**[201533661 이승수’s Data Structure PHW1,2]**

**[PHW1-1]**

**Implement and Test a Stack Program, Using a Singly Linked List. (15 point)**

**Code)**

#include<stdio.h>

#include<stdlib.h>

#define size 10

struct listNode

{

int data;

struct listNode\* nextPtr;

};

int front = -1, rear = -1;

typedef struct listNode ListNode;

typedef ListNode \*ListNodePtr;

int push(ListNodePtr \*sPtr, int value);

int pop(ListNodePtr \*sPtr);

int stack\_empty(ListNodePtr sPtr);

int stack\_full(ListNodePtr sPtr);

void printList(ListNodePtr currentPtr);

void main()

{

ListNodePtr startPtr = NULL;

int choice;

int value;

printf("Enter your choice: 1(push), 2(pop), 3(end the program)\n");

scanf("%d", &choice);

while (choice)

{

if (stack\_full(startPtr))

{

printf("stack is full!\n");

break;

}

switch (choice)

{

case 1:/\*push and printList\*/

printf("Enter a data value: ");

scanf("\n%d", &value);

push(&startPtr, value);

printList(startPtr);

rear++;

break;

case 2:/\*pop and printList\*/

if (!stack\_empty(startPtr))

{

if (pop(&startPtr))

{

printList(startPtr);

rear--;

}

}

else/\*if stack is empty\*/

{

printf("List is empty.\n");

}

break;

default:

printf("Invalid choice.\n");

break;

}

printf("Enter your choice: 1(push), 2(pop), 3(end the program)\n");

scanf("%d", &choice);

}

}

/\*

input: sPtr node and value(for data to structure node)

output: return 1 if successfully pushed, 0 if not

purpose of function:malloc for newPtr and tracing for latest node.Then add value to newPtr's data.

newPtr will be inserted between previousPtr and currentPtr at last cycle.

\*/

int push(ListNodePtr\* sPtr, int value)

{

ListNodePtr newPtr;// newly created node

ListNodePtr previousPtr;

ListNodePtr currentPtr;

newPtr = malloc(sizeof(ListNode));

if (newPtr != NULL)/\*if newPtr's memory space is successfully allocated\*/

{

newPtr->data = value;

newPtr->nextPtr = NULL;

previousPtr = NULL;

currentPtr = \*sPtr;

while (currentPtr != NULL)/\*looking for last node(last node will be previousPtr)\*/

{

previousPtr = currentPtr;

currentPtr = currentPtr->nextPtr;

}

if (previousPtr == NULL)/\* push new node for empty list\*/

{

newPtr->nextPtr = \*sPtr;

\*sPtr = newPtr;

front++;

}

else/\* if list isn't empty, link newPtr between previousPtr and newPtr\*/

{

previousPtr->nextPtr = newPtr;

newPtr->nextPtr = currentPtr;

}

return 1;

}

else/\* if newPtr's memory isn't allocated\*/

{

printf("%d not inserted. No memory available.\n", value);

return 0;

}

}

/\*

input: sPtr(start node)

output: return 1 if last node is popped successfully, 0 if not.

purpose of function: trace from sPtr to last node and pop the last node of the stack.

\*/

int pop(ListNodePtr\* sPtr)

{

ListNodePtr previousPtr;

ListNodePtr currentPtr;

ListNodePtr tempPtr;

previousPtr = NULL;

currentPtr = \*sPtr;

if (front == rear)/\*when linked list has only one node, pop it\*/

{

tempPtr = \*sPtr;

\*sPtr = (\*sPtr)->nextPtr;

free(tempPtr);

return 1;

}

else/\*pop last node\*/

{

previousPtr = \*sPtr;

currentPtr = (\*sPtr)->nextPtr;

while ((currentPtr->nextPtr) != NULL)

{

previousPtr = currentPtr;

currentPtr = currentPtr->nextPtr;

}

if (currentPtr != NULL)

{

tempPtr = currentPtr;

previousPtr->nextPtr = currentPtr->nextPtr;

free(tempPtr);

return 1;

