4. ПОДРЕЖДАНЕ С ПРОЛОГ

Въведение. Предимство на Пролог е предоставяната възможност отношенията в реалния свят да се описват по естествен начин, което прави програмния код леко читаем от различните програмисти, работещи над общ проект. Една програма е толкова по-добра, колкото по-добре са подредени данните, с които тя работи. Когато се описват предметни области, естествено е наредбата да се диктува от естествената наредба на нещата в описваната област.

Най-често подреждането на последователност (списък) от n елемента a_1, a_2,. . ., a_n е по големина и се нарича сортиране. Когато елементите на списъка представляват последователност например от символи, подредбата може да бъде и по подредбата на символите, например в азбуката (лексикографско сортиране). По този начин са подредени речниците, телефонните указатели, библиотечните каталози и др.

При лексикографското подреждане на думи с фиксирана дължина се извършват толкова обхождания на думите, колкото е дължината им, като при всяко обхождане се осъществява подреждане по символа от една и съща позиция на думите. При първото обхождане подреждане е по най-десния символ. Следващите обхождания извършват подреждане съответно по символите от втора, трета и т.н. позиции отдясно наляво. Всяко следващо подреждане не нарушава наредбата на думите, получена от предходните подреждания. При всяко обхождане се създават по няколко списъка, всеки от които сответства на символ. Списъците са подрени според реда на символите в азбуката. Думата се записва в съответния списък в зависимост от символа в подрежданата позиция. При всяко обхождане думите се четат от списъците (отляво надясно и отдолу нагоре), получени от предишното обхождане, и се записват в нови списъци. След последното обхождане думите се четат от списъците и се записват в общ списък.

Постановка. В следващото разглеждане ще покажем как някои от най-често използваните методи за подреждане могат да се запрограмират на Пролог и как машината за извод върши своята работа.

Пример 1. Да се сортира във възходящ ред по метода на непосредствената размяна (известен още като метод на мехурчето) списък от цели числа.

Упътване. Нека елементите на списъка са разположени вертикално, както е показано в следващата таблица. Приема се, че сортираната част на списъка заема неговата горна част и в началото тя не съдържа нито един елемент (лявата колона на таблицата). Несортираната част на списъка се обхожда отдолу нагоре. При всяко обхождане се сравняват двойките съседни елементи, като ако този, който е отгоре, е по-голям, двата елемента си разменят местата. Ето един малък пример:

Начало	1-во обхождане	2-ро обхождане
5	1	1
6	5	3
3	6	5
1	3	6

При всяко обхождане минималният елемент от несортираната част ще изплува на нейния връх, тъй като всяко сравнение, в което той участва, ще води до размяна. При размяната позицията му се придвижва в посока на следващото сравнение и следваща размяна. Това продължава, докато той не отиде най-горе на несортираната част, след което той се включва в сортираната част. При следващото преминаване несортираната част е с намален размер. Това продължава до достигане на размер 1 на несортираната част.

Да наречем подреждащия предикат sort(X,Y), където X е несортираният списък, а Y – резултатът, т.е. сортираният списък. Условието за край на сортирането е двата списъка да съвпадат (sort(L,L).).

Предикатът за размяна е наречен swap. Той е с два аргумента. В първия е списъкът преди размяната, а във втория – списъкът след размяната, т.е малко по-подреден списък.

В програмата този предикат се използва два пъти. Първото включване е да извърши размяната (това става за първите два елемента на списъка), а второто - да премине към сравняване в следващата позиция на списъка (текущо обработваният списък се съкращава с по-големия от двата участвали в сравнението елемента).

В правилото за sort са включени три подцели. Първата извиква предикат swap. Предикатът cut (!) спира по-нататъшното търсене на решения от предикат swap и предотвратява търсенето на други размени. Третата подцел рекурсивно извиква sort за получения междинен (полуподреден) списък.

