបន្តបន្ទាប់ដូចខាងក្រោម៖

- $\text{push}(st, x)$
- $\text{push}(st, y)$
- $x = \text{pop}(st)$
- $y = \text{pop}(st)$

៣- គេអោយ Stack st ផ្ទុកចំនួនគត់ហើយ x, y, z ដែលមានប្រភេទទិន្នន័យដូចគ្នាគ្នាបស់ Stack ដែរ។ ចូរបង្ហាញពី Output ដូចខាងក្រោម៖

- $\text{clear stack}(st)$
- $x = 1 ; y = 0 ; z = 4$
- $\text{push}(st, y) ;$
- $\text{push}(st, x) ;$
- $\text{push}(st, x + y) ;$
- $\text{print f}(" x = \%d , y = \%d , Z = \%d \n" , x , y , z) ;$
- $\text{while}(! \text{empty stack}(st))$
 - {
 - $x = \text{pop}(st) ;$
 - }
- $y = \text{pop}(st) ;$
- $\text{push}(st, y) ;$
- $\text{push}(st, 3) ;$
- $x = \text{pop}(st) ;$

៤-ឃ្លាំងជួសជុលរថភ្លើងមួយមានលំនាំដូច Stack។ ឧបមាថានៅផ្លូវចូលមានក្បាលរថភ្លើង ៤ ត្រូវបានដាក់តាមលេខលំដាប់ 1 2 3 4។ គេមានវិធានក្នុងការនាំក្បាលរថភ្លើងចូលឃ្លាំងតាងដោយ I (Input) និងវិធាននាំក្បាលរថភ្លើងចេញពីឃ្លាំងតាងដោយ O (output)។ បើគេអនុវត្ត៖ I IOI IOO នោះក្បាលរថភ្លើងចេញពីឃ្លាំងមានលំនាំ 2 4 3 1។ ប្រសិនបើមានក្បាលរថភ្លើង ៦ ត្រូវបានដាក់តាមលេខលំដាប់ 1 2 3 4 5 6 នោះ

ក-ដើម្បីអោយរថភ្លើងចេញពីឃ្លាំងមានលំនាំ 5 4 6 3 2 1 តើត្រូវអនុវត្តវិធាន I និង O យ៉ាងដូចម្តេច?

ខ-បើគេអនុវត្ត៖ I I I I O I O O I O O O តើក្បាលរថភ្លើងចេញពីឃ្លាំងមានលំនាំយ៉ាងដូចម្តេច?

៥-Define Stack with neat diagram? Discuss the Push and Pop Operation.

៦-ចូរសរសេរកម្មវិធីដើម្បីកំណត់ប្រមាណវិធីរបស់ Stack ដោយប្រើ Arrays ។

៧-Define Queue with neat diagram?

៨-ចូរពន្យល់អំពីប្រភេទនៃ Queues។

៩-Discuss the Insert/Delete operation from the rear and front end of the queue.

១០-ឧបមាថាគេមានប្រមាណវិធី(អនុគមន៍)ដែលអនុវត្តន៍លើ Stack ដែលត្រូវបានបង្កើតរួចហើយ។ ចូរសរសេរដោយប្រើអនុគមន៍ទាំងនោះដើម្បីផ្លាស់ប្តូរតម្លៃនៃពីរចំនួនគត់។

១១-

ក-ចូរសរសេរអនុគមន៍ដើម្បីដកយកធាតុនីមួយៗចេញពី Queue តាមគោលការណ៍ដំណើរការទី៣។

ខ-តើ Stack និង Queue មានលក្ខណៈខុសគ្នាយ៉ាងណាខ្លះ?



មេរៀនទី៥

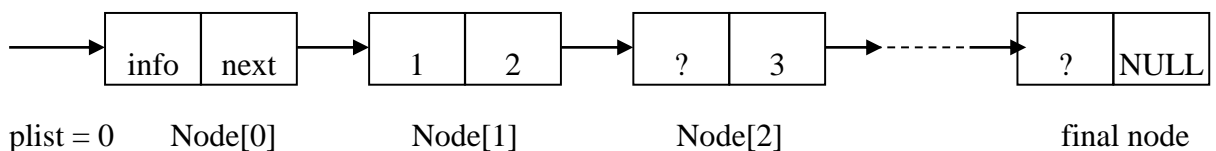
Linked List

5-1: ស្ថាប័នក្រុម

មេរៀននេះណែនាំឲ្យយើងស្វែងយល់អំពីបញ្ហាជាមូលដ្ឋាន ក្នុងការបង្កើត Linked List ដូចជា Structure Linked List និងប្រមាណវិធីមួយចំនួនរបស់ Structure ទាំងនោះ ដើម្បីជួយឲ្យយើងមានលទ្ធភាពបង្កើត List ដោយប្រើលក្ខណៈ Linked List ។

ដូចនេះ Linked List គឺជាសំណុំធាតុ (Node) ជាច្រើនត្រូវបានភ្ជាប់គ្នាអាស្រ័យ ដោយតំបន់ភ្ជាប់របស់វា (field next) ហើយរាល់ Node នីមួយៗរួមមានពីរ field គឺ field info សម្រាប់ផ្ទុកទិន្នន័យនិង field next សម្រាប់ផ្ទុក Pointer ចង្អុលទៅកាន់ Node បន្ទាប់។ លំដាប់ផ្ទុករបស់ Node នីមួយៗមានលំដាប់ Node ដំបូង (Node[0]) មាន field next ចង្អុលទៅ កាន់ Node[1],..., Node ចង្អុលចុងក្រោយមាន field next ស្មើ Null និងមានអថេរ plist ចង្អុល ទៅកាន់ Node ដំបូង (Node[0]) ។

គេមាន Structure ទូទៅរបស់ Linked List ដែលមាន Node ដូចខាងក្រោម៖



គេអាចបង្កើត Linked List តាមពីរបៀបគឺ បង្កើត Linked List ដោយ Array និង បង្កើត Linked List ដោយ Dynamic ។

១-ការបង្កើត Linked List ដោយប្រើ Array

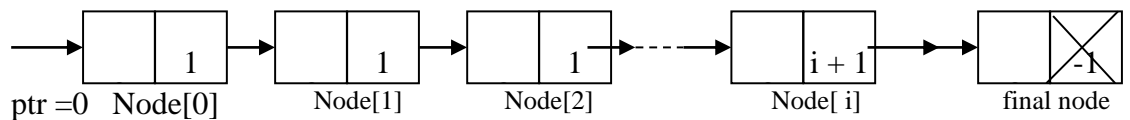
ដើម្បីបង្កើត Linked List ដោយប្រើ Array ជាដំបូងគេត្រូវបង្កើត Available Node (គឺ បង្កើតសំនុំ Node កំណត់ទុកជាមុនហើយរាល់ Node នីមួយៗរបស់វាត្រូវបានប្រើសម្រាប់ បញ្ជូនទៅឲ្យ Linked List)។ Available Node ដែលបានបង្កើតដំបូងមានលំដាប់ផ្ទុក Node នីមួយៗរបស់វាតាមលំដាប់ធម្មជាតិ Node[0], Node[1], Node[2],..., Node[i] ,...,Node[n-1] (ជាមួយ index $i = 0, 1, 2, \dots, n-1$) និងមានអថេរ ptr ចង្អុលទៅកាន់ Node ដំបូង (Node[0]) ហើយរាល់ Node នីមួយៗរបស់វារួមមាន 2 fields គឺ field info សម្រាប់ផ្ទុកទិន្នន័យ និង

field next សម្រាប់ផ្ទុកទីតាំង Index របស់ Node បន្ទាប់នៃ Array។ គេមានទម្រង់ទូទៅរបស់ Available Node ដែលមាន n ធាតុដូចខាងក្រោម៖

Node	pts = 0	1	2	3	...	i	...	n-1
info	?	?	?	?		?	...	?
Next	1	2	3	4		i+1	...	-1

ptr = 0 និង Node[i].next = i + 1

គេអាចគូសទម្រង់ Linked List របស់ Available Node នេះដូចខាងក្រោម៖



គេមានរបៀបបង្កើត Data Structure របស់ Linked List ដោយ Array គឺរាល់ Node នីមួយៗរបស់វាជាប្រភេទ struct ដែលមានពីរ field (info ផ្ទុក data, next ផ្ទុកទីតាំង index បន្ទាប់) ត្រូវបានសរសេរតាម C Language ដូចខាងក្រោម៖

```
# define MaxArray
struct Node type
{
    data type info;
    int next ;
}
Node[MaxArray];
```

ឧទាហរណ៍៖

(ក)-ចូរបង្កើត Data Structure របស់ Linked List ដោយ Array ដែលមានទំហំស្មើ 20 និងរាល់ Node នីមួយៗផ្ទុកទិន្នន័យជាតួអក្សរ។

(ខ)-ចូរបង្កើត Data Structure របស់ Linked List ដោយ Array ដែលមានទំហំស្មើ 100 និងរាល់ធាតុនីមួយៗរបស់វាជាព័ត៌មាននិស្សិត (ID, Name , Date: (mm,dd,yy) , sex , Average ។

គេបាន

```
(ក)- # define MaxArray 20
structure Nodetype
{
    char info ;
    int next;
} Node[MaxArray] ;
```

```
(2)- # define MaxArray 100
struct Date
{
    int mm ;
    int dd ;
    int yy ;
};
struct student
{
    int ID ;
    char * Name ;
    struct Date Age ;
    char Sex ;
    float Average ;
};
struct Node type
{
    struct student info ;
    int next ;
}
Node [MaxArray] ;
```

គេមាន Algorithms មួយចំនួនរបស់ Linked List ដោយ Array ដូចខាងក្រោម៖

1- Algorithms: Available Node

ដើម្បីបង្កើតសំណុំ Node កំណត់ទុកជាមុន ហើយរាល់ Node នីមួយៗ របស់វាត្រូវបានផ្ទុកតាមលំដាប់ធម្មជាតិគឺ Node[0] , Node[1] , Node[2] ,..., Node[n-1] និង មានអថេរ ptr ចង្អុលទៅកាត់ Node ដំបូង Node[0] និង Node[i].next = i + 1 ដែលត្រូវបាន សំដែងដោយភាសាកម្មវិធី (C Language) ដូចខាងក្រោម៖

```
void Available_Node ( )
{
    ptr = 0 ;
    for ( int i = 0 ; i < MaxArray -1; i ++ )
        Node [ i ] . next = i +1 ;
    Node[ MaxArray - 1 ] = -1;
}
```

2- Algorithms: Get node

ដើម្បីបញ្ជូន Node ដែលកំពុងចង្អុលដោយ ptr ពី Available Node ទៅដោយ Linked List និងឲ្យអថេរ ptr ចង្អុលទៅកាន់ Node បន្ទាប់របស់វា។

```
int Getnode( )
{
    int p ;
    p = ptr ;
    ptr = Node[ ptr ] . next ;
    return(p) ;
}
```

3- Algorithms: Freenode

សម្រាប់បញ្ចូល Node ដែលបានលុបចេញពី Linked List ឲ្យទៅ Available Node ត្រង់ទីតាំងមុនបន្ទាប់ពី Node កំពុងចង្អុលដោយអថេរ ptr និងឲ្យអថេរ ptr ចង្អុលទៅកាន់ Node នោះ។

```
void FreeNode(int p)
{
    Node[ p] . next = ptr ;
    ptr = p ;
}
```

ឧទាហរណ៍៖

(ក)-ចូរគូស Structure Available Node ដែលមាន MaxArray = 10។

(ខ)-ប្រសិនគេបញ្ចូល Node ជាបន្តបន្ទាប់ពី Available Node នោះទៅកាន់ Linked List ដើម្បីផ្គុំកទិន្នន័យ: M , B , A , D , E។

• ចូរគូសទម្រង់ Linked List របស់ List ដែលមានអថេរ plist ចង្អុលទៅកាន់ Node ដំបូងរបស់វា។

• ចូរគូសទម្រង់ Linked List របស់ Available Node។

(គ)-ប្រសិនគេលុប Node មានទិន្នន័យ D និង B ចេញពី List

• ចូរគូសទម្រង់ Linked List របស់ Available Node។

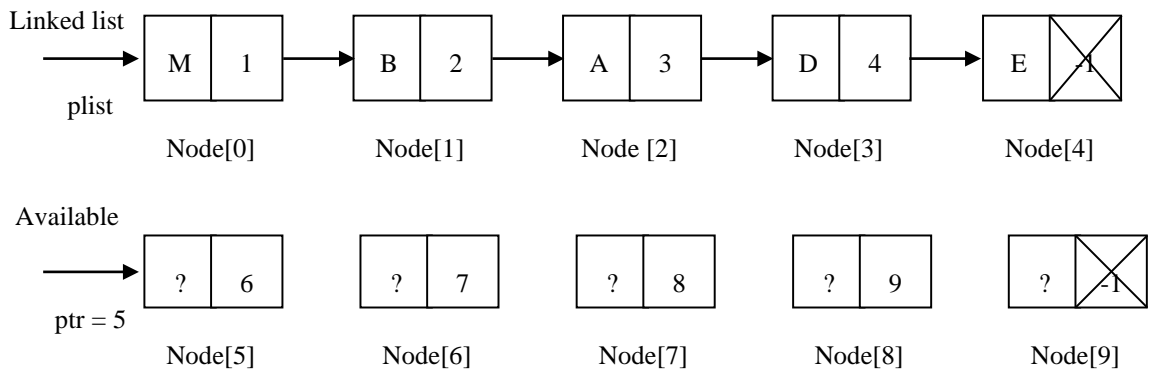
• ចូរគូសទម្រង់ Linked List របស់ List។

គេបាន៖

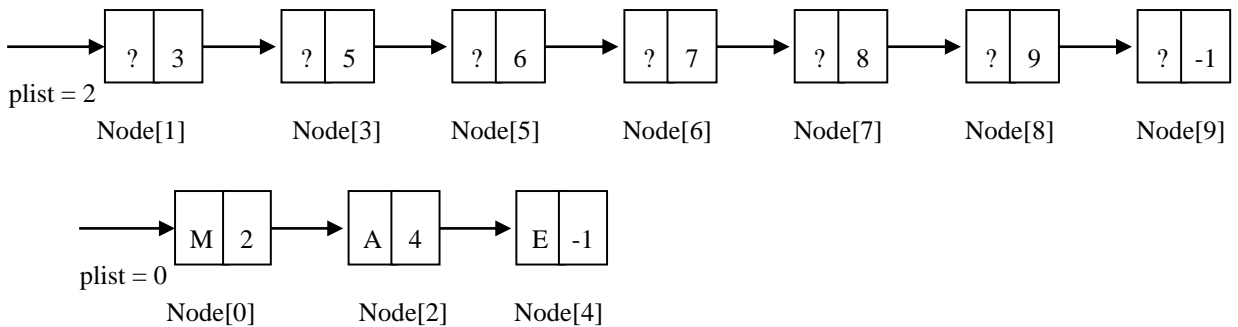
(ក)-Structure Available Node ដែលមាន MaxArray ស្មើ 10

Node	ptr=0	1	2	3	4	5	6	7	8	9
Info	?	?	?	?	?	?	?	?	?	?
Next	1	2	3	4	5	6	7	8	9	-1

(ខ)-ទម្រង់ Linked List របស់ List ដែលមានអថេរ plist ចង្អុលទៅកាន់ Node ដំបូងរបស់វា



(គ)-ភ្ជាប់ទម្រង់ Linked List របស់ Available Node



២-ការបង្កើត Linked List ដោយប្រើ Dynamic

ដើម្បីបង្កើត Linked List ដោយប្រើ Dynamic ជាដំបូងយើងត្រូវស្វែងយល់អំពីបញ្ហា Pointer ។

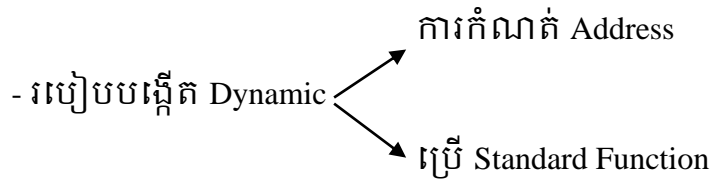
-Pointer គឺជាប្រភេទអថេរម្យ៉ាងដែលផ្ទុក Address សម្រាប់ Point ទៅកាន់អថេរដទៃ មិនផ្ទុកទិន្នន័យ។

ឧទាហរណ៍៖

```
int * px ;
float * py ;
```

គេបាន *px , *py ជា pointer ។

-Dynamic គឺជាប្រភេទអថេរដែលចាប់យក space នៃ memory ដើម្បីផ្ទុកទិន្នន័យរបស់អថេរ Pointer ។



ឧទាហរណ៍២៖

```

int x = 45 ;
px = &x ;
* px = 90 ;
py = (float *) malloc(sizeof( float));
* py = 23.46 ;
  
```

មានន័យថាអថេរ px និង py ជា Dynamic ។

គេមានរបៀបបង្កើត Data Structure របស់ Linked List ដោយ Dynamic គឺ

```

struct Nodetype
{
    datatype info ;
    struct Nodetype * next ;
    struct Nodetype *next ;
} Node ;
type def struct Nodetype *pointertype;
  
```

គេមាន Algorithms របស់ Linked List ដោយ Dynamic គឺ

* Algorithms Getnode:

សម្រាប់បញ្ជូន Node ពី Location ទំនេររបស់ memory អោយ Linked List ។

```

pointertype Getnode( )
{
    struct Nodetype *p ; /* pointertype p ; */
    p = (struct Nodetype *) malloc (sizeof ( struct Nodetype )) ;
    reture (p) ;
}
  
```

***Algorithms Free node**

សម្រាប់បញ្ជូន Node ដែលបានលុបចេញពី Linked List ទៅអោយតំបន់ផ្ទុកទីតាំងទំនេរ។

```
void Freenode( pointertype p )
{
    Free ( p ) ;
}
```

គេមានប្រមាណវិធីមួយចំនួនរបស់ Linked List ដែលរួមមានដូចខាងក្រោម៖

- Singly Linked List និង Circular Singly Linked List
- Doubly Linked List និង Circular Doubly Linked List

ត្រូវបានសរសេរតាម Pseudo Code ដូចខាងក្រោម៖

*** គេកំណត់**

- plist គឺជា Pointer ចង្អុលទៅកាន់ Node ដំបូងរបស់ Linked List។
- p គឺជា Pointer ចង្អុលទៅកាន់ Node ណាមួយរបស់ Linked List។
- info(p) គឺសម្រាប់ផ្ទុក Data រីបង្ហាញចេញ Data របស់ Node កំពុងចង្អុលដោយ

p (Node p)។

- next(p) គឺសម្រាប់ផ្ទុក Address របស់ Node បន្ទាប់ពី Node[p]។
- info(next(p))
- Getnode()
- Freenode()

*** Algorithms**

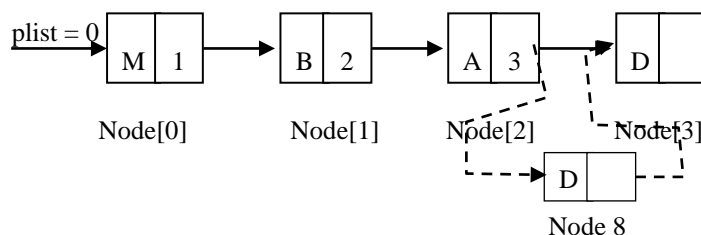
• Initalize

សម្រាប់បង្កើត Linked List ទំនេរគឺ

plist = Null

• Insert first

សម្រាប់បន្ថែម Node[p] ដែលមានទិន្នន័យ Item ចូលទីតាំងដំបូងរបស់ Linked List។



• **Insert after**

សម្រាប់បន្ថែម Node[p] ដែលមានទិន្នន័យ Item ចូលទីតាំងបន្ទាប់របស់ Linked List។

ដំណើរការ

```
p = Getnode( ) ;
info(q) = Item ;
next(q) = next(p);
next(q) = q ;
```

• **Traverse**

សម្រាប់បង្ហាញទិន្នន័យរបស់ Node នីមួយៗពី Node ដំបូងដល់ Node ចុងក្រោយ។

ដំណើរការ

```
p = plist;
while ( p != NULL)
{
    info (q) ;
    p = next ( p ) ;
}
```

• **Search**

សម្រាប់ស្វែងរកធាតុ Item ណាមួយនៅក្នុង List ដែលមានទិន្នន័យពុំមានតម្លៃ

សូន្យ។

ដំណើរការ

```
p = plist;
while( p != NULL)
{
    if( Item == info(p)) /* ផ្តល់ទីតាំង Node[p] និងបញ្ឈប់ការងារស្វែងរក */
        else
            p = next( p ) ;
} /* ប្រសិន p = NULL នោះផ្តល់តម្លៃទទេ */
```

• **Sort**

សម្រាប់តម្រៀបទិន្នន័យតាមលំដាប់កើនជាបន្តបន្ទាប់។

ដំណើរការ

```
for( p = plist ; next (p) != NULL ; p = next ( p ) )
    for(q = next ( p ) ; q!= NULL ; q = next ( q ))
        if (info(p) > info ( q ))
            ប្តូរ info( p ) និង info(q )
```

• **Clear list**

សម្រាប់លុប Node ទាំងអស់ចេញពី List។

ដំណើរការ

```
p = plist ;
while(p != NULL)
{
    Free node( p ) ;
    p = next( p ) ;
}
```

5-2: Singly Linked List

5-2-1: និយមន័យ

Singly Linked List គឺជាសំណុំ Node ជាច្រើនត្រូវបានតភ្ជាប់គ្នាហើយរាល់ Node នីមួយៗរបស់វាមានពីរ field គឺ field info សម្រាប់ផ្ទុកទិន្នន័យ និង field next សម្រាប់ផ្ទុក Pointer ចង្អុលទៅកាន់ Node បន្ទាប់ដោយមានអថេរ plist ចង្អុលទៅកាន់ Node ដំបូង (Node[0]) ហើយ Node[0] មាន field next ចង្អុលទៅកាន់ Node[1] , ..., Node ចុងក្រោយមាន field next ស្មើ NULL។

5-2-2: ប្រមាណវិធីលើ Singly Linked List

(ក)- របៀបបង្កើត Data Structure ដោយប្រើ Dynamic

```
struct Nodetype
{
    datatype info ;
    struct Nodetype * next ;
} * Node ;
typedef struct Nodetype * pointer type ;
```

(ខ)-បណ្តា Algorithms

គេមាន Algorithms មួយចំនួនរបស់ Singly Linked List ត្រូវបានសរសេរតាម ភាសា C ដូចខាងក្រោម៖

1-Algorithms

សម្រាប់បង្កើត Linked List ទំនេរ។

```
void initialize( pointertype * plist)
{
    * plist = NULL;
}
```

2-Algorithms

សម្រាប់បន្ថែម Node[p] ដែលមានទិន្នន័យ Item ចូលទីតាំងដំបូងរបស់ Linked List។

```
void Insert_first (pointertype * plist, datatype Item)
{
    piontertype p = Getnode( ) ;
    p → info = Item ;
    p → next = * plist;
    * plist = p ;
}
```

3-Algorithms

សម្រាប់បន្ថែម Node[q] ដែលមានទិន្នន័យ Item ចូលទីតាំងបន្ទាប់របស់ Linked List។

```
void Insert_after (pointertype p, datatype Item)
{
    pointertype q = Getnode( ) ;
    q → info = Item ;
    q → next = p → next ;
    p → next = q ;
}
```

4-Algorithms

សម្រាប់លុប Node[p] ដែលមានទិន្នន័យ Item ចេញពីទីតាំងដំបូងរបស់ Linked List ។

```
datatype Delete_first( pointertype * plist )
{
    pointertype p = * plist;
    datatype Element = p → info;
    plist = p → next;
    Free (P) ;
    return(Element);
}
```

5-Algorithms

សម្រាប់បង្ហាញទិន្នន័យរបស់ Node នីមួយៗដោយចាប់ផ្តើមពី Node ដំបូងរហូតដល់ Node ចុងក្រោយ។

```
void Traverse( pointertype * plist )
{
    pointertype p = *plist ;
    while ( p != NULL)
    {
        print f("%...", p → info) ;
        p = p → next ;
    }
}
```

6-Algorithms

សម្រាប់ស្វែងរកធាតុ Item ណាមួយនៅក្នុង List ដែលមានទិន្នន័យពុំមានតម្លៃសូន្យ។

```
pointertype search( pointertype * plist; datatype Item )
{
    pointertype p = * plist ;
    while (p!= NULL)
    {
        if (p → info = Item)
            return ( p );
    }
}
```

```

        else
            p = p → next;
        }
    return (NULL);
}

```

7-Algorithms

សម្រាប់តម្រៀបទិន្នន័យតាមលំដាប់កើនជាបន្តបន្ទាប់។

```

void sort( pointertype *plist )
{
    pointertype p ;
    datatype Item ;
    for( p = * plist ; p → next != NULL ; p = p → next )
        for( q = p → next ; q != NULL ; q = q → next )
            if ( p → info > q → info )
            {
                Item = p → info ;
                p → info = q → info ;
                q → info = Item ;
            }
}

```

8- Algorithms

សម្រាប់លុប Node ទាំងអស់ចេញពី List។

```

void clear_list( pointertype *plist )
{
    pointertype p = * plist;
    while ( p != NULL )
    {
        Free ( p ) ;
        p = p → next ;
    }
}

```

ឧទាហរណ៍៖

Illustrate the Singly Linked List operations using Pointers

```

/* Singly Linked List using Pointers */
#include <stdio.h>
#include <conio.h>
#include <alloc.h>
#define NewNode( Node *)malloc(sizeof(Node))
typedef struct node
{
    int item;
    struct node next;
}Node;
Node *Create(Node);
void Display(Node *);
int Count(Node *);
Node *Insert(Node *);
Node *Delete(Node *);
void Search(Node *);
void main( )
{
    Node *ptr = NULL, *start = NULL;
    int ch, cnt;
    start = Create( start );
    Display(start);
    do
    {
        printf("\n Singly Linked List Operations \n");
        printf(" 1 ->      Count \n");
        printf(" 2 ->      Display \n");
        printf(" 3 ->      Insert \n");
        printf(" 4 ->      Delete \n");
        printf(" 5 ->      Search \n");
        printf(" \n Enter w. choice\n");
        scanf("%d",&ch);
        switch(ch);
        {
            case 1:
                printf("\nNo of Nodes = %d\n", Count(start));
                break;
            case 2:
                Display(start);
                break;
            case 3:
                start = Insert(start);
                break;
            case 4:
                start = Delete(start);

```

```

        break;
    case 5:
        Search(start);
        break;
    default:
        printf(" Invalid Selection\n ");
        break;
    }
}while(ch != 0);
}
Node *Create( Node *s )
{
    Node *tmp = NULL, *t1 = NULL;
    int nwn;
    t1 = s;
    do
    {
        printf("\n Enter the element \n");
        scanf(" %d",&num);
        if(num != -99)
        {
            tmp = New Node;
            tmp -> item = num;
            tmp -> next = NULL;
            if(s == NULL)
                s = t1 = tmp;
            else
            {
                t1 -> next = tmp;
                t1 = t1 -> next;
            }
        }
    }while(num != -99);
    return( s );
}
void Display( Node *s)
{
    if(s == NULL)
        printf("MT Linked List\n");
    else
    {
        while(s != NULL)
        {
            printf("%5d",s -> item);
            s = s -> next;
        }
    }
}

```

```

        printf("\n");
    }
}
int Count(Node *s)
{
    int total = 0;
    while(s != NULL)
    {
        total ++;
        s = s -> next;
    }
    return(total);
}
Node *Insert(Node * s)
{
    Node *t1 = NULL, *tmp = NewNode;
    int pos;
    printf("Enter the position to be inserted \n");
    scanf("%d",&pos);
    if(pos > 0 && pos <= Count( s ) + 1)
    {
        printf("Enter the element to be inserted\n");
        scanf("%d",&tmp -> item)
        if(pos == 1)
        {
            tmp -> next = s;
            s = tmp;
        }
        else
        {
            t1 = s;
            while(pos > 2)
            {
                t1 = t1 -> next;
                pos --;
            }
            tmp -> next = t1 -> next;
            t1 -> next = tmp;
        }
    }
    else
        printf("invalid position\n");
    return( s );
}
Node * Display(Node *s)
{
    Node *t1 = s, *tmp = NULL;
    int pos;

```



```

printf("Enter the position to be deleted\n");
scanf("%d",&pos);
if(pos>0 && pos <= Count ( s ))
{
    if(pos == 1)
    {
        s = s -> next;
        free(t1);
    }
    else
    {
        while(pos > 2)
        {
            t1 = t1 -> next;
            pos --;
        }
        tmp = t1 -> next;
        t1 -> next = tmp -> next;
        free(tmp);
    }
}
else
    printf("Invalid Position\n");
return( s );
}

void Search(Node *s)
{
    int ele;
    int flag = 0;
    printf("Enter the element to be searched\n");
    scanf("%d",&ele);
    if(s != NULL)
    {
        while(s != NULL)
        {
            if(s -> item == ele)
            {
                printf("\n %d is present\n",ele);
                flag = 1;
            }
            s = s -> next;
        }
        if(flag == 0)
            printf("Element Not Found \n");
    }
    else
        printf("List is MT, Key element can't be searched\n");
}

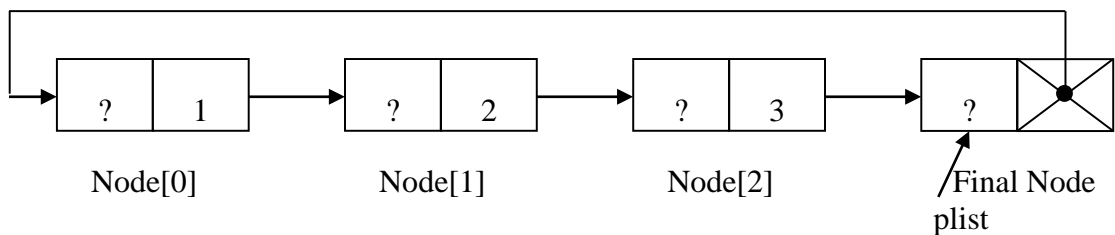
```

