Стеки

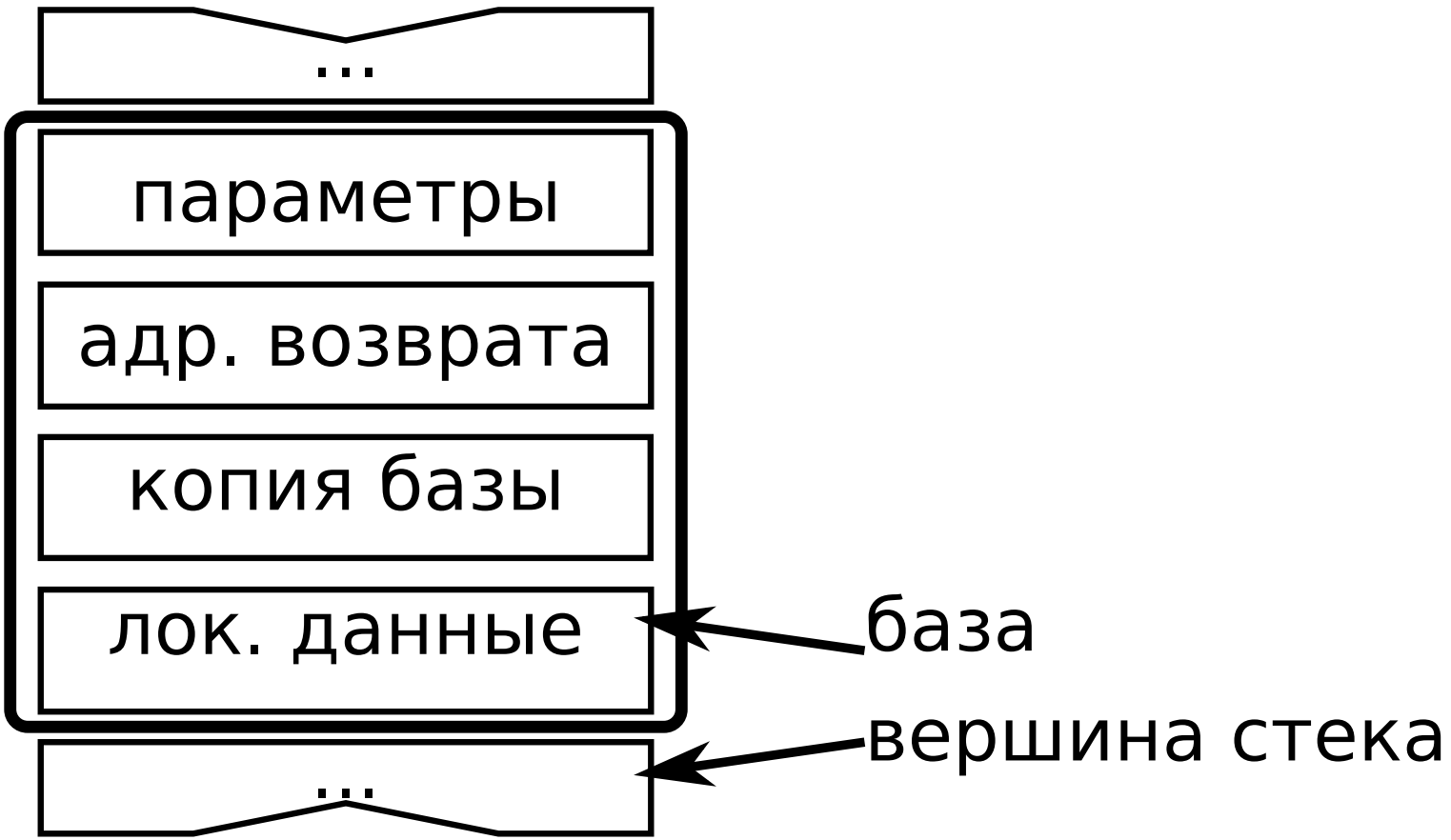
Д. В. Луцив

Кафедра системого программирования СПбГУ

CS103

Содержание

C (но уже не C++)


...
параметры
адр. возврата
копия базы
лок. данные
...
база
вершина стека

Статическая и динамическая цепочки

- Динамическая цепочка — последовательность кадров вызывающих друг друга процедур на стеке.
- Статическая цепочка (одно из определений) — последовательность кадров лексических контекстов.

