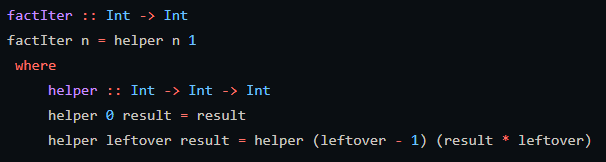
**Типове езици за програмиране от високо ниво**  
**Процедурни (императивни)  
програма = алгоритъм + структури от данни  
Декларативни (дескриптивни)  
програма = списък от дефиниции на функции   
или списък от равенства   
или списък от факти и правила  
  
Функцията е програмна част, която връща стойност (резултат).  
  
Булевите константи True и False, също както и числата, са известни под името литерали (изрази, които не се нуждаят от истинско оценяване, защото стойността на всеки от тях съвпада с името му). Литералите True и False (както и всички други литерали) могат да бъдат използвани като аргументи при дефинирането на функции.  
  
При рекурсивни процеси се поражда верига от обръщения към дефинираната функция с все по-прости в определен конкретен смисъл стойности на аргументите, докато се стигне до обръщение с т. нар. базов (прост, граничен) вариант на стойности на аргументите, след което започва последователно пресмятане на генерираните вече обръщения.   
Text

Description automatically generated with medium confidence  
При итеративни процеси във всеки момент състоянието на пресмятанията се описва (като при това може при необходимост да бъде прекъснато и после - възстановено) от няколко променливи (state variables, променливи на състоянието) и правило, с чиято помощ се извършва преходът от дадено състояние към следващото.   
  
  
Векторът (tuple) представлява наредена n-торка от елементи, при това броят n на тези елементи и техните типове трябва да бъдат определени предварително.  
A picture containing text, gauge, watch, device

Description automatically generated  
  
Списъкът в Haskell е редица от (променлив брой) елементи от определен тип. За всеки тип t в езика е дефиниран също и типът [t], който включва списъците с елементи от t.  
Text

Description automatically generated  
  
Операторите в Haskell са инфиксни функции, т.е. такива (двуаргументни) функции, означенията на които се записват между аргументите им, а не преди тях.  
  
Образецът от тип конструктор на списъци може да бъде или [ ], или да има вида (p:ps), където p и ps също са образци.  
  
Функция от по-висок ред се нарича всяка функция, която получава поне една функция като параметър (аргумент) или връща функция като резултат.  
Text

Description automatically generatedText

Description automatically generated  
Дефинирането на някаква функция на функционално ниво предполага действието на тази функция да се опише не в термините на резултата, който връща тя при прилагане към подходящо множество от аргументи, а като директно се посочи връзката ѝ с други функции.  
Graphical user interface

Description automatically generated  
Text

Description automatically generated**

**Прости са тези структури от данни, за които операцията достъп се осъществява до структурата като цяло. Такива структури са числата, символните (знаковите) и булевите данни. Обикновено на всяка от тези структури в езиците за програмиране съответстват вградени типове данни.  
Съставни са тези структури от данни, за които операцията достъп се осъществява до компонентите (елементите) на структурата, а не до структурата като цяло. Такива структури са масивът, записът, списъкът (т. нар. свързан списък), опашката, стекът и др.  
Съставна структура от данни, която се състои от фиксиран брой елементи и за която не са допустими операциите включване и изключване на елемент, се нарича статична. За статичните структури от данни в паметта на компютъра се отделя фиксирано количество памет. Такива структури са масивът и записът.  
Съставна структура от данни, която се състои от променлив брой елементи и за която са допустими операциите включване и изключване на елемент, се нарича динамична. Такива структури са стекът, опашката, дървото, графът и др.  
  
Масивът е крайна редица от фиксиран брой елементи от един и същ тип. Към всеки елемент от редицата е възможен пряк достъп, който се осъществява чрез индекс. Операциите включване и изключване на елемент от масив са недопустими, т.е. масивът е статична структура.  
  
