មេរៀនទី ៤

Arrays, Stacks និង Queues

សេចក្តីផ្តើម

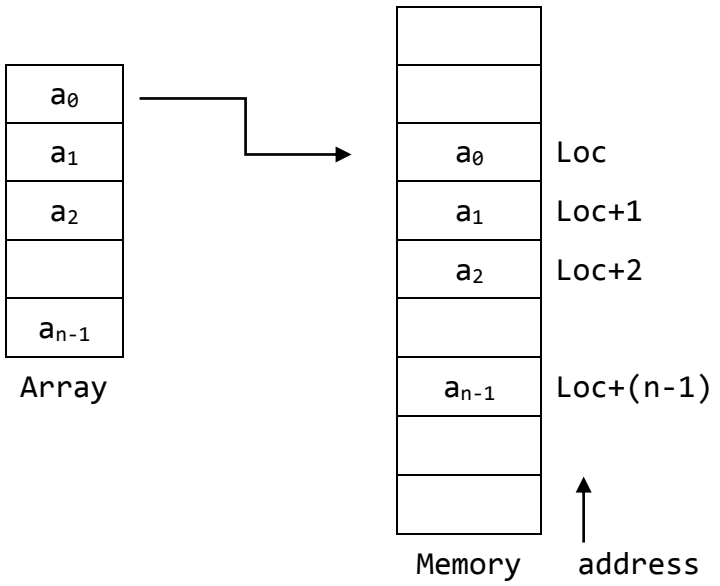
មេរៀននេះ មានគោលបំណងណែនាំឱ្យនិស្សិតស្វែងយល់អំពី៖

- ✓ ការប្រើប្រាស់របស់ Arrays
- ✓ Stacks និង Queues ព្រមទាំងជួយឱ្យនិស្សិតស្វែងយល់ពីការអនុវត្តទៅលើវាផងដែរ

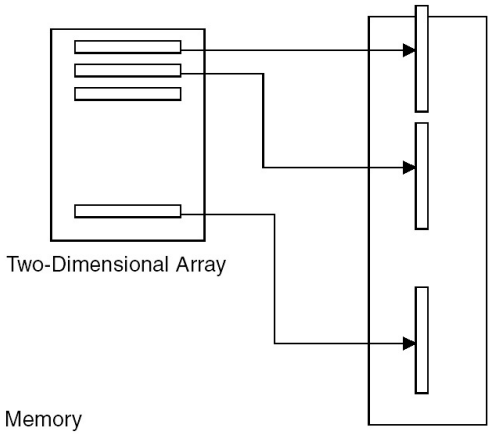
៤.១ Arrays

Array គឺជាសំណុំធាតុដែលមានឈ្មោះរួម មាន Data Type ដូចគ្នា និងមាន Index (Subscript) កំណត់ទុកមុន ហើយរាល់ធាតុនីមួយៗរបស់ Array ដែលមាន n ធាតុត្រូវបានផ្ទុកជាបន្តបន្ទាប់តាមលក្ខណៈរ៉ឹចទ័រពីទីតាំង Index = 0 រហូតដល់ទីតាំងចុងក្រោយរបស់ Array ដែលមាន Index = n-1 (n ជាចំនួនធាតុរបស់ Array)។

