```
/*
 * @DEC_COPYRIGHT@
 */
/*
 * HISTORY
 * $Log: branch.anal.c,v $
 * Revision 1.1.2.2  1995/04/27  19:41:39  Greg_Lueck
 *      First revision of productized Atom and tools.
 *      [1995/04/26  21:12:32  Greg_Lueck]
 *
 * $EndLog$
 */
#pragma ident "@(#)$RCSfile: branch.anal.c,v $ $Revision: 1.1.2.2 $ (DEC) $Date: 1995/04/27 19:41:39 $"


#include <stdlib.h>
#include <stdio.h>
#include <assert.h>

#define TABLESIZE 256
int branchHistory[TABLESIZE];

struct BranchType {
  long pc;
  long predicts;
  long references;
} *branches;

int branchct;
FILE *file;

void OpenFile(int number)
{
  branchct = number;
  branches = (struct BranchType *) calloc(sizeof(struct BranchType),number);
  file = fopen("branch.out","w");
}

int takenTable[] =    { 1, 2, 3, 3 };
int notTakenTable[] = { 0, 0, 1, 2 };

void CondBranch(int number, long pc, long taken)
{
  long index, history;

  index = (pc >> 2) & (TABLESIZE-1);
  history = branchHistory[index];
  if (taken) {
    if (history >= 2) branches[number].predicts++;
    branchHistory[index] = takenTable[history];
  } else {
    if (history <= 1) branches[number].predicts++;
    branchHistory[index] = notTakenTable[history];
  }
  branches[number].references++;
  branches[number].pc = pc;
}

void PrintResults(void)
{
  long references;
  long predicts;
  int i;

  fprintf(file,"%18s %20s %20s\n","PC","References","Predicted");
  references = 0;
  predicts = 0;
  for (i = 0; i < branchct; i++) {
    if (branches[i].references > 0) {
```



```
      references += branches[i].references;
      predicts += branches[i].predicts;
      fprintf(file,"0x%016lx %20ld %20f\n",
              branches[i].pc, branches[i].references,
              (float) branches[i].predicts / branches[i].references);
    }
  }
  fprintf(file,"%18s %20ld %20f\n","Totals", references,
                  (float) predicts / references);
  fclose(file);
}
```

```
/*
 * @DEC_COPYRIGHT@
 */
/*
 * HISTORY
 * $Log: branch.inst.c,v $
 * Revision 1.1.2.2  1995/04/27  19:41:48  Greg_Lueck
 *      First revision of productized Atom and tools.
 *      [1995/04/26  21:12:48  Greg_Lueck]
 *
 * $EndLog$
 */
#pragma ident "@(#)$RCSfile: branch.inst.c,v $ $Revision: 1.1.2.2 $ (DEC) $Date: 1995/04/27 19:41:48 $"


/*
        Branch prediction: instruments all conditional branches to
        determine how many are predicted correctly.
 */
#include <stdio.h>
#include <cmplrs/atom.inst.h>

static int branch = 0;

void InstrumentInit(int argc, char **argv)
{
   AddCallProto("OpenFile(int)");
   AddCallProto("CondBranch(int,REGV,VALUE)");
   AddCallProto("PrintResults()");
   AddCallProgram(ProgramAfter,"PrintResults");
}

Instrument(int argc, char **argv, Obj *obj)
{
   Proc *p;  Block *b; Inst *i;

   for (p = GetFirstObjProc(obj); p != NULL; p = GetNextProc(p)) {
     for (b = GetFirstBlock(p); b != NULL; b = GetNextBlock(b)) {
       i = GetLastInst(b);
       if (IsInstType(i,InstTypeCondBr)) {
         AddCallInst(i,InstBefore,"CondBranch",branch++,
                     REG_PC,BrCondValue);
       }
     }
   }
}

void InstrumentFini(void)
{
   AddCallProgram(ProgramBefore,"OpenFile",branch);
}
```



```
/*
 * @DEC_COPYRIGHT@
 */
/*
 * HISTORY
 * $Log: cache.anal.c,v $
 * Revision 1.1.2.2  1995/04/27  19:41:58  Greg_Lueck
 *      First revision of productized Atom and tools.
 *      [1995/04/26  21:13:05  Greg_Lueck]
 *
 * $EndLog$
 */
#pragma ident "@(#)$RCSfile: cache.anal.c,v $ $Revision: 1.1.2.2 $ (DEC) $Date: 1995/04/27 19:41:58 $"


/*
        Cache: determine cache miss rate if application runs in 8K
        direct-mapped cache.
