

ວິຊາ ໂຄງສ້າງຂອງຂໍ້ມູນ ແລະ ຂັ້ນຕອນວິທີສຳລັບການພັດທະນາເວັບໃຊ 2  
   
 **ກຸ່ມ 2**

ນາງ ມ່ວນທະລາ ສີພັນທອງ

ນາງ ດຽວຄຳ ໃຫຍ່ວິດວ

ນາງ ບີວ່າງ

ທ້າວ ສຸວັນ

ພຣະ ບຸນຄ້ຳ

ທ້າວ ນູຊົ່ວ  
 ພຣະ ມີໄຊ

ທ້າວ ຕູ້ຢ່າງທໍ່ຕູ້ ຈົ່ງສືຢ່າງ

ທ້າວ ນອ້ຍຈັນຫານ

ຫ້ອງ 2cw1

ໂດຍສອນ: ອ ຈ ຈິດນະວັນ

ສາຂາພັດທະນາເວບໄຊ

ຄະນະວິທະຍາສາດທຳມະຊາດ

ມະຫາວິທະຍາໄລແຫ່ງຊາດ

ສົກສຶກສາ 2021-20

ສາລະບານ

[ບົດທີ່ 4 Stack ແລະ Queue 1](#_Toc75423836)

[4.1 Function Push (e) 1](#_Toc75423837)

[4.2 Function PoP() 2](#_Toc75423838)

[4.3 ການນຳໄຊ້ Stack 2](#_Toc75423839)

[4.3.2 ສຳນວນແບບ Infix ແລະ Postfix 4](#_Toc75423840)

[4.4 ຄົວ (Queue) 5](#_Toc75423841)

[4.4.1 ຟັງຊັນ Enqueue 6](#_Toc75423844)

[4.4.2 ຟັງຊັນ Dequeue 7](#_Toc75423845)

[4.4.3 ຟັງຊັນ Queue Front 8](#_Toc75423846)

[4.4.4 ຟັງຊັນ Queue Rear 9](#_Toc75423847)

[4.5 Abstr Data Type ຂອງ Queue (Queue ADT) 10](#_Toc75423848)

[4.5.1 ການອອກແບບຄິວດ້ວຍ Array (Queue Array Design 11](#_Toc75423849)

[ບົດທີ່ 5 (Linear Lists) 12](#_Toc75423850)

[5.1 ແນວຄິດພື້ນຖາມກຽ່ວກັບລາຍການແແບບເສັ້ນຊື່ (Linear list concepts) 12](#_Toc75423851)

[5.2 ລາຍການເຊື່ອມຕໍ່ແບບເສັ້ນຊື່ດ່ຽວ (Single Linked List) 12](#_Toc75423852)

[5.2.1 ການດຳເນິນງານພື້ນຖານຂອງລິສ (Basic Operations) 13](#_Toc75423853)

[5.2.1.1 ການເພີ່ມເຂົ້າ 14](#_Toc75423854)

[5.2.1.2 ການເພີ່ມເຂົ້າໂນດທີ່ຕຳແໝ່ງທຳອິດ(Insert at Beginning) 14](#_Toc75423855)

[5.2.1.3 ການເພີ່ມເຂົ້າໂນດທາງທ້າຍ( Insert at End ) 15](#_Toc75423856)

[5.2.2 ການລືບໂນດ (Delete Node) 16](#_Toc75423857)

[5.2.2.1 ການລືບໂນດທີ່ຕຳແໜ່ງທຳອິດ (Delete first Node ) 17](#_Toc75423858)

[5.2.2.2 ການລືບໂນດທີ່ຕຳແໜ່ງສຸດທ້າຍ (Delete at end) 18](#_Toc75423859)

[5.3 ລາຍການເຊື່ອມຕໍ່ແບບເສັ້ນຊື່ ຊະນິດອື່ນໆ(Others Linked Lists) 18](#_Toc75423860)

[5.3.1 ລາຍການເຊື່ອມຕໍ່ແບບວົງມົນ (Circular-linked List) 19](#_Toc75423861)

[5.3.2 ການເພີ່ມເຂົ້າ node ທາງດ້ານໜ້າ 20](#_Toc75423862)

[5.3.3 ການເພີ່ມເຂົ້າ Rear node 20](#_Toc75423863)

[5.3.4 ລາຍການເຊື່ອມຕໍ່ແບບເສັ້ນຊື່ຄູ່ (Doble Linked List) 21](#_Toc75423864)

[5.3.5 ການເພີ່ມເຂົ້າໂນດໃນ (Double Linked List) 21](#_Toc75423865)

[5.3.6 ການລືບໂນດໃນ (Double Linked List) 21](#_Toc75423866)

# ບົດທີ່ 4 Stack ແລະ Queue

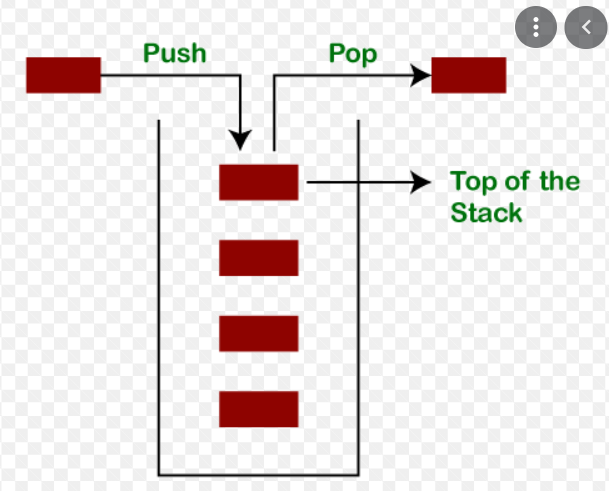
ໃນວິທະຍາສາດຄອມພິວເຕີ, stack ແມ່ນປະເພດຂໍ້ມູນທີ່ບໍ່ມີຕົວຕົນເຊິ່ງເຮັດ ໜ້າ ທີ່ເກັບ ກຳ ຂໍ້ມູນຂອງອົງປະກອບ, ໂດຍມີສອງປະຕິບັດການຫລັກຄື: Push, ເຊິ່ງເພີ່ມອົງປະກອບເຂົ້າໃນການເກັບ ກຳ ຂໍ້ມູນ, ແລະ. Pop, ເຊິ່ງ ກຳ ຈັດອົງປະກອບທີ່ເພີ່ມເຂົ້າມາ ໃໝ່ ທີ່ສຸດທີ່ຍັງບໍ່ໄດ້ເອົາອອກເທື່ອ.

## 4.1 Function Push (e)

ການປະຕິບັດການຍູ້ຫມາຍເຖິງການໃສ່ອົງປະກອບໃນຊັ້ນ. ເນື່ອງຈາກວ່າມັນມີພຽງແຕ່ ຕຳ ແໜ່ງ ໜຶ່ງ ທີ່ອົງປະກອບ ໃໝ່ ສາມາດໃສ່ໄດ້ - ເທິງສຸດຂອງຊັ້ນ, ສ່ວນປະກອບ ໃໝ່ ຈະຖືກໃສ່ຢູ່ເທິງສຸດຂອງ stack ການປະຕິບັດງານ POP. ການເຮັດວຽກຂອງ Pop ແມ່ນ ໝາຍ ເຖິງການ ກຳ ຈັດອົງປະກອບໃດ ໜຶ່ງ.

push (E element) ວິທີການແມ່ນໃຊ້ເພື່ອຍູ້ອົງປະກອບ ໜຶ່ງ ເຂົ້າໃນ Stack. ອົງປະກອບດັ່ງກ່າວຖືກຍູ້ລົງເທິງສຸດຂອງ Stack. ພາລາມິເຕີ: ວິທີການຍອມຮັບເອົາ ໜຶ່ງ ພາລາມິເຕີຂອງ type Stack ແລະ ໝາຍ ເຖິງອົງປະກອບທີ່ຈະຖືກຍູ້ລົງໃນຊັ້ນ ມູນຄ່າການກັບຄືນ: ວິທີການສົ່ງຄືນການໂຕ້ຖຽງທີ່ຜ່ານໄປ.