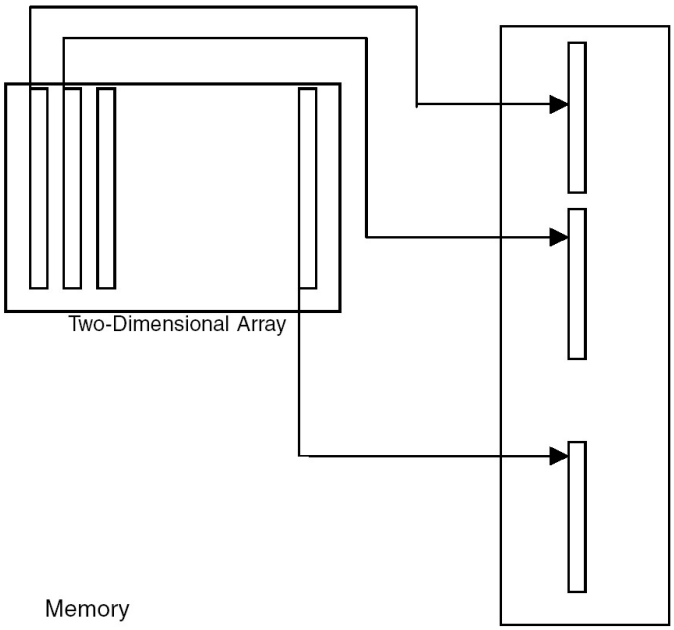
ឧទាហរណ៍ ៤.១៖



រូបភាព ៤.១៖ Representation of an array



រូបភាព ៤.២៖ Row-major representation of a two-dimensional array

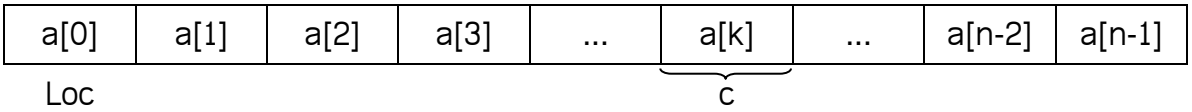


រូបភាព ៤.៣៖ Column major representation of a two-dimensional array

៤.១.១ Array Storage Structure

Array គឺជា Data Structure អាចគណនា Address បាន (Computed Address) និងមានរបៀបផ្ទុកជាបន្តបន្ទាប់ (Sequential Storage Allocation) តាមលក្ខណៈវិចទ័រ គឺធាតុដំបូង a[0] ផ្ទុកត្រង់ទីតាំងដំបូង Index = 0 ជាបន្តបន្ទាប់រហូតដល់ធាតុចុងក្រោយ a[n-1] ផ្ទុកត្រង់ទីតាំង Index = n-1 ដែល n ជាចំនួនធាតុសរុប។

ឧបមាថា យើងមាន Array មួយវិមាត្រ (វិចទ័រ) a[i] ដែលមាន n ធាតុ ហើយរាល់ធាតុនីមួយៗ របស់វាចាប់យក Space memory ទំហំ (c WORD) នោះ a[i] ត្រូវការ memory allocation ស្មើនឹង
(c * n) WORD ជាបន្តបន្ទាប់ និងមានរបៀបផ្ទុករបស់វាដូចខាងក្រោម៖



ដូចនេះ Address របស់ធាតុ a[i] ណាមួយ ត្រូវបានគណនាដោយរូបមន្ត៖

$$Addr(a[i]) = Loc + i*c$$

ដែល $i = 1, 2, \cdots$

📖 កំណត់សម្គាល់

Addr(a[i]) ជា Address នៃធាតុ a[i]

Loc ជា Address ដើម (Base Address ឬ Address នៃធាតុ a[0])

c (constant) ជាទំហំ memory នៃធាតុនីមួយៗរបស់ Array ដែលចាប់យកគិតជា WORD
(1 WORD = 2 Bytes)

i ជា index (Subscript) របស់ Array

ឧទាហរណ៍ ៤.២៖

គេមាន Array៖

int a[7];

ដោយដឹងថា ធាតុដំបូងរបស់ Array នេះ ស្ថិតនៅទីតាំង 100។

ចូរគណនា Address របស់ធាតុ a[3] និង a[6]។

ដំណោះស្រាយ

ដោយធាតុនីមួយៗរបស់ Array នេះ ជាចំនួនគត់ int ចាប់យក Memory ស្មើនឹង 2 Bytes= 1 WORD ($\Rightarrow$ c = 1) និងមាន Address ដើម Loc=100។

គេបាន Address a[3] និង a[6] ដូចខាងក្រោម៖

Addr(a[3])= 100 + 3 * 1=103

Addr(a[6])= 100 + 6 * 1=106

ម៉្យាងទៀត បើយើងមាន Array មួយវិមាត្រ (វិចទ័រ) a[i] ដែលមាន i មានតម្លៃចាប់ផ្តើមពី LB (Lower Bound) ហើយរាល់ធាតុនីមួយៗរបស់វាចាប់យក memory ទំហំ (c WORD) និងមាន address ធាតុដំបូង Loc នោះការគណនា address របស់ធាតុ a[i] ណាមួយត្រូវបានគណនាដោយរូបមន្តខាងក្រោម៖

$$Addr(a[i])=Loc+(i-LB)*c$$

ដែល $i = \mathrm{LB}, \cdots, (n-1)$

ឧទាហរណ៍ ៤.៣៖

ឧបមាថា គេមាន Array a[i] ដែល $3 \le i \le 9$ ហើយរាល់ធាតុនីមួយៗរបស់វាចាប់យក 1 WORD និង address របស់ធាតុដំបូងស្មើនឹង 100។ ចូរគណនា Address របស់ធាតុ a[5] និង a[7]។

