

-Prolog入門(自習用)-

CNS上で利用可能なPrologの処理系は、SICStus Prolog 3.8です。zz.???やccz.???で利用することができます。

1 Prologの起動と終了

Prologを起動するためには、コマンドプロンプトから、

```
% sicstus
```

と入力してください。そうすると、Prologが起動し、以下のように表示されます。

```
SICStus 3.8.3 (SunOS-5.5.1-sparc): Tue Oct 06 13:38:15 MET DST 1998  
Licensed to sfc.keio.ac.jp  
| ?-
```

ここで、| ?- は、Prologのプロンプトです。

次にプログラムをPrologにロード(読み込ませる)させます。プログラムをロードするためには、[File-Name]のように、プログラムが書かれたファイル名を、カッコとシングルクォートで囲みます(ファイルには、.plの拡張子をつけてください)。ここで、最後の:(ピリオド)を忘れないように注意しよう。試しに、/home/tozaki/Pub/prologというディレクトリにあるfamily.plというファイルをロードしてみます(このプログラムの中身は、2節に載っています)。

```
| ?- ['/home/tozaki/Pub/prolog/family.pl'],  
{consulting /a/fs0023a/tozaki/Pub/prolog/family.pl...}  
{/a/fs0023a/tozaki/Pub/prolog/family.pl consulted, 20 msec 7560 bytes}
```

```
yes  
| ?-
```

これで、family.plというファイルが読み込まれました。

この様にPrologを起動したディレクトリと、ファイルのあるディレクトリが違う場合は、ディレクトリごと指定してください。また、Prologを起動したディレクトリと、ファイルのあるディレクトリが同じ場合は、ファイル名だけでもロードできます。

次に、ロードしたプログラムを実行します。Prologの実行とは「データベースへ質問を発行する」と考えれば分かり易いでしょう。例えば「tomは男であるか」という質問をしてみます。Prologのプロンプトから、以下のように入力します(最後の:を忘れないように注意しましょう)。

```
| ?- male(tom).  
yes  
| ?-
```

'yes'と返事が返ってきました。すなわち質問の結果は「tomは男である」ということとなります。次に、maryの子供を検索してみます。その場合、変数を用いて以下のように入力します。

```
| ?- parent(mary,X).  
X = tim ?
```

まずは、X = tim ? となり、timがmaryの子供であることが分かりました。さらに、Prologは以下のように入力:セミコロンを入力することで、別の可能性を調べてくれます。

```
| ?- parent(mary,X).  
X = tim ? ;  
X = sally ? ;  
no  
| ?-
```

timの他にsallyもmaryの子供であることが分かりました。この様に:を入力することで別の可能性を検索できますが、もうこれ以上解がない場合は、noが返ってきます。

Prologを終了させるためには、プロンプトから'halt.'と入力します。

```
| ?- halt.  
%
```

2 簡単な例題1 - 血縁関係の定義 -

親子関係のデータベース

% 行のはじめにこの様に%をつけると、その行はコメントになります

```
%File:Name: family.pl  
%parent(P,C).はPはCの親であるの意味
```

```
%male(X).はXは男性であるの意味
```

```
parent(mary,tim).  
parent(bob,tim).  
parent(jack,jimmy).  
parent(sam,peg).  
parent(tom,mary).  
parent(tom,jack).  
parent(joseph,elizabeth).  
parent(elizabeth,mary).  
parent(elizabeth,jack).  
parent(ellen,joan).  
parent(virginia,elizabeth).  
parent(martin,joseph).  
parent(mary,sally).  
parent(bob,sally).  
parent(betty,jimmy).  
parent(jane,peg).  
parent(tom,jane).  
parent(joseph,tom).  
parent(joseph,peter).  
parent(elizabeth,jane).  
parent(peter,joan).  
parent(virginia,tom).  
parent(virginia,peter).  
parent(cathy,joseph).
```

```
%female(X).はXは女性であるの意味
```

```
female(sally).  
female(betty).  
female(elizabeth).  
female(cathy).  
female(ellen).  
male(tom).  
male(sam).  
male(jack).  
male(jimmy).  
male(joseph).  
male(joan).
```

問題 2.1 このPrologプログラム(/home/tozaki/Pub/prolog/family.pl)は、事実のみから構成されています。

す。この基本的な関係を使って、以下の述語を定義してみよう (*family.pl* に以下の述語の定義を追加してください)。また、定義した述語を利用して、*Prolog* へ様々な質問をしてみましょう。