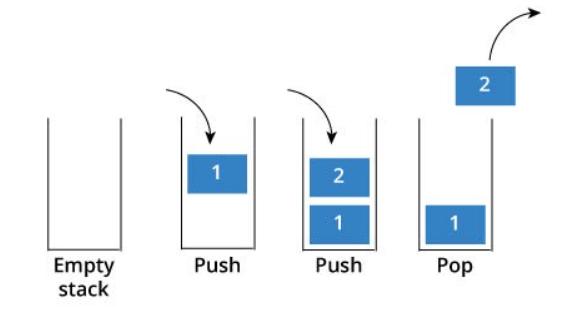
ຟັງຊັນ PUSH ໃນລະຫັດແມ່ນໃຊ້ໃນການໃສ່ອົງປະກອບ ໜຶ່ງ ໄປທາງເທີງຂອງ stack, ໜ້າ ທີ່ POP ໃຊ້ເພື່ອເອົາອົງປະກອບອອກຈາກສ່ວນເທິງຂອງ stack. ... ໜ້າ ທີ່ທຸກຂັ້ນຕອນຖືກຈັດຕັ້ງປະຕິບັດໃນລະຫັດ C



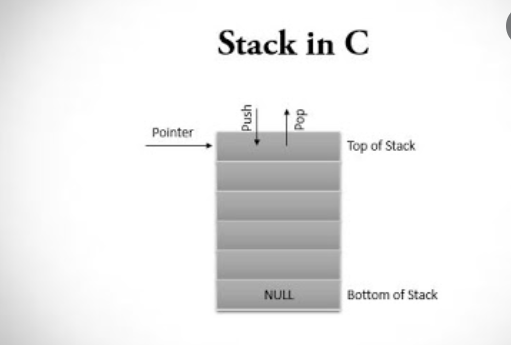
## 4.2 Function PoP()

ຟັງຊັນ Pop () ຖືກໃຊ້ເພື່ອເອົາອົງປະກອບ ໜຶ່ງ ອອກຈາກສ່ວນເທິງຂອງ stack (ອົງປະກອບ ໃໝ່ ສຸດໃນ stack) ອົງປະກອບທີ່ຖືກຍ້າຍອອກໄປໃນຖັງບັນຈຸ stack ແລະຂະຫນາດຂອງ stack ໄດ້ຫຼຸດລົງ

pop () ວິທີການ. ວິທີການນີ້ບໍ່ ຈຳ ເປັນຕ້ອງມີພາລາມິເຕີໃດໆແລະມັນກໍ່ ກຳ ຈັດອົງປະກອບທີ່ຢູ່ເທິງສຸດຂອງວາງ. ມັນສົ່ງຄືນອົງປະກອບທີ່ຖືກຖອດອອກ



## 4.3 ການນຳໄຊ້ Stack

Stack ແມ່ນໂຄງສ້າງຂໍ້ມູນເຊິ່ງໃຊ້ເພື່ອເກັບຂໍ້ມູນຕາມ ລຳ ດັບສະເພາະ. ການປະຕິບັດງານສອງຢ່າງທີ່ສາມາດປະຕິບັດງານໄດ້ໃນ Stack ແມ່ນ: ການ ດຳ ເນີນງານທີ່ກົດດັນເຊິ່ງຈະໃສ່ອົງປະກອບເຂົ້າໃນຂັ້ນໄດ ການປະຕິບັດງານ Pop ເຊິ່ງເອົາອົງປະກອບສຸດທ້າຍທີ່ຖືກເພີ່ມເຂົ້າໃນ stack. ມັນປະຕິບັດຕາມ ຄຳ ສັ່ງ First First First (LIFO).

4.3.1 ໂປຣແກຣມແບບRecursive (Recursion Program)

#include <stdio.h>

int sum(int n);

int main() {

int number, result;

printf("Enter a positive integer: ");

scanf("%d", &number);

result = sum(number);

printf("sum = %d", result);

return 0;

}

int sum(int n) {

if (n != 0)

// sum() function calls itself

return n + sum(n-1);

else

return n;

}

ຜົນ run of the project top

Enter a positive integer:3

sum = 6

ຄຳອະທິບາຍຂອງ program

nitially, the sum () is called from the main () function with number passed as an argument.

Suppose, the value of n inside sum () is 3 initially. During the next function call, 2 is passed to the sum () function. This process continues until n is equal to 0.

When nis equal to 0, the if condition fails and the else part is executed returning the sum of integers ultimately to the main () function.

## 4.3.2 ສຳນວນແບບ Infix ແລະ Postfix

ການປ່ຽນແປງ ສຳ ນວນ infix ກັບ postfix ຄໍາສັ່ງຂອງຜູ້ປະຕິບັດງານຕ້ອງເປັນທີ່ຮູ້ຈັກກ່ອນ.ຄໍາສັ່ງຂອງການດໍາເນີນງານຂອງຜູ້ປະຕິບັດງານກ່ຽວກັບເລກຈາກສູງຫາຕໍ່າ.

1. ວົງເລັບ (ບໍ່ແມ່ນຜູ້ປະກອບການ)

2. ຜູ້ປະຕິບັດການພະລັງງານ (^)

3. ຜູ້ປະຕິບັດການຄູນ (\*) ແລະພະແນກ (/)

4. ຜູ້ປະກອບເພີ່ມ (+) ແລະເຄື່ອງຍ່ອຍ (-)

ບ່ອນທີ່ຜູ້ປະຕິບັດງານຂອງຄໍາສັ່ງດຽວກັນດໍາເນີນການຈາກຊ້າຍຫາຂວາ.ຍົກເວັ້ນ ສຳ ລັບຜູ້ປະຕິບັດງານທີ່ ກຳ ລັງເຮັດວຽກຈາກຂວາຫາຊ້າຍ.

ຍົກຕົວຢ່າງ

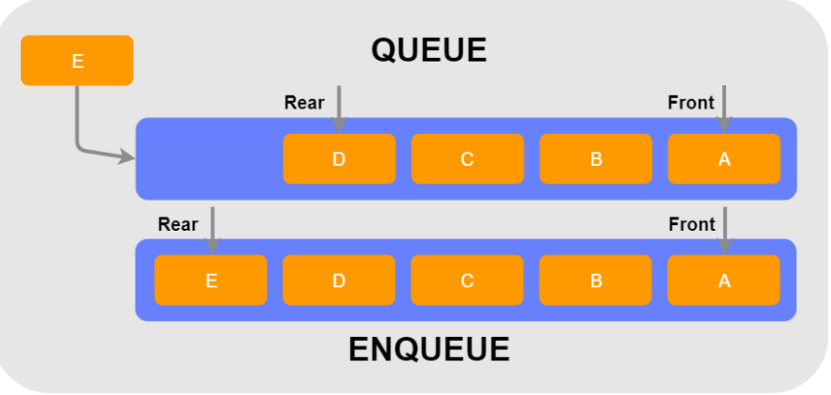
A + B + C ໝາຍ ຄວາມວ່າ (A + B) + C.