}

}

return 0;

}

/\*

input: sPtr node(head node of linked list)

output: return 1 if sPtr==1, else(head node isn't empty) return 0

purpose of function: to notice whether list is empty or not

\*/

int stack\_empty(ListNodePtr sPtr)

{

return sPtr == NULL;

}

/\*

input: sPtr(startPtr)

output: return 1 if stack is full(stack's size is 4), 0 if isn't.

purpose of function: check whether stack is full(supposed stack size is 4)

\*/

int stack\_full(ListNodePtr sPtr)

{

if (rear >= (size-1))

return 1;

else

return 0;

}

/\*

input: node of currentPtr

output: none

purpose of function: print data of currentPtr.Then repeatly print next node's data at while loop.

\*/

void printList(ListNodePtr currentPtr)

{

if (currentPtr == NULL)/\*if list is empty\*/

{

printf("List is empty.\n");

}

else

{

printf("The list is:\n");

while (currentPtr != NULL)

{

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

currentPtr = currentPtr->nextPtr;

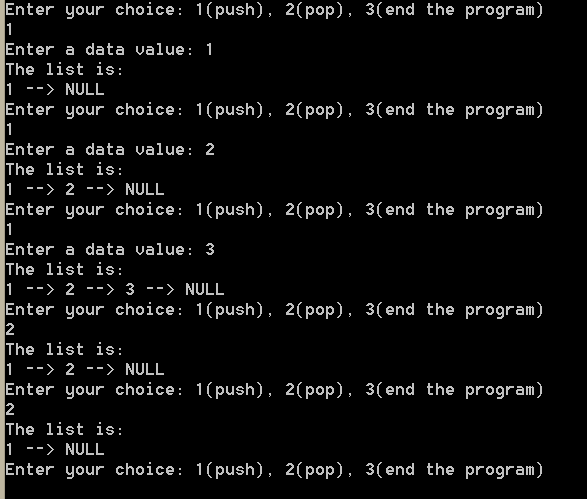
}

printf("NULL\n");

}

}

**Result 1-1)**



**[PHW1-2]**

**Implement and Test a Queue Program, Using a Singly Linked List. (10 points)**

**Code)**

#include<stdio.h>

#include<stdlib.h>

#define size 4

struct listNode

{

int data;

struct listNode\* nextPtr;

};

int front = -1, rear = -1;

typedef struct listNode ListNode;

typedef ListNode \*ListNodePtr;

int push(ListNodePtr \*sPtr, int value);

int pop(ListNodePtr \*sPtr);

int queue\_empty(ListNodePtr sPtr);

int queue\_full(ListNodePtr sPtr);

void printList(ListNodePtr currentPtr);

void main()

{

ListNodePtr startPtr = NULL;

int choice;

int value;

printf("Enter your choice: 1(push), 2(pop), 3(end the program)\n");

scanf("%d", &choice);

while (choice)

{

if (queue\_full(startPtr))

{

printf("stack is full!\n");

break;

}

switch (choice)

{

case 1:/\*push and printList\*/

printf("Enter a data value: ");

scanf("\n%d", &value);

push(&startPtr, value);

printList(startPtr);

break;

case 2:/\*pop and printList\*/

if (!queue\_empty(startPtr))

{

if (pop(&startPtr))

{

printList(startPtr);

}

}

else/\*if stack is empty\*/

{

printf("List is empty.\n");

}

break;

default:

printf("Invalid choice.\n");

break;

}

printf("Enter your choice: 1(push), 2(pop), 3(end the program)\n");

scanf("%d", &choice);

}

}

/\*

input: sPtr node and value(for data to structure node)

output: return 1 if successfully pushed, 0 if not

purpose of function:malloc for newPtr and tracing for latest node.Then add value to newPtr's data.

newPtr will be inserted between previousPtr and currentPtr at last cycle.