1. F は C の父親であるという述語 `father(F,C)` を定義しなさい
2. M は C の母親であるという述語 `mother(M,C)` を定義しなさい
3. X は Y の兄弟姉妹であるという述語 `sibling(X,Y)` を定義しなさい
4. GF は C の祖父であるという述語 `grandFather(GF,C)` を定義しなさい
5. GM は C の祖母であるという述語 `grandMother(GM,C)` を定義しなさい
6. GP は C の祖父であるという述語 `grandParent(GP,C)` を定義しなさい
7. C は GP の孫であるという述語 `grandChild(C,GP)` を定義しなさい
8. U は X の叔父であるという述語 `uncle(U,X)` を定義しなさい
9. A は X の叔母であるという述語 `unt(A,X)` を定義しなさい
10. N は X の甥であるという述語 `nephew(N,X)` を定義しなさい
11. N は X の姪であるという述語 `niece(N,X)` を定義しなさい
12. X は Y のいとこであるという述語 `cousin(X,Y)` を定義しなさい
13. A は X の祖先であるという述語 `ancestor(A,X)` を定義しなさい
14. X と Y は親類であるという述語 `relation(X,Y)` を定義しなさい。ただし、(1) 一方が他方の祖先、(2) 共通の祖先を持つ、(3) 共通の子孫を持つ、場合を親類とします。

3 数式処理の基礎

生年、没年のデータベース

```
%今年
this_year(1998).

%生年
birth_year(cathy,1882).
birth_year(alizabeth,1953).
birth_year(joan,1978).
birth_year(sam,1970).
birth_year(sally,1992).

birth_year(martin,1880).
birth_year(tom,1948).
birth_year(mary,1970).
birth_year(jack,1976).
birth_year(peg,1995).

birth_year(virginia,1920).
birth_year(peter,1949).
birth_year(boob,1963).
birth_year(betty,1972).
birth_year(jimmy,1994).

%没年
dead_year(cathy,1948).
dead_year(betty,1995).

dead_year(virginia,1992).
dead_year(peter,1979).
```

まず、*family.pl* に問題 2.1 の答えを追加したファイルに、このプログラムを追加してください。

X は Y より若いという述語、`younger(X,Y)` を定義してみましょう。若いということは、生まれた年が違っていることですから、プログラムは以下になります。

```
younger(X,Y):- birth_year(X,Xb), birth_year(Y,Yb), Xb > Yb.
```

ここでは、算術計算の組み込み述語、<を使っていきます。例えば、

```
| ?- younger(joan,X).
X = cathy ?
yes
となります。
```

次に、X と Y の年の差を計算する述語、`diffage(X,Y)` を定義してみましょう。X の生年と Y の生年の差を

とれば良いわけですから、

```
diffage(X,Y,Z):- birth_year(X,Xb), birth_year(Y,Yb), Z is abs(Xb-Yb).
```

となります。

```
| ?- diffage(cathy,sam,X).
X = 88 ?
yes
```

ここで、`abs(X)` は X の絶対値を計算してくれます。また、数値演算結果をある変数へ代入する場合は、`=` (コンフィケーション) ではなく、`is` を使います。`=` は単に右辺と左辺を単一化し、`is` は、式を評価した後、値を代入します。両者の違いを見るために、以下のプログラムを実行してみてください。

```
| ?- Z = 1+2.
| ?- Z is 1+2.
```

次に、ある人の年齢を計算する述語を考えてみましょう。ここでの年齢は、

```
IF      すでに亡くなっている THEN 年齢=(没年-生年)
ELSE   年齢=(今年-生年)
```

と定義します。この場合のプログラムは以下のようになります。

```
age(X, Age) :- dead_year(X, Pyear), !, birth_year(X, Byear), Age is Dyear-Byear.
age(X, Age) :- this_year(Tyear), birth_year(X, Byear), Age is Tyear-Byear.
```

第一番目の節は、すでに亡くなっている (没年が記述されている) ならば、没年から生年を引いて年齢を計算することを表しています。また第二番目の節は、今年から生年を引いて年齢を計算しています。また第一番目の節に現れる、`!` はカットと呼ばれる組み込み述語です。カットの正確な定義は、結構難しいのですが、ここでは、`IF .. THEN .. ELSE` を表現するためのものと考えてください。すなわち、

```
age(X, Age) :- dead(X, Dyear) !, birth_year(X, Byear),...
               IF(条件)
age(X, Age) :- this_year(Tyear),.....
               ELSE...
               THEN...
```

となります。

問題 3.1 以下の関係を定義してみよう。また、以下の質問に答えなさい。

1. X は Y の兄である (`elder_brother(X,Y)`)
2. X は Y の妹である (`younger_sister(X,Y)`)
3. X は Y と同世代である (`same_age(X,Y)`) (今回は年の差が 5 以内の場合同世代とします。)
4. 興さんの方が年上の夫婦はどれですか?
5. 年の差が 5 つ以上ある夫婦はどれですか?