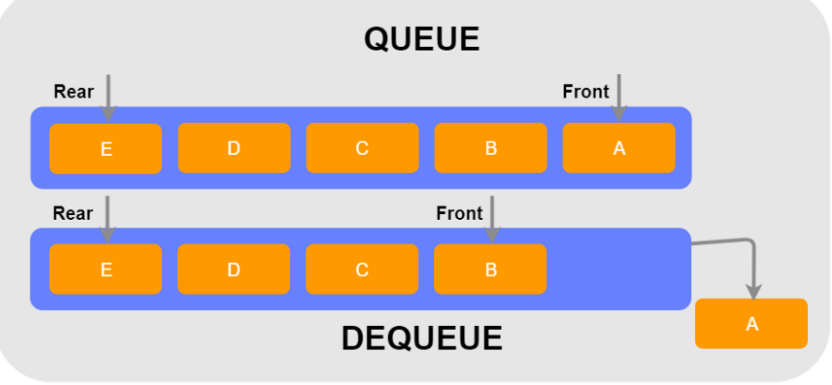
A ^ B ^ C ໝາຍ ຄວາມວ່າ A ^ (B ^ C)

## 4.4 ຄົວ (Queue)

* Queue ເປັນໂຄງສ້າງຂໍ້ມູນແບບເສັ້ນຊື່ (linear data structure) ເຊິ່ງຄ້າຍຄືກັບ stack ແຕ່ບໍ່ຄືກັນໝົດ, queue ຈະເປີດທັງຫົວແລະປາຍ, ສົ້ນ ໜຶ່ງ ແມ່ນໃຊ້ເພື່ອໃສ່ຂໍ້ມູນ (Enqueue) ສ່ວນທ້າຍອື່ນໆແມ່ນໃຊ້ ສຳ ລັບການລຶບຫລືລຶບຂໍ້ມູນ (Dequeue).

ໃນການ dequeue data ຕົວ pointer ຈະຊື້ໄປທີ່ “front” and Enqueing data ຕົວ pointer ຈະຊີ້ໄປທີ່ “rear”.

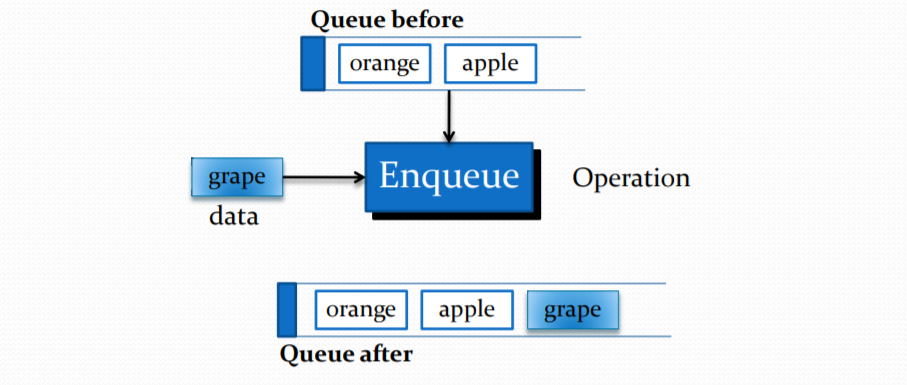
* Queue ແມ່ນອີງໃສ່ຫຼັກການ First First First First (FIFO), ມັນ ໝາຍ ຄວາມວ່າສິ່ງໃດທີ່ເຂົ້າໄປກ່ອນກໍ່ອອກໄປກ່ອນ Imagine, ຕົວຢ່າງ, ຖະ ໜົນ ໜຶ່ງ ແລວ ໜຶ່ງ ເສັ້ນເມື່ອທ່ານຈອດລົດໄຟແດງ ທຳ ອິດ, ລົດຕໍ່ໆມາຕິດຕາມທ່ານເມື່ອທ່ານ ແສງສີຂຽວແມ່ນກ່ຽວກັບ. ຈະຕ້ອງໄປກ່ອນ ຫລືວາງແຖວຂຶ້ນເພື່ອຮັບເງິນຟຣີ ຖ້າທ່ານຄິວກ່ອນ ທ່ານຕ້ອງໄປກ່ອນມັນງ່າຍທີ່ຈະຈື່ໄດ້ວ່າທ່ານຕິດຕາມແຖວ.



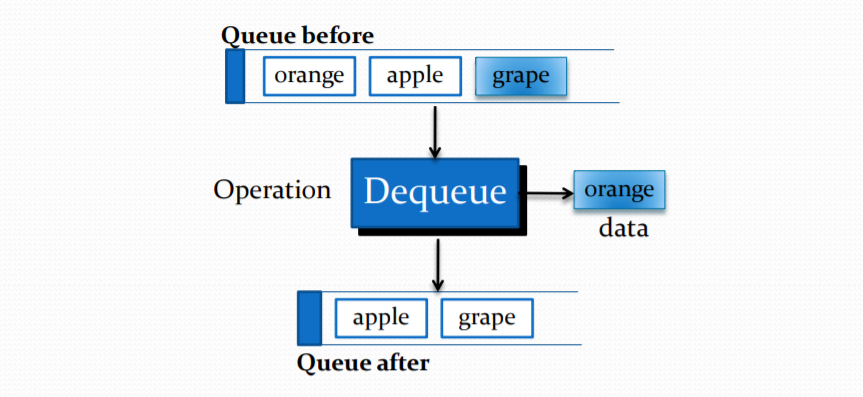
## 

## 4.4.1 ຟັງຊັນ Enqueue

* ເປັນການເພີ່ມສະມາຊິກດ້ວຍນຳໄປຕໍ່ສ່ວນທ້າຍຄິວທີ່ເລີ່ມກວ່າ Raer
* ຫາກພື້້້້ນທີ່ບໍ່ພຽງພໍຕ້ອງການເພີ່ມສະມາຊິກໃໝ່ເຂົ້າໄປໃນຄິວກໍຈະເກີດສະຖານະ Overflow

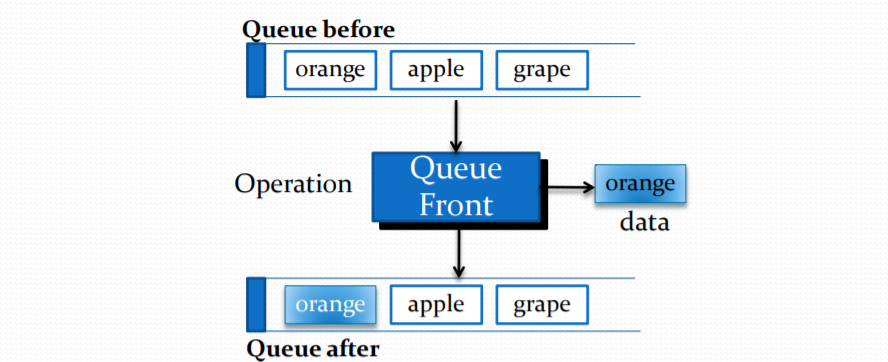


## 4.4.2 ຟັງຊັນ Dequeue

* ເປັນການລືບສະມາຊິກທີຕຳແໜ່ງສ່ວນຫົວທີ່ເລີ່ມກ່ວາ front
* ຫາກກາຍໃນຄີວບໍ່ມີຂໍ້ມູນກໍຈະທຳໃຫ້ເກີດສະຖານະ Underflow

## 4.4.3 ຟັງຊັນ Queue Front

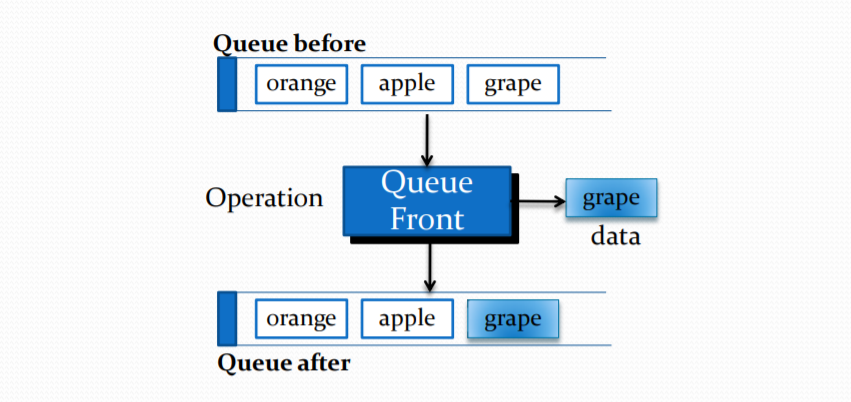
## Queue Front ຈະທຳການດື່ງຂໍ້ມູນສະມາຊິກໂດຍກົງອອກມາໃຊ້ງານ