 */

#include <stdio.h>
#include <assert.h>

#define CACHE_SIZE 8192
#define BLOCK_SHIFT 5

long tags[CACHE_SIZE >> BLOCK_SHIFT];
long references, misses;

void Reference(long address)
{
  int index = (address & (CACHE_SIZE-1)) >> BLOCK_SHIFT;
  long tag = address >> BLOCK_SHIFT;

  if (tags[index] != tag) {
      misses++;
      tags[index] = tag;
    }
  references++;
}

void PrintResults()
{
  FILE *file;
  file = fopen("cache.out","w");
  assert(file != NULL);
  fprintf(file,"References: %ld\n",references);
  fprintf(file,"Misses:    %ld\n",misses);
  fprintf(file,"Miss Rate: %lf\n",(double) misses / references);
  fclose(file);
}
```

```
/*
 * @DEC_COPYRIGHT@
 */
/*
 * HISTORY
 * $Log: cache.inst.c,v $
 * Revision 1.1.2.2  1995/04/27  19:42:07  Greg_Lueck
 *      First revision of productized Atom and tools.
 *      [1995/04/26  21:13:18  Greg_Lueck]
 *
 * $EndLog$
 */
#pragma ident "@(#)$RCSfile: cache.inst.c,v $ $Revision: 1.1.2.2 $ (DEC) $Date: 1995/04/27 19:42:07 $"


#include <stdio.h>
#include <cmplrs/atom.inst.h>

void InstrumentInit(int p1, char **p2) {
   AddCallProto("Reference(VALUE)");
   AddCallProto("PrintResults()");
   AddCallProgram(ProgramAfter,"PrintResults");
}

Instrument(int argc, char **argv, Obj *obj)
{
   Proc *p;  Block *b; Inst *i;

   for (p = GetFirstObjProc(obj); p != NULL;    p = GetNextProc(p)) {
     for (b = GetFirstBlock(p); b != NULL; b = GetNextBlock(b)) {
       for (i = GetFirstInst(b); i != NULL; i = GetNextInst(i))  {
         if (IsInstType(i,InstTypeLoad)) {
           AddCallInst(i,InstBefore,"Reference",EffAddrValue);
         }
       }
     }
   }
}
```



```
/*
 * @DEC_COPYRIGHT@
 */
/*
 * HISTORY
 * $Log: dtb.anal.c,v $
 * Revision 1.1.2.2  1995/04/27  19:42:15  Greg_Lueck
 *      First revision of productized Atom and tools.
 *      [1995/04/26  21:13:33  Greg_Lueck]
 *
 * $EndLog$
 */
#pragma ident "@(#)$RCSfile: dtb.anal.c,v $ $Revision: 1.1.2.2 $ (DEC) $Date: 1995/04/27 19:42:15 $"


/*
        dtb: Determine the number of dtb misses if the application uses
        8Kb pages and a fully associative translation buffer.
 */

#include <stdio.h>
#include <assert.h>

#define NEXTDTB(index)  (((index)+1) & (DTBSIZE-1))
#define PAGESIZE 8192
#define PAGESHIFT 13
/* must be power of 2 */
#define DTBSIZE 32

long dtb[DTBSIZE];

int nextMapping = 0;
long hits,misses;

#define MAPSIZE 8192
int pageMap[MAPSIZE];

void Reference(long address)
{
  long page = address >> PAGESHIFT;
  int i = pageMap[page & (MAPSIZE-1)];

  if (dtb[i] != page) {
    for (i = 0; i < DTBSIZE; i++) {
      if (dtb[i] == page) {
        pageMap[page & (MAPSIZE - 1)] = i;
        break;
      }
    }
  }
  if (i >= DTBSIZE) {
    misses++;
    dtb[nextMapping] = page;
    pageMap[page & (MAPSIZE-1)] = nextMapping;
    nextMapping = NEXTDTB(nextMapping);
  } else {
    hits++;
    if (nextMapping == i) {
      nextMapping = NEXTDTB(i);
    }
  }
}

void PrintResults()
{
  FILE *file;
  file = fopen("dtb.out","w");
  assert(file != NULL);
  fprintf(file,"DTB hits:     %12ld\n",hits);
```

```
  fprintf(file,"DTB Misses:    %12ld\n",misses);
  fprintf(file,"DTB references: %12ld\n",hits+misses);
  fprintf(file,"Miss Rate:     %12.4lf\n",(double) misses/(hits+misses));
  fclose(file);
}
```



```
/*
 * @DEC_COPYRIGHT@
 */
/*
 * HISTORY
 * $Log: dtb.inst.c,v $
 * Revision 1.1.2.2  1995/04/27  19:42:28  Greg_Lueck
 *      First revision of productized Atom and tools.