演算

$X + Y$	整数(浮動小数点)の和
$X - Y$	整数(浮動小数点)の差
$X * Y$	整数(浮動小数点)の積
X / Y	浮動小数点の商
$X // Y$	整数の商(小数以下切捨て)
$X \bmod Y$	余り
$\text{abs}(X)$	X の絶対値

比較

$X > Y$	X は Y より大きい
$X < Y$	X は Y より小さい
$X >= Y$	X は Y 以上である
$X <= Y$	X は Y 以下である
$X =:= Y$	等しい
$X \backslash= Y$	等しくない
$X \text{ is } Y$	式の評価値を unify する
$X = Y$	X と Y の unify の可能性を判定

参考(項(atom)の比較)

$A == B$	A と B は項として同一である.
$A \backslash= B$	A と B は項として同一でない.
$A @< B$	A は標準順序で B より小
$A @<= B$	A は標準順序で B 以下
$A @> B$	A は標準順序で B より大
$A @>= B$	A は標準順序で B 以上

4 リスト処理の基礎

要素 E_1, E_2, \dots, E_n が並んだものをリストと呼び $[E_1, E_2, \dots, E_n]$ と書きます. 例えば, リスト `[mary, jack, jimmy]` は, オブジェクト `mary, jack, jimmy` の並びのことを意味します. また, リストに含まれる要素の数をリストの長さといいます. 例えば, リスト `[mary, jack, jimmy]` の長さは3です. また, 要素一つも持たないリストも存在します. 長さが0(要素を持たない)リストのことを空リストと呼び`[]`と書きます.

リストは, その要素にリストも持つことができます. 例えば, `[mary, [tom, elizabeth], joseph]` は, 2番目の要素として, リスト `[tom, elizabeth]` を持つリストです. また, `[mary, [tom, elizabeth], joseph]` の長さは3です. リストは, その順番に意味があります. つまり, `[mary, tom, joseph]` と `[tom, joseph, mary]` は, 各要素は同じですが, その順序が異なるので, 異なるリストとして認識されます.

リストに対してできるもっとも基本的なオペレーションは, リストの分解と合成です. これについて簡単に見てみましょう.

まず, リストは, `head` と `tail` からなります. `head` とはリストの先頭の要素, `tail` とはリストから `head` を取り除いた残りのリストです. このリストを `head` と `tail` にわけける処理は,

```
[Head|_Tail]
```

で, 実現できます. 例えば,

```
[?- [1,2,3]=[_N|_Y].
```

を実行すると, リスト `[1,2,3]` が, $X=1, Y=[2,3]$ に分解されます. X が `head` で, Y が `tail` です. また逆に,

```
[?- X=1, Y=[2,3], Z=[_N|_Y].
```

とすると, $Z=[1,2,3]$ が得られ, リストが合成されます. この様に `Prolog` では, リストの分解も合成も, 同じ操作で実現されます. では, 次に長さが1のリストを `head` と `tail` に分解してみましょう.

```
[?- [1] = [_N|_Y].
```

とすると, $X=1, Y=[]$ となります. つまりリストの最後は, 空リストになっています. これら例から分かるように,

```
[1,2,3] = [[2|3]|[]]
```

と, `[1,2,3]` を `|` を使って, 先頭の要素と残りのリストとして表現することができます.

問題 4.1 次のリストを, `|` を使って表わしてみましょう.

- `[a,b,c]`
- `[a,b]_1[b,c]`

では, 次にリストの2番目の要素を取り出してみましょう. この場合,

```
[?- [1,2,3]=[_N0_|_X|_Y].
```

とします. これにより, $X=2$ となり, リスト `[1,2,3]` の2番目の要素が取り出せます. この様にリスト基本は一つのリストを `|` の前後で分けることです. ただし, `|` の後ろは一つでなければなりません. つまり,

```
[?- [1,2,3]=[_N0|_N02|_N03].
```

などとはできません.

次に基本的なリスト処理プログラムである, `member`, `append`, `reverse` について見てみましょう.

`member`: `member(X,Y)` は要素 X はリスト Y に含まれている, すなわち, リスト Y は要素に X を持っている, ことを表わします. この `member` 述語は以下のように定義されます.

```
member(H, [_|_]).
member(H, [_:_T]) :- member(H, _T).
```

プログラムを簡単に見てみましょう. 第一引数の `H` が第二引数のリストの先頭にある場合, `H` はリストに含まれる, ことを表わしています. 第二引目の節は「もし `H` が `T` に含まれるのならば, `H` は `T` の先頭にある要素の一つ追加したリストにも含まれる, ことを表わしています」.

```
[?- member(a, [q, w, a, s]).
Yes
[?- member(1, [a, b, c]).
no
```

`append`: `append(X,Y,Z)` はリスト X とリスト Y を連結したリストは Z である, ことを表わし, 以下のように定義されます.

```
append([], Y, Y).
append([_N|X], Y, [_N|Z]) :- append(X, Y, Z).
```

第一引目の節は「空リストとリスト Y を連結したリストは Y である, ことを表わします. 第二引目の節は「もし X と Y を連結したリストが Z ならば, X の先頭に `W` を加えたリストと Y を連結したリストは Z の先頭に `W` を加えたリストである, ことを表わします」.

```
l ?- append([1,2,3],[a,b,c],Z).
Z = [1,2,3,a,b,c] ?
yes
```