5-3 Circular Singly Linked List

5-3-1: និយមន័យ

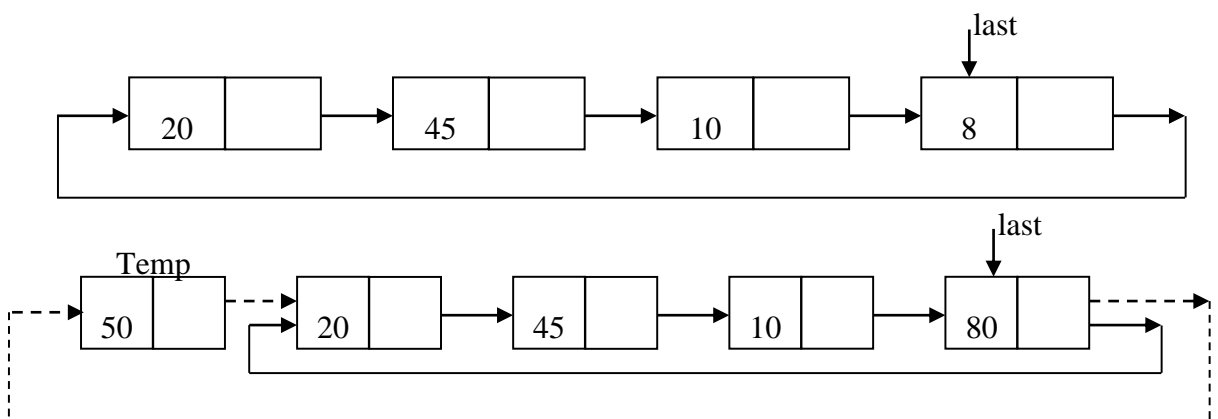
Circular Singly Linked List គឺជាសំណុំ Node ជាច្រើនត្រូវបានភ្ជាប់គ្នា ហើយរាល់ Node នីមួយៗរបស់វាមាន 2 fields គឺ field info សម្រាប់ផ្ទុកទិន្នន័យ និង field next សម្រាប់ផ្ទុក Pointer ចង្អុលទៅកាន់ Node បន្ទាប់ដោយមានអថេរ plist ចង្អុលទៅកាន់ចុងក្រោយ,..., Node ចុងក្រោយមាន field next ចង្អុលទៅកាន់ Node ដំបូង។

គេមាន Structure ទូទៅរបស់ Circular Singly Linked List គឺ



5-3-2: ប្រមាណវិធីលើ Circular Singly Linked List

* បន្ថែម item នៅខាងមុខនៃ List (Insert a node at the front end)



ឧទាហរណ៍៖

អនុគមន៍សម្រាប់បន្ថែម item នៅខាងមុខនៃ List។

NODE insert _ fron (int item, NODE last)

{

 NODE temp ;

 temp = getnode() ; /* Geate a new to be temp → insert */

 temp → info = item;

 if(last == NULL) /* Make temp as the first node */

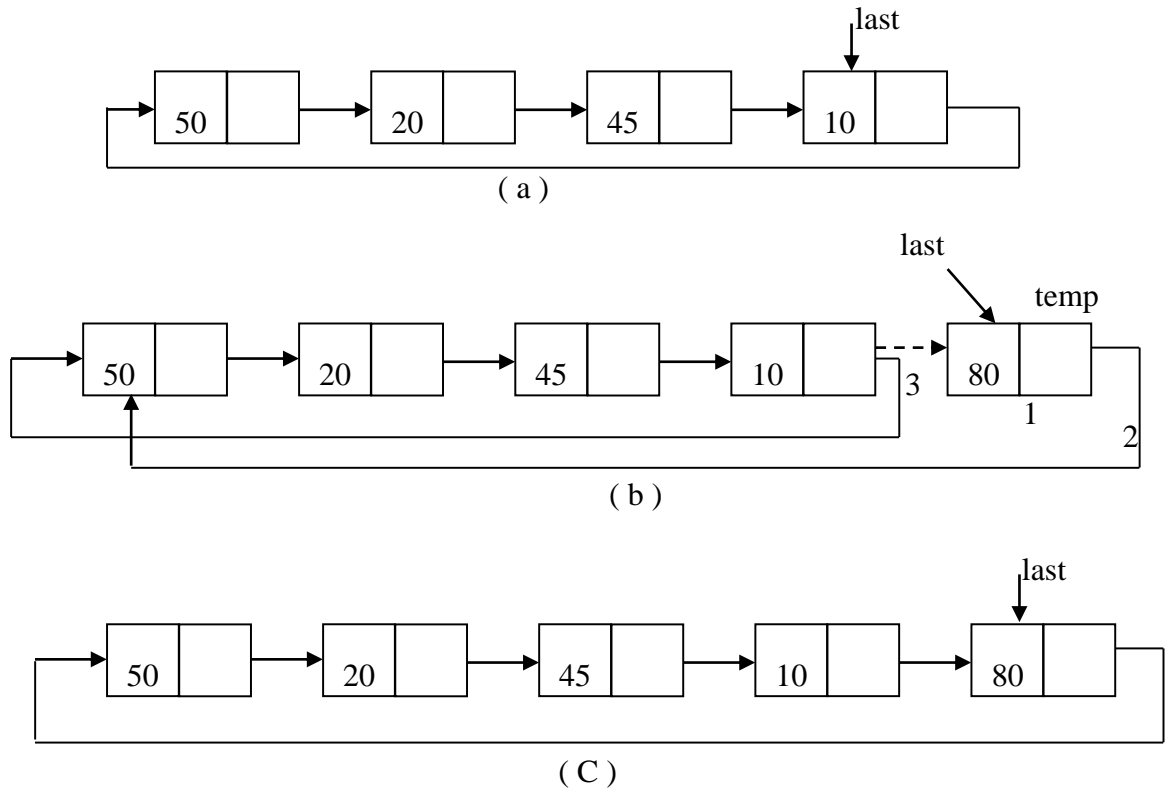
 last = temp ;

```

else
temp → link = last → link;
last → link = temp ;  /* link last node to first node */
return(last );        /* Return the last node */
}

```

- បន្ថែម item នៅខាងក្រោយនៃ **List(Insert a node at the rear end)**



(Figure to insert at the rear end)

ឧទាហរណ៍៖

អនុគមន៍សម្រាប់បន្ថែម item នៅខាងក្រោយនៃ **List**។

```

NODE insert_rear ( int item, NODE last )

```

```

{
    NODE temp ;
    temp = getnode ( ) ;          /* បង្កើត Node ថ្មីសម្រាប់បន្ថែម */
    temp → inf = Item;
    if( last == NULL)             /* Node ថ្មីក្លាយជា Node ដំបូង */

```

```

last = temp ;

else                                     /* បន្ថែម Node ថ្មីនៅទីតាំងចុងក្រោយ */

temp → link = last → link ;

Last → link = temp ;                  /* ភ្ជាប់ Node ចុងក្រោយទៅកាន់ Node ដំបូង */

return(temp) ;                         /* Make the new node as the last node */

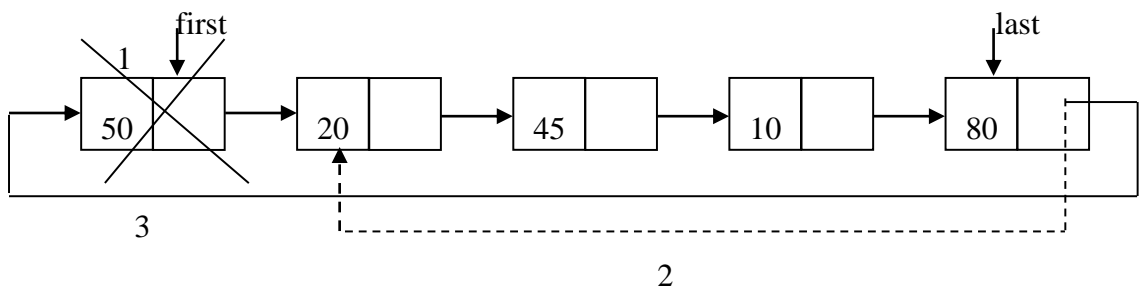
}

```

• **Note:**

Compare the function insert_front () and insert_rear (). All statement in both the functions are same except that in insert_front () function , address of last node is returned and in the function insert_rear () address of the new is returned.

* លុប item ចេញពីមុខ List (Delete a node from the front end)



(Figure to delete an item from the front end)

ឧទាហរណ៍៖

អនុគមន៍សម្រាប់លុប item ចេញពីមុខ List ។

```

1- first = last → Link ;                /* obtain address of the first node */
2- Last → Link = first → Link ;         /* Link the last node and new first node */
3- printf(" the item delete is % d \n ", first → info ) ;

Freenode ( first ) ;                    /* delete the old first node */

/* if there is only one node, delete if */

if( last → link == last )
{
    printf( " The item deleted is % d\n ", last → info ) ;
    freenode (last) ;
    last = NULL ;
    return (last) ;
}

```

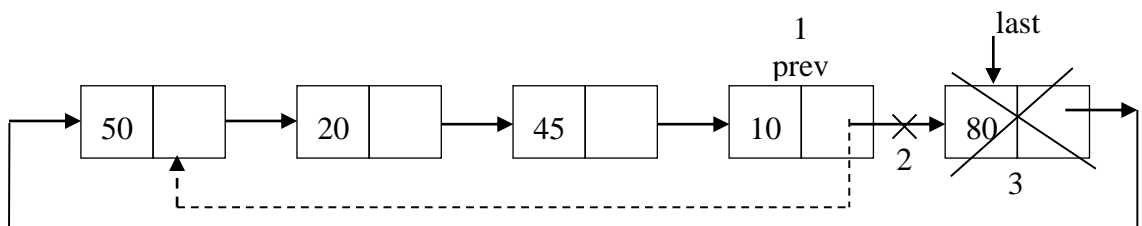
គេហទំព័រ

អនុគមន៍សម្រាប់លុប **item** ចេញពីមុខ List គឺ

```

NODE delete_front ( NODE Last )
{
    NODE temp , first ;
    if(last == NULL)
    {
        printf( "list is empty \n" ) ;
        return (NULL) ;
    }
    if(last → link == last)          /* Only node is present */
    {
        printf( "The item deleted is % d \n ", last → info ) ;
        freenode ( last ) ;
        return (NULL) ;
    }
    /* List contains more than one node */
    first = last → link;              /* obtain node to be deleted */
    last → link = last → link ;      /* store new first node in link of last */
    printf( " The item deleted is  %d \n ", first→info )
    freenode( first ) ;               /* delete the old first node */
    return(last ) ;
}
    
```

* លុប **item** ចេញពីផ្នែកខាងចុងនៃ List (**Delete a node from the rear end**)



- 1- `prev = last → link ;`
`while(prev → link != last)`
`{`
`prev = prev → link ;`
`}`
- 2- `prev → link = last → link ;`
- 3- The last node can be deleted using the statement
`freenode (last);`
`return(prev) ;`

គេហទំព័រ

អនុគមន៍សម្រាប់លុប **item** ចេញពីផ្នែកខាងចុងនៃ List គឺ

```

NODE delete_rear ( NODE last )
{
    NODE prev ;
    if( last == NULL)
    {
        printf(" list is empty \n ") ;
        return (NULL) ;
    }
    if(last → link == last)    /* only one node is present */
    {
        printf(" The item deleted is  %d\n" , last → info) ;
        freenode (last) ;
        return (NULL) ;
    }
    /* Obtain address of previous node */
    prev = last → link ;
    while(prev → link != last )
    {
        prev = prev → link ;
    }
}

```

```
prev → Link = last → link ;          /* prev node is made the last node */
printf( " The item deleted is %d \n ", last → info) ;
freenode (last) ;                      /* delete the old last node */
return( prev) ;                        /* return the new last node */
}
```

ឧទាហរណ៍៖

Function to display the contents of the circular queue.

```
void display ( NODE Last )
{
    NODE temp ;
    if( Last == NULL)
    {
        printf( " List is empty \n ") ;
        return ;
    }
    printf( "Contents of the list is \n ") ;
    for( temp = last → link; temp != last ; temp = temp → link)
        printf( " % d", temp → info ) ;
    printf( "% d \n ", temp → info ) ;
}
```

ឧទាហរណ៍៖

The C Program to implement queue using circular list.

```
# include < stdio.h >
# include < alloc.h >
# include < process.h >
struct node
{
    int info ;
    struct node *link ;
};
```

```
typedef struct node * NODE ;
/* function to get a node from the availability list */
/* function to return node to availability list */
/* function to insert an item at the front end */
/* function to insert an item at the rear end */
/* function to delete an item from the front end */
/* function to delete an item from the rear end */
/* function to display the contents of the list */

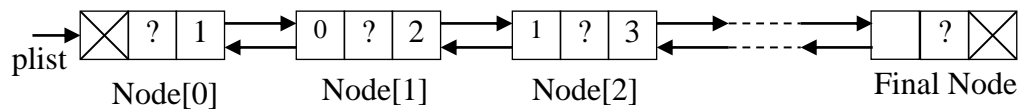
void main( )
{
    NODE last ;
    int choice, item ;
    last = NULL ;
    for( ; ; )
    {
        printf( " 1: Insert_Front 2: Insert_Rear \n " )
        printf( " 3: Delete_Front 4 : Delete_Rear \n " ) ;
        printf( " 5: Display 6 : Exit \n " ) ;
        printf( " Enter the choice \n " ) ;
        scanf( " %d ", &choice ) ;
        switch(choice)
        {
            case 1:
                printf( " Enter the item to be inserted \n " ) ;
                scanf( " %d ", &item ) ;
                last = insert_front( item, last ) ;
                break ;
            case 2:
                printf( " Enter the item to be inserted \n " ) ;
                scanf( " %d ", &item ) ;
                last = insert_rear( item, last ) ;
                break ;
            case 3:
                last = delete_front( last ) ;
                break ;
            case 4:
                last = delete_rear( last ) ;
                break ;
            case 5:
                display( last ) ;
                break ;
            default:
                exit( 0 ) ;
        }
    }
}
```


5-4 Doubly Linked List

5-4-1: និយមន័យ

Doubly Linked List គឺជាសំណុំ Node ជាច្រើនដែលត្រូវបានតភ្ជាប់គ្នា ហើយរាល់ Node នីមួយៗរបស់វាមាន 3 fields គឺ field info សម្រាប់ផ្ទុកទិន្នន័យ រីឯ field Left និង Right សម្រាប់ផ្ទុក Pointer ចង្អុលទៅកាន់ Node បន្ទាប់ ហើយ Node ដំបូងមាន field Left ស្មើ NULL រីឯ Node ចុងក្រោយមាន field Right ស្មើ NULL ។

គេមានទម្រង់ទូទៅរបស់ Doubly Linked List ដូចខាងក្រោម៖



5-4-2: ប្រមាណវិធីលើ Doubly Linked List ដោយប្រើ Dynamic

(ក)-របៀបបង្កើត Data Structure ដោយប្រើ Dynamic

```
struct Nodetype
{
    datatype info ;
    struct Nodetype * right ;
    struct Nodetype * left ;
} * Node ;
typedef struct Nodeype * pointype;
```

(ខ)- បណ្តា Algorithms

(1)- Algorithm: Getnode

```
int Getnode ( )
{
    int p ;
    p = ptr ;
    ptr = Node[ ptr ] . next ;
    return( p ) ;
}
```

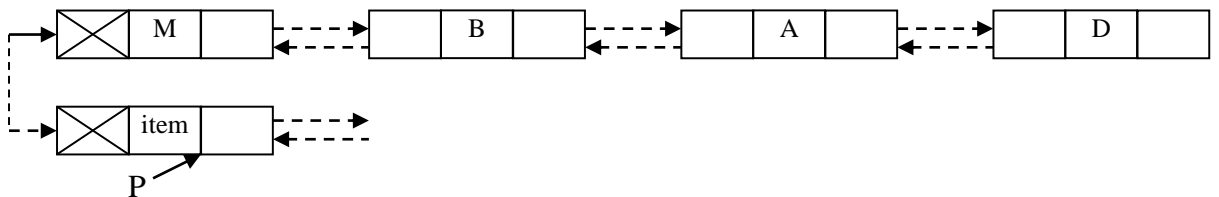
(2)- Algorithm: Freenode

```
void Freenode (int p )
{
    Node[p] . next = ptr;
    ptr = p ;
}
```

(3)- Algorithm: Initialize

```
void Initialize(pointertype *plist)
{
    * plist = NULL ;
}
```

(4)- Algorithm : Insert first



```
void Insert_first( pointertype * plist , datatype Item)
{
    pointertype p = Getnode( ) ;
    p → info = Item ;
    p → right = * plist ;
    ( * plist) → left = p ;
    p → left = NULL ;
    * plist = p ;
}
```

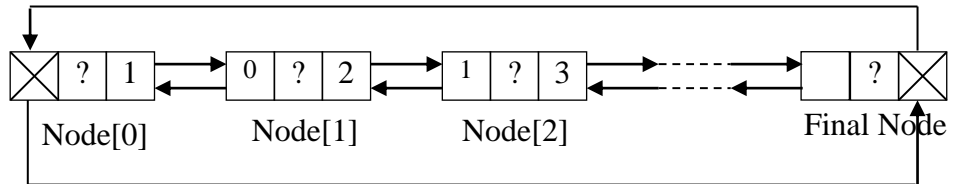
5-5 Circular Doubly Linked List

5-5-1: និយមន័យ

Circular Doubly Linked List គឺជាសំណុំ Node ជាច្រើនត្រូវបានតភ្ជាប់គ្នា ហើយរាល់ Node នីមួយៗរបស់វាមាន 3 fields គឺ field info សម្រាប់ផ្ទុកទិន្នន័យ រីឯ field Left និង Right សម្រាប់ផ្ទុក Pointer ចង្អុលទៅកាន់ Node បន្ទាប់ ហើយ Node ដំបូងមាន field

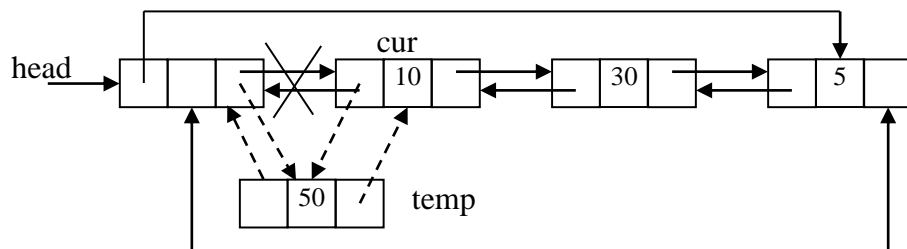
Left ចង្អុលទៅកាន់ចុងក្រោយ រីឯ Node ចុងក្រោយមាន field Right ចង្អុលទៅកាន់ Node ដំបូង។

គេមាន Structure ទូទៅរបស់ Circular Doubly Linked List ដូចខាងក្រោម៖



5-5-2: ប្រមាណវិធីលើ Circular Doubly Linked List ដោយប្រើ Dynamic

(1)-បន្ថែម Node នៅខាងមុខនៃ List (Insert a node at the front end)



(Figure to insert a node at the front end)

- cur = head → rlink ;
- head → rlink = temp ;
- temp → llink = head ;
- temp → rlink = cur ;
- cur → llink = temp ;

ឧទាហរណ៍៖

អនុគមន៍សម្រាប់បន្ថែម Node នៅខាងមុខ List គឺ

NODE insert_front (int item, NODE hea)

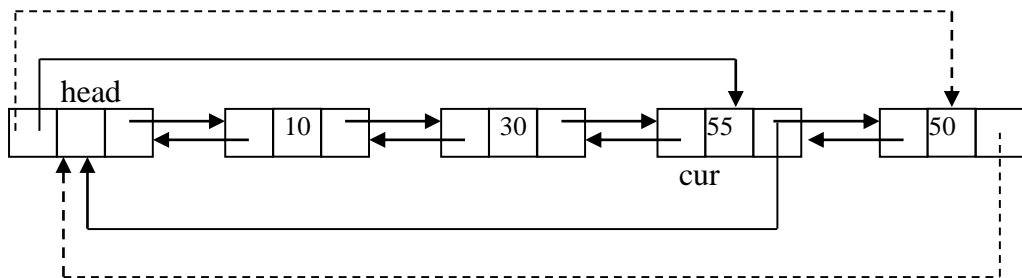
```
{
    NODE temp, cur ;          /* Node to be inserted */
    temp = getnonde ( ) ;
    temp → info = item ;
    /* obtain address of first node */
    cur = head → rlink ;
    /* insert between beader node and first node */
}
```

```

head → r Link = temp ;
temp Llink = head ;
temp → r link = cur ;
cur → L link = temp ;
return(head) ;    /* return the header node */
}

```

(2)-បន្ថែម Node នៅខាងក្រោយនៃ List(Insert node at the rear end)



(Figure to insert a node at the rear end)

ឧទាហរណ៍៖

អនុគមន៍សម្រាប់បន្ថែម Node នៅខាងក្រោយនៃ List គឺ

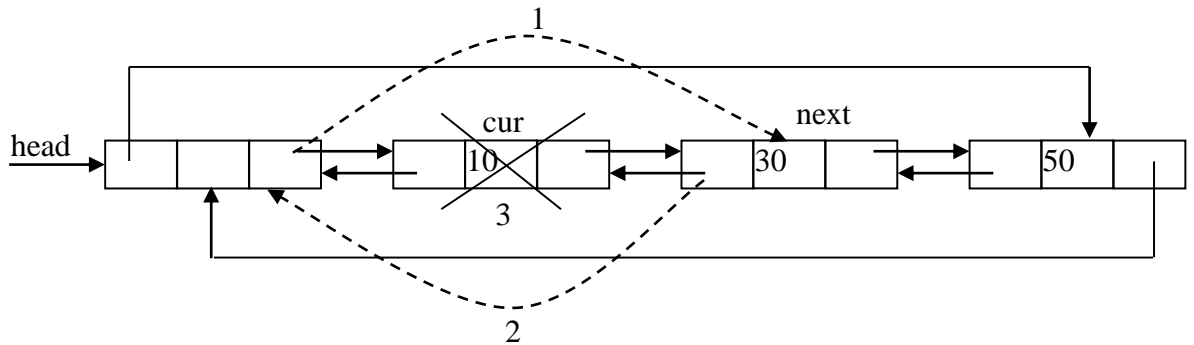
NODE insert_rear (int item, NODE head)

```

{
    NODE temp , cur ;
    /* Node to be inserted */
    temp = getnode ( ) ;
    temp → info = item ;
    /* obtain address of the last node */
    cur = head → Llink;
    /* insert at the end of the list */
    head → Llink = temp ;
    temp → rlink = head ;
    temp → rlink = cur ;
    cur → rlink = temp ;
    /* return the header node */
    return(head) ;
}

```

(3)-លុប Node ចេញពីខាងមុខ List(Delete a node from the front end)



(Figure to delete a node at the front end)

- cur = head → rlink ;
- next = cur → rlink ;
- 1- head → rlink = next ;
- 2- next → Llink = head ;
- 3- printf("The item deleted is %d \n ", cur → info);

ឧទាហរណ៍៖

អនុគមន៍សម្រាប់លុប Node ចេញពីខាងមុខ List គឺ

```

NODE delete_front( NODE head )
{
    NODE cur, next ;
    if( head -> rlink == head )
    {
        printf( " Dequeue is empty \n " ) ;
        return( head ) ;
    }

    cur = head -> rlink ;      /* first node is known */
    next = cur -> rlink ;      /* second node which will be the first node */
    /* adjust pointers and delete the node */
    head -> rlink = next ;
    next -> Llink = head ;
    printf( " The node to be deleted is %d \n ", cur -> info ) ;
    freenode( cur ) ;
    return( head ) ;
}
    
```

ឧទាហរណ៍១៖

Program to implement insert at front / rear / before / after a node, to delete front / rear / based on item / all nodes.

```
# include < stdio.h >
# include < alloc.h >
# include < process.h >
struct node
{
    int info ;
    struct node *Llink ;
    struct node *rlink ;
};
typedef struct node *NODE ;
/* function to insert at the front end */
/* function to insert at the rear end */
/* function to delete the first node */
/* function to display the contents of the list */
void display( NODE head )
{
    NODE temp ;
    if( head -> rlink == head)
    {
        printf( " Deque is empty\n " ) ;
        return ;
    }
    printf( " Contents of the Deque is \n " ) ;
    for( temp = head -> rlink ; temp != head ; temp = temp -> rlink )
        printf( " %d ", temp -> info ) ;
    printf( " \n" ) ;
}
void main( )
{
    NODE head ;
    int choice, item ;
    head = Getnode( ) ;
    head -> rlink = head ;
    head -> Llink = head ;
    for( ; ; )
    {
        printf( " 1: Insert_front 2: Insert_rear \n " ) ;
        printf( " 3: Delete_front 4: Delete_rear \n " ) ;
        printf( " 5: Display " ) ;
        printf( " 6: Exit \n " ) ;
        printf( " Enter the choice \n " ) ;
        scanf( " %d ", &choice ) ;
        switch( choice )
        {
```

```

case 1:
    printf( " Enter the item to be inserted \n " );
    scanf( " %d ", &item );
    head = insert_front( item, head );
    break ;
case 2:
    printf( " Enter the item to be inserted \n " );
    scanf( " %d ", &item );
    head = insert_rear( item, head );
    break ;
case 3:
    head = delete_front( head );
    break ;
case 4:
    head = delete_rear( head );
    break ;
case 5:
    display( head );
    break ;
default : exit( 0 );
    }
}
}

```

ឧទាហរណ៍២៖

សរសេរកម្មវិធីសម្រាប់ភ្ជាប់ **Linked List** ពីរ

```

#include< stdio.h >
#include < conio.h >
#include < alloc.h >
#include < stdlib.h >
#include < process.h >
typedef struct node
{
    int data;
    struct node *LINK;
}NO;
NO *START, *p, *q ;
void DISPLAY( NO* )
NO* COMBINE( );
NO* CREATE( );
void main( )
{

```

```

int n;
NO *START, *t;
clrscr( );
COMBINE( );
getch( );
}
NO *COMBINE( )
{
    NO *p, *q, *r;
    printf("\nEnter Number of nodes in the first Linked List");
    r = CREATE( );
    p = r;
    printf("\nFirst List\n");
    DISPLAY( r );
    printf("\nEnter Number of nodes in the second Linked List");
    q = CREATE( );
    printf("\nSecond List is\n");
    DISPLAY( q );
    while( p -> LINK!=NULL)
    {
        printf("%d",p -> data);
        p = p -> LINK;
    }
    p -> LINK = q;
    printf("\n\n Combine List is\n");
    DISPLAY( r );
}
NO* CREATE( )
{
    NO *p, *START, *LAST;
    int i, k, num, firstnode;
    firstnode = 1;
    START = NULL;
    i = 1;
    n = 0;
    scanf("%d",&num);
    while( i <= num )
    {
        printf("Enter value for node %d : ",i);
        scanf("%d",&n);
        p = (NO*)malloc( sizeof(NO));
        p -> data = n;
        p -> LINK = NULL;
        i ++;
        if( firstnode )
        {
            START = p;
            LAST = START;
        }
    }
}

```



```

        firstnode = 0;
    }
    else
    {
        LAST -> LINK = p;
        LAST = p;
    }
}
return( START );
}
void DISPLAY( NO *START )
{
    No *p;
    p = START;
    do
    {
        printf("%d ->", p -> data );
        p = p -> LINK;
    }while( p != NULL );
    printf(" NULL");
}

```

នៅពេល **RUN** កម្មវិធីបង្ហាញដូចខាងក្រោម៖

Enter Number of nodes in first Linked List 3

Enter value for node 1: 1

Enter value for node 2: 2

Enter value for node 3: 3

First List is

1 -> 2 -> 3 -> NULL

Enter Number of nodes in second Linked List

Enter value for node 1: 10

Enter value for node 2: 20

Enter value for node 3: 30

Enter value for node 4: 40

Second List is

10 -> 20 -> 30 -> 40 -> NULL

Combined List is

1 -> 2 -> 3 -> 10 -> 20 -> 30 -> 40 -> NULL

ឧទាហរណ៍៣៖

សរសេរកម្មវិធីដើម្បីបង្កើត និងបង្ហាញ **Nodes** នៃ **Doubly Linked List**

```
#include <stdio.h>
#include <conio.h>
#include <alloc.h>
#include <stdlib.h>
#include <process.h>
typedef struct doublenode
{
    int data;
    struct doublenode *LLink, *RLink;
} DN;

DN *CREATE( DN *);
DN *DISPLAY(DN *);

void main( )
{
    int i, n;
    DN *START, *f, *p, *q;
    clrscr( );
    printf("\nEnter data for nodes of the doubly Linked List, 0 to stop");
    printf("\nEnter data for node 1:");
    scanf("%d",&n);
    f = ( DN *)malloc( sizeof( DN ));
    f -> data = n;
    f -> LLink = NULL;
    f -> RLink = NULL;
    START = f;
    CREATE( START );
    DISPLAY( START );
    getch( );
}

DN *CREATE( DN * START )
{
    DN *r, *p, *f ;
    int i = 2, n ;
    p = START;
    do
    {
        f = (DN *)malloc(sizeof(DN));
        printf("\nEnter data for node %d :", i);
        scanf("%d", &n);
        f -> data = n;
```

```

        f -> RLink = NULL;
        f -> LLink = p;
        p -> RLink = f
        p = p -> RLink;
        i ++;
    }while( n != 0 );
    return( 0 );
}
DN *DISPLAY(DN* START)
{
    DN *p, *f;
    int i = 0;
    p = START;
    printf("\nThe doubly Linked List is \n");
    while( p -> RLink != NULL)
    {
        printf("%d <==>", p -> data);
        p = p -> RLink;
        i ++;
    }
    printf("NULL\n");
}