Без передачи процедур

Проблема фунарга

▸ Проблема фунарга (в два слова, но с маленькой буквы).

При наличии в языке вложенных процедур необходимо:

- передавать представление замыкания — адрес кода процедуры и данные, необходимые для доступа к внешним лексическим контекстам;
- обеспечивать существование внешних контекстов до окончания возможности доступа к ним (целостность статической цепочки).

История вопроса

John McCarthy

Richard Matthew Stallman

Первые диалекты LISP поддерживали *динамическое* *лексическое замыкание* — поиск в динамической цепочке по имени. Самый старый диалект LISP, использующийся до сих пор — Emacs LISP — по умолчанию работает [illegible], хотя в новых версиях это можно переключить.

Помимо старых диалектов LISP, [illegible] (близок к старым LISP'ам с точностью до синтаксиса) и Perl.

История вопроса

John McCarthy

Richard Matthew Stallman

Первые диалекты LISP поддерживали *динамическое лексическое замыкание* — поиск в динамической цепочке по имени. Самый старый диалект LISP, использующийся до сих пор — Emacs LISP — по умолчанию работает так же, хотя в новых версиях это можно переключить.

Помимо старых диалектов LISP, поиск по имени используется в Logo (близок к старым LISP'ам с точностью до синтаксиса) и Perl.

История вопроса

John McCarthy

Richard Matthew Stallman

Первые диалекты LISP поддерживали *динамическое лексическое замыкание* — поиск в динамической цепочке по имени. Самый старый диалект LISP, использующийся до сих пор — Emacs LISP — по умолчанию работает ▸ так же, хотя в новых версиях это можно переключить.

Помимо старых диалектов LISP, ▸ поиск по имени используется в Logo (близок к старым LISP'ам с точностью до синтаксиса) и Perl.

Лексическое замыкание

```
(define (middle f)
  (let ((x 998))
    (f 1)
    )
  )
(define (outer)
  (let ((x 122))
    (let ((inner (lambda (y)
                   (display (+ x y))
                   )
                 ))
      (middle inner)
      )
    )
  )
(outer)
```

Это Scheme (Guile, R5RS). Выдаст 123. Как и ожидалось.

Динамическое замыкание

```
(defun middle(f)
  (let ((x 998)) ; comment this out
    (apply f '(1))
    )            ; comment this out
  )
(defun outer()
  (let ((x 122))
    (let ((inner (lambda (y)
                   (print (+ x y))
                   )
                 ))
      (middle inner)
      )
    )
  )
(outer)
```

Это Emacs LISP (версия 23). Выдаст 999!

Передача конца цепочки и мониторного массива

Вместе с адресом кода можно передать:

- ссылку на нижний объемлющий кадр;
- ссылки на все объемлющие кадры.

Таким образом, дополнительных осложнений не возникает.

Algol 68, Pascal (простые диалекты), блоки в Ruby.

