List Processing Procedural Language (LPPL)

Brian Smith
June 11, 2007

The List Processing Procedural Language (LPPL) is a functional language, such as C, specifically designed to benefit the processing of arrays of data (lists). The syntax caters to the list, easing the complexity of reading and manipulating them.

There is a concept of a containment list. A containment list is a list which contains other lists in its definition. For example consider the following two lists defined:

```
list1 as int list;
list2 as int list;
```

A containment list could be defined like so:

```
list3 as int list contains(list1,me,list2);
```

The number of elements in list3 would be the size of list1 + the size of list2 + any elements specifically added to list3 (me). This makes these lists dynamic, such that whenever a contained list is modified, the containment list is also modified. Lists can also be defined to contain other lists returned from a function: for example consider a module with these functions:

```
Function getParents;
  input;
  output: int list;
Function getChildren;
  input;
  output: int list;
```

A containment list could be defined like so:

```
list3 as int list contains(getParents(),me,getChildren());
```

It could also be a combination of the two:

```
list3 as int list contains(me,list1,getParents());
```

If unsorted, when iterating through list3, the order will be defined by the *contains* clause, *me* being any elements specifically added to list3. Also any operations on the containment list which effect the contained list do not directly modify the actual contained list, only how the containing list perceives it. However manipulating a contained list directly will effect how a containing list perceives its list.

The primitive data types are:
int – 32 bits
char – 8 bits
string – char list

Multi-dimensional lists are not allowed (Due to time constraints). The only exception is string, you can have a string list (which itself is a char list). All elements in a list must be of the same type.

Some list specific statements in the language
remove regex in *list* – remove entries which match a given regular expression
keep *regex* in *list* – keep entries which match a given regular expression
find *regex* in *list* – return a copy of the list with only the entries that match a given regular expression
arrange *list* ascending [all]– sort the list in an ascending order.
arrange *list* descending [all]– sort the list in a descending order.
list1 Union *list2* – combine *list1* and *list2*, producing a new list with no duplicate results
list1 Intersect *list2* – produce a new list which contains entries both *list1* and *list2* contain
list1 Except *list2* – produce a new list which contains *list1* without entries which were also contained in *list2*

The regular expression syntax will be further defined in the language reference.

The main module example of syntax

```
/* Prints arguments supplied to Main to standard out
Function Main;
input:
  argLength as int,
  args as string list;
output:
  returnCode as int;
Do;
  listEntry as string;
  listArgs as string list;
  listArg = listArg Union args;     /* Copy args to listArg */
  filter args remove null;          /* Remove nulls from list */
  foreach(listEntry in listArgs)
  Do;
     < listEntry | "\n";            /* print listEntry to standard out */
  End;
End;
```

A module which calls a function and prints the returning list

```
Module SomeModule;

/* Returns a list of ints > 6, and a list of ints <= 6 based on an
   input int list
Function processIt;
input:
  inputList as int list;         /* The list to separate */
output:
  list1 as int list;             /* > 6 list */
  list2 as int list;             /* <= 6 list */
  rc as int;                     /* return code */
Do;
  list1 = find ">6" in inputList;
  list2 = find "<=6" in inputList;
  rc = 0;
```

```
End;
```

```
Function Main;
input;
output:
  returnCode as int;
Do;
  listEntry as int;
  retCode as int;
  inputList as int list;
  listYes as int list;
  listNo as int list;

  inputList = inputList Union (1,2,3,4,5,6,7,8,9,10);
  (listYes,listNo,retCode) = processIt(inputList);

  check(retCode is 0)       /* Fail if return code bad */
  Do;
    returnCode = 4;         /* First set the return code */
  End;

  /* Return code okay, keep going */
  > "Allowed Args\n";
  foreach(listEntry in listYes)
  Do;
    > listEntry | "\n";
  End;
  > "Not Allowed\n";
  foreach(listEntry in listNo)
  Do;
    > listEntry | "\n";
  End;
End;
```