Програма.

```
sort(X,Y):-swap(X,L),!,sort(L,Y).
sort(L,L).
swap([A,B|T],[B,A|T]):-A>B.
swap([H|T1],[H|T2]):- swap(T1,T2).
```

Goal

```
sort([5,6,3,1],Y)
```

```
Y= [1,3,5,6]
1 Solution
```

Работа на Пролог. При разглеждане изпълнението на програмата се вижда, че когато предикатът swap вече не може да се прилага, защото са се изчерпали

всички елементи на списъка и не е направена никаква размяна, първата клауза на sort ще завърши с false и се извиква втората клауза от програмата, която унифицира окончателния списък с междинния, т.е. с подредения списък.

```
Ready
Reconsulted from: C:\My Documents\sort_mehurche.pro

sort([5,6,3,1],Y)
Trace: >> CALL:   sort([5,6,3,1],_)
Trace: >> CALL:   swap([5,6,3,1],_)
Trace: >> CALL:   5 > 6
Trace: >> FAIL:   5 > 6
Trace: >> CALL:   swap([6,3,1],_)
Trace: >> CALL:   6 > 3
Trace: >> RETURN: 6 > 3
Trace: >> RETURN: swap([6,3,1],[3,6,1])
Trace: >> RETURN: swap([5,6,3,1],[5,3,6,1])
Trace: >> CALL:   sort([5,3,6,1],_)
Trace: >> CALL:   swap([5,3,6,1],_)
Trace: >> CALL:   5 > 3
Trace: >> RETURN: 5 > 3
Trace: >> RETURN: swap([5,3,6,1],[3,5,6,1])
Trace: >> CALL:   sort([3,5,6,1],_)
Trace: >> CALL:   swap([3,5,6,1],_)
Trace: >> CALL:   3 > 5
Trace: >> FAIL:   3 > 5
Trace: >> CALL:   swap([5,6,1],_)
Trace: >> CALL:   5 > 6
Trace: >> FAIL:   5 > 6
Trace: >> CALL:   swap([6,1],_)
Trace: >> CALL:   6 > 1
Trace: >> RETURN: 6 > 1
Trace: >> RETURN: swap([6,1],[1,6])
Trace: >> RETURN: swap([5,6,1],[5,1,6])
Trace: >> RETURN: swap([3,5,6,1],[3,5,1,6])
Trace: >> CALL:   sort([3,5,1,6],_)
Trace: >> CALL:   swap([3,5,1,6],_)
Trace: >> CALL:   3 > 5
Trace: >> FAIL:   3 > 5
Trace: >> CALL:   swap([5,1,6],_)
Trace: >> CALL:   5 > 1
Trace: >> RETURN: 5 > 1
Trace: >> RETURN: swap([5,1,6],[1,5,6])
Trace: >> RETURN: swap([3,5,1,6],[3,1,5,6])
Trace: >> CALL:   sort([3,1,5,6],_)
```

```
Trace: >> CALL:   swap([3,1,5,6],_)
Trace: >> CALL:   3 > 1
Trace: >> RETURN: 3 > 1
Trace: >> RETURN: swap([3,1,5,6],[1,3,5,6])
Trace: >> CALL:   sort([1,3,5,6],_)
Trace: >> CALL:   swap([1,3,5,6],_)
Trace: >> CALL:   1 > 3
Trace: >> FAIL:   1 > 3
Trace: >> CALL:   swap([3,5,6],_)
Trace: >> CALL:   3 > 5
Trace: >> FAIL:   3 > 5
Trace: >> CALL:   swap([5,6],_)
Trace: >> CALL:   5 > 6
Trace: >> FAIL:   5 > 6
Trace: >> CALL:   swap([6],_)
Trace: >> CALL:   swap([],_)
Trace: >> FAIL:   swap([],_)
Trace: >> FAIL:   swap([6],_)
Trace: >> FAIL:   swap([5,6],_)
Trace: >> FAIL:   swap([3,5,6],_)
Trace: >> FAIL:   swap([1,3,5,6],_)
Trace: >> RETURN: sort([1,3,5,6],[1,3,5,6])
Trace: >> RETURN: sort([3,1,5,6],[1,3,5,6])
Trace: >> RETURN: sort([3,5,1,6],[1,3,5,6])
Trace: >> RETURN: sort([3,5,6,1],[1,3,5,6])
Trace: >> RETURN: sort([5,3,6,1],[1,3,5,6])
Trace: >> RETURN: sort([5,6,3,1],[1,3,5,6])
Y= [1,3,5,6]
Trace: >> REDO:   sort([5,6,3,1],[1,3,5,6])
Trace: >> REDO:   sort([5,3,6,1],[1,3,5,6])
Trace: >> REDO:   sort([3,5,6,1],[1,3,5,6])
Trace: >> REDO:   sort([3,5,1,6],[1,3,5,6])
Trace: >> REDO:   sort([3,1,5,6],[1,3,5,6])
Trace: >> REDO:   sort([1,3,5,6],[1,3,5,6])
Trace: >> FAIL:   sort([1,3,5,6],_)
Trace: >> FAIL:   sort([3,1,5,6],_)
Trace: >> FAIL:   sort([3,5,1,6],_)
Trace: >> FAIL:   sort([3,5,6,1],_)
Trace: >> FAIL:   sort([5,3,6,1],_)
Trace: >> FAIL:   sort([5,6,3,1],_)
  1 Solution
```