 *      [1995/04/26  21:13:47  Greg_Lueck]
 *
 * $EndLog$
 */
#pragma ident "@(#)$RCSfile: dtb.inst.c,v $ $Revision: 1.1.2.2 $ (DEC) $Date: 1995/04/27 19:42:28 $"


#include <stdio.h>
#include <cmplrs/atom.inst.h>

void InstrumentInit(int p1, char **p2) {
   AddCallProto("Reference(VALUE)");
   AddCallProto("PrintResults()");
   AddCallProgram(ProgramAfter,"PrintResults");
}

Instrument(int argc, char **argv, Obj *obj)
{
   Proc *p; Block *b; Inst *i;

   for (p = GetFirstObjProc(obj); p != NULL; p = GetNextProc(p)) {
     for (b = GetFirstBlock(p); b != NULL; b = GetNextBlock(b)) {
       for (i = GetFirstInst(b); i != NULL; i = GetNextInst(i))  {
         if (IsInstType(i,InstTypeLoad) || IsInstType(i,InstTypeStore)) {
           AddCallInst(i,InstBefore,"Reference",EffAddrValue);
         }
       }
     }
   }
}
```

```
/*
 * @DEC_COPYRIGHT@
 */
/*
 * HISTORY
 * $Log: dyninst.anal.c,v $
 * Revision 1.1.2.2  1995/04/27  19:42:43  Greg_Lueck
 *      First revision of productized Atom and tools.
 *      [1995/04/26  21:14:06  Greg_Lueck]
 *
 * $EndLog$
 */
#pragma ident "@(#)$RCSfile: dyninst.anal.c,v $ $Revision: 1.1.2.2 $ (DEC) $Date: 1995/04/27 19:42:43 $"


/*
        dyninst: fundamental dynamic counts (instructions, loads,
        stores, blocks, procedures).
 */

#include <stdio.h>
#include <assert.h>

long insts = 0;
long blocks = 0;
long procs = 0;
long loads = 0;
long stores = 0;

void AddProcedure()
{
   procs++;
}

void AddBlock(int instsBlock, int loadsBlock, int storesBlock)
{
  insts += instsBlock;
  loads += loadsBlock;
  stores += storesBlock;
  blocks++;
}

void PrintCounts()
{
  FILE *file;
  file = fopen("dyninst.out","w");
  assert(file != NULL);
  fprintf(file,"Instructions %ld\n",insts);
  fprintf(file,"Loads %ld\n",loads);
  fprintf(file,"Stores %ld\n",stores);
  fprintf(file,"Blocks %ld\n",blocks);
  fprintf(file,"Procedures: %ld\n",procs);
  fclose(file);
}
```



```
/*
 * @DEC_COPYRIGHT@
 */
/*
 * HISTORY
 * $Log: dyninst.inst.c,v $
 * Revision 1.1.2.2  1995/04/27  19:42:55  Greg_Lueck
 *      First revision of productized Atom and tools.
 *      [1995/04/26  21:14:21  Greg_Lueck]
 *
 * $EndLog$
 */
#pragma ident "@(#)$RCSfile: dyninst.inst.c,v $ $Revision: 1.1.2.2 $ (DEC) $Date: 1995/04/27 19:42:55 $"


#include <stdio.h>
#include <cmplrs/atom.inst.h>

void InstrumentInit(int p1, char **p2) {
   AddCallProto("AddProcedure()");
   AddCallProto("AddBlock(int,int,int)");
   AddCallProto("PrintCounts()");
   AddCallProgram(ProgramAfter,"PrintCounts");
}

Instrument(int argc, char **argv, Obj *obj)
{
   Proc *p; Block *b;  Inst *i;
   int instPerBlock;
   int loadsPerBlock;
   int storesPerBlock;

   for (p = GetFirstObjProc(obj); p != NULL; p = GetNextProc(p)) {
     AddCallProc(p,ProcBefore,"AddProcedure");
     for (b = GetFirstBlock(p); b != NULL; b = GetNextBlock(b)) {
       instPerBlock = 0;
       loadsPerBlock = 0;
       storesPerBlock = 0;
       for (i = GetFirstInst(b); i != NULL; i = GetNextInst(i)) {
         instPerBlock++;
         if (IsInstType(i,InstTypeLoad)) {
           loadsPerBlock++;
         } else if (IsInstType(i,InstTypeStore)) {
           storesPerBlock++;
         }
       }
       AddCallBlock(b,BlockBefore,"AddBlock",instPerBlock,
                    loadsPerBlock, storesPerBlock);
     }
   }
}
```

```
/*
 * @DEC_COPYRIGHT@
 */
/*
 * HISTORY
 * $Log: inline.anal.c,v $
 * Revision 1.1.2.2  1995/04/27  19:46:32  Greg_Lueck
 *      First revision of productized Atom and tools.