```

នៅពេល **RUN** កម្មវិធីបង្ហាញដូចខាងក្រោម៖

Enter data for nodes of the doubly Linked List, 0 to stop

Enter data for node 1: 12

Enter data for node 2: 23

Enter data for node 3: 56

Enter data for node 4: 45

Enter data for node 5: 78

Enter data for node 6: 89

Enter data for node 7: 65

Enter data for node 8: 98

Enter data for node 9: 32

Enter data for node 10: 21

Enter data for node 11: 0

The doubly Linked List is

12 <==> 23 <==> 56 <==> 45 <==> 78 <==> 89 <==> 65 <==> 98 <==> 32
 <==>21<==>NULL

ឧទាហរណ៍៤៖

សរសេរកម្មវិធីដើម្បីតំរៀប **Linked List** តាមលំដាប់ពីតូចទៅធំ

```
# include< stdio.h >
# define NULL 0
struct List_element
{
    int item;
    struct List_element *next;
};
typedef struct List_element node;

void main( )
{
    node *start, *List, *pass;
    int key, choice, temp;
    node *create( );
    void display( );

    printf("Creating List\n");
    printf("\nTYPE-999 to STOP");
    start = create(start);

    printf("\n\nLIST :\n\n");
    display(start);
    printf("\n\n");

    /* Sorting the List */

    List = start;
    for(; List -> next != NULL; List = List -> next)
    {
        for(pass = List -> next; pass != NULL; pass = pass -> next)
        if(List -> item > pass -> item)
        {
            temp = List -> item;
            List -> item = pass -> item;
            pass -> item = temp;
        }
    }
    printf("\nSORTED LIST:\n");
    display(start);
    printf("\n\n");
}

/* Creating a Linked List */
node *create(record)
node *record;
{
```

```

node *temp, *prev;
int info;

prev = NULL;
record = NULL;

printf("\nData item?");
scanf("%d", &info);
printf("%d", info);

while(info != -999)
{
    /* Allocate memory space for next node */
    temp = (node *)malloc(sizeof(node));

    temp -> item = info;
    temp -> next = NULL;
    if(prev == NULL)
        record = temp;
    else
        prev -> next = temp;
    prev = temp;

    printf("\nData item:");
    scanf("%d", &info);
    printf("%d", info);
}
return(record);
}
/* Displaying the Linked List */

void display(record)
node *record;
{
    while(record != NULL)
    {
        /* Display the data item */
        printf("-> %d", record -> item);

        /* Get the next data item for display */
        record = record -> next;
    }
    return;
}

```

នៅពេល RUN កម្មវិធីបង្ហាញលទ្ធផលដូចខាងក្រោម៖

Creating List

TYPE-999 to STOP

Data item ?45

Data item ?3

Data item ?78

Data item ?9

Data item ?1

Data item ?25

Data item ?6

Data item ?13

Data item ?11

Data item ?99

Data item ?10

Data item ?-999

LIST:

-> 45 -> 3 -> 78 -> 9 -> 1 -> 25 -> 6 -> 13 -> 11 -> 99 -> 10

SORTED LIST:

->1 -> 3 -> 6 -> 9 -> 10 -> 11 -> 13 -> 25 -> 45 -> 78 -> 99

ឧទាហរណ៍៥៖

ចូរសរសេរកម្មវិធីដើម្បីរកតម្លៃធំបំផុត និងតូចបំផុតនៃ Linked List

```
# include< stdio.h >
```

```
# include< malloc.h >
```

```
typedef struct LIST
```

```
{
```

```
    int data;
```

```
    struct LIST *next;
```

```
}LIST;
```

```
/* Print List */
```

```
void PrintList(FIRST)
```

```
LIST *FIRST;
```

```
{
```

```
    LIST *List;
```

```
    printf("\nRoot");
```

```
    for(List = FIRST; List; List = List -> next)
```

```
        printf("->%d", List -> data);
```

```
    printf("\n");
```

```
}
```

```
/* Enter data into the List */(បញ្ចូលទិន្នន័យទៅក្នុងList)
```

```
LIST *InsertList(data, FIRST);
```

```
int data;
```

```

LIST *FIRST;
{
    LIST *NEW, *current, *pred;
    NEW = (LIST*)malloc(sizeof(LIST));
    NEW -> data = data;
    NEW -> next = NULL;
    if(!FIRST)
        return(NEW);
    for(current = FIRST; current; current = current -> next)
        pred = current;
    pred -> next = NEW;
    return(FIRST);
}

/* To find Maximum & Minimum in the List */(រកតម្លៃធំបំផុត និងតូចបំផុតនៅក្នុង
List)
int MaxMinList(max, min, FIRST)
int *max, *min;
LIST *FIRST;
{
    LIST *List;
    if(!FIRST)
        return(0);
    *max = *min = FIRST -> data;
    for(List = FIRST -> next; List; List = List -> next)
    {
        if(List -> data > *max)
            *max = List -> data;
        else
            if(List -> data < *min)
                *min = List -> data;
    }
    return(1);
}

void main( )
{
    int data, max, min;
    LIST *List = NULL;
    while(1)
    {
        printf("Key to insert < 999 to Stop >?");
        scanf("%d", &data);
        printf("%d", data);
        if(data == 999)
            break;
        List = InsertList(data, List);
    }
}

```

```
PrintList(List);
if(MaxMinList(&max, &min, List))
{
    printf("\nThe maximum Number in the List: %d\n", max);
    printf("\nThe Minmum Number in the List: %d\n", min);
}
else
    printf("\nNo Elements in the List");
}
```

នៅពេល **RUN** កម្មវិធីបង្ហាញលទ្ធផលដូចខាងក្រោម៖

```
Key to Insert< 999 to Stop >? 77
Key to Insert< 999 to Stop >? 41
Key to Insert< 999 to Stop >? -28
Key to Insert< 999 to Stop >? 0
Key to Insert< 999 to Stop >? 99
Key to Insert< 999 to Stop >? -59
Key to Insert< 999 to Stop >? -1
Key to Insert< 999 to Stop >? 7
Key to Insert< 999 to Stop >? 999
Root -> 77 -> 41 -> -28 -> 0 -> 99 -> -59 -> -1 -> 7
The Maximum Number in the List : 99
The Minimum Number in the List : -59
```

ឧទាហរណ៍៦

ចូរសរសេរកម្មវិធីដើម្បីត្រួតលំដាប់លេខដោយប្រើ Linked List

```
# include< stdio.h >
# include< malloc.h >
# define NULL 0
struct List_element
{
    int item;
    struct List_element *next;
};
typedef struct List_element node;

void main( )
{
```



```

node *start, *prev, *current, *save;
int key;
int choice;

node *create( );
void display( );

/* Reversing a List */
printf("Creatin List\n");
printf("\nTYPE -999 to STOP");
start = create(start);

printf("\n\nGIVEN LIST :\n\n");
display(start);
printf("\n\n");

/* reversing the list */
current = start;
prev = NULL;
while(current != NULL)
{
    save = current -> next;
    current -> next = prev;
    prev = current;
    current = save;
}
start = prev;

printf("\n\nREVERSED LIST :\n\n");
display(start);
printf("\n\n");
}
node *create(record) /* Create a Linked List */
node *record;
/* Argument point to the current node */
{
    node *prev, *temp;
    int info;

    prev = NULL;
    record = NULL;

    printf("\nData item:");
    scanf("%d", &info);
    printf("%d", info);

    while(info != -999)
    {
        /* Allocate memory space for new node */
        temp = (node *)malloc(sizeof(node));
        temp -> item = info;
        temp -> next = NULL;
    }
}

```

```

        if(prev == NULL)
            record = temp;
        else
            prev -> next = temp;
        prev = temp;

        printf("\nData item:");
        scanf("%d", &info);
        printf("%d", info);
    }
    return(record);
}
/* display the Linked List */
void display(record)
node *record;
{
    while(record != NULL)
    {
        /* display the data item */
        printf("->%d", record -> item);

        /* Get the next data item */
        record = record -> next;
    }
    return(0);
}

```

នៅពេល **RUN** កម្មវិធីបង្ហាញលទ្ធផលដូចខាងក្រោម៖

Creating List

TYPE -999 to STOP

Data item : 5

Data item : 4

Data item : 3

Data item : 2

Data item : 1

Data item : -999

GIVEN LIST :

-> 5 -> 4 -> 3 -> 2 -> 1

REVERSED LIST :

-> 1 -> 2 -> 3 -> 4 -> 5

លំហាត់អនុវត្ត

1-What is List? Discuss the functions defined to operate on List.

2-Explain the typical basic operated on Linked List.

3-ចូរពន្យល់អំពី Single Linked List ដោយគូសដ្យាក្រាមបញ្ជាក់។

4-Write notes on Circular Singly Linked List.

5-Explain the Insert/Delete a node at the rear and front end from circular singly linked list.

6-ទិន្នន័យរបស់ទំនិញរួមមាន៖ លេខកូដ ឈ្មោះ និងតម្លៃ។ ហើយទិន្នន័យរបស់ទំនិញនីមួយៗត្រូវបានគេផ្ទុកជា Singly Linked List មួយ។

ចូរបង្កើត Data Structures សម្រាប់ Singly Linked List ខាងលើនេះ។

7-ចូរសរសេរប្រៀបបង្កើត Data Structure របស់ Array Linked List និង Dynamic Linked List ព្រមទាំងលើកឡើងពីភាពខុសគ្នារបស់វា។

8-គេមាន Singly Linked List មួយ។ ចូរសរសេរ Algorithm ជា C code(Function)ដើម្បីលុប Node មួយដែលនៅខាងមុខ Node ចុងក្រោយ។

9-តើ Algorithm Singly Linked List ខាងក្រោមនេះសម្រាប់រកអ្វី?

```
pointertype *Linkedlist(pointertype *plist)
{
    pointertype *ptr = plist;
    datatype temp;
    while(ptr->next!=NULL)
        ptr = ptr -> next;
}
```

10-គេមាន Circular Singly Linked List មួយមិនទទេ៖

ក-ចូរសរសេរ Algorithm(C code) ដើម្បីបន្ថែម Node ថ្មីមួយដែលផ្ទុកតម្លៃ Newdata ចូលទីតាំងចុងក្រោយនៃ Circular Singly Linked List នេះ។

ខ-ចូរសរសេរ Algorithm(C code)ដើម្បីលុប Node ទីមួយចេញពី Circular Singly Linked List នេះ។

គ-ចូរសរសេរ Algorithm(C code)ដើម្បីបន្ថែម Node ថ្មីមួយដែលផ្ទុកតម្លៃ Newdata

ចូលទីតាំងដំបូងនៃ Circular Singly Linked List នេះ។

11-ចូរពន្យល់ពីភាពខុសគ្នារវាង Singly Linked List និង Doubly Linked List។

12-គេមាន Doubly Linked List មួយមិនទទេ៖

ក-ចូរសរសេរ Algorithm(C code) ដើម្បីបន្ថែម Node ថ្មីមួយដែលផ្ទុកតម្លៃ Newdata ចូលទីតាំងចុងក្រោយនៃ Doubly Linked List ខាងលើនេះ។

ខ-ចូរសរសេរ Algorithm(C code)ដើម្បីលុប Node ទីមួយចេញពី Doubly Linked List ខាងលើនេះ។

គ-ចូរសរសេរ Algorithm(C code)ដើម្បីបន្ថែម Node ថ្មីមួយដែលផ្ទុកតម្លៃ Newdata ចូលទីតាំងដំបូងនៃ Doubly Linked List នេះ។



មេរៀនទី៦

Trees

6-1 សេចក្តីផ្តើម

មេរៀននេះណែនាំឲ្យយើងស្វែងយល់អំពីបញ្ហាជាមូលដ្ឋានរបស់ Trees ដែលរួមមាន Structure មួយចំនួនរបស់វា និងការធ្វើប្រមាណវិធីនៅលើ Structure នីមួយៗរបស់ Trees ដើម្បីឲ្យដោះស្រាយបញ្ហាតាមគោលការណ៍របស់ Trees ។

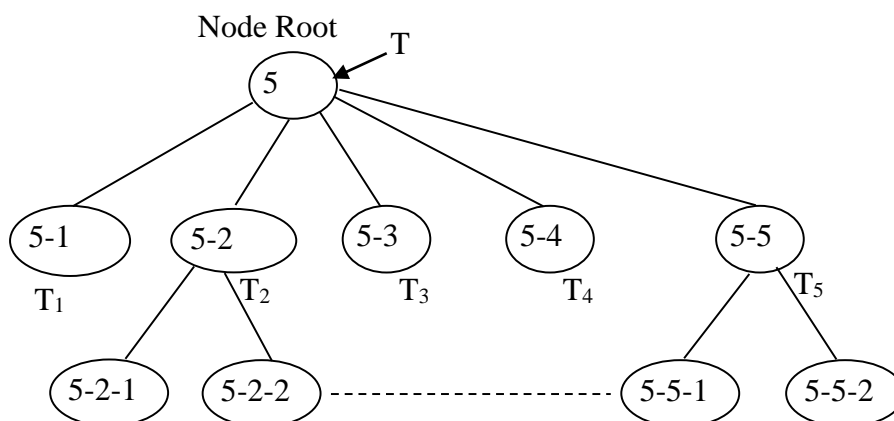
Trees ត្រូវបានឲ្យនិយមន័យថា គឺជាសំណុំ Node ជាច្រើនត្រូវបានភ្ជាប់គ្នាក្នុងនោះមាន Node ពិសេសមួយឲ្យឈ្មោះថា Node Root ហើយរាល់ Node នីមួយៗត្រូវបានបែងចែក Subtrees ជាបន្តបន្ទាប់ និងរាល់ទំនាក់ទំនងរវាង Node នីមួយៗក្នុង Trees ត្រូវបានហៅថា ទំនាក់ទំនង Father Son ។ ម៉្យាងទៀត Trees ត្រូវបានឲ្យនិយមន័យតាមបែប Recursive ដូចខាងក្រោម៖

-មាន Node មួយគឺមាន Trees មួយ ដែល Node នោះគឺជា Node Root របស់ Trees ។

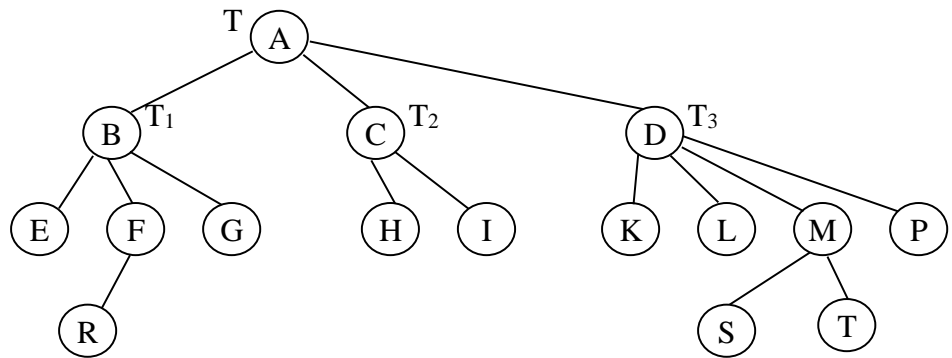
-ប្រសិន Trees T មួយមាន n Node ដែលមាន $T_1; T_2; \dots; T_k$ ជា Subtrees ដោយមាន $n_1; n_2; \dots; n_k$ ជា Node Root របស់វា ។

គេមាន Structure របស់បញ្ហាមួយចំនួនដែលសំដែងដោយ Trees ដូចខាងក្រោម៖

1- Trees សំដែងឲ្យមាតិកានៃមេរៀនទី៥ គឺ



2- Trees សំដែងឲ្យសំណុំនៃបញ្ហាដូចខាងក្រោម៖



- Node[A] គឺជា Node Root របស់ Trees ។
- Node[B] គឺ Node[C], Node[D] ជា Node Root របស់ Subtrees ។
- Node[A] គឺជា Father Node របស់ Node[B] , Node[C] , Node[D] ។
- Node[B] គឺ Node[C], Node[D] ជា Son Node របស់ Node[A] ។

គេមានលក្ខណៈជាមូលដ្ឋានមួយរបស់ Trees ដូចខាងក្រោម៖

- Node degree គឺជាចំនួន son របស់ Node នោះ។
- Trees degree គឺជា degree ខ្ពស់បំផុតរបស់ Node ដែលមាននៅលើ Trees ។
- Leaf Node គឺជាបណ្តា Node ដែលមាន degree ស្មើសូន្យ។
- Branch Node គឺជាបណ្តា Node ដែលមាន degree ខុសពីសូន្យនិងពុំមែនជា Node Root របស់ Trees ។

-Level of Node គឺជា Structure របស់ Node ដែលត្រូវបានកំណត់តាមលំដាប់ Node Root របស់ Trees ស្ថិតនៅលើ Level[0] ហើយប្រសិនបើ Father Node ស្ថិតនៅលើ Level[i] នោះបណ្តា Son Node របស់វាស្ថិតនៅលើ Level[i + 1] (ជាមួយ $i = 0, 1, 2, \dots$) ។

- Height រឺ Depth of Trees គឺជា Level ខ្ពស់បំផុតរបស់បណ្តា Node ដែលមាននៅលើ Trees ។

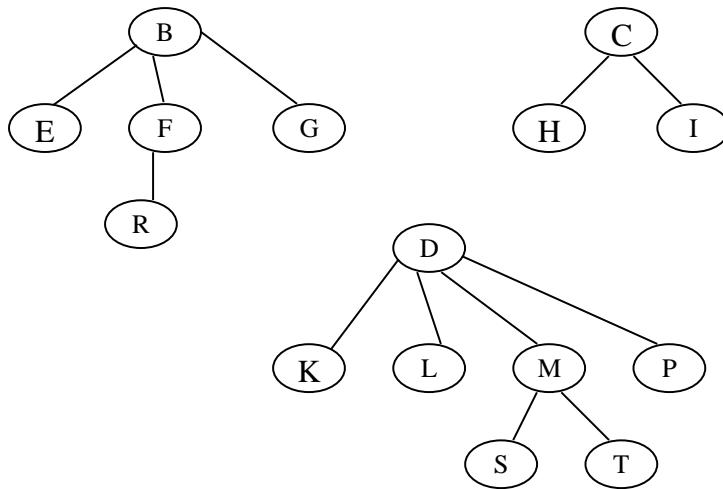
-Path of Node គឺជាចំនួន Node ស្ថិតនៅលើខ្សែ Node ដែលបានឆ្លងកាត់ពី Node ណាមួយដល់ Node ណាមួយទៀត។

-Path of length គឺជាចំនួន Node នៅលើ Path -1 ហើយគឺជាប្រវែងដែលឆ្លងកាត់នៅលើ Path - 1 ស្មើនឹង Path នៃ Node -1 ។

-Forest of Trees គឺជាបណ្តា Subtrees ដែលត្រូវបានគេលុប Node Root របស់ Trees ។

ឧទាហរណ៍៖

Forest of trees



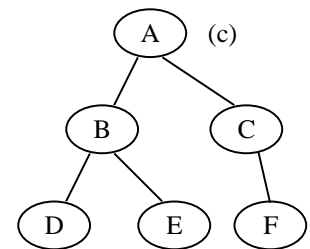
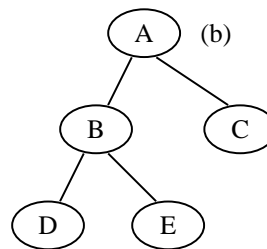
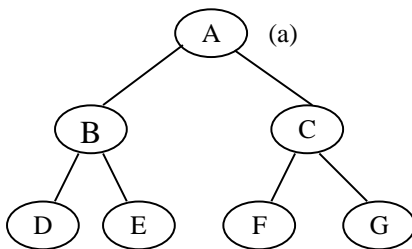
6-2 Binary Trees

6-2-1 និយមន័យ

Binary Trees គឺជា Trees ដែលរាល់ Node នីមួយៗរបស់វាមាន degree មិនលើសពី២ ហើយលំដាប់របស់ Node នីមួយៗត្រូវបានបែងចែកជាពីរ Subtrees គឺ Left Subtrees និង Right Subtrees ។

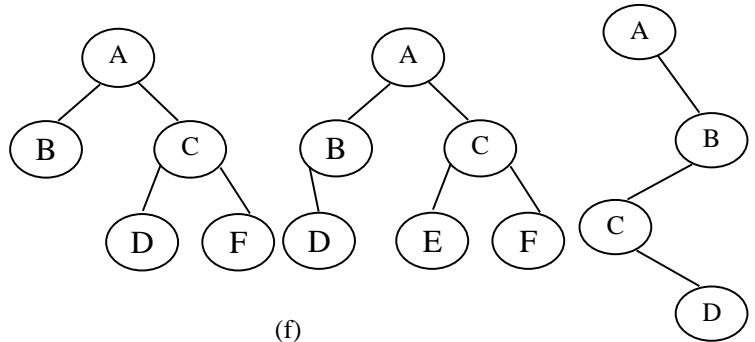
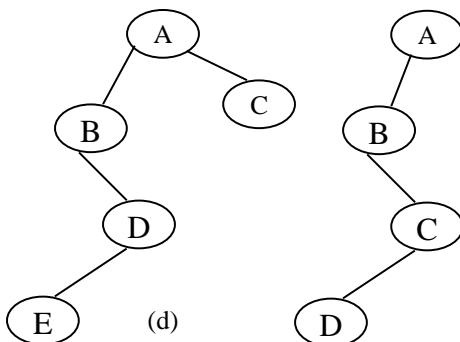
គេមាន Structure Binary Trees ដែលសំដែងឲ្យបញ្ជាក់មួយចំនួនដូចខាងក្រោម៖

1-Binary Trees សំដែងឲ្យបញ្ជាក់នៃសំណុំធាតុដូចខាងក្រោម៖



Full Binary Trees

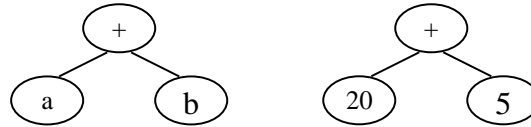
Complete Binary Trees



2-Binary Trees សំដែងឱ្យកន្សោមពិជគណិតដូចខាងក្រោម៖

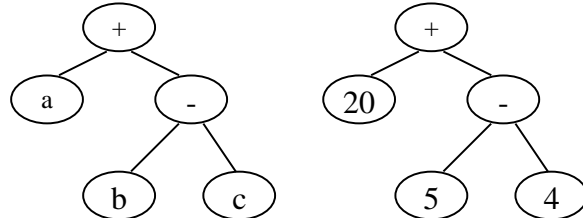
$a + b$

$20 + 5$



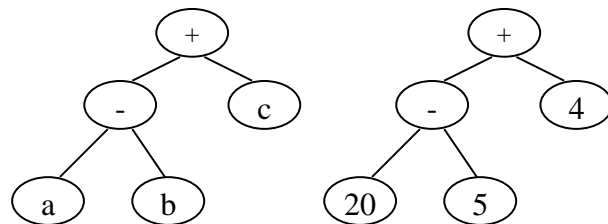
$a + (b - c)$

$20 + (5 - 4)$



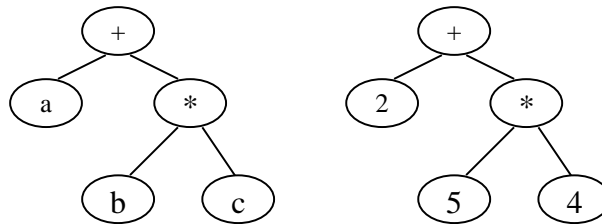
$(a - b) + c$

$(20 + 5) + 4$



$a - (b * c)$

$2 - (5 * 4)$



* លក្ខណៈមូលដ្ឋានមួយចំនួនរបស់ Binary Trees គឺ

-ចំនួន Node នៅលើ Level[i] $\leq 2^i$

-ចំនួន Node នៅលើ Binary Trees ដែលមាន Height រឺ Depth ជា h $\leq 2^{h+1}-1$ ។

6-2-2 របៀបផ្ទុក Node នីមួយៗរបស់ Binary Trees

គេមានរបៀបផ្ទុក Node នីមួយៗរបស់ Binary Trees តាមពីរបៀប គឺផ្ទុកជាបន្តបន្ទាប់ និងផ្ទុកដោយការតភ្ជាប់គ្នា។

១-របៀបផ្ទុកជាបន្តបន្ទាប់(ផ្ទុកជា Array)

ផ្ទុកជាបន្តបន្ទាប់ គឺជាការផ្ទុកតាមលក្ខណៈរ៉ឺចទ័រ គឺផ្ទុកអស់ពី Level មួយចូលដល់ Level មួយទៀតដោយផ្ទុកតាមលំដាប់ពីឆ្វេងទៅស្តាំពី Level ដំបូងដល់ Level ចុងក្រោយរបស់ Binary Trees ។

លំនាំផ្ទុកដោយ Array គឺអនុវត្តការផ្ទុក Node Root ត្រូវបានផ្ទុកនៅទីតាំងដំបូងរបស់

Array ហើយប្រសិន Father Node ផ្ទុកនៅទីតាំងទី i នោះ Left Son Node របស់វាផ្ទុកនៅទីតាំង $2i + 1$ ហើយ Right Son Node របស់វាផ្ទុកនៅទីតាំងទី $2i + 2$ (ដែល $i = 0, 1, 2, \dots$) ។

ឧទាហរណ៍១៖

ជាមួយ Trees (រូប a) , (រូប b) និង (រូប f) គេអាចសរសេររបៀបផ្ទុកជាបន្តបន្ទាប់(ផ្ទុកដោយ Array) ដូចខាងក្រោម៖

A	B	C	D	E	F	G	(រូប a)
[0]	[1]	[2]	[3]	[4]	[5]	[6]	

A	B	C	D	E				(រូប b)
[0]	[1]	[2]	[3]					

A	B	C		D	E	F	(រូប f)

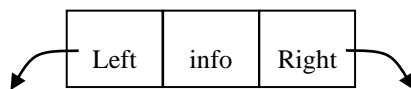
ឧទាហរណ៍២៖

ជាមួយ Trees (រូប d) គេអាចសរសេររបៀបផ្ទុកដោយ Array ដូចខាងក្រោម៖

A	B	C		D			E									
[0]	[1]	[2]	[3]	[4]	[5]	[6]	[7]	[8]	[9]	[10]	[11]	[12]	[13]	[14]	[15]	[16]

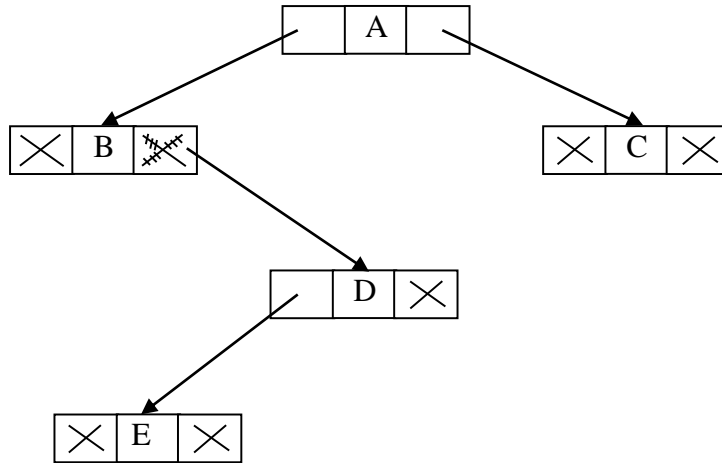
២-របៀបផ្ទុកដោយការតភ្ជាប់(ផ្ទុកជា Linked List)

ផ្ទុកដោយការតភ្ជាប់គ្នា គឺជាការផ្ទុកតាមលំនាំ Node ដែលមាន 3 field គឺ field info សម្រាប់ផ្ទុក Data រីឯ field Left និង field Right សម្រាប់ផ្ទុក Pointer ចង្អុលទៅកាន់ Left Son Node និង Right Son Node របស់វា។



ឧទាហរណ៍៖

ជាមួយ Trees (រូប d) គេអាចសរសេររបៀបផ្ទុក ដោយការតភ្ជាប់គ្នា (ផ្ទុកដោយ Linked List) ដូច ខាងក្រោម៖



6-2-3 របៀបសង់ Binary Trees

ដើម្បីសង់ Binary Trees ដែលមាន n Nodes ឲ្យមាន Height រឺ Depth ទាបបំផុត គឺត្រូវអនុវត្តតាមបីដំណាក់កាល Recursive ដូចខាងក្រោម៖

- + យក Node មួយធ្វើជា Node Root
- + គណនា ចំនួន Node នៅ Left Subtrees តាម $nl = \frac{n}{2}$
- + គណនាចំនួន Node នៅ Right Subtrees តាម $nr = n - nl - 1$

ឧទាហរណ៍

ចូរសរសេរដំណាក់កាលដំណើរការ Recursive សម្រាប់សង់ Binary Trees មួយដែលមាន 8 Nodes ឲ្យមាន Height រឺ Depth ទាបបំផុត និងគូស Structure Binary Trees នោះ។

គេបានដំណាក់កាលដំណើរការ Recursive សម្រាប់សង់ Binary Trees ដែលមាន 8 Nodes ឲ្យមាន Height និង Depth ទាបបំផុតគឺ

- + យក Node មួយធ្វើជា Node Root

$$+ nl = \frac{8}{2} = 4 \text{ nodes}$$

$$+ nr = 8 - 4 - 1 = 3 \text{ nodes}$$

* ដំណាក់កាលដំណើរការ Recursive ដែលមាន 4 Nodes គឺ

- + យក Node មួយធ្វើជា Node Root

$$+ nl = \frac{4}{2} = 2 \text{ nodes}$$

$$+ nr = 4 - 2 - 1 = 1 \text{ node}$$

* ដំណាក់កាលដំណើរការ Recursive ដែលមាន 2 Nodes គឺ

+ យក Node មួយធ្វើជា Node Root

+ $nr = \frac{2}{2} = 1$

+ $nr = 1$

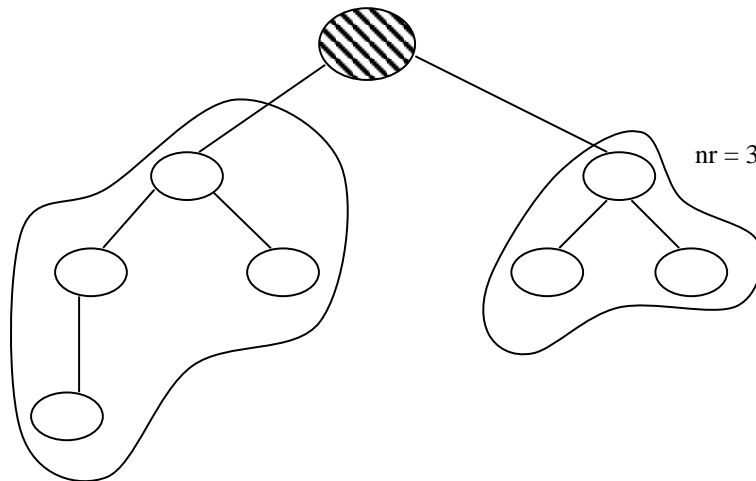
* ដំណាក់កាលដំណើរការ Recursive ដែលមាន 3 Nodes គឺ