Записът е крайна редица от фиксиран брой елементи, които могат да са от различни типове. Възможен е пряк достъп до всеки елемент от редицата, който се осъществява чрез име. Елементите на редица, представляваща запис, се наричат полета на записа. Операциите включване и изключване на елемент са недопустими, т.е. структурата е статична.  
Text

Description automatically generated with medium confidence  
  
Множеството е съвкупност от елементи от един и същ тип. Операциите включване и изключване на елемент са допустими. Възможен е достъп както до отделните компоненти на множеството, така и до структурата като цяло. Достъпът е пряк.   
Физическо представяне Използва се последователно представяне на структурата в паметта. За целта за всеки допустим елемент се посочва дали принадлежи или не принадлежи на множеството.  
  
Свързаният списък е крайна редица от елементи от един и същ тип. Операциите включване и изключване на елемент са допустими в произволно място на редицата. Възможен е достъп до всеки елемент на списъка, като достъпът до първия елемент е пряк, а до останалите елементи – последователен.  
  
Стекът е крайна редица от елементи от един и същ тип. Операциите включване и изключване на елемент са допустими само за единия край на редицата, който се нарича връх на стека. Възможен е достъп само до елемента, намиращ се на върха на стека, при това достъпът е пряк.** **При тази организация на логическите операции, последният включен в стека елемент се изключва първи. Затова стекът се определя още като структура „последен влязъл – пръв излязъл“ (last in – first out, LIFO).  
  
Опашката е крайна редица от елементи от един и същ тип. Операцията включване на елемент е допустима само за единия (например десния) край на редицата, който се нарича край на опашката. Операцията изключване на елемент е допустима само за другия (левия) край на редицата, който се нарича начало на опашката. Възможен е пряк достъп само до елемента, намиращ се в началото на опашката.** **При описаната организация на логическите операции, последният включен в опашката елемент се изключва последен, а първият – първи. Затова опашката се определя още като структура от данни „пръв влязъл – пръв излязъл“ (first in – first out, FIFO).  
  
Нека Te е даден тип данни. Дърво от тип Te е структура, която е образувана от:**

**• стойност от тип Te, наречена корен на дървото от тип Te;   
• крайно, възможно празно множество с променлив брой елементи – дървета от тип Te, наречени поддървета на дървото от тип Te.   
Състои се от върхове, свързани с ребра. Всеки връх може да има 0 или повече наследници. Всеки връх има най – много един баща (коренът няма). Всички върхове са достижими от корена.   
Листо – връх който няма наследници  
Предшественици – бащата на един връх и неговите предшественици  
Пряк наследник (син) на един връх е всеки връх, за който първоначалния е родител. Листата нямат синове.   
Наследниците на един връх са неговите синове и наследниците на неговите синове.   
Път – поредица от ребра между върховете   
Дължина на път – броя на ребрата свързващ върховете  
Височина – най-дългия път от лист до корен   
  
Алгебричните типове са съставни типове данни, дефинирани от програмиста. Дефиницията на един алгебричен тип започва с ключовата дума data, след която се записват името на типа, знак за равенство и конструкторите на типа. Името на типа и имената на конструкторите задължително започват с главни букви. Пример   
data Day = Monday | Tuesday | Wednesday | Thursday | Friday | Saturday | Sunday  
Вместо използването на вектори можем да дефинираме тип с определен брой компоненти като алгебричен тип. Такива типове често се наричат производни типове (резултатни типове; product types).   
Пример   
data People = Person Name Age   
type Name = String   
type Age = Int  
Горната дефиниция на People може да бъде интерпретирана както следва: За да се конструира елемент на типа People, е необходимо да се предвидят (дадат като аргументи) две стойности: едната (нека я наречем st) от тип Name, а другата (нека я наречем n) – от тип Age. Елементът на People, конструиран по този начин, ще има вида Person st n.   
Примери за стойности от тип People:   
Person “Aunt Jemima” 77   
Person “Ronnie” 14  
  
data Shape = Circle Float | Rectangle Float Float   
Дефиниция от вида на посочената означава, че съществуват два алтернативни начина за конструиране на елемент на Shape.  
Circle 3.0   
Rectangle 45.9 87.6  
  