 *      [1995/04/26  21:18:52  Greg_Lueck]
 *
 * $EndLog$
 */
#pragma ident "@(#)$RCSfile: inline.anal.c,v $ $Revision: 1.1.2.2 $ (DEC) $Date: 1995/04/27 19:46:32 $"


/*
        inline: identify potential candidates for inlining
 */

#include <stdlib.h>
#include <stdio.h>
#include <assert.h>

long *counts;
FILE *file;

void OpenFile(int number)
{
  counts = (long *) calloc(sizeof(long),number);
  file = fopen("inline.out","w");
  assert(file != NULL);
  fprintf(file,"%12s %25s %25s %12s\n", "PC", "Parent", "Called", "Number");
}

void CallSiteCount(int index)
{
  counts[index]++;
}

void CallSitePrint(int index, long pc, char *nameCalled, char *nameParent)
{
  if (counts[index] > 0) {
    fprintf(file,"0x%010lx %25s %25s %14ld\n",
            pc, nameParent, nameCalled, counts[index]);
  }
}

void CloseFile()
{
  fclose(file);
}
```



```
/*
 * @DEC_COPYRIGHT@
 */
/*
 * HISTORY
 * $Log: inline.inst.c,v $
 * Revision 1.1.2.2  1995/04/27  19:46:43  Greg_Lueck
 *      First revision of productized Atom and tools.
 *      [1995/04/26  21:19:06  Greg_Lueck]
 *
 * $EndLog$
 */
#pragma ident "@(#)$RCSfile: inline.inst.c,v $ $Revision: 1.1.2.2 $ (DEC) $Date: 1995/04/27 19:46:43 $"


#include <stdio.h>
#include <cmplrs/atom.inst.h>

static int numberCallSites=0;

static const char *     SafeProcName(Proc *);


void InstrumentInit(int argc, char **argv)
{
   AddCallProto("OpenFile(int)");
   AddCallProto("CallSiteCount(int)");
   AddCallProto("CallSitePrint(int,long,char *,char *)");
   AddCallProto("CloseFile()");
   AddCallProgram(ProgramAfter,"CloseFile");
}

Instrument(int argc, char **argv, Obj *obj)
{
   Proc *p; Block *b; Inst *i;
   ProcRes target;
   int opcode,function;

   for (p = GetFirstObjProc(obj); p != NULL; p = GetNextProc(p)) {
     for (b = GetFirstBlock(p); b != NULL; b = GetNextBlock(b)) {
       for (i = GetFirstInst(b); i != NULL; i = GetNextInst(i)) {
         /* JSR and BSR */
         opcode = GetInstInfo(i,InstOpcode);
         function = (GetInstInfo(i,InstMemDisp) >> 14) & 0x3;
         if ((opcode == 0x1A && function == 1 ) || opcode == 0x34) {
           ResolveTargetProc(i, &target);
           if (target.name != NULL) {
             AddCallInst(i,InstBefore,"CallSiteCount", numberCallSites);
             AddCallObj(obj, ObjAfter,"CallSitePrint",
                        numberCallSites, InstPC(i), target.name,
                        SafeProcName(p));
             numberCallSites++;
           }
         }
       }
     }
   }
}

void InstrumentFini(void)
{
   AddCallProgram(ProgramBefore,"OpenFile",numberCallSites);
}


static const char *SafeProcName(Proc *p)
{
    const char *        name;
    static char         buf[128];
```

```
    name = ProcName(p);
    if (name)
        return(name);
    sprintf(buf, "proc_at_0x%lx", ProcPC(p));
    return(buf);
}
```



```
/*
 * @DEC_COPYRIGHT@
 */
/*
 * HISTORY
 * $Log: iprof.anal.c,v $
 * Revision 1.1.2.2  1995/04/27  19:46:54  Greg_Lueck
 *      First revision of productized Atom and tools.
