*Стек* (stack) является как бы противоположностью очереди, поскольку он работает по принципу "последним пришел — первым вышел" (last-in, first-out, LIFO)[[1]](http://www.chitay.org/c/22/2203.htm" \l "11). Чтобы наглядно представить себе стек, вспомните стопку тарелок. Первая тарелка, стоящая на столе, будет использована последней, а последняя тарелка, положенная наверх — первой. Стеки часто применяются в системном программном обеспечении, включая компиляторы и интерпретаторы.

При работе со стеками операции занесения и извлечения элемента являются основными. Данные операции традиционно называются "затолкать в стек" (push)[[2]](http://www.chitay.org/c/22/2203.htm" \l "22) и "вытолкнуть из стека" (pop)[[3]](http://www.chitay.org/c/22/2203.htm" \l "33). Поэтому для реализации стека необходимо написать две функции: push(), которая "заталкивает" значение в стек, и pop(), которая "выталкивает" значение из стека. Также необходимо выделить область памяти, которая будет использоваться в качестве стека. Для этой цели можно отвести массив или динамически выделить фрагмент памяти с помощью функций языка С, предусмотренных для динамического распределения памяти. Как и в случае очереди, функция извлечения получает из списка элемент и удаляет его, если он не хранится где-либо еше. Ниже приведена общая форма функций push() и pop(), работающих с целочисленным массивом. Стеки данных другого типа можно организовывать, изменив базовый тип данных массива.

int stack[MAX];

int tos=0; /\* вершина стека \*/

/\* Затолкать элемент в стек. \*/

void push(int i)

{

if(tos >= MAX) {

printf("Стак полон\n");

return;

}

stack[tos] = i;

tos++;

}

/\* Получить верхний элемент стека. \*/

int pop(void)

{

tos--;

if(tos < 0) {

printf("Стек пуст\n");

return 0;

}

return stack[tos];

}

Переменная tos ("top of stack" — "вершина стека"[[4]](http://www.chitay.org/c/22/2203.htm" \l "44)) содержит индекс вершины стека. При реализации данных функций необходимо учитывать случаи, когда стек заполнен или пуст. В нашем случае признаком пустого стека является равенство tos нулю, а признаком переполнения стека — такое увеличение tos, что его значение указывает куда-нибудь за пределы последней ячейки массива. Пример работы стека показан в табл. 22.2.

|  |  |
| --- | --- |
| **Таблица 22.2. Действие стека** | |
| ***Действие*** | ***Содержимое стека*** |
| push(A) | A |
| push(B) | В А |
| push(C) | C B A |
| рор() извлекает С | В А |
| push(F) | F В А |
| рор() извлекает F | В А |
| рор() извлекает В | А |
| рор() извлекает А | пусто |

Прекрасный пример использования стека — калькулятор с четырьмя действиями. Большинство современных калькуляторов воспринимают стандартную запись выражений, называемую *инфиксной записью*[[5]](http://www.chitay.org/c/22/2203.htm" \l "55), общая форма которой выглядит как *операнд-оператор-операнд*. Например, чтобы сложить 100 и 200, необходимо ввести 100, нажать кнопку "плюс" ("+"), затем ввести 200 и нажать кнопку "равно" ("="). Напротив, во многих ранних калькуляторах (и некоторых из производимых сегодня) применяется *постфиксная запись*[[6]](http://www.chitay.org/c/22/2203.htm" \l "66), в которой сначала вводятся оба операнда, а затем оператор. Например, чтобы сложить 100 и 200 в постфиксной записи, необходимо ввести 100, затем 200, а потом нажать клавишу "плюс". В этом методе операнды при вводе заталкиваются в стек. При вводе оператора операнды извлекаются (выталкиваются) из стека, а результат помещается обратно в стек. Одно из преимуществ постфиксной формы заключается в легкости ввода длинных сложных выражений.

Следующий пример демонстрирует использование стека в программе, реализующей постфиксный калькулятор для целочисленных выражений. Для начала необходимо модифицировать функции push() и pop(), как показано ниже. Следует знать, что стек будет размешаться в динамически распределяемой памяти, а не в массиве фиксированного размера. Хотя применение динамического распределения памяти и не требуется в таком простом примере, мы увидим, как использовать динамическую память для хранения данных стека.

int \*p; /\* указатель на область свободной памяти \*/

int \*tos; /\* указатель на вершину стека \*/

int \*bos; /\* указатель на дно стека \*/

/\* Занесение элемента в стек. \*/

void push(int i)

{

if(p > bos) {

printf("Стек полон\n");

return;

}

\*p = i;

p++;

}

/\* Получение верхнего элемента из стека. \*/

int pop(void)

{

p--;

if(p < tos) {

printf("Стек пуст\n");

return 0;

}

return \*p;

}

Перед использованием этих функций необходимо выделить память из области свободной памяти с помощью функции malloc() и присвоить переменой tos адрес начала этой области, а переменной bos — адрес ее конца.

Текст программы постфиксного калькулятора целиком приведен ниже.

/\* Простой калькулятор с четырмя действиями. \*/

#include <stdio.h>

#include <stdlib.h>

#define MAX 100

int \*p; /\* указатель на область свободной памяти \*/

int \*tos; /\* указатель на вершину стека \*/

int \*bos; /\* указатель на дно стека \*/

void push(int i);

int pop(void);

int main(void)

{

int a, b;

char s[80];

p = (int \*) malloc(MAX\*sizeof(int)); /\* получить память для стека \*/

if(!p) {

printf("Ошибка при выделении памяти\n");

exit(1);

}

tos = p;

bos = p + MAX-1;

printf("Калькулятор с четырьмя действиями\n");

printf("Нажмите 'q' для выхода\n");

do {

printf(": ");

gets(s);

switch(\*s) {

case '+':

a = pop();

b = pop();

printf("%d\n", a+b);

push(a+b);

break;

case '-':

a = pop();

b = pop();

printf("%d\n", b-a);

push(b-a);

break;

case '\*':

a = pop();

b = pop();

printf("%d\n", b\*a);

push(b\*a);

break;

case '/':

a = pop();

b = pop();

if(a==0) {

printf("Деление на 0.\n");

break;

}

printf("%d\n", b/a);

push(b/a);

break;

case '.': /\* показать содержимое вершины стека \*/

a = pop();

push(a);

printf("Текущее значение на вершине стека: %d\n", a);

break;

default:

push(atoi(s));

}

} while(\*s != 'q');

return 0;

}

/\* Занесение элемента в стек. \*/

void push(int i)

{

if(p > bos) {

printf("Стек полон\n");

return;

}

\*p = i;

p++;

}

/\* Получение верхнего элемента из стека. \*/

int pop(void)

{

p--;

if(p < tos) {

printf("Стек пуст\n");

return 0;

}

return \*p;

}