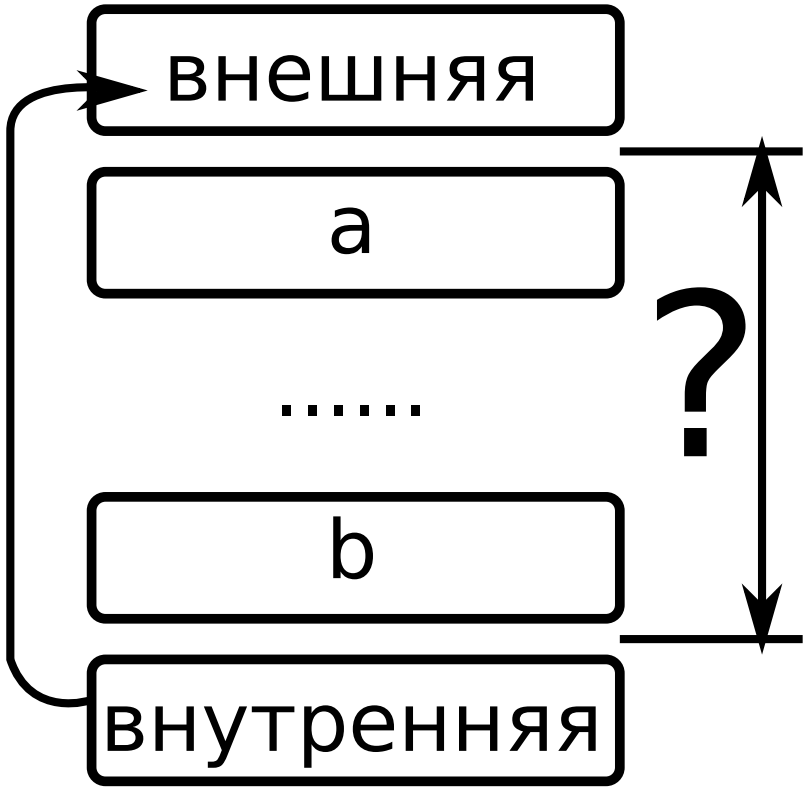
Передача конца цепочки и мониторного массива

Вместе с адресом кода можно передать:

- ссылку на нижний объемлющий кадр;
- ссылки на все объемлющие кадры.

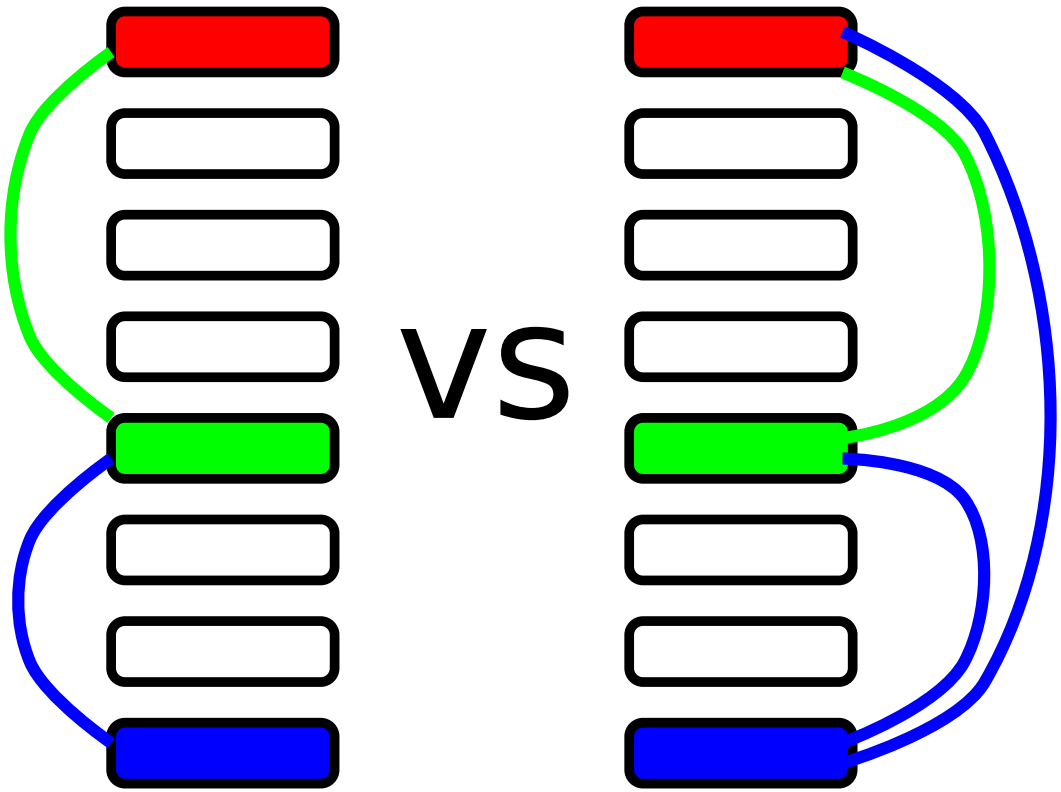
Таким образом, дополнительных осложнений не возникает.
Algol 68, Pascal (простые диалекты), блоки в Ruby.

