Stack

스택

1) 스택(Stack)은 한쪽으로 들어가서 한쪽으로 나오는 자료 구조(Data Structure)입니다.
2) 이러한 특성 때문에 수식 계산 등의 알고리즘에서 다방면으로 활용됩니다.

- PUSH: 스택에 데이터를 넣습니다.
- POP: 스택에서 데이터를 빼냅니다.

스택

1) 스택(Stack)은 한쪽으로 들어가서 한쪽으로 나오는 자료 구조(Data Structure)입니다.

PUSH(7) - PUSH(5) - PUSH(4) - POP() - PUSH(6) - POP()

[배 열]

- 스택의 선언

```
#include <stdio.h>
#define SIZE 10000
#define INF 99999999

int stack[SIZE];
int top = -1;
```

- 스택 삽입 함수

```
void push(int data) {
  if (top == SIZE - 1) {
    printf("스택 오버플로우가 발생했습니다.\n");
    return;
  }
  stack[++top] = data;
}
```

- 스택 추출 함수

```
int pop() {
  if (top == -1) {
    printf("스택 언더플로우가 발생했습니다.\n");
    return -INF;
  }
  return stack[top--];
}
```

- 스택 전체 출력 함수

```
void show() {
  printf("--- 스택의 최상단 ---\n");
  for (int i = top; i >= 0; i--) {
    printf("%d\n", stack[i]);
  }
  printf("--- 스택의 최하단 ---\n");
}
```

```
//#define _CRT_SECURE_NO_WARNINGS
#include <stdio.h>
#include <stdlib.h>

#define SIZE 10000
#define INF 99999999

int stack[SIZE];
int top = -1;

void push(int nData)
{
        if (top == SIZE - 1)
        {
                printf("스택 오버플로우가 발생했습니다.\n");
                return;
        }

        stack[++top] = nData;
```

```
}

int pop()
{
        if (top == -1)
        {
                printf("스택 언더플로우가 발생했습니다.\n");
                return -INF;
        }

        return stack[top--];
}

void show()
{
        printf("---- 스택의 최상단 ----\n");
        for (int i = top; i >= 0; i--)
                printf("[%d] : %d\n", i, stack[i]);
        printf("---- 스택의 최하단 ----\n");
}

int main()
{
        push(1);
        push(2);
        push(3);
        push(4);
        pop();
        push(5);
        show();
```

```
        system("pause");
        return 0;
}
```

[연결 리스트]

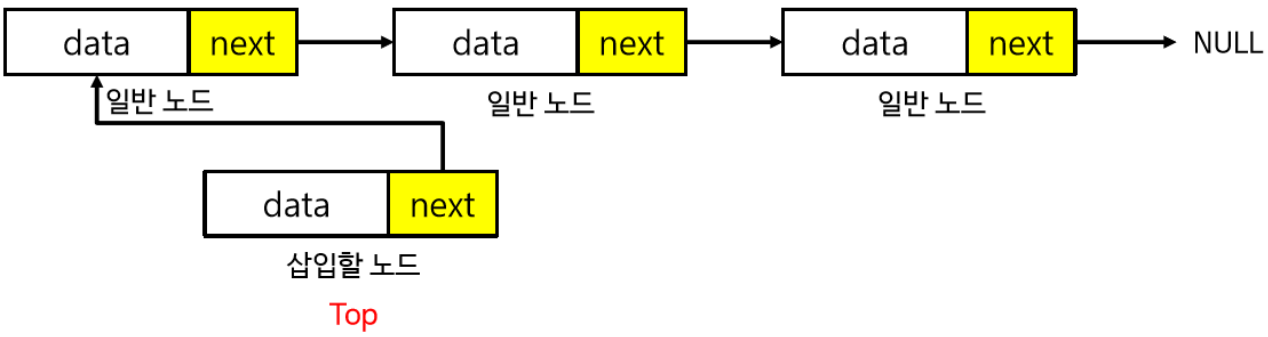
- 스택의 선언

```
#include <stdio.h>
#include <stdlib.h>
#define INF 99999999

typedef struct {
  int data;
  struct Node *next;
} Node;

typedef struct {
  Node *top;
} Stack;
```

