Семинар 11: Синтаксис языка

Igor Zhirkov

16 ноября 2021 г.

Содержание

1 Аспекты языка (20 минут)

В каждом языке программирования есть три аспекта: синтаксис, семантика и прагматика.

1.1 Синтаксис

Синтаксис определяет, какие программы на этом языке считаются “правильно сформированными”.

Примеры синтаксически-неправильных программ:

```
int x = "hello" ; // несоответствие типов

int x = y + 23 ; // если переменная y не объявлена

int int = 4;   // после int ожидается имя переменной,
// не может совпадать с ключевым словом int
```

1.2 Семантика

Семантика определяет значение конструкций языка. Описывать её можно на естественном языке (например, в стандарте языка C она описана на английском) или на языке формальной логики. Семантика наделяет каждую конструкцию каким-то поведением в зависи-

мости от контекста, например, сопоставляет операции “*” смысл “умножить левую часть на правую, если звёздочка между двумя выражениями”, или смысл “слева от звёздочки — тип-указатель” в другом контексте.

Бывает, что у какой-то конструкции нет смысла, хотя программа и синтаксически-корректна. Нет смысла значит что стандарт языка или явно отказался от того, чтобы определять, что же произойдёт, или просто обошёл этот момент стороной, и теперь нам непонятно. Такая ситуация называется *неопределённым поведением*.

Примеры программ, в которых нет семантики:

```
int y = 0;
int x = 1 / y; // деление на ноль

int z = *(NULL); // обращение по указателю NULL неопределено.


int k = x++ + ++x; // неизвестно, какая операция выполнится раньше:
// x++ или ++x, отсюда и результат может быть разный.
```

1.3 Прагматика

Прагматика в языках программирования — то, как меняется семантика при трансляции в целевую архитектуру. Например, в программе на C есть неопределённое поведение, а в программе на ассемблере — нет. Значит, после трансляции неопределённое поведение доопределится — а как? Некоторые дополнительные директивы для компилятора и конструкции языка нужны, чтобы учитывать особенности вычислительной машины, реализации, для повышения эффективности

Примеры прагматических аспектов:

- Выравнивание в памяти
- Упаковка полей структур (как поля расположены в памяти, есть ли между ними отступы?)
- Выбор способа трансляции формул с плавающей точкой.

Например, при компиляции `gcc` может принимать опцию `-ffast-math`, которая:

- Выключает строгую совместимость со стандартом IEEE
- Сокращает количество записей в переменную `errno`.
- Вносит предположение что в формулах не встречается NaN, ноль или бесконечность.
- и включает ещё некоторые оптимизации.

Вопрос скомпилируйте следующий файл и посмотрите содержимое секции `.data` с помощью `objdump`. По какому адресу начинается переменная `x`? Попробуйте убрать `_Alignas(128)` и объясните эффект.

2 Синтаксис

Мы привыкли смотреть на программы как на текст. Однако текстовое представление имеет ряд недостатков.

Прежде всего, текст может неоднозначно трактоваться.

Как думаете, что выведет эта программа если `x = -4`?

```
if (x > 0)
  if (y > 0) {
    print(”yes”);
  }
  else {
    print(”no”);
  }
```

А если посмотреть на неё так?

```
if (x > 0)
  if (y > 0) {
    print(”yes”);
  }
else {
  print(”no”);
}
```

Вопрос напишите минимальный пример с таким кодом, скомпилируйте его со строгим соответствием стандарту C17 (`-std=c17 -pedantic -Wall`). Что выведет компилятор? Объясните сообщение.

Как мы видим из этого примера, в плохо спроектированных языках текстовое представление бывает неоднозначным.

