**psh010209@gc.gachon.ac.kr, 010-7661-7035**

**Sanghee Park**

# **개발 과제 #1:**

|  |  |
| --- | --- |
|  | **Implement the Stack using a linked list** |

**Requirements**

* **Step 1: Understand the problem - 간략한 핵심 요구 조건/작업 설명/예시 작성**

**Create an implementation of stack using linked list with a user interface to use.**

* **Memory should be dynamically allocated.**
* **Step 2: Outline a solution – 간단한 logic설명**

**Save basePointer and stackPointer, basePointer saves front pointer, stackPointer saves endpoint.**

**Allocate struct, save current key. And add pointer to point next struct.**

* **Step 3: Form a program structure – 프로그램 전체 구조 요약**

**Push: if stack empty -> basePointer, stackPointer = newNode, else stackPointer->nextPtr = newNode, stackPointer = newNode.**

**Pop: if stack empty -> do nothing, else tmpPointer = stackPointer, stackPointer = prevNode(stackPointer), int n = tmpPointer->key, free(tmpPointer) return n**

**Empty: until empty: pop()**

**Print: until end: print currentNode.**

* **Step 4: write a pseudo coode – 수도코드 작성**

**Push: if stack empty -> basePointer, stackPointer = newNode, else stackPointer->nextPtr = newNode, stackPointer = newNode.**

**Pop: if stack empty -> do nothing, else tmpPointer = stackPointer, stackPointer = prevNode(stackPointer), int n = tmpPointer->key, free(tmpPointer) return n**

**Empty: until empty: pop()**

**Print: until end: print currentNode.**

## **SOURCE CODE with comments**

|  |
| --- |
| *#include* <stdio.h>  *#include* <stdlib.h>  */\* === COMMON === \*/*  *// safe memory allocation:*  *// exits program when it failed to allocation*  void \*safeMemoryAllocation(size\_t size) {  *// memory allocation*  void \*ptr = (void \*)malloc(size);    *// exception handling*  *if* (!ptr) { exit(1); }  *return* ptr;  }  */\* === LINKED LIST === \*/*  *// struct definition*  typedef struct ListNode {  *// integer key*  int key;  *// next struct pointer*  struct ListNode \*nextPtr;  } ListNode;  *// constructor for ListNode struct,*  *// creates listNode with given argument.*  *// returns: memory allocated listNode*  ListNode \*listNodeConstructor(int key, ListNode \*nextPtr) {  ListNode \*newNode = (ListNode \*)safeMemoryAllocation(sizeof(ListNode));  newNode->key = key;  newNode->nextPtr = nextPtr;  *return* newNode;  }  */\* === STACK === \*/*  *// basePointer: pointer of the beginning of the stack*  *// ebp in the x86 assembly.*  ListNode \*basePointer = NULL;  *// stackPointer: pointer of current location at the stack*  *// esp in the x86 assembly language.*  ListNode \*stackPointer = NULL;  */\* === STACK INTERNAL UTILITY FUNCTIONS === \*/*  *// finds previous node of given node pointer*  ListNode \*findPreviousNode(ListNode \*ptr) {  *// if there is no stack, return NULL*  *if* (basePointer == NULL) {  *return* NULL;  }  *// for all, when current Pointer's next is ptr, return*  ListNode \*currPtr = basePointer;  *while* (currPtr->nextPtr != NULL) {  *if* (currPtr->nextPtr == ptr) *return* currPtr;  currPtr = currPtr->nextPtr;  }  *// if none is found, return NULL.*  *return* NULL;  }  */\* === STACK INTERNAL FUNCTIONS === \*/*  *// pushes new key into stack.*  void push(int n) {  *// if the stack was not started at all*  *if* (basePointer == NULL) {  *// initialize stack. allocate memory and get pointers.*  basePointer = listNodeConstructor(n, NULL);  stackPointer = basePointer;  *return*;  }  *// in else, stackPointer should be exist.*  stackPointer->nextPtr = listNodeConstructor(n, NULL);  stackPointer = stackPointer->nextPtr;  }  *// pops key from stack and returns the key.*  int pop(int\* n) {  *// if stack was not started at all*  *if* (basePointer == NULL) {  *// return false*  *return* 0;  }  *// set temp pointer as current stack pointer*  ListNode \*tmpPointer = stackPointer;  \*n = tmpPointer->key;  *// if this is the first node, reset.*  *if* (basePointer == stackPointer) {  basePointer = NULL;  stackPointer = NULL;  } *else* {  *// if this is not the first node, find previous one.*  stackPointer = findPreviousNode(tmpPointer);  stackPointer->nextPtr = NULL;  }  *// free the popped struct*  free(tmpPointer);  *// return true*  *return* 1;  }  *// empty the stack*  void empty() {  *// until stack is undefined,*  *while* (basePointer != NULL) {  *// temporary variable*  int a = 0;  *// pop.*  pop(&a);  }  *return*;  }  *// print the stack*  void print() {  *// temporary variables for printing.*  ListNode \*tmpPointer;  int i;  *// if stack was not defined, give user a message and return early.*  *if* (basePointer == NULL) {  printf("Stack is Empty!!\n");  *return*;  }  *// for basePointer to NULL, print key.*  *for* (tmpPointer = basePointer, i = 1; tmpPointer != NULL; tmpPointer = tmpPointer->nextPtr, i++) {  printf("%d. %d\n", i, tmpPointer->key);  }  *// return*  *return*;  }  */\* === MAIN === \*/*  int main() {  *while* (1) {  *// header*  printf("=== STACK ===\n");  *// print stack*  print();    *// line return*  printf("\n");  *// command prompt*  printf("Please Enter Command (1. push, 2. pop, 3. empty, 4. quit) : ");    *// select the menu*  int menuSelect = 0;  scanf("%d", &menuSelect);  *// in case user inputed multiple variables at once,*  *// flush stdin.*  fflush(stdin);  *// temporary var to save user input.*  int tmp;  *// for each menu select.*  *switch*(menuSelect) {  *case* 1:  *// make user enter the key*  printf("Please enter the key you want to push : ");  scanf("%d", &tmp);  *// just push it*  push(tmp);  *break*;  *case* 2:  *// pop and save to tmp.*  *if* (pop(&tmp)) {  printf("Popped %d\n", tmp);  } *else* {  printf("Popping failed!\n");  }  *break*;  *case* 3:  *// empty the stack*  empty();  printf("Stack Emptied!\n");  *break*;  *case* 4:  *// terminate program*  printf("Bye!\n");  exit(0);  *break*;  *default*:  *// invalid command*  printf("Invalid Command...\n");  *break*;  }  *// in case user inputted multiple variables at once,*  *// flush stdin.*  fflush(stdin);  printf("\n");  }  *return* 0;  } |