* ຫາກກາຍໃນຄີວບໍ່ມີຂໍ້ມູນກໍຈະທຳໃຫ້ເກີດສະຖານະ Underflow

## 4.4.4 ຟັງຊັນ Queue Rear

## Queue Rear ຈະທຳການດື່ງຂໍ້ມູນສະມາຊິກໂດຍກົງອອກມາໃຊ້ງານ

* ຫາກກາຍໃນຄີວບໍ່ມີຂໍ້ມູນກໍຈະທຳໃຫ້ເກີດສະຖານະ Underflow



## 4.5 Abstr Data Type ຂອງ Queue (Queue ADT)

ປະເພດຂໍ້ມູນແບບຫຍໍ້ແມ່ນ ຄຳ ນິຍາມຂອງປະເພດ ໃໝ່ ທີ່ອະທິບາຍກ່ຽວກັບຄຸນສົມບັດແລະການ ດຳ ເນີນງານຂອງມັນ.ໂຄງສ້າງຂໍ້ມູນແມ່ນການຈັດຕັ້ງປະຕິບັດ ADT, ຫຼາຍ ADT ສາມາດ ນຳ ໃຊ້ກັບໂຄງສ້າງຂໍ້ມູນດຽວກັນ.ຖ້າຂ້ອຍຄິດວ່າມັນຖືກຕ້ອງ, ຂບວນແມ່ນ ADT, ໝາຍ ຄວາມວ່າເປັນການລວບລວມອົງປະກອບ, ແລະມັນແມ່ນໂຄງສ້າງຂໍ້ມູນໃນການເກັບຮັກສາໃນຄວາມຊົງ ຈຳ. ແຕ່ພວກເຮົາສາມາດເວົ້າກ່ຽວກັບໂຄງສ້າງຂອງຂໍ້ມູນ stack ຖ້າຂ້ອຍຫມາຍຄວາມວ່າບໍ? ແລະເປັນຫຍັງ heap ຈຶ່ງບໍ່ແມ່ນ ADT ມັນສາມາດໃຊ້ເປັນຕົ້ນໄມ້ຫລືຂບວນ.

## 4.5.1 ການອອກແບບຄິວດ້ວຍ Array (Queue Array Design)

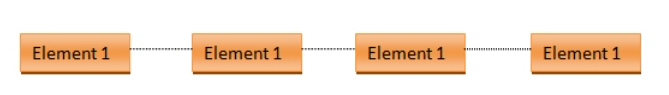
ສຳ ລັບໂຄງສ້າງຂໍ້ມູນແຕ່ລະແຖວ, ພວກເຮົາ ຈຳ ເປັນຕ້ອງເກັບຄ່າຄຸນຄ່າຕໍ່ໄປນີ້:

Queue [] ອາເລທີ່ເກັບສະມາຊິກແຕ່ລະຄົນໃນແຖວ. ດ້ານໜ້າ ຕຳ ແໜ່ງ ຂອງຫົວທ້າຍຂອງແຖວ.ຕຳ ແໜ່ງ ຂອງທ້າຍແຖວ.size ຈຳ ນວນສະມາຊິກທັງ ໝົດ ທີ່ຢູ່ໃນແຖວ.

# ບົດທີ່ 5 (Linear Lists)

## 5.1 ແນວຄິດພື້ນຖາມກຽ່ວກັບລາຍການແແບບເສັ້ນຊື່ (Linear list concepts)

ການຈັດຮຽງ ລຳ ດັບຂອງຂໍ້ມູນພາຍໃນບັນຊີທີ່ເປັນ ລຳ ດັບ. ສາມາດອະທິບາຍໄດ້ສະມາຊິກແຕ່ລະຄົນຫຼືອົງປະກອບໃດ ໜຶ່ງ ເຊື່ອມໂຍງກັບອົງປະກອບຕໍ່ໄປເປັນລາຍຊື່ຕໍ່ເນື່ອງ.ຕົວຢ່າງ, ອົງປະກອບທີສອງແມ່ນຢູ່ຕິດກັບອົງປະກອບ ທຳ ອິດ. ຫຼືອົງປະກອບທີສາມແມ່ນຕໍ່ໄປຈາກອົງປະກອບທີສອງ ເຊິ່ງຈະ ດຳ ເນີນຕໍ່ໄປໃນ ຄຳ ສັ່ງນີ້ ຈົນກ່ວາອົງປະກອບລໍາດັບ n + 1 ເຊິ່ງຈະຢູ່ຖັດຈາກອົງປະກອບ nth ແລະກັບຊັບສິນນັ້ນພວກເຮົາເອີ້ນມັນວ່າບັນຊີ.



## 

## 5.2 ລາຍການເຊື່ອມຕໍ່ແບບເສັ້ນຊື່ດ່ຽວ (Single Linked List)

* ເປັນໂຄງສ້າງທີ່ໃຊ້ node. ໃຫ້ເຊື່ອມຕໍ່ເຊິ່ງກັນແລະກັນເປັນບັນຊີລາຍຊື່, ເຊິ່ງແຕ່ລະ node ມີສ່ວນແມ່ນຂໍ້ມູນ (ຂໍ້ມູນ) ແລະສ່ວນ ໜຶ່ງ ທີ່ເຊື່ອມຕໍ່ກັບ node ຕໍ່ໄປ (link) ຮູບ 1.1 ສະແດງລາຍຊື່ທາງໜຶ່ງ.
* ການອອກແບບ node ໃຊ້ສ່ວນທີ່ມີຊື່ຂໍ້ມູນເປັນພາຊະນະຂໍ້ມູນແລະສ່ວນຕໍ່ໄປເປັນການເຊື່ອມຕໍ່ ສຳ ລັບ node. ບັນຊີລາຍຊື່ທີ່ເຊື່ອມໂຍງຢ່າງເປັນຮູບສະແດງ 1.2 ສະແດງສ່ວນປະກອບຂອງໂຫນດແລະຂໍ້ສຸດທ້າຍຂອງບັນຊີ.

ຮູບທີ 1.2 ໂຄງສ້າງຂອງ node [data ແລະ link] ແລະວິທີການສະແດງຂໍ້ສຸດທ້າຍຂອງລາຍຊື່ທີ່ເຊື່ອມໂຍງໃນຕາຕະລາງ.

ພວກເຮົາຈະໃຊ້ສອງ Nodes ພິເສດເພື່ອຊີ້ໄປທີ່ node ທາງ ໜ້າ ແລະຂໍ້ສຸດທ້າຍຢູ່ທາງລຸ່ມບັນຊີລາຍຊື່ແມ່ນຫົວແລະຫາງ. ການອອກແບບແບບນີ້ຈະເປັນໄປໄດ້ ການໃຫ້ການເຂົ້າເຖິງຂໍ້ມູນທັງທາງ ໜ້າ ແລະດ້ານຫຼັງ (ຂໍ້ເສັ້ນທາງ ທຳ ອິດຂອງສອງເສັ້ນທາງດັ່ງທີ່ສະແດງໃນຮູບ 1.1) ແມ່ນເປັນໄປໄດ້ໃນເວລາດຽວກັນ ສຳ ລັບ node ທີ່ຫາງຖືກຊີ້ແລະມັນແມ່ນຂໍ້.ໃນຕອນທ້າຍຂອງບັນຊີພວກເຮົາຈະ ພວກເຮົາລະບຸວ່າມັນຊີ້ໄປທີ່ null (ຫຼື nil), ເຊິ່ງພວກເຮົາໃຊ້ແທນØ.ເພື່ອສ້າງບັນຊີລາຍຊື່, ຫົວແລະຫາງຊີ້ໃຫ້ເຫັນຢູ່ສະ ເໝີ. ກຳ ນົດເປັນລາຍຊື່ທີ່ຍັງບໍ່ມີຂໍ້ມູນ (empty list).