Более того, многие элементы программы не несут никакой операционной нагрузки (комментарии, ключевые слова). Некоторые очевидно одинаковые программы представляются разным текстом:

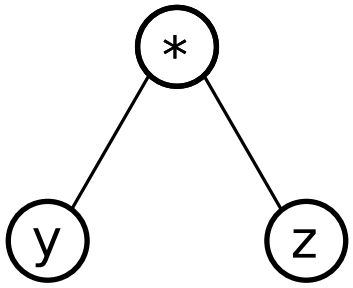
```
int x = a * b + c;
int x = (a * b) + c;
int x = ((a * b) + c);
int x = (((a * b) + c));
// скобки нужны только чтобы правильно разобрать выражение
```

```
// но здесь они избыточны
// этот комментарий тоже ни на что не влияет,
// хотя в тексте программы присутствует.
```

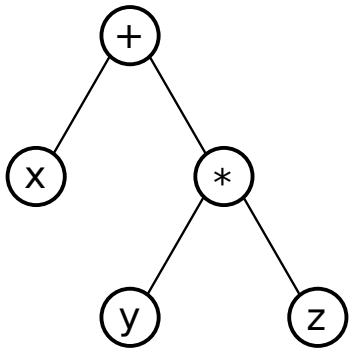
О программе удобнее думать не как о её текстовом представлении. Текст это лишь проекция её структуры на экран; на самом деле лучшее представление о её структуре даёт так называемое *дерево абстрактного синтаксиса*, abstract syntax tree, AST.

В нём каждой конструкции языка сопоставляется тип вершин дерева, например:

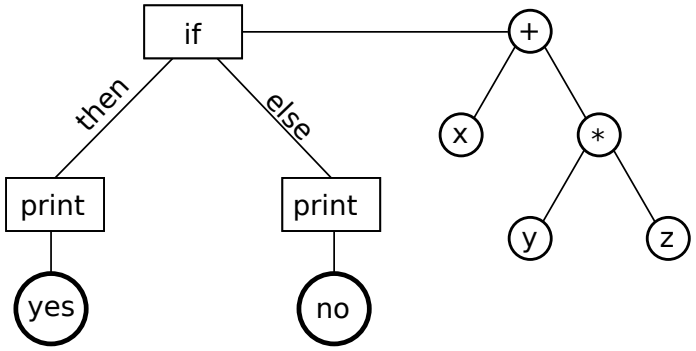
y*z



x+y*z




```
if (x+y*z) {
  print(”yes”);
  }
else {
  print(”no”);
  }
```



Смысл такого представления программы в том, что оно чётко и недвусмысленно определяет её структуру и не зависит от форматирования (пробелы, переносы строк и т.д.). Кроме того, оно не зависит и от точных названий ключевых слов. Действительно, станет ли язык C другим, если всё, что мы поменяем — заменим ключевое слово `while` на `whiiile` с тем же смыслом?

На таком древовидном представлении программы легче говорить про её семантику. Значение языковых конструкций задаётся именно для вершин AST. Где это встречается:

- При разработке компиляторов или интерпретаторов практически первое, что происходит с программой — её преобразование в AST.

Также этим занимаются IDE и текстовые редакторы.

- Форматы хранения данных типа JSON тоже представимы как AST.
- Разметки текста (Markdown, Org…)
- Вообще любые структурированные текстовые форматы (файлы конфигураций и т.д.)
- В документах описывающих языки программирования (стандарты языков).

Например, вот описание конструкции `_Alignas` из стандарта C. Как видите, в нём используется `type-name` и `constant-expression` — типы узлов AST-дерева кода (немного упрощая).

6.7.5 Alignment specifier

Syntax

1 *alignment-specifier:*
_Alignas (*type-name*)
_Alignas (*constant-expression*)

Constraints

2 An alignment specifier shall appear only in the declaration specifiers of a declaration, or in the *specifier-qualifier* list of a member declaration, or in the type name of a compound literal. An alignment specifier shall not be used in conjunction with either of the storage-class specifiers **typedef** or **register**, nor in a declaration of a function or bit-field.

3 The constant expression shall be an integer constant expression. It shall evaluate to a valid fundamental alignment, or to a valid extended alignment supported by the implementation for an object of the storage duration (if any) being declared, or to zero.

4 An object shall not be declared with an over-aligned type with an extended alignment requirement not supported by the implementation for an object of that storage duration.

- Знание AST бывает полезным при попытке понять сообщения об ошибках, выдающиеся компилятором (см. первое задание :)

Теперь изучим AST простой программы на Java с помощью `https://astexplorer.net/`

```
public class Program {
    public static void main(String[] args) {
        int x = 100;
        if (x > 0) {
            System.out.println(”Greater than zero”);
        }
        else {
            System.out.println(”Less than or equal to zero”);
        }
    }
}
```

Вопрос найдите в AST для примера выше имя класса. Какие поля хранятся в узле этого дерева? Как вы думаете, зачем?

