A PROGRAMMING ENVIRONMENT IN C

by Larry Simonsen and Keven Miller
VALTEK INCORPORATED
Moutain Springs Parkway
Springville, UT 84663

All programmers have a style of their own. This style is reflected in the data structures they create. The C programming language provides the means to create a programming environment which reflects these dynamic differences of style. This paper describes the programming environment (software) of VALTEK's C programming project.

The C programming language has relatively few data structures and directives. However, each is well defined (though sometimes confusing to those with experience in other languages). We have taken these few data structures, with the capability of creating others, and have built a unique programming environment.

Much like a building is built on a foundation, we (as others can) have built on features defined by the ANSI draft standard.

Our programming block structures can be compared to the levels of the OSI networking structure much like a software network. These levels are as follows:

LEVEL	NAME	SOFTWARE APPLICATION	
7	APPLICATION	MULTI-TRANSACTION ENVIRONMENT	
6	PRESENTATION	TRANSACTION DEFINITION	
5	SESSION	DATA ITEM HANDLER updimqoh	
4	TRANSPORT	USER ENVIRONMENT strinput	USER DBMS INTERFACE imserial
3	NETWORK	DBMS / LANGUAGE SPECIFIC dbget, dbopen / strcopy, float	
2	LINK	OPERATING SYSTEM / FILE SYSTEM fread, dateline	
1	PHYSICAL	HARDWARE OF THE SYSTEM load, store, add, branch	

As in the OSI model, where each layer provides a specific function and can be replaced if the interface to adjacent layers is maintained, so the software model can support this type of procedure/level replacement. The interface between layers needs to be well defined and a given layer should not skip layers to call procedures from lower levels.

Design criteria of new programming environment:

1. System structure (global data) of about 400 bytes so that no global data storage is used in routines. All procedures can then be placed in SL libraries.

2. Each dataset of every database will have a structure corresponding to its record layout. These dataset structures, combined with organized constant definitions, become our data dictionary (i.e. the compiler would know about the dictionary and its structure would appear in program listings).

3. For each command the same database record would not be read twice.

4. Records could be resorted several times without reselecting or rereading the database (local file I/O is faster and does not impead other processes).

5. All database fields can be classified as cost, hours, percent, price, quantity, string, and summary price (if you have others please let us know).

   Each field will have defined field descriptors of ITEM, PROMPT, TITLE, LEN (for arrays), WIDTH, PREC, FORMAT, EPSILON, and MAX. Another possible addition is MIN. Our environment treats the above types as if they were fields and defines the PROMPT, TITLE, WIDTH, PREC, FORMAT, EPSILON, and MAX for each as shown below:

```
#define  costPROMPT  "cost"
#define  costTITLE   "COST"
#define  costWIDTH   12
#define  costPREC    3
#define  costFORMAT  "f"
#define  costEPSILON .001
#define  costMAX     99999999.999
```

   These are then used to define the database fields. For example:

```
#define  poucWIDTH   qtyWIDTH

#define  itnoITEM    1
#define  itnoPROMPT  "Part Number"
#define  itnoTITLE   "PART  NUMBER"
#define  itnoLEN     18
#define  itnoWIDTH   18
#define  itnoPREC    18
#define  itnoFORMAT  "s"
```

6. The most commonly passed parameters to procedures will be the first four.

7. No routine should be longer than 150 lines. These small, easy to prove correct routines are used to build larger procedures. Small procedures can be used elsewhere.

8. Normalize your code as you would your data.

9. Adopt and build on debugging ideas presented from others:

   David Greer - Las Vegas. Build a test suite from user's failure data.

   Denis Heidner - Las Vegas. Save a copy of data that causes failure (snapshot).

   Bruce Toback - Detroit. Be able to (without recompiling) turn on debug printout information.

10. Heavy user envolvement as system evolves. Rely heavily on the small core of knowledgable users. Trust their comments as to user interface and data, realizing, of course, that you're ultimately responsible.