Пример 2.Да се състави програма на Пролог за сортиране чрез вмъкване на списък с цели числа.

Упътване. Списъкът се разделя на сортирана (лява) част и несортирана (дясна) част. На всяка стъпка първият елемент от несортираната част се вмъква на правилното място в сортираната част така, че тя да остане сортирана след вмъкването. Това изисква част от елементите на сортираната част да се изместят така, че да освободят място за вмъквания елемент.

Пример. 3,5,9,1,6

Номер на вмъкването	Сортирана част	Несортирана част
Начало		3,5,9,1,6
1	3	5,9,1,6
2	3,5	9,1,6
3	3,5,9	1,6
4	1,3,5,9	6
5	1,3,5,6,9	

Програмата използва два предиката - sort и insert.

Предикатът sort има два аргумента. В първия се помества текущата част от зададения за сортиране списък, а във втория – резултатът от сортирането на тази част от списъка. Вмъкването на поредния елемент започва от края на подредената част, т.е. от празни списъци (sort([],[]).).

Предикатът insert има три аргумента. Първият е елементът, който трябва да се вмъкне, вторият е разглежданата част от подлежащия на сортиране списък, в която ще се вмъква елементът, а третият е същата тази част в сортиран вид. Рекурсията по insert търси мястото за вмъкване на елемента и когато го намери, извършва самото вмъкване.

Програма.

```
sort([],[]).
sort([H|T],L):- sort(T,L1), insert(H,L1,L).
insert(E, [H|T1],[H|T]):- E>H,!,insert(E,T1,T).
insert(E,T,[E|T]).
```

Goal

```
sort([3,5,9,1,6],Y)
```

Работа на Пролог. Втората клауза на предиката sort отделя главата и извиква рекурсивно предиката sort с опашката на зададения списък, като използва нова променлива за междинния подсписък. Това рекурсивно извикване ще продължава, докато се стигне до празен списък и се изпълни първата клауза sort. Тогава H = 6, T=[], L1=[], L=_ . Следва извикване на предиката insert. Правилото за insert не се изпълнява, защото списъците са празни и не може да се вземе глава. При първото повикване втората клауза на

insert вмъква елемента 6 в сортираната част, която досега нямаше елементи. Така започва формирането на сортирана част от един елемент и правилото за insert може да се изпълни.

