19. Функционално програмиране. Обща характеристика на функционалния стил на програмиране. Дефиниране и ползване на функции. Модели на оценяване. Функции от по-висок ред.

1. **Характерни особености на функционалния стил на програмиране**
   Във функционалния стил на програмиране, основният логически елемент са функциите, като те могат да бъдат композирани.

   Функция е програмна единица, която връща резултат. Те могат да бъдат от произволно висок ред - подавани като параметри, връщани, събирани, и т.н. Функциите:
   - Могат да имат параметри
   - Могат да се прилагат над аргументи
   - Аргументите им могат да бъдат други функции (функции от по-висок ред)
   - Могат да се дефинират чрез себе си (рекурсия)

   Във функционалното програмиране :
   - Основното действие е обръщение към функции
   - Основният начин за разделяне на програмата на части е въвеждането на име на функция и задаването за това име на израз, който пресмята стойността на функцията (дефиниране на функция)
   - Основното правило за композиция е суперпозицията на функции

   Функционалният стил е представител на декларативния стил на програмиране.
   - Отсъстват традиционните елементи от процедурния стил - цикли, заделяне на клетки в паметта, прескачане (break, return), присвояване
   - В декларативните езици (ДЕ) програмата явно посочва какви свойства има желаният резултат. В ДЕ, в частност ФП програмите се изграждат по схемата:
     Програма = списък от дефиниции на функции или **списък от равенства**

2. **Основни компоненти на функционалните програми**
   Основният компонент на функционалните програми са функциите, тяхното дефиниране, използване и оценяване.
   Дефинирането на функции представлява свързване на имена с изрази, които се оценяват. Рекурсията е основен метод за дефиниране.

3. **Примитивни изрази**
   Това са изрази, които не могат да се декомпозират повече, т.е. атомарни изрази на функционални езици: символи, числа, идентификатори - имена, които може да се реферират

   В Scheme атомарни изрази са:
   - Булеви константи: ($\#f$, $\#t$)
   - Числови константи: (15, 2/3, -1.23)
   - Знакови константи: (#\a, #\newline)
   - Низови константи: ("Scheme", "hi ")
   - Вградена функция: (+, -, *, /, sqrt)

4. **Средства за комбиниране и абстракция**

   Изразът е или *атом* (атомарна константа или променлива) или *обръщение към функция*, което има следният вид в Scheme:

   $$(\underbrace{\langle \text{означение на функция} \rangle}_{\text{оператор}} \ \underbrace{\langle \text{арг}_1 \rangle \ \langle \text{арг}_2 \rangle \ \ldots \ \langle \text{арг}_n \rangle}_{\text{операнди}})$$

В Scheme **комбинация** се нарича израз, конструиран като списък от изрази, заградени в скоби. Има следният синтаксис:

(⟨израз$_0$⟩ ⟨израз$_1$⟩ … ⟨израз$_n$⟩)

- ⟨израз$_0$⟩ се нарича *оператор,* очаква се да се оцени до функция, която да се приложи върху оценките на осаналите аргументи, наричани *операнди*.
- Стойността на комбинацията се получава чрез прилагане на процедурата, зададена чрез оператора, към аргументите, които са стойностите на операндите

Средство за абстракция е дефинирането на функция, защото ни позволява да модулизираме програмата.

В Scheme функция се дефинира, използвайки следният синтаксис: (***define*** ⟨име⟩ ⟨израз⟩).

Пример: $(define\ size\ 2)$

За дефиниране на съставна функция в Scheme се използва синтаксиса:

(**define** (⟨име⟩ ⟨формални параметри⟩) ⟨тяло⟩)

Пример: $\left(define\ (square\ x)\ (*\ x\ x)\right)$

Условна форма if. Синтаксис в Scheme: (if ⟨условие⟩ ⟨израз$_1$⟩ ⟨израз$_2$⟩)

Оценява се ⟨условие⟩:

- Ако оценката е $\#t$, връща се оценката на ⟨израз$_1$⟩
- Ако оценката е $\#f$, връща се оценката на ⟨израз$_2$⟩

Форма за многозначен избор cond:

($cond$ (⟨условие$_1$⟩ ⟨израз$_1$⟩)
…
(⟨условие$_n$⟩ ⟨израз$_n$⟩)
($else$ ⟨израз$_{n+1}$⟩)

Оценка:

- Оценява се ⟨условие$_1$⟩, при $\#t$, връща се оценката на ⟨израз$_1$⟩, а при $\#f$:
- …
- Оценява се ⟨условие$_n$⟩, при $\#t$, връща се оценката на ⟨израз$_n$⟩, а при $\#f$:
- Връща се ⟨израз$_{n+1}$⟩

Специална форма let:

(let ((⟨символ$_1$⟩ ⟨израз$_1$⟩)
…
(⟨символ$_n$⟩ ⟨израз$_n$⟩))
⟨тяло⟩)

Оценка на let в среда E:

- Създава се нова среда E_1, разширение на текущата среда E.
- Оценката на ⟨израз$_i$⟩ в E се свързва със ⟨символ$_1$⟩ в E$_1$, за $1 \le i \le n$
- Връща се оценката на ⟨тяло⟩ в средата E_1

Друго средство за абстракция е, използвайки *where*, което дава възможност за създаване на локални дефиниции в haskel. Синтаксис:

⟨дефиниция на функция⟩
where ⟨дефиниция$_1$⟩
…
⟨дефиниция$_n$⟩

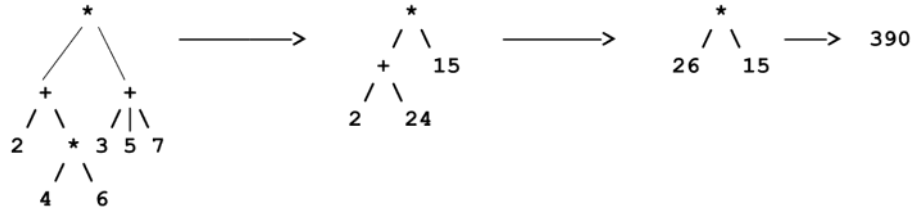
При тази конструкция, областта на действие на дефинициите е само в рамките на дефиницията на функцията. Може да са взаимно рекурсивни.