`reverse: reverse(X,Y)` はリスト X の要素の並びを逆転したリストは Y である、ことを表わし、以下のよう
に定義されます。

```
reverse([],[]).
reverse([W|X],Y):- reverse(X,Xr),append(Xr,[W],Y).
```

第一番目の節は「空リストを反転したリストは空リストである」ことを表わし、第二番目の節は「もし X を
反転したリストが Xr ならば、 X の先頭に W を加えたリストを反転させたリストは、 Xr の最後に W を加えた
リストである」、ことを表わします。

```
l ?- reverse([1,2,3],X).
X = [3,2,1] ?
yes
```

ところで、`member` 述語や `append` 述語は、非決定的に利用することができます。つまり例えば、

```
l ?- member(X,[1,2,3]).
X = 1 ? ? ;
X = 2 ? ? ;
X = 3 ?
yes
```

のように、 X がリストに含まれているかをチェックするだけでなく、リストに含まれている要素を取り出す
ことにも利用できます。

問題 4.2 `append,member,reverse` プログラムをファイルに書いて、いろいろ実行してみましょう。また、以下
の実行結果はどのようになるか確認してください。

```
1. l ?- append([1,2],X,[1,2,3,4]).
2. l ?- append(X,[1,2],[3,4,1,2]).
3. l ?- append(X,Y,[1,2,3,4]).
```

次に、もう少し複雑なリスト処理についてみてみましょう。対象は、授業でも扱った集合論演算です。

`subset: subset(X,Y)` は X は Y の部分集合である、という意味です。プログラムは、以下のようになります。

```
subset([],_Y).
subset([W|X],Y):- member(W,Y),subset(X,Y).
```

第一番目の節は「空リストは任意のリストの部分集合である」を表わし、第二番目の節は「 $[W|X]$ が Y の部
分集合であるためには、 W が Y の要素で X が Y の部分集合である必要がある」、ことを表わします。

```
l ?- subset([3,2,4],[1,2,3,4,5,6]).
yes
l ?- subset([3,2,1],[2,3,4,5]).
no
```

`intersection: intersection(X,Y,Z)` は集合 X と集合 Y の共通集合は Z である、ことを表わし、以下のよう
に定義されます。

```
intersection([],_,[]).
intersection([W|X],Y,[W|Z]):- member(W,Y),intersection(X,Y,Z).
intersection([W|X],Y,Z):- \member(W,Y),intersection(X,Y,Z).
```

ここで `\+` は否定を表します。ではプログラムを見てみましょう。第一番目の節は「空集合と任意の集合の共
通集合は空集合である」を表わします。第二番目の節は「 $[W|X]$ と Y の共通集合は、 W が Y に含まれるので
あれば、 X と Y の共通集合に W を加えた集合である」を表わします。また第三番目の節は「 $[W|X]$ と Y の共
通集合は、 W が Y に含まれないのであれば、 X と Y の共通集合と同じである」を表わします。

```
l ?- intersection([1,2,3],[4,3,2],X).
X = [2,3] ?
yes
l ?- intersection([1,2,3],[a,b,c],X).
X = [] ?
yes
```

ところで、この `intersection` の定義では、第二番目の節と第三番目の節でそれぞれ、`member` と `\+member`
を使っています。しかし、これらは両方一度に成り立ちことばないので、`IF..THEN..ELSE` 文を使った表現
したほうが自然です。つまり、

```
IF W が Y の要素ならば (member(W,Y)), THEN 二番目の節を適用
ELSE 三番目の節を適用
```

となり、`!(カット)` を使って、以下のように書き直すことができます。

```
intersection([],_,[]).
intersection([W|X],Y,[W|Z]):- member(W,Y),!,intersection(X,Y,Z).
intersection([W|X],Y,Z):- intersection(X,Y,Z).
```

問題 4.3 以下の集合論演算プログラム書きなさい。また、作成したプログラムを実行し、きちんと動いている
かを確認してください。

1. X は Y の要素である (`member(X,Y)`.)
2. 集合 X と集合 Y は等しい (`eqset(X,Y)`)
3. 集合 X と集合 Y の和集合は Z である (`union(X,Y,Z)`.)
4. 集合 X から集合 Y を引いた差集合は Z である (`setdiff(X,Y,Z)`.)