Поредният елемент от стека е 1. Сега 1 трябва да се вмъкне в [6]. За целта е необходимо да се намери мястото на елемента “1”. Взема се главата на [6] – това е новосформираният до момента сортиран списък. Тъй като 1 < 6, първата клауза в правилото завършва с false и втората клауза вмъква “1” преди “6” и резултатът е сортиран подсписък [1,6]. Обратният ход на рекурсията продължава с елемента “9”, който трябва да се вмъкне в списъка [1,6]. Докато “9” е по-голям от елементите в сортирания подсписък (1 и 6), от подсписъка се отстраняват (изпращат се в стека) елементи. Тъй като подсписъкът остава празен, правилото за insert завършва с false и второто insert поставя “9” в празния списък. Следва възстановяване на “6” и “1” и се достига до подсписък [1,6,9].

И така, втората клауза на предиката insert реализира самото вмъкване на елемента E за глава на формирания сортиран подсписък. Правилото за предиката insert завършва с false, когато вмъкваният елемент е по-малък от главата на сортирания подсписък или когато в сортирания подсписък няма повече елементи. Тогава се извиква втората клауза на insert. Докато елементът, който трябва да се вмъква е по-голям от главата на сортирания подсписък, се вика рекурсивно първата клауза на insert с опашката на сортирания списък. Ето и трасировката за пример sort([3,5,9,1,6],Y).

```
Trace: >> CALL:   sort([3,5,9,1,6],_)
Trace: >> CALL:   sort([5,9,1,6],_)
Trace: >> CALL:   sort([9,1,6],_)
Trace: >> CALL:   sort([1,6],_)
Trace: >> CALL:   sort([6],_)
Trace: >> CALL:   sort([],_)
Trace: >> RETURN: sort([],[])
Trace: >> CALL:   insert(6,[],_)
Trace: >> RETURN: insert(6,[],[6])
Trace: >> RETURN: sort([6],[6])
Trace: >> CALL:   insert(1,[6],_)
Trace: >> CALL:   1 > 6
Trace: >> FAIL:   1 > 6
Trace: >> RETURN: insert(1,[6],[1,6])
Trace: >> RETURN: sort([1,6],[1,6])
Trace: >> CALL:   insert(9,[1,6],_)
Trace: >> CALL:   9 > 1
Trace: >> RETURN: 9 > 1
Trace: >> CALL:   insert(9,[6],_)
Trace: >> CALL:   9 > 6
Trace: >> RETURN: 9 > 6
Trace: >> CALL:   insert(9,[],_)
```

```
Trace: >> RETURN: insert(9,[],[9])
Trace: >> RETURN: insert(9,[6],[6,9])
Trace: >> RETURN: insert(9,[1,6],[1,6,9])
Trace: >> RETURN: sort([9,1,6],[1,6,9])
Trace: >> CALL:   insert(5,[1,6,9],_)
Trace: >> CALL:   5 > 1
Trace: >> RETURN: 5 > 1
Trace: >> CALL:   insert(5,[6,9],_)
Trace: >> CALL:   5 > 6
Trace: >> FAIL:   5 > 6
Trace: >> RETURN: insert(5,[6,9],[5,6,9])
Trace: >> RETURN: insert(5,[1,6,9],[1,5,6,9])
Trace: >> RETURN: sort([5,9,1,6],[1,5,6,9])
Trace: >> CALL:   insert(3,[1,5,6,9],_)
Trace: >> CALL:   3 > 1
Trace: >> RETURN: 3 > 1
Trace: >> CALL:   insert(3,[5,6,9],_)
Trace: >> CALL:   3 > 5
Trace: >> FAIL:   3 > 5
Trace: >> RETURN: insert(3,[5,6,9],[3,5,6,9])
Trace: >> RETURN: insert(3,[1,5,6,9],[1,3,5,6,9])
Trace: >> RETURN: sort([3,5,9,1,6],[1,3,5,6,9])
Y= [1,3,5,6,9]
1 Solution
```

Пример 3. Да се програмира на Пролог алгоритъмът за бързо сортиране на списък от цели числа.