## 5.2.1 ການດຳເນິນງານພື້ນຖານຂອງລິສ (Basic Operations)

ລາຍຊື່ການປະຕິບັດງານຂັ້ນພື້ນຖານ ມັນປະກອບດ້ວຍການແຊກ (ການແຊກ), ການລຶບ (ລຶບ).

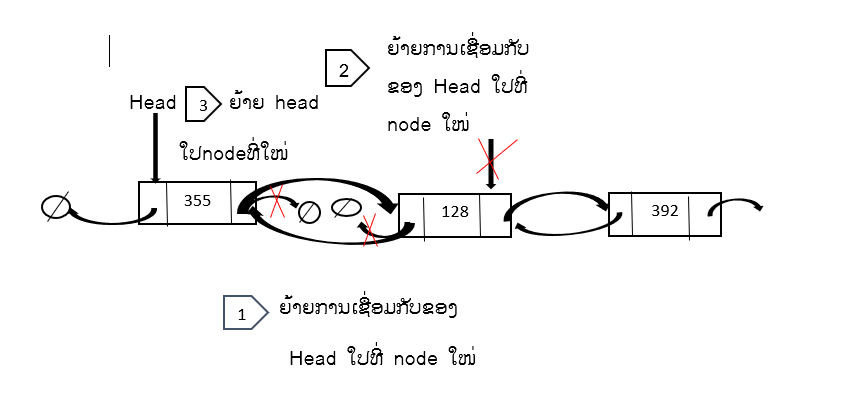
ການອ່ານ (ດຶງຂໍ້ມູນ) ແລະຜ່ານລາຍຊື່ (Traversal) ໂດຍການໃສ່ແມ່ນການເພີ່ມສະມາຊິກ.

ເຂົ້າ ໃໝ່ ການລຶບແມ່ນ ກຳ ຈັດສະມາຊິກອອກຈາກບັນຊີ. ການອ່ານແມ່ນ ດຳ ເນີນໃນແຕ່ລະ Eli.

ຄຳ ເຫັນພາຍໃນລາຍການຕາມ ລຳ ດັບ, ເຊັ່ນການຊອກຫາຜ່ານບັນຊີເພື່ອຊອກຫາຜົນລວມຂອງຊັ້ນຮຽນດິບຂອງນັກຮຽນແລະ ຄຳ ນຶງເຖິງຄະແນນສະເລ່ຍ, ອື່ນໆ.

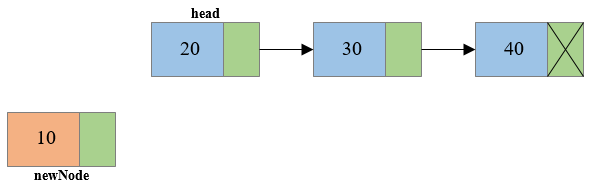
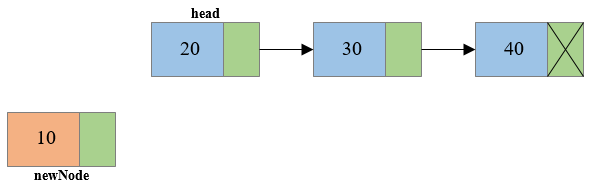
## 5.2.1.1 ການເພີ່ມເຂົ້າ

ໃນເວລາທີ່ການ ນຳ ເຂົ້າຂໍ້ມູນເຂົ້າໃນບັນຊີທີ່ເຊື່ອມໂຍງ, ພວກເຮົາມີແນວໂນ້ມທີ່ຈະ ນຳ ເຂົ້າມັນຢູ່ທາງຫລັງ (ບ່ອນທີ່ຫາງ ກຳ ລັງຊີ້), ແຕ່ວ່າ ມັນສາມາດຖືກນໍາເຂົ້າຢູ່ທາງຫນ້າຫຼືຢູ່ສະຖານທີ່ອື່ນໆໃນບັນຊີທີ່ເຊື່ອມໂຍງເຊັ່ນກັນ.ຄຳ ນຶງເຖິງສອງຄະດີຕົ້ນຕໍ: head point to null, ເຊິ່ງ ໝາຍ ເຖິງລາຍຊື່ທີ່ເຊື່ອມໂຍງເຊິ່ງບໍ່ມີຂໍ້ມູນເລີຍ, ຫຼື ບັນຊີລາຍຊື່ທີ່ເຊື່ອມໂຍງມີຂໍ້ມູນທີ່ສາມາດໃຊ້ໄດ້ ການ ນຳ ເຂົ້າແມ່ນການເຂົ້າຮ່ວມ node ໃໝ່ ເຂົ້າໃນບັນຊີລາຍຊື່ທີ່ມີຈຸດຊີ້ຫົວຫລືຫາງ. ໄປທີ່ໂຫນດທີ່ ນຳ ເຂົ້າມາ ໃໝ່ ນີ້.

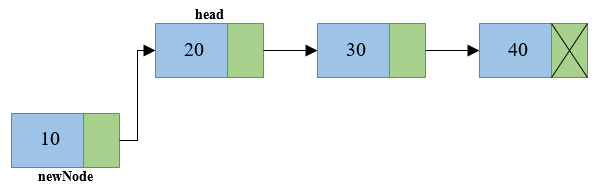


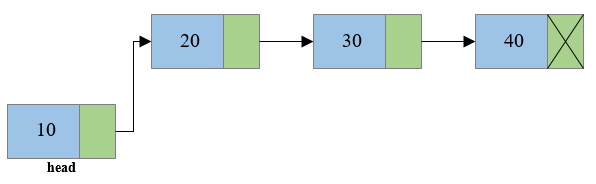
## 5.2.1.2 ການເພີ່ມເຂົ້າໂນດທີ່ຕຳແໝ່ງທຳອິດ(Insert at Beginning)

ຂັ້ນ​ຕອນ​ໃນ​ການ​ແທກ​ໂຫນດ​ໃຫມ່​ທີ່​ຈຸດ​ເລີ່ມ​ຕົ້ນ​ຂອງ​ລາຍ​ການ​ທີ່​ເຊື່ອມ​ໂຍງ.

1.ສ້າງ​ໂຫນດ​ໃຫມ່​ຊີ້​ໄປ​ທີ​ໂຫນດ​ສ້າງ​ຂື້ນ​ໃຫມ່.

1. ເຊື່ອມ​ໂຍງ​ໂຫນດ​ທີ່​ສ້າງ​ຂື້ນ​ໃຫມ່​ກັບ​ໂຫນດ​ຫລັກ​ເຊັ່ນ: ໂຫນດ​ໃຫມ່​ຈະ​ຊີ້​ໄປ​ທີ່​ຫົວໂຫນດ.



3.ສ້າງ​ໂຫນດ​ໃຫມ່​ເປັນ​ໂຫນດຫຼັກ​ເຊັ່ນ: ຫົວ​ໂຫນດ​ຈະ​ຊີ້​ໄປ​ໂຫນດ​ໃຫມ່.

* ໂປ​ຣ​ແກ​ຣມ​ແທກ​ໂຫນດ​ທີ່​ຈຸດ​ເລີ່ມ​ຕົ້ນ.

