Стек и очередь

Николай Пономарев

2 октября 2025 г.

Санкт-Петербургский
государственный университет

Замечания по домашкам

- Следуйте стилю кода!
- Не путайте соглашения в Python и в Си
- Инициализируйте переменные
- Стиль WebKit и настройки Clang Format расходятся :(
  - `int *a` vs. `int* a`
  - Выберите один и используйте его единообразно!
- Дата в названии Pull Request — плохо
  - Лучше использовать номер задачки или её человеческое название
  - Можно даже так: «ДЗ № X.X Название задачки, Фамилия Имя»

Рекомендации по домашкам

- Нужен содержательный main и пример использования
- Старайтесь, чтобы весь ввод-вывод был в main-е
- Старайтесь писать переиспользуемый код (будто ваши функции кому-то могут понадобиться), проектируйте для переиспользования
- Не выкладывайте .exe, .o и т.п.
- Пожалуйста, не называйте массив «Arr», стек «stek» и т.д.
- Закомментированный код не нужен
- a == *true* равносильно a == *true* == *true* и т.д., пишите if (a) или if (!a)
- Именование файлов — тоже в camelCase

Ещё рекомендации

- Тернарный оператор: `printf(x == 0 ? "true" :  "false");`
- Используйте assert (из assert.h) для проверки инвариантов
- Не пишите `x += 1`, пишите `++x`
- Надо ли передавать в функции длину строки?
- Можно ли считать strlen в цикле?
- Не возвращайте 0, если программа завершилась с ошибкой
- Не используйте exit
  - Если функция может не выполниться, используйте коды возврата

Коды возврата

```
int fibonacci(int n, int* result)
{
    if (n <= 0)
        return 1;
    if (n <= 2) {
        *result = 1;
        return 0;
    }
    int previous = 0;
    fibonacci(n - 1, &previous);
    int prePrevious = 0;
    fibonacci(n - 2, &prePrevious);
    *result = previous + prePrevious;
    return 0;
}
...
    int result = 0;
    const int errorCode = fibonacci(x, &result);
    if (errorCode != 0)
        printf("Всё очень плохо");
    else
        printf("%d-ое число Фибоначчи равно %d", x, result);
```

Или так

```
int fibonacci(int n, int* errorCode)
{
    if (n <= 0) {
        *errorCode = 1;
        return 0;
    }
    *errorCode = 0;
    if (n <= 2) {
        return 1;
    }
    return fibonacci(n - 1, errorCode) + fibonacci(n - 2, errorCode);
}
```

...

```
    int errorCode = 0;
    const int result = fibonacci(x, &errorCode);
    if (errorCode != 0)
        printf("Всё очень плохо");
    else
        printf("%d-ое число Фибоначчи равно %d", x, result);
```

if-else и return

```
void f(int x) {
    if (x == 0) {
        ...
    } else {
        ...
    }
}
```

или

```
void f(int x) {
    if (x == 0) {
        ...
        return;
    }
    ...
}
```

Ещё комментарии

Не пишите так:

```
if (x == 10)
    return true;
else
    return false;
```

Пишите так:

```
return x == 10;
```

И ещё комментарии

- Предупреждения компилятора
- Выделение памяти с инициализацией нулями

```
int* array = (int*)calloc(size, sizeof(int));
if (array == NULL) {
    printf("Всё очень плохо :(")
    return 1;
}
...
free(array);
```

Структуры I/II

- Способ группировки родственных по смыслу значений
- Структура — это тип
  - В памяти представляется как поля, лежащие друг за другом, возможно, с «дырками» (padding)
  - Объявляется вне функции

Объявление структуры:

```
struct Point {
    int x;
    int y;
};
```

Использование:

```
struct Point p;
p.x = 10;
```

Структуры II/II

- Или, чтобы `struct` каждый раз не писать:

```
typedef struct {
    int x;
    int y;
} Point;
```

  - `typedef` — объявление синонима типа
- Использование:

```
Point p = {10, 20};
printf("(%d, %d)", p.x, p.y);
```

- Продвинутая инициализация:

```
Point p = {.x = 10, .y = 20};
```

Указатели и структуры

Структуры и указатели настолько часто используются вместе, что есть оператор → (разыменовать указатель на структуру и обратиться к её полю)

```
int main(void)
{
    Point* p = malloc(sizeof(Point));
    if (p == NULL) {
        return 1;
    }
    p->x = 10;
    p->y = 20;
    printf("(%d, %d)", p->x, p->y);
    free(p);

    return 0;
}
```

