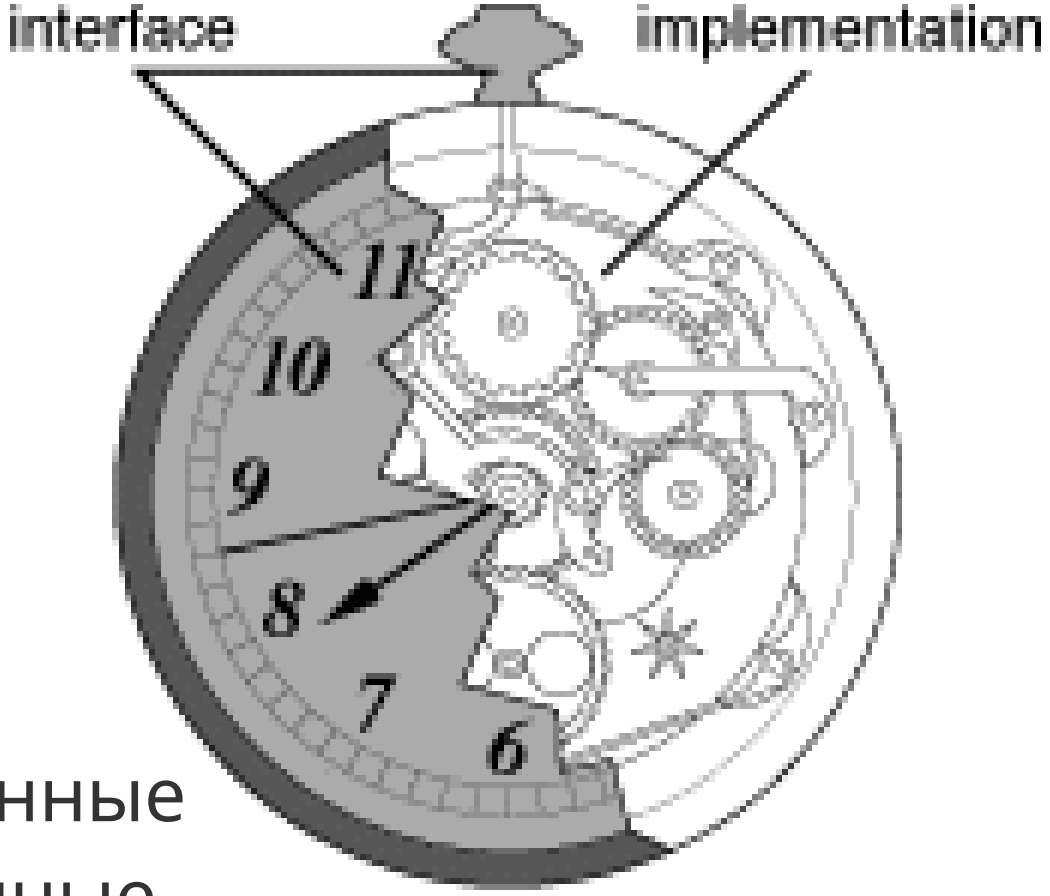
Структуры данных
Стек на основе динамического массива

