Концепции языков программирования

Структуры управления на уровне операторов

Структуры управления на уровне операторов

Уровни управления потоком

- В рамках одного выражения
- Между программными единицами
- Между операторами программы

Операторы управления потоком

Эволюция

- Операторы управления языка FORTRAN I основаны непосредственно на архитектуре IBM 704
- В 1960-ые гг. проводились многочисленные исследования и дискуссии по теме
  - Важный результат: Было доказано, что все алгоритмы, представимые в виде блок-схем, могут быть закодированы при помощи только операторов бинарного ветвления и циклов с предварительной проверкой условия

Структура управления

- Инструкция (оператор) управления — инструкция, которая осуществляет

  выбор между альтернативными путями выполнения

  или

  повтор выполнения блока инструкций

Структура управления

- Структура управления — оператор управления и операторы, выполнение которых он контролирует
- Вопрос проектирования
  - Следует ли структуре управления иметь несколько точек входа?

Операторы ветвления

- Оператор ветвления предоставляет возможность выбора между двумя или более ветвями выполнения программы
- Две общие категории:
  - Двух-вариантные операторы ветвления
  - Многовариантные операторы ветвления

Двух-вариантные операторы ветвления

- Общая форма

```
if control_expression
   then clause
   else clause
```

- Вопросы проектирования
  - Каковы внешний вид и тип управляющего выражения?
  - Как указываются предложения `then` и `else`?
  - Как определить значение вложенных операторов ветвления?

Двух-вариантные операторы ветвления

Примеры

- FORTRAN: IF (boolean_expr) statement
- Проблема: можно выбрать только один оператор; если нужно выбрать больше, необходимо использовать GOTO

Пример

```
IF (.NOT.condition)GOTO20
   ...
20 CONTINUE
```

- Отрицательная логика ухудшает читабельность
- Эта проблема была решена в языке FORTRAN 77
- Большинство последующих языков разрешают составные операторы в качестве выбираемого сегмента

Двух-вариантные операторы ветвления

Примеры

- ALGOL 60:
  if (boolean_expr)
    then statement (оператор then)
    else statement (оператор else)
- Операторы могут быть простыми или составными

Вложение операторов ветвления

Пример (Java)

```
if (0 == sum)
   if (0 == count)
      result = 0;
else result = 1;
```

- Какому `if` соответствует `else`?

Вложение операторов ветвления

Пример (Java)

```
if (0 == sum)
   if (0 == count)
      result = 0;
else result = 1;
```

- Какому `if` соответствует `else`?
- Статическое семантическое правило языка Java: `else` соответствует ближайшему `if`

Вложение операторов ветвления

Продолжение

- Чтобы реализовать альтернативную семантику, нужно использовать составные операторы:

```
if (0 == sum) {
   if (0 == count)
      result = 0;
}
else result = 1;
```

- Это решение работает также в C, C++ и C#
- В языке Perl требуется, чтобы все операторы then и else были составными

Многовариантные операторы ветвления

- Позволяют осуществлять выбор из произвольного числа операторов или групп операторов
- Вопросы проектирования:
  1. Каковы внешний вид и тип управляющего выражения?
  2. Как указывать альтернативные сегменты кода?
  3. Должен ли поток выполнения, проходящий через структуру, включать только один из альтернативных сегментов?
  4. Что делать, если значение выражения не соответствует ни одному из представленных вариантов?

Многовариантные операторы ветвления

Примеры

- Первые многовариантные операторы ветвления:
  - Арифметический `IF` языка FORTRAN
    `IF` (арифметическое выражение) `N1,N2,N3`
  - Для выбора сегментов, состоящих из нескольких операторов, требуется использование `GOTO`
  - Нет инкапсуляции (выбираемые сегменты могут располагаться где угодно)

Многовариантные операторы ветвления

Примеры

- Современные многовариантные операторы ветвления:
  - Оператор `switch` языка C

```
switch (выражение) {
    case const_expr_1 : stmt_1;
    ...
    case const_expr_n : stmt_n
    [default : stmt_n+1]
}
```

