Объяснение механизмов ветвлений и циклов на основании использования концепции условных и безусловных переходов

Горшков С.С.
serggorsar@yandex.ru
НИУ Высшая школа экономики, г. Москва, Россия
Московский государственный университет им. М.В. Ломоносова, г. Москва, Россия

В статье рассматривается наглядный подход к изучению конструкции ветвлений и циклов на основании концепции условных и безусловных переходов с применением идей, используемых в языках ассемблера и польской инверсной записи. Приводится разбор этого подхода для базовых конструкций языков C/C++ (с проведением аналогий с языком Pascal), а также для специфичных для некоторых других языков – Python, Go и Perl, – базовых конструкций.

Ключевые слова: ПОЛИЗ, ассемблер, условный переход, безусловный переход.

Введение

При обучении программированию школьников старших классов и студентов младших курсов (в том числе специальностей, где программирование в каком бы то ни было виде не является профильным) могут возникнуть проблемы с пониманием базовых конструкций языка. Для учащихся, не занимавшихся раньше программированием, особую сложность из базовых конструкций языков императивного программирования представляют собой ветвления и циклы [1], т.к. они нарушают естественный порядок выполнения простейшего кода (а простейший код линеен). В разных языках программирования, начиная с более низкоуровневого Си и заканчивая современными языками высокого уровня, есть достаточно много различных видов циклов и их форм, в связи с чем может возникать путаница с последовательностью выполнения действий в программе. Однако на языке ассемблера [2] всё выглядит очень похоже, и понимая, как это устроено изнутри (даже без знания ассемблера), легко не ошибиться и масштабировать свои знания об устройстве циклов на самые разные языки.

Базовые нелинейные конструкции

В наиболее популярных языках программирования можно выделить три основные конструкции, отвечающие за нелинейности. Это условный оператор (if – else в C/C++, if – then – else в Pascal), оператор выбора (switch – case – default и case – of – else соответственно) и различного рода циклы. Разумеется, не во всех популярных языках высокого уровня есть подобные конструкции, например, в языке Python нет оператора выбора и цикла с постусловием (когда сначала

выполняется тело цикла, потом проверяется условие, таким образом, тело цикла выполнится хотя бы один раз, это do – while в C/C++, и repeat – until в Pascal), однако, их семантику можно реализовать с помощью других средств языка.

На самом деле, различные альтернативные варианты записи циклов являются, как правило, синтаксическим сахаром для обычного цикла с предусловием (while). Код программы можно несложно модифицировать, заменяя один из других вариантов цикла (ещё один пример – цикл for без аргументов в Go, являющийся бесконечным циклом без условия) на цикл с предусловием. Разумеется, нельзя забывать о инструкциях break и continue, break с меткой для перехода, связках while – else, for – else (и разумеется, их аналогами из других языков с другими названиями). Опишем особенности этих конструкций.

Инструкция break без аргументов позволяет выйти из цикла, то есть совершить переход на следующую строку кода после завершения блока цикла. Break с меткой для перехода (Golang, в Perl этой возможностью обладает инструкция last) позволяет прервать выполнение цикла и перейти не на следующую строчку за циклом, а на указанную метку. Это напоминает отчасти оператор goto в Basic, Fortran и других языках, но специфично для циклов. Continue же позволяет перейти на следующую итерацию цикла, игнорируя код, который располагается в теле цикла после этой инструкции. Else после циклов в языке Python, к примеру, имеет следующую семантику: блок else выполняется только если выход из цикла произошёл после полного завершения последней итерации (то есть не через break).

Отдельно стоит рассмотреть for в стиле языка Си: for (init; expr; increment), где один раз перед началом цикла выполняется блок init, далее проверяется на каждой итерации истинность условия, и в конце каждой итерации выполняется блок increment. Каждый из этих блоков может быть пустым. По сути, это аналогично блоку while с соответствующей последовательностью действий.

У учащихся бывает путаница с этим, что в какой последовательности выполняется, когда происходит выход из цикла (и куда), даже бывают вопросы, связанные с тем, почему после блока if не выполняется блок else в условном операторе. И в обратную сторону, почему при switch – case нужно писать break после альтернатив. В промышленном коде встречаются случаи, когда break забывают написать и происходит «проваливание» в следующую альтернативу. Ровно для контроля этого в стандарте C++17 появился атрибут [[fallthrough]], позволяющий лучше контролировать подобное поведение.