Упътване. При класическия вариант на алгоритъма списъкът се дели на 2 равни (при четен брой елементи) или почти равни части - лява и дясна. Лявата и дясна част се сортират по същия алгоритъм. Това продължава за всяка част, докато в частта остане един елемент. Следва процес на сливане на получените сортирани подсписъци, като размерите им нарастват от 1 на 2, 4, 8 и т.н. елементи в подсписък.

При всяко сливане елементите на два сортирани подсписъка се обединяват в трети списък, който също е сортиран. Това се извършва чрез последователно обхождане на двата сливани подсписъка от по-малките елементи към по-големите, като на всяка стъпка от двата най-малки разглеждани елемента се избира този с по-малка стойност и той се записва в третия списък.

Тъй като броенето не е силна страна на Пролог, при програмата ще се използва вариант на алгоритъма, при който списъкът се разделя на две части спрямо конкретен елемент от списъка, като в единия подсписък се поставят по-големите от избрания елементи, а в другия подсписък - елементите по-малки

или равни на него. Разделянето е удобно да стане спрямо главата на списъка. Процедурата е рекурсивна и се прилага към всеки нов подсписък, който има непразна опашка.

Предикатът за разделяне на списъка на две части спрямо един елемент split е четириместен. В първа позиция се поставя елементът, спрямо който е разделянето, във втора позиция - списъкът (подсписъкът), който трябва да се раздели, в трета позиция се формира подсписъкът от по-малките от зададения елемент елементи и накрая, в четвърта позиция се изпращат елементите равни или по-големи от зададения елемент.

Двата други предиката от програмата са ни вече известни - append и sort.

Програма.

```
sort([],[]).
sort([H|T],L):- split(H,T,F,S),sort(F,L1),
            sort(S,L2), append(L1,[H|L2],L).

split(_,[],[],[]).
split(H,[A|T],[A|F],S):- H>A,!,split(H,T,F,S).
split(H,[A|T],F,[A|S]):- split(H,T,F,S).

append([],L,L).
append([X|L1],L2,[X|L3]):- append(L1,L2,L3).
```

Работа на Пролог. Втората клауза на предиката sort изразява правилото, че сортирането става, като се разбие опашката T на два списъка F и S, те се сортират и получените сортирани списъци L1 и L2 се съединяват с главата H, спрямо която са разделени. Елементите на списъка F са по-малки или равни на главата H, а елементите на списъка S са по-големи от главата H. Първата клауза на предиката sort спира рекурсията.

Втората клауза на предиката split прехвърля главата на зададения списък (подсписък) в третия аргумент, ако тя е по-малка от зададения елемент. В противен случай се изпълнява третата клауза на предиката split, която прехвърля главата на зададения списък в четвъртия аргумент. Първата клауза на предиката split спира рекурсията.

Действията на двете клаузи на предиката append са вече добре изяснени в Пример 4 на упражнение 3.

Задание

1. Проследете изпълнението на програмата от Пример 1 по стъпки при сортирането на различни целочислени списъци.

2. Разменете местата на първите две клаузи и отново трасирайте. Какво се промени и кой от двата варианта е за предпочитане?

Забележка. С размяната се спазва правилото рекурсията да се започва с клаузата за край.

3. Проследете изпълнението на програмата от Пример 2 по стъпки при сортирането на различни целочислени списъци.

4. Експериментирайте с програмата от Пример 2 като отстраните например cut (!).

5. Променете програмата от Пример 2 така, че сортирането да е в обратен ред.

6. Пробвайте програмата за бързо сортиране (Пример 3) за няколко целочислени списъка. Проследете изпълнението ѝ, като включите проследяването по стъпки Trace Calls On.

7. Сравнете по брой операции трите програми за сортиране на целочислени списъци.
Забележка. Например като трасирате сортирането на един и същ списък.

8. Съставете програма за лексикографско подреждане на думи в речник.
Забележка. Тъй като думите са с различна дължина, за да ги приведете към случая "думи с еднаква дължина", добавете в азбуката интервала и го поставете на първо място.