+ យក Node មួយធ្វើជា Node Root

+ $nr = \frac{3}{2} = 1.5$

+ $nr = 1.5$

គេអាចគូស Structure Binary Trees ដែលមាន 8 Nodes ឲ្យមាន Height និង Depth ទាបបំផុតគឺ



6-2-4 Traversing Binary Trees

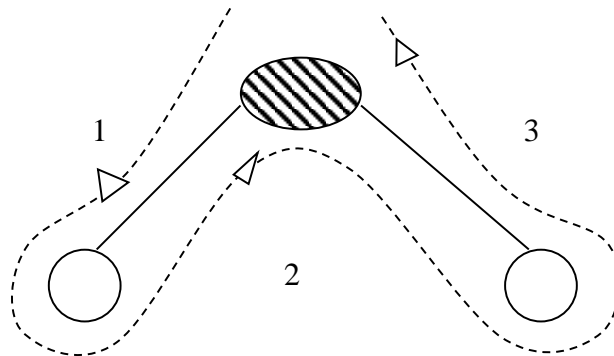
Traversing Binary Trees គឺជាវិធាននៃការឆ្លងកាត់ (visit) នូវរាល់ Node នីមួយៗក្នុង Binary Trees តែមួយលើក។ ដំណាក់កាលដំណើរការរបស់ Traversing Binary Trees ត្រូវបានអនុវត្តតាមបីដំណាក់កាល Recursive ដែលតាងដោយនិមិត្តសញ្ញាដូចខាងក្រោម៖

+ N ជា Visit Node Root

+ L ជា Left Subtrees Traversal

+ R ជា Right Subtrees Traversal

ឧបមាថាគេមាន Structure Binary Trees គឺ



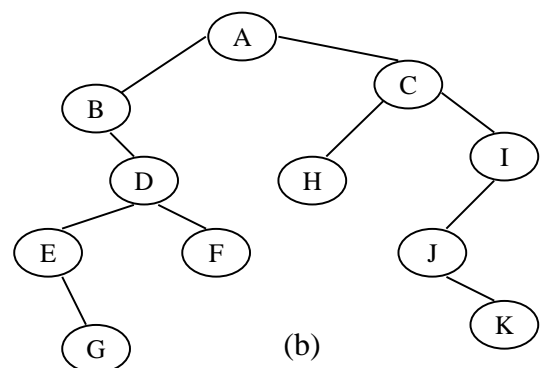
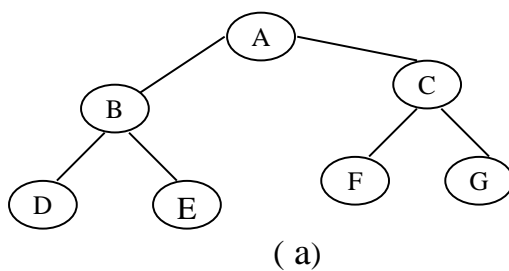
យើងឃើញថានៅលើគំនូសតាងនេះ គឺរាល់ Node នីមួយៗរបស់វាត្រូវបានឆ្លងកាត់ ឬជួប (visit) បីលើកជាបន្តបន្ទាប់។ ដូចនេះគេត្រូវប្រើនូវវិធាន Traversing Binary Trees បី យ៉ាងផ្សេងគ្នាតាមលំនាំដំណើរការដូចខាងក្រោម៖

1-Preorder Traversal (NLR)

- + Visit Node Root
- + Preorder Left Subtrees Traversal
- + Preorder Right Subtrees Traversal

ឧទាហរណ៍៖

គេមានទម្រង់ Binary Trees គឺ



ចូរសរសេរលំដាប់ទិន្នន័យរបស់ Node នីមួយៗក្នុង Binary Trees នេះតាមវិធាន

Preorder Traversal ។

គេបាន៖

លំដាប់ទិន្នន័យរបស់ Node នីមួយៗនៅក្នុង Binary Trees នេះតាមវិធាន Preorder Traversal គឺ

(រូប a) A BDE CFG

(រូប b) A BDEGF CHIJK

2-Inorder Traversal (LNR)

-Inorder Left Subtrees Traversal

-Visit Node Root

-Inorder Right Subtrees Traversal

ឧទាហរណ៍៖

គេអាចសរសេរលំដាប់ទិន្នន័យរបស់ Node នីមួយៗក្នុង Binary Trees រូប (a) និងរូប(b) តាម Inorder Traversal គឺ

(រូប a) DBE A FCG

(រូប b) BEGDF A HCJKI

3-Postorder Traversal (LRN)

-Postorder Left Subtrees Traversal

-Postorder Right Subtrees Traversal

-Visit Node Root

ឧទាហរណ៍

គេអាចសរសេរលំដាប់ទិន្នន័យរបស់ Node នីមួយៗក្នុង Binary Trees រូប (a) និងរូប (b) តាម Postorder Traversal គឺ

(រូប a) DEB FGC A

(រូប b) GEFDB HKJIC A

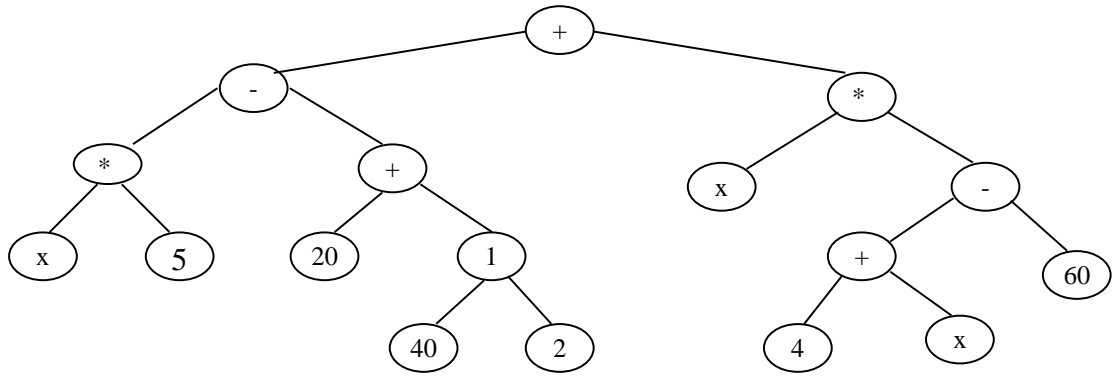
ឧទាហរណ៍៖

គេមានទម្រង់ Binary Trees របស់កន្សោមពិជគណិតគឺ

+ ab : prefix

a + b : infix

ab + : postfix



ចូរសរសេរលំដាប់ទិន្នន័យរបស់ Node នីមួយៗក្នុង Trees តាមវិធាន Traversing Binary និងប្រាប់ឈ្មោះរបស់កន្សោមពិជគណិតទាំងនោះ។

6-2-5 ប្រមាណវិធីលើ **Binary Trees**

គេមានប្រមាណវិធីមួយចំនួនរបស់ Binary Trees ត្រូវបានកំណត់ប្រើដូចខាងក្រោម៖

- * proot គឺជា pointer សម្រាប់ចង្អុលទៅកាន់ Node Root ។
- * p គឺជា pointer សម្រាប់ចង្អុលទៅកាន់ Node ណាមួយ។
- * visit(p) សម្រាប់ដំណើរការលើ Node នីមួយៗក្នុង Binary Trees ។

* របៀបបង្កើត Data Structures

គឺជាការប្រកាសទម្រង់ Node នីមួយៗក្នុង Binary Trees តាមលំនាំ struct ដែលរួមមានបី field (info, left, Right) ។

```
struct nodetype
{
    datatype info;
    struct nodetype *left;
    struct nodetype *Right;
}*node;

typedef struct nodetype *pointertype;
```

***Getnode**

អនុគមន៍សម្រាប់បញ្ជូន Node អោយ Binary Trees។

```
pointertype Getnode( )
{
    pointertype p;
    p = (struct nodetype *)malloc(sizeof(struct nodetype));
    return( p );
}
```

***Freenode**

```
void Freenode( pointertype p)
{
    Free( p );
}
```

*** initialize**

អនុគមន៍សម្រាប់បង្កើត Trees ទំនេរគឺ

```
void initialize ( pointer type * proot )
{
    proot = NULL;
}
```

*** make node**

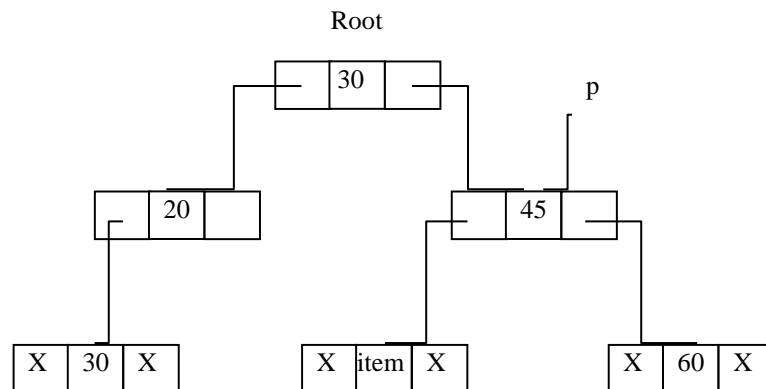
អនុគមន៍សម្រាប់បង្កើត Node[p] មានទិន្នន័យ item និង field Left និង field Right ស្មើ NULL

```
pointertype make_node (datatype item)
{
    pointertype p = Getnode( );
    p → info = item ;
    p → left = NULL ;
    p → Right = NULL ;
    return ( p );
}
```

*** make left**

អនុគមន៍សម្រាប់បង្កើត Left Son Node របស់ Node[p]

```
void make_left( pointertype p, datatype item )
{
    if( p == NULL )
        printf( " Node p is NULL " );
    else
    {
        if( p -> left != NULL )
            printf( " Node p has left son node " );
        else
            p -> left = make_node( item );
    }
}
```



*** make right**

អនុគមន៍សម្រាប់បង្កើត Right Son Node របស់ Node[p]

```
void make_right( pointertype p, datatype item )
{
    if( p == NULL )
        printf( " Node p is NULL " );
    else
    {
        if( p -> right != NULL )
            printf( " Node p has right son node " );
        else
            p -> right = make_node( item );
    }
}
```


* Builtrees

សម្រាប់សង់ Binary Trees ដែលមាន n Nodes ឲ្យមាន Height រឺ Depth ទាបបំផុត

```
pointertype builtrees(int n)
{
    datatype item ;
    int nl , nr ,
    pointertype p ;
    if (n == 0 )
        return (NULL) ;
    else
    {
        printf ( “ info of node ” ) ;
        scanf ( “ %....” , & item) ;


$$nL = \frac{n}{2} ;$$

        nr = n - nl - 1 ;
        p = make_node( item);
        p → Left = builtrees( nl ) ;
        p → Right = buildtrees(nr )
        return( p ) ;
    }
}
```

* Traversing Binary

សម្រាប់បង្ហាញចេញនូវទិន្នន័យរបស់ Node នីមួយៗតាមលំនាំដំណើរការ Recursive របស់ Preorder, Inorder , និង Postorder Traversal ។

*Algorithms: Preorder(NLR)

```
void preorder (pointertype *proot)
{
    if ( proot != NULL )
    {
        printf (“%....”, proot -> info)
        /* visit( proot ) ; */
        preorder ( proot → Left ) ;
        preorder ( proot → Right) ;
    }
}
```

*** Algorithms: Inorder (LNR)**

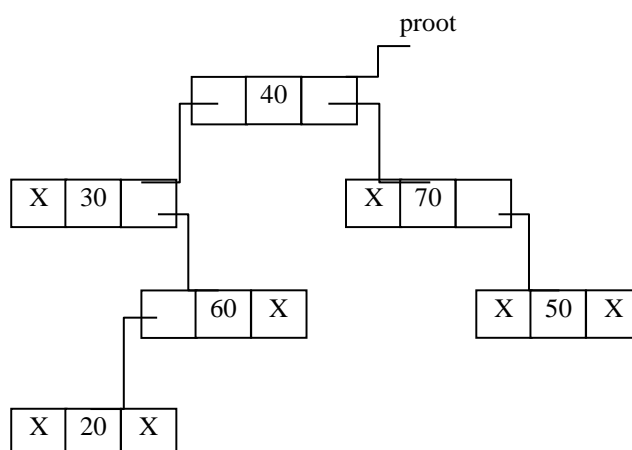
```
void inorder( pointertype * proot)
{
    if (proot != NULL)
    {
        inorder(proot → Left) ;
        /* visit (proot) ; */
        printf(“ % ...” , proot → info);
        Inorder( proot -> Right )
    }
}
```

*** Algorithms: postorder (LRN)**

```
void postorder( pointertype * proot )
{
    if( proot != NULL )
    {
        postorder(proot → Left ) ;
        postorder(proot → Right) ;
        printf(" %...", proot → info ) ;
        /* visit( proot ) ; */
    }
}
```

*** clear trees**

អនុគមន៍សម្រាប់លុប Node ទាំងអស់នៅក្នុង Binary Trees



```
void clear_trees (pointer type * proot)
{
    pointer type p;
    p = proot ;
    if ( p!= NULL )
    {
        clear_trees (p → Left ) ;
        clear_trees ( p → Right ) ;
        Free node (p) ;
    }
    proot = NULL ;
}
```

*** Search**

អនុគមន៍សម្រាប់ស្វែងរកធាតុ item ណាមួយនៅក្នុង Binary Trees ។

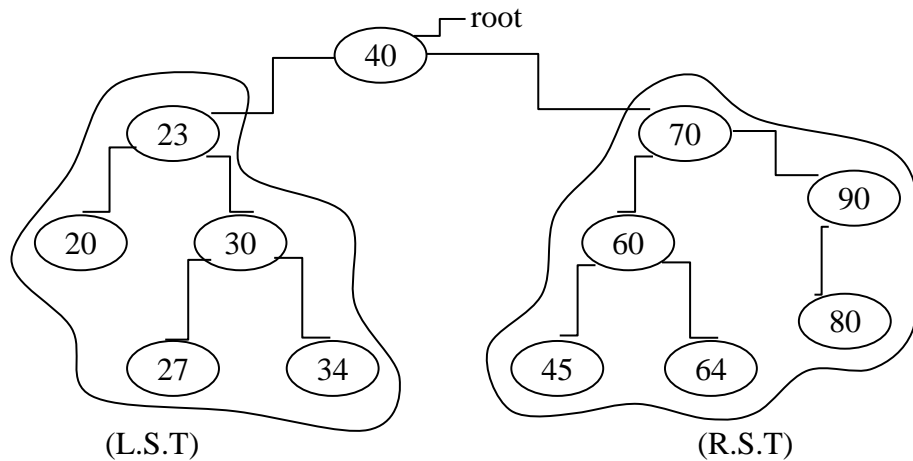
```
pointertype search (pointertype * proot , datatype item)
{
    pointer type p ;
    if (proot == NULL) return ( NULL ) ;
    if (proot → info == item ) return (proot) ;
    p = search (proot → Left , item) ;
    if (p == NULL )
    p = search (proot → Right , item) ;
    return ( p ) ;
}
```

3-6 Binary Search Trees

6-3-1 និយមន័យ

Binary Search Trees គឺជា Trees ដែលរាល់ Node នីមួយៗរបស់វាត្រូវបានផ្ទុកតាមលំនាំ **Left Subtrees < Node root < Right Subtrees** មានន័យថារាល់ទិន្នន័យរបស់ Node នីមួយៗនៅក្នុង Left Subtrees ត្រូវតូចជាងទិន្នន័យរបស់ Node Root ហើយ Node Root មានទិន្នន័យតូចជាងទិន្នន័យរបស់ Node នីមួយៗនៅក្នុង Right Subtrees ។

គេមាន Structure របស់ Binary Search Trees ដូចខាងក្រោម៖

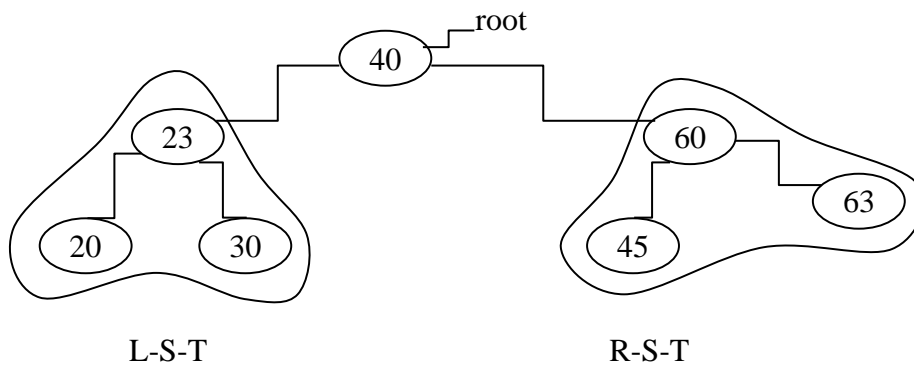


6-3-2 ប្រមាណវិធីលើ **Binary Search Trees**

ប្រមាណវិធីលើ Binary Search Trees មានលក្ខណៈដូចប្រមាណវិធីរបស់ Binary Trees ដែរ ប៉ុន្តែវាមានប្រមាណវិធីមួយចំនួនដែលមានលក្ខណៈដំណើរការតាមគោលការណ៍ជាមូលដ្ឋានរបស់ Binary Search Trees ដូចខាងក្រោម៖

1-Algorithms: Search Trees

សម្រាប់ស្វែងរកធាតុ item ណាមួយនៅក្នុង List ដែលមានលំនាំជា Binary Search Trees គឺ



Search item = 45

```
pointertype searchtrees(pointertype * proot , datatype item)
```

```
{
    pointertype p ;
    if(proot != NULL)
    {
        if(item == proot -> info)
```

```

        return( p ) ;
    else if( item < proot → info)
        p = searchtrees(proot → Left , item);
    else
        p = searchtrees(proot → Right, item) ;
    }
    return(NULL)
}

```

គេអាចសរសេរ Algorithms នេះដោយមិនប្រើ Recursive ដូចខាងក្រោម៖

```

pointertype searchtrees(pointertype * proot, datatype item)
{
    pointertype p ;
    p = proot ;
    while( p != NULL)
    {
        if(item == p → info)
            return(p) ;
        else if(item < p → info)
            p = p → Left
        else
            p = p → Right ;
    }
    return (NULL) ;
}

```

2-Algorithms: Insert

សម្រាប់បន្ថែម Node[p] ដែលមានទិន្នន័យ item ចូលទៅក្នុង Binary Search Trees ដែលក្រោយពីការបន្ថែម Node[p] ឲ្យវានៅតែមានលក្ខណៈជា Binary Search Treesដដែល។

```

void insert(pointertype * proot , datatype item)
{
    pointertype p ;
    if(proot == NULL)
    {

```

```

        p = Getnode ( ) ;
        p → info = item ;
        p → Left = NULL ;
        p → Right = NULL ;
        proot = p ;
    }
    else
    {
        if(item == proot → info)
            printf (“node p has in binary search trees”) ;
        else if(item < proot → info)
            insert(proot → Left , item) ;
        else
            insert(proot → Right , item) ;
    }
}

```

ឧទាហរណ៍១៖

អនុគមន៍សម្រាប់បញ្ចូល item ទៅក្នុងលំដាប់នៃ Binary Tree (with duplicate elements) ។

```

NODE insert(int item, NODE root)
{
    NODE temp, cur, prev;
    temp = getnode( );    /* Obtain new node from the availability list */
    temp -> info = item ; /* Copy appropriate data */
    temp -> Llink = NULL;
    temp -> Rlink = NULL;
    if(root == NULL)
        return( temp); /* Insert a node for the first time */
    /* find the position to insert */
    prev = NULL;
    cur = root;
    while(cur != NULL)

```

```

{
    /* Obtain parent and appropriate left or right child */
    prev = cur;
    cur = (item < cur -> info)? cur -> Llink : cur -> Rlink;
}
if(item < prev -> info)      /* if node to be inserted is less than parent */
    prev -> Llink = temp;      /* Insert towards left of the parent */
else
    prev -> Rlink = temp; /* otherwise, insert towards right of the parent */
return(root);                /* Return the root of the tree */
}

```

ឧទាហរណ៍២

អនុគមន៍ដើម្បីបញ្ចូល item ទៅក្នុងលំដាប់នៃ Binary Tree (No duplicate items are allowed)

```

NODE insert(int item, NODE root)
{
    NODE temp, cur, prev;
    temp = getnode( );    /* Obtain new node from the availability list */
    temp -> info = item; /* Copy appropriate data */
    temp -> Llink = NULL;
    temp -> Rlink = NULL;
    if(root == NULL)
        return(temp); /* Insert a node for the first time */
    /* find the position to insert */
    prev = NULL;
    cur = root;
    while(cur != NULL)
    {
        /* Obtain parent */
        prev = cur;
        /* do not insert duplicate item */
        if( item == cur -> info)

```

```

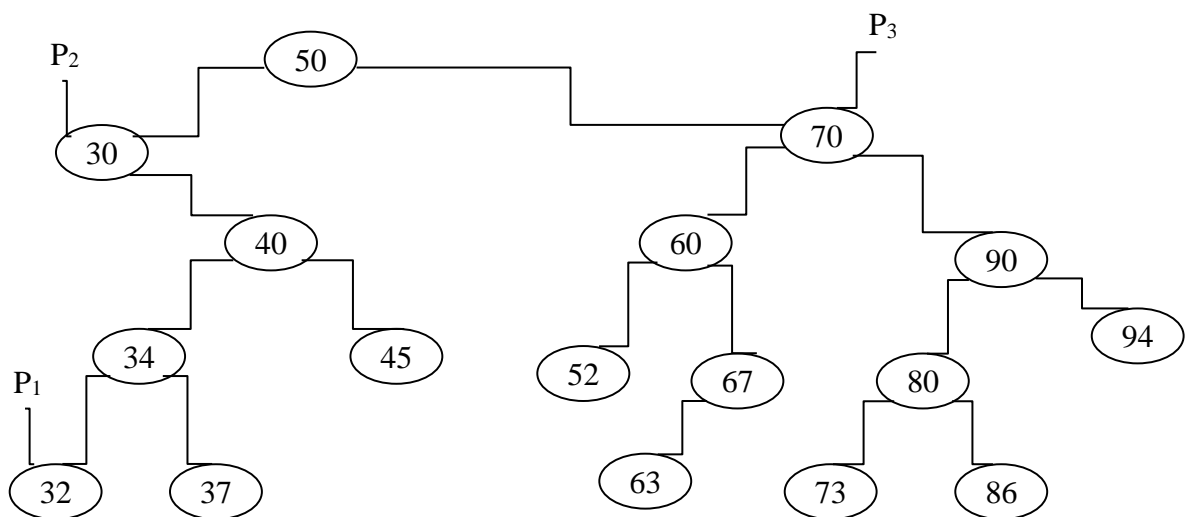
{
    printf(" Duplicate items not allowed \n");
    freenode(temp);
    return( root );
}
/* Find the appropriate left or right child */
cur = (item < cur -> info)? cur -> Llink : cur -> Rlink;
}
if(item < prev -> info)      /* If node to be inserted is less than parent */
    prev -> Llink = temp; /* Insert towards left of the parent */
else
    prev -> Rlink = temp; /* otherwise, insert towards right of the parent */
return(root);               /* Return the root of the tree */
}

```

3-Algorithms: Delete node

សម្រាប់លុប Node[p] ចេញពី Binary Search Trees ដែលក្រោយពីការលុប Node[p] ឲ្យវានៅតែមានលក្ខណៈជា Binary Search Trees ដដែល។

គេមានបីករណីដែលអាចកើតឡើងនៅពេលលុប Node[p] គឺ



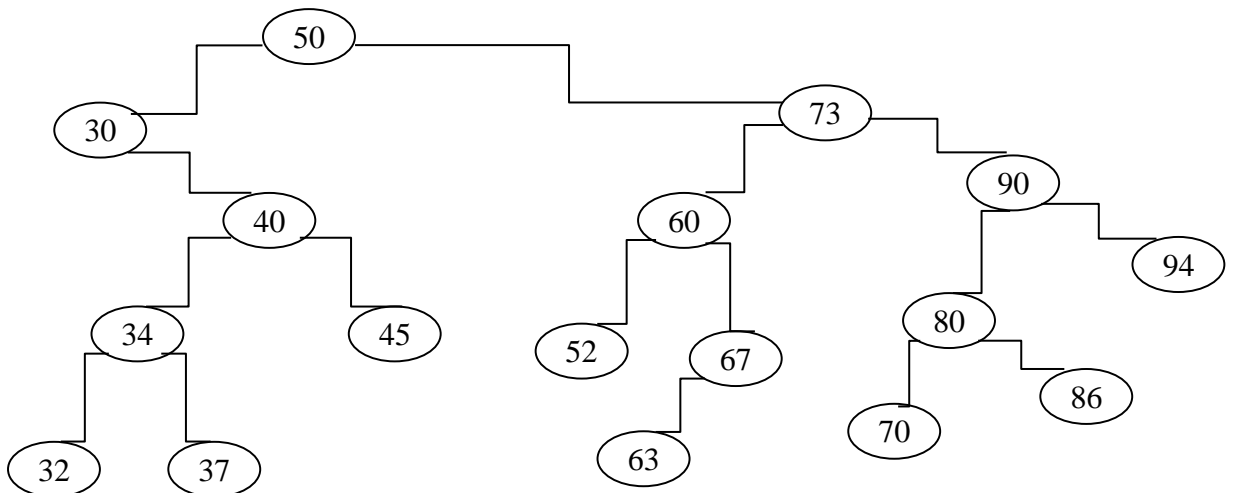
first Left son node of right subtrees

1-ករណី Node[p] មាន degree ស្មើ ១ (Node[p₁]) គឺជ្រើសរើស stand for node សម្រាប់ Node[p] ស្មើ NULL រួចអនុវត្តការលុប Node[p] ។

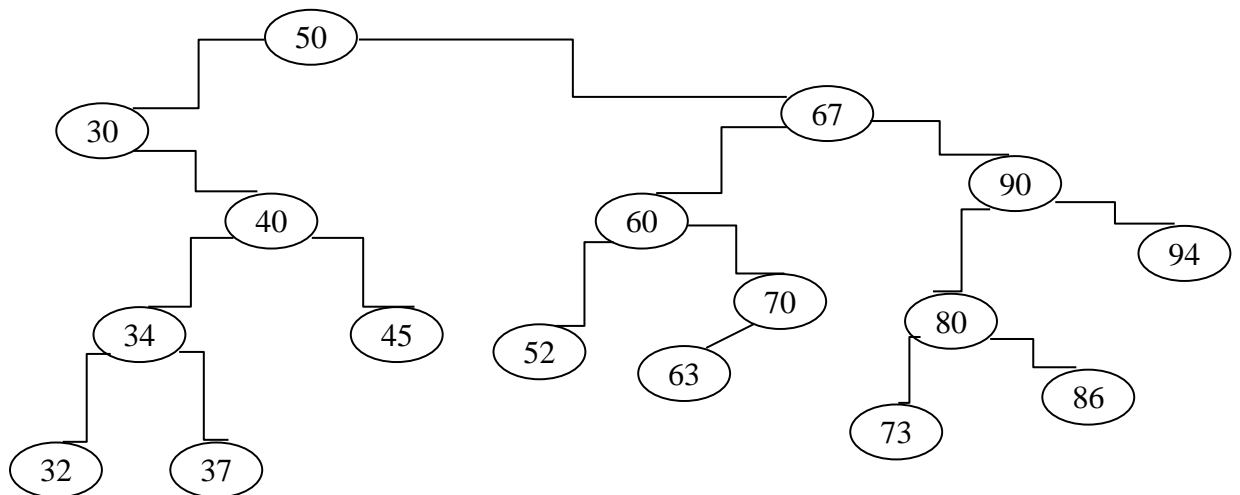
2-ករណី Node[p] មាន degree ស្មើ 1 (Node[p₂]) គឺជ្រើសរើស stand for node សម្រាប់ Node[p] ជា son node របស់ Node[p] បន្ទាប់មកតភ្ជាប់ stand for node ជាមួយ Father Node របស់ Node[p] រួចអនុវត្តការលុប Node[p] ។

3-ករណី Node[p] មាន degree ស្មើ 2 (Node[p₃]) គឺជ្រើសរើស stand for node សម្រាប់ Node[p] អាចជា first left son node of Right subtrees ឬ last right son node of left subtrees របស់ Node[p] (មានន័យថា Node ដែលមានតម្លៃទិន្នន័យតូចបំផុតក្នុង Right Subtrees ឬ Node ដែលមានតម្លៃធំបំផុតក្នុង left subtrees របស់ Node[p]) ។ បន្ទាប់មកផ្លាស់ប្តូរ stand for node ជាមួយ Node[p] រួចអនុវត្តការលុបដូចករណី 1 ឬករណី 2 ។

* First left son node of right subtrees (node 73)



* Last right son node of left subtrees (node 67)



គេមានបីករណីអាចកើតឡើងនៅពេលលុប Node[p] និងមានដំណើរការអនុវត្តជាបន្តបន្ទាប់ដូចខាងក្រោម៖

+ជ្រើសរើស stand for node អោយ Node[p] ប្រសិនបើ

-Node[p] មាន degree ស្មើសូន្យនោះ stand for node ស្មើ NULL ។

-Node[p] មាន degree ស្មើ 1 នោះ stand for node គឺជា son node of Node[p] ។

-Node[p] មាន degree ស្មើ 2 នោះ stand for node គឺជា first left son node of right subtrees or last right son node of left subtrees ។