\*/

int push(ListNodePtr\* sPtr, int value)

{

ListNodePtr newPtr;// newly created node

ListNodePtr previousPtr;

ListNodePtr currentPtr;

newPtr = malloc(sizeof(ListNode));

if (newPtr != NULL)/\*if newPtr's memory space is successfully allocated\*/

{

newPtr->data = value;

newPtr->nextPtr = NULL;

previousPtr = NULL;

currentPtr = \*sPtr;

while (currentPtr != NULL)/\*looking for last node(last node will be previousPtr)\*/

{

previousPtr = currentPtr;

currentPtr = currentPtr->nextPtr;

}

if (previousPtr == NULL)/\* push new node for empty list\*/

{

newPtr->nextPtr = \*sPtr;

\*sPtr = newPtr;

front++;

rear++;

}

else/\* if list isn't empty, link newPtr between previousPtr and newPtr\*/

{

previousPtr->nextPtr = newPtr;

newPtr->nextPtr = currentPtr;

rear++;

}

return 1;

}

else/\* if newPtr's memory isn't allocated\*/

{

printf("%d not inserted. No memory available.\n", value);

return 0;

}

}

/\*

input: sPtr(start node)

output: return 1 if last node is popped successfully, 0 if not.

purpose of function: trace from sPtr to last node and pop the last node of the stack.

\*/

int pop(ListNodePtr\* sPtr)

{

ListNodePtr previousPtr;

ListNodePtr currentPtr;

ListNodePtr tempPtr;

previousPtr = NULL;

currentPtr = \*sPtr;

if (front == rear)/\*when linked list has only one node, pop it\*/

{

tempPtr = \*sPtr;

\*sPtr = (\*sPtr)->nextPtr;

free(tempPtr);

front--;

rear--;

return 1;

}

else/\*pop front node\*/

{

previousPtr = \*sPtr;

currentPtr = (\*sPtr)->nextPtr;

if (currentPtr != NULL) /\* if data in list, pop front node(currentPtr) \*/

{

tempPtr = currentPtr;

previousPtr->nextPtr = currentPtr->nextPtr;

front++;

free(tempPtr);

return 1;

}

}

return 0;

}

/\*

input: sPtr node(head node of linked list)

output: return 1 if sPtr==1, else(head node isn't empty) return 0

purpose of function: to notice whether list is empty or not

\*/

int queue\_empty(ListNodePtr sPtr)

{

return sPtr == NULL;

}

/\*

input: sPtr(startPtr)

output: return 1 if stack is full(stack's size is 4), 0 if isn't.

purpose of function: check whether stack is full(supposed stack size is 4)

\*/

int queue\_full(ListNodePtr sPtr)

{

if (rear >= (size-1))

return 1;

else

return 0;

}

/\*

input: node of currentPtr

output: none

purpose of function: print data of currentPtr.Then repeatly print next node's data at while loop.

\*/

void printList(ListNodePtr currentPtr)

{

if (queue\_empty(currentPtr))/\*if list is empty\*/

{

printf("List is empty.\n");

}

else

{

printf("front(%d),rear(%d)",front,rear);//

printf("The list is:\n");

currentPtr = currentPtr->nextPtr;

while (currentPtr != NULL)

{

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

currentPtr = currentPtr->nextPtr;

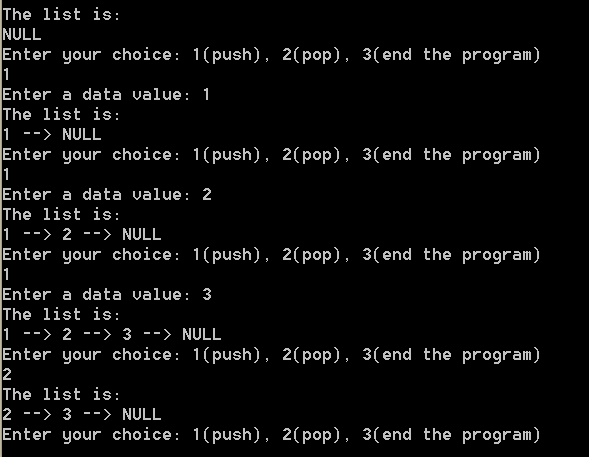
}

printf("NULL\n");