## **OUTPUT (Screen Shots)**

|  |  |
| --- | --- |
| **#1** | **Example of Stack** |
|  | |

**//(필요한 만큼 더 생성하여 실행 결과를 잘 파악할 수 있도록 제시)**

**psh010209@gc.gachon.ac.kr, 010-7661-7035**

**Sanghee Park**

# **개발 과제 #2:**

|  |  |
| --- | --- |
|  | **Implement the Queue using a linked list** |

**Requirements**

* **Step 1: Understand the problem - 간략한 핵심 요구 조건/작업 설명/예시 작성**

**Create an implementation of queue using linked list with a user interface to use.**

* **Memory should be dynamically allocated.**
* **Step 2: Outline a solution – 간단한 logic설명**

**Save basePointer and queuePointer, basePointer saves front pointer, queuePointer saves endpoint.**

**Allocate struct, save current key. And add pointer to point next struct.**

* **Step 3: Form a program structure – 프로그램 전체 구조 요약**

**Push: if queue empty -> basePointer, queuePointer = newNode, else queuePointer->nextPtr = newNode, queuePointer = newNode.**

**Pop: if queue empty -> do nothing, else int n = basePointer->key, tmpPointer = basePointer  
basePointer = basePointer->next, free(tmpPointer), return n**

**Empty: until empty: pop()**

**Print: until end: print currentNode.**

* **Step 4: write a pseudo coode – 수도코드 작성**

**Push: if queue empty -> basePointer, queuePointer = newNode, else queuePointer->nextPtr = newNode, queuePointer = newNode.**

**Pop: if queue empty -> do nothing, else int n = basePointer->key, tmpPointer = basePointer  
basePointer = basePointer->next, free(tmpPointer), return n**