Вытянутая статическая цепочка



Адрес объемлющего кадра не известен сам по себе (процедуру могли передать через много вызовов), поэтому его надо указать явно.

Мониторный массив



Можно разом передать адреса всех объемлющих кадров в специальной структуре данных — «мониторе», которую для работы внутренней создаёт и размещает внешняя процедура. Быстрее, но больше памяти.

Возврат и глобальное хранение процедур

Проблема более общая и решение тоже.

Когда вложенную процедуру возвращают наружу (наверх) или ссылку на неё сохраняют в глобальной переменной, возникает техническая проблема:

Статическая цепочка перестаёт быть подмножеством динамической.

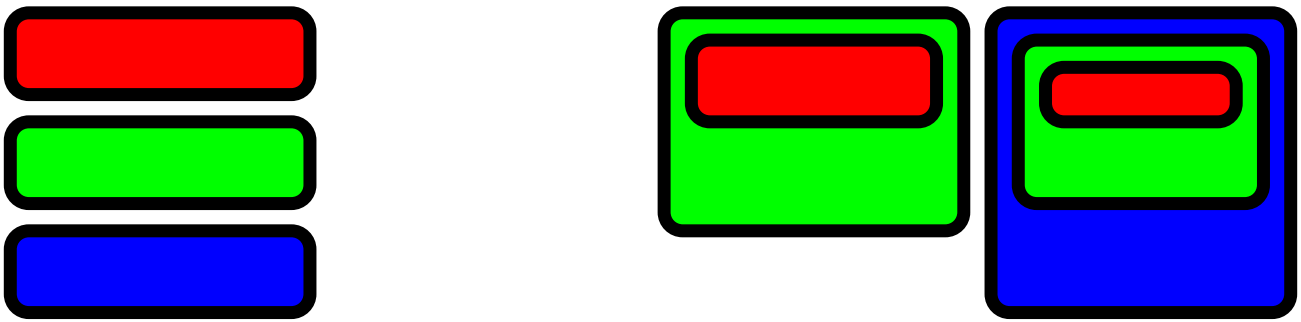
Возврат и глобальное хранение процедур

Проблема более общая и решение тоже.
Когда вложенную процедуру возвращают наружу (наверх) или ссылку на неё сохраняют в глобальной переменной, возникает техническая проблема:
Статическая цепочка перестаёт быть подмножеством динамической.

Возврат и глобальное хранение процедур

Проблема более общая и решение тоже.
Когда вложенную процедуру возвращают наружу (наверх) или ссылку на неё сохраняют в глобальной переменной, возникает техническая проблема:
Статическая цепочка перестаёт быть подмножеством динамической.

Константные кадры на стеке

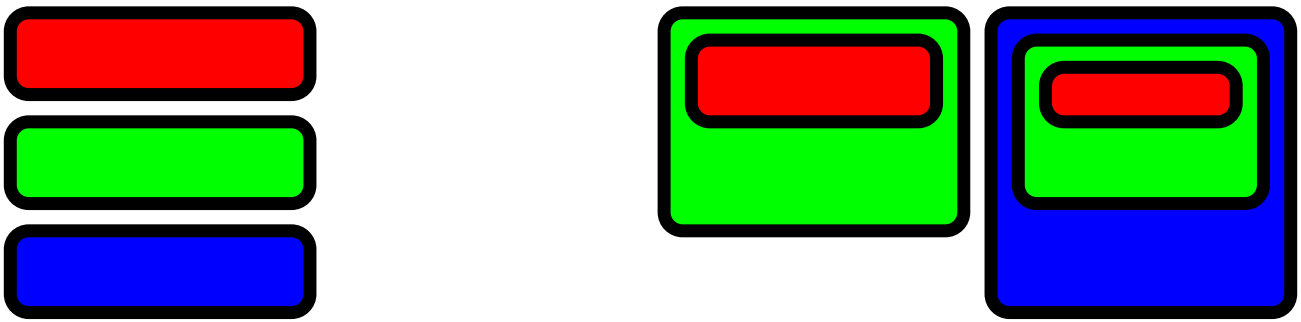


Если внешние данные изменять не нужно или нельзя (а особенно если их немного), то данные для работы внутренней процедуры можно просто скопировать и передать вместе со ссылкой на код.