Рекурсивни алгебрични типове  
алгебричните типове, които потребителят дефинира, да се описват в термините на самите себе си  
data NTree = NilT | Node Int NTree NTree   
Тази дефиниция е подходяща за моделирането на двоични дървета от цели числа (двоични дървета от тип Int).  
Дефинициите на алгебрични типове могат да съдържат променливи на типове (типови променливи, type variables) a, b и т.н. По този начин се дефинират полиморфни типове.  
Дърветата, които дефинирахме по-горе, бяха дървета от цели числа (дървета от тип Int). Ако искаме да дефинираме двоично дърво от произволен тип a, това може да стане с помощта на конструкция от вида data Tree a = Nil | Node a (Tree a) (Tree a)  
Генеричните функции, например == (проверка за равенство), + (събиране на числа от един и същ тип) и show (конвертиране на число, булева стойност и др. в низ), могат да бъдат прилагани към данни от много типове, но за различните типове в действителност се използват различни дефиниции (различни методи на генеричната функция).  
Всеки тип, който включва едно или повече ограничения върху класа, се нарича overloaded (пренатоварен; додефиниран). Функция, чийто тип е додефиниран, се нарича генерична (додефинирана). Следователно, Num a => a -> a -> a е додефиниран тип и (+) е генерична (додефинирана) функция.  
Най-общо, понятието клас в езика Haskell се определя като колекция от типове, за които се поддържа множество додефинирани операции, наречени методи.  
Например, за функцията elem, която е „вградена“ в Haskell, може да се предположи, че е от тип elem :: a -> [a] -> Bool Това обаче ще бъде вярно само за такива типове a, за които е дефинирана операцията (функцията) за проверка за равенство ==. Следователно, би било полезно да се разполага със средства, които позволяват да се зададат експлицитно определени ограничения върху даден тип, описан с помощта на типова променлива (type variable).** **Например, множеството от типове, за които е дефинирана функцията за проверка на равенство (==), се означава като клас Eq.  
Типовете, които принадлежат на даден клас, се наричат екземпляри на този клас. Най-важно (определящо) за класа Eq е наличието на функцията == от тип a -> a -> Bool, която проверява дали два елемента на даден тип a, който е екземпляр на Eq, са равни:   
class Eq a where   
 (==) :: a -> a -> Bool  
allEqual :: Eq a => a -> a -> a -> Bool allEqual m n p = (m==n) && (n==p)  
Един тип се определя като екземпляр на даден клас, като се дефинират функциите от сигнатурата на класа за елементите на този тип. Например дефиницията   
instance Eq Bool   
 where   
True == True = True   
False == False = True   
\_ == \_ = False   
определя вградения тип Bool като екземпляр на класа Eq.  
  
Езикът Haskell позволява да бъдат дефинирани класове, които са подкласове (производни класове, derived classes) на други класове. Най-прост пример в това отношение е класът на наредените типове, Ord. За да бъде нареден, един тип трябва да поддържа операциите за сравнение ==, /=, >, >=, <=. Наличието на първите две операции означава, че наредените типове образуват подмножество на класа Eq, т.е. те образуват подклас (производен клас, derived class) на класа Eq.  
class Eq a => Ord a where   
(<=), (>), (>=) :: a -> a -> Bool   
max, min :: a -> a -> a   
compare :: a -> a -> Ordering   
От друга страна може да се каже, че класът Ord наследява операциите на Eq.  
Възможно е също множествени ограничения да бъдат включени в дефиницията на клас, например   
class (Ord a,Visible a) => OrdVis a  
В случаите, когато даден клас се дефинира на базата на два или повече класа, се казва, че е налице множествено наследяване (multiple inheritance).  
  
