Концепции языков программирования

Выражения и операторы присваивания

Выражения и операторы присваивания

- Введение
- Арифметические выражения
  - Операторы
  - Приоритет операций
  - Ассоциативность
  - Условные выражения
  - Порядок вычисления операндов и побочные эффекты
- Перегруженные операторы
- Преобразования типов
- Выражения отношений и булевы выражения
- Сокращенное вычисление
- Операторы присваивания
- Смешанное присваивание

Введение

- Выражения — основное средство определения вычислений в языке программирования
- Чтобы понять, как вычисляется значение выражения, нужно знать порядок вычисления операторов и операндов
- Существенная черта императивных языков — доминирующая роль операторов присваивания

Арифметические выражения

- Арифметические вычисления — один из побудительных мотивов создания первых языков программирования
- Арифметические выражения состоят из
  - операторов,
  - операндов,
  - скобок и
  - вызовов функций

Арифметические выражения

Вопросы проектирования

- Вопросы проектирования арифметических выражений
  - приоритет операторов
  - правила ассоциативности для операторов
  - порядок вычисления операндов
  - побочные эффекты вычисления операндов
  - перегрузка операторов
  - смешанные выражения

Арифметические выражения

Операторы

- Унарный оператор имеет один операнд
- Бинарный оператор имеет два операнда
- Тернарный оператор имеет три операнда

Арифметические выражения

Приоритет операторов

- Правила приоритета операторов при вычислении выражения определяет порядок, в котором вычисляются «смежные» операторы с различным приоритетом
- Типичные приоритеты
  - скобки
  - унарные операторы
  - ** (если есть в языке)
  - *, /
  - +, −

Пример

`a * b + c`

Если `a = 3, b = 4, c = 5`, то `a * b + c = 17`

Арифметические выражения

Правила ассоциативности операторов

- Правила ассоциативности операторов при вычислении выражения определяют порядок, в котором вычисляются смежные операторы с одинаковым приоритетом
- Типичные правила ассоциативности
  - Слева направо. кроме **, который вычисляется справа налево
  - Иногда унарные операторы вычисляются справа налево
- Правила приоритета и ассоциативности можно обойти при помощи скобок

Арифметические выражения

Правила ассоциативности операторов

- APL отличается от прочих языков:

  все операторы имеют одинаковый приоритет и
  все операторы вычисляются справа налево

Пример

$\mathtt{A} \times \mathtt{B} + \mathtt{C}$
Если $\mathtt{A} = 3, \mathtt{B} = 4, \mathtt{C} = 5$, то $\mathtt{A} \times \mathtt{B} + \mathtt{C} = 27$

Арифметические выражения

Правила ассоциативности операторов

- В математике сложение ассоциативно: порядок вычисления операторов не имеет значения

$$\texttt{A} + \texttt{B} + \texttt{C} + \texttt{D}$$

  → Компилятор оптимизирует код, изменяя порядок
- A и C — большие положительные числа
- B и D — отрицательные числа с большими абсолютными значениями
  → A + B + C + D — OK
  → сложение A и C вызывает переполнение

- Условные выражения
  - Языки, основанные на C (например, C, C++, C#)

Пример

```
average = (count == 0)?0 : sum/count
```

- Вычисляется как если бы было записано следующим образом:

```
if (count == 0) {average = 0}
else {average = sum/count}
```

- Порядок вычисления операндов
  1. Переменные: получают значение из памяти
  2. Константы: иногда получают значение из памяти; иногда константа — часть инструкции машинного языка
  3. Выражения в скобках: сначала вычисляются все операнды и операторы

Арифметические выражения

Возможные побочные эффекты

- Функциональные побочные эффекты: когда функция изменяет значение входного/выходного параметра или нелокальной переменной
- Проблема, связанная с функциональными побочными эффектами:
  - Когда функция, вызываемая в выражении, меняет значение другого операнда выражения:

Пример (Изменение параметра)

```
a = 10;
/* допустим, что fun меняет значение своего параметра */
b = a + fun(a)
```