ដំណោះស្រាយ

គេមាន c=1 ; Address ធាតុដំបូង Loc=100 និង LB=3

គេបាន Address a[5] និង a[7] ដូចខាងក្រោម៖

Addr(a[5])= 100 + (5-3) * 1=102

Addr(a[7])= 100 + (7-3) * 1=104

ឧបមាថា យើងមាន Array ពីរវិមាត្រ (Matrix ឬ Two Dimensional Array) a[i][j] ដែលមាន m rows និង n columns នោះ យើងមាន របៀបផ្ទុកធាតុនីមួយៗនៃ Matrix a[i][j] តាមលំនាំដូចខាងក្រោម៖

$$\left(a_{[i][j]}\right)_{m\times n} = \begin{pmatrix} a[0][0] & a[0][1] & a[0][2] & a[0][3] \\ a[1][0] & a[1][1] & a[1][2] & a[1][3] \\ a[2][0] & a[2][1] & a[2][2] & a[2][3] \end{pmatrix}_{3\times 4}$$

i) ផ្ទុកតាមជួរដេកជាចម្បង (Row Major Order)

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

ii) ផ្ទុកតាមជួរឈរជាចម្បង (Column Major Order)

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

គេមានរបៀបគណនា address របស់ធាតុ array ពីរវិមាត្រ (Two Dimensional Array) a[i][j] ណាមួយ ត្រូវបានគណនាដោយរូបមន្ត៖

⊙ Row Major Order តាមជួរដេកជាចម្បង

$$Addr(a[i, j])=Loc+(n*i+j)*c$$

ដែល n ជាចំនួន column សរុប

⊙ Column Major Order តាមជួរឈរជាចម្បង

$$Addr(a[i, j])=Loc+(m*j+i)*c$$

ដែល m ជាចំនួន row សរុប

ឧទាហរណ៍ ៤.៤៖

គេមាន Array

int a[7][8];

ដោយដឹងថា address ដើមរបស់ Array នេះមានតម្លៃស្មើនឹង 256 ។

ចូរគណនា Address របស់ធាតុ a[3][5] តាម Row Major Order និង តាម Column Major Order។

ដំណោះស្រាយ

⊙ Row Major Order តាមជួរដេកជាចម្បង

Addr(a[3, 5])= 256 + (8*3+5)*1 =285

⊙ Column Major Order តាមជួរឈរជាចម្បង

Addr(a[3, 5])= 256 + (7*5+3)*1 =294

ជាទូទៅ យើងមានរបៀបផ្ទុកធាតុនីមួយៗនៃ Array ដែលមាន n វិមាត្រ (n Dimensional Arrays) $a[s_0, s_1, s_2, \cdots, s_{n-1}]$ ជាមួយ $LB_i \le s_i \le UB_i$ ដែល LB_i (Lower Bound), UB_i (Upper Bound), s_i (index នៃ Array) និង $i = \overline{0,(n-1)}$ ហើយធាតុនីមួយៗរបស់វាចាប់យក memory ស្មើនឹង c WORD និងមាន address ដំបូង Loc គឺមានការគណនា address របស់ធាតុនីមួយៗនៃ array នេះ និងមានលំនាំផ្ទុក ២ របៀបដូចខាងក្រោម៖

i) ផ្ទុកតាមជួរដេកជាចម្បង (Row Major Order) គឺផ្ទុកអស់ពីជួរដេកមួយចូលដល់ជួរដេកមួយទៀត ដោយអនុវត្តការផ្ទុកជាបន្តបន្ទាប់ពីជួរដេកដំបូងរហូតដល់ជួរដេកចុងក្រោយនៃ Array។

$$Addr(a[s_0, s_1, s_2, \cdots, s_{n-1}]) = Loc + \sum_{i=0}^{n-1} (P_i * (s_i - LB_i) * c)$$