}

}

**Result 1-2)**



**[PHW1-3]**

**Implement a Virtual Integer Stack using 2 Queues. (20 points)**

**Code)**

#include <stdio.h>

/\*

<global variables>

q1 : queue1(main queue)

q2 : queue2(temporary queue)

front1, rear1 : front and rear of queue1

front2, rear2 : front and rear of queue2

\*/

int q1[10];

int q2[10];

int front1 = -1;

int front2 = -1;

int rear1 = -1;

int rear2 = -1;

/\*

<function declaration>

1)pop : fake pop function of a virtual stack. This function is implemented by using 2 queues.

2)push1, pop1: inserting and deleting funciton of queue1

3)push2, pop2: inserting and deleting function of queue2

4)stack\_full : check if queue1 is full

5)stack\_empty : check if queue1 is empty

6)print : print all elements in queue1

\*/

int pop();

void push1(int);

int pop1();

void push2(int);

int pop2();

int stack\_full();

int stack\_empty();

void print();

/\*

Implement a Virtual Stack Using 2 Queues

\*/

int main(int argc, const char \* argv[]) {

int choice; //read a user's choice

int key; //read a user's key when inserting

printf("This is a virtual stack\n");

while (choice != 3) { //loop until the user enters 'e' to end.

printf("Enter your choice: 1(push), 2(pop), 3(end the program)\n");

scanf("%d", &choice);

switch (choice) {

case 1:

//check if full

if(stack\_full() == -1){

printf("Error : Virtual Stack is full!\n");

break;

}

//read a key and call inserting function(fake push of the virtual stack)

printf("Enter a data value: ");

scanf("%d",&key);

push1(key);

print();

break;

case 2:

//check if empty

if (stack\_empty() == -1) {

printf("Error : Virtual Stack is empty!\n");

break;

}

//call deleting function(fake pop of the virtual stack)

pop();

print();

break;

case 3:

printf("End of program\n");

break;

default:

printf("Error : Invalid button. Try again\n");

break;

}

}

return 0;

}

/\*

<description of function>

pop :

implements stack-popping by using 2 queues.

input : none

output : deleted key which is stack-popped.

\*/

int pop(){

int i, delkey;

for (i = 0; i < rear1; i++) //delete all elements of q1, except for the last element ,and insert them into q2

push2(pop1());

delkey = q1[rear1]; //the last element is the deleted key

front1 = -1; //initalization to totally empty queue1

rear1 = -1; //initalization to totally empty queue1

for (i = 0; i <= rear2; i++) //delete all elements of q2 and insert them into q1

push1(pop2());

front2 = -1; //initalization to totally empty queue2

rear2 = -1; //initalization to totally empty queue2

return delkey;

}

/\*

<description of function>

push1 :

increments rear1 by one and add the key in the q1

but if it is the first insertion, front1 is set to 0.

input : integer inserting key in the queue1

output : none

\*/

void push1(int key){

//the first insertion

if(front1 == -1){

front1 = 0;

}

q1[++rear1] = key;

}

/\*

<description of function>

pop1 :

stores the deleted key and increments front1 by one

and returns the deleted key

input : none

output : the deleted key

\*/

int pop1(){

int delkey = q1[front1];

front1++;

return delkey;

}

/\*

<description of function>

push2 :

increments rear2 by one and add the key in the q2

but if it is the first insertion, front2 is set to 0.

input : integer inserting key in the queue2

output : none

\*/

void push2(int key){

//the first insertion

if (front2 == -1) {

front2 = 0;

}

q2[++rear2] = key;

}

/\*

<description of function>

pop2 :

stores the deleted key and increments front2 by one

and returns the deleted key

input : none

output : the deleted key

\*/

int pop2(){

int delkey = q1[front2];

front2++;

return delkey;

}

/\*

<description of function>

stack\_empty :

if the queue is empty, it returns -1

else returns 0

input : none

output : -1 if empty, 0 if not empty

\*/

int stack\_empty(){

if (front1 <= -1) {

return -1;