Функциональные побочные эффекты

- Два возможных решения проблемы
  1. Запретить побочные эффекты в определении языка
     - Никаких входных/выходных параметров у функций
     - Никаких нелокальных ссылок в функциях
     - Преимущество: работает
     - Недостаток: отсутствие гибкости
  2. Зафиксировать порядок вычисления операндов в определении языка (Java)
     - Недостаток: ограничивает возможности компилятора при оптимизации

Перегруженные операторы

- Использование оператора для более чем одной цели называется перегрузкой оператора
- Некоторые варианты перегрузки общеприняты (например, + для `int` и `float`)
- Осторожно:

```
float avg;
int sum = 100, count = 3;
avg = sum/count;
```

`avg` имеет значение 30 (а не 30,33333)

- Можно избежать введением новых символов (например, в языке Pascal для целочисленного деления используется **div**)

Перегруженные операторы

- Некоторые операторы потенциально проблематичны (например, * и & в C и C++)
  - Потеря возможности обнаружить ошибку на этапе компиляции (пропуск операнда должен обнаруживаться как ошибка)
  - Некоторое ухудшение читабельности

Перегруженные операторы

Продолжение

- В C++, C# и Ada пользователь может самостоятельно перегружать операторы, что особенно полезно при работе с абстрактными типами данных

Пример (Для абстрактного типа данных матриц)

`A * B + C * D`

вместо

`MatrixAdd(MatrixMult(A, B), MatrixMult(C, D))`

- Потенциальные проблемы:
  - Пользователи могут определить бессмысленные операции
  - Читабельность может пострадать даже тогда, когда операторы имеют смысл
  - В C++, но не в Java, но снова в C#

Преобразования типов

Сужающее преобразование : объект преобразуется к типу, который не может включить все значения исходного типа, например, `float` к `int`

Расширяющее преобразование : объект преобразуется к типу, который может включить, по крайней мере, все приблизительные значения исходного типа, например, `int` к `float`

Преобразования типов

Смешанные выражения

- Смешанное выражение — выражение, содержащее операнды разных типов
- Приведение — неявное преобразование типов
- Недостатки приведения:
  - меньше возможностей обнаружить ошибку при компиляции
  - проверка типов (надежность) против гибкости
- В большинстве языков для всех числовых типов в выражениях определены приведения, соответствующие расширяющим преобразованиям
- В языке Ada практически не применяются приведения в выражениях

Явные преобразования типов

Пример

C : `(int)angle`

Ada : `Float(sum)`

В языке Ada используется такой же синтаксис, как при вызове функций

Преобразования типов

Ошибки в выражениях

- Причины
  - Естественные арифметические ограничения, например, деление на ноль
  - Ограничения компьютерной арифметики, например, переполнение
- Часто игнорируются во время выполнения или обрабатываются при помощи исключительных ситуаций

Выражения отношений и булевы выражения

- Выражения отношений
  - Используются операторы выражений и операнды различных типов
  - Значение выражения — булево
  - Символы операторов в разных языках несколько различаются (!=, /=, .NE., <>, #)
  - Типичные операторы: ==, !=, >, <, >=, <=
  - В языке JavaScript также есть === и !===
    “7” == 7 : результат — true
    “7” === 7 : результат — false

Выражения отношений и булевы выражения

- Булевы выражения
  - Операнды — булевы и результат — булев

Пример (Операторы)

FORTRAN 77	FORTRAN 90	C	Ada
.AND.	and	&&	and
.OR.	or	||	or
.NOT.	not	!	not
			xor

Выражения отношений и булевы выражения

В C нет булевых типов

- В C нет булевых типов; используются значения типа `int`: 0 — для ‘ложь’, 1 — для ‘истина’
- Странное свойство выражений в C:
  `a` $<$ `b` $<$ `c` — допустимое выражение, но результат может показаться неожиданным:
  - Вычисляется левый оператор; результат — 0 или 1
  - Полученный результат сравнивается с третим операндом (т.е. `c`)

Выражения отношений и булевы выражения

Приоритет операторов

- Приоритет операторов в языках, основанных на C:

  постфиксные $++, --$
  унарные $+, -$, префиксные $++, --, !$
  $*, /, \%$
  бинарные $+, -$
  $<, >, <=, >=$
  $=, != $
  $\&\&$
  $||$

Сокращенные вычисления

- Выражение, в котором определение результата не требует вычисления всех операндов и/или операторов

Пример

$(13 * a) * (b/13 - 1)$

Если a — ноль, то не нужно вычислять $(b/13 - 1)$

Пример

$(a >= 0)\&\&(b < 10)$

Если $a < 0$ — ноль, то не нужно вычислять $(b < 10)$

Сокращенные вычисления

- Проблема, связанная с применением полного вычисления

```
index = 0

while ((index < length) && (LIST[index] != value))
   {index++;}
```

  - При `index = length`, обращение к `LIST[index]` вызовет ошибку индексации (если последний элемент в `LIST` имеет индекс `length - 1`)

Сокращенные вычисления

Продолжение

- C, C++ и Java: используют сокращенные вычисления для обычных булевых операторов (&& и ||), но также предоставляют побитовые булевы операторы, которые вычисляются полностью (& и |)
- Сокращенные вычисления делают проблематичными побочные эффекты в выражениях

Пример

$(\texttt{a} > \texttt{b}) \,||\, (\texttt{b}{+}{+} \,/\, 3)$

$\rightarrow$ Если $\texttt{a} > \texttt{b}$, то $\texttt{b}{+}{+}$ никогда не выполняется

Операторы присваивания

- Синтаксис в общем виде

  ⟨target_var⟩⟨assign_operator⟩⟨expression⟩
- Оператор присваивания
  - $=$ FORTRAN, BASIC, PL/I, C, C++, Java
  - $:=$ версии языка ALGOL, Pascal, Ada
- $=$ — плохой выбор, если он перегружен как оператор отношения равенства

Операторы присваивания

Условные целевые объекты

- Условные целевые объекты (C, C++ и Java)

  `(flag)? total : subtotal = 0`

  что эквивалентно

```
if (flag)
   total = 0
else
   subtotal = 0
```

Операторы присваивания

Составные операторы

- Способ сокращенной записи часто встречающегося вида присваивания
- Появился в языке ALGOL; заимствован языком C

Пример

`a = a + b`

записывается как

`a += b`

Операторы присваивания

Унарные операторы присваивания

- Унарные операторы присваивания в языках, основанных на C, комбинируют операции увеличения и уменьшения на единицу с присваиванием

Пример

`sum = ++count` (`count` увеличивается на единицу, присваивается переменной `sum`)

`sum = count++` (`count` присваивается переменной `sum`, увеличивается на единицу)

`count++` (`count` увеличивается на единицу)

`-count++` (`count` увеличивается на единицу, берется с противоположным знаком)

Присваивание как выражение

- В языках C, C++ и Java оператор присваивания имеет результат, который может использоваться в качестве операнда

Пример

`while ((ch = getchar()) != EOF){...}`
Выполняется `ch = getchar()`; результат (присвоенный переменной `ch`) используется как значение условия для оператора `while`

Присваивание как выражение

- Проблема в языке C

  `if (x = y){...}`

  вместо

  `if (x == y){...}`

  - Присваивание как бинарный оператор
  - Арифметические выражения в качестве булевых операндов
  - = и == имеют различные значения
- Java, C#: только булевы выражения в операторах if

Смешанное присваивание

- Операторы присваивания могут быть смешанными

Пример

```
int a,b;
float c;
c = a/b;
```

→ Преобразование типов после вычисления правой части

- В языке Pascal значения целочисленных переменных могут быть присвоены действительным переменным, но значения действительных переменных не могут быть присвоены целочисленным переменным
- В языках Java и C# осуществляются только расширяющие преобразования
- В языке Ada при присваивании преобразования не производятся

Резюме

- Выражения
- Приоритет и ассоциативность операторов
- Перегрузка операторов
- Смешанные выражения
- Различные формы присваивания