5. Оценяване на изрази

Общо правило за оценяване на комбинациите:

- Оценяват се подизразите на комбинацията
- Прилага се функцията, която е оценка на най-левия подизраз (операторът) към операндите, които са оценки на останалите подизрази

Пример: оценяването на $\left(*\ \ (+\ 2\ (*\ 4\ 6))\ (+\ 3\ 5\ 7)\right)$ изисква прилагането на общото правило върху четири различни комбинации. Оценчването на комбинацията може да бъде представено по следния начин:



Правила за оценяване на числа, низове, имена и вградени оператори:

- Оценката на число е самото число
- Оценката на низ е самият низ
- Оценката на вграден оператор е поредица от машинни инструкции, които реализират този оператор
- Оценките на останалите имена са стойностите, свързани с тези имена в текущата среда

Оператори , които са изключения от общото правило за оценяване на комбинации се наричат специални форми. Примери за такива оператори са *define, if, cond*.

Ако езикът е чист (без странични ефекти), то оценката винаги е една и съща, ако се изпълнява в един и същи контекст.

6. Дефиниране на функция и оценяване на приложение на функция

По-горе посочихме как се дефинира функция в Scheme.

При оценяване на комбинация, чиито оператор е функция, се оценяват елементите на комбинацията и се прилага процедурата, която е стойност на оператора на комбинацията, към аргументите, които са стойностите на операндите на комбинацията.

Пример: $(define\ \ (square\ x)\ (*\ x\ x))$

При *оценяване* на **define**, символът *square* се свързва със зададената дефиниция (поредица от инструкции, които пресмятат тялото при подадени стойности за параметъра x) в текущата среда.

Модел на заместването при оценяване на обръщения към функции:

- При оценяване на комбинация най-напред се оценяват подизразите на тази комбинация, след което оценката на първия подизраз се прилага върху оценките на останалите подизрази
- Прилагането на функция към получените аргументи става като формалните параметри се заместят със съответните аргументи. *Оценката на последния израз от тялото става оценка на обръщението към функцията.*

7. Модели на оценяване. *Апликативно (стриктно, call-by-value) и нормално (лениво, call-by-name) оценяване*

При посочения модел на оценяване чрез заместване са възможни два подхода за оценяване: апликативен и нормален.

Апликативен подход на оценяване - отначало се оценяват операндите и след това към получените оценки се прилага резултатът от оценяването на оператора, т.е. иска да се оцени всичко, което е дадено като аргумент преди да почне да прилага функцията.

Нормално оценяване - замества се името на процедурата с тялото й, докато се получат означения на примитивни процедури и след това се прилагат техните правила за оценка, т.е. първо се оценява функцията и после аргументите

Има случаи, в които едни и същи програми могат да дадат различни резултати при двата типа оценяване.

В стандартните езици за програмиране също има лениво оценяване в някаква форма - примерно при конюнкция, ако едно е лъжа, останалите не се пресмятат.

При нормалния подход е необходимо да се вземат мерки за избягване на многократното оценяване на един и същ израз. То влачи всички изрази, пестеливо е откъм време за сметка на памет.

Апликативното е пестеливо откъм памет, преди да се извика функцията всичко е смачкано до една стойност.

8. Функции от по-висок ред. Функциите като параметри и оценки на обръщения към функции

Деф: Функция, която приема функция за параметър или връща функция като резултат се нарича *функция от по-висок ред*.

Функции като параметри, пример на Scheme:

$(define\ (fixed\_point?\ f\ x)\ (=\ (f\ x)\ x))$

Къринг в haskel: можем да разглеждаме функция с n+1 аргумента, като функция с един аргумент, която връща функция с n аргумента. Това представяне се нарича къринг. Пример:

$div50 :: Int\ -> Int$
div50 x = div 50 x
div50 4 -> 12

9. Анонимни (ламбда) функции

Можем да конструираме параметрите на функциите от по-висок ред "на място", без да им задаваме имена.

Синтаксис на анонимна функция в Scheme:

$(lambda\ (\{\langle параметър \rangle\})\ \langle тяло \rangle)$

Оценява се до функционален обект със съответните параметри и тяло;

Пример за анонимна функция в Scheme:

$(lambda\ (x)\ (+\ x\ 3))\ \rightarrow\ \#\langle procedure \rangle$
Приложена: $((lambda\ (x)\ (+\ x\ 3))\ 5)\ \rightarrow\ 8$

Функции, които връщат функции. Пример в Scheme:

$(define\ (twice\ f)\ (lambda\ (x)\ (f\ (f\ x))))$
Приложена: $(twice\ square) \rightarrow \#\langle procedure \rangle$
$((twice\ square)\ 3)\ \rightarrow\ 81$

В този пример се връща като резултат анонимната функция.

Друг пример:

$(define\ (n+\ n)\ (lambda\ (i)\ (+\ i\ n)))$
Използване: $(define\ 1+\ (n+\ 1))$
$(1+\ 7)\ \rightarrow 8$