

Разработка объектно-ориентированного языка со сборкой мусора

Студент: Масягин М.М.
Научный руководитель: Синявин А.В.



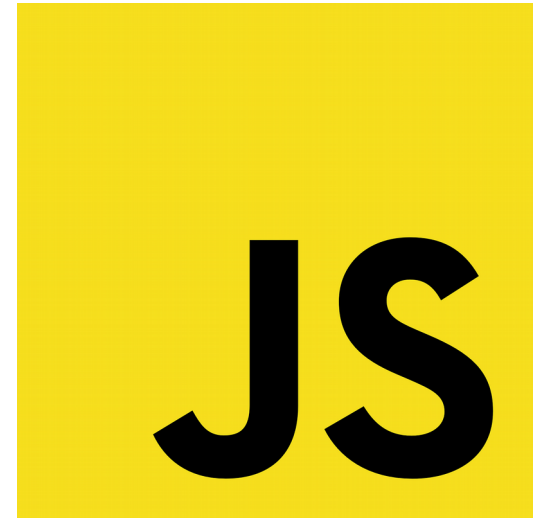
Постановка задачи

- Разработать спецификацию собственного объектно-ориентированного языка с поддержкой массивов и объектов;
- Написать для разработанного языка интерпретатор;
- Создать и протестировать подсистему сборки мусора;



Дизайн языка

- С-подобный синтаксис;
- Объекты как в JavaScript;
- Автоматическая сборка мусора;
- Поддержание ссылок всегда актуальными;
- Сильная динамическая типизация;
- Максимальная простота и однозначность языка:
 - отсутствие побочных результатов операций;
 - объявление не более одной переменной за раз;
 - отсутствие goto-меток, операторов ++, --, null;



```
function test() {  
  let a = {  
    b : [1, 2, 3, 4, 5, 6, 7, 8, 9, -1]  
  }; // Инициализация объекта и вложенного в него массива.  
  a.b[9] = 10;  
  a.c = a; // Циклическая зависимость.  
  let i = -1;  
  let sum = 0;  
  while (i < 1000) { // While.  
    i = i + 1;  
    if (i == 10) { // if-then-else.  
      break; // break.  
    } else if (a.b[i] % 2 != 0) {  
      continue; // continue.  
    } else {  
      sum = sum + a.c.b[i] * a.c.b[i];  
    }  
  }  
  return sum;  
}
```

Я хотел создать удобный и современный
язык, а получился JavaScript...

Front-end

- Собственноручно написанные лексер и парсер;
- Лексер является объектно-ориентированным и способен восстанавливаться при невалидных лексемах;
- Парсер использует предсказывающий метод рекурсивного спуска (без возвратов), при ошибках выводит рекомендации по их исправлению;



Back-end

- Генератор байт-кода;
- Виртуальная машина;
- Сборщик мусора;

`return a.b[i][1];`



CONSTANT POOL:

`INT_CNST` IDX = 1 VALUE = 1

`FIELDREF_CNST` IDX = 1 VALUE = b

OP_CODES:

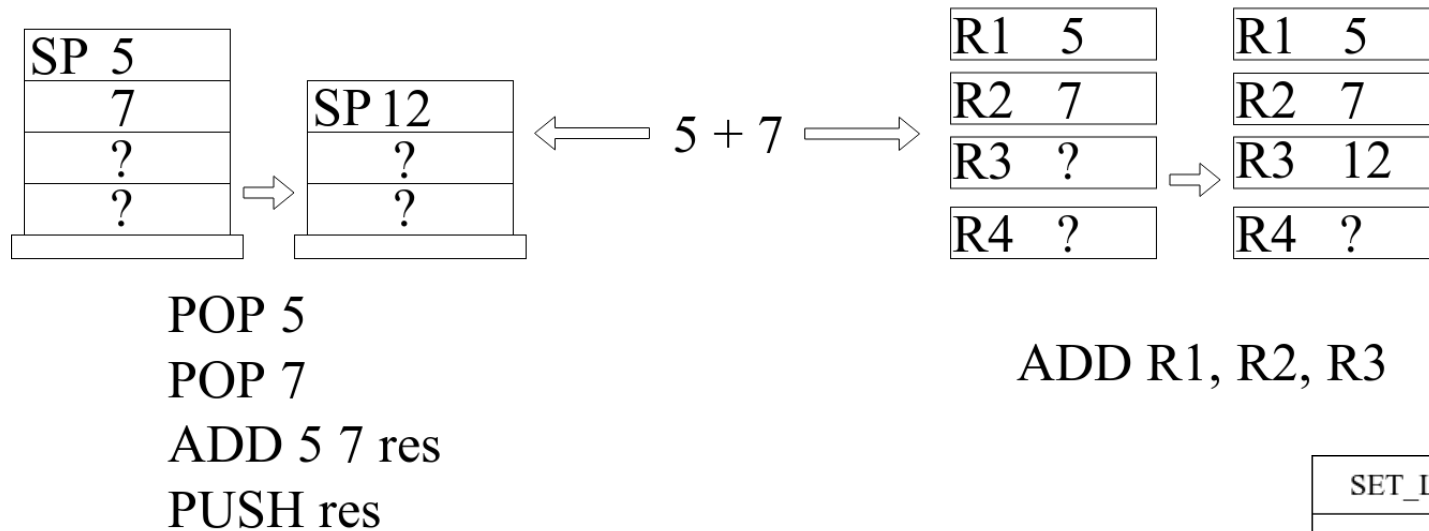
`GET_LOCAL` 1

`CONSTANT` 1

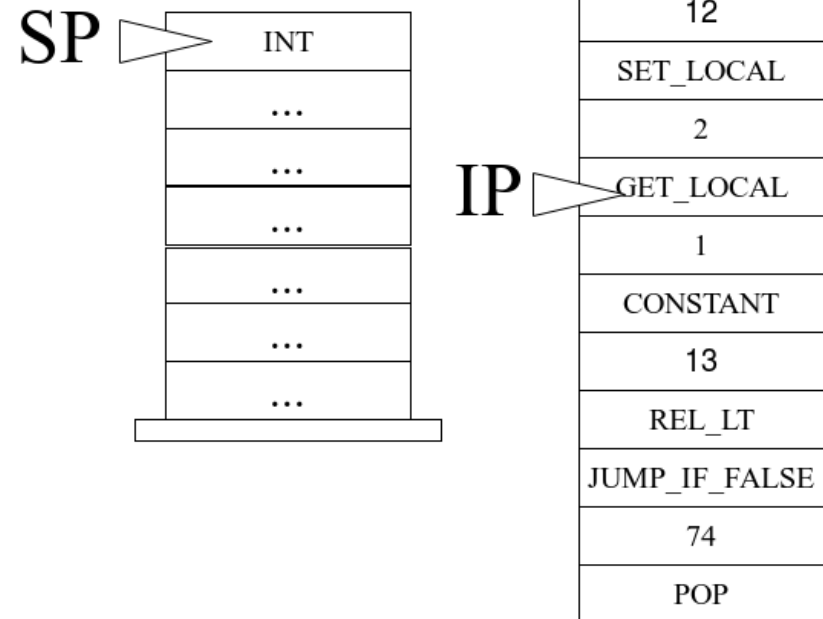
`GET_HEAP` 0 3 1 2 1 1 1 0



STACK-BASED VS REGISTER-BASED

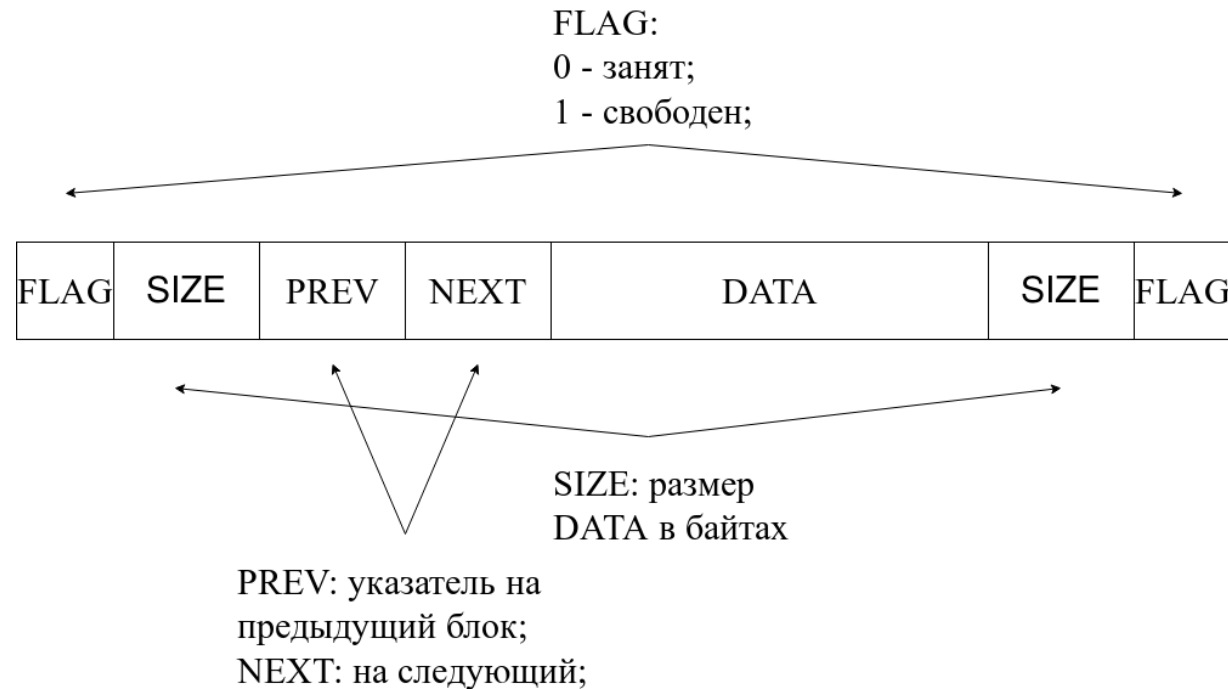


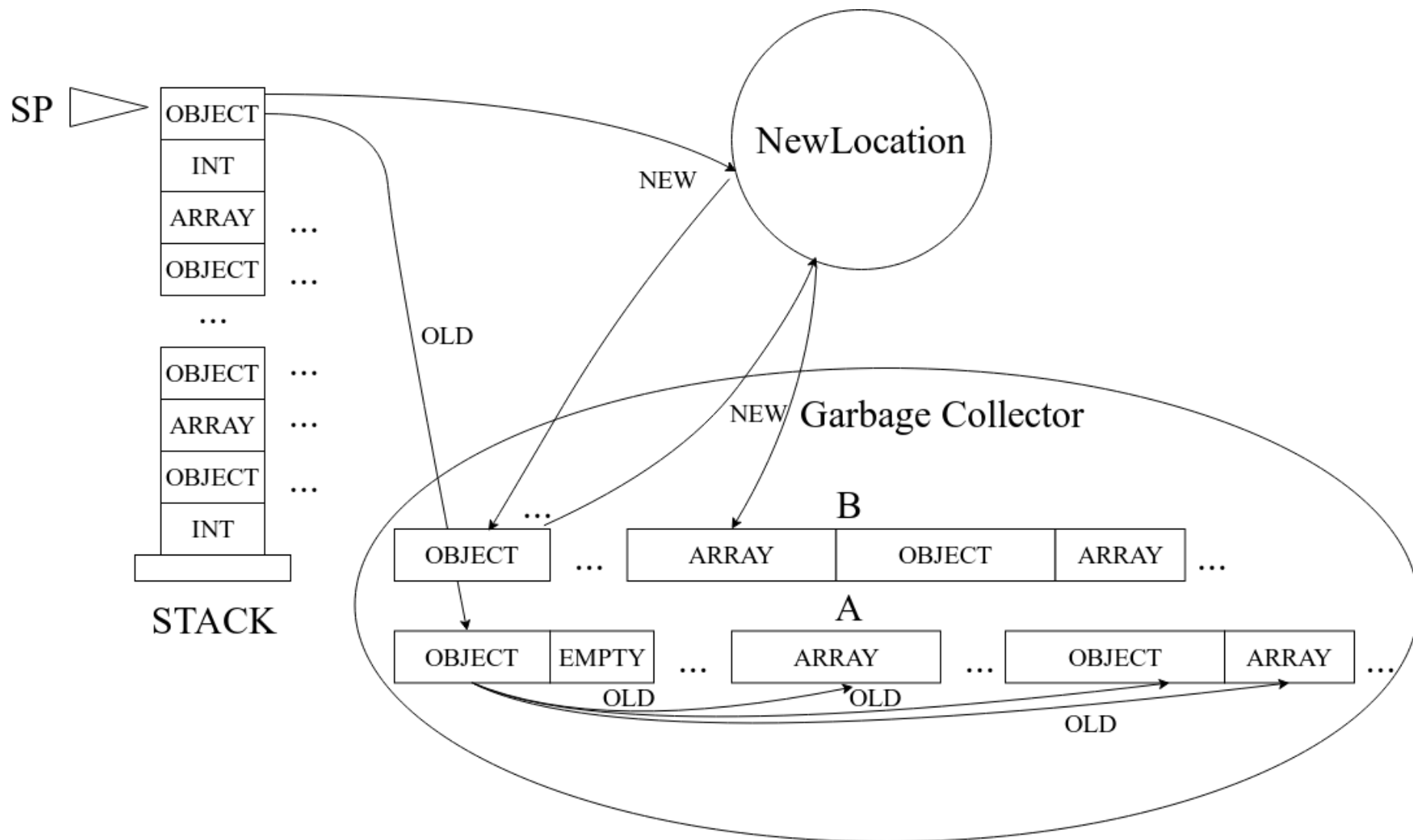
- Стековая архитектура ВМ;
- Два регистра:
 - SP (Stack Pointer);
 - IP (Instruction Pointer);



Сборщик мусора