+Stand for Node និងជំនួសឱ្យ Node[p] មានន័យថា៖

-Left Subtrees និង Right Subtrees របស់ Node[p] ក្លាយជា Left Subtrees និង Right Subtrees of stand for node ។

-អនុវត្តការតភ្ជាប់រវាង Father Node របស់ Node[p] ជាមួយ stand for node ។

-អនុវត្តការលុប Node[p] ។

+បណ្តា Node ក្រៅពី Subtrees មាន Node Root ជា Node[p] ត្រូវប្រែប្រួល រីឯបណ្តា Node ដទៃក្រៅពី Subtrees នោះមិនមានការប្រែប្រួលទេ ។

+ក្រោយពេលលុប Node[p] គឺ stand for node និងក្លាយជា Node Root ថ្មីរបស់ Subtrees នោះ ។

```
pointertype Delete( pointertype p)
{
    pointertype px, f ;    /* px is stand for node */
                          /* f is father node */

    if( p == NULL )
        return( NULL);
    else
    {
        if( p -> left == NULL && p -> right == NULL )
            px = NULL ;
        else if( p -> right == NULL )
            px = p -> left ;
        else if( p -> left == NULL ) ;
            px = p -> right ;
```

```

        else
        {
            f = p ;
            px = p -> right;
            while( px -> left != NULL )
            {
                f = px; px = px -> left ;
            }
            if(f != p)
            {
                f -> left = px -> right ;
                px -> right = p -> right ;
                px -> left = p -> left;
            }
            else
                px -> left = p -> left ;
        }
        freenode( p );
        return(px) ;
    }
}

```

ឧទាហរណ៍១៖

ចូរសរសេរកម្មវិធីដើម្បីបង្កើត Trees ដែលជា Traverse Trees និងលុប item ចេញពី Trees ។

```

#include< stdio.h >
#include< alloc.h >
#include< process.h >
#include< string.h >
struct node
{
    int info;
    struct node *Llink;
    struct node *Rlink;
};
typedef struct node * NODE;
/* Function to delete an item from the tree if it exists using above function example */
void main( )

```

```
{
    NODE root, temp;
    int choice, item;
    root = NULL;
    for( ; ; )
    {
        printf(" 1 : delete 4 : Exit \n");
        printf(" Enter the choice \n ");
        scanf(" %d",&choice);
        switch(choice)
        {
            case 1:
                printf(" Enter the item to be deleted \n");
                scanf(" %d", &item);
                root = delete_item(item, root);
                break;
            default:
                exit(0);
        }
    }
}
```

ឧទាហរណ៍២៖

ចូរសរសេរកម្មវិធីដើម្បីបង្កើត Trees និង Copy Trees

```
# include< stdio.h >
# include< malloc.h >
/* Tree structure */
typedef struct TREE
{
    int data;
    struct TREE *left;
    struct TREE *right;
}TREE;

/* Insert data into Tree */
TREE *InsertTree(data, p)
int data;
TREE *p;
{
    /* Is Tree NULL */
    if(!p)
    {
        p = (TREE*)malloc(sizeof(TREE));
        p->data = data;
        p->left = NULL;
        p->right = NULL;
        return(p);
    }
}
```

```

/* Is data is less than the parent element */
if(data < p -> data)
    p -> left = InsertTree(data, p -> left)
else
    /* Is data is greater than the parent element */
    if(data > p -> data)
        p -> right = InsertTree(data, p -> right)
    return(p);
}
/* Print Tree */
void PrintTree( tree, level)
TREE *tree;
int level;
{
    int i;
    if(tree)
    {
        PrintTree(tree -> right, level + 1);
        printf("\n");

        for(i = 0; i < level; i++)
            printf("  ");

        printf("%d", tree -> data);

        PrintTree(tree -> left, level + 1);
    }
}
/* Function to copy the original tree 'tree' into 'tdup'
Note: 'tdup' is passed as pointer to pointer */
void CopyTree(tree, tdup)
TREE *tree, **tdup;
{
    int i;
    if(tree)
    {
        *tdup = (TREE*)malloc(sizeof(TREE));
        /* '*tdup' is pointer value */
        (*tdup) -> data = tree -> data;
        /* the pointer value '(*tdup) -> left' is by reference */
        CopyTree(tree -> left, &(*tdup) -> left);
        /* the pointer value '(*tdup) -> right' is by reference */
        CopyTree(tree -> right, &(*tdup) -> right);
    }
    else
        *tdup = NULL;
}

```

```

void main( )
{
    int data;
    TREE *tree = NULL;
    TREE *tdup;

    printf("\nTree-Insert and Copy Operations:\n");
    while(1)
    {
        printf("\nKey to Insert <0>?");
        scanf("%d", &data);
        printf("%d\n", data);

        if(data == 0)
            break;

        tree = InsertTree(data, tree);
        printf("\nTree Display:\n");
        PrintTree(tree, 1);
    }
    CopyTree(tree, &tdup);
    printf("\nCopy of Original Tree:\n");
    PrintTree(tdup, 1);
}

```

នៅពេល RUN កម្មវិធីបង្ហាញលទ្ធផលដូចខាងក្រោម៖

```

Tree-Insert and Copy Operations:
Key to Insert <0>? 6 /*បញ្ចូលលេខ6*/
Tree Display:
6
Key to Insert <0>? 4 /*បញ្ចូលលេខ4*/
Tree Display:
6
    4
Key to Insert <0>? 8 /*បញ្ចូលលេខ8*/
Tree Display:
8
6
    4
Key to Insert <0>? 3 /*បញ្ចូលលេខ3*/
Tree Display:
8
6
    4
        3

```

Key to Insert <0>? 7 /*បញ្ចូលលេខ7*/

Tree Display:

```
      8
     / \
    6   7
   / \
  4   3
```

Key to Insert <0>? 5 /*បញ្ចូលលេខ5*/

Tree Display:

```
      8
     / \
    6   7
   / \
  4   5
   \
    3
```

Key to Insert <0>? 9 /*បញ្ចូលលេខ9*/

Tree Display:

```
      9
     / \
    8   7
   / \
  6   5
   \
    4
   \
    3
```

Key to Insert <0>? -3 /*បញ្ចូលលេខ-3*/

Tree Display:

```
      9
     / \
    8   7
   / \
  6   5
   \
    4
   \
    3
   \
  -3
```

Key to Insert <0>? 11 /*បញ្ចូលលេខ11*/

Tree Display:

```

          11
        9
      8
    6
  4
  3
-3
    
```

Key to Insert <0>? 0 /*បញ្ចូលលេខ0*/

Copy of Original Tree:

```

          11
        9
      8
    6
  4
  3
-3
    
```

ឧទាហរណ៍៣៖

ចូរសរសេរកម្មវិធីដើម្បីជ្រើសរើស Binary Search Trees និងបង្ហាញជា Traversal

```

#include< stdio.h >
#include< malloc.h >
/* TREE structure */
typedef struct TREE
{
    struct TREE *left;
    int data;
    struct TREE *right;
}TREE;

TREE *BinaryTree(nArray, low, high)
int *nArray, low, high;
{
    TREE *tree;
    int mid;

    mid = (int)(low + high)/2;
    if(!(tree = (TREE*)malloc(sizeof(TREE))))
    {
        printf("Error: Out of Memory.!");
        return(tree);
    }
}
    
```



```

tree -> data = nArray[mid];
/* Table lower ptr is equal to higher ptr then no further division */
if(low >= high)
{
    tree -> left = NULL;
    tre -> right = NULL;
    return(tree);
}
/* Divide and Conquer Method: i.e. Divide the table into two equal
halves until no further division is possible */

if(low <= mid - 1)
    tree -> left = BinaryTree(nArray, low, mid - 1);
else
    tree -> left = NULL;

if(mid + 1 <= high)
    tree -> right = BinaryTree(aArray, mid + 1, high);
else
    tree -> right = NULL;
return(tree);
}
/* Pre order Traversal */
void PreOrder(tree)
TREE *tree;
{
    if(tree)
    {
        printf("%d", tree -> data);    /* Process node */
        PreOrder(tree -> left);
        PreOrder(tree -> right);
    }
}
/* In Order Traversal */
void inOrder(tree)
TREE *tree;
{
    if(tree)
    {
        InOrder(tree -> left);
        printf("%d", tree -> data);    /* Process node */
        InOrder(tree -> right);
    }
}
/* Post Order Traversal */
void PostOrder(tree)
TREE *tree;
{

```

```

        if(tree)
        {
            PostOrder(tree -> left);
            PostOrder(tree -> right);
            printf("%d", tree -> data);    /* Process node */
        }
    }
    /* Tree Printing in Triangle Form */
    void PrintTreeTriangle(tree, level)
    TREE *tree;
    int level;
    {
        int i;
        if(tree)
        {
            PrintTreeTriangle(tree -> right, level + 1);
            printf("\n");
            for(i = 1; i < level; i++)
                printf(" ");

            printf("%d", tree -> data);
            PrintTreeTriangle(tree -> left, level + 1);
        }
    }
    /* Tree Printing in Diagonal Form */
    void PrintTreeDiagonal(tree, level)
    TREE *tree;
    int level;
    {
        int i;
        if(tree)
        {
            printf("\n");
            for(i = 0; i < level; i++)
                printf(" ");

            printf("%d", tree -> data);
            PrintTreeDiagonal(tree -> left, level + 1);
            PrintTreeDiagonal(tree -> right, level + 1);
        }
    }
    void main( )
    {
        int array[20], i, ndata, disptype;
        TREE *btree;

        printf("This Program Demonstrates the Binary Tree Creation\n");
        printf("and Traversal for Binary Search:\n\n");
    }

```

```

printf("Enter Number of Elements in Data list:");
scanf("%d", &ndata);
printf("%d\n", ndata);
printf("\nEnter Data in Ascending Order;\n");
for(i = 0; i < ndata; i++)
{
    printf("Data[%d]? = ", i);
    scanf("%d", &array[i]);
    printf("%d\n", array[i]);
}
btree = BinaryTree(array, 0, ndata - 1);

printf("Binary Tree Creation Done !.\n");
printf("\nTree Display Style:[1]-Triangular [2]-Diagonal form:");
scanf("%d", &disptype);
printf("%d\n", disptype);

printf("\nBinary Tree is :\n");
if(disptype == 1)
    PrintTreeTriangle(btree, 1);
else
    PrintTreeDiagonal(btree, 1);

printf("\nPr-Order Traversal:");
PreOrder(btree);

printf("\nIn-Order Traversal:");
InOrder(btree);

printf("\nPost-Order Traversal:");
PostOrder(btree);
printf("\n");
}

```

នៅពេល **RUN** កម្មវិធីបង្ហាញលទ្ធផលដូចខាងក្រោម៖

This Program Demonstrates the Binary Tree Creation
and Traversal for Binary Search:

Enter Number of Elements in Data List: 7

Enter Data in Ascending Order:

Data[0]? = 1

Data[1]? = 2

Data[2]? = 3

Data[3]? = 4

Data[4]? = 5

Data[5]? = 6

Data[6]? = 7

Binary Tree Creation Done!.

Tree Display Style: [1]-Triangular [2]-Diagonal form: 1
Binary Tree is :

```

          7
        6
       5
      4
     3
    2
   1
Pre-Order Traversal: 4      2      1      3      6      5      7
In-Order Traversal:  1      2      3      4      5      6      7
Post-Order Traversal: 1      3      2      5      7      6      4
    
```

នៅពេល **RUN** កម្មវិធីបង្ហាញលទ្ធផលដូចខាងក្រោម៖

This Program Demonstrates the Binary Tree Creation
and Traversal for Binary Search:

Enter Number of Elements in Data List: 7

Enter Data in Ascending Order:

Data[0]? = 1

Data[1]? = 2

Data[2]? = 3

Data[3]? = 4

Data[4]? = 5

Data[5]? = 6

Data[6]? = 7

Binary Tree Creation Done!.

Tree Display Style: [1]-Triangular [2]-Diagonal form: 2
Binary Tree is :

```

      4
     2
    1
   3
  6
 5
7
Pre-Order Traversal: 4      2      1      3      6      5      7
In-Order Traversal:  1      2      3      4      5      6      7
Post-Order Traversal: 1      3      2      5      7      6      4
    
```

ឧទាហរណ៍៤៖

ចូរសរសេរកម្មវិធីដើម្បីបញ្ចូល item និងលុប item នៅក្នុង Trees

```
# include< stdio.h >
# include< malloc.h>
# define TRUE 1
# define FALSE 0

/* Tree structure */
typedef struct TREE
{
    int data;
    struct TREE *left;
    struct TREE *right;
}TREE;

/* Insert data into Tree */
TREE *InsertTree(data, p)
int data;
TREE *p;
{
    /* Is Tree NULL */
    if(!p)
    {
        p = (TREE*)malloc(sizeof(TREE));
        p -> data = data;
        p -> left = NULL;
        p -> right = NULL;
        return(p);
    }
    /* Is data less than the parent element */
    if(data < p -> data)
        p -> left = InsertTree(data, p -> left);
    else
        /* Is data greater than the parent element */
        if(data > p -> data)
            p -> right = InsertTree(data, p -> right);
    return(p);
}

/* Print Tree */
void PrintTree(tree, level)
TREE *tree;
int level;
{
    int i;
    if(tree)
    {
```

```

        PrintTree(tree -> right, level + 1);
        printf("\n");

        for(i = 0; i < level; i++)
            printf("  ");
        printf("%d", tree -> data);
        PrintTree(tree -> left, level + 1)
    }
}
/* Computes Tree Depth, Before Calling This function
Initialize 'depth' to ZERO */

void TreeDepth(tree, depth, level)
TREE *tree;
int *depth;
int level;
{
    if(tree)
    {
        if(level > *depth)
            *depth = level;
        TreeDepth(tree -> left, depth, level + 1);
        TreeDepth(tree -> right, depth, level + 1);
    }
}
/* Replaces the Node at which key is found with last right key
of a left child */
TREE *Del(r, q)
TREE *r, *q;
{
    TREE *dnode;

    if( r -> right != NULL)
        r -> right = Del(r -> right, q);
    else
    {
        dnode = r;
        q -> data = r -> data;
        r = r -> left;
        free(dnode);
    }
    return( r );
}
/* Deletes the key from the tree */
TREE *DeleteElement(p, data)
TREE *p;
int data;
{

```

```

TREE *q;
if(!p)
{
    printf("\nNon Existent Data\n");
    return(p);
}
else
{
    if(data < p -> data)
        p -> left = DeleteElement(p -> left, data);
    else
        if(data > p -> data)
            p -> right = DeleteElement(p -> right, data);
        else
        {
            q = p;
            if(q -> right == NULL)
            {
                p = q -> left;
                free(q);
            }
            else
                if(q -> left == NULL)
                {
                    p = q -> right;
                    free(q);
                }
            else
                q -> left = Del(q -> left, q);
        }
    }
    return(p);
}

void main( )
{
    int data, depth;
    TREE *tree = NULL;
    printf("\nTree-Insert and Delete Operations:\n");

    while(1)
    {
        printf("\nKey to Insert <0>?");
        scanf("%d", &data);
        printf("%d\n", data);
        if(data == 0)
            break;
        tree = InsertTree(data, tree);
        printf("\nTree Display:\n");
    }
}

```

```

        PrintTree(tree, 1);
        depth = 0;
        TreeDepth(tree, &depth, 0);
        printf("\nTree Depth = %d\n", depth);
    }
    while(1)
    {
        printf("\nKey to Delete <0>?");
        scanf("%d", &data);
        printf("%d\n", data);
        if(data == 0)
            break;
        tree = DeleteElement(tree, data);
        printf("\nTree Display:\n");
        PrintTree(tree, 1);
        depth = 0;
        TreeDepth(tree, &depth, 0);
        printf("\nTree Depth = %d\n", depth);
    }
}

```

នៅពេល **RUN** កម្មវិធីបង្ហាញលទ្ធផលដូចខាងក្រោម៖

Tree-Insert and Delete Operations:

Key to insert <0>? 25 /*បញ្ចូលលេខ25*/

Tree Display:

25

Tree Depth = 0

Key to Insert <0>? 20 /*បញ្ចូលលេខ20*/

Tree Display:

25

20

Tree Depth = 1

Key to Insert <0>? 30 /*បញ្ចូលលេខ30*/

Tree Display:

30

25

20

Tree Depth = 1

Key to Insert <0>? 22 /*បញ្ចូលលេខ22*/

Tree Display:


```

        30
      25
        22
      20
Tree Depth = 2
Key to Insert <0>? 35 /*បញ្ចូលលេខ35*/
Tree Display:

```

```

        35
      30
        22
      25
        20
Tree Depth = 2
Key to Insert <0>? 15 /*បញ្ចូលលេខ15*/
Tree Display:

```

```

        35
      30
        22
      25
        20
          15
Tree Depth = 2
Key to Insert <0>? 10 /*បញ្ចូលលេខ10*/
Tree Display:

```

```

        35
      30
        22
      25
        20
          15
            10
Tree Depth = 3
Key to Insert <0>? 8 /*បញ្ចូលលេខ8*/
Tree Display:

```

```

        35
      30
        22
      25
        20
          15
            10
              8

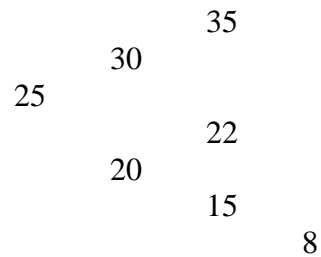
```

Tree Depth = 4

Key to Insert <0>? 0 /*បញ្ចូលលេខ0*/

Key to Delete <0>? 10 /*លុបលេខ10*/

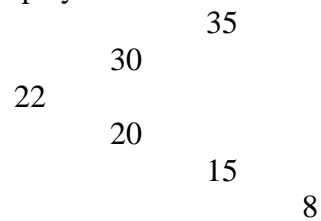
Tree Display:



Tree Depth = 3

Key to Delete <0>? 25 /*លុបលេខ25*/

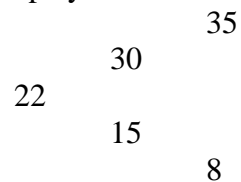
Tree display:



Tree Depth = 3

Key to Delete <0>? 20 /*លុបលេខ20*/

Tree Display:



Tree Depth = 2

Key to Delete <0>? 0 /*លុបលេខ0*/

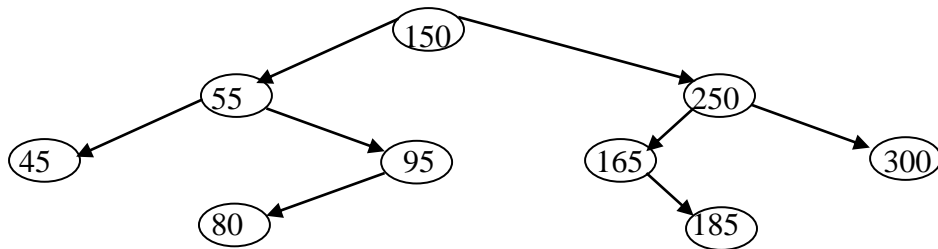
លំហាត់អនុវត្ត

- 1-Define Tree? Explain Binary trees with its properties.
- 2-Explain the storage representation of Binary tree with sequential representation of the tree.
- 3-Explain the types of algorithms for tree traversal.
- 4-Write a note on Binary Search Tree (BST)
- 5-Explain the two types of cases to delete node from tree.
- 6-គេមានទិន្នន័យជាប្រាក់ខែរបស់បុគ្គលិកម្នាក់ៗនៃក្រុមហ៊ុនមួយ ត្រូវបានផ្ទុកជា Array

ដូចខាងក្រោម៖

40	30	60	70	45		23			90	20			34	50
----	----	----	----	----	--	----	--	--	----	----	--	--	----	----

- (ក)-ចូរសង់ Binary Tree តាងឱ្យ Array ខាងលើ។
 - (ខ)-ចូរបង្ហាញពីលំដាប់ទិន្នន័យតាមវិធាន Preorder, Inorder និង Postorder Traversal ។
 - (គ)-ចូរសរសេរ Algorithm (C/C++ code) សម្រាប់ Update លើប្រាក់ខែរបស់បុគ្គលិកម្នាក់ៗចំនួន 15% នៃប្រាក់ខែដើម។
- 7-គេមាន Binary Search Tree ដូចខាងក្រោម៖



- (ក)-ចូរផ្ទុក Binary Search Trees នេះជា Array ។
 - (ខ)-ចូរសរសេរលំដាប់ item នៃ Node តាម Inorder, Preorder និង Postorder Traversal ។
 - (គ)-ចូរសង់ Binary Search Tree ខាងលើឡើងវិញនៅពេលគេបន្ថែម item មួយទៀតគឺ 155 ។
- 8-គេមាន List មួយជា Binary Tree ដែល Node នីមួយៗផ្ទុកចំនួនប្រាក់ខែរបស់កម្មករម្នាក់ៗនៃក្រុមហ៊ុនមួយ ។ ចូរសរសេរ Algorithm (C code) ដើម្បីតំឡើងប្រាក់ខែចំនួន 25% ទៅដល់កម្មករម្នាក់ៗ។

9-គេមានទិន្នន័យដូចខាងក្រោម៖

25 12 30 10 5 35 6 40 8 21

- (a)-ចូរសង់ Binary Search Tree ទៅតាមទិន្នន័យខាងលើ។
- (b)-ចូរសរសេរលំដាប់ item នៃ Node តាម Inorder, Preorder និង Postorder Traversal ។



មេរៀនទី៧

Searching Techniques

7-1: Sequential Search

7-1-1: គោលការណ៍ដំណើរការ

Sequential Search គឺជាវិធានស្វែងរកធាតុណាមួយនៅក្នុងស៊េរីទិន្នន័យដោយអនុវត្តការស្វែងរកតាមលំនាំប្រៀបធៀបធាតុ ដែលត្រូវស្វែងរកជាមួយបណ្តាធាតុនីមួយៗនៅក្នុង List (List ជា Array: $a[0], a[1], \dots, a[n-1]$) ពីទីតាំងដំបូងនៃ Array (ធាតុ $a[0]$) ជាបន្តបន្ទាប់។ ប្រសិនបើរកឃើញធាតុដែលត្រូវរក (item) នៅទីតាំងណាមួយនៃ Array នោះផ្តល់តម្លៃ index របស់ធាតុនៃ Array និងបញ្ចប់ការងារស្វែងរកផ្ទុយមកវិញអោយតម្លៃ index នៃ Array កើនមួយទីតាំងជាបន្តបន្ទាប់រហូតដល់ទីតាំងចុងក្រោយនៃ Array គឺផ្តល់តម្លៃទទេ រួចបញ្ចប់ការងារស្វែងរក។

ឧទាហរណ៍:

ចូរប្រើគោលការណ៍នៃ Algorithms Sequential Search សម្រាប់ស្វែងរកធាតុ $item = 45$ នៅក្នុង List នៃ Array ចំនួនគត់ដូចខាងក្រោម៖

90 60 70 50 30 45 20 80 40

គេបាន

ស្វែងរកធាតុ $item = 45$ នៅក្នុង List

List ដំបូង	90	60	70	50	30	45	20	80	40
	a_0	a_1	a_2	a_3	a_4	a_5	a_6	a_7	a_8

index $i = 0$

90 60 70 50 30 45 20 80 40

↑

item = 45

index $i = 1$

90 60 70 50 30 45 20 80 40

↑

item = 45

index $i = 2$

90 60 70 50 30 45 20 80 40

↑

item = 45


```

for( i = 0 ; i < n ; i ++ )
    scanf ( "%d", & a [ i ] ) ;
printf( "Enter the item to be search \n" ) ;
scanf( "%d", & key ) ;
pos = seq_search ( key, n , a ) ;
if( pos == 0)
    printf( "Search unsuccessful \n" ) ;
else
    printf( "key found at position = %d \n", pos) ;
}

```

ឧទាហរណ៍២៖

ចូរសរសេរកម្មវិធីដើម្បីស្វែងរក item ដោយប្រើ Linear Search

```

#include< stdio.h >
void main( )
{
    int i, j, k, n, key, a[100], flag = 1;

    printf("Number of elements in the array?");
    scanf("%d", &n);
    printf("%d", n);

    printf("\n\nEnter array elements\n");
    for(i = 0; i < n; i ++ )
        scanf("%d", &a[i]);

    for(i = 0; i < n; i ++ )
        printf("%5d", a[i]);
    printf("\n");

    printf("\nElement to be searched?");
    scanf("%d", &key);
    printf("%d", key);
}

```

```
printf("\n");

/* Linear Search for the key */
for(i = 0; i < n; i++)
{
    if(a[i] == key)
    {
        printf("\nSearch Successful.\n");
        printf("Element %d found at location %d.\n", key, i + 1);
        flag = 0;
        break;
    }
}
if(flag)
    printf("\nSearch unsuccessful.\n");
}
```

នៅពេល **RUN** កម្មវិធីបង្ហាញលទ្ធផលដូចខាងក្រោម៖

Number of elements in the array? 7

Enter array elements

34 56 78 21 90 45 100

Element to be searched? 21

Search Successful.

Element 21 found at location 4.

នៅពេល **RUN** កម្មវិធីបង្ហាញលទ្ធផលដូចខាងក្រោម៖

Number of elements in the array? 5

Enter array elements

23 81 45 32 10

Element to be searched? 16

Search unsuccessful.

7-2: Binary Search

7-2-1: គោលការណ៍ដំណើរការ

Binary Search គឺជាវិធានស្វែងរកធាតុ item ណាមួយនៅក្នុងស៊េរីទិន្នន័យនៃ List ដែលបានរៀបតាមលំដាប់កំណត់ (លំដាប់កើន រឺលំដាប់ថយ) គោលការណ៍នៃវិធាននេះ គឺធ្វើការប្រៀបធៀបធាតុ item ជាមួយធាតុស្ថិតនៅទីតាំងពាក់កណ្តាលនៃ List ($a[mid]$) ដែល

$mid = (Left + Right) / 2$ ដែល index Left និង Right មានតម្លៃចាប់ផ្តើមជាមួយ

Left = 0 និង Right = $n - 1$ ។

-ប្រសិនបើធាតុ $item = a[mid]$ នោះផ្តល់តម្លៃ index mid រួចបញ្ចប់ការងារស្វែងរក។

-ប្រសិនបើធាតុ $item < a[mid]$ នោះស្វែងរកធាតុ item នៅក្នុង Sublist ទី១ពីទីតាំង

Left ដល់ Right = $mid - 1$ ។

-ប្រសិនបើធាតុ $item > a[mid]$ នោះស្វែងរកធាតុ item នៅក្នុង Sublist ទី២ ពីទីតាំង

Left = $mid + 1$ ដល់ Right ។

លំនាំអនុវត្តនេះត្រូវបានអនុវត្តជាបន្តបន្ទាប់រហូតដល់ពេលណា List មិនអាចធ្វើការបែងចែកទៀតបានទើបឈប់។

7-2-2: Algorithms Binary Search

គេមាន Algorithm Binary Search ត្រូវបានអនុវត្តតាមដំណាក់កាលដូចខាងក្រោម៖

-ឱ្យ index Left ដំបូងស្មើ 0

-ឱ្យ index Right ដំបូងស្មើ $n - 1$

-អនុវត្ត Loop Statement

+ ឱ្យ $index\ mid = (Left + Right) / 2$

+ ប្រសិន $item < a[mid]$ នោះស្វែងរកធាតុ item នៅក្នុង

Sublist1: $a[Left], \dots, a[mid-1]$

+ ប្រសិន $item = a[mid]$ នោះផ្តល់តម្លៃទីតាំង mid រួចបញ្ចប់ការងារស្វែងរក។

+ ប្រសិន $item > a[mid]$ នោះស្វែងរកធាតុ item នៅក្នុង

Sublist2: $a[mid+1], \dots, a[Right]$

ការងារនេះត្រូវបានអនុវត្តរហូតដល់ពេល $index\ Left > index\ Right$ ។

ឧទាហរណ៍៖

ចូរប្រើគោលការណ៍ Algorithms Binary Search សម្រាប់ស្វែងរកធាតុ item ស្មើ 45 នៅក្នុង List នៃ Array ចំនួនគត់ដូចខាងក្រោម៖

20 23 30 40 43 45 50 60 63 70 72 80 84 94 96

គេបាន

ធាតុត្រូវរកគឺ item ស្មើ 45 នៅក្នុង List នៃ Array ចំនួនគត់ដែលមាន 15 ធាតុ

List ដំបូងគឺ 20 23 30 40 43 45 50 60 63 70 72 80 84 94 96

• ដំណើរការទី១

$$\text{Left} = 0, \text{Right} = n - 1 = 15 - 1 = 14$$

$$\text{គេបាន } \text{mid} = (\text{Left} + \text{Right}) / 2 = (0 + 14) / 2 = 7$$

20 23 30 40 43 45 50 60 63 70 72 80 84 94 96

$a[\text{mid}]$



item = 45

ដោយ $\text{item} = 45 < a[\text{mid}] = a_7 = 60$

ដូចនេះការស្វែងរកធាតុ item ត្រូវអនុវត្តនៅក្នុង Sublist: 20 23 30 40 43 45 50

• ដំណើរការទី២

$$\text{Left} = 0, \text{Right} = \text{mid} - 1 = 7 - 1 = 6$$

$$\text{គេបាន } \text{mid} = (\text{Left} + \text{Right}) / 2 = (0 + 6) / 2 = 3$$

20 23 30 40 43 45 50

$a[\text{mid}]$



item = 45

ដោយ $\text{item} = 45 > a[\text{mid}] = 40$

ដូចនេះការស្វែងរកធាតុ item ត្រូវបានអនុវត្តនៅក្នុង Sublist: 43 45 50

• ដំណើរការទី៣

$$\text{Left} = \text{mid} + 1 = 3 + 1 = 4, \text{Right} = 6$$

$$\text{គេបាន } \text{mid} = (\text{left} + \text{Right}) / 2 = (4 + 6) / 2 = 5$$