Основы языка C, лекция 12

Интерфейс и реализация

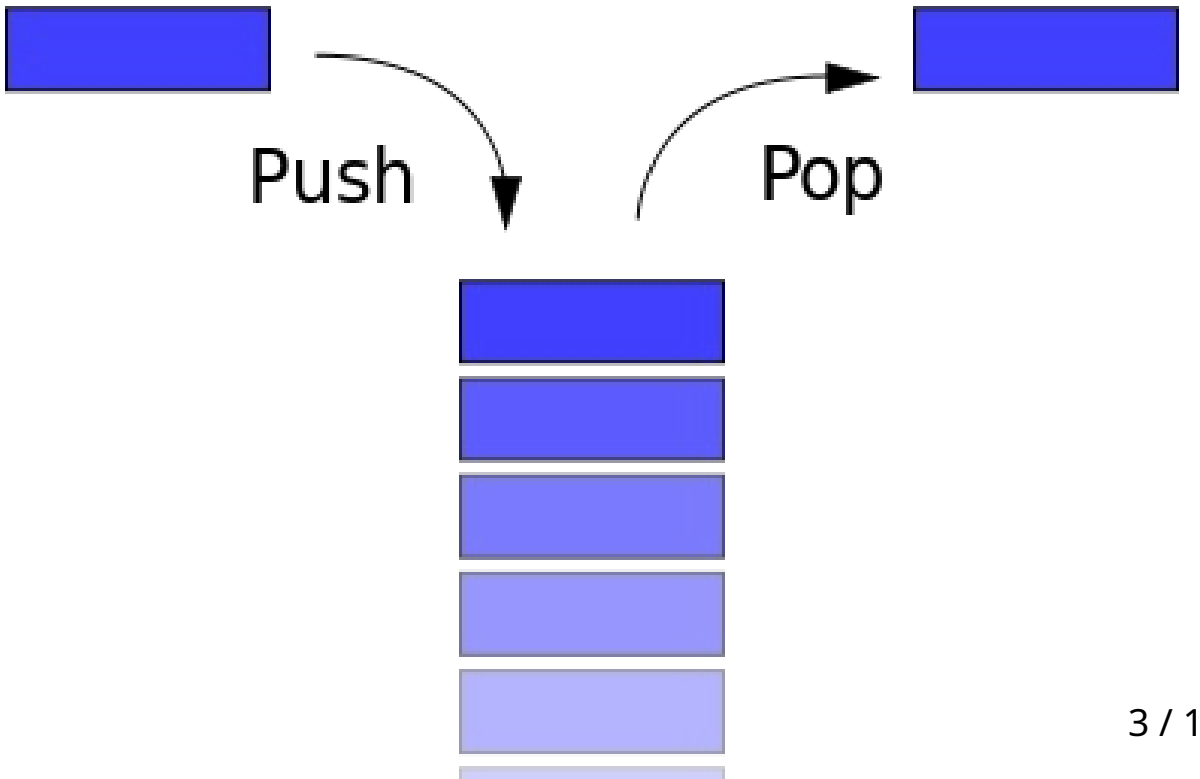
- Интерфейс — набор функций
- Реализация спрятана



- Интерфейс
  - get_time
  - set_time
  - завести часы
- Реализация
  механические, электронные
  солнечные, свечи, водяные

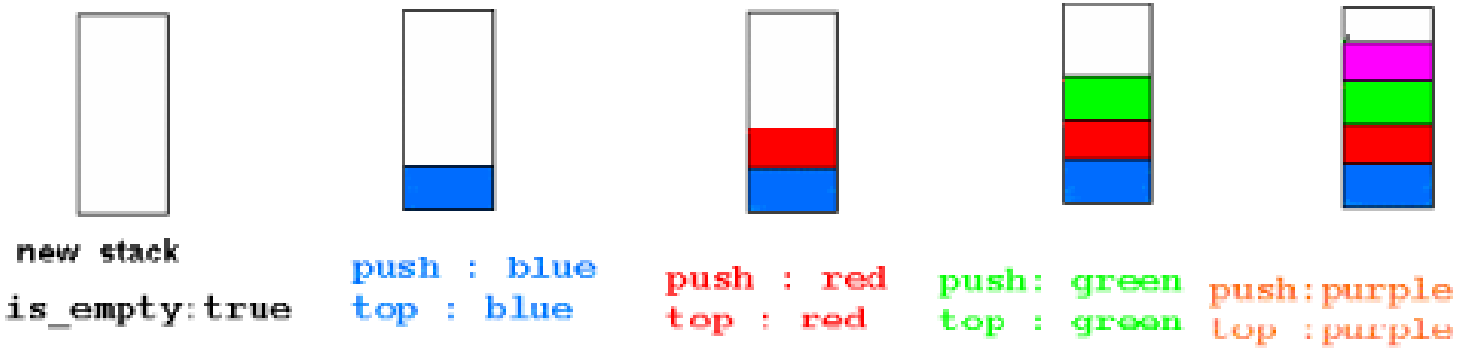
Стек (stack)

