Share Session - 8145

Using SAS C/C++ Release 7.00

Gary T. Ciampa
Don T. Poitras
Software Development Environments

Gary.Ciampa@sas.com
Don.Poitras@sas.com

The Power to Know.

Introduction and Overview

- Review of 7.00 Compiler and Library Features
  - SAS/C and Utility Enhancements
  - SAS/C++ Enhancements
  - Runtime Library and Debugger Enhancements
  - CICS Transaction Server Support
- Implementing / Exploiting Selected Features
  - long long data type
  - Access Register Mode Support
  - C++ Exception Handling
  - Standard C++ and Tools++ Library
- Future Product Enhancements

SAS/C and Utility Enhancements

- "long long" support - ANSI/C99
  - Format Specifiers; Feature test macro
- Access Register Mode Support
  - Eight byte pointer - Far Pointer
  - Create/delete/allocate data spaces
- Object Module Dis-assembler
  - Extended names and C++ mangled names
- COOL - Serviceability and Reliability
  - Extended names processing; CLET; Cross-reference
  - Compression algorithims to reduce LMOD sizes
  - Improved validity checking for input objects

SAS/C++ Development System

- C++ Exception Handling Semantics
- Namespace support
  - namespace; using; “std:” headers
- Standard C++ Library
  - STL; Allocator, String, Numeric Classes
- Tools++ Class Library
  - string; time; expression libraries
- Ancillary Features or enhancements
  - template template; default template arguments
  - mutable; bool; explicit; wchar_t; new[]; delete[]
  - long long; asciiout;

Library and Debugger Enhancements

- "long long" library family of functions
  - fprintf, printf, sprintf; fscanf, scanf, sscanf; strtoll; llabs
- Access Register Mode, far pointers
- OS/390 eNetwork Communications Server
  - Integrated Sockets Native and USS Applications
  - Stack Affinity Support
- Debugger Support for C/C++ and Library
  - long long; templates; exception handling;
  - Redesigned Debugger User's Guide, native, remote (cics/batch) and cross-compiler debugging (unix, windows)

- CICS Transaction Server
  - TS 1.2 and TS 1.3
  - Application, System and Business Programming Interface
  - long long data type
- Additional Information
  - SAS Institute Web Server
  - http://www.sas.com/products/sasc/release70.html
- Summary of Release 7.00 - Questions

Implementing “long long” support

- Syntax Specification - 64 bit integer
  - long long l_int; unsigned long long ul_int;
  - function declaration long long taxes(void);
  - LLONG_MIN, LLONG_MAX; ULLONG_MAX; <limits.h>
- Constant Definition: 14LL; 0XFF00ULL;

```
#define LLONG_MAX 9223372036854775807LL
```

- Format Specifier: “ll”, “j”

```
unsigned long long ul_int = ULLONG_MAX;
l_int = strtoll("-9223372036854775808", NULL, 10);
printf(”signed and unsigned long long = %lli %016llX\n",
             l_int, ul_int);
```

- Implementation Features
  - doubleword alignment, unless _noalignmem in affect
  - Feature test macro: _SASC_HIDE_LLLIB

Access Register Mode Feature

- Compiler option ARMODE (-Karmode)
- Preprocessor symbol _O_ARMODE
- Functions to allocate and free memory in dataspaces.
- printf() format modifier @ to allow __far pointers as parameters in conjunction with conversion specifiers that dereference pointers.
- Enabled for SPE programs
- C and assembler samples

ARMODE Option

This option generates the use of "far pointers" whenever a pointer is used in the compilation unit.

- SAC 512 issued at start of every function to enter AR Mode.
- Whenever a sub-function call is made, it is bracketed by SAC 0/SAC 512 so that prolog/epilog code is never entered in anything but primary mode.
- New header file arjmp.h should be used when an AR mode program uses setjmp/longjmp etc. to ensure that the jmp_buf buffer is large enough to save and restore the AR registers.

Preprocessor Symbol _O_ARMODE

Set to 1 when the ARMODE compiler option is enabled

- Can be used in user code to conditionally include alternate functions that exploit AR Mode support.
- SAS/C header files have been modified to support __far pointers in string and other functions. For example, in lcstring.h there is conditional code to allow string functions to be remapped to far pointer versions. This header must be included so that the proper functions will be called when the ARMODE option is used for the compile.
- SAS/C headers using symbol: string.h, lcstring.h, lcjmp.h and setjmp.h

Using _O_ARMODE

```
#if _O_ARMODE
#pragma map(memxlt, "#FMEMXLT")
extern void __far *memxlt(void __far *, const char __far *, size_t);
#else
extern void *memxlt(void *, const char *, size_t);
#endif
```

When calling memxlt() in a module compiled with the ARMODE option, and lcstring.h has been #included, all pointers will be converted to __far before the function is called. Pointers that are already __far will not be affected, but __near pointers will be expanded to __far with the ALET portion of the pointer set to 0 to show that the pointer is to data in the primary address space.