$a[\text{mid}]$

43 45 50



item = 45

យើងឃើញថាត្រង់ទីតាំង mid ស្មើ 5 រកឃើញធាតុ item ស្មើ 45 ។

+អនុគមន៍ដើម្បីស្វែងរក item ក្នុង BST

```
int Bs(int a [ ] , int n , int item )
{
    int left, Right , mid ;
    left = 0 ;
    Right = n - 1 ;
    do
    {
        mid = (left + Right)/2;
        if(item == a [ mid ])
            return (mid) ;
        else if(item > a [ mid ] )
            Left = mid + 1
        else
            Right = mid - 1 ;
    } while(left <= Right) ;
    return (0) ;
}
```

ឧទាហរណ៍១៖

អនុគមន៍សម្រាប់ស្វែងរក item ក្នុង BST ដោយប្រើ iteration ។

```
NODE iteration_search( int item, NODE root )
{
    /* Search for the item */
    while( root != NULL && item != root -> info )
    {
        Root = ( item < root -> info ) ? root -> Llink : root -> Rlink ;
    }
    return( root );
}
```

ឧទាហរណ៍២៖

អនុគមន៍សម្រាប់ស្វែងរក item ក្នុង BST ដោយប្រើ Recursive ។

```
NODE recursive_search( int item, NODE root )
{
    if( root == NULL || item == root -> info )        return( root );
    if( item < root -> info )    return( recursive_search(item, root -> Llink );
    return( recursive_search( item, root -> Rlink );
}
```

ឧទាហរណ៍៣៖

Function to return the address of highest item in BST

```

NODE maximum( NODE root )
{
    NODE cur;
    if( root == NULL )      return( root );
    cur = root;
    while( cur -> Rlink != NULL ) cur = cur -> Rlink;    /* Obtain right most node
in BST */
    return( cur );
}

```

ឧទាហរណ៍៤៖

Function to return the address of least item in BST

```

NODE minimum(NODE root)
{
    NODE cur;
    if( root == NULL ) return( root );
    cur = root;
    while( cur -> Llink != NULL )
        cur = cur -> Llink;    /* Obtain left most node in BST */
    return( cur );
}

```

ឧទាហរណ៍៥៖

ចូរសរសេរកម្មវិធីដើម្បីស្វែងរក item ដោយប្រើ Binary Search ។

```

#include< stdio.h >

int search( item, a, low, high )
int item;    /* Element to search */
int a[ ];    /* Element to be search */
int low;     /* Points to the first element */
int high;    /* Points to the last element */
{
    int mid;    /* Points to the middle element of the table */
    if( low > high )    /* No item found */
        return( - 1 );
}

```

```

mid = low + (high - low)*((item - a[low])/(a[high] - a[low]));
return(item == a[mid]? mid + 1 :          /* return the middle element */
        item < a[mid]?
            search(item, a, low, mid - 1) :      /* search left part */
            search(item, a, mid + 1, high));      /* search right part */
}
void main( )
{
    int i, n, a[20], item, pos;
    printf(" Enter the number of elements\n");
    scanf(" %d", &n);
    printf("Enter %d items\n", n);
    for(i = 0; i < n ; i ++ )
    {
        scanf(" %d ", &a[i]);
    }
    printf(" Enter the item to be searched\n ");
    scanf(" %d ", &item);
    pos = search(item, a, 0, n - 1);    /* 0-low index and n - 1 is the high index */
    if(pos == -1)
        printf(" item not found\n");
    else
        printf(" item found at %d position \ n" , pos);
}

```

ឧទាហរណ៍៦៖

ចូរសរសេរកម្មវិធីដើម្បីស្វែងរក item ដោយប្រើ Binary Search ។

```

#include< stdio.h >
main( )
{
    int a[100], i, n, low, high, mid, term, flag = 1;

    printf("Number of elements in the array?");
    scanf("%d", &n);
    printf("%d", n);
    printf("\n\nEnter elements in ascending order:\n");
    for(i = 0; i < n; i ++ )
        scanf("%d", &a[i]);
    for(i = 0; i < n; i ++ )
        printf("%5d", a[i]);
    printf("\n");

    printf("\nElement to be searched?");
    scanf("%d", &term);
    printf("%d", term)
}

```

```

printf("\n");

low = 0;
high = n - 1;
while(low <= high)
{
    mid = (low + high)/2;
    if(term < a[mid])
        /* term in lower haft */
        high = mid - 1;
    else
        if(term > a[mid])
            /* term in upper haft */
            low = mid + 1;
        else
            if(term == a[mid])
            {
                printf("\nSearch successful.\n");
                printf("%d found at location %d.\n", term, mid + 1);
                flag = 0;
                break;
            }
    }
    if(flag)
        printf("\nSearch Unsuccessful.");
}

```

នៅពេល **RUN** កម្មវិធីបង្ហាញលទ្ធផលដូចខាងក្រោម៖

Number of elements in the array? 8

Enter elements in ascending order:

12 23 34 45 56 789 432 543

Element to be searched? 789

Search to be successful.

789 found at location 6.

នៅពេល **RUN** កម្មវិធីបង្ហាញលទ្ធផលដូចខាងក្រោម៖

Number of elements in the array? 5

Enter elements in ascending order:

1 2 3 4 5

Element to be searched? 6

Search unsuccessful.

ឧទាហរណ៍៧៖

ចូរសរសេរកម្មវិធីដើម្បីស្វែងរក item និងជំនួស item ថ្មីនៅក្នុង Linked List ។

```
# include< stdio.h >
# include< malloc.h >
# define NULL 0
struct list_element
{
    int item;
    struct list_element *next;
};
typedef struct list_element node;

main( )
{
    node *start;
    int key;
    int choice;

    void create( );
    void display( );
    node *lookup( );

    /* Create the linked list */(បង្កើត Linked List)

    start = (node *)malloc(sizeof(node));
    printf("\nTYPE-999 to STOP entry");

    create(start);

    printf("\n\nCreated List:\n");
    display(start);
    printf("\n\n");
    /* Looking up an item in the list */
    start = lookup(start);
    printf("\nLIST after Lookup:\n");
    display(start);
    printf("\n\n")
}

/* Creating the linked list recursively */(បង្កើត Linked List ជា Recursive)

void create(record)
node *record;

/* 'record' points to the current node */

{
    printf("\nDATA item?");
    scanf("%d", &record -> item);
```

```

printf("%6d", record -> item);

if(record -> item == -999)
    record -> next = NULL;
else
{
    /* Allocate memory space for next node */
    record -> next = (node *)malloc(sizeof(node));

    /* Create the next node */
    create(record -> next);
}
return;
}

/* Display the linked list recursively */(បង្ហាញ Linked List ជា Recursive)
void display(record)
node *record;

/* 'record' points to the current node */
{
    if(record -> next != NULL)
    {
        /* display the data item */
        printf("-> %d", record -> item);

        /* get the next data item for display */
        display(record -> next);
    }
    return;
}

/* Locate a particular node */
node *locate(record, target)
node *record;
int target;
/* return a pointer to the target node */
{
    static pos = 1;
    if(record -> item == target)
    {
        /* Node found */
        printf("\nItem FOUND at position : %4d\n", pos);
        return(record);
    }
    else if(record -> item == -999)
        /* End of list */
        return(NULL);
    else
    {
        /* try next node */

```

```

        pos ++;
        locate(record -> next, target);
    }
    return;
}

/* Look up an item in linked list and replace
with a new item if necessary */

node *lookup(first)
node *first;
{
    node *locate( );    /* function declaration */
    node *tag;          /* pointer to target node */
    node *temp;         /* temporary pointer */
    int target;         /* data item to be searched */
    int newitem;        /* replacement item */
    int opt;            /* choice variable */

    tag = first;

    printf("\nData item to be searched?");
    scanf("%d", &target);
    printf("%d\n", target);
    /* locate the target node */
    tag = locate(first, target);

    if(tag == NULL)
    {
        printf("\nELEMENT NOT FOUND.\n");
        return(first);
    }

    printf("\nDo you wish to change the item(1-YES, 2-NO)?");
    scanf("%d", &opt);
    printf("%d\n", opt);

    if(opt == 1)
    {
        printf("\nEnter the newitem:");
        scanf("%d", &newitem);
        printf("%d\n", newitem);
        tag -> item = newitem;
    }
    return(first);
}

```


នៅពេល **RUN** កម្មវិធីបង្ហាញលទ្ធផលដូចខាងក្រោម៖

TYPE -999 to STOP entry

Data item? 1

Data item? 2

Data item? 3

Data item? 4

Data item? 5

Data item? -999

Created List:

-> 1 -> 2 -> 3 -> 4 -> 5

Data item to be searched? 6

ELEMENT NOT FOUND.

LIST after Lookup:

-> 1 -> 2 -> 3 -> 4 -> 5

នៅពេល **RUN** កម្មវិធីបង្ហាញលទ្ធផលដូចខាងក្រោម៖

TYPE-999 to STOP entry

Data item? 2

Data item? 4

Data item? 6

Data item? 8

Data item? 9

Data item? -999

Created LIST:

-> 2 -> 4 -> 6 -> 8 -> 9

Data item to be searched? 9

Item FOUND at position: 5

Do you wish to change the item(1-YES, 2-NO)? 1

Enter the newitem: 10

LIST after Lookup:

-> 2 -> 4 -> 6 -> 8 -> 10

លំហាត់អនុវត្ត

- 1-ចូរពន្យល់អំពីវិធីសាស្ត្រ Searching Technique ។
- 2-ចូរពន្យល់អំពីវិធីសាស្ត្រ Sequential (Linear Search) Search with appropriate algorithm ។
- 3-ចូរសរសេរកម្មវិធីជាភាសា C ដើម្បីស្វែងរក item ដោយប្រើ Linear Search ។
- 4-ចូរសរសេរ Algorithm វិភាគអំពីដំណើរការរបស់ Binary Search ។
- 5-ចូរសរសេរកម្មវិធីជាភាសា C ដើម្បីស្វែងរក item ដោយប្រើ Binary Search ។



មេរៀនទី៨**Sorting Techniques****8-1: ស្ថានភាពក្តីផ្អែម**

មេរៀននេះណែនាំឲ្យយើងស្វែងយល់អំពីវិធីសាស្ត្រមួយចំនួន នៃការតំរៀបជាស៊េរី ទិន្នន័យដូចជា៖ វិធីសាស្ត្រ Bubble Sort, Insertion Sort, Quick Sort, Selection Sort, Heap Sort ,... ។

តំរៀប គឺជាការរៀបឡើងវិញនូវទិន្នន័យនៅក្នុង List នៅតាមលំដាប់កំណត់(លំដាប់កើត ឬលំដាប់ថយជាបន្តបន្ទាប់)គោលការណ៍នៃការតំរៀបគឺធ្វើការប្រៀបធៀបនិងផ្លាស់ប្តូរ ទីតាំងធាតុជាបន្តបន្ទាប់នៅក្នុង List ប្រសិនបើពុំទាន់ត្រឹមត្រូវតាមលំដាប់កំណត់។

8-2: Bubble Sort (n^2)**8-2-1: គោលការណ៍ដំណើរការ**

Bubble Sort គឺជាវិធានតំរៀបទិន្នន័យជាស៊េរីនៅក្នុង List ដោយអនុវត្តតាម លំដាប់ដំណើរការ List ច្រើនសារឃើញក្នុងដំណាក់កាលដំណើរការ List ម្តងៗគឺធ្វើការប្រៀប ធៀបធាតុ $a[i]$ ជាមួយ $a[i+1]$ និងផ្លាស់ប្តូរទីតាំងធាតុទាំងពីរប្រសិនបើពុំទាន់ត្រឹមត្រូវតាម លំដាប់កំណត់។

ក្រោយពេលដំណើរការ List ម្តងៗ គឺធាតុធំបំផុតក្នុង List ត្រូវកំណត់ចូលទីតាំងត្រឹមត្រូវរបស់វា ហើយដំណើរការនេះត្រូវបានអនុវត្តជាបន្តបន្ទាប់រហូតធាតុទាំងអស់នៅក្នុង List ត្រូវបានកំណត់ចូលទីតាំងត្រឹមត្រូវរបស់វាក្នុង List ដែលក្រោយពេលដំណើរការលើក ទី $n-1$ ទើបបញ្ចប់ការងារតំរៀប List ។

ឧទាហរណ៍៖

ចូរប្រើគោលការណ៍នៃ Algorithms Bubble Sort ដើម្បីតំរៀប List ដូចខាងក្រោម៖

40 30 23 16 45 20 70 50 34 60 42 53

តាមលំដាប់កើនជាបន្តបន្ទាប់។

ដើម្បីអនុវត្តលើការតំរៀប List តាម Bubble Sort ជាដំបូងយើងត្រូវស្វែងយល់អំពី ដំណើរការ List ជាលើកដំបូងតាមតារាងបំភ្លឺដំណើរការដូចខាងក្រោម៖

លើកទី i ៖	$a[i]$	$a[i+1]$	$a[i] > a[i+1]$	$a[i]$	$a[i+1]$
	មុន	មុន	ត្រួតពិនិត្យ	បន្ទាប់	បន្ទាប់
0	40	30	True \rightarrow change	30	40
1	40	23	True \rightarrow change	23	40
2	40	16	True \rightarrow change	16	40
3	40	45	False	40	45
4	45	20	True \rightarrow change	20	45
5	54	70	False	45	70
6	70	50	True \rightarrow change	50	70
7	70	34	True \rightarrow change	34	70
8	70	60	True \rightarrow change	60	70
9	70	42	True \rightarrow change	42	70
10	70	53	True \rightarrow change	53	70

យើងឃើញថា តាមតារាងបំភ្លឺដំណើរការ List ជាលើកដំបូងគឺ List ត្រូវបានប្រែប្រួល

30 23 16 40 20 45 50 34 60 42 53 70

មានន័យថាក្រោយពេលដំណើរការរបស់ Bubble Sort គឺជាតុំធំបំផុត (Node 70) នៃ List ត្រូវបានកំណត់ចូលទីតាំងត្រឹមត្រូវរបស់វា (index ទី $n-1$) ។ គេបានដំណើរការ Bubble Sort ដូចខាងក្រោម៖

List ដំបូង៖ 40 30 23 16 45 20 70 50 34 60 42 53
 $a_0 \quad a_1 \quad a_2 \quad a_3 \quad a_4 \quad a_5 \quad a_6 \quad a_7 \quad a_8 \quad a_9 \quad a_{10} \quad a_{11}$

• បន្ទាប់ពីដំណើរការលើកទី១

30 23 16 40 20 45 50 34 60 42 53 70

• បន្ទាប់ពីដំណើរការលើកទី២

23 16 20 30 40 45 34 50 42 53 60 70

• បន្ទាប់ពីដំណើរការលើកទី៣

16 23 20 30 40 34 45 42 50 53 60 70

• បន្ទាប់ពីដំណើរការលើកទី៤

16 20 23 30 34 40 42 45 50 53 60 70

• បន្ទាប់ពីដំណើរការលើកទី៥

16 20 23 30 34 40 42 45 50 53 60 70

- បន្ទាប់ពីដំណើរការលើកទី១១

16 20 23 30 34 40 42 45 50 53 60 70

8-2-2: Algorithms: Bubble Sort

គេអាចសរសេរ Algorithms របស់ Bubble Sort តាមភាសា C ដូចខាងក្រោម៖

```
void Bubble_sort(int a[ ], int n )
{
    int i , j , Temp ;
    for(j = 1 ; j < n ; j ++ )
    {
        for (i = 0 ; i < n - j ; i ++ )
        {
            if( a [ i ] > a [ i + 1 ] )
            {
                Temp = a [ i ] ;
                a[ i ] = a [ i + 1 ] ;
                a[ i + 1 ] = Temp ;
            }
        }
    }
}
```

ឧទាហរណ៍១៖

ចូរសរសេរកម្មវិធីដើម្បីតំរៀបលេខតាមលំដាប់ពីតូចទៅធំដោយប្រើ Bubble Sort។

```
# include< stdio.h >

void bubble_sort( int a[ ], int n )
{
    int i;          /* To access subsequent item while comparing */
    int j;          /* Keep track of the passes */
    int temp;       /* Used to exchange the item */
    int sum;        /* Holds the total number of exchanges */
    int pass;       /* Holds the number of passes required */
}
```

```

int exchag; /* Holds the number of exchanges in each pass */
int flag;   /* Indicate any exchange has been done or not */
sum = 0;
pass = 0;
for( j = 1; j < n; j ++ )
{
    exchag = 0; /* Number of exchanges just before the pass */
    flag = 0;   /* No exchange been done */
    for( i = 0; i < n; i ++ )
    {
        if( a[ i ] >= a[ i + 1 ] )
        {
            /* Exchange and update the number of exchange in the current pass */
            temp = a[ i ];
            a[ i ] = a[ i + 1 ];
            a[ i + 1 ] = temp;
            exchag ++ ;
            sum ++ ; /* Update the total number of exchanges */
            flag = 1; /* Exchange has been done */
        }
    }
    pass ++ ; /* Update the number of passes */
    printf( " Number of exchanges in pass : %d = %d\n " , j, exchag );
    printf( " Total number of exchanges = %d\n" , sum );
}
void main( )
{
    int i, n, a[ 20 ];

    printf( " Enter the number of items to sort\n " );
    scanf( " %d", &n);

    printf( " Enter the items to sort\n" );
    for( i = 0; i < n; i ++ )
        scanf( " %d " , &a[ i ] );

    bubble_sort( a, n );

    printf( " The sorted items are\n " );
    for(i = 0; i < n; i ++ )
    {
        printf( " %d\n", a[ i ] );
    }
}

```

ឧទាហរណ៍២

សរសេរកម្មវិធីដើម្បីតំរៀបលេខតាមលំដាប់ពីតូចទៅធំដោយប្រើ Bubble Sort ។

```
#include< stdio.h >
main( )
{
    int i, j, k, n, flag, limit, a[50];

    printf("Number of array elements?");
    scanf("%d", &n);
    printf("%d", n);

    printf("\n\nEnter array elements\n");
    for(i = 0; i < n; i++)
        scanf("%d", &a[i]);
    for(i = 0; i < n; i++)
        printf("%5d", a[i]);
    printf("\n");

    /* Bubble Sort */
    limit = n - 1;
    flag = 1;

    for(i = 0; i < n - 1; i++)
    {
        for( j = 0; j < limit - i; j++)
        {
            if(a[ j ] > a[ j + 1])
            {
                k = a[ j ];
                a[ j ] = a[ j + 1];
                a[ j + 1] = k;
                flag = 0;
            }
        }
        if(flag)
            break;          /* No swaps */
        else
            flag = 1;
    }
    printf("\nSorted array(Bubble Sort):\n");
    for(i = 0; i < n; i++)
        printf("%5d", a[i]);
    printf("\n");
}
```

នៅពេល RUN កម្មវិធីបង្ហាញលទ្ធផលដូចខាងក្រោម៖

Number of array elements? 7

Enter array elements

12 34 67 89 54 678 41

Sorted array(Bubble array):

12 34 41 54 67 89 678

8-3: Quick Sort ($O(\log_2 n)$)

8-3-1: គោលការណ៍ដំណើរការ

Quick Sort គឺជាវិធានតំរៀបទិន្នន័យជាស៊េរីតាមផ្នែកដោយអនុវត្តការបែងចែក List ជាពីរផ្នែកគឺ Sublist1 និង Sublist2 តាមលំនាំនៃការកំណត់ pivot node ដែលជាធាតុស្ថិតនៅទីតាំងពាក់កណ្តាលនៃ List គឺ

$$\text{pivot} = a[\text{mid}] \text{ ជាមួយ index } \text{mid} = (\text{Left} + \text{Right})/2$$

បន្ទាប់មកធ្វើការអនុវត្តដំណើរការតំរៀបតាមលំនាំដូចខាងក្រោម៖

+ ប្រៀបធៀបធាតុ $a[i]$ និងធាតុ $a[j]$ ជាមួយ pivot node ដែល index $i = 0, 1, 2, \dots$ និង $j = n - 1, \dots$

- នៅពេល $a[i] < \text{pivot node}$ អោយ i កើន

- នៅពេល $a[j] < \text{pivot node}$ អោយ j កើន

- ប្រសិន index $i \leq j$ នោះ

- ប្តូរ $a[i]$ និង $a[j]$

- អោយ index $i = i + 1$

- អោយ index $j = j - 1$

លំនាំនេះត្រូវបានអនុវត្តជាបន្តបន្ទាប់រហូតដល់ index ដំបូង index $\text{Left} > \text{index Right}$ ដែល Left ដំបូងស្មើ 0 និង Right ដំបូងស្មើ $n-1$ គឺគេបែងចែកបាន Sublist ពីរ (Sublist1 ពីទីតាំង Left ដល់ j និង Sublist2 ពីទីតាំង i ដល់ទីតាំង Right) ដំណើរការអនុវត្តនេះត្រូវបានអនុវត្តចំពោះ Sublist1 និង Sublist2 ជាបន្តបន្ទាប់រហូតដល់មិនអាចធ្វើការបែងចែក List បន្តទៀតបានទើបឈប់។

8-3-2. Algorithms Quick sort

គេមាន Algorithms Quick Sort ត្រូវបានអនុវត្តតាមដំណាក់កាលដូចខាងក្រោម៖

* អោយ index $i = \text{left}$, $j = \text{Right}$ ($\text{Left} = 0$ & $\text{Right} = n-1$)

* អោយ $\text{pivot} = a[(\text{Left} + \text{Right}) / 2]$

* Loop Statement

+ នៅពេល $a[i] < \text{pivot}$ នោះ $i = i + 1$

+ នៅពេល $a[j] < \text{pivot}$ នោះ $j = j - 1$

+ ប្រសិនបើ $i \leq j$ នោះប្តូរ $a[i]$ និង $a[j]$ និងអោយ $i = i + 1$, $j = j - 1$ រហូតដល់

index $i > \text{index } j$ គឺបាន Sublist1 និង Sublist2 ដែល Sublist1: $a[\text{Left}]$ ដល់ $a[i]$ និង Sublist2 : $a[i]$ ដល់ $a[\text{Right}]$ ។

* ប្រសិន index $\text{Left} < \text{index } j$ នោះដំណើរការ Recursive ចំពោះ Sublist1 ។

* ប្រសិន index $i < \text{Right}$ នោះដំណើរការ Recursive ចំពោះ Sublist2 ។

គេអាចសរសេរ Algorithms Quick Sort តាម C language ដូចខាងក្រោម៖

```
void Quick_Sort (int a[ j ] , int L , int R )
```

```
{
    int i , j , pivot , temp ;
    i = L ;
    j = R ;
    pivot = a [( L + R ) / 2 ] ;
    do
    {
        while ( a [ i ] < pivot ) i ++ ;
        while ( a [ i ] > pivot ) j -- ;
        if ( i <= j )
        {
            temp = a [ i ] ;
            a [ i ] = a [ j ] ;
            a [ j ] = temp ;
            i ++ ;
            j -- ;
        }
    }
```

```

    } while ( i <= j ) ;
    if ( L < j ) Quick sort ( a , L , j ) ;
    if ( i < R ) Quick sort ( a , i , R ) ;
}

```

ឧទាហរណ៍៖

គេមាន List នៃ Array ដូចខាងក្រោម៖

30 40 20 45 90 23 72 16 80 18 50 60

ចូរប្រើគោលការណ៍ Algorithms Quick Sort សម្រាប់តំរៀប List នេះតាមលំដាប់កើនជាបន្តបន្ទាប់។

គេបានដំណើរការ List នេះជាលើកដំបូងដោយប្រើ Quick Sort ត្រូវបានបង្ហាញដូចខាងក្រោម៖

List ដំបូង៖

30	40	20	45	90	23	72	16	80	18	50	60
a_0	a_1	a_2	a_3	a_4	a_5	a_6	a_7	a_8	a_9	a_{10}	a_{11}

ឧទាហរណ៍៖

ចូរសរសេរកម្មវិធីដើម្បីតំរៀបលេខតាមលំដាប់ពីតូចទៅធំដោយប្រើ QuickSort ។

```

#include< stdio.h >
/* Function to partition the array for quick sort */
int partition( int a[], int low, int high )
{
    int i, j, temp, key;
    key = a[low];
    i = low + 1;
    j = high;
    while(1)
    {
        while( i < high && key >= a[i])
            i ++;
        while( key < a[j] )
            j --;
        if( i < j )
        {
            temp = a[i];
            a[i] = a[j];
            a[j] = temp;
        }
        else

```

```

        {
            temp = a[low];
            a[low] = a[j];
            a[j] = temp;
        }
    }
}
/* Function to sort the numbers in ascending order using quick sort */
void quicksort( int a[], int low, int high )
{
    int j;
    if(low < high )
    {
        j = partition(a, low, high);    /* Partion the array into two subtables */
        quicksort(a, low, j - 1);      /* Sort the left part of the array */
    }
}
void main( )
{
    int i, n, a[20];
    printf("Enter the value for n\n");
    scanf("%d",&n);
    printf("Enter the number to be sorted\n");
    for( i = 0; i < n; i ++ )
        scanf("%d",&a[i]);
    quicksort( a, 0, n - 1 );
    printf("The sorted array is \n");
    for( i = 0; i < n; i ++ )
        printf("%d\n", a[i]);
}

```

8-4: Selection Sort

8-4-1: គោលការណ៍ដំណើរការ

Selection Sort គឺដំណើរការច្រើនដងហើយរាល់ពេលដំណើរការម្តងៗគឺរកធាតុតូចបំផុតក្នុងបណ្តាធាតុដែលស្ថិតនៅក្នុង List ពីទី $i + 1$ រហូតដល់ n បន្ទាប់មកផ្លាស់ប្តូររវាងធាតុតូចបន្ទាប់នេះ និងធាតុទី i ។

ឧទាហរណ៍៖

គេមាន List នៃ Array ចំនួនគត់ដូចខាងក្រោម៖

53 40 92 34 72 90 85 26 21

គេអាចប្រើ Selection Sort ដើម្បី Sort List នេះទៅតាមលំដាប់កើនជាបន្តបន្ទាប់។

បន្ទាប់ពីដំណើរការជាបន្តបន្ទាប់នោះគេបាន៖

List ដើម ៖	53	40	92	34	72	90	85	26	21
ដំណើរការលើកទី១ ៖	(21)	40	92	34	72	90	85	26	53
ដំណើរការលើកទី២ ៖	21	(26)	92	34	72	90	85	40	53
ដំណើរការលើកទី៣ ៖	21	26	(34)	92	72	90	85	40	53
ដំណើរការលើកទី៤ ៖	21	26	34	(40)	72	90	85	92	53
ដំណើរការលើកទី៥ ៖	21	26	34	40	(53)	90	85	92	72
ដំណើរការលើកទី៦ ៖	21	26	34	40	53	(72)	85	92	90
ដំណើរការលើកទី៧ ៖	21	26	34	40	53	72	(85)	92	90
ដំណើរការលើកទី៨ ៖	21	26	34	40	53	72	85	(90)	92

8-4-2: Algorithms(C code for Selection Sort)

អនុគមន៍ដើម្បីតំរៀបទិន្នន័យដោយប្រើ Selection Sort ។

```

void selection_sort(int n, int A[ ])
{
    int minindex, j, p, temp;
    for(p = 0; p < n; p ++ )
    {
        minindex = p;
        for(j = p + 1; j < n; j ++ )
            if(A[j] < A[minindex])
                minindex = j;
        temp = A[p];
        A[p] = A[minindex];
        A[minindex] = temp;
    }
}
    
```

ឧទាហរណ៍១៖

ចូរសរសេរកម្មវិធីដើម្បីតំរៀបលេខតាមលំដាប់ពីតូចទៅធំដោយប្រើ Selection Sort ។

```
#include< stdio.h >
void selection_sort( int a[], int n )
{
    int i, j, pos, small, temp;
    for( i = 0; i < n; i ++ )
    {
        small = a[i];           /* Initial small number in ith pass */
        pos = i;                /* Position of smaller number */
        /* Find the minimum of remaining elements along with the position */
        for(j = i + 1; j < n; j ++ )
        {
            if( a[j] < small )
            {
                small = a[j];
                pos = j;
            }
        }
        /*Exchange ith item with last item */((ប្តូរធាតុនៅទីតាំងដំបូងនិងធាតុនៅទីតាំងចុងក្រោយ))
        temp = a[pos];
        a[pos] = a[i];
        a[i] = temp;
    }
}
void main( )
{
    int i, n, a[20];
    printf("Enter the number of elements to sort\n");
    scanf("%d",&n);
    printf("Enter %d elements to sort\n",n);
    for( i = 0; i < n; i ++ )
        scanf("%d",&a[i]);
    selection_sort(a, n);
    printf("The sort element are\n");
    for( i = 0; i < n; i ++ )
        printf("%d",&a[i] );
}
```

ឧទាហរណ៍២៖

ចូរសរសេរកម្មវិធីដើម្បីតំរៀបលេខតាមលំដាប់ពីតូចទៅធំដោយប្រើ Selection Sort។

```
# include< stdio.h >
void main( )
{
    int i, j, k, n, pass, min, a[100];
    printf("Number of elements in array?");
    scanf("%d", &n);
    printf("%d", n);
    printf("\n\nArray elements?\n");
    for(i = 0; i < n; i++)
        scanf("%d", &a[i]);
    for(i = 0; i < n; i++)
        printf("%5d", a[i]);
    printf("\n");

    /* Selection Sort */
    for(pass = 0; pass < n - 1; pass++)
    {
        /* assume that pass is the minimum index */
        min = pass;
        for( j = pass + 1; j < n; j++)
        {
            if(a[min] > a[ j])
                /* update min */
                min = j;
        }
        if( min != pass)
        {
            /* swap elements */
            k = a[pass];
            a[pass] = a[min];
            a[min] = k;
        }
    }
    printf("\nSorted array(Selection Sort):\n");
    for(i = 0; i < n; i++)
        printf("%5d", a[i]);
    printf("\n");
}
```

នៅពេល RUN កម្មវិធីបង្ហាញលទ្ធផលដូចខាងក្រោម៖

Number of elements in array? 7

Array elements?