 *      [1995/04/26  21:19:20  Greg_Lueck]
 *
 * $EndLog$
 */
#pragma ident "@(#)$RCSfile: iprof.anal.c,v $ $Revision: 1.1.2.2 $ (DEC) $Date: 1995/04/27 19:46:54 $"


#include <stdio.h>
#include <string.h>
#include <stdlib.h>
#include <unistd.h>

long instrTotal = 0;
long *instrPerProc;
long *callsPerProc;

FILE *OpenUnique(char *fileName, char *type)
{
  FILE *file;
  char Name[200];

  if (getenv("ATOMUNIQUE") != NULL)
    sprintf(Name,"%s.%d",fileName,getpid());
  else
    strcpy(Name,fileName);

  file = fopen(Name,type);
  if (file == NULL)
    {
      fprintf(stderr,"Atom: can't open %s for %s\n",Name, type);
      exit(1);
    }
  return(file);
}

static FILE *file;
void OpenFile(int number)
{
  file = OpenUnique("iprof.out","w");
  fprintf(file,"%30s %15s %15s %12s\n","Procedure","Calls",
          "Instructions","Percentage");
  instrPerProc = (long *) calloc(sizeof(long), number);
  callsPerProc = (long *) calloc(sizeof(long), number);
  if (instrPerProc == NULL || callsPerProc == NULL) {
    fprintf(stderr,"Malloc failed\n");
    exit(1);
  }
}

void ProcedureCalls(int number)
{
  callsPerProc[number]++;
}

void ProcedureCount(int number, int instructions)
{
  instrTotal += instructions;
  instrPerProc[number] += instructions;
}
```

```
void ProcedurePrint(int number, char *name)
{
  if (instrPerProc[number] > 0) {
    fprintf(file,"%30s %15ld %15ld %12.3f\n",
            name, callsPerProc[number], instrPerProc[number],
            100.0 * instrPerProc[number] / instrTotal);
  }
}

void CloseFile()
{
  fprintf(file,"\n%30s %15s %15ld\n", "Total", "", instrTotal);
  fclose(file);
}
```



```
/*
 * @DEC_COPYRIGHT@
 */
/*
 * HISTORY
 * $Log: iprof.inst.c,v $
 * Revision 1.1.2.2  1995/04/27  19:47:03  Greg_Lueck
 *      First revision of productized Atom and tools.
 *      [1995/04/26  21:19:34  Greg_Lueck]
 *
 * $EndLog$
 */
#pragma ident "@(#)$RCSfile: iprof.inst.c,v $ $Revision: 1.1.2.2 $ (DEC) $Date: 1995/04/27 19:47:03 $"


#include <stdio.h>
#include <cmplrs/atom.inst.h>

static int n = 0;

static const char *     SafeProcName(Proc *);


void InstrumentInit(int argc, char **argv)
{
   AddCallProto("OpenFile(int)");
   AddCallProto("ProcedureCalls(int)");
   AddCallProto("ProcedureCount(int,int)");
   AddCallProto("ProcedurePrint(int,char*)");
   AddCallProto("CloseFile()");
   AddCallProgram(ProgramAfter,"CloseFile");
}

Instrument(int argc, char **argv, Obj *obj)
{
   Proc *p; Block *b;

     for (p = GetFirstObjProc(obj); p != NULL; p = GetNextProc(p)) {
       AddCallProc(p,ProcBefore,"ProcedureCalls",n);
       for (b = GetFirstBlock(p); b != NULL; b = GetNextBlock(b)) {
         AddCallBlock(b,BlockBefore,"ProcedureCount",
                      n,GetBlockInfo(b,BlockNumberInsts));
       }
       AddCallObj(obj, ObjAfter,"ProcedurePrint",n,SafeProcName(p));
       n++;
     }
}

void InstrumentFini(void)
{
   AddCallProgram(ProgramBefore,"OpenFile",n);
}

static const char *SafeProcName(Proc *p)
{
    const char *        name;
    static char         buf[128];

    name = ProcName(p);
    if (name)
        return(name);
    sprintf(buf, "proc_at_0x%lx", ProcPC(p));
    return(buf);
}
```

```
/*
 * @DEC_COPYRIGHT@
 */
/*
 * HISTORY
 * $Log: malloc.anal.c,v $
 * Revision 1.1.2.2  1995/04/27  19:47:19  Greg_Lueck
 *      First revision of productized Atom and tools.