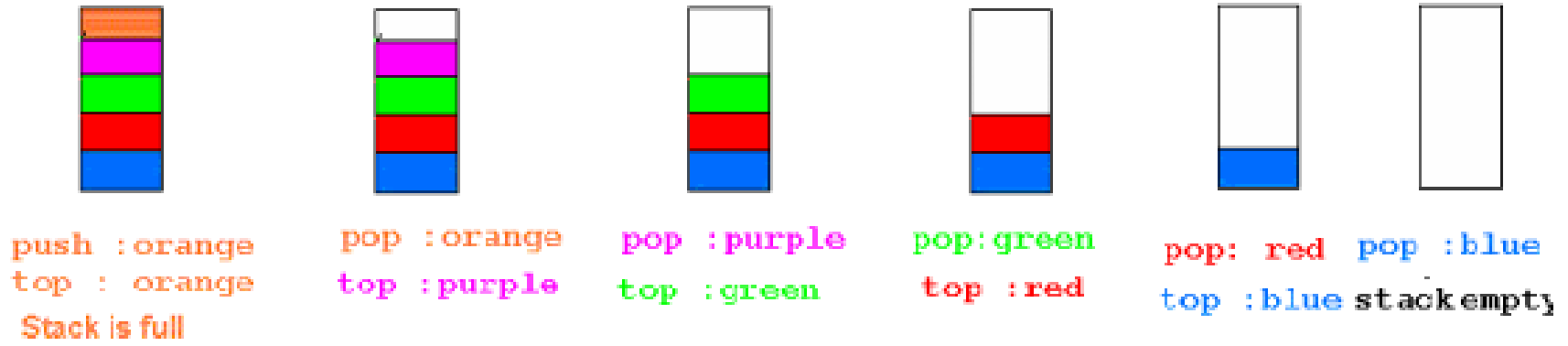
- LIFO - last in, first out
  (последним пришел, первым вышел)
- **push** (x) — положить элемент x в стек
- x = **pop** ( ) - вынуть элемент из стека и вернуть его



push и pop

- **push** (x) — положить элемент x в стек
- x = **pop** ( ) - вынуть элемент из стека и вернуть его





Задачи, где нужен стек

Последовательность	Корректна?
(( )) (( ) ( ))	Да
( ))	Нет
) (	Нет

- Напечатать корректна ли скобочная последовательность
- Стек не нужен, счетчик открывающих и закрывающих скобок

Скобочки — 2 типа

Последовательность	Корректна?
(< >) < < > ( ) >	Да
( ))	Нет
(<) >	Нет

- Добавим еще один тип скобок
- Последний пример на счетчиках не работает

Скобки со стеком

- **<()>** или **(<>())** или **())** или **((** или **(<)>**
- Пока есть скобки
  если скобка открывающая
  положить ее в стек **push**
  если скобка закрывающая
  достать скобку из стека **pop**
  если достать не можем — плохо
  если эти скобки не пара — плохо