Functions to allocate and free dataspace memory

Dataspaces may be created via assembler services or by using SAS/C Interfaces:

- dspserv() and aleserv()
  - Thin wrappers of equivalent assembler interfaces.
  - Allow specification of dataspace attributes needed for authorized programs.
- falloc() and ffree()
  - High level allocation, simple to use as malloc.
  - Default behavior is reasonable for non-authorized uses of dataspaces.
  - Provided as source code for flexibility.

Sample Program

```
#include <osmain.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <lcstring.h>
#include <osdspc.h>
#include <tput.h>
#include <spetask.h>
#define out(x) TPUT(x, strlen(x), 0)
void osmain()
{
char buf[134];
char __far * fpData;
void __far * fptr;
pDSPC pdspc = 0;
pDSPC pdspc2 = 0;
   fpData = falloc(&pdspc, 1000000000); // ho hum, I'll take a billion
                                        // bytes please.
   if (!fpData)
      ABEND(0x111, 0, DUMP, SYSTEM);
```

```
strcpy(fpData, "Some long string1");
sprintf(buf, "String in dataspace = %@s", fpData);
out(buf);
sprintf(buf, "ptr value as long long  = %llp", fpData);
out(buf);
sprintf(buf, "ptr value as __far ptr  = %@p", fpData);
out(buf);
sprintf(buf, "ptr value as hex string = %016llX", fpData);
out(buf);
/* a billion wasn't quite enough... */
fptr = falloc(&pdspc, 50);
strcpy(fptr, "Some long string2");
sprintf(buf, "2nd string = %@s%@n", fptr, fptr);
out(buf);
sprintf(buf, "num chars = %i", *(int __far *) fptr);
out(buf);
sprintf(buf, "2nd allocation in same dataspace = %016llX", fptr);
out(buf);
fpData = falloc(&pdspc2, 1000000000); // yawn, I'll take another
                                      // billion bytes please.
sprintf(buf, "1st allocation in 2nd dataspace = %016llX", fpData);
out(buf);
```

Sample program output

```
String in dataspace = Some long string1
ptr value as long long  = 1000300000000
ptr value as __far ptr  = 1000300000000
ptr value as hex string = 0001000300000000
2nd string = Some long string2
num chars = 30
2nd allocation in same dataspace = 000100033B9AD000
1st allocation in 2nd dataspace = 0001000400000000
```

Ancillary Issues

- _ossvc changed to switch to primary mode before calling SVC and AR mode after SVC completes if ARMODE compiler option set.
- Function _osarmsvc to call an SVC while staying in AR mode.
- For bldexit, if the exit routine does not save access registers and you need the access regs restored when the exit returns control, specify the bldexit attribute _ACCSV.
- If writing AR mode assembler subroutine, save any used access regs. Normally this will be AR2 through AR14. The compiler will not put auto variables or function parms in a dataspace so AR1 can be safely set to 0 if parms are passed to the function. If a far pointer is to be returned then AR15 should be set correctly before exiting.

C++ Exception Handling Standards

- try() catch() and throw() specifications

```
void Xhandler(int test) {         // localize a try/catch to a function
        try {
              if(test)  throw test;
              }
        catch(int i) {                cout << "Caught Exception #: " << i << '\n';   }
        }
void main() {
        cout << "Start\n";
        Xhandler(0);                          Xhandler(1);
        Xhandler(2);                          Xhandler(3);
        cout << "End";                        return 0; }
```

- Built-in types and Objects
- Derived class, Restricted, Re-throw semantics

C++ Exception Handling cont...

- Derived Class Example

```
class B { };
class D: public B { };  // Catching a Derived Class
int main()  {
            D derived;
            try {
                 throw derived;
                 }
            catch(B b) {
                cout << "Caught a base class.\n"; }
            catch(D d) {
                cout << "This won't execute.\n";}
            return 0; }
```

C++ Exception Handling cont...