Описание синтаксиса псевдоязыка, основанного на ассемблере и польской обратной записи

Разумеется, не нужно объяснять простое через более сложное и глубокое. Создадим форму записи нелинейных переходов, использующую следующие компоненты:

– Метка (label). Некоторое имя, которое находится в некотором месте программы и обозначает, по сути, место в исходном коде программы, на которое можно перейти. После имени метки ставится двоеточие.

– Команда безусловного перехода jmp label. Осуществляет переход на метку label, т.е. после перехода выполнение программы продолжится с точки label.

– Команда условного перехода «Переход по лжи» ifFalse jmp label. Осуществляется переход на метку label, если только значение предыдущего выражения ложно.

– Все остальные конструкции кода, которые будем обозначать просто какими-то именами. Префикс body будет означать тело цикла или ветви, expr – некоторое выражение, результат которого будет проверяться в ifFalse, increment – также некоторый блок кода.

Запишем во введённой выше нотации простой оператор условного перехода if – else.

Листинг 1. Условное ветвление в нотации

```
if_branch:
expr                // проверяем условие
ifFalse jmp else_branch // если не выполнено, переходим в else
body_if                 // если выполнено, то выполняем ветвь if
jmp end             // и выходим из конструкции
  else_branch:
          body_else         // иначе выполняем код ветви else
  end:
```

Таким образом, мы видим, что если условие expr не истинно, то происходит условный переход на метку, где начинается код else-ветки, иначе происходит выполнение кода в ветви if. После её выполнения происходит безусловный переход на метку end, находящуюся после завершения ветвления. Ветка else же выполняется с метки else_branch, после окончания которой продолжается последовательное выполнение, и оно так же продолжится с метки end. У обучающихся отпадают вопросы, почему после if-ветки не выполняется else, потому что они верят в силу ассемблера.

Конструкция оператора выбора представляет собой в более явном виде реализацию идеи с метками и условными и безусловными переходами. В ней есть метки, на которые происходит переход, причем, break (или его аналоги), означают переход на метку, располагающуюся после всей конструкции. Если условие на равенство определенному значению не выполняется, то происходит переход на метку следующей альтернативы (условный переход). Предлагается реализовать учащимся в этой нотации оператор выбора самостоятельно по аналогии.

Представление различных операторов в описанной нотации

Обратимся теперь к записи цикла while с break и else-частью.

Листинг 2. Цикл while – else на языке Python

```
while expr:
    body_while_before_break
    if expr_break:
            break
    body_while_after_break
else:
    body_else
```

Этот код будет идентичен:

Листинг 3. Цикл while – else в нотации

```
start:
        expr                // условие нахождения в цикле
        ifFalse jmp else_branch // если не выполнено, выходим в else-ветку
        body_while_before_break // выполняем код в теле цикла
        expr_break          // вычисляем выражение для выхода
        ifFalse jmp end     // если оно истинно, то выходим
        body_while_after_break  // если нет, то продолжаем выполнение
        jmp start           // переходим на следующую итерацию
else_branch:
        body_else           // если вышли не по break, попали сюда
end:
```

Понятно, что для реализации цикла while с break без else вместо else_branch будет end, а для реализации цикла while с continue переход будет происходить не на метку end, а на метку start, чтобы перейти на следующую итерацию цикла.

Осталось разобрать for в стиле Си, остальное предлагается описать обучающимся для закрепления.

Листинг 4. Цикл for на языке Си

```
for(init;expr;increment){
    body;
}
```

Этот код будет идентичен:

Листинг 5. Цикл for в стиле Си в нотации

```
        init
start:
        expr        // условие нахождения в цикле
        ifFalse jmp end // выходим из цикла
        body        // выполняем тело цикла
        increment   // выполняем инкремент (как правило, i++ или i--)
        jmp start   // переходим на следующую итерацию цикла
end:
```

Заключение

Запись различных циклов и ветвлений в описанной нотации позволяет крайне наглядно показать последовательность исполнения команд и приблизит обучающих к пониманию, как всё устроено на самом деле.

Список литературы

[1] *Мельникова Д.Ю.* Решение задач, основанных на алгоритме, раскладывающем число на цифры // Сборник материалов, XI Всероссийская научно-практическая конференция «Информационные технологии в образовании», с. 155-158. Саратов, 2019

[2] *Е.А. Кузьменкова, В.С. Махнычев, В.А. Падарян.* Семинары по курсу "Архитектура ЭВМ и язык ассемблера" (учебно-методическое пособие). Часть 1. МАКС-Пресс, Москва, 2014