  В конце проверить, что стек пустой
  Последовательность корректная

Реализация

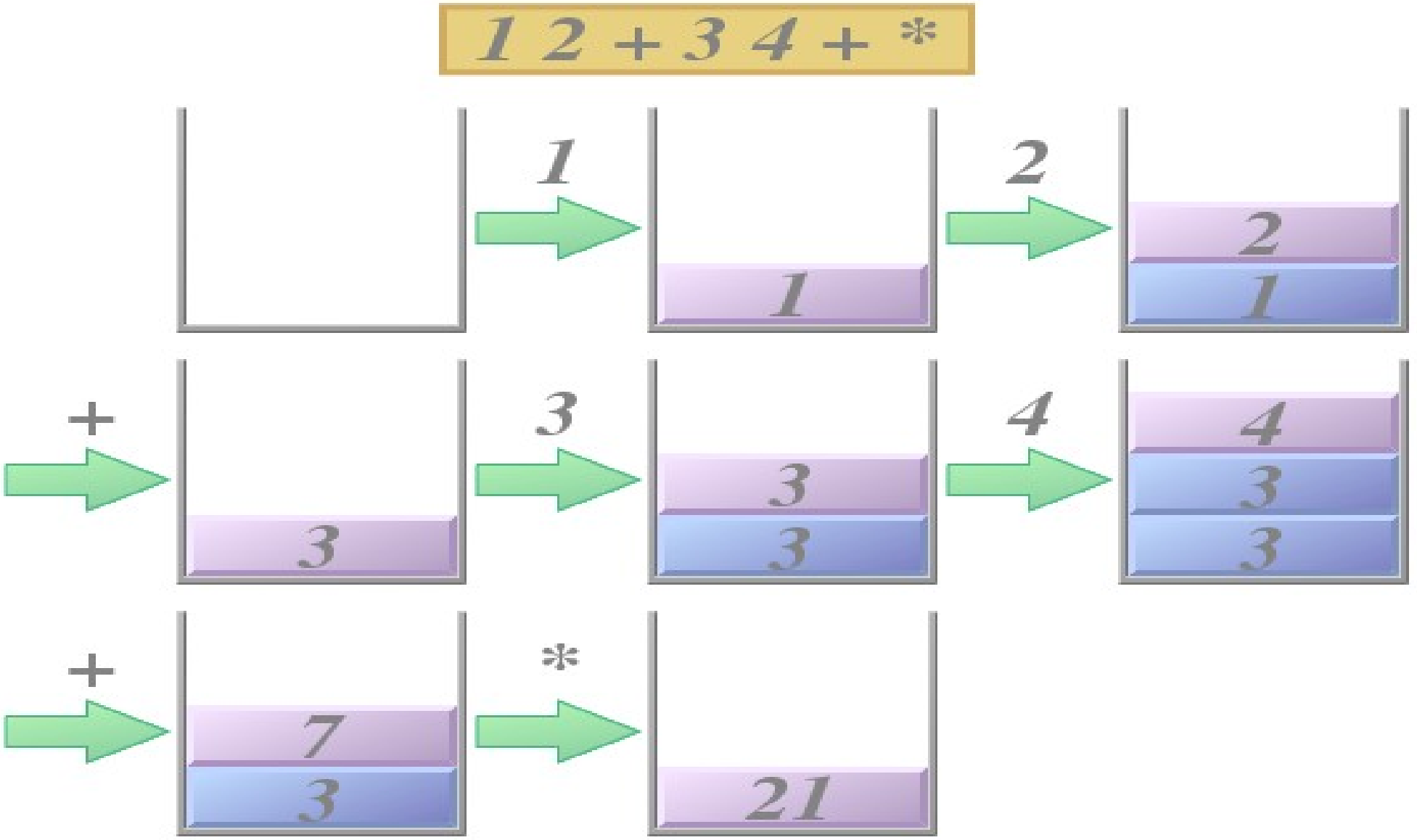
- Написать так, чтобы еще одна пара скобок добавлялась легко
- char * begin = "(<{";
  char * end = ")>}";
- char * bracket [ ] = {"()", "<>", "{}"};
- Сложность реализации требований
  - Сделать 1 вид скобок ( )
  - Добавить еще один < >
  - Добавить еще один { }

Постфиксная запись

Инфиксная запись	Постфиксная запись
2 + 3	2 3 +
2 + 3 * 4	2 3 4 * +
(2 + 3) * 4	2 3 + 4 *
(2 + 3) * (4 + 5)	2 3 + 4 5 + *

- Пока есть токены
  - если это число
    - положить в стек
  - если это оператор
    - извлечь из стека 2 операнда
    - выполнить операцию
    - результат положить в стек

(1 + 2) * (3 + 4)

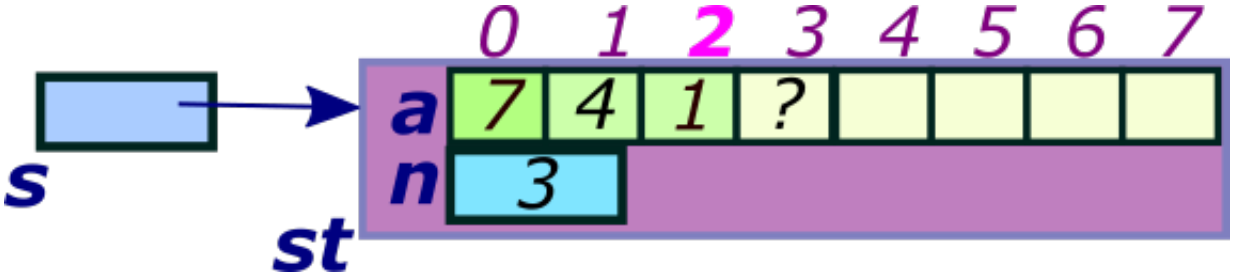


git — система контроля версий

- хранилище (репозиторий) версии
  несохраненные изменения
  сундучок (потом отнесем в хранилище)
- **git init** *директория*
  создать репозиторий *директория*
- **git add** файл
  добавить файл (он станет отслеживаться (track))
  готовим изменения из файла к commit
- **git commit** -m "тут пишем, что мы изменили"
  подготовленные изменения занести в хранилище
- **git status** **git log** **git diff**