**Empty: until empty: pop()**

**Print: until end: print currentNode.**

## **SOURCE CODE with comments**

|  |
| --- |
| *#include* <stdio.h>  *#include* <stdlib.h>  */\* === COMMON === \*/*  *// safe memory allocation:*  *// exits program when it failed to allocation*  void \*safeMemoryAllocation(size\_t size) {  *// memory allocation*  void \*ptr = (void \*)malloc(size);    *// exception handling*  *if* (!ptr) { exit(1); }  *return* ptr;  }  */\* === LINKED LIST === \*/*  *// struct definition*  typedef struct ListNode {  *// integer key*  int key;  *// next struct pointer*  struct ListNode \*nextPtr;  } ListNode;  *// constructor for ListNode struct,*  *// creates listNode with given argument.*  *// returns: memory allocated listNode*  ListNode \*listNodeConstructor(int key, ListNode \*nextPtr) {  ListNode \*newNode = (ListNode \*)safeMemoryAllocation(sizeof(ListNode));  newNode->key = key;  newNode->nextPtr = nextPtr;  *return* newNode;  }  */\* === QUEUE === \*/*  *// basePointer: pointer of the beginning of the queue*  ListNode \*basePointer = NULL;  *// queuePointer: pointer of endpoint of the queue*  ListNode \*queuePointer = NULL;  */\* === QUEUE INTERNAL UTILITY FUNCTIONS === \*/*  *// finds previous node of given node pointer*  ListNode \*findPreviousNode(ListNode \*ptr) {  *// if there is no queue, return NULL*  *if* (basePointer == NULL) {  *return* NULL;  }  *// for all, when current Pointer's next is ptr, return*  ListNode \*currPtr = basePointer;  *while* (currPtr->nextPtr != NULL) {  *if* (currPtr->nextPtr == ptr) *return* currPtr;  currPtr = currPtr->nextPtr;  }  *// if none is found, return NULL.*  *return* NULL;  }  */\* === QUEUE INTERNAL FUNCTIONS === \*/*  *// insertes new key into queue.*  void insert(int n) {  *// if the queue was not started at all*  *if* (basePointer == NULL) {  *// initialize queue. allocate memory and get pointers.*  basePointer = listNodeConstructor(n, NULL);  queuePointer = basePointer;  *return*;  }  *// in else, queuePointer should be exist.*  queuePointer->nextPtr = listNodeConstructor(n, NULL);  queuePointer = queuePointer->nextPtr;  }  *// deletes key from queue and returns the key.*  int delete(int\* n) {  *// if queue was not started at all*  *if* (basePointer == NULL) {  *// return false*  *return* 0;  }  *// set temp pointer as current queue pointer*  ListNode \*tmpPointer = basePointer;  \*n = tmpPointer->key;  *// if this is the first node, reset.*  *if* (basePointer == queuePointer) {  basePointer = NULL;  queuePointer = NULL;  } *else* {  *// if this is not the first node, make second node to base node.*  basePointer = basePointer->nextPtr;  }  *// free the deleted struct*  free(tmpPointer);  *// return true*  *return* 1;  }  *// empty the queue*  void empty() {  *// until delete is undefined,*  *while* (basePointer != NULL) {  *// temporary variable*  int a = 0;  *// delete.*  delete(&a);  }  *return*;  }  *// print the queue*  void print() {  *// temporary variables for printing.*  ListNode \*tmpPointer;  int i;  */// if queue was not defined, give user a message and return early.*  *if* (basePointer == NULL) {  printf("Queue is Empty!!\n");  *return*;  }  *// for basePointer to NULL, print key.*  *for* (tmpPointer = basePointer, i = 1; tmpPointer != NULL; tmpPointer = tmpPointer->nextPtr, i++) {  printf("%d. %d\n", i, tmpPointer->key);  }  *// return*  *return*;  }  */\* === MAIN === \*/*  int main() {  *while* (1) {  *// header*  printf("=== QUEUE ===\n");  *// print queue*  print();    *// line return*  printf("\n");  *// command prompt*  printf("Please Enter Command (1. insert, 2. delete, 3. empty, 4. quit) : ");    *// select the menu*  int menuSelect = 0;  scanf("%d", &menuSelect);  *// in case user inputed multiple variables at once,*  *// flush stdin.*  fflush(stdin);  *// temporary var to save user input.*  int tmp;  *// for each menu select.*  *switch*(menuSelect) {  *case* 1:  *// make user enter the key*  printf("Please enter the key you want to insert : ");  scanf("%d", &tmp);  *// just insert it*  insert(tmp);  *break*;  *case* 2:  *// delete and save to tmp.*  *if* (delete(&tmp)) {  printf("deleted %d\n", tmp);  } *else* {  printf("deleting failed!\n");  }  *break*;  *case* 3:  *// empty the queue*  empty();  printf("Queue Emptied!\n");  *break*;  *case* 4:  *// terminate program*  printf("Bye!\n");  exit(0);  *break*;  *default*:  *// invalid command*  printf("Invalid Command...\n");  *break*;  }  *// in case user inputted multiple variables at once,*  *// flush stdin.*  fflush(stdin);  printf("\n");  }  *return* 0;  } |

## **OUTPUT (Screen Shots)**

|  |  |
| --- | --- |
| **#1** | **Example of Queue** |
|  | |

**//(필요한 만큼 더 생성하여 실행 결과를 잘 파악할 수 있도록 제시)**