Многовариантные операторы ветвления

Примеры

- Особенности оператора `switch` языка C
  1. Управляющее выражение должно иметь целочисленный тип
  2. Альтернативные сегменты должны быть последовательностями операторов, блоков или составных операторов
  3. При выполнении конструкции может быть выполнено любое число сегментов (выход из конструкции в конце альтернативных сегментов не осуществляется автоматически)
  4. Предложение `default` — для значений, не указанных явным образом (при выборе такого значения в отсутствие `default` оператор ветвления не приводит ни к каким действиям)

Многовариантные операторы ветвления

Примеры

- C# надежнее C:

  статическое семантическое правило: каждая выбираемая инструкция должна заканчиваться инструкцией ветвления `break` или `goto`
- C#: управляющее выражение может быть булевым, символьным или строковым
- PHP: управляющее выражение может быть строковым, целочисленным или с плавающей точкой

Многовариантные операторы ветвления

Примеры

- Оператор `case` языка Ada

  `case` (выражение) `is`
  `when` список альтернатив => `stmt_sequence`;
  ...
  `when` список альтернатив => `stmt_sequence`;
  [`when others` => `stmt_sequence`;]
  `end case`

- Надежнее, чем оператор `switch` языка C (как только завершается выполнение stmt_sequence, управление переходит к первому оператору, следующему за оператором `case`)

Многовариантное ветвление при помощи `if`

- Многовариантное ветвление может быть задано как непосредственное расширение двух-вариантного ветвления при помощи else–if

Пример (Ada)

```
if...
  then...
elsif...
  then...
elsif...
  then...
  else...
end if
```

Операторы цикла

- Многократное выполнение оператора или составного оператора осуществляется посредство итераций или рекурсии
- Общие вопросы проектирования операторов цикла
  1. Как управлять итерациями?
  2. Где должен находиться управляющий механизм цикла?

Циклы со счетчиком

- В цикле со счетчиком есть переменная цикла и способ задания начального и конечного значений переменной и значения шага
- Вопросы проектирования
  1. Каковы тип и область видимости переменной цикла?
  2. Каким должно быть значение переменной цикла по завершении цикла?
  3. Можно ли изменять значения переменной цикла или параметров цикла в теле цикла, и должно ли такое изменение влиять на управление циклом?
  4. Нужно ли вычислять значение параметров цикла лишь однажды или при каждой итерации?

Операторы цикла

Примеры

- Синтаксис FORTRAN 90
  `DO label var = start, finish[, stepsize]`
- Шаг может быть любым, но не нулевым
- Параметры могут быть заданы выражениями
- Особенности:
  1. Переменная цикла должна иметь тип `INTEGER`
  2. По завершении цикла переменная цикла сохраняет свое последнее значение
  3. Значение переменной цикла не может быть изменено в цикле, но параметры могут быть изменены; так как они вычисляются лишь однажды, это не влияет на управление циклом
  4. Параметры цикла вычисляются лишь однажды

Операторы цикла

Примеры

- FORTRAN 95: вторая форма
  [`name` :] `DO varible` = `start`, `finish`[, `stepsize`]
  ...
  `END DO` [`name`]
  - Переменная цикла должна иметь тип `INTEGER`

Операторы цикла

Примеры

- Оператор `for` языка Pascal

```
for variable := initial (to | downto) final do
    statement
```

- Особенности:
  1. Переменная цикла имеет порядковый тип и обычную область видимости
  2. По завершении цикла значение переменной цикла не определено
  3. Значение переменной цикла не может быть изменено в цикле, но параметры могут быть изменены; так как они вычисляются лишь однажды, это не влияет на управление циклом
  4. Параметры цикла вычисляются лишь однажды

Операторы цикла

Примеры

- Оператор `for` языка Ada

```
for var in [reverse] discrete_range loop
   ...
end loop
```

- Особенности:
  1. Тип переменной цикла — под-область целочисленного или перечислимого типа; область видимости — цикл
  2. По завершении цикла переменная цикла невидима
  3. Значение переменной цикла не может быть изменено в цикле, но параметры могут быть изменены; так как они вычисляются лишь однажды, это не влияет на управление циклом
  4. Параметры цикла вычисляются лишь однажды

Операторы цикла

Примеры

- Оператор `for` языка C

  `for` ([`expr_1`]; [`expr_2`]; [`expr_3`]) `statement`
- Выражения могут быть операторами или даже последовательностями операторов, разделенных запятыми
  - Значение выражения, состоящего из нескольких операторов, — это значение последнего оператора в выражении
- Нет явной переменной цикла
- В цикле можно менять все
- Первое выражение вычисляется один раз, но другие два — при каждой итерации