5 数式処理とリスト処理の応用

ここでは、第 3 章で作成したプログラムを利用します。
今「`elizabeth` には何人の子供がいるか」を、親子関係データベースから検索したいと思います。その場合、

```

| ?- parent(elizabeth,C).
C = mary ? ;
C = jane ? ;
C = jack ? ;
no
| ?-

```

とすれば、elizabeth には3人の子供がいるということがわかります。ですが、これは「ユーザが;を入力することで別の可能性を求め、最後にユーザ自身で数を数えて」得た結果です。ここでの目的は、この一連の作業を Prolog 自身にやらせることです。そのためには、elizabeth の子供を全部集めて、その数を数えるという作業を Prolog にやらせる必要があります。

すべての解を求める: すべての解を求めるには、組み込み述語findall を利用します。findall には3つの引数があります。それぞれの引数の意味は以下の通りです。

- 第一引数: 求めたい解
- 第二引数: その解が満たしている条件
- 第三引数: 結果 (リストの形で出てくる)

例えば、以下のように利用します。

```

| ?- findall(X,parent(elizabeth,X),C).
C = [mary,jane,jack] ?

```

yes

つまり、今求めたいのは、parent(elizabeth,X) という条件 (elizabeth の子供) を満たす X であり、それをリストの形で集めた結果が C ということになります。また、複数の要素の組を求めることもできます。例えば、子供の名前と一っしよに年齢も求めたければ、以下のようにします。

```

| ?- findall([X,Age],(parent(elizabeth,X),age(X,Age)),Z).
Z = [[mary,28],[jane,25],[jack,22]] ?

```

yes

また、実際には、findall を使うと重複した答えが得られる場合があります。その場合は、findall のかわりに setof を使ってください。setof は findall とおなじように利用できますが、得られる結果から自動的に重複を取り除くという処理をしてくれます。

集めた解の数を数える: findall で解を集めた後で、その要素数を数えることで「elizabeth には何人の子供がいるか」という問いに答えることができます。リストのようその数を数えるプログラムは、以下のように定義できます。

```

length([],0).
length([_:H|_],L):- length(T,TL),L is TL+1.
| ?- length([[],1,[],2],X).
X = 4 ?

```

第一番目の節は「空リストの長さは0である」を示し、第二番目の節は「もしリスト T の長さが TL ならば、リスト T の先頭に要素を一つ加えたリストの長さは、TL+1 になる」を示します。

では、先ほどの findall とこの length を使って、X の子供の数を求める述語、how_many_children(X,Y) を定義してみましょう。その定義は、

```

how_many_children(X,Y):- findall(C,parent(X,C),CList),length(CList,Y).

```

となります。

上の length(X,Y) を様々な述語に変えることで、様々な処理が可能になります。例えば、

1. 子供の年齢の合計を求める
2. 子供の平均年齢を求める
3. 最年少 (最年長) の子供を求める
4. 第二子を求める

などです。これらは、基本的に、上の length(X,Y) を変えることで可能です。それぞれについて簡単に見ていきましょう。

数値リストの要素を合計する: リストの要素を合計するプログラムは以下のようになります。

```

sumlist([],0).
sumlist([H|_],S):- sumlist(T,ST), S is ST+H.
| ?- sumlist([1,2,3,4],X).
X = 10 ?

```

第一番目の節は「空リストの要素の合計は0である」を意味し、第二番目の節は「もしリスト T の要素の合計が ST ならば、T に要素 H を追加したリストの要素の合計は、ST+H である」ことを意味します。

数値リストの要素の平均を求める: リストの要素の平均は、これまで出てきた、length と sumlist を使って、簡単に、実現することができます。

リストの n 番目の要素を取り出す: リストの n 番目の要素を取り出すプログラムは、以下のようになります。(n は 1 から数えます。)

```

nth(1, [H|_T], H).
nth(N, [_H|_T], E):-N>1, N1 is N-1, nth(N1, T, E).
| ?- nth(4, [1,2,3], X).
X = 1 ?

```

じっくり眺めて、何をしているか理解してください。

数値リストの要素の最大(小)値を求める: リストの要素の最大値を求めるプログラムは以下ようになります。

```

max(X|_X),
max([H|_T], M):-max(T, MT), max1(H, MT, M).
max1(_H, MT, MT).
| ?- max([3,4,2,1], X).
X = 4 ?

```

数値リストをソートする: リストの要素をソートするプログラムは以下ようになります。他にも色々なソートの仕方があります。

```

%リストの要素を大きい順に並び変える %
%Selection Sort
sort_list([], []).
sort_list(List, [Max|Sorted]):-
  select_max(List, Max, Rest),
  sort_list(Rest, Sorted).

select_max([X]_X, [], []).
select_max([X|Y]_M, Max, Rest):-select_max(Y, X, Max, Rest).
select_max([], Max, Max, []).
select_max([X|Y]_M, Max, [M|Rest]):-X > M, !, select_max(Y, X, Max, Rest).
select_max([X|Y]_M, Max, [X|Rest]):-select_max(Y, M, Max, Rest).

| ?- sort_list([2,3,4,1,2,3,5], X).
X = [1,2,2,3,3,4,5] ?