Below is a sample of one of our reports which utilizes the levels we've discussed. Our code has the appearance of a fourth generation language. When compiled it generates considerably faster run time code than our old third generation software.

```
#include "env.h"

#define  cmdNAME  rg100
#define  Dispatcher  gl

#include "accmas.h"  /* def. of   accmas database record */

SUBROUTINE( cmdNAME )
{
  VALBASE_DEF;

/*
REPORTS ARE FORMATED AS

HEAD
TITLE
LINE
LINE
TRAIL
TITLE
LINE
TRAIL
FOOT
<<page break>>

if you do not have a given type of format then delete that format.  each of
these types of formats may be multi-lined.
*/

#define   hdfmtVARMAX        20
#define   hdfmtMAX           1000
```

```
#define   DETAILLINESMAX   2 + 2

#define   ttl0fmtVARMAX    4
#define   ttl0fmtMAX       350
#define   ttl0fmtNUM       2

#define   ln0fmtVARMAX     11
#define   ln0fmtMAX        100
#define   ln0fmtNUM        3

  char   datestr[28];

  UNIT_DEF( unitout, DETAILLINESMAX );
  i2   unitoutbrkflag;

  FMT_DEF ( unitout, UNITFMTNUMHEAD, hdfmtMAX,  hdfmtVARMAX );
  FMT_DEF ( unitout, ttl0fmtNUM, ttl0fmtMAX, ttl0fmtVARMAX );
  FMT_DEF ( unitout, ln0fmtNUM, ln0fmtMAX, ln0fmtVARMAX );

#define   sortvarMAX       2

  char    key0prompt[] = accseqPROMPT;
  char    key1prompt[] = accnumPROMPT;
  i2     sortmask;
#define   fieldsMAX        11

/* define here the db record buffers */
  DEFPTR( accmas_t,  accmasptr);
  db_rec_t accmascurrent, accmashighwater;

/*———————— Program Variables ————————*/

  data_t     *data;
  i2         datanum;
  fieldlist_t  cmdfield[1+11] = { (char*) 11, 0 };

/*      TEMP CALCULATION        */
  char   outputacctype[acctypWIDTH+1];
  char   outputdesc[accdesWIDTH+10+1];
  char   outputstmtype[accsttWIDTH+1];
  char   outputnorbal[accnorWIDTH+1];

#pragma   page
/*————————————————————————————*/
/*                            */
/*          CODE              */
/*                            */
/*————————————————————————————*/
  entervoidroutine( stringof(cmdNAME) );

  _dateline(datestr);
```

```
  UNIT_INIT(unitout);
  fmt_init (unitout, UNITFMTNUMHEAD, UNITFMTTYPEHEAD);
  fmt_init0(unitout, UNITFMTNUMFOOT, UNITFMTTYPEFOOT);
  fmt_init (unitout, ttl0fmtNUM, UNITFMTTYPETITLE);
  fmt_init (unitout, ln0fmtNUM, UNITFMTTYPELINE);

  data = env->data[ datanum = recalloc(0l, 10) ];
  dataalloc(datanum, fieldsMAX, sortvarMAX);

/*custom*/
  /* add the formats */
  datafieldsort(datanum, accmasptr->accseq,
    SORTASCENDING, SORTREAL, key0prompt);
  fmtvar(unitout, ln0fmtNUM, accmasptr->accseq,
    fmtfield(accseq));
  fmt(unitout, ttl0fmtNUM, accseqTITLE, accseqWIDTH);

  datafieldstrsort(datanum, accmasptr->accnum, accnumLEN,
    SORTASCENDING, SORTBYTE, key1prompt);
  fmtstr(unitout, ln0fmtNUM, accmasptr->accnum,
    fmtfield(accnum));
  fmt(unitout, ttl0fmtNUM, accnumTITLE, accnumWIDTH);

  datafieldstr(datanum, accmasptr->accdes, accdesLEN);
  fmtstr(unitout, ln0fmtNUM, outputdesc,
   fmtfield(accdesindent));
  fmt(unitout, ttl0fmtNUM, accdesTITLE, accdesindentWIDTH);

  datafield(datanum, accmasptr->acctyp);
  fmtstr(unitout, ln0fmtNUM, outputacctype, fmtfield(acctyp));
  fmt(unitout, ttl0fmtNUM, acctypTITLE, acctypWIDTH);

  datafieldstr(datanum, accmasptr->accstt, accsttLEN);
  fmtstr(unitout, ln0fmtNUM, outputstmtype, fmtfield(accstt));
  fmt(unitout, ttl0fmtNUM, accsttTITLE, accsttWIDTH);

  datafield(datanum, accmasptr->acccla);
  fmtvar(unitout, ln0fmtNUM, accmasptr->acccla, fmtfield(acccla));
  fmt(unitout, ttl0fmtNUM, accclaTITLE, accclaWIDTH);

  datafield(datanum, accmasptr->accnor);
  fmtstr(unitout, ln0fmtNUM, outputnorbal, fmtfield( accnor));
  fmt(unitout, ttl0fmtNUM, accnorTITLE, accnorWIDTH);

  datafield(datanum, accmasptr->acctol);
  fmtvar(unitout, ln0fmtNUM, accmasptr->acctol, fmtfield(acctol));
  fmt(unitout, ttl0fmtNUM, acctolTITLE, acctolWIDTH);

  datafield(datanum, accmasptr->accskp);
  fmtvar(unitout, ln0fmtNUM, accmasptr->accskp,