Достаточно буквально соответствует β-редукции λ-исчисления.

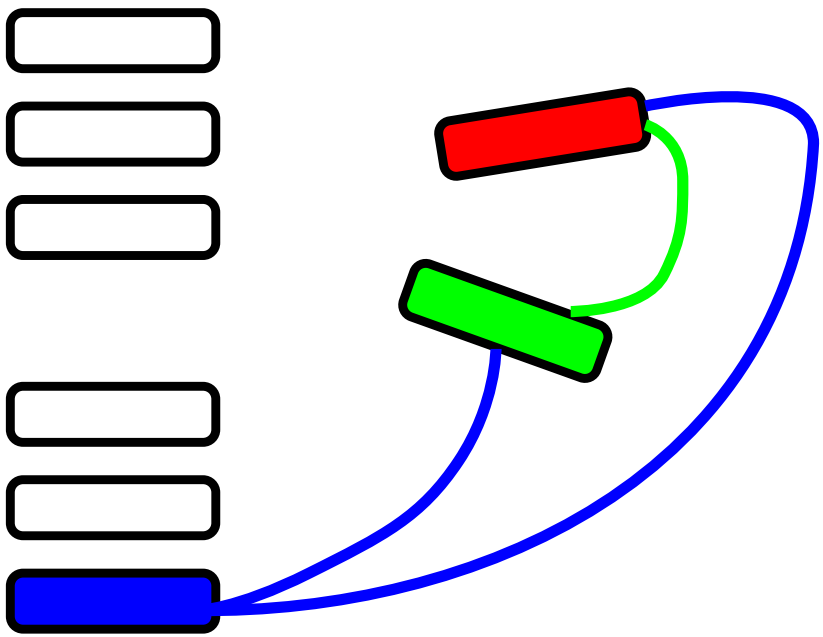
Используется в LISP, ML, Java.

Константные кадры на стеке



Если внешние данные изменять не нужно или нельзя (а особенно если их немного), то данные для работы внутренней процедуры можно просто скопировать и передать вместе со ссылкой на код.
Достаточно буквально соответствует β-редукции λ-исчисления.
Используется в LISP, ML, Java.