ដែល $P_{n-1} = 1$ និង $P_i = \prod_{k=i+1}^{n-1} (UB_k - LB_k + 1)$

ii) ផ្ទុកតាមជួរឈរជាចម្បង (Column Major Order) គឺផ្ទុកអស់ពីជួរឈរមួយចូលដល់ជួរឈរមួយទៀត ដោយអនុវត្តការផ្ទុកជាបន្តបន្ទាប់ពីជួរឈរដំបូងរហូតដល់ជួរឈរចុងក្រោយនៃ Array។

$$Addr(a[s_0, s_1, s_2, \cdots, s_{n-1}]) = Loc + \sum_{i=n-1}^{0} (P_i * (s_i - LB_i) * c)$$

ដែល $P_0 = 1$ និង $P_i = \prod_{k=i-1}^{0} (UB_k - LB_k + 1)$

ឧទាហរណ៍ ៤.៥៖

គេមាន Array ពីរវិមាត្រ $a[s_0,s_1]$ ដែល $0 \le s_0 \le 2$ និង $0 \le s_1 \le 3$ ហើយរាល់ធាតុនីមួយៗចាប់យក 1 WORD និងមាន address ដើមស្មើនឹង 100។

ចូរគណនា Address របស់ធាតុ a[1 , 3] និង a[2 , 1] តាម Row Major Order និង តាម Column Major Order។

ដំណោះស្រាយ

⊙ Row Major Order តាមជួរដេកជាចម្បង

តាមរូបមន្ត

$$Addr(a[s_0, s_1, s_2, \cdots, s_{n-1}]) = Loc + \sum_{i=0}^{n-1} (P_i * (s_i - LB_i) * c)$$

ដោយ

n = 2

$0 \leq s_0 \leq 2 \Rightarrow LB_0 = 0;\ UB_0 = 2$

$0 \leq s_1 \leq 3 \Rightarrow LB_1 = 0;\ UB_1 = 3$

c = 1 និង

Loc = 100

គេបាន

$$Addr(a[s_0, s_1]) = Loc + \sum_{i=0}^{1} (P_i * (s_i - LB_i) * c)$$

$$Addr(a[s_0, s_1]) = Loc + (P_0 * (s_0 - LB_0) * c) + (P_1 * (s_1 - LB_1) * c)$$

ដែល $P_1 = 1$ (ព្រោះ $P_{n-1} = 1$ ករណីពិសេស) និង $P_i = \prod_{k=i+1}^{n-1} (UB_k - LB_k + 1)$

នាំឲ្យ

$$P_0 = \prod_{k=0+1}^{1} (UB_k - LB_k + 1) = \prod_{k=1}^{1} (UB_k - LB_k + 1)$$

$$P_0 = (UB_1 - LB_1 + 1) = 3 - 0 + 1 = 4$$

ដូចនេះ

Addr(a[1 , 3])= 100+(4 *(1-0) *1)+(1 *(3-0) *1)=107

Addr(a[2 , 1])= 100+(4 *(2-0) *1)+(1 *(1-0) *1)=109

⊙ Column Major Order តាមជួរឈរជាចម្បង

តាមរូបមន្ត

$$Addr(a[s_0, s_1, s_2, \cdots, s_{n-1}]) = Loc + \sum_{i=n-1}^{0} (P_i * (s_i - LB_i) * c)$$

ដោយ

n = 2

$0 \le s_0 \le 2 \Rightarrow LB_0 = 0;\ UB_0 = 2$
$0 \le s_1 \le 3 \Rightarrow LB_1 = 0;\ UB_1 = 3$
c = 1 និង

Loc = 100

គេបាន

$$Addr(a[s_0, s_1]) = Loc + \sum_{i=1}^{0} (P_i * (s_i - LB_i) * c)$$
$$Addr(a[s_0, s_1]) = Loc + (P_1 * (s_1 - LB_1) * c) + (P_0 * (s_0 - LB_0) * c)$$

ដែល $P_0 = 1$ (ករណីពិសេស) និង $P_i = \prod_{k=i-1}^{0} (UB_k - LB_k + 1)$
នាំឲ្យ

$$P_1 = \prod_{k=1-1}^{0} (UB_k - LB_k + 1) = \prod_{k=0}^{0} (UB_k - LB_k + 1)$$

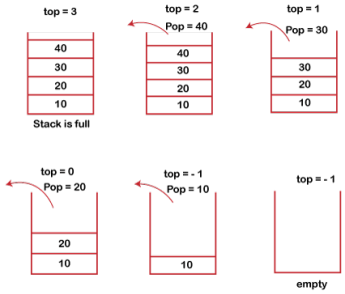
$$P_1 = (UB_0 - LB_0 + 1) = 2 - 0 + 1 = 3$$