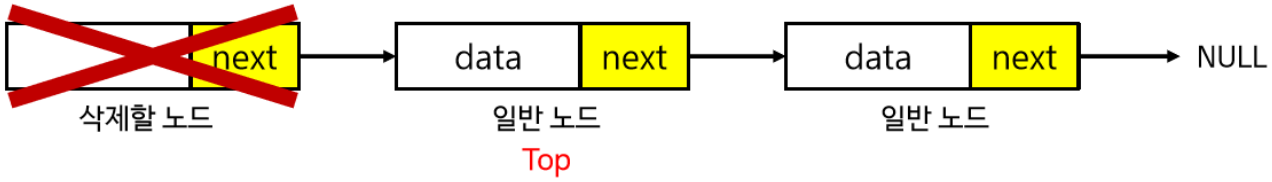
- 스택 삽입 과정 4




```
void Push(Stack* pStack, int nData)
```

```
{
        Node* pNode = (Node*)malloc(sizeof(Node));
        pNode->data = nData;
        pNode->pNext = pStack->pTop;
        pStack->pTop = pNode;
}
```

- 스택 추출 과정 3




```
int Pop(Stack* pStack)
{
        if (NULL == pStack->pTop)
        {
                printf("스택 언더플로우가 발생했습니다.\n");
                return -INF;
        }

        Node* pTop = pStack->pTop;
        pStack->pTop = pTop->pNext;

        int nData = pTop->data;
        free(pTop);
        return nData;
}
```

- 스택 전체 출력 함수

```
void show(Stack *stack) {
  Node *cur = stack->top;
  printf("--- 스택의 최상단 ---\n");
  while (cur != NULL) {
    printf("%d\n", cur->data);
    cur = cur->next;
  }
  printf("--- 스택의 최하단 ---\n");
}
```

```
void Show(Stack* pStack)
{
	Node* pCur = pStack->pTop;

	printf("---- 스택의 최상단 ----\n");
	while (NULL != pCur)
	{
		printf("%d\n", pCur->data);
		pCur = pCur->pNext;
	}
	printf("---- 스택의 최하단 ----\n");
}
```

