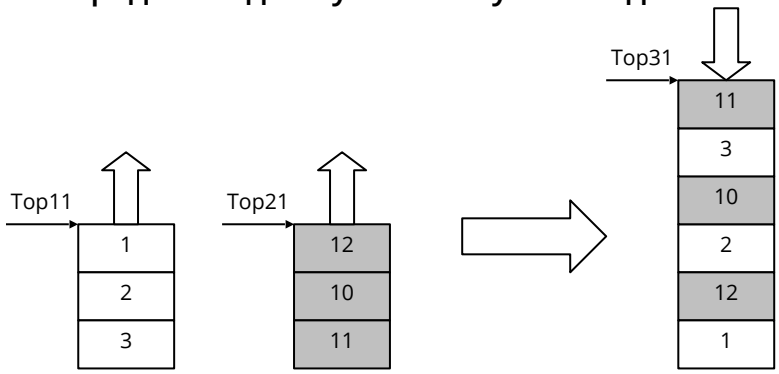
**Балашов, А-06-19, Вариант 2**

**Из элементов двух стеков равной длины собрать один стек, изменив связи между элементами (не выделяя новую память). При создании нового стека элементы брать по-очереди по одному элементу из каждого стека.n**




**Элементы стека**

**typedef int TInfo; //удобство для написания абстрактных функций**

**struct stack //Элемент стэка**

**{**

**TInfo data; //информация элемента**

**stack* next; //указатель на следующий элемент**

**};**

**Функциональные тесты**

| № | Исходные данные | Ожидаемый результат (элемент стека 1 , затем элемент стека 2 по очереди) | Смысл теста |
|---|---|---|---|
| **1** | 1<br>2 | The task result: 1 2 | Один элемент в каждом стеке. Проверка работоспособности |
| **2** | 1 2 3<br>4 5 6 | The task result: 1 4 2 5 3 6 | Проверка работы с несколькими элементами |
| **3** | 1 2 3 4 5 6<br>0 0 0 0 0 0 | The task result: 1 0 2 0 3 0 4 0 5 0 6 0 | Типовое решение |
| **4** | | The task result: Empty stack | Пустой стек (количество элементов=0) |

## Код:

## main.cpp

```
#include "func.h"
#include <cctype>
int main()
{
	stack *afStack, *asStack, *atStack; // Stacks (first, second, third)
	InitNewStack(&afStack);		// Stacks initialization
	InitNewStack(&asStack);
	InitNewStack(&atStack);
	bool end = false;	// if true = stop
	while(!end)
	{
		printf("Choose what to do:\n"
			"\'N\' - enter stack from keyboard\n"
```

```
                "\'T\' - do the task\n"
        "\'V\' - view answer\n"
                "\'E\' - exit\n");
        char choice;                    // Letter of choice
        scanf("%c", &choice);       // Scan input for choice
        fflush(stdin);      // Clearing buffer
        int n;      // Number of elements in stack
        choice = (char)tolower(choice);     // Lowing letters
        switch(choice)
        {
                case 'n':
                        printf("Enter number of elements in stack:\n");
                        scanf("%d", &n);        // Scan input for number of elements
    printf("Enter the first stack:\n");
                        InKeyboard(&afStack, n);        // Run function of creating stack 1 from keyboard
    printf("Enter the second stack:\n");
    InKeyboard(&asStack, n);        // Run function of creating stack 2 from keyboard
                        break;
                case 'v':
                        printf("The task result:\n");
                        StackShow(&atStack);        // Run function of viewing stack
                        break;
                case 't':
                        atStack = Task(afStack, asStack);       // Assigning result of task function to answer stack
                        break;
                case 'e':
                        end = true;
                        break;
                default:
                        printf("Unknown command\n");
                        break;
        }
        printf("----------------------------------------\n");
    }
    End(&afStack, &asStack, &atStack);                  // Clearing memory
    return 0;
}
```

## func.h

```
#ifndef CLAB10_FUNC_H
#define CLAB10_FUNC_H
#include "stack.h"
#include <cstdio>

//// Initialization of new stack by clearing and assigning null
void InitNewStack(stack** aStack)
{
    if (!aStack) destroy(aStack);
    (*aStack) = nullptr;
}

//// Assigning data from keyboard to stack elements
void InKeyboard(stack** aStack, int n)
{
    InitNewStack(aStack);       // Stack initialization
    int data;       // integer of data for elements
    // Entering of data in cycle until the end of elements
  for (int i = 0; i < n; i++) {
    scanf("%d", &data);
    fflush(stdin);
    push(aStack, data);
  }
    n == 0 ? printf("Empty stack was created\n") :
    printf("Stack was created with %d elements\n", n);
}

//// View stack
void StackShow(stack** aStack)
{
    stack* tmp = (*aStack);     // Creating temporary stack
    if (!tmp) printf("Empty stack\n");
```

```
        while (tmp)
        {
                printf("%d\n", tmp -> data);
                tmp = tmp -> next;
        }
        printf("\n");
        free(tmp);
}

//// DO the task
stack* Task(stack *fStack, stack* sStack)
{
        stack *tStack;     // Create the result stack
        InitNewStack(&tStack);     // Initialize the result stack
        // Putting elements from stacks 1 and 2 to the result stack in cycle
    while (fStack && sStack)
    {
      TopToTop(&sStack, &tStack);
      TopToTop(&fStack, &tStack);
    }
        return tStack;
}

//// Clearing memory
void End(stack** fStack, stack** sStack, stack** tStack)
{
        destroy(fStack);
        destroy(sStack);
        destroy(tStack);
}
#endif //CLAB10_FUNCTION_H
```

## stack.h

```
#ifndef CLAB10_STACK_H
#define CLAB10_STACK_H
#include <cstdlib>

typedef int TInfo;     // Making an integer type TInfo
struct stack          // Structure stack
{
        TInfo data;        // Data of stack object
        stack* next;       // Pointer to the next object of stack
};

//// Put an element into stack function
void push(stack** aStack, TInfo data)
{
        if (*aStack == nullptr)     // If stack is empty
        {
                stack* newElement = new stack;     // Creating a new element of stack
                newElement -> data = data;     // Assigning fetched data to the new element's data
                newElement -> next = nullptr;      // Setting the connection of element by setting next element as null
                *aStack = newElement;      // Assigning elements to stack
        } else     // If stack is not empty
        {
                stack* newElement = new stack;     // Creating a new element of stack
                newElement -> data = data;     // Assigning fetched data to the new element's data
                newElement -> next = *aStack;      // Setting the connection of element by setting first stack element as next
                *aStack = newElement;      // Assigning elements to stack
        }
}

//// Put an element from one stack to another function
void TopToTop (stack **StackOne, stack **StackAnother){
   stack *Elem, *StTop=*StackOne, *Dop=*StackAnother;     // Creating temporary stacks
   Elem = StTop;
   StTop = StTop->next;
   Elem->next = Dop;
   Dop = Elem;
   // Assigning temporary stack's data to real stacks
   *StackOne = StTop;
```

```
   *StackAnother = Dop;
}

//// Destroying last stack element function
void destr_last(stack** aStack)
{
	stack* tmp = (*aStack) -> next;
	free(aStack);	// Clearing current object of stack
	*aStack = tmp;
}

//// Destroing the whole stack using destr_last in cycle function
void destroy(stack** aStack)
{
	while (*aStack)
	{
		destr_last(aStack);
	}
}
#endif //CLAB10_STACK_H
```