```

sort_list の第二番目の前に着目してください。これは何をしているかというと、(1)Listの要素を最大のもの(NMax)、それ以外 (Rest)に分ける(2)残りをソートする(sort)=ソート済みリストを得る、(3)ソート済みリストの先頭に最大値であった要素を追加する、ということを行っています。

その他のソートプログラム

参考までに、その他のソートプログラムを紹介します。これらのプログラムは、数値のリストを小さい順に並べ替えるというものです。

```

%
%Naive Sort
%
msort(List, Sorted):-perm(List, Sorted), sorted(Sorted), !.
sorted([]).
sorted(_L_).
sorted([H, _H1|_T]):-H <= H1, sorted([H1|_T]).

perm([], []).
perm([X|Y], Z):-perm(Y, U), select(X, Z, U).

select(H, [H|_T], T).
select(H, [_W|_T], [W|_T1]):-select(H, T, T1).

%
%Selection Sort
%
ssort([], []).
ssort(List, [Min|Sorted]):-min(List, Min), select(Min, List, Rest), ssort(Rest, Sorted).

min([X]_X),
min([H|_T], M):-min(T, TM), min1(H, TM, M).
min1(_H, TM, TM).

```

```

%
%Insertion Sort
%
insert([], []).
insert([H|_T], Sorted):-
  insert(T, SortedT),
  insert(H, SortedT, Sorted).

insert(X, [Y|_S], [Y|S1]):-X > Y, !,
insert(X, S, S1).
insert(X, S, [X|S]).

```

```

%
%Bubble Sort
%
bsort([], []).
bsort([H|_T], Y):-bsort(1, [H|_T], Y).

bsort(0, X, X).
bsort(1, X, Y):-swap(X, Z, 0, N), bsort(N, Z, Y).

swap([X, Y|Z], [Y|Z1], M, M0):-X > Y, !, swap([X|Z], Z1, 1, M0).
swap([X|Z], [X|Z1], M, M0):-swap(Z, Z1, M, M0).
swap([], [], M, M0).

```

```

%
%Merge sort
%
msort(□, □):-!.
msort([X], [X]):-!.
msort(X,Y):-div_list(X,X1,X2),msort(X1,Y1),msort(X2,Y2),!,merge(Y1,Y2,Y).

div_list(□,□,□).
div_list([H|_], [H|X1], X2):-div_list(T,X2,X1).

merge(□, Y, Y).
merge(X, □, X).
merge([A|X], [B|Y], [A|R]):-A < B,!,merge(X,[B|Y],R).
merge(X,[B|Y],[B|R]):-merge(X,Y,R).

%
%Quick Sort
%
qsort(List,Sorted):-qsort(List,Sorted,□).

qsort(□, H, H).
qsort([A|B], H, T):-partition(B,A,S,L),qsort(S,H,[A|T]),qsort(L,T,T).

partition(□,_,_,□,□).
partition([A|B],X,[A|S],L):-AX,!,partition(B,X,S,L).
partition([A|B],X,S,[A|L]):-partition(B,X,S,L).

```

次に X の子供を年齢順に並び替えるプログラムを紹介します。基本的には、上で出てきたプログラムを少し変形させて利用します。

```

sort_child(Parent,Childs):-
  findall([Name,Att], (parent(Parent,Name),age(Name,Att)),Lists),
  sort_with_att(Lists,Childs).

sort_with_att(□,□).
sort_with_att(List,[Max|Sorted]):-
  select_max_with_att(List,Max,Rest),
  sort_with_att(Rest,Sorted).

select_max_with_att([X],X,□):-!.
select_max_with_att([X|Y],Max,Rest):-
  select_max_with_att(Y,X,Max,Rest).
select_max_with_att(□,Max,Max,□).
select_max_with_att([[H,Att]|Y],[H,Att],Max,[H1,Att1|Rest]):-
  Att > Att1,!,
  select_max_with_att(Y,[H,Att],Max,Rest).
select_max_with_att([X|Y],M,Max,[X|Rest]):-
  select_max_with_att(Y,M,Max,Rest).

```

問題 5.1 以下の述語を定義してみよう。また以下の質問に答えなさい。

- length,average,max,sort 述語を実際に動かしてみよう。
- 数値リストの要素の平均値を求める述語 average(List,Avg) を定義しなさい。
- 数値リストの要素の最小値を求める述語 min(List,Min) を定義しなさい。

- 数値リストの要素を小さい順に並び替える述語を定義しなさい。
- 男性の合計年齢を求めなさい。
- 女性の平均年齢を求めなさい。
- 子供の数がもっとも多い女性はどれか?
- 男性を年齢順に並び替えなさい
- 男性を年齢が高い順に3人求めよ。

6 出力

Prolog では、出力のための述語として、write が用意されている。また、nl は改行を表す。

```

| ?- write('Hello World!'),nl.
Hello World!

yes
| ?- X='Hello World!', write(X),nl.
Hello World!

X = 'Hello World!' ?

