

## Арифметика указателей

Пусть есть одномерный статический массив и указатель на 4-й элемент этого массива:

```
int numbers[6] = {4, 8, 15, 16, 23, 42};  
int* p = &numbers[3];
```

Чему равны следующие выражения:

- |               |                 |                                 |
|---------------|-----------------|---------------------------------|
| 1. numbers[5] | 5. p[0]         | 9. p - numbers                  |
| 2. *p         | 6. p[-1]        | 10. (short*)p - (short*)numbers |
| 3. *(p+1)     | 7. *numbers     | 11. (int)numbers                |
| 4. *(p-2)     | 8. *(numbers+4) | 12. (int)p - (int)numbers       |

Подсказка: имя массива во многих случаях ведёт себя как указатель на первый элемент массива.

## Malloc и free:

Основные функции для динамического выделения памяти:

- **void\*** malloc(**size\_t** n) – выделяет n байт и возвращает указатель **void\*** на начало этой памяти
- **void** free(**void\*** p) – освобождает выделенную память
- **void\*** realloc (**void\*** p, **size\_t** new\_n) – перевыделяет выделенную память

Нужно подключить библиотеку stdlib.h. Если забудете освободить выделенную память, то произойдёт утечка памяти. Пример работы:

```
void* p = malloc(50); // Выделяем 50 байт памяти, адрес первого байта будет храниться в указателе p  
free(p);             // Освободим только-что выделенные 50 байт  
  
int* p1 = malloc(36); // Выделяем 36 байт памяти, с указателем p1 теперь можно обращаться как с  
                // массивом размера 9 (на системах, где размер int равен 4 байта)  
int* p2 = malloc(15 * sizeof(int)); // Выделяем объём памяти достаточный для хранения 15-ти int-ов  
  
// С p1 и p2 можно работать также как и с массивами типа int:  
// Говорят, что p1 и p2 -- динамические массивы (хоть это и просто указатели)  
for (int i = 0; i < 15; ++i)  
    scanf("%d", &p2[i]);  
printf("%d", p2[0] + p2[14]);  
  
p2 = realloc(p2, 25 * sizeof(int)); // Увеличим размер нашего массива с 15 до 25-ти  
free(p1);  
free(p2);
```

### Задачи:

1. **Основы malloc/free**
  - (a) Выделить 123 байта памяти и записать адрес на эту память в указатель типа void\*. Попробуйте разыменовать этот указатель, что при этом произойдёт?
  - (b) Выделить 100 байт памяти и записать адрес на эту память в новый указатель типа int\*.
  - (c) Создать динамический массив для хранения 1 элемента типа int.
  - (d) Создать динамический массив для хранения 10 элементов типа unsigned long long.
  - (e) Создать динамический массив для хранения 100 элементов типа float\*.
  - (f) Создать динамический массив для хранения 10 элементов типа double. Изменить размер этого динамического массива с 10 до 50.
  - (g) Освободить всю память, которую вы выделили.

## Стек

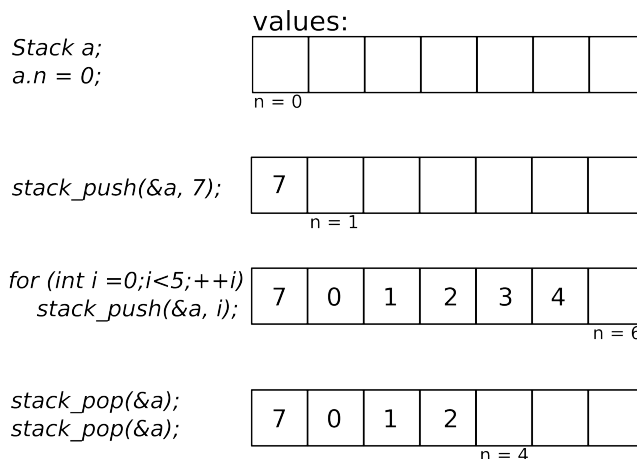
```
struct stack
{
    int n;
    int values[100];
};
typedef struct stack Stack;

void stack_push(Stack* s, int x)
{
    s->values[s->n] = x;
    s->n += 1;
}

int main()
{
    Stack a;
    a.n = 0;
    stack_push(&a, 4);
    stack_push(&a, 10);
    stack_pop(&a);
    printf("%d\n", stack_pop(&a));
}
```

Стек — абстрактный тип данных, представляющий собой список элементов, организованных по принципу «последним пришёл — первым вышел».

Реализация с помощью массива:



### Задачи:

1. Написать функцию **int** stack\_pop(Stack\* s, **int** x). Протестируйте стек: проверьте, что выведет программа, написанная выше.
2. Написать функцию **int** stack\_is\_empty(Stack\* s), которая возвращает 1 если стек пуст и 0 иначе.
3. Написать функцию **int** stack\_get(Stack\* s), которая возвращает элемент, находящийся в вершине стека, но не изменяет стек.
4. Написать функцию **void** stack\_print(Stack\* s), которая распечатывает все элементы стека.
5. Одна из проблем текущей реализации: размер массива 100 захардкожен. Чтобы решить эту проблему введите #define-константу CAPACITY:

```
#define CAPACITY 100
```

6. Что произойдёт, если вызвать stack\_push() при полном стеке? Обработайте эту ситуацию. Программа должна печатать сообщение об ошибке и завершаться с аварийным кодом завершения. Чтобы завершить программу таким образом можно использовать функцию exit() из библиотеки stdlib.h. Пример вызова: exit(1);
7. Аналогично при вызове stack\_pop() и stack\_get() при пустом стеке.
8. Введите функцию stack\_init(), которая будет ответственна за настройку стека сразу после его создания. В данном случае, единственное, что нужно сделать после создания стека это занулить n.
9. Предположим, что вы однажды захотите использовать стек не для целочисленных чисел типа int, а для какого-нибудь другого типа (например char). Введите синоним для типа элементов стека:

```
typedef int Data;
```

Измените тип элемента стека во всех функциях с int на Data (тип поля n менять не нужно). Теперь вы в любой момент сможете изменить тип элементов стека, изменив лишь одну строчку.

10. Сложные скобки. Решить задачу определения правильной скобочной последовательности, используя стек символов. Виды скобок: () {} [] <>. Пример неправильной последовательности: (<>>)

11. Стек с динамическим выделением памяти. Описание такого стека выглядит следующим образом:

```
struct stack
{
    int capacity;
    int n;
    Data* values;
};
typedef struct stack Stack;
```

Введено новое поле `capacity`. В нём будет храниться количество элементов стека, под которые уже выделена память. В отличие от предыдущего варианта стека, это значение будет меняться.

`values` теперь не статический массив, а указатель. Вы должны выделить необходимое место для стека в функции `stack_init()`. Начальное значение `capacity` можно выбрать самостоятельно либо передавать на вход функции `stack_init()`. При заполнении стека должно происходить перевыделение памяти с помощью функции `realloc()`.

### Структуры и `malloc` (дополнительное задание)

1. Объявить структуру, которая будет описывать замеры температуры. Она должна содержать следующие поля:
  - `char scale` – символ, обозначающий шкалу измерений: 'C' – градусы Целься, 'K' – Кельвина, 'F' – Фаренгейта.
  - `double temperature` – вещественное число – температура в этой шкале
  - `int data` – целое число – день замера температуры
2. Чему равен размер этой структуры в байтах? Найти её размер с помощью `sizeof()` и напечатать его на экран.
3. Создать динамический массив для хранения 50-ти таких структур.
4. Увеличить размер этого массива с 50 до 1000.
5. Освободить всю память, которую вы выделили.