#include <stdio.h>

#include <stdlib.h>

/\* Structure of a node \*/

struct node {

int data; // Data

struct node \*next; // Address

}\*head;

void createList(int n);

void insertNodeAtBeginning(int data);

void displayList();

int main()

{

int n, data;

/\*

\* Create a singly linked list of n nodes

\*/

printf("Enter the total number of nodes: ");

scanf("%d", &n);

createList(n);

printf("\nData in the list \n");

displayList();

/\*

\* Insert data at the beginning of the singly linked list

\*/

printf("\nEnter data to insert at beginning of the list: ");

scanf("%d", &data);

insertNodeAtBeginning(data);

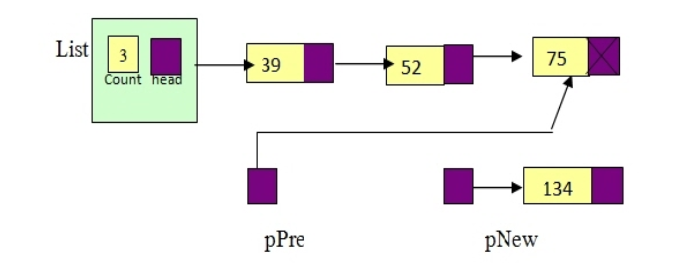
## 

## 5.2.1.3 ການເພີ່ມເຂົ້າໂນດທາງທ້າຍ( Insert at End )

ເມື່ອ node ຖືກເພີ່ມໃສ່ໃນຕອນທ້າຍຂອງບັນຊີ ພວກເຮົາພຽງແຕ່ຕ້ອງການຕົວຊີ້ບອກກ່ອນ.

ເພື່ອຊີ້ໄປທີ່ຂໍ້ ໃໝ່ ໃນນັ້ນບໍ່ມີ node Successor ເພາະວ່າມັນຖືກໃສ່ໃນຕອນທ້າຍບັນຊີລາຍຊື່, ດັ່ງນັ້ນພາກສະຫນາມເຊື່ອມຕໍ່ຂອງໂຫນດ ໃໝ່ ຖືກ ກຳ ນົດໃຫ້ບໍ່ມີປະໂຫຍດ

ເຖິງຢ່າງໃດກໍ່ຕາມ, ມັນມີອີກປະເພດ ໜຶ່ງ ຂອງເຫດຜົນພິເສດເພື່ອ ນຳ ໃຊ້ກັບລະບົບການແຊກຂອງຂໍ້ມູນໃນຕອນທ້າຍຂອງບັນຊີ, ເຊິ່ງແມ່ນ ໜຶ່ງ ໃນຂໍ້ດີຂອງໂຄງສ້າງບັນຊີລາຍຊື່ການເຊື່ອມໂຍງ. ໃຫ້ຮູ້ແລ້ວວ່າຂໍ້ສຸດທ້າຍຂອງບັນຊີລາຍຊື່ມີຕົວຊີ້ທີ່ກ່ຽວຂ້ອງກັບ null, ໝາຍ ຄວາມວ່າຂໍ້ສຸດທ້າຍທີ່ບໍ່ມີ node ຕໍ່ໄປ.ຖ້າພວກເຮົາມີຄວາມຕ້ອງການໃຊ້ຕົວຊີ້ບອກແທນທີ່ຈະກ່ວາຄ່າຄົງທີ່.



## 5.2.2 ການລືບໂນດ (Delete Node)

ຕ້ອງມີການຄຳນວນສຳ ລັບການເອົາຂໍ້ມູນອອກຈາກບັນຊີນອກ ເໜືອ ໄປຈາກການສົ່ງຂໍ້ມູນທີ່ຖືກລຶບໄປໃນຄວາມຊົງ ຈຳຈື່ງສາມາດຂໍ້ມູນການລະບົບດັ່ງນັ້ນມັນສາມາດຖືກນໍາໃຊ້ໃນຄໍາຮ້ອງສະຫມັກອື່ນໆ. ຕົວຊີ້ ໃໝ່ ຍັງຕ້ອງໄດ້ປັບປ່ຽນ. ສຳ ລັບຂັ້ນຕອນທຳ ອິດຂອງການລຶບ node ຂໍ້ທີ່ຕ້ອງຖືກລຶບ (pLoc) ຕ້ອງຖືກພົບເຫັນຢູ່ໃນບັນຊີກ່ອນ.

ສະຖານທີ່ຂອງຂໍ້ທີ່ຕ້ອງລຶບໄດ້ຖືກພົບເຫັນຢູ່ໃນບັນຊີ. ນີ້ຈະຊ່ວຍໃຫ້ທ່ານຮູ້ສະຖານທີ່ຂອງທີ່ຢູ່ຂອງຜູ້ຖືຮຸ້ນກ່ອນ.

(pPre), ເຊິ່ງແມ່ນຂໍ້ທີ່ຢູ່ເບື້ອງຕົ້ນຂອງຂໍ້ທີ່ຈະຖືກລຶບອອກ. ຫລັງຈາກນັ້ນ, ມັນໄດ້ ກຳ ນົດເຂດເຊື່ອມຕໍ່ຂອງ

Node ຂອງ Predecessor ຊີ້ໃຫ້ເຫັນ Successor node ເຊິ່ງແມ່ນຂໍ້ທີ່ຢູ່ເບື້ອງຫຼັງຂອງ node ທີ່ຖືກລຶບອອກ. ມັນຈະສົ່ງຄືນພື້ນທີ່ໜ່ວຍ ຄວາມ ຈຳ ທີ່ເກັບຂໍ້ມູນທີ່ຖືກລຶບຖິ້ມກັບໄປຍັງລະບົບເພື່ອ ນຳ ໃຊ້ຕໍ່ໄປ.

## 5.2.2.1 ການລືບໂນດທີ່ຕຳແໜ່ງທຳອິດ (Delete first Node )

ເມື່ອສະຖານທີ່ ທຳ ອິດ (pLoc) ຮູ້ແລ້ວ, ຫຼັງຈາກນັ້ນປັບຫົວຫົວຄືນເພື່ອຊີ້ໄປທີ່ຂໍ້.ຜູ້ສືບທອດຕໍ່ໄປຈາກຂໍ້ ທຳ ອິດທີ່ຈະລຶບ. ຫຼັງຈາກນັ້ນ, ໂຫນດທີ່ຖືກລຶບອອກຈະຖືກສົ່ງກັບຄືນສູ່ລະບົບແລະເນື່ອງຈາກວ່າໃນຄໍາສັ່ງທີ່ຈະເອົາ node ທໍາອິດອອກຈາກບັນຊີ, ໂຫນດ Predecessor (pPre) ກ່ອນມັນ.ສະນັ້ນ, pPre node ຖືກ ກຳ ນົດໃຫ້ null, ນັ້ນ ໝາຍ ຄວາມວ່າມັນຈະລຶບ node ຢູ່ ຕຳ ແໜ່ງ ທຳ ອິດຢູ່ທີ່ນັ້ນ

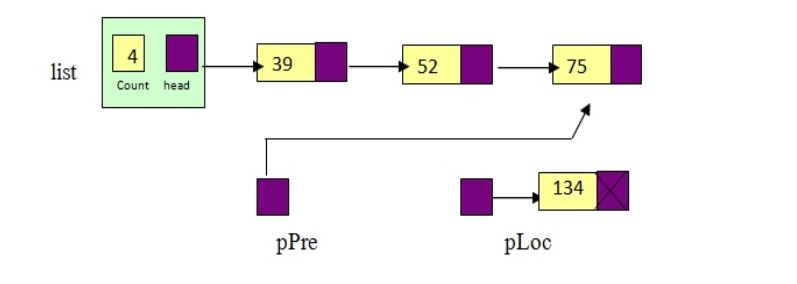
## 5.2.2.2 ການລືບໂນດທີ່ຕຳແໜ່ງສຸດທ້າຍ (Delete at end)

ການລຶບຂໍ້ອອກຈາກບັນຊີລາຍຊື່ໃນກໍລະນີທົ່ວໄປ ນີ້ປະກອບດ້ວຍໃນການຖອດຂໍ້ທີ່ອອກເປັນຈຸດສູນກາງ.