<계산기 만들기>

```
#define _CRT_SECURE_NO_WARNINGS
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

typedef struct _Node {
        char data[100];
        struct _Node *pNext;
} Node;

typedef struct _Stack{
        Node *pTop;

        void Push(char *pData)
        {
                Node *pNode = (Node*)malloc(sizeof(Node));
                strcpy(pNode->data, pData);
                pNode->pNext = pTop;
                pTop = pNode;
        }

        char* Pop()
        {
                if (NULL == pTop)
                {
                        printf("Stack Underflow.");
                        return NULL;
                }

                Node* pNode = pTop;
```

```
                pTop = pNode->pNext;
                pNode->pNext = NULL;



                char *pData = (char*)malloc(sizeof(char) * 10
0);
                strcpy(pData, pNode->data);

                free(pNode);

                return pData;
        }

        char* GetTop()
        {
                char *pData = (char*)malloc(sizeof(char) * 10
0);
                strcpy(pData, pTop->data);

                return pData;
        }

} Stack;

int GetPriority(char *i);
char* Transition_PostfixnotationFromInfixNotation(Stack *pStac
k, char **ppSrc, int nSize);
void Calculate_PostfixnotationFromInfixNotation(Stack *pStack,
char **ppSrc, int nSize);

int main(void)
```

```
{
        Stack stack;
        stack.pTop = NULL;

        char a[100] = "( ( 3 + 4 ) * 5 ) - 5 * 7 * 5 - 5 * 1
0"; // -190

        int nSize = 1;
        for(int i = 0; i < strlen(a); ++i)
        {
                if (a[i] == ' ') nSize++;
        }

        char *ptr = strtok(a, " ");
        char **ppInput = (char**)malloc(sizeof(char*) * nSiz
e);
        for (int i = 0; i < nSize; ++i)
        {
                ppInput[i] = (char*)malloc(sizeof(char) * 10
0);
        }

        for (int i = 0; i < nSize; ++i)
        {
                strcpy(ppInput[i], ptr);
                ptr = strtok(NULL, " ");
        }

        char b[1000] = "";
        strcpy(b, Transition_PostfixnotationFromInfixNotation
(&stack, ppInput, nSize));
```

```
        printf("후위 표기법 : %s\n", b);

        nSize = 1;
        for (int i = 0; i < strlen(b) - 1; ++i) // 마지막은 항상 공백이 들어가므로 1을 빼기
        {
                if (b[i] == ' ') nSize++;
        }

        char *ptr2 = strtok(b, " ");
        for (int i = 0; i < nSize; ++i)
        {
                strcpy(ppInput[i], ptr2);
                ptr2 = strtok(NULL, " ");
        }

        Calculate_PostfixnotationFromInfixNotation(&stack, ppInput, nSize);

        system("pause");
        return 0;
}

int GetPriority(char *i)
{
        if (!strcmp(i, "(")) return 0;
        if (!strcmp(i, "+") || !strcmp(i, "-")) return 1;
        if (!strcmp(i, "*") || !strcmp(i, "/")) return 2;
        return 3;
}
```

```
// Input Param List
// ppSrc : "1", "+", "2"
// nSize : 3
char* Transition_PostfixnotationFromInfixNotation(Stack *pStac
k, char **ppSrc, int nSize)
{
        char arrRes[100] = "";

        size_t i;
        for (i = 0; i < nSize; ++i)
        {
                if (!strcmp(ppSrc[i], "+") || !strcmp(ppSrc
[i], "-") || !strcmp(ppSrc[i], "*") || !strcmp(ppSrc[i], "/"))
                {
                        while (NULL != pStack->pTop && GetPrio
rity(pStack->GetTop()) >= GetPriority(ppSrc[i]))
                        {
                                strcat(arrRes, pStack->Pop());
                                strcat(arrRes, " ");
                        }

                        pStack->Push(ppSrc[i]);
                }
                else if (!strcmp(ppSrc[i], "("))
                        pStack->Push(ppSrc[i]);
                else if (!strcmp(ppSrc[i], ")"))
                {
                        while (strcmp(pStack->GetTop(), "("))
                        {
```

```
                                strcat(arrRes, pStack->Pop());
                                strcat(arrRes, " ");
                        }

                        pStack->Pop();
                }
                else
                {
                        strcat(arrRes, ppSrc[i]);
                        strcat(arrRes, " ");
                }
        }

        while (NULL != pStack->pTop)
        {
                strcat(arrRes, pStack->Pop());
                strcat(arrRes, " ");
        }

        return arrRes;
}

void Calculate_PostfixnotationFromInfixNotation(Stack *pStack,
char **ppSrc, int nSize)
{
        int x, y, z;
        for (int i = 0; i < nSize; ++i)
        {
                if (!strcmp(ppSrc[i], "+") || !strcmp(ppSrc
[i], "-") || !strcmp(ppSrc[i], "*") || !strcmp(ppSrc[i], "/"))
```

```
                {
                        x = atoi(pStack->Pop());
                        y = atoi(pStack->Pop());

                        if (!strcmp(ppSrc[i], "+")) z = y + x;
                        if (!strcmp(ppSrc[i], "-")) z = y - x;
                        if (!strcmp(ppSrc[i], "*")) z = y * x;
                        if (!strcmp(ppSrc[i], "/")) z = y / x;

                        char buffer[100];
                        sprintf(buffer, "%d", z);
                        pStack->Push(buffer);
                }
                else
                        pStack->Push(ppSrc[i]);
        }

        printf("%s\n", pStack->Pop());
}
```