}

return 0;

}

/\*

stack\_full :

1) if the queue is full, it returns -1

else returns 0

input : none

output : -1 if full, 0 if not full

\*/

int stack\_full(){

if (rear1 >= 9) {

return -1;

}

return 0;

}

/\*

print : prints all elements in the virtual stack

input : none

input : none

\*/

void print(){

int i;

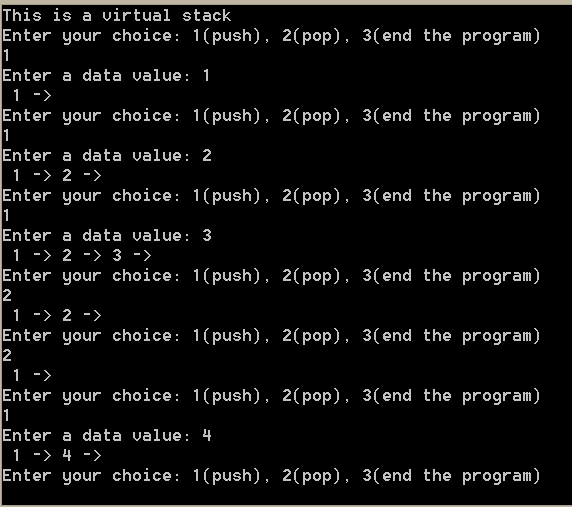
for (i = 0; i <= rear1; i++)

printf(" %d ->",q1[i]);

printf("\n");

}

**Result 1-3)**



**[PHW1-4]**

**Implement a Virtual Integer Queue using 2 Stacks. (10 point)**

**Code)**

#include <stdio.h>

/\*

<global variables>

stack1 : stack1(main stack)

stack2 : stack2(temporary stack)

top1, top2 : top of stack1 and stack2

\*/

int stack1[10];

int stack2[10];

int top1 = -1;

int top2 = -1;

/\*

<function declaration>

1)queue\_insert : fake inserting function of a virtual queue. This function is implemented by using 2 stacks.

2)push1, pop1: inserting and deleting funciton of stack1

3)push2, pop2: inserting and deleting function of stack2

4)queue\_full : check if stack1 is full

5)queue\_empty : check if stack1 is empty

6)print : print all elements in stack1

\*/

int queue\_delete();

void push1(int);

int pop1();

void push2(int);

int pop2();

int queue\_empty();

int queue\_full();

void print();

/\*

Implement a Virtual Queue Using 2 Stacks

\*/

int main(int argc, const char \* argv[]) {

int key; //read a key to insert in the virtual queue

int choice = 0; //read a user's choice

printf("This is a virtual queue\n");

while (choice != 3) {

printf("Enter your choice: 1(push), 2(pop), 3(end the program)\n");

scanf("%d", &choice);

switch (choice) {

case 1:

if(queue\_full() == -1){ //if the virtual queue is full, it prints an error message

printf("Error : Virtual Queue is full!\n");

break;

}

//else insert a key

printf("Enter the inserting key : ");

scanf("%d",&key);

//call push

push1(key);

print();

break;

case 2:

if (queue\_empty() == -1) { //if the virtual queue is empty, it prints an error message

printf("Error : Virtual Queue is empty!\n");

break;

}

//call queue\_insert(fake insert)

queue\_delete();

print();

break;

case 3:

printf("End of program\n");

break;

default:

printf("Error : Invalid button. Try again\n");

break;

}

}

return 0;

}

/\*

<description of function>

queue\_delete :

implements queue deleting by using 2 queues

input : none

output : the deleted key of the virtual queue

\*/

int queue\_delete(){

int i, n, delkey;

n = top1; //keep the current size of the stack1

for (i = 0; i < n; i++) { //pop and push for the size-1 times

push2(pop1());

}

delkey = pop1(); //deleted key is the first element(the only remaining element) of the stack1

top1 = -1; //totally empty the stack1

for (i = 0; i < n; i++) { //pop and push for the size-1 times

push1(pop2());