Eq – клас на типовете, за които са дефинирани операциите за проверка на равенство и неравенство  
(==) :: a -> a -> Bool   
(/=) :: a -> a -> Bool  
Разгледаните досега типове Bool, Char, String, Int, Integer и Float са екземпляри на класа Eq. Такива са също и типовете, обхващащи списъци и вектори, чиито елементи са от тип, който е екземпляр на Еq.  
Text

Description automatically generated  
Enum – клас на изброимите типове  
Bounded – клас на ограничените типове   
Този клас се специфицира посредством декларацията   
class Bounded a where   
 minBound, maxBound :: a   
Тук minBound и maxBound определят най-малката и най-голямата допустима стойност на елементите на съответния тип.  
Show – клас на „видимите“ типове (типовете, чиито елементи могат да се преобразуват в символни низове)  
Read – клас на типовете, чиито стойности могат да бъдат четени от низове  
  
При „мързеливото“ оценяване интерпретаторът оценява даден аргумент на дадена функция само ако (и доколкото) стойността на този аргумент е необходима за пресмятането на целия резултат. Нещо повече, ако даден аргумент е съставен (например е вектор или списък), то се оценяват само тези негови компоненти, чиито стойности са необходими от гледна точка на получаването на резултата. При това дублиращите се подизрази се оценяват по не повече от един път.**

**Text

Description automatically generated   
Text

Description automatically generated  
A picture containing text, device, meter, gauge

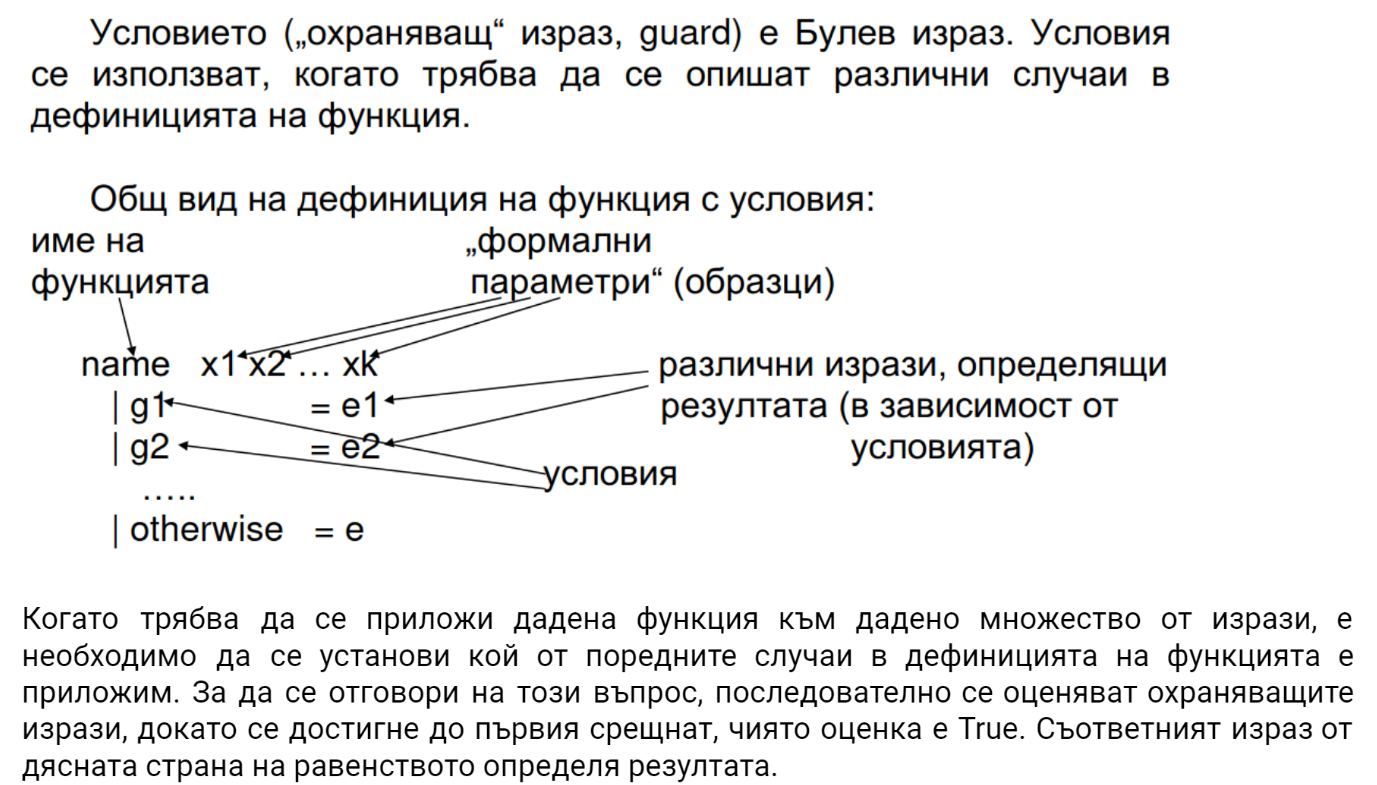
Description automatically generated  
Text