Операторы цикла

Примеры

- Два отличия C++ от C:
  1. Управляющее выражение может также быть булевым
  2. Инициализирующее выражение может включать определения переменных (область действия — от определения до конца тела цикла)
- Java и C#
  - Отличаются от C++ тем, что управляющее выражение должно быть булевым

Инструкция for в языке Python

- Общая форма

```
for loop_variable in object:
    Loop_body
else:
    Else clause
```

- Объект может быть интервалом

Пример

```
...in range(5, 11, 2)
```

Операторы цикла

Логически управляемые циклы

- Управление повторением основано на булевом выражении
- Вопросы проектирования:
  - Проверять условие до или после тела цикла?
  - Должен ли логически управляемый цикл быть особым видом цикла со счетчиком?
    выражение вместо счетчика
- Общий вид:

```
while (ctrl_expr)
   loop body
```

```
do
   loop body
while (ctrl_expr)
```

Операторы цикла

Логически управляемые циклы: Примеры

- В языке Pascal есть самостоятельные операторы логически управляемого цикла с проверкой условия в начале и в конце (`while – do` и `repeat – until`)
- В C и C++ также есть оба вида, но управляющее выражение в версии с проверкой в конце обрабатывается точно так же, как и в случае с проверкой в начале (`while – do` и `do – while`)
- Java — как C, но управляющее выражение должно быть булевым (и тело цикла имеет единственную точку входа — в начале: в языке Java нет `goto`)

Операторы цикла

Логически управляемые циклы: Примеры

- В языке Ada есть цикл с проверкой условия в начале, но нет цикла с проверкой в конце
- В FORTRAN 77 и 90 нет ни того, ни другого
- В языке Perl есть два логически управляемых цикла с проверкой в начале — `while` и `until`, но нет логически управляемого цикла с проверкой в конце

Операторы цикла

Циклы с механизмами управления, размещенными пользователем

- Иногда удобно иметь возможность самостоятельно решить, где осуществлять проверку условия выхода из цикла (кроме как в начале или конце цикла)
- Простое решение для простых циклов (например, `break`)
- Вопрос проектирования для вложенных циклов
  - Должна ли быть возможность выйти одновременно из более чем одного цикла?

Операторы цикла

Циклы с механизмами управления, размещенными пользователем, `break` и `continue`

- C, C++: оператор `break`
- Безусловный выход; для любого цикла или `switch`; только один уровень
- В языках Java и C# есть оператор `break` с меткой: управление передается по метке
- Альтернатива: оператор `continue` — не выполняются оставшиеся операторы в данной итерации, но выход из цикла не происходит

Операторы цикла

Циклы с механизмами управления, размещенными пользователем, break и continue

Пример (`break` в C#)

```
outerLoop:
   for (row = 0; row < numRows; row++)
      for (col = 0; col < numCols; col++) {
         sum += mat[row][col];
         if (sum > 1000.0)
            break outerLoop;
      }
```

Операторы цикла

Циклы с механизмами управления, размещенными пользователем, break и continue

Пример (continue в C/C++)

```
while (sum < 1000) {
   getnext(value);
   if (value < 0) continue;
   sum += value;
}
```

Операторы цикла

Циклы с механизмами управления, размещенными пользователем, break и continue

Пример (`break` в C/C++)

```
while (sum < 1000) {
   getnext(value);
   if (value < 0) break;
   sum += value;
}
```

Операторы цикла

Циклы с механизмами управления, размещенными пользователем, break и continue

Пример (C#)

```
void Go() {
   SetUp();
   for (;;) {    — Бесконечный цикл
     string c = f.GetControl();
     if (c == null) break;    — Выход из цикла
     ActionPerformed(c);
   }
   f.CloseGUI();
 }
```

Операторы цикла

Циклы, основанные на структурах данных

- Число итераций цикла зависит от числа элементов в структуре данных
- Управление осуществляется вызовом функции-итератора, возвращающей следующий элемент в некотором порядке, если таковой элемент имеется; в противном случае, выполнение цикла завершается
- В языке C пользовательский итератор можно симулировать при помощи `for`:

```
for (p = root; NULL == p; traverse(p)){
}
```