- Restricted Function Definition

```
void Xhandler(int test) throw(int, char, double) {
    if(test==0) throw test; // throw int
    if(test==1) throw 'a'; // throw char
    if(test==2) throw 123.23; // throw double }
int main() {
    cout << "start\n";
    try  {  Xhandler(0); } // also, try passing 1 and 2 to Xhandler()
    catch(int i) {
        cout << "Caught an integer\n";   }
      //  add a catch clause for a char and double to prevent unexpected exception
cout << "end";
     return 0; }
```

Implementation features

- terminate() and unexpected() handlers
- Compiler Option: except, -Kexcept
- Tracing exceptions - xtrace(); _XTRACE; =xtrace

```
LSCX251 **** NOTE **** Generated in @@596726 called from line 8 of
   @@457683(EXCEPT), offset 0000B4, C++ name: Xhandler, Exception thrown: non-standard type
    Function          Offset                        Line          Context
@@457683(EXCEPT)  == "Xhandler"
                  0000B4                        8
MAIN(EXCEPT)      000070                        9
```

- new operator fails - bad_alloc exception
- C++: The Complete Reference, 3rd Edition
  - Herbert Schildt, ISBN: 0-07-882476-1
  - http://www.osborne.com/program/cpluspluscr.htm

Standard C++ Library - Roguewave

- Comprehensive collection of classes and functions for fine-grained, low-level programming.
- STL - Data structures and algorithms.
  - Ten forms of containers, including vector, list, set and map.
  - Sixty-four generic algorithms for performing operations on the Standard containers, such as initializing, searching, inserting and removing.
- Iterators - cycle through objects; directional
  - forward, back, bi-directional, linked lists, binary tree

Building for OS/390 and CMS/ESA

- OS/390 - Installation
  - RWSTLCLG - Customized Procedure; prefix.RW.*
  - MACLIBC SAMPLE.CXX LIBSTD.A
  - Options: autoinst except refdef rent rtti

```
//ALG  JOB jobcard statements
// JCLLIB ORDER=sasc.prefix.PROCLIB
//STEP1      EXEC RWSTLCLG,MEMBER=RWCX009,SNAME=ALG1
```

- CMS/ESA - Installation
  - Samples and Headers shipped in a MACLIB
  - Global required MACLIBS and TXTLIBS

```
GLOBAL MACLIB RW LCXX370 LC370
GLOBAL TXTLIB LC370BAS LC370STD
LCXX alg1 (autoinst except refdef rent rtti sname alg1
COOL alg1 libstd (cxx genmod
```

Tools++ Library - Roguewave

- Encapsulates the Standard C++ Library with an object-oriented interface.
  - Includes low-level ISO/ANSI features
  - Portable across development / OS environments
  - Library functions implemented in libtools.a archive
  - time, date, enhanced strings, regular expression classes

- OS/390
  - RWTLSCLG - Customized Procedure, prefix.RW.*
  - LIBTOOLS.A for Tools++ functions

```
//BENCH  JOB jobcard statements
// JCLLIB ORDER=sasc.prefix.PROCLIB
//STEP1      EXEC RWTLSCLG,MEMBER=RWCX015,SNAME=BENCH
```

Tools++ - CMS/ESA and Cross-platform

- CMS/ESA
  - Samples and Headers shipped in a MACLIB
  - Global required MACLIBS and TXTLIBS

```
GLOBAL MACLIB RW LCXX370 LC370
GLOBAL TXTLIB LC370BAS LC370STD
LCXX bench (autoinst except refdef rent rtti sname bench
COOL bench libtools libstd (cxx genmod
```

- **Cross-Compiler Development - Tools++**
  - Traditional directory file organization: headers and library
  - Link specifications support two formats:

    libtools.a libstd.a   -L***lib/path*** -ltools -lstd

```
sascc370 -o map.o -Kautoinst -Kexcept -Krefdef -Krent -Krtti \
    -Ksname=map libtools.a libstd.a map.cpp
```

Future Product Enhancements

- Development Strategies
  - Customer (Internal and External) Requirements
  - ISO/ANSI C and C++ Standards
  - OS Developments
- Development Opportunities
  - IEEE Floating Point
  - Linux Hosted Cross-compiler
  - Exploitation of future architectural features
  - XPlink Exploitation
  - LE Enabled Compiler
  - GOFF Object File Format
  - C++ and C99 Improvements

Summary and Conclusion

- Release 7.00 SAS/C, SAS/C++ Compiler Product
- Questions or Comments

Gary.Ciampa@sas.com	Product Manager
Don.Poitras@sas.com	Library Developer
Andrea.Herrin@sas.com	Marketing Manager

- Technical Information
  - http://www.sas.com/products/sasc
  - http://www.sas.com/service/index.html
  - http://www.roguewave.com/support/docs/