```

その他の入出力関係の述語に関しては、マニュアルを参照してください。

7 その他

日本語の扱いについて

基本的に、日本語の使用は避けたほうが良いです。どうしても使用したい場合は、日本語の部分を(シングルクォート)で囲ってください。また、文字コードは、EUCを使用してください。さらにKTermの文字コードもEUCにセットしてください。

Prologのオンラインマニュアルについて

SICStus Prologのマニュアル(英語のみ)を、WWW上から利用することができます。

- URL: http://www.sfc.keio.ac.jp/~mukai/sicstus3doc/sicstus_toc.html
- URL: <http://www.sics.se/ps/sicstus.html>

PC上でのPrologの処理系について

ライセンスの問題で、SICStus PrologをPCにインストールすることができません。ですが、Windows95/98上で動作するフリーのPrologの処理系がいくつかあります。以下のURLから入手可能です。

- <http://swi.psych.uva.nl/nst/jan/SWI-Prolog.html>
- <http://clement.info.umoncton.ca/BinProlog/>
- <http://www.cad.msc.kyutech.ac.jp/people/zhou/bprolog.html>

1.のSWI-Prologは、利用方法がSICStus Prologに近いので、SWI-Prologをインストールすることを勧めます。尾崎(tozaki@sfc.keio.ac.jp)の方で、最新版をCNS上にDownloadしてあります。そちらを使ってください。

参考文献

- [1] 古川康一, Prolog 入門, オーム社, 1986.
- [2] I. プラトコ著 (安部憲広訳). "Prolog と AI" (全 2 巻). 近代科学社, 1983
- [3] 塚本龍男 著."わかる Prolog". 情報処理入門シリーズ 12, 共立出版後藤滋樹. "Prolog への入門". サイエンス社, 1984.
- [4] 黒川利明 著."Prolog のソフトウェア作法". 岩波コンピュータサイエンス, 岩波書店
- [5] 黒川利明, 田村直之 著."Prolog プログラミング入門". KE 養成講座 6, オーム社
- [6] コワルスキ 著, 浦昭二 監, 山田眞市 ほか訳."論理による問題の解法-Prolog 入門". 情報処理シリーズ 8, 培風館
- [7] スターリング ほか著, 松田利夫 訳."Prolog の技芸". 共立出版
- [8] 高野真 著."Prolog で学ぶ AI 手法 — 推論システムと自然言語処理". 啓学出版
- [9] 後藤滋樹 著."記号処理プログラミング". 岩波講座 ソフトウェア科学 8, 岩波書店

解答 2.1 この *Prolog* プログラムは、事実のみから構成されています。この基本的な関係を使って、以下の述語を定義してみましょう (*family.pl* に以下の述語の定義を追加してください)。また、定義した述語を利用して、*Prolog* へ様々な質問をしてみましょう。

1. father(F,C):-parent(F,C),male(F).
2. mother(M,C):-parent(M,C),female(C).
3. sibling(X,Y):-parent(P,Y),parent(P,X),X \= Y
4. grandfather(GF,C):-father(GF,P),parent(P,C).
5. grandmother(GM,C):-mother(GM,P),parent(P,C).
6. grandparent(GP,C):-parent(GP,P),parent(P,C).
7. grandchild(C,GP):-grandparent(GP,C).
8. uncle(U,X):-sibling(U,P),male(U),parent(P,X).
9. aunt(A,X):-sibling(A,P),female(A),parent(P,X).
10. nephew(N,X):-male(N),parent(P,N),sibling(P,X).
11. niece(N,X):-female(N),parent(P,N),sibling(P,X).
12. cousin(X,Y):-parent(P,X),parent(P,Y),sibling(P,X,PY).
13. ancestor(A,X):-parent(A,X).
 ancestor(A,X):-parent(A,T),ancestor(T,X).
 relation(X,Y):-ancestor(X,Y).
14. relation(X,Y):-ancestor(Y,X).
 relation(X,Y):-ancestor(Z,X),ancestor(Z,Y).
 relation(X,Y):-ancestor(X,Z),ancestor(Y,Z).

解答 3.1 以下の関係を定義してみよう。また、以下の質問に答えなさい。

1. X は Y の兄である (elder_brother(X,Y)).
 elder_brother(X,Y):-male(X),parent(P,X),parent(P,Y),
 birth_year(X,Xy),birth_year(Y,Yy),Xy < Yy.
2. X は Y の妹である (younger_sister(X,Y)).
 younger_sister(X,Y):-female(X),parent(P,X),parent(P,Y),
 birth_year(X,Xy),birth_year(Y,Yy),Xy > Yy.
3. X は Y と同世代である (same_age(X,Y)) (今回は年の差が 5 以内の場合同世代とします。)
 same_age(X,Y):-birth_year(X,Xy),birth_year(Y,Yy),
 X \= Y, 5 >= abs(Xy-Yy).
4. 興さんの方が年上の夫婦はどれですか?
 | ?- male(M),parent(M,C),female(F),parent(F,C),
 birth_year(M,My),birth_year(F,Fy),Fy < My.