12 45 32 789 76 43 890

Sorted array(Selection Sort):

12 32 43 45 76 789 890

8-5: Insertion Sort

8-5-1: គោលការណ៍ដំណើរការ

Insertion មានលំនាំដូចវិធាននៃការតំរៀបសន្លឹកបៀតាមលំដាប់កំណត់។ តំរៀបដោយយកសន្លឹកបៀទី i ចេញពីសន្លឹកបៀដែលត្រូវតំរៀប និងធ្វើការប្រៀបធៀបជាមួយបៀដែលមាន។ ប្រសិនបើសន្លឹកបៀ $a[i-1]$ ធំជាងសន្លឹកបៀដែលត្រូវ Insert នោះរំកិលសន្លឹកបៀទី $i-1$ ទៅកាន់ទីតាំងទី i បន្ទាប់មកធ្វើការប្រៀបធៀបជាមួយសន្លឹកបៀទី $i-2$ ។ ប្រសិនបើសន្លឹកបៀទី $i-2$ នៅតែធំជាងទៀតនោះរំកិលសន្លឹកបៀទី $i-2$ ទៅកាន់ទីតាំងទី $i-1$ ហើយការងារប្រៀបធៀបនេះត្រូវបានអនុវត្តច្រើនដងដោយចាប់ផ្តើមពី $i = 1$ រហូតដល់ $i = n - 1$ ។

វិភាគ៖

ប្រសិនបើ List មាន n ធាតុនោះ៖

+មាន $n - 1$ ដងនៃការ Insert

+រាល់ពេល Insert ធាតុថ្មីចូល List នេះត្រូវអនុវត្ត

-ស្វែងរកទីតាំងដែលត្រូវ Insert

-រំកិលរាល់ធាតុដែលស្ថិតនៅទីតាំងនៃ $i-1$ មកកាន់ទីតាំងទី i

-យកធាតុដែលត្រូវ Insert បញ្ចូលទីតាំងនោះ

ឧទាហរណ៍៖

គេមាន List នៃ Array ដូចខាងក្រោម៖

25 50 46 40 12 90 83 29 34

ដោយប្រើគោលការណ៍ Insertion Sort ដើម្បី Sort List នេះទៅតាមលំដាប់កើន។

យើងពិនិត្យមើលដំណើរការជាបន្តបន្ទាប់ដែលបង្ហាញដូចខាងក្រោម៖

List ដើម៖

	25	50	46	40	12	90	83	29	34
+ដំណើរការទី 1	25	50	46	40	12	90	83	29	34
+ដំណើរការទី 2	25	46	50	40	12	90	83	29	34
+ដំណើរការទី 3	25	40	46	50	12	90	83	29	34
+ដំណើរការទី 4	12	25	40	46	50	90	83	29	34
+ដំណើរការទី 5	12	25	40	46	50	90	83	29	34
+ដំណើរការទី 6	12	25	40	46	50	83	90	29	34
+ដំណើរការទី 7	12	25	29	40	46	50	83	90	34
+ដំណើរការទី 8	12	25	29	34	40	46	50	83	90

8-5-2: Algorithms(C Code) for Insertion Sort

អនុគមន៍ដើម្បីតំរៀបទិន្នន័យដោយប្រើ Insertion Sort ។

```
void insertion_sort( int a[], int n )
{
    int i, j, item;
    for( i = 0; i < n; i ++ )
    {
        item = a[i];
        j = i - 1;
        while( item < a[j] && j >= 0 )
        {
            a[j + 1] = a[j];
            j - -;
        }
        a[j + 1] = item;
    }
}
```


ឧទាហរណ៍១៖

ចូរសរសេរកម្មវិធីដើម្បីតំរៀបលេខតាមលំដាប់ពីតូចទៅធំដោយប្រើ Insertion Sort ។

```
#include< stdio.h >
void insertion_sort( int a[], int n )
{
    int i,j, item;
    for( i = 0; i < n; i + + )
    {
        /* item to be inserted */
        item = a[i];
        j = i - 1;
        while( j >= 0 && item < a[j] )
        {
            A[ j + 1 ] = a[j];      /* Move the item to the next position */
            j - -;                  /* and update the position */
        }
        A[ j + 1 ] = item;          /* appropriate position found and so insert item*/
    }
}

void main( )
{
    int i, n, a[20];
    printf(" Enter the no. of elements to sort \n" );
    scanf("%d",&n);
    printf(" Enter n elements\n");
    for(i = 0; i < n; i + + )
        scanf("%d",&a[i])
    insertion_sort(a,n);
    printf("The sorted array is \n");
    for(i = 0;i < n; i + +)
        printf("%d\n",a[i]);
}
```

ឧទាហរណ៍២៖

ចូរសរសេរកម្មវិធីដើម្បីតំរៀបលេខតាមលំដាប់ពីតូចទៅធំដោយប្រើ Insertion Sort ។

```
#include< stdio.h >
void main( )
{
    int i, j, k, x, a[50];
    printf("Element to be inserted(-999 to terminate)?");
    scanf("%d", &x);
    printf("%d", x);
    printf("\n");
    i = 0;
    while(x != -999)
    {
        k = i - 1;
        while((x < a[k]) && (k >= 0))
        {
            a[k + 1] = a[k];
            - - k;
        }
        a[k + 1] = x;
        printf("Array after inserting %d : \n", x);
        for(j = 0; j < i; j++)
            printf("%5d", a[j]);
        printf("\n");
        printf("\nElement to be inserted(-999 to terminate)?");
        scanf("%d", &x);
        printf("%d", x);
        printf("\n");
        ++ i;
    }
    printf("The final sorted array(insertion sort):\n");
    for(j = 0; j < i; j++)
        printf("%5d", a[j]);
    printf("\n");
}
```

នៅពេល RUN កម្មវិធីបង្ហាញលទ្ធផលដូចខាងក្រោម៖

Element to be inserted(-999 to terminate)? 45

Array after inserting 45:

45

Element to be inserted(-999 to terminate)? 23

Array after inserting 23:

23 45

Element to be inserted(-999 to terminate)? 78

Array after inserting 78:

23 45 78

Element to be inserted(-999 to terminate)? 12

Array after inserting 12:

12 23 45 78

Element to be inserted(-999 to terminate)? 2

Array after inserting 2:

2 12 23 45 78

Element to be inserted(-999 to terminate)? 89

Array after inserting 89:

2 12 23 45 78 89

Element to be inserted(-999 to terminate)? -999

The final sorted array(insertion sort):

2 12 23 45 78 89

8-6: Merge Sort

8-6-1: គោលការណ៍ដំណើរការ

Merge Sort គឺជាការ Merge បញ្ចូលគ្នារវាង List ដែលមានលំដាប់ពីរ(Sublist) ឲ្យទៅជា List តែមួយជាលំដាប់។ លំនាំនៃ Merge Sort ដោយធ្វើការប្រៀបធៀបធាតុពីរ របស់ Sublist ទាំងពីរដើម្បីជ្រើសរើសយកធាតុដែលតូចជាងមកដាក់ក្នុង List បណ្តោះ អាសន្នមួយលំដាប់នេះចេះតែបន្តរហូតដល់អស់ធាតុពី Sublist ណាមួយក្នុងចំណោម Sublist ទាំងពីរបន្ទាប់មកយកធាតុដែលសល់ក្នុង Sublist ដាក់បញ្ចូលក្នុង List បណ្តោះអាសន្ន។

ឧទាហរណ៍៖

គេមាន Sublist ពីរដូចខាងក្រោម៖

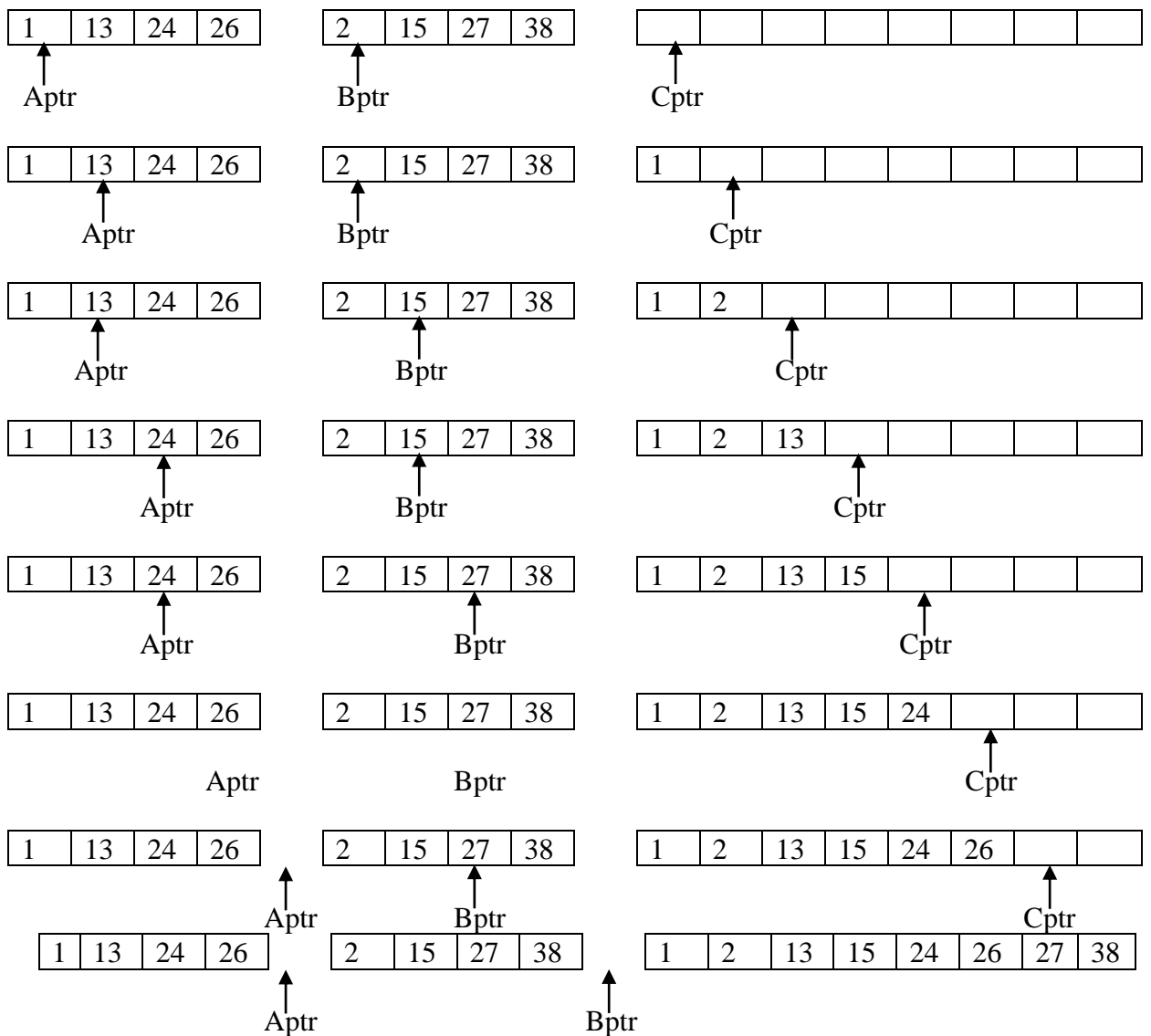
Sublist1

1	13	24	26
---	----	----	----

Sublist2

2	15	27	38
---	----	----	----

ដំណើរការនៃ Merge Sort គឺ



8-6-2: Algorithms (C code) for Merge Sort

1-អនុគមន៍សម្រាប់តំរៀបទិន្នន័យដោយប្រើ Merge Sort

```
void MSort( ElementType A[], ElementType TmpArray[], int Left, int Right )
{
    int Center;
    if( Left < Right )
    {
        Center = ( Left + Right )/2;
        MSort( A, TmpArray, Left, Center );
        MSort( A, TmpArray, Center + 1, Right );
        Merge( A, TmpArray, Left, Center + 1, Right );
    }
}

void Mergesort( ElementType A[], int N )
{
    ElementType *TmpArray;
    TmpArray = malloc( N * sizeof( ElementType));
    if( TmpArray != NULL )
    {
        MSort( A, TmpArray, 0, N - 1 );
        free( TmpArray );
    }
    else
        FatalError( "No space for tmp array !!!" );
}
```

2-អនុគមន៍សម្រាប់តំរៀបទិន្នន័យដោយប្រើ Merge Sort

```
/* Lpos = start of left half, Rpos = start of right half */
void Merge( ElementType A[], ElementType TmpArray[],
            int Lpos, int Rpos, int RightEnd )
{
    int i, LeftEnd, NumElements, TmpPos;
    LeftEnd = Rpos - 1;
```

```

    TmpPos = Lpos;
    NumElements = RightEnd - Lpos + 1;
    /* main loop */
    while( Lpos <= LeftEnd && Rpos <= RightEnd )
        if( A[ Lpos ] <= A[Rpos] )
            TmpArray[TmpPos ++] = A[Lpos ++];
        else
            TmpArray[TmpPos ++] = A[Rpos ++];
    while( Lpos <= LeftEnd ) /* Copy rest of first half */
        TmpArray[TmpPos ++] = A[ Lpos ++];
    while( Rpos <= RightEnd ) /* Copy rest of second half */
        TmpArray[TmpPos ++] = A[Rpos ++];
    /* Copy TmpArray back */
    for( i = 0; i < NumElements; i ++, RightEnd - -)
        A[RightEnd] = TmpArray[RightEnd];
}

```

ឧទាហរណ៍៖

```

#include< stdio.h >
main( )
{
    float vector[100];
    int i, j, k, n;
    void merge_sort( );
    void merge_pass( );

    printf("Size of the list?");
    scanf("%d", &n);
    printf("%d\n", n);
    printf("\nElements of the list?\n");
    for( i = 0; i < n; i ++ )
        scanf("%f", &vector[i]);
    for( i = 0; i < n; i ++ )
        printf("%8.1f", vector[i]);
    printf("\n");
    i = 0;
    merge_pass(vector, i, n - 1);
    printf("\n\nMerge_sorted List:\n");
    for( i = 0; i < n; i ++ )
        printf("%8.1f", vector[i]);
}

```

```

        printf("\n");
    }
    void merge_sort(list, top, size, bot)

    float list[100];
    int top, size, bot;
    {
        int u, f1, s1, t1;
        float tmp[100];
        f1 = top;
        s1 = size + 1;
        t1 = top;

        while((f1 <= size) && (s1 <= bot))
        {
            if(list[f1] <= list[s1])
            {
                tmp[t1] = list[f1];
                f1 = f1 + 1;
            }
            else
            {
                tmp[t1] = list[s1];
                s1 = s1 + 1;
            }
            t1 = t1 + 1;
        }
        if(f1 <= size)
        {
            for(f1 = f1; f1 <= size; f1++)
            {
                tmp[t1] = list[f1];
                t1 = t1 + 1;
            }
        }
        else
        {
            for(s1 = s1; s1 <= bot; s1++)
            {
                tmp[t1] = list[s1];
                t1 = t1 + 1;
            }
        }
        for(u = top; u <= bot; u++)
            list[u] = tmp[u];
        return(0);
    }

```

```
void merge_pass(a, m, n)
float a[100];
int m, n;
{
    int h;
    if(m != n)
    {
        h = (m + n)/2;
        merge_pass(a, m, h);
        merge_pass(a, h + 1, n);
        merge_sort(a, m, h, n);
    }
    return(0);
}
```

នៅពេល RUN កម្មវិធីបង្ហាញលទ្ធផលដូចខាងក្រោម៖

Size of the list? 8

Elements of the list?

234.0 -9.0 23.0 10.0 77.0 -11.0 0.0 2.0

Merge_sorted List:

-11.0 -9.0 0.0 2.0 10.0 23.0 77.0 234.0

8-7 Heap Sort

8-7-1 គោលការណ៍ដំណើរការ

Heap Sort ត្រូវបានបែងចែកជាពីរជំហានដូចខាងក្រោម៖

-ជំហានទី១

Binary Tree ដែលតាងឱ្យ List ដែលត្រូវ sort បំប្លែងទៅជាទម្រង់ Heap ដែលហៅថាការបង្កើត Heap។ Heap ជា Complete Binary Tree ដែលគ្រប់ Parent Node សុទ្ធតែមាន item ធំជាង Parent Node របស់វា។

-ជំហានទី២

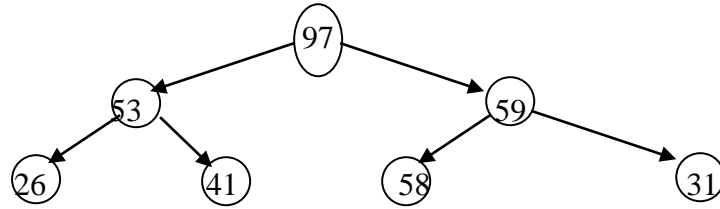
ជាដំណាក់កាលនៃការតម្រៀប គឺដំណើរការជាច្រើនដង ដែលរាល់ពេល ដំណើរការម្តងៗត្រូវអនុវត្តដូចខាងក្រោម៖

+យក item នៃ Node Root (item ដែលធំជាងគេបំផុត) ត្រូវដាក់នៅទីតាំង ត្រឹមត្រូវដោយប្រើរបៀប swape ជាមួយ item ដែលស្ថិតនៅទីតាំងនោះ។

+ផ្តុំឡើងវិញអោយក្លាយទៅជា Heap (List Heap) នៅរាល់ Node ដែលនៅ សល់ក្រោយពីយក Node Root ចេញ។

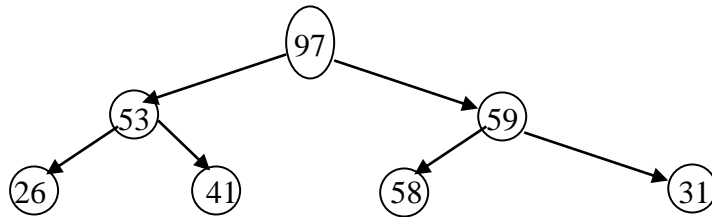
ឧទាហរណ៍៖

គេមាន Complete Binary Tree ដូចខាងក្រោម៖



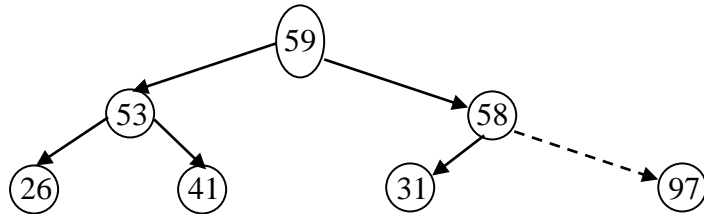
ចូរប្រើគោលការណ៍ Heap Sort ដើម្បី Sort List ទៅតាមលំដាប់កើន។

+Heap after build heap phase

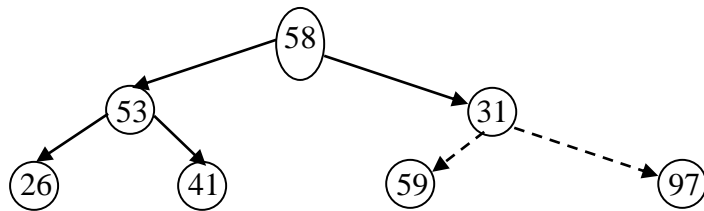


	97	53	59	26	41	58	31			
0	1	2	3	4	5	6	7	8	9	10

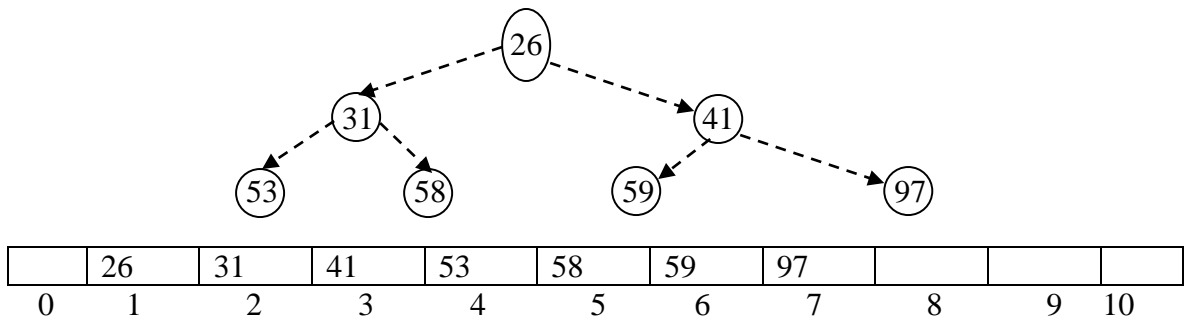
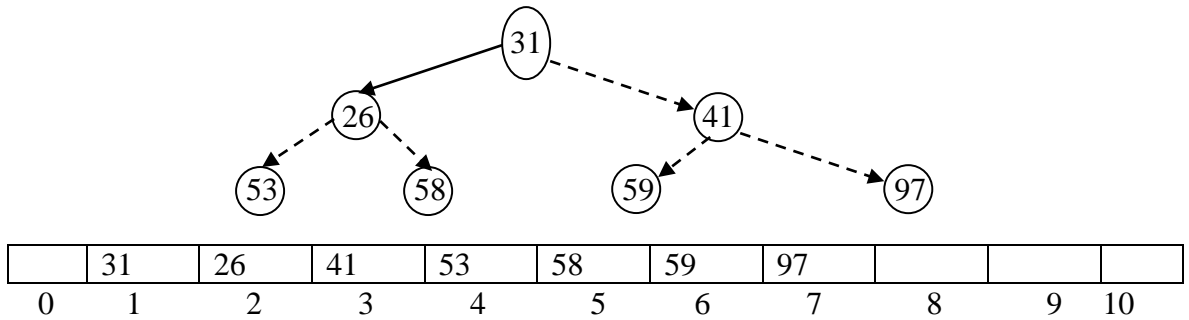
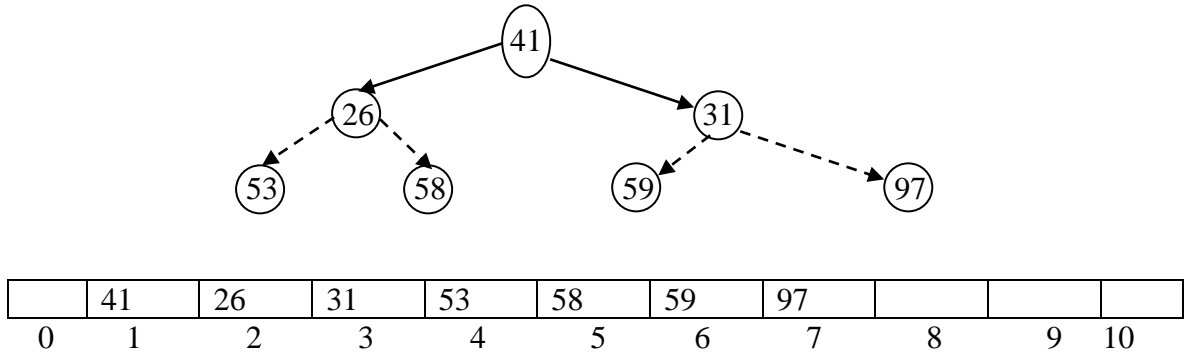
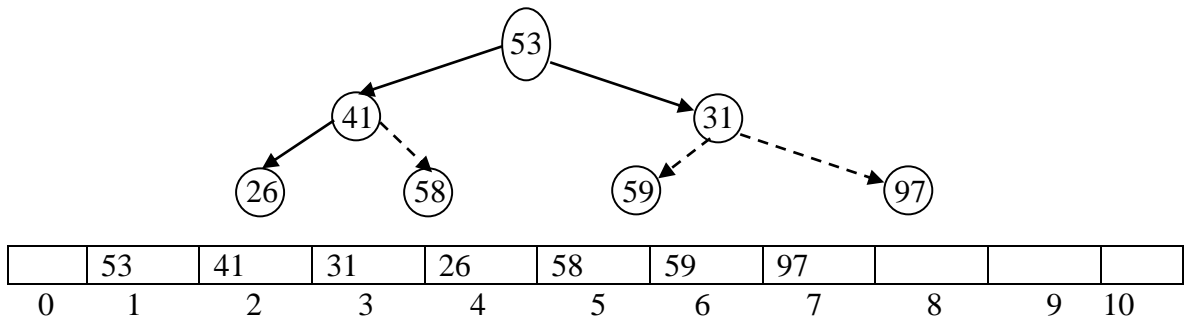
+Heap after first Delete Maximum



	59	53	58	26	41	31	97			
0	1	2	3	4	5	6	7	8	9	10



	58	53	31	26	41	59	97			
0	1	2	3	4	5	6	7	8	9	10



8-7-2 Algorithms (C code) for Heap Sort

```
# define LeftChild( i ) ( 2 * ( i ) + 1 )
void PercDown( ElementType A[], int i, int N )
{
    int Child;
    ElementType Tmp;
    for( Tmp = A[i]; LeftChild( i ) < N; i = Child )
    {
        Child = LeftChild( i );
```

```

        if( Child != N - 1 && A[Child + 1] > A[Child] )
            Child ++;
        if( Tmp < A[Child] )
            A[i] = A[Child];
        else
            break;
    }
    A[i] = Tmp;
}
void Heapsort( ElementType A[], int N )
{
    int i;
    for( i = N/2; i >= 0; i -- )    /* BuildHeap */
        PercDown( A, i, N );
    for( i = N - 1; i > 0; i -- )
    {
        Swap(&A[0], &A[i]);    /* Delete Max */
        PercDown( A, 0, i );
    }
}

```

ឧទាហរណ៍៖

ចូលរសេរកម្មវិធីដើម្បីតំរៀបលេខតាមលំដាប់ពីតូចទៅធំដោយប្រើ Heap Sort។

```

#include< stdio.h >
void main( )
{
    int k[20], i1, n;

    printf("Enter the size of the vector to be sorted\n");
    scanf("%d", &n);
    printf("%d\n", n);

    /* print the elements of the array */
    for(i1 = 1; i1 <= n; ++ i1)
        scanf("%d", &k[i1]);
    printf("\n");

    /* print the sorted elements of the array */
    printf("The sorted vector is :\n");
    for( i1 = 0; i1 <= n; ++ i1)
        printf("%4d", k[i1]);
    printf("\n");
}
crheap(k, n)
int [20], n;
{
    int i, j, key, q;

```

```

for(q = 2; q <= n; ++ q)
{
    i = q;
    key = k[q];
    j = (int)(i/2);

    while((i > 1) && (key > k[ j ]))
    {
        k[i] = k[ j ];
        i = j;
        j = (int)(i/2);
        if(j < 1)
            j = 1;
    }
    k[i] = key;
}
}
/* Function heap sort */
void heapsort(k, n)
{
    int k[20], n;
    int q, temp, i, j, key;
    crheap(k, n);
    for(q = n; q >= 2; -- q)
    {
        temp = k[1];
        k[1] = k[q];
        k[q] = temp;
        i = 1;
        key = k[1];
        j = 2;
        if((j + 1) < q)
            if(k[ j + 1] > k[ j ])
                j = j + 1;
        while((j <= (q-1)) && (k[ j ] > key))
        {
            k[i] = k[ j ];

```

```
        i = j;
        j = 2*I;
        if((j + 1) < q)
            if(k[j + 1] > k[j])
                j = j + 1;
            else if(j > n)
                j = n;
        k[i] = key;
    } /* while */
} /* for */
} /* main */
```

នៅពេល **RUN** កម្មវិធីបង្ហាញលទ្ធផលដូចខាងក្រោម៖

Enter the size of the vector to be sorted

5

Input the given vector:

89 56 5 34 23

The sorted vector is:

5 23 34 56 89

លំហាត់អនុវត្ត

- 1-Discuss the different ideas which lead to Sorting Algorithms.
- 2-Explain any two Internal Sorting with Algorithms.
- 3-Write a C program to sort N numbers using insertion sort.
- 4-Write the algorithm and C program for sorting the numbers in ascending order using Quick Sort Technique.
- 5-Write notes on:

a)-Heap Sort

b)-Merge Sort

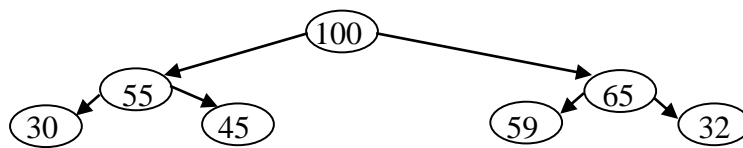
6-គេអោយ List មួយដូចខាងក្រោម៖

40 30 23 17 45 22 75 56 36 65

ចូរសរសេរពីដំណើរការរបស់ Algorithm Selection Sort និង Algorithm Insertion Sort

ក្នុងការ Sort List នេះតាមលំដាប់កើនជាបន្តបន្ទាប់។

7-គេមាន Complete Binary Tree ដូចខាងក្រោម៖



ចូរសរសេរពីដំណើរការរបស់ Algorithm Heap Sort ក្នុងការ Sort List នេះតាមលំដាប់កើនជាបន្តបន្ទាប់។



មេរៀនទី៩

ក្រាហ្វ

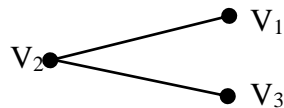
(Graphs)

9-1 សេចក្តីផ្តើម

ក្រាហ្វគឺជាសំណុំនៃ $G(V, E)$ ដែលក្នុងនោះ V គឺជា Vertex (Node) ហើយ E គឺជា Edge(arc) ដែលសម្រាប់ភ្ជាប់ Nodes ។

ឧទាហរណ៍៖

គេមានក្រាហ្វដូចខាងក្រោម៖



- V_1, V_2, V_3 ហៅថា Node ឬ Vertex ហើយបន្ទាត់មកភ្ជាប់រវាង Nodes ពីរហៅថា Edge ។

-បើ (V_1, V_2) ជា Node ដែលអាស្រ័យដោយ E នោះគេថា Edge មួយភ្ជាប់រវាង V_1 និង V_2 ។

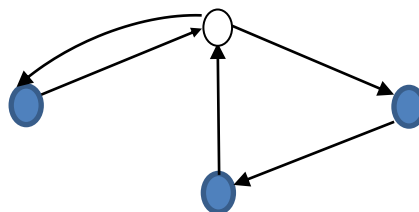
-បើ $Edge:(V_1, V_2)$ ខុសពី (V_2, V_1) នោះក្រាហ្វត្រូវបានគេហៅថា Directed Graph ឬ Digraph ។

