Лекция 5: Структуры в языке Си и простейшие структуры данных (стек и очередь)

Д. А. Караваев

Санкт-Петербургский государственный университет телекоммуникаций им. проф. М. А. Бонч-Бруевича

Факультет РТС, Кафедра РОС

Факультатив «Программирование в ЦОС»

Осень 2019

18.11.2019 Санкт-Петербург

Структуры в языке Си

В языке Си существует возможность создать свой собственный *тип* (пользовательский) данных на основе композиции из уже существующих типов. Такая композиция называется **структурой**. Части структуры называются **полями**.

```
/* Объявление структуры, задающей комплексное число: */
struct complex_t
{
    double re; /* Поле: вещественная часть. */
    double im; /* Поле: мнимая часть. */
};
/* Создание переменной типа complex_t */
struct complex_t x;
/* Обращение к полям структуры: */
x.re = 1.4;
x.im = 2.1245;
```

Функции от структур

Для типа `struct complex_t` не определена ни одна операция. Все операции над данным типом необходимо реализовывать через *функции*.

```c
/* Вычисление модуля в квадрате: */
double complex_abs2(struct complex_t x)
{
    return x.re * x.re + x.im * x.im;
}
/* Вычисление суммы двух комплексных чисел: */
struct complex complex_sum(struct complex_t lhs, struct complex_t rhs)
{
    /* Создание и инициализация: */
    struct complex result = {.re = lhs.re + rhs.re,
                             .im = lhs.im + rhs.im};
    return result;
}
```

Функции от структур

Для типа `struct complex_t` неопределена ниодна операция. Все операции над данным типом необходимо реализовывать через *функции*.

```c
/* Вычисление модуля в квадрате: */
double complex_abs2(struct complex_t x)
{
    return x.re * x.re + x.im * x.im;
}
/* Вычисление суммы двух комлексных чисел: */
struct complex complex_sum(struct complex_t lhs, struct complex_t rhs)
{
    /* Создание и инициализация: */
    struct complex result = {.re = lhs.re + rhs.re,
                             .im = lhs.im + rhs.im};
    return result;
}
```

Указатели на структуру

```c
/* Объявление с typedef чтобы избежать использование struct: */
typedef struct
{
    complex_ t samples[1000]; /* Поле: массив с отсчётами. */
    size_t      N; /* Поле: реальная длина сигнала. */
    /* Различные метаданные ... */
} signal_t; /* Имя типа. */
/* Передача в функцию через указатель (во избежание копирования): */
double signal_mlp_const(double value, signal_t* signal)
{
    for (size_t n = 0; n < signal->N; ++n)
    {
        signal->samples[n] *= value; /* Обращение через ->. */
    }
}
```

Указатели на структуру

```c
/* const указатель для выделения входного параметра: */
double signal_energy(const signal_t* signal)
{
    double sum = 0.0;
    for (size_t n = 0; n < signal->N; ++n)
    {
        sum += complex_abs2(signal->values[n]);
    }
    return sum / signal->N;
}
/* Создание: */
signal_t signal;
/* Инициализация ... */
signal_energy(&signal); /* Вызов. */
```

Структуры в Си

Вывод: структура (*данные*) и функции направленные на работу с ней (*методы*) формируют пользовательский тип (**класс**), оперирование с которыми лежит в основе парадигмы *объектно-ориентированного программирования* (ООП).

```c
/* Данные: */
typedef struct
{
    /* ... */
} signal_t;
/* Методы: */
double signal_energy(const signal_t* signal);
uint64_t signal_duration_nanos(const signal_t* signal);
void signal_correlate(const signal_t* lhs, const signal_t* rhs,
                      signal_t* result);
/* ... */
```

Стек и очередь

Структуры данных

Способ организации однотипных данных в памяти, удовлетворяющий некоторому интерфейсу a.k.a *Application Programming Interface* (**API**).