 C = jimmy,
 F = betty,
 M = jack,
 Fy = 1972,
 My = 1976 ? ;
5. 年の差が 5 つ以上ある夫婦はどれですか?
 | ?- parent(F,C),parent(M,C),F \= M,
 birth_year(F,Fy),birth_year(M,My),5 = < abs(Fy-My).

```
C = tim,  
F = mary,  
M = bob,  
Fy = 1970,  
My = 1963 ? ;  
(以下省略)
```

解答 4.1 次のリストを、| を使って表わしてみましょう。

1. [a,b,c] = [a|b|c|[]]

```
2. [(a,b)|b,c] = [(a|b|c)|(|b|c|)|(|c|)|(|)|]
```

解答 4.2 *append, member, reverse* プログラムをファイルに書いて、いろいろ実行してみよう。また、以下の実行結果はどのようになるか確認してください。

```
1. l ?- append([1,2],X,[1,2,3,4]).
   X = [3,4] ?
   yes
2. l ?- append(X,[1,2],[3,4,1,2]).
   X = [3,4] ?
   yes
3. l ?- append(X,Y,[1,2,3,4]).
   X = [], Y = [1,2,3,4] ? ;
   X = [1], Y = [2,3,4] ? ;
   X = [1,2], Y = [3,4] ? ;
   X = [1,2,3], Y = [4] ? ;
   X = [1,2,3,4], Y = [] ? ;
   no
```

解答 4.3 以下の集合論演算プログラム書きなさい。また、作成したプログラムを実行し、きちんと動いているかを確認してください。

```
%XはYの要素である (member(X,Y).)
member(H,[_|_]).
member(H,[_:T]) :- member(H,T).

subset([],_).
subset([_|X],Y) :- member(W,Y),subset(X,Y).

%集合Xと集合Yは等しい (eqset(X,Y).)
eqset(X,Y) :- subset(X,Y),subset(Y,X).

%集合Xと集合Yの和集合はZである (union(X,Y,Z).)
union([],_,Y).
union([_|X],Y,Z) :- member(W,Y),!,union(X,Y,Z).
union([_|X],Y,[_|Z]) :- union(X,Y,Z).

%集合Xから集合Yを引いた差集合はZである (setdiff(X,Y,Z).)
setdiff([],_,[]).
setdiff([_|X],Y,Z) :- member(W,Y),!,setdiff(X,Y,Z).
setdiff([_|X],Y,[_|Z]) :- setdiff(X,Y,Z).
```

解答 5.1 以下の述語を定義してみよう。また以下の質問に答えなさい(必要に応じてプログラムをする必要があります)。

```
1. 省略
2. 数値リストの要素の平均値を求める述語 average(List,Avg). を定義しなさい。
   average(List,Avg) :-
   sumList(List,Sum),length(List,Len),Avg is Sum/Len.
3. 数値リストの要素の最小値を求める述語 min(List,Min). を定義しなさい。
   min([_],_).
   min([_|_],M) :- min(T,TM),min(H,TH),M.
   min(H,TH,H) :- H < TH,!.
   min(_H,TH,TH).
4. 数値リストの要素を小さい順に並び替える述語を定義しなさい。
   ssort([],[]).
   ssort(List,[_|Sorted]) :-
   min(List,Min),
   del(Min,List,Rest),
   ssort(Rest,Sorted).
```

```
5. 男性の合計年齢を求めなさい。
   l ?- findall(Age,(male(X),age(X,Age)),AgeList),sumList(AgeList,Result).
   Result = 335.
   AgeList = [9,50,35,28,22,4,30,68,70,20] ?
6. 女性の平均年齢を求めなさい。
   l ?- findall(Age,(female(X),age(X,Age)),AgeList),average(AgeList,Result).
   Result = 34.555555555555556.
   AgeList = [28,6,25,23,3,45,72,66,43] ?
7. 子供の数がもっとも多い女性はどれか?
   l ?- findall([Name,No],[female(Name),findall(C,parent(Name,C),Child),length(Child,No)],List),
   sort_with_att(List,[Result|_]).
   List = [[mary,2],[sally,0],[jane,1],[betty,1],[peg,0],[elizabeth,3],[virginia,3],
   [cathy,1],[ellen,1],[...|...]].
   Result = [elizabeth,3] ? ;
8. 男性を年齢順に並び替えなさい
   l ?- findall([Name,Age],[male(Name),age(Name,Age)],List),sort_with_att(List,Result).
9. 男性を年齢が高い順に3人求めよ。
   l ?- findall([Name,Age],[male(Name),age(Name,Age)],List),sort_with_att(List,Sorted),
   length(Result,3),append(Result,_,Sorted).
   Result = [[martin,70],[joseph,68],[tom,50]],
```