То же самое, что (*p).x и (*p).y

Операция взятия адреса

```c
int main(void)
{
    Point p1 = { 10, 20 };
    Point* p = &p1;
    int* test = &(p1.x);
    printf("(%d, %d)\n", p->x, p->y);
    *test = 30;
    printf("(%d, %d)\n", p->x, p->y);
    printf("(%d, %d)\n", p1.x, p1.y);
}
```

Структуры и строки

```c
typedef struct {
    char* name;
    char phone[30];
} PhoneBookEntry;

int main(void)
{
    PhoneBookEntry entry;
    const char* name = "Ivan Ivanov";
    entry.name = malloc(sizeof(char) * (strlen(name) + 1));
    if (entry.name == NULL) {
        return 1;
    }
    strcpy(entry.name, name);
    strcpy(entry.phone, "+7 (911) 123-45-67");
    printf("%s - %s", entry.name, entry.phone);
    free(entry.name);
    return 0;
}
```

Структуры могут указывать сами на себя

```c
typedef struct ListElement {
    int value;
    struct ListElement* next;
} ListElement;

int main(void)
{
    ListElement* element1 = malloc(sizeof(ListElement));
    element1->value = 1;
    ListElement* element2 = malloc(sizeof(ListElement));
    element2->value = 2; element2->next = NULL;
    element1->next = element2;

    printf("%i - %i", element1->value, element1->next->value);

    free(element1); free(element2);

    return 0;
}
```

Стек на указателях

Структура данных, в которой элемент можно добавлять только в начало и забирать только из начала (LIFO, Last In — First Out)

- Может хранить «сколько угодно» данных
  - Можно сделать стек ещё на массивах, но тогда он будет ограничен по размеру

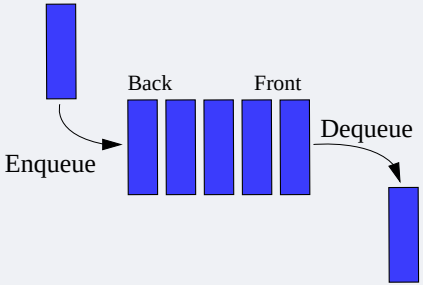
Vectorization: Alhadis, CC BY-SA 3.0, via Wikimedia Commons

- Используется везде
  - Для организации рекурсии
  - Для синтаксического анализа программ
  - Для проверки корректности скобок
  - Для арифметических вычислений
  - Для исполнения программ (.NET — пример стековой машины)
  - ...

Очередь

Структура данных, в которой элемент можно добавлять только в конец и забирать только из начала (FIFO, First In — First Out)

- Используется для
  - Обмена сообщениями между параллельными потоками
  - Обработки событий от пользователя или от операционной системы
  - ...



Практика

Вместе реализуем стек на указателях, оперирующий целыми числами, с тремя операциями:

- `push` Положить элемент на стек
- `pop` Взять элемент со стека
- `peek` Посмотреть на элемент на вершине стека

и служебными функциями:

- `new` Создать пустой стек
- `delete` Удалить весь стек (освободить память)

Домашнее задание

Все задачи решаются с помощью стека — его надо реализовать единожды в отдельном модуле, и использовать во всех этих задачах. Чтобы каждую задачу можно было сдавать в отдельной ветке, надо сначала сделать ветку для модуля «стек», реализовать там стек, а затем уже от неё отвести три ветки для конкретных задач. При этом правки к самому стеку надо делать в ветке для стека, а потом вливать изменения из неё в ветки с задачами. При этом пуллреквест из ветки со стеком открывать не надо. Комментарии ко всем функциям из заголовочного файла обязательны.

1. Написать программу проверки баланса скобок в строке, скобки могут быть трёх видов: (), [], {}.
2. Написать программу, преобразующую выражение из инфиксной формы в постфиксную. В выражении могут быть знаки +, -, *, /, скобки и цифры. Пример: (1 + 1) * 2 должно преобразовываться в 1 1 + 2 *. Алгоритм перевода предлагается найти самостоятельно (алгоритм «сортировочной станции» Э. Дейкстры).