    fmtfield(accskp));
  fmt(unitout, ttl0fmtNUM, accskpTITLE, accskpWIDTH);
```

```
  datafield(datanum, accmasptr->accind);
  fmtvar(unitout, ln0fmtNUM, accmasptr->accind,
    fmtfield(accind));
  fmt(unitout, ttl0fmtNUM, accindTITLE, accindWIDTH);

  strcpy(data->sort.defaultinput, "1,2");
  fmtnewline(unitout, ln0fmtNUM);
  fmtnewline(unitout, ttl0fmtNUM);

  unithd(unitout, dateTITLE, 0);
  unithdstr(unitout, datestr, fmtfield(date));
  unithdend(unitout, pageTITLE, 0);
  unithdendvar(unitout, unitout->page, fmtfield( page));
  unithdnewline(unitout);

  unithdcenter(unitout, "G/L ACCOUNT MASTER", 0);
  unithdnewline(unitout);

  unithdcenter(unitout, "==================", 0);
  unithdnewline(unitout);

  unithdcenter(unitout, "SORTED BY ", 0);
  unithdcenterstrprec(unitout, data->sort.desc,
    data->sort.desclen, -fmtfield( sortdesc));
  unithdnewline(unitout);

  unitstatus (unitout, DETAILLINESMAX);
#pragma   page
/*-------------------------------------*/
/ *                                     */
/ *              SELECTION              */
/ *                                     */
/*-------------------------------------*/
/*  open other databases  */

  while (unitselect( unitout ) == INOK) {

    if (!unitopen(unitout, stringof(cmdNAME))) continue;
    headvalpac(unitout, stringof(cmdNAME));

#pragma   page
/*-------------------------------------*/
/ *                                     */
/ *       GATHER AND SORT               */
/ *                                     */
/*-------------------------------------*/

    do {
      sortreset (datanum);
      /* add sort keys if always at begining */
      if ((sortmask = sortinput(datanum)) != INOK) break;
      /* add additional sort keys if always at end  */
```

```
      if (!data->reccnt) {  /* no records selected yet */
        env->anyoutstdout =  0;
      if (env->small)
        start_working( &data->work, WORKING_SELECT,
        "accmas", data->current, &env->smallsize);
      else {
        accmashighwater = accmascap(100);
        start_working(&data->work, WORKING_GATHER,
        "accmas", &accmascurrent, &accmashignwatter);
      }

      while(accmascurrent = accmasserial( accmasptr, 10) {
        if (!env->small) im_working(&data->work);

        recwrite(datanum, 200);
        if (env->small) {
          if (data->reccnt >= env->smallsize) break;
          im_working (&data->work);
        }
      }
    }

  if (data->reccnt == 0) {
    unitnooutput( unitout);
    if (IMAJOB) continue;
    break;
  }

  sort(datanum);

#pragma  page
/*-------------------------------------*/
/*                                     */
/*              OUTPUT                 */
/*                                     */
/*-------------------------------------*/
  unitoutbrkflag = 0;
  unitout->page=-1;

  env->anyoutstdout = ( unitout->stream == stdout);

  start_working(&data->work, WORKING_OUTPUT,
    stringof( cmdNAME), &data->current, &data->reccnt);

  while (data->current < data->reccnt && unitoutbrkflag >= 0) {
    recread(datanum, 300);
    im_working(&data->work);

    memset(outputdesc, ' ',40);
    strncpy(&outputdesc[accmasptr->accind]
      ,accmasptr->accdes ,accdesWIDTH );
    glacctypedesc( outputacctype, accmasptr->acctyp );
```

```
    glstmtypedesc( outputstmtype, accmasptr->accstt );
    glnorbaldesc( outputnorbal, accmasptr->accnor );

    unitoutbrkflag = fmtwritettl( unitout, ln0fmtNUM,
      ttl0fmtNUM);
    if (unitoutbrkflag >= 0)
      unitskip( unitout,
      (accmasptr->accskp != 10 ? accmasptr->accskp : 9999));
  } /* till all records written */

  unitend(unitout);

    } while (TRUE);  /* till all sorts done for this output file */
      unitclose( unitout);

  } /* till no more output */
  recrelease(datanum);

  exitvoidroutine;
}

/*-------------------------------------------------------------------*/

#include "driver.h"
/*-------------------------------------------------------------------*/
```