Description automatically generated  
Стойностите в клаузите where се пресмятат при необходимост („при поискване“): пресмятането на дадена стойност започва едва когато се окаже, че тази стойност е необходима.  
  
Работа с безкрайни списъци в Haskell  
Text

Description automatically generated  
  
Unary function – accepts 1 argument (isPrime)  
Binary function – accepts 2 arguments (div, mod, +, \*)  
  
Анонимна функция (или т.нар. функционален литерал или ламбда функция) в програмирането е дефинирана функция, която не е обвързана с идентификатор, тоест не е именувана.**

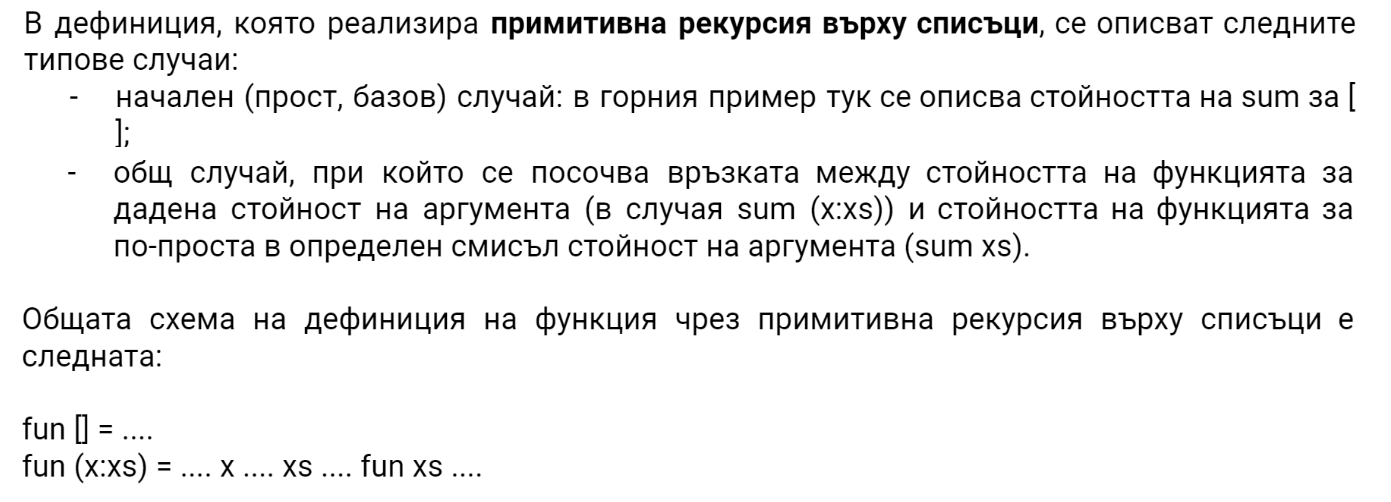
**Най-често анонимните функции могат да бъдат използвани:**

* **като аргументи, които се прехвърлят към функции от по-висок ред, или**
* **за построяване на резултата на функция от по-висок ред, която трябва да върне функция.**

**Text

Description automatically generatedОбразците се използват за pattern matching Text

Description automatically generatedText

Description automatically generated  
Graphical user interface, text, application, letter, email

Description automatically generated  
Принцип на структурната индукция при работата със списъци  
Базов случай: замести xs с []  
Индуктивна стъпка: замести xs с (x:xs) при условие, че е вярно когато е xs  
  
Екземпляр на клас == deriving (…)**