Операторы цикла

Циклы, основанные на структурах данных

- Оператор foreach языка C# перечисляет элементы массивов и прочих коллекций

```
Strings[] strList = {"Bob" "Carol", "Ted"};
foreach (Strings name in strList)
    Console.WriteLine("Name : {0}",name);
```

- Запись {0} указывает позицию, в которой нужно отбразить строку

Операторы цикла

Циклы, основанные на структурах данных

- Языки Java, Perl, JavaScript и PHP также располагают предопределенными итераторами для массивов

Итераторы и генераторы

```
class IteratorPattern {
  class MonthCollection : IEnumerable {
    string[] months = {"Jan", "Feb"...};
    public IEnumerator GetEnumerator () {
      foreach (string element in months)
        yield return element;
    }
  }
  static void Main() {
    MonthCollection collection = new
      MonthCollection();
    foreach (string n in collection)
      Console.WriteLine(n + "  ");
  }
}
```

LINQ

```
class IteratorPattern {
  static void Main() {
    Dictionary < string, int > daysInMonths = new
      Dictionary < string, int > {
       {"January", 31},   {"February", 28},
       {"March", 31},     {"April", 30},
       {"May", 31},       {"June", 30},
       {"July", 31},      {"August", 31},
       {"September", 30}, {"October", 31},
       {"November", 30}, {"December", 31}};
```

LINQ

продолжение

```
        var selection = from n in daysInMonths
                        where n.Key.Length > 5
                        select n;

            selection = from n in selection
                        where n.Value == 31
                        orderby n.Key
                        select n;

        foreach (KeyValuePair<string,int> n in selection)
            Console.WriteLine(n + "  ");
    }
}
```

Безусловный переход

- Передает управление в определеннное место программы
- Один из наиболее активно обсуждаемых вопросов в 1960-ых 1970-ых гг.
- Хорошо известный механизм — оператор `goto`
- Основная проблема — читабельность
- В некоторых языках нет оператора `goto` (например, Module-2 и Java)
- В языке C# есть оператор `goto` (можно использовать в операторе `switch`)
- Операторы выхода из цикла представляют собой ограниченные и закамуфлированные варианты оператора `goto`

Защищенные команды

- Предложены Дейкстрой
- Цель — поддержка новой методологии программирования, позволяющей обеспечить корректность программы на этапе разработки
- Основа двух лингвистических механизмов параллельного программирования (в CSP и Ada)
- Основная идея — если порядок вычисления не важен, он не должен указываться в программе