}

top2 = -1; //totally empty the stack2

return delkey;

}

/\*

<description of function>

push1 :

increments top1 by one and store the key into the stack1

input : integer inserting key

output : none

\*/

void push1(int key){

stack1[++top1] = key;

}

/\*

<description of function>

pop1 :

stores top1 th key in the stack1 and decrement top1 by one

input : none

output : deleted key from stack1

\*/

int pop1(){

int delkey = stack1[top1--];

return delkey;

}

/\*

<description of function>

push2:

increments top1 by one and store the key into the stack2

input : none

output : none

\*/

void push2(int key){

stack2[++top2] = key;

}

/\*

<description of function>

pop1 :

stores top2 th key in the stack1 and decrement top2 by one

input : none

output : deleted key from stack2

\*/

int pop2(){

int delkey = stack2[top2--];

return delkey;

}

/\*

<description>

queue\_empty :

1) if the stack is empty, it returns -1

else returns 0

input : none

output : -1 if empty , 0 if not empty

\*/

int queue\_empty(){

if (top1 <= -1) {

return -1;

}

return 0;

}

/\*

<description>

queue\_full :

1) if the queue is full, it returns -1

else returns 0

input : none

output : -1 if full, 0 if not empty

\*/

int queue\_full(){

if (top1 >= 9) { //stack size is 10 and top starts from 0

return -1;

}

return 0;

}

/\*

<description of function>

print :

prints all elements in the virtual stack

input : none

output : none

\*/

void print(){

int i;

for (i = 0; i <= top1; i++) {

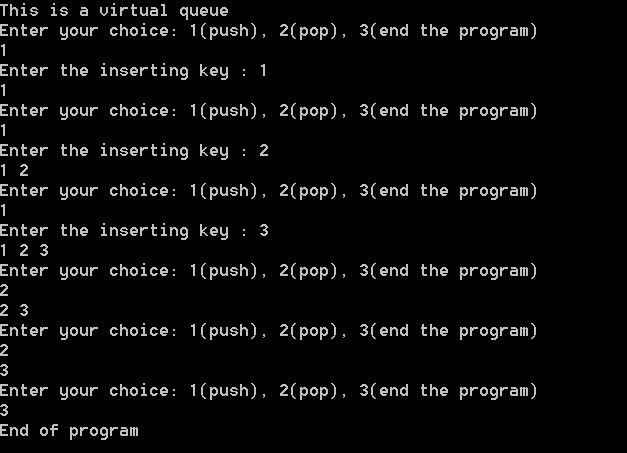
printf("%d ",stack1[i]);

}

printf("\n");

}

**Result 1-4)**



**[PHW2]**

**Implement a ring buffer with an array of 5 elements that uses buffer overflow.**

**Test the program using the sequence of inserts and deletes shown in class**

**(pages 22-23). (30 point)**

**Code)**

#include<stdio.h>

#include<stdlib.h>

#define size 4

struct listNode

{

int data;

struct listNode\* nextPtr;

};

int front = -1, rear = -1;

typedef struct listNode ListNode;

typedef ListNode \*ListNodePtr;

int push(ListNodePtr \*sPtr, int value);

int pop(ListNodePtr \*sPtr);

int queue\_empty(ListNodePtr sPtr);

int queue\_full(ListNodePtr sPtr);

void printList(ListNodePtr currentPtr);

void main()

{

ListNodePtr startPtr = NULL;

int choice;

int value;

printf("Enter your choice: 1(push), 2(pop), 3(end the program)\n");

scanf("%d", &choice);

while (choice)

{

if (queue\_full(startPtr))

{

printf("stack is full!\n");

break;

}

switch (choice)

{

case 1:/\*push and printList\*/

printf("Enter a data value: ");

scanf("\n%d", &value);

push(&startPtr, value);

printList(startPtr);

break;

case 2:/\*pop and printList\*/

if (!queue\_empty(startPtr))

{

if (pop(&startPtr))

{

printList(startPtr);

}

}

else/\*if stack is empty\*/

{

printf("List is empty.\n");

}

break;

default:

printf("Invalid choice.\n");

break;

}

printf("Enter your choice: 1(push), 2(pop), 3(end the program)\n");

scanf("%d", &choice);

}

}

/\*

input: sPtr node and value(for data to structure node)

output: return 1 if successfully pushed, 0 if not

purpose of function:malloc for newPtr and tracing for latest node.Then add value to newPtr's data.

newPtr will be inserted between previousPtr and currentPtr at last cycle.