ຢູ່ໃນລາຍຊື່ແລະລຶບຂໍ້ສຸດທ້າຍຂອງບັນຊີ ໃນທັງສອງກໍລະນີ, ເຫດຜົນດຽວກັນສາມາດຖືກ ນຳ ໃຊ້ເພື່ອຈັດຕັ້ງປະຕິບັດ

ລຶບ node ກຳ ຈັດຂໍ້ມູນອອກຈາກບັນຊີ. node ແມ່ນຢູ່ໃນບັນຊີກາງຫຼືຢູ່ໃນຕອນທ້າຍຂອງບັນຊີ. ຫນ້າທໍາອິດ , ມັນຈໍາເປັນຕ້ອງຮູ້ສະຖານທີ່ຂອງໂຫນດທີ່ຖືກລຶບ. ຈາກນັ້ນໃຫ້ ກຳ ນົດຕົວຊີ້ຂອງ Node ຂອງຈຸດກ່ອນໄປທີ່ຂໍ້ມູນ Succesor. ຖັດຈາກ node ຢູ່ໃນ ຕຳ ແໜ່ງ pLoc ຫຼື node ທີ່ຈະຖືກລຶບໂດຍ

ຂັ້ນຕອນການປະຕິບັດງານແມ່ນສະແດງຢູ່ໃນຮູບ.



## 5.3 ລາຍການເຊື່ອມຕໍ່ແບບເສັ້ນຊື່ ຊະນິດອື່ນໆ(Others Linked Lists)

ຄວາມຮູ້ກ່ຽວກັບລາຍຊື່ link ທີ່ກ່າວມາຂ້າງເທິງແມ່ນບັນຊີລາຍຊື່ link ດຽວ. (ບັນຊີລາຍຊື່ທີ່ເຊື່ອມໂຍງດຽວ) ເພາະວ່າບັນຈຸມີພຽງແຕ່ link ດຽວທີ່ຊີ້ໄປທີ່ node ຕໍ່ໄປ. ເພາະສະນັ້ນ, ປະເພດຂອງບັນຊີລາຍຊື່ການເຊື່ອມຕໍ່ນີ້ຖືກຈົດຈໍາ.Bite ແມ່ນຄວາມບໍ່ສາມາດທີ່ຈະໄປຫາບັນຊີລາຍຊື່ການເຊື່ອມຕໍ່ໃນແບບຫລັງ. ການກະ ທຳ ໃດໆພາຍໃນບັນຊີລາຍຊື່ຕ້ອງເລີ່ມຈາກ node ທຳ ອິດຈົນເຖິງຂໍ້ສຸດທ້າຍ. ແລະບໍ່ສາມາດຍ່າງກັບຫລັງໄດ້ພື້ນຖານກ່ຽວກັບປະເພດອື່ນໆຂອງບັນຊີລາຍຊື່ການເຊື່ອມຕໍ່ ເຊິ່ງມີຄວາມສັບສົນຫຼາຍກວ່າບັນຊີລາຍຊື່ລິງດຽວ, ນີ້ແມ່ນບັນຊີລາຍຊື່ການເຊື່ອມຕໍ່.

ປະເພດຕໍ່ໄປນີ້.

## 5.3.1 ລາຍການເຊື່ອມຕໍ່ແບບວົງມົນ (Circular-linked List)

ປະເພດຂອງບັນຊີລາຍຊື່ການເຊື່ອມຕໍ່ນີ້ ເກີດມາຈາກການປັບປຸງມູນຄ່າບັນຊີການເຊື່ອມຕໍ່ ສໍາລັບການປຸງແຕ່ງທີ່ດີກວ່າ ໂດຍການທົດແທນການເຊື່ອມຕໍ່ທີ່ເປັນ

NULL ຂອງ node ສຸດທ້າຍຂອງບັນຊີລາຍຊື່ການເຊື່ອມຕໍ່ກັບທີ່ຢູ່ຂອງ node ທຳ ອິດ. ການເຊື່ອມໂຍງໃນລັກສະນະນີ້ເອີ້ນວ່າການເຊື່ອມຕໍ່.

ບັນຊີລາຍຊື່ວົງຈອນ (ບັນຊີລາຍຊື່ເສັ້ນລວດເຊື່ອມຕໍ່ເປັນວົງກົມ) ຫຼືບັນຊີລາຍຊື່ວົງຈອນ

ການເຊື່ອມຕໍ່ບັນຊີລາຍຊື່ວົງ ມີປະໂຫຍດຫຼາຍກ່ວາບັນຊີລາຍຊື່ການເຊື່ອມໂຍງທີ່ງ່າຍດາຍ.

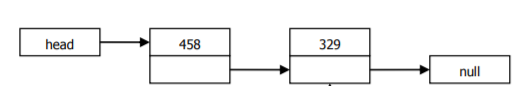
ນັ້ນແມ່ນການເວົ້າ

1. ເພື່ອເຂົ້າເຖິງຂໍ້ມູນ ຂອງຂໍ້ມູນທັງ ໝົດ nodes circle link nodes ສາມາດເຂົ້າເບິ່ງໄດ້ຈາກ node ໃດກໍ່ໄດ້ທີ່

ໃຫ້ຜ່ານລະບົບຕ່ອງໂສ້ (link) ຂອງລາຍການ

2. ເພື່ອລຶບ node, ເພື່ອຄົ້ນຫາ node ທີ່ຢູ່ເບື້ອງຕົ້ນຂອງ node ໃດ ໜຶ່ງ, ການຄົ້ນຫາສາມາດລິເລີ່ມໃນ node ນັ້ນໄດ້.

## 5.3.2 ການເພີ່ມເຂົ້າ node ທາງດ້ານໜ້າ

ເພື່ອ ນຳ ເອົາຂໍ້ມູນໄປທາງ ໜ້າ ຂອງ Linked-List, ພວກເຮົາ ຈຳ ເປັນຕ້ອງຊອກຫາຂໍ້ມູນ.ເພື່ອເຮັດໃຫ້ມັນຮ້າຍແຮງກວ່າເກົ່າ, ພວກເຮົາສາມາດເຮັດສິ່ງນີ້ໄດ້ໂດຍການໃຊ້ node ປັດຈຸບັນເພື່ອເຮັດ ໜ້າ ທີ່ເປັນ deni.ເຂົ້າຫາທຸກໆໂຫນດເພື່ອກວດສອບວ່າ node ທີ່ເຮົາ ກຳ ລັງມາໃກ້ຈະມີຄ່າຕໍ່ໄປnull ຫຼືບໍ່, ຖ້າວ່າແມ່ນ, ຫຼັງຈາກນັ້ນພວກເຮົາພົບເຫັນ node ສຸດທ້າຍ, ຫຼັງຈາກນັ້ນພວກເຮົາເພີ່ມ node ໃຫມ່ໃສ່ບັນຊີລາຍຊື່ທີ່ເຊື່ອມໂຍງໂດຍການເຊື່ອມຕໍ່ node ສຸດທ້າຍກັບ node ໃຫມ່ນີ້ດັ່ງທີ່ເຫັນໃນຮູບ 2.5. 