ដូចនេះ Digraph គឺជាក្រាហ្វមួយដែល Edge នីមួយៗរបស់វាមានទិសដៅដែលតាងដោយបន្ទាត់ភ្ជាប់មានសញ្ញាក្បាលព្រួញ ។

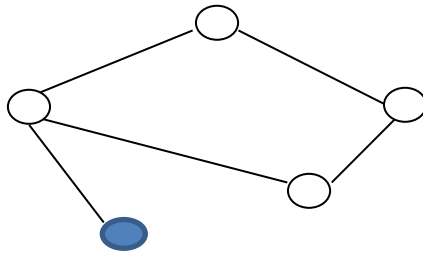
-បើលំដាប់នៃបណ្តា Node ដែលនៅលើ Edge ពុំចាត់ទុកថាសំខាន់នោះក្រាហ្វត្រូវបានគេហៅថា Undirected ។ ដូចនេះ Undirected Graph គឺជាក្រាហ្វដែល Edge របស់វាពុំមានទិសដៅពី Node ណាមួយទៅកាន់ Node ណាមួយ ។

ឧទាហរណ៍៖

គេមានក្រាហ្វពីរដូចខាងក្រោម៖



រូបទី១ ជា Directed Graph(Digraph)



រូបទី២ ជា Undirected Graph

9-2: Graphs និង Multigraphs

+និយមន័យ

ក្រាហ្វ G មួយរួមមានពីរវត្ថុគឺ

(i)-សំណុំ V ដែលមានធាតុត្រូវបានគេហៅថាកំពូល (Vertice), ចំណុច (Points) រឺផ្ទាំង (Nodes)។

(ii)-សំណុំ E នៃគូរផ្សេងគ្នា ហើយគ្មានលំដាប់នៃកំពូលផ្សេងគ្នាហៅថាផ្ទេរ (Edges)។
ដូចនេះគេកំណត់សរសេរក្រាហ្វនេះដោយ $G(V, E)$ ។

+គេថាកំពូល A និង B ជាប់គ្នា (Adjacent) កាលណាវាមានផ្ទេរ $\{A, B\}$ ។

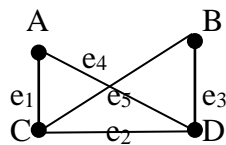
+គេតាងក្រាហ្វដោយដ្យាក្រាមក្នុងប្លង់ដូចខាងក្រោម៖

-កំពូល A នៃសំណុំ V តាងដោយចំណុចមូល \square (dot)។

-ផ្ទេរ $\{A, B\}$ នៃសំណុំ E តាងដោយខ្សែភ្ជាប់រវាងកំពូល A និង B ។

ឧទាហរណ៍

គេមានក្រាហ្វ $G(V, E)$ ដូចខាងក្រោម៖



គេបាន

(i)- $V = \{A, B, C, D\}$

(ii)- $E = \{e_1, e_2, e_3, e_4, e_5\}$

ដែល

$e_1 = \{A, C\}$

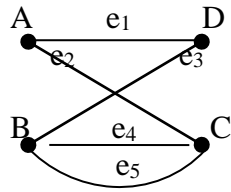
$e_2 = \{C, D\}$

$e_3 = \{B, D\}$

$$e_4 = \{A, D\}$$

$$e_5 = \{B, C\}$$

+ដ្យាក្រាមខាងក្រោមនេះមិនមែនជាគ្រាហ្វទេ



គេហៅថា Multigraph ព្រោះ e_5 ជា Multiple Edge (ផ្ទៃដែលមានកំពូលទាំងពីរដូចគ្នា)។
ដូចនេះគ្រាហ្វមិនមាន Multiple រឺ Loop ទេ។ គេថាគ្រាហ្វគឺជា Multigraph ដែលគ្មាន
Multiple Edges និង Loops ។

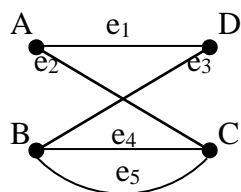
9-3 Degree

9-3-1 និយមន័យ

បើ A ជាចុងកំពូលនៃឆ្នេរ e នោះគេថា e កើតឡើងដោយ A ។
Degree នៃកំពូល A គេកំណត់សរសេរដោយ $\deg(A)$ ហើយវាគឺជាចំនួនឆ្នេរដែលបានកើត
ឡើងដោយកំពូល A ។

9-3-2 ទ្រឹស្តីបទ

ផលបូក Degree នៃកំពូលទាំងអស់នៃគ្រាហ្វស្មើនឹងពីរដងនៃចំនួនរបស់ឆ្នេរ។
ឧទាហរណ៍៖



គេបាន៖

$$-\deg(A) = 2$$

$$-\deg(B) = 3$$

$$-\deg(C) = 3$$

$$-\deg(D) = 2$$

ដូចនេះ

$$\deg(A) + \deg(B) + \deg(C) + \deg(D) = 2 \times 5 \text{ (5edges)} = 10$$

+កំពូលដែលមាន Degree ជាចំនួនគូរត្រូវបានគេហៅថាកំពូលគូរ។

+កំពូលដែលមាន Degree ជាចំនួនសេសត្រូវបានគេហៅថាកំពូលសេស។

នៅក្នុងឧទាហរណ៍ខាងលើគេបាន A, D ជាកំពូលគូរហើយ B, C ជាកំពូលសេស។

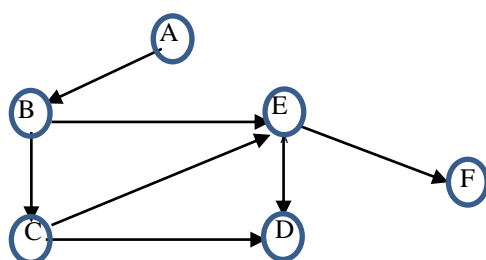
9-3-3 កំណត់សំគាល់

នៅក្នុង Directed ក្រាហ្វ Degree មានពីរគឺ Outdegree និង Indegree។
ដែល

-Outdegree គឺជាចំនួន arcs ដែលមានទិសដៅចេញពី Vertex។

-Indegree គឺជាចំនួន arcs ដែលមានទិសដៅចូលទៅកាន់ Vertex។

ឧទាហរណ៍៖



គេបាន៖

-Indegree របស់ Verticex B ស្មើមួយ ហើយ Outdegree របស់ Vertex B ស្មើពីរ។

-Indegree របស់ Verticex E ស្មើពីរ ហើយ Outdegree របស់ Vertex E ស្មើពីរ។

9-4 ការភ្ជាប់

+និយមន័យ

-Walk នៅក្នុង Multigraph កើតឡើងដោយការឆ្លាស់លំដាប់នៃកំពូល និង ធ្មេរតាមទំរង់ $A_0, e_1, A_1, e_2, A_2, \dots, e_n, A_n$ ដែលធ្មេរ e_i នីមួយៗកើតឡើងដោយកំពូល A_{i-1} និង A_i ។

-ចំនួន n នៃធ្មេរត្រូវបានគេហៅថាប្រវែងនៃ Walk។

-គេអាចកំណត់ Walk មួយដោយលំដាប់នៃធ្មេរ (e_1, e_2, \dots, e_n) រឺដោយលំដាប់នៃកំពូល (A_0, A_1, \dots, A_n) ។

-គេហៅថា Walk បិទ (Closed Walk) កាលណា $A_0 = A_n$ ។

-គេអាចហៅ Walk ថាជា Walk ពី A_0 ទៅ A_n រឺ Walk រវាង A_0 និង A_n រឺ Walk ដែលភ្ជាប់ពី A_0 ទៅ A_n ។

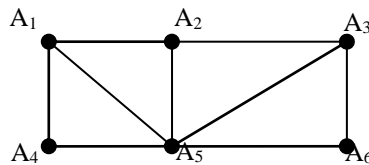
-Trail គឺជា Walk ដែលក្នុងនោះគ្រប់ធ្មេរសុទ្ធតែផ្សេងគ្នា។

-Path គឺជា Walk ដែលក្នុងនោះគ្រប់កំពូលសុទ្ធតែផ្សេងគ្នា។ ដូចនេះគ្រប់ Path សុទ្ធតែជា Trail ។

-Cycle គឺជា Walk មិនដែលគ្រប់កំពូលសុទ្ធតែផ្សេងគ្នាលើកលែងតែ $A_0 = A_n$ បើ Cycle មានប្រវែង K គេហៅ K -Cycle។ នៅក្នុងក្រាហ្វ Cycle សុទ្ធតែមានប្រវែងច្រើនជាង រឺស្មើបី។

ឧទាហរណ៍៖

គេមានក្រាហ្វ $G(V, E)$ ដូចខាងក្រោម៖



$-(A_4, A_1, A_2, A_5, A_1, A_2, A_3, A_6)$ គឺជា Walk ពី A_4 ទៅ A_6 តែវាពុំមែនជា Trail ទេ ព្រោះ Edge $\{A_1, A_2\}$ ប្រើទៀង។

$-(A_4, A_1, A_5, A_2, A_6)$ មិនមែនជា Walk ទេព្រោះវាគ្មាន Edge $\{A_2, A_6\}$ ។

$-(A_4, A_1, A_5, A_2, A_3, A_5, A_6)$ គឺជា Trail ព្រោះវាគ្មាន Edge ប្រើទៀងប៉ុន្តែវាពុំមែនជា Path ទេនៅពេល Vertex A_5 ដែលប្រើទៀង។

$-(A_4, A_1, A_5, A_3, A_6)$ គឺជា Path ពី A_4 ទៅ A_6 ប៉ុន្តែ Path ដែលខ្លីជាងគេបំផុតពី A_4 ទៅ A_6 គឺ (A_4, A_5, A_6) ដែលមានប្រវែងស្មើនឹងពីរ។

9-5 Adjacency Matrix និង Adjacency List

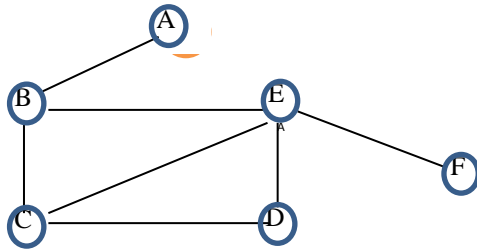
9-5-1 Adjacency Matrix

Adjacency Matrix គឺជា Matrix ដែលកំណត់ដោយ $A = (a_{ij})_{mm}$ ហើយ

$$a_{ij} = \begin{cases} 1 & \text{បើ } \{A_i, A_j\} \text{ គឺជា Edge} \\ 0 & \text{ករណីផ្សេងពីនេះ} \end{cases}$$

-បើ $a_{ij} = a_{ji}$ នោះគេថា A ជា Symmetric Matrix ។

ឧទាហរណ៍៖



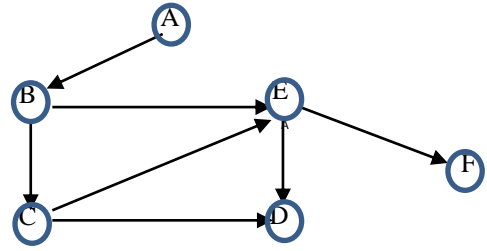
(a)-Adjacency matrix for non-directed graph

(a)-

	A	B	C	D	E	F
A	0	1	0	0	0	0
B	1	0	1	0	1	0
C	0	1	0	1	1	0
D	0	0	1	0	1	0
E	0	1	1	1	0	1
F	0	0	0	0	1	0

(b)-

	A	B	C	D	E	F
A	0	1	0	0	0	0
B	0	0	1	0	1	0
C	0	0	0	1	1	0
D	0	0	0	0	0	0
E	0	0	0	1	0	1
F	0	0	0	0	0	0



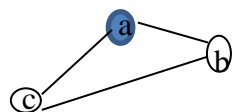
(b)-Adjacency matrix for directed graph

9-5-2: Adjacency List

Adjacency List គឺជាការតាងឱ្យរាល់ Edges រឺ arcs នៅក្នុងក្រាហ្វដូចជា List។ ប្រសិនបើវាជា Undirected ក្រាហ្វនោះធាតុចូលរបស់វាគឺជាសំណុំ Nodes ពីរដែលផ្ទុកចុងបញ្ចប់ពីរដែលអាស្រ័យដោយ Edge។ ប្រសិនបើវាជា Directed ក្រាហ្វនោះរាល់ធាតុចូលរបស់វាគឺជាខ្សែនៃ Nodes ពីរ។ មានន័យថាមួយតំណាងឱ្យប្រភពនៃ Node និងមួយទៀតតំណាងឱ្យទិសដៅរបស់ Node ដែលអាស្រ័យដោយ Edges។

ឧទាហរណ៍១៖

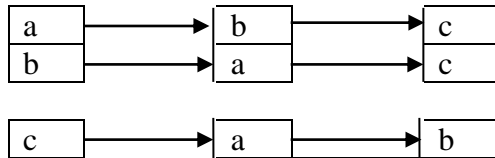
គេមានក្រាហ្វដូចខាងក្រោម៖



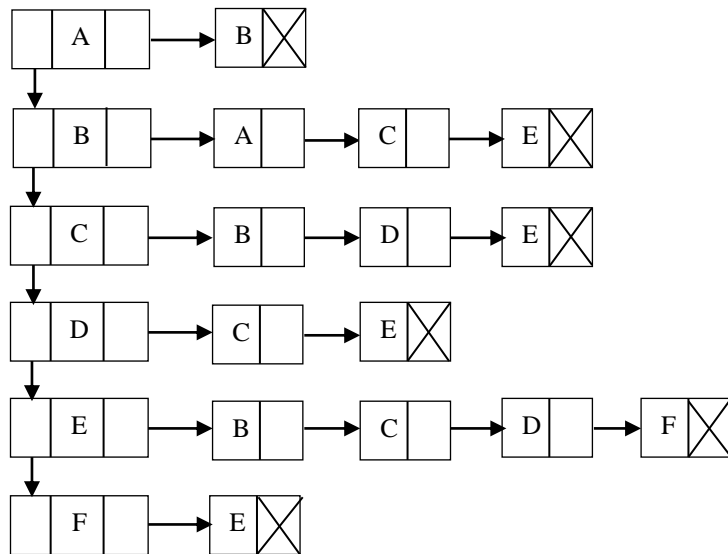
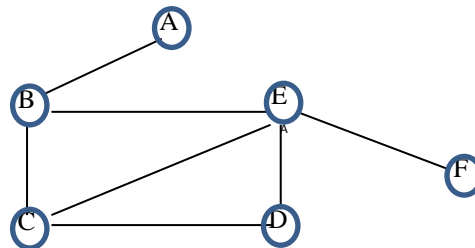
គេបាន Adjacency List:

a	Adjacent to	b, c
b	Adjacent to	a, c
c	Adjacent to	a, b

ឧទាហរណ៍៖



ឧទាហរណ៍២៖



9-6: Traversal

Traversal ចែកចេញជាពីរគឺ Depth-First Traversal និង Breadth-First

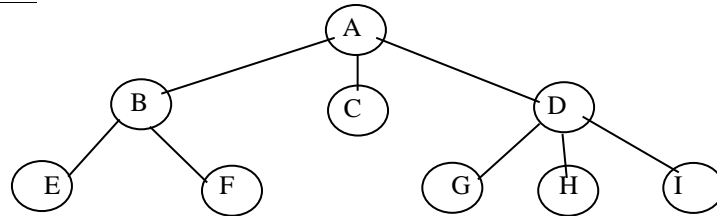
Traversal ។

9-6-1: Depth-First Traversal

នៅក្នុង Depth-First Traversal ដំណើរការរបស់វាមានលក្ខណៈដូច Preorder Traversal ដែរគឺវាចាប់ផ្តើមរាល់ Vertex's descendants មុនពេលដំណើរការ Adjacent Vertex ។ ដំណើរការរបស់វាងាយបំផុត គឺនៅពេលដែលក្រាហ្វជា Tree ។ Depth-First Traversal របស់

ក្រាហ្វមួយ គឺជំណើរការដោយចាប់ផ្តើមពី Vertex ដំបូងនៃក្រាហ្វ បន្ទាប់មក Vertex ដែលភ្ជាប់ទៅនឹង Vertex ដំបូងនោះ ហើយដំណើរការនេះត្រូវបានអនុវត្តជាបន្តបន្ទាប់។ លំដាប់របស់ Adjacent Vertices ត្រូវបានដំណើរការស្រាយទៅតាមរូបរាងរបស់ក្រាហ្វ។

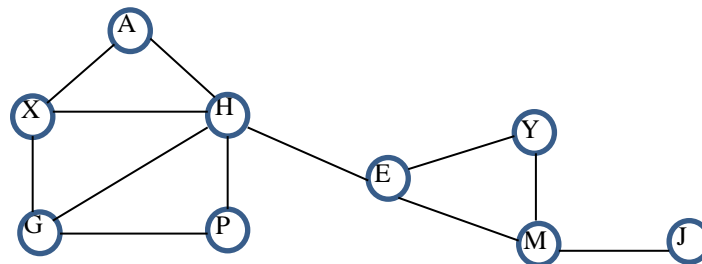
ឧទាហរណ៍ទី១៖



រូបទី១ ជា Depth-first traversal of a tree

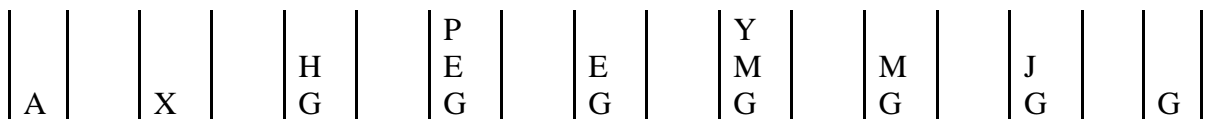
ដូចនេះគេបាន Depth-First Traversal គឺ A B E F C D G H I

ឧទាហរណ៍ទី២៖



(a)-The graph

ដូចនេះគេបាន Depth-First Traversal គឺ A X H P E Y M J G



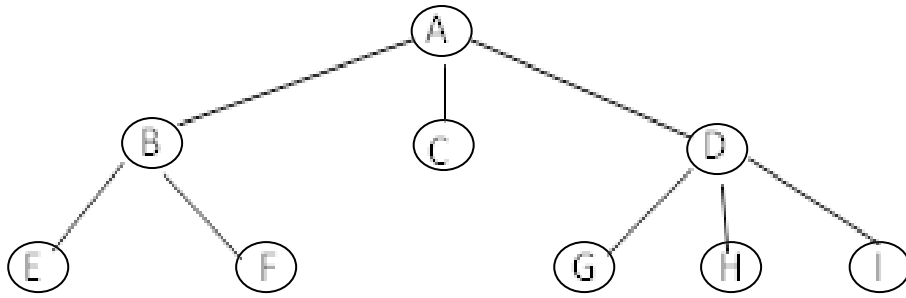
(b)-Stack contents

រូបទី២ ជា Depth-first traversal of a graph

9-6-2 Breadth-First Traversal

នៅក្នុង Breadth-First Traversal របស់ក្រាហ្វមួយដំណើរការរបស់វា គឺចាប់ផ្តើមពី Adjacent Vertices នៃ Vertex ដំបូងបន្ទាប់មកចាប់ផ្តើម Level បន្ទាប់។ មានន័យថាចាប់ផ្តើមពី Level ទី១, Level ទី២ , Level ទី៣ ...។

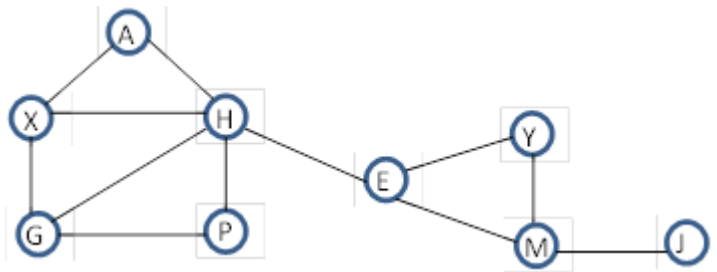
ឧទាហរណ៍ទី១៖



ដូចនេះគេបាន Breadth-First Traversal គឺ A B C D E F G H I

រូបទី១ជា Breadth-first traversal of a tree

ឧទាហរណ៍ទី២៖



(a)-The graph

ដូចនេះគេបាន Breadth-First Traversal គឺ A X G H P E M Y J

A	X	G H	H P	P E	E	M Y	Y J	J
1	2	3	4	5	6	7	8	9

(b)-Queue contents

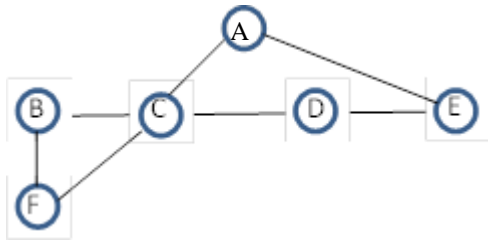
រូបទី២ជា Breadth-first traversal of a graph

9-7 Spanning Trees

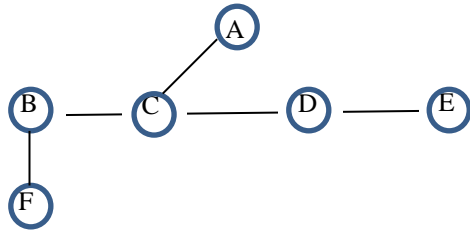
Spanning Tree របស់ក្រាហ្វមួយគឺជា Undirected Tree ដែលផ្ទុក Edges ពិសេស ដើម្បីភ្ជាប់រាល់ Nodes ដែលស្ថិតនៅក្នុងក្រាហ្វដើម។ ចំពោះ Nodes ដែលមាន Path តែមួយ គត់ចន្លោះវា និងការបញ្ចូល Edge ណាមួយទៅក្នុងទំរង់ spanning tree គឺជា unique cycle។ រាល់ Edges ដែលនៅក្រៅនៃ Spanning Tree គឺជំណាងអោយក្រាហ្វដើមដែលភ្ជាប់ Path ជាមួយគ្នានៅក្នុង Tree។

If a DFS is used, those edges traversed by the algorithm form the edges of the tree, referred to as depth first spanning tree. If a BFS is used, the spanning tree is formed from those edges traversed during the search, producing a breadth first spanning tree.

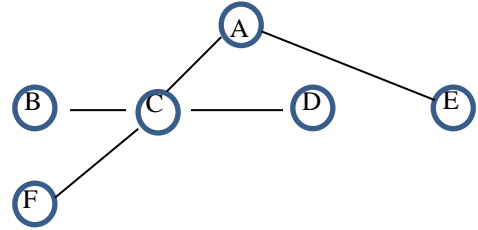
ឧទាហរណ៍១៖



(a)-The graph



(b)-DFS spanning tree

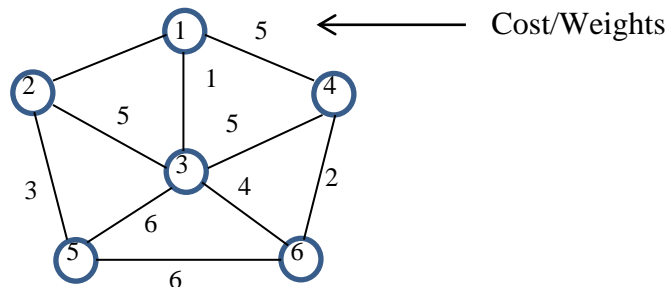


(c)-BFS spanning tree

Network:

A network is a graph that has weights or costs associated with its edges. It is also called weighted graph.

ឧទាហរណ៍២៖



Minimum spanning tree:

This is a spanning tree that covers all vertices of a network such that the sum of the costs of its edges is minimum. There are two algorithms for finding the minimum spanning tree, given a network.

1-Kruskal's algorithm

2-Prim's algorithm

1-Kruskal's algorithm to find the minimum spanning tree:

Step1:

Arrange all edges in ascending order of their cost to form an input set.

Step2:

From this input set, take each edge and if it does not form a cycle, include it in the output set, which forms the minimum spanning tree. The sum of the costs of the edges in the output set is the least.

Note:

If a vertex U is reachable from vertex W and W is reachable from U, then the path between U and w is called a cycle.

+The set [(1, 2), (2, 3), (3, 4), (4, 1)] is a cycle and the superset

[(1, 2), (2, 5), (5, 6), (6, 4), (4, 1)] also is cycle.

+Applying Kruskal's algorithm to the network. We get

-Step1:

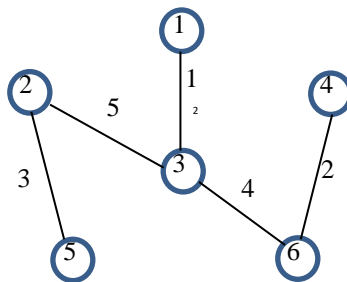
Input set = {(1, 3), (4, 6), (2, 5), (3, 6), (2, 3), (3, 4), (1, 4), (1, 2), (3, 5), (5, 6)}

-Step2:

Output set = {(1, 3), (4, 6), (2, 5), (3, 6), (2, 3)}

Minimum cost = 1 + 2 + 3 + 4 + 5 = 15

Thus, minimum spanning tree is shown:



2-Prim's Algorithm to find minimum spanning tree:

Step1:

Let S be the set of vertices in the spanning tree and A be the set of all vertices in the network. Let E be the set of all edges forming the spanning tree. Initially let $E = \{ \}$ and $S = \{V_1\}$ where V_1 is the starting vertex in A.

Step2:

```
while (S not equal to A) do
{
    find (a, b) to be a low-cost edge such that a is S and b is in (A-S).
    E = E Union {(a, b)}
    S = S Union {b}
}
```

ឧទាហរណ៍៖

We get outputs after applying steps of Prim's algorithm, as follows:

Step1:

$E = \{ \}, \quad S = \{ 1 \}$

$A = \{ 1, 2, 3, 4, 5, 6 \}$

Step2:

Pass 1[of while loop] $\rightarrow E = \{ 1, 3 \}, S = \{ 1, 3 \}$

Pass 2 $\rightarrow E = \{ (1, 3), (3, 6) \}, S = \{ 1, 3, 6 \}$

Pass 3 $\rightarrow E = \{ (1, 3), (3, 6), (6, 4) \}, S = \{ 1, 3, 6, 4 \}$

Pass 4 $\rightarrow E = \{ (1, 3), (3, 6), (6, 4), (3, 2) \}, S = \{ 1, 2, 3, 4, 6 \}$

Pass 2 $\rightarrow E = \{ (1, 3), (3, 6), (6, 4), (3, 2), (2, 5) \}, S = \{ 1, 2, 3, 4, 5, 6 \}$

Now since $S = A$, while loop terminates and E has the set of edges forming the minimum spanning tree.

លំហាត់អនុវត្ត

១-ចូរសង់ក្រាហ្វ G ទៅតាម Adjacency matrix ដូចខាងក្រោម៖

(a)-

	A	B	C	D	E
A	0	1	1	1	0
B	1	0	0	1	0
C	1	0	0	1	1
D	1	1	1	0	1
E	0	0	1	1	0

(b)-

	A	B	C	D	E	F
A	0	1	0	0	0	0
B	0	0	1	0	1	0
C	0	0	0	1	1	0
D	0	0	0	0	0	0
E	0	0	0	1	0	1
F	0	0	0	0	0	0

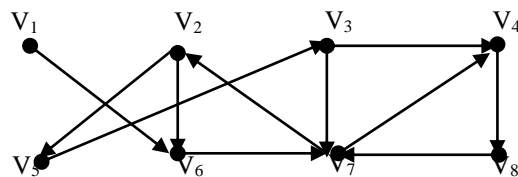
២-ចូរសង់ក្រាហ្វ $G(V, E)$ ទៅតាម multigraph នីមួយៗដូចខាងក្រោម៖

$$V = \{a, b, c, d, e\}$$

$$(ក)-E = [\{a, a\}, \{b, c\}, \{b, d\}, \{c, b\}, \{d, a\}, \{e, d\}]$$

$$(ខ)-E = [\{b, d\}, \{b, c\}, \{c, e\}, \{e, d\}]$$

៣-គេឲ្យក្រាហ្វ G ដូចខាងក្រោម៖

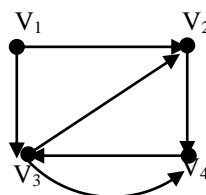


(ក)-ចូររកគ្រប់ Simple paths ពី V_1 ទៅ V_4 ។

(ខ)-ចូររកគ្រប់ Non simple paths ពី V_1 ទៅ V_4 ។

(គ)-ចូររកគ្រប់ Cycles នៅក្នុងក្រាហ្វ G ដែលរួមមាន V_4 ។

៤-គេឲ្យក្រាហ្វ G ដូចខាងក្រោម៖

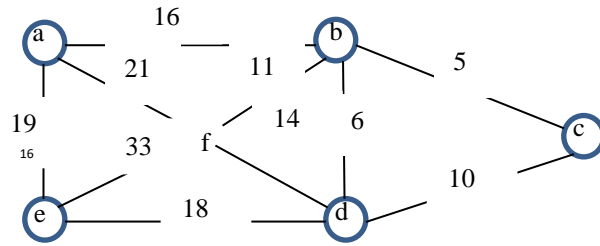


(ក)-ចូររកនូវគ្រប់ Simple paths ពី V_1 ទៅ V_3 ។

(ខ)-ចូររកនូវគ្រប់ Non simple paths ពី V_2 ទៅ V_3 ។

(គ)-ចូររកនូវគ្រប់ Cycles នៅក្នុងក្រាហ្វ G ។

៥-ចូររក Minimum Spanning Tree នៃក្រាហ្វដូចខាងក្រោម៖



៦-គេឱ្យ Binary Tree មួយដែលមាន 10 Nodes។ ដំណើរការរបស់ Preorder និង Inorder

Traversal បង្ហាញដូចខាងក្រោម៖

Preorder: J C B A D E F I G H

Inorder : A B C E D F J G I H

ចូរសង់ Tree។



គន្ថនិទ្ទេស

1-Data Structures and Algorithms Analysis in C (Second Edition)

Mark Allen Weiss.

2-Data Structures Using 'C' (SIKKIM MANIPAL UNIVERSITY)

3-PROGRAMMING WITH PASCAL AND C

(KR VENUGOPAL AND HS VIMALA)