\*/

int push(ListNodePtr\* sPtr, int value)

{

ListNodePtr newPtr;// newly created node

ListNodePtr previousPtr;

ListNodePtr currentPtr;

newPtr = malloc(sizeof(ListNode));

if (newPtr != NULL)/\*if newPtr's memory space is successfully allocated\*/

{

newPtr->data = value;

newPtr->nextPtr = NULL;

previousPtr = NULL;

currentPtr = \*sPtr;

if(queue\_full(sPtr)){

printf("queue is overflowed!!\n");

return 0;

}

while (currentPtr != NULL)/\*looking for last node(last node will be previousPtr)\*/

{

previousPtr = currentPtr;

currentPtr = currentPtr->nextPtr;

}

if (previousPtr == NULL)/\* push new node for empty list\*/

{

newPtr->nextPtr = \*sPtr;

\*sPtr = newPtr;

front++;

rear++;

}

else/\* if list isn't empty, link newPtr between previousPtr and newPtr\*/

{

previousPtr->nextPtr = newPtr;

newPtr->nextPtr = currentPtr;

rear++;

}

return 1;

}

else/\* if newPtr's memory isn't allocated\*/

{

printf("%d not inserted. No memory available.\n", value);

return 0;

}

}

/\*

input: sPtr(start node)

output: return 1 if last node is popped successfully, 0 if not.

purpose of function: trace from sPtr to last node and pop the last node of the stack.

\*/

int pop(ListNodePtr\* sPtr)

{

ListNodePtr previousPtr;

ListNodePtr currentPtr;

ListNodePtr tempPtr;

previousPtr = NULL;

currentPtr = \*sPtr;

if (front == rear)/\*when linked list has only one node, pop it\*/

{

tempPtr = \*sPtr;

\*sPtr = (\*sPtr)->nextPtr;

free(tempPtr);

front--;

rear--;

return 1;

}

else/\*pop front node\*/

{

previousPtr = \*sPtr;

currentPtr = (\*sPtr)->nextPtr;

if (currentPtr != NULL) /\* if data in list, pop front node(currentPtr) \*/

{

tempPtr = currentPtr;

previousPtr->nextPtr = currentPtr->nextPtr;

front++;

free(tempPtr);

return 1;

}

}

return 0;

}

/\*

input: sPtr node(head node of linked list)

output: return 1 if sPtr==1, else(head node isn't empty) return 0

purpose of function: to notice whether list is empty or not

\*/

int queue\_empty(ListNodePtr sPtr)

{

return sPtr == NULL;

}

/\*

input: sPtr(startPtr)

output: return 1 if stack is full(stack's size is 4), 0 if isn't.

purpose of function: check whether stack is full(supposed stack size is 4)

\*/

int queue\_full(ListNodePtr sPtr)

{

if (rear >= size)

return 1;

else

return 0;

}

/\*

input: node of currentPtr

output: none

purpose of function: print data of currentPtr.Then repeatly print next node's data at while loop.

\*/

void printList(ListNodePtr currentPtr)

{

if (queue\_empty(currentPtr))/\*if list is empty\*/

{

printf("List is empty.\n");

}

else

{

printf("front(%d),rear(%d)",front,rear);//

printf("The list is:\n");

currentPtr = currentPtr->nextPtr;

while (currentPtr != NULL)

{

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

currentPtr = currentPtr->nextPtr;

}

printf("NULL\n");

}

}

**Result)**