ដូចនេះ

Addr(a[1 , 3])= 100+(3 * (3-0) * 1)+(1 * (1-0) * 1)=110

Addr(a[2 , 1])= 100+(3 * (1-0) * 1)+(1 * (2-0) * 1)=105

៤.២ Stack



Stack model: input to a stack is by *Push*, output by *Pop*

Stack គឺជាទម្រង់ List មួយបែប ដែលគេអាចអនុវត្តការបន្ថែមធាតុ (Push) ឬការកាត់បន្ថយធាតុ (Pop) របស់ Stack នៅតែផ្នែកម្ខាងហៅថា កំពូល (Top)។

ដូចនេះ យើងឃើញថា ធាតុដែលបានដាក់ចូលក្នុង Stack ក្រោយគេ ត្រូវបានគេយកចេញមុនគេ។ ហេតុនេះ គេនិយាយថា Stack គឺជា List ពិសេសទម្រង់ LIFO (Last-In, First-Out) មានន័យថា ការងារអនុវត្តការថែមធាតុចូល Stack ឬយកធាតុចេញពី Stack ត្រូវបានអនុវត្តតាមលំនាំ "**ចូលក្រោយយកចេញមុន**"។

ការធ្វើប្រមាណវិធីលើ Stack

ដើម្បីធ្វើប្រមាណវិធីលើ Stack គេត្រូវអនុវត្តនូវប្រមាណវិធីមួយចំនួនដូចខាងក្រោម៖

- initialize( ) សម្រាប់បង្កើត Stack ទទេ (Creates an empty Stack.)
- **isFull**( ) សម្រាប់ត្រួតពិនិត្យមើល Stack ពេញ ឬទេ? (Tests if this stack is full. Returns: true if and only if this stack contains items; false otherwise.)
- **isEmpty**( ) សម្រាប់ត្រួតពិនិត្យមើល Stack ទទេដែរ ឬទេ? (Tests if this stack is empty. Returns: true if and only if this stack contains no items; false otherwise.)
- **peek**( ) សម្រាប់ត្រួតពិនិត្យមើលធាតុកំពុងស្ថិតនៅកំពូល Stack (Returns the value at the top of the Stack without removing it.)
- **push**( ) សម្រាប់បន្ថែមធាតុចូលកំពូល Stack (Adds values into the Stack. It allows value of any datatype.)
- **pop**( ) សម្រាប់ដកយកធាតុចេញពីកំពូល Stack (Removes and returns the value at the top of the Stack or the value that was added last to the Stack.)
- **contains**(): Checks whether the specified item exists in a Stack collection or not. It returns true if it exists; otherwise it returns false.
- clear(): Removes all the values from the stack.
- count(): Gets the number of elements contained in the Stack.

Add Values into Stack

The Push() method adds values into the Stack. It allows value of any datatype.

Push() method signature: void Push(object obj);

Peek()
The Peek() method returns the last (top-most) value from the stack. Calling Peek() method on empty stack will throw InvalidOperationException. So always check for elements in the stack before retrieving elements using the Peek() method.

Peek() method signature: object Peek();

Pop()
You can also retrieve the value using the Pop() method. The Pop() method removes and returns the value that was added last to the Stack. The Pop() method call on an empty stack will raise an InvalidOperationException. So always check for number of elements in stack must be greater than 0 before calling Pop() method.
Pop() signature: object Pop();

Contains()
The Contains() method checks whether the specified item exists in a Stack collection or not. It returns true if it exists; otherwise it returns false.
Contains() method signature: bool Contains(object obj);

Clear()
The Clear() method removes all the values from the stack.
Clear() signature: void Clear();

Properties
Count Gets the number of elements contained in the Stack.

Methods
Clear() Removes all objects from the Stack.

Contains(Object) Determines whether an element is in the Stack.

Peek() Returns the object at the top of the Stack without removing it.

Pop() Removes and returns the object at the top of the Stack.

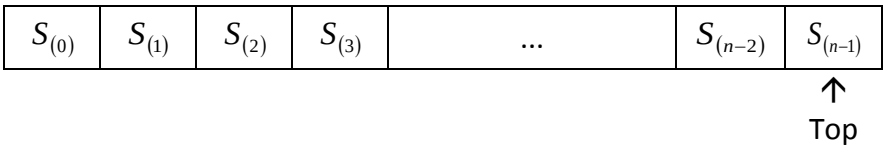
Push(Object) Inserts an object at the top of the Stack.