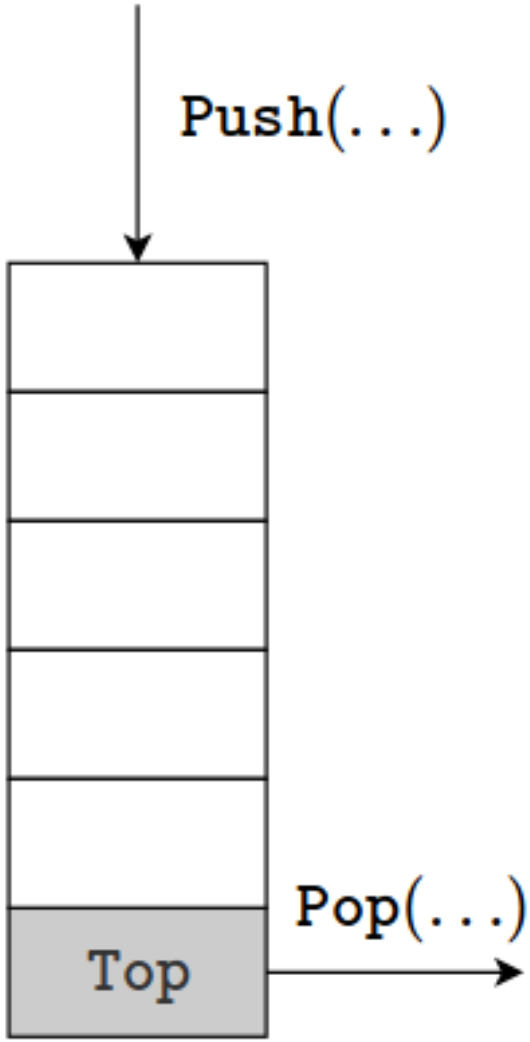
Очередь

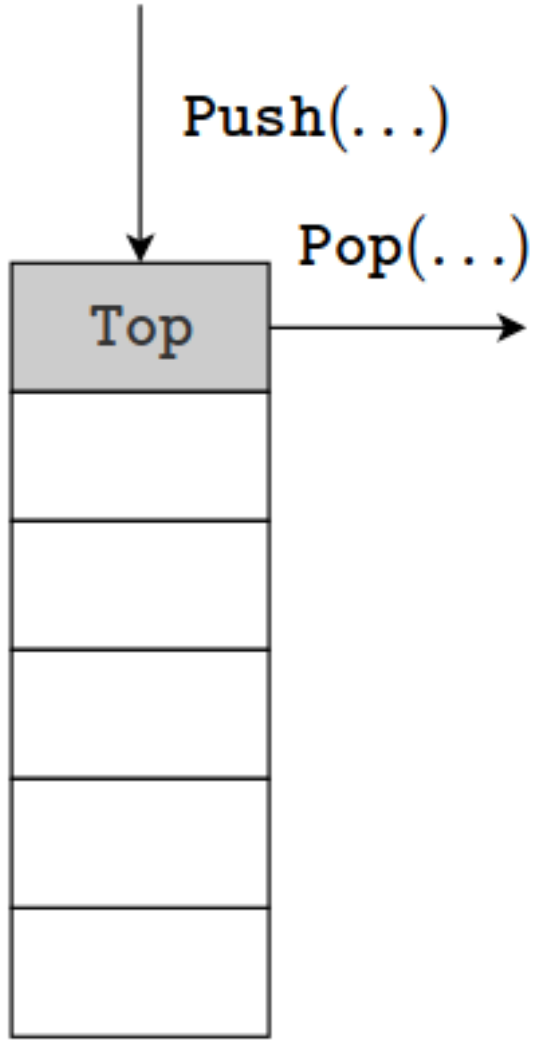
Набор элементов одного типа, в который можно поместить элемент и взять *первый* помещенный. "Честная" очередь. Принцип First-In-First-Out (**FIFO**).

Стек (стопка)

Набор элементов одного типа, в который можно поместить элемент и взять *последний* помещенный. "Нечестная" очередь. Принцип Last-In-First-Out (**LIFO**).

Иллюстрация очереди и стека





Задание

Реализовать:

1. `Push(...)` - поместить элемент (целое число) в очередь (стек);
2. `Pop(...)` - удалить элемент из очереди (стека);
3. `Top(...)` - вернуть "первый" элемент очереди (стека);
4. `Full(...)` - проверка на полноту очереди (стека);
5. `Size(...)` - вернуть число элементов очереди (стека).

Замечание: На самом деле поведение стека и очереди совпадают с точностью до реализации методов `Push(...)` и `Pop(...)`.

Структуры для стека и очереди

```c
/* Структура для стека: */
typedef struct
{
    int    values[100]; /* Элементы. */
    size_t size; /* Число элементов. */
    size_t top; /* Индекс первого элемента. */
} stack;
/* Структура для очереди: */
typedef struct
{
    int    values[100]; /* Элементы. */
    size_t size; /* Число элементов. */
    size_t top; /* Индекс первого элемента. */
} queue;
```

Спасибо за внимание!