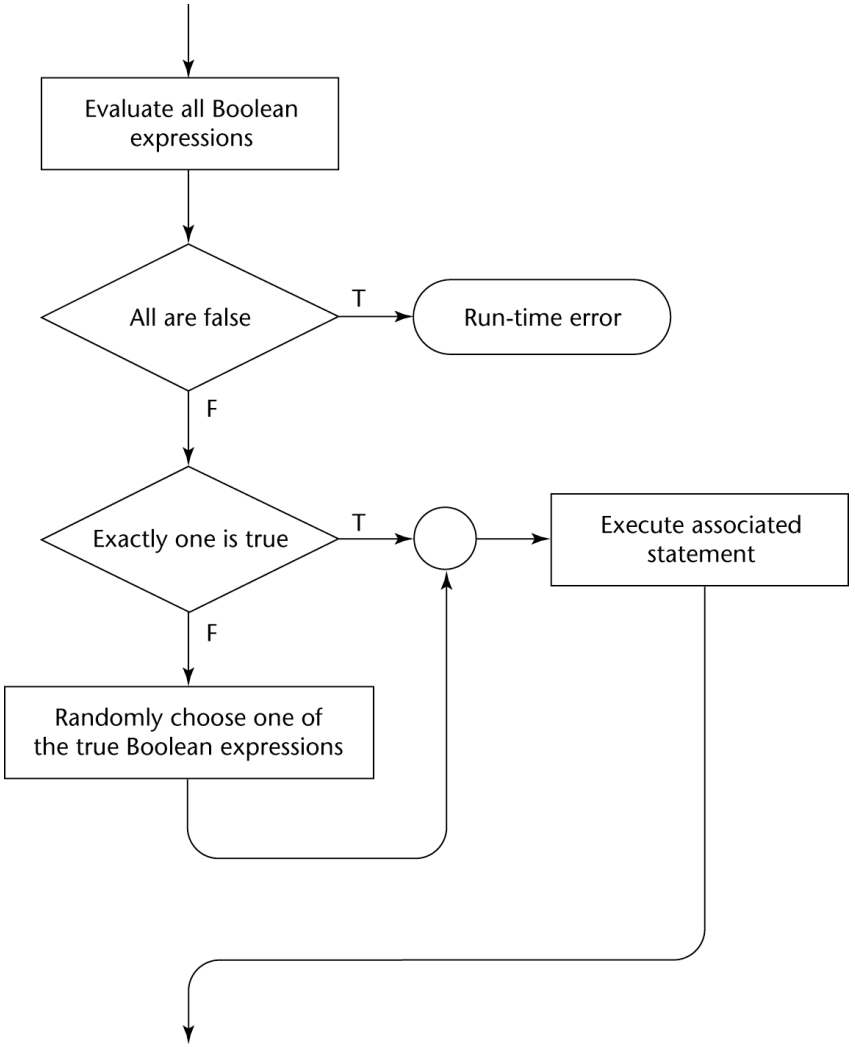
Защищенные команды ветвления

- Форма

  `if` ⟨`Boolean exp`⟩ → ⟨`statement`⟩
  [] ⟨`Boolean exp`⟩ → ⟨`statement`⟩
  ...
  [] ⟨`Boolean exp`⟩ → ⟨`statement`⟩
  `fi`

- Семантика:
  - Вычислить все булевы выражения
  - Если истинными являются более одного выражения, выбрать одно выражение недетерминированным образом
  - Если ни одно выражение не является истинным, происходит ошибка выполнения

Защищенные команды ветвления



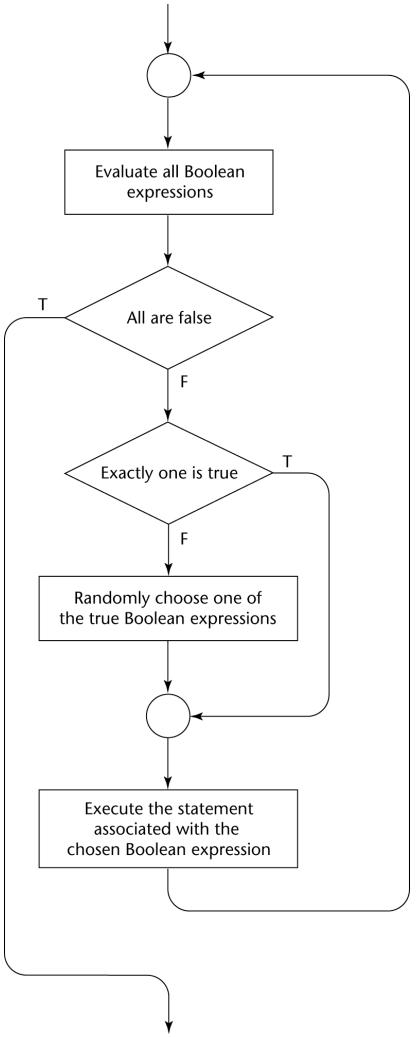
Защищенные команды цикла

- Форма

  do ⟨Boolean exp⟩ → ⟨statement⟩
  [] ⟨Boolean exp⟩ → ⟨statement⟩
  ...
  [] ⟨Boolean exp⟩ → ⟨statement⟩
  od

- Семантика: на каждой итерации
  - Вычислить все булевы выражения
  - Если истинными являются более одного выражения, выбрать одно выражение недетерминированным образом; затем перейти к следующей итерации
  - Если ни одно выражение не является истинным, выйти из цикла

Защищенные команды цикла



Защищенные команды

Мотивация

- Тесная связь между операторами управления и верификацией программ
- Верификация невозможна при наличии оператора `goto`
- Верификация возможна, если есть только операторы ветвления и циклы с проверкой логического условия в начале
- Верификация относительна проста, если есть только защищенные команды

Выводы

- Структуры управления на уровне операторов разнообразны
- Выбор операторов управления помимо операторов ветвления и циклов с проверкой логического условия в начале подразумевает компромисс между размером языка и лекгкостью написания программ
- В функциональных и логических языках программирования — особенные структуры управления