ទម្រង់ផ្ទុកធាតុរបស់ Stack ដោយ Array

គេអាចបង្កើត Stack ដោយរបៀបប្រកាស Array មួយវិមាត្រជាមួយទំហំ Index ជា MaxStack។

ដូចនេះ Stack អាចផ្ទុក n ធាតុ ពី Index = 0 ដល់ Index = (maxStack – 1) ដែលស្ថិតនៅទីតាំងកំពូលរបស់ Stack គឺ top មានន័យថា នៅក្នុង Stack កំពុងផ្ទុក (top + 1) ធាតុ។

ដើម្បីប្រកាស Stack មួយ យើងចាំបាច់ប្រើ Array មួយវិមាត្រ S ប្រើអញ្ញាត top សម្រាប់ចង្អុលកំពូល Stack និងអញ្ញាត n សម្រាប់កំណត់ទំហំ Index របស់ Stack ដែលមានរបៀបផ្ទុកដូចខាងក្រោម៖



ការបង្កើត Data Structure សម្រាប់ Stack

```
#define SIZE 5 //The maximum size of the stack
struct Stack {
      int top;
      int node[SIZE];
};
```

បណ្តា Algorithms **សម្រាប់អនុវត្ត** Stack

Algorithm Initialize សម្រាប់បង្កើត Stack ទទេ

```
void initialize (struct Stack *stk){
      stk -> top = -1 ;
}
```

Algorithm isFull សម្រាប់ពិនិត្យលើ Stack ថាតើ Stack ពេញឬទេ?

```
int isFull(struct Stack *stk){
      if( stk -> top == SIZE - 1 ) return 1;
      else return 0;
}
```

Algorithm isEmpty សម្រាប់ពិនិត្យលើ Stack ថាតើ Stack ទទេឬទេ?

```
int isEmpty(struct Stack *stk){
      if( stk -> top == -1 ) return 1;
      else return 0;
}
```

Algorithm peek (Stack Top) សម្រាប់បង្ហាញតម្លៃក្នុង Stack ត្រង់ទីតាំង top

```
int peek (struct Stack *stk){
```

```
        if( stk -> top == -1 )
              printf("\nStack underflow.\n");
        else return stk -> node[stk -> top];
}
```

Algorithm Push **សម្រាប់បន្ថែមធាតុថ្មីចូលក្នុង** Stack

ដើម្បីធ្វើការបន្ថែមធាតុ item ចូលទីតាំងកំពូល Stack គេត្រូវអនុវត្តន៍

- ប្រសិនបើ Stack ពេញ នោះបង្ហាញប្រាប់ថា "Stack Overflow"
- ប្រសិនបើ Stack មិនទាន់ពេញ នោះត្រូវអនុវត្តដូចខាងក្រោម៖
  - បង្កើនមួយតម្លៃទៅឲ្យ top ដោយសរសេរ top = top + 1
  - កំណត់តម្លៃថ្មីនៅត្រង់ទីតាំងកំពូល Stack ដោយសរសេរ node[top] = item

```
void push (struct Stack *stk, int item){
        if( stk -> top == SIZE - 1 )
              printf("\nStack overflow.\n");
        else{
              stk -> top = stk -> top + 1;
              stk -> node[stk -> top] = item;
        }
}
```

ឬ

```
void push (struct Stack *stk, int item){
        if( isFull(stk) )
              printf("\nStack overflow.\n");
        else{
              stk -> top = stk -> top + 1;
              stk -> node[stk -> top] = item;
        }
}
```

Algorithm Pop **សម្រាប់លុបធាតុចេញពី** Stack

ដើម្បីធ្វើដកយកធាតុ ពីទីតាំងកំពូល Stack គេត្រូវអនុវត្តន៍

- ប្រសិនបើ Stack ទទេ នោះបង្ហាញប្រាប់ថា "Stack Underflow"
- ផ្ទុយមកវិញ នោះត្រូវអនុវត្តដូចខាងក្រោម៖
  - ដកយកធាតុត្រង់ទីតាំង top ដោយសរសេរ element = node[top]
  - បន្ថយតម្លៃ top មួយ ដោយសរសេរ top = top - 1

```
int pop(struct Stack *stk){
```