Вопрос найдите в AST для примера выше конструкцию if. Добавьте код так, чтобы повторить ситуацию с вложенным в if предложением if-else из первого примера. Нарисуйте примерный вид AST для него.

Теперь изучим простой принтер для AST-дерева. Дерево будем задавать прямо в коде с помощью небольшого доменно-специфичного языка, который определим прямо в программе.

Вопрос изучите файл `printer0.c`.

```c
#include <inttypes.h>
#include <malloc.h>
#include <stdio.h>
#include <string.h>

struct AST {
  enum AST_type { AST_BINOP, AST_UNOP, AST_LIT } type;
  union {
    struct binop {
      enum binop_type { BIN_PLUS, BIN_MINUS, BIN_MUL } type;
      struct AST *left, *right;
    } as_binop;
    struct unop {
      enum unop_type { UN_NEG } type;
      struct AST *operand;
    } as_unop;
    struct literal {
      int64_t value;
    } as_literal;
  };
};

/* DSL */
static struct AST *newnode(struct AST ast) {
  struct AST *const node = malloc(sizeof(struct AST));
  *node = ast;
  return node;
}

struct AST _lit(int64_t value) {
  return (struct AST){AST_LIT, .as_literal = {value}};
}

struct AST *lit(int64_t value) {
  return newnode(_lit(value));
}
struct AST _unop(enum unop_type type, struct AST *operand) {
  return (struct AST){AST_UNOP, .as_unop = {type, operand}};
}

struct AST *unop(enum unop_type type, struct AST *operand) {
```

```
  return newnode(_unop(type, operand));
}

struct AST _binop(enum binop_type type, struct AST *left, struct AST *right) {
  return (struct AST){AST_BINOP, .as_binop = {type, left, right}};
}
struct AST *binop(enum binop_type type, struct AST *left, struct AST *right) {
  return newnode(_binop(type, left, right));
}
#define DECLARE_BINOP(fun, code)                                              \
  struct AST *fun(struct AST *left, struct AST *right) {                      \
    return binop(BIN_##code, left, right);                                    \
  }
DECLARE_BINOP(add, PLUS)
DECLARE_BINOP(mul, MUL)
DECLARE_BINOP(sub, MINUS)

#undef DECLARE_BINOP
#define DECLARE_UNOP(fun, code)                                               \
  struct AST *fun(struct AST *operand) {                                      \
    return unop(UN_##code, operand);                                          \
  }
DECLARE_UNOP(neg, NEG)
#undef DECLARE_UNOP
/* printer */

static const char *BINOPS[] = {
    [BIN_PLUS] = "+", [BIN_MINUS] = "-", [BIN_MUL] = "*"};
static const char *UNOPS[] = {[UN_NEG] = "-"};

typedef void(printer)(FILE *, struct AST *);

void print(FILE *f, struct AST *ast);

void print_binop(FILE *f, struct AST *ast) {
  fprintf(f, "(");
  print(f, ast->as_binop.left);
  fprintf(f, ")");
  fprintf(f, "%s", BINOPS[ast->as_binop.type]);
  fprintf(f, "(");
  print(f, ast->as_binop.right);
```

```
  fprintf(f, ")");
}
void print_unop(FILE *f, struct AST *ast) {
  fprintf(f, "(%s", UNOPS[ast->as_unop.type]);
  print(f, ast->as_unop.operand);
  fprintf(f, ")");
}
void print_lit(FILE *f, struct AST *ast) {
  fprintf(f, "%" PRId64, ast->as_literal.value);
}

static printer *ast_printers[] = {
    [AST_BINOP] = print_binop, [AST_UNOP] = print_unop, [AST_LIT] = print_lit};

void print(FILE *f, struct AST *ast) {
  if (ast)
    ast_printers[ast->type](f, ast);
  else
    fprintf(f, "<NULL>");
}

int main() {
  struct AST *ast = NULL;
  print(stdout, ast);
  return 0;
}
```

Допишите в `main` код, чтобы вывести следующие выражения (можно со скобками):

- 999 + 728
- 4+2*9

Расширьте этот пример, добавив тип вершин AST для деления и принтер для него. Выведите следующее выражение:

- (3+5) * (9 / 7)