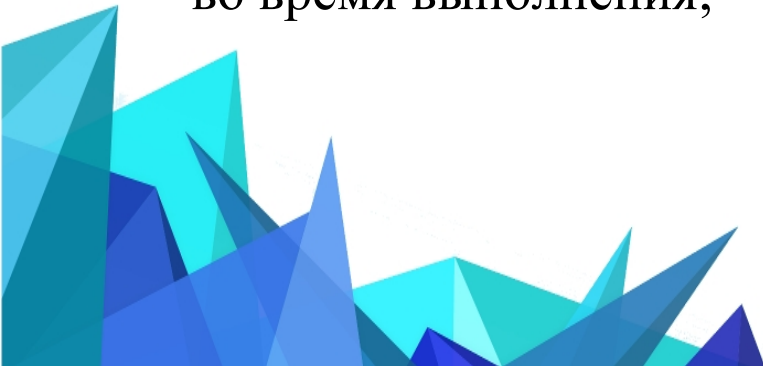
- Аллокатор со стратегией “наилучшего подходящего”;
- Сборщик мусора Чейни;





Реализация

- Pure C99 (практически C89);
- Нет зависимостей;
- Легкая расширяемость;
- Дамп всех возможных фаз интерпретатора в xml:
 - Лексера;
 - Парсера;
 - Генератора байткода;
- Трассировка программы во время выполнения;



Тестирование

- Тестирование корректности работы управляющих конструкций языка;
- Тестирование корректности работы аллокатора и сборщика мусора;
- Тестирование всего интерпретатора на отсутствие утечек памяти даже при некорректных входных данных (для этого использовался Valgrind);
- Тестирование всего интерпретатора на программе, “моделирующей” сражение двух армий;
- Производительность на уровне Python 3;