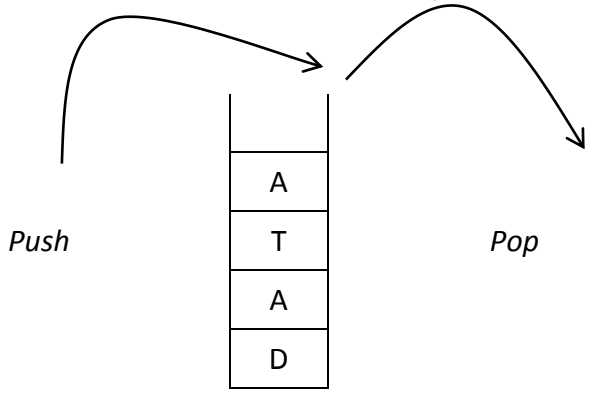
```
        int element;
        if( stk -> top == -1 ) printf("\nStack underflow.\n");
        else{
              element = stk -> node[stk -> top] ;
              stk -> top = stk -> top - 1;
              return element;
        }
}
```

ឬ

```
int pop(struct Stack *stk){
        int element;
        if( isEmpty(stk) )
              printf("\nStack underflow.\n");
        else{
              element = stk -> node[stk -> top] ;
              stk -> top = stk -> top - 1;
              return element;
        }
}
```

ឧទាហរណ៍អនុវត្តលើ Stack

ការបង្ហាញចេញតម្លៃបញ្ច្រាសរបស់ស្ទឹតតួអក្សរដែលបានកំណត់ ឬបញ្ចូលពី Keyboard។ ចូរប្រើលក្ខណៈរបស់ Stack សម្រាប់សរសេរនូវ Algorithm និងកម្មវិធី (Coding in C Language) ដើម្បីធ្វើការបង្ហាញចេញនូវតម្លៃបញ្ច្រាសរបស់ស្ទឹតតួអក្សរដែលបានកំណត់ ឬបញ្ចូលពី Keyboard។



INPUT: DATA**$** *Stack* *OUTPUT:* ATAD

Algorithm **សម្រាប់ដោះស្រាយចំណោទខាងលើ**

ដើម្បីដោះស្រាយចំណោទបញ្ហានេះ ដោយប្រើ Stack គេមាន Algorithm សម្រាប់ដោះស្រាយតាមដំណាក់កាលដូចខាងក្រោម៖

- បង្កើត Stack ទទេ
- ប្រសិនបើ Stack មិនទាន់ពេញ នោះអនុវត្ត

- កំណត់ ឬបញ្ចូលតម្លៃស្ទឹតតួអក្សរពី Keyboard
- ដាក់ស្ទឹតតួអក្សរនីមួយៗចូលក្នុង Stack

- ប្រសិនបើ Stack ពេញ នោះអនុវត្ត
  - ដកយកធាតុនីមួយៗចេញពីក្នុង Stack
  - បង្ហាញតម្លៃធាតុនីមួយៗ ដែលបានដកចេញពីក្នុង Stack លើ Screen

Coding in C Language (Program)

```
//Stack.c
#include <stdio.h>
#include <conio.h>
#define MAX_LEN 1000
#define EMPTY -1
#define FULL (MAX_LEN-1)
typedef enum boolean {false, true} boolean;
typedef struct stack {
      char s[MAX_LEN];
      int top;
}stack;
void reset(stack *stk){
      stk->top = EMPTY;
}

void push(stack *stk, char c){
      stk->top++;
      stk->s[stk->top] = c;
}
char pop(stack *stk){
      return (stk->s[stk->top--]);
}
char top(const stack *stk){
      return (stk->s[stk->top]);
}
boolean empty(const stack *stk){
      return ((boolean) (stk->top == EMPTY));
}
boolean full(const stack *stk){
      return ((boolean) (stk->top == FULL));
}
int main(void){
      char str[] = "My name is Chen Wen Ping!";
      int i;
      stack s;
      clrscr();
      reset(&s);
      printf(" In the string: %s\n", str);
      for(i = 0; str[i] != '\0'; ++i)
            if(!full(&s))
            push(&s, str[i]);
```

```
        printf("From the stack: ");
        while(!empty(&s))
              putchar(pop(&s));
        putchar('\n');
        getch();
        return 0;
  }
```