Реализация стека на основе массива

- #define **N** 8
  typedef int **Data**;
  typedef struct {
  Data a[N]; // храним данные в стеке
  int **n**; // сколько данных в стеке
  } **Stack**;

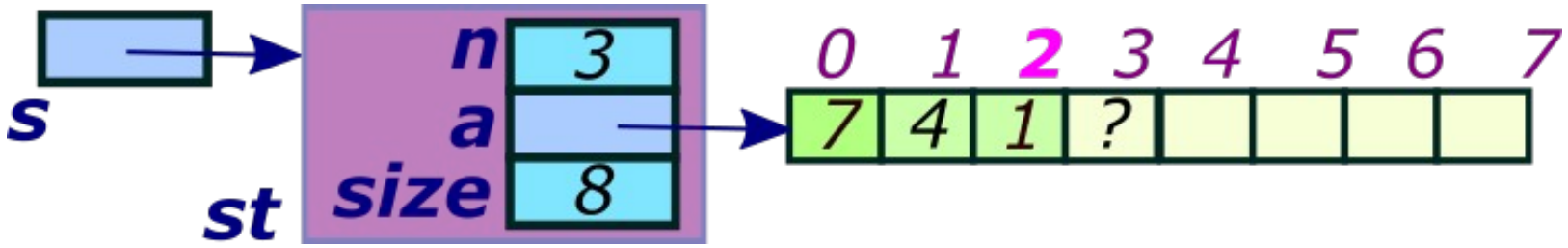


- **Stack** st; push (&st, 5)
- void **push** (**Stack** * s, **Data** x);
  Data **pop** (**Stack** * s); // top
  void **init** (**Stack** * s); // начало работы
  void **print** (**Stack** * s);
  int **is_empty** (**Stack** * s); // int is_full (Stack * s);

Плюсы и минусы

- Плюсы
  - просто (очень просто!)
- Минусы
  - стек маленький (сделаем N больше, еще больше)
  - неэффективно используется память
    (сделаем N меньше?)
- Хотим:
  стек, который растет и уменьшается сам
  ограничен размером памяти

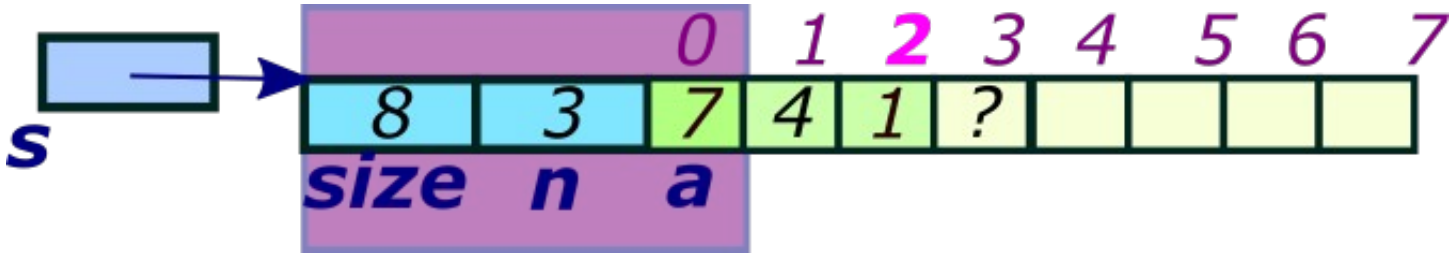
на основе динамического массива



- typedef int **Data**;
  typedef struct {
  unsigned int **n**; // сколько данных в стеке
  Data * **a**; // храним данные в стеке
  size_t **size**; // емкость стека
  } **Stack**;

- интерфейс тот же:
  void **push** (**Stack** * s, **Data** x);
  Data **pop** (**Stack** * s);
  void **init** (**Stack** * s); и так далее

1 malloc



- typedef int **Data**;
  typedef struct {
  size_t **size**; // емкость стека
  unsigned int **n**; // сколько данных в стеке
  Data a[1]; // должно быть последним полем
  } **Stack**;

- интерфейс изменится:
  Stack * **push** (**Stack** * s, **Data** x);
  Stack * **pop** (**Stack** * s);
  Stack * **init** (**Stack** * s); далее останется прежним