## 5.3.3 ການເພີ່ມເຂົ້າ Rear node

ວິທີການສະກັດເອົາຂໍ້ມູນຈາກ nodeພື່ອ ນຳ ເອົາການ ນຳ ໃຊ້ນັ້ນອອກມາ ຈະເລີ່ມຕົ້ນໂດຍການຊອກຫາຂໍ້ມູນຈາກສະຖານທີ່ຂໍ້ມູນພາຍໃນບັນຊີ, ຖ້າພົບ.ຂໍ້​ມູນ​ທີ່​ຕ້ອງ​ການ ມັນຈະຍ້າຍຂໍ້ມູນໄປສູ່ພື້ນທີ່ຜົນຜະລິດຂອງໂມດູນການປະຕິບັດ.ແລະຈະສົ່ງຄ່າເຫດຜົນກັບຄືນສູ່ຄວາມຈິງ ແຕ່ຖ້າບໍ່ພົບມັນຈະສົ່ງມູນຄ່າເຫດຜົນທີ່ບໍ່ຖືກຕ້ອງໃຫ້

Pseudocode ເພື່ອດຶງຂໍ້ມູນຈາກ nodes ພາຍໃນບັນຊີ.

## 5.3.4 ລາຍການເຊື່ອມຕໍ່ແບບເສັ້ນຊື່ຄູ່ (Doble Linked List)

ໃນບັນຊີລາຍຊື່ການເຊື່ອມໂຍງແບບນີ້ມີ node ປະກອບມີສອງລາຍຊື່ link ເພື່ອເປັນຕົວແທນຂອງຜູ້ ນຳ ກ່ອນແລະແຫຼ່ງຂໍ້ມູນ source.

ຕໍ່ມາ, node ທີ່ຢູ່ເບື້ອງຕົ້ນຂອງພວກເຮົາຖືກເອີ້ນວ່າການເຊື່ອມຕໍ່ຊ້າຍ, ເຊິ່ງເປັນຕົວແທນໂດຍຕົວຊີ້ LLINK ແລະຕົວເຊື່ອມຕໍ່ທີ່ເປັນຕົວແທນຂອງ node source.

ອັນທີ່ເອີ້ນວ່າການເຊື່ອມຕໍ່ທີ່ຖືກຕ້ອງ, ເຊິ່ງສະແດງໂດຍຕົວຊີ້ R R, ເຊິ່ງແມ່ນບັນຊີລາຍຊື່ການເຊື່ອມໂຍງທີ່ມີຄຸນສົມບັດ.

ພວກເຮົາເອີ້ນວ່າ "ບັນຊີລາຍຊື່ເສັ້ນລວດຄູ່" ຫຼື "ເສັ້ນຄູ່

## 5.3.5 ການເພີ່ມເຂົ້າໂນດໃນ (Double Linked List)

ການສະແດງກິ່ງງ່າຂອງຂໍ້ໃນບັນຊີລາຍຊື່ການເຊື່ອມຕໍ່ຄູ່ແມ່ນຖືວ່າເປັນໄປໄດ້ໃນກໍລະນີຕໍ່ໄປນີ້:

1. ໃນເວລາທີ່ລາຍຊື່ການເຊື່ອມຕໍ່ແມ່ນຫວ່າງ ແທນທີ່ຈະ, ໃຫ້ ກຳ ນົດຕົວຊີ້ L ແລະຕົວຊີ້ R ໄປທີ່ ຕຳ ແໜ່ງ node ໃໝ່.

ແລະ ກຳ ນົດລິ້ງເບື້ອງຊ້າຍແລະລິ້ງຂວາຂອງ node ໃໝ່ ໃຫ້ເປັນ NULL blockquote>

2. ເມື່ອໃສ່ node ໃໝ່ ຢູ່ໃຈກາງຂອງລາຍຊື່ link ບັນຊີລາຍຊື່ການເຊື່ອມໂຍງກ່ອນການແຊກແລະການຕິດຕາມການແຊກ

ຕົວຊີ້ວັດການປ່ຽນແປງແມ່ນມີຄວາມ ສຳ ຄັນຫຼາຍ. ລໍາດັບທີ່ບໍ່ຖືກຕ້ອງອາດຈະເຮັດໃຫ້ເກີດ ນີ້ເຮັດໃຫ້ຂໍ້ທີ່ມີຄຸນຄ່າຕົ້ນສະບັບຖືກສູນຫາຍ.

3. ໃນເວລາທີ່ node ໃຫມ່ຖືກໃສ່ເຂົ້າໄປໃນເບື້ອງຊ້າຍຂອງຂໍ້ມູນທີ່ຢູ່ເບື້ອງຊ້າຍຂອງບັນຊີ, ມັນຈະເຮັດໃຫ້ເກີດ pointer L. ມີການປ່ຽນແປງ

## 5.3.6 ການລືບໂນດໃນ (Double Linked List)

ເພື່ອລຶບປະເພດ node ນີ້ ມັນແຕກຕ່າງຈາກການລຶບ node ໃນບັນຊີລາຍຊື່ link ດຽວໃນນັ້ນມັນບໍ່ ຈຳ ເປັນ. ຕ້ອງມີການຄົ້ນຫາຂໍ້ຂອງຂໍ້. ທີ່ມາກ່ອນຂໍ້ທີ່ຈະຖືກລຶບອອກ, ພຽງແຕ່ ກຳ ນົດທີ່ຕັ້ງຂອງຂໍ້ທີ່ຈະຕ້ອງຖືກລຶບອອກ, ມັນກໍ່ສາມາດຮູ້ໄດ້ ຕຳ ແໜ່ງ ຂອງໂຫດກ່ອນແລະ; ຂໍ້ທີ່ອອກມາຫຼັງຈາກຂໍ້ນັ້ນ ຖ້າບັນຊີລາຍຊື່ການເຊື່ອມຕໍ່ມີ node ດຽວ ການລຶບຂໍ້ອອກຈາກລາຍຊື່ລິ້ງຈະຢູ່ ເພື່ອໃຫ້ໄດ້ຮັບລາຍຊື່ລິ້ງກະເປົ່າແມ່ນຕົວຊີ້ເບື້ອງຊ້າຍ. ແລະທາງໄກຈະຖືກ ກຳ ນົດ ຕັ້ງຄ່າ NULL ຖ້າພິຈາລະນາຂັ້ນຕອນການລຶບແລະໃສ່ຂໍ້ມູນໃສ່ໃນລາຍຊື່ link. ບັນຊີລາຍຊື່ການເຊື່ອມຕໍ່ຄູ່ສາມາດເປັນ ສາມາດເຮັດໃຫ້ມັນງ່າຍຂຶ້ນ ນັ້ນແມ່ນ, ຖ້າບັນຊີລາຍຊື່ການເຊື່ອມໂຍງແມ່ນຫວ່າງເປົ່າ, ມັນສາມາດຖືກກໍານົດໃຫ້ເຊື່ອມໂຍງບັນຊີລາຍຊື່ບໍ່ສາມາດເຮັດໃຫ້ເປົ່າຫວ່າງໄດ້ໂດຍການມອບ ໝາຍ ໃຫ້ເປັນ node ພິເສດໃຫ້ 1 node, ເຊິ່ງແມ່ນຂໍ້ ໜຶ່ງ ທີ່ມີຢູ່ໃນລາຍຊື່ link.ລົງບັນຊີລາຍຊື່ການເຊື່ອມຕໍ່ທີ່ຫວ່າງໄວ້ໂດຍມີພຽງ Node ພິເສດດຽວເທົ່ານັ້ນ. ເຊິ່ງເອີ້ນວ່າ node ຫົວຂອງລາຍຊື່ link ແລະ

ມັນຍັງເປັນໄປໄດ້ທີ່ຈະສ້າງບັນຊີລາຍຊື່ການເຊື່ອມໂຍງຄູ່ເປັນລະບົບບັນຊີລາຍຊື່ການເຊື່ອມຕໍ່ເປັນວົງກົມ ສຳ ລັບການແຊກແລະລຶບຂໍ້ມູນຕໍ່ໄປນີ້. ສິ່ງໃດກໍ່ຕາມທີ່ຫຼຸດລົງຈະຊ່ວຍຫຼຸດຜ່ອນ ລຳ ດັບຂັ້ນຕອນ.