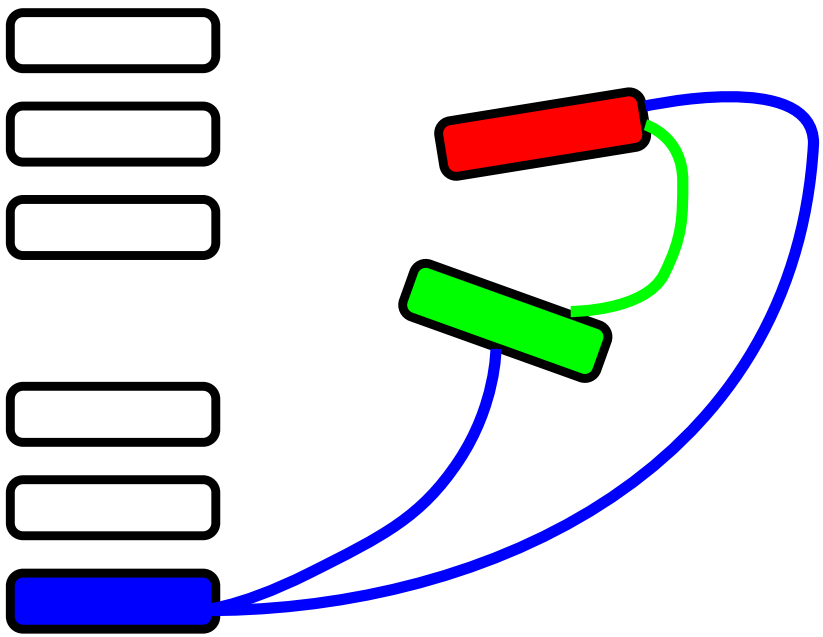
кадры в куче



Если данных вешней процедуры много (копировать не хочется) или если их требуется изменять, можно выделить кадр (или нужную часть кадра) внешней процедуры в куче.

Python, C#, Scala, λ-выражения в Ruby.

кадры в куче



Если данных вешней процедуры много (копировать не хочется) или если их требуется изменять, можно выделить кадр (или нужную часть кадра) внешней процедуры в куче.
Python, C#, Scala, λ-выражения в Ruby.

Оптимизация

- Распознавать частные случаи и применять максимально простые допустимые решения.
- Умудряться при этом не нарушать общности, чтобы разные процедуры можно было вызывать одинаково — вызывающая-то не знает.
  - Зато вызываемая знает, *что* ей должны передать. Так что можно просто передавать адрес процедуры и «n байтов данных», в которых разбираются она и внешняя, а вызывающие не обязаны.

Пример: «родной» (от M$) компилятор C# распознаёт вложенные процедуры, которые не возвращают наверх, и передаёт им конец цепочки со стека (оптимизация).

Оптимизация

- Распознавать частные случаи и применять максимально простые допустимые решения.
- Умудряться при этом не нарушать общности, чтобы разные процедуры можно было вызывать одинаково — вызывающая-то не знает.
  - Зато вызываемая знает, *что* ей должны передать. Так что можно просто передавать адрес процедуры и «n байтов данных», в которых разбираются она и внешняя, а вызывающие не обязаны.

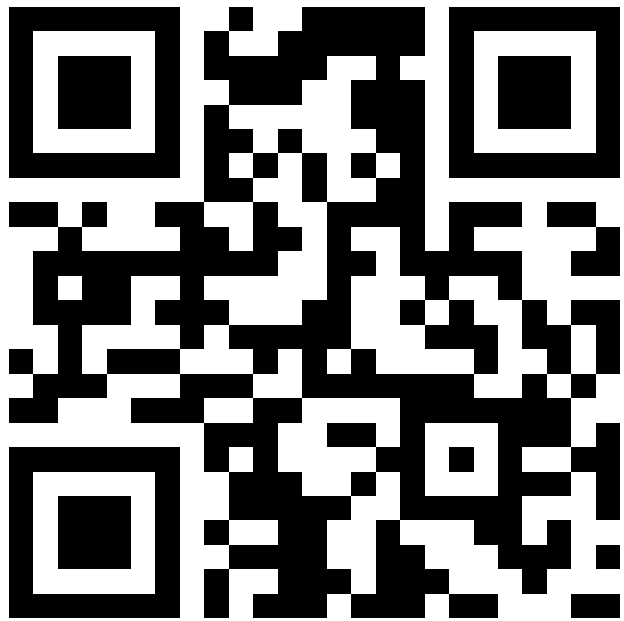
Пример: «родной» (от M$) компилятор C# распознаёт вложенные процедуры, которые не возвращают наверх, и передаёт им конец цепочки со стека (оптимизация).

Оптимизация

- Распознавать частные случаи и применять максимально простые допустимые решения.
- Умудряться при этом не нарушать общности, чтобы разные процедуры можно было вызывать одинаково — вызывающая-то не знает.
  - Зато вызываемая знает, *что* ей должны передать. Так что можно просто передавать адрес процедуры и «n байтов данных», в которых разбираются она и внешняя, а вызывающие не обязаны.

Пример: «родной» (от M$) компилятор C# распознаёт вложенные процедуры, которые не возвращают наверх, и передаёт им конец цепочки со стека (оптимизация).

Вопросы



EDU.DLUCIV.NAME