 *      [1995/04/26  21:19:51  Greg_Lueck]
 *
 * $EndLog$
 */
#pragma ident "@(#)$RCSfile: malloc.anal.c,v $ $Revision: 1.1.2.2 $ (DEC) $Date: 1995/04/27 19:47:19 $"

/*
**  malloc.anal.c -     Analysis routines for malloc recording tool.
**
**      This tool records each call to the routine malloc() and prints a
**      summary of the application's allocated memory.  See "malloc.inst.c"
**      for a description of how to use this tool.
*/

#include <stdlib.h>
#include <stdio.h>
#include <assert.h>
#include <unistd.h>


/*
 * We record a histogram of allocation request.  Requests for less than
 * this amount are included in the histogram.  Requests for more than this
 * amount are reported outside of the histogram.
 */
#define MAXSIZE         16384

/*
 * Allocate an array of structures to record information about various
 * sized allocations.  Allocate an additional structure to hold
 * information about large allocations.
 */
struct {
    long        calls;
    long        memoryAllocated;
} mallocHistory[MAXSIZE], mallocOther;


/*
 * Malloc() may call itself recursively.  This keeps track of recursive
 * calls, so we only record allocations made directly by the application.
 */
int     depth = 0;


/*
 * These keep track of the initial, current, and maximum 'break' value
 * for the application.  The 'break' value is the last address of the
 * application's heap area.  We use these to determine the amount of
 * heap memory actually used by the application.
 */
long    StartBrk;
long    CurBrk;
long    MaxBrk;


/*
 * The handle for the output file.
 */
FILE *  file;
```



```

/*
**  Record a call to malloc().
*/
void BeforeMalloc(long size)
{
   /*
    * Ignore recursive calls.
    */
   if (depth++ != 0)
        return;

   /*
    * Record the allocation request.
    */
   if (size < MAXSIZE) {
        mallocHistory[size].calls++;
        mallocHistory[size].memoryAllocated += size;
   } else {
        mallocOther.calls++;
        mallocOther.memoryAllocated += size;
   }
}


/*
**  Note that we have returned from a malloc() call, so we can keep track
**  of recursive calls.
*/
void AfterMalloc()
{
    depth--;
}


/*
**  Record a call to brk().  This call changes the application's
**  'break' value.
*/
void BeforeBrk(long newbrk)
{
    CurBrk = newbrk;
    if (CurBrk > MaxBrk)
        MaxBrk = CurBrk;
}


/*
**  Record a call to sbrk().  This call increases the application's
**  'break' value.
*/
void BeforeSbrk(long size)
{
    CurBrk += size;
    if (CurBrk > MaxBrk)
        MaxBrk = CurBrk;
}


/*
**  Create the output file and set the initial address for the
**  application's 'break' value.
*/
void Initialize(long startbrk)
{
    file = fopen("malloc.out", "w");
    if (file == NULL) {
        fprintf(stderr, "malloc: Unable to create file 'malloc.out'.\n");
```

```
        exit(1);
    }

    StartBrk = startbrk;
    MaxBrk = startbrk;
    CurBrk = startbrk;
}


/*
**  Print the summary information about this application's malloc() calls.
*/
void PrintResults()
{
    long        calls;
    long        allocated;
    int         i;


    fprintf(file, "%8s %15s %20s\n", "Size", "calls", "allocated");

    calls = 0;
    allocated = 0;
    for (i = 0; i < MAXSIZE; i++) {
        if (mallocHistory[i].calls > 0) {
            fprintf(file, "%8ld %15ld %20ld\n", i,
                mallocHistory[i].calls, mallocHistory[i].memoryAllocated);
            calls += mallocHistory[i].calls;
            allocated += mallocHistory[i].memoryAllocated;
        }
    }

    if (mallocOther.calls > 0) {
        fprintf(file, ">=%7ld %15ld %20ld\n", MAXSIZE,
            mallocOther.calls, mallocOther.memoryAllocated);
        calls += mallocOther.calls;
        allocated += mallocOther.memoryAllocated;
    }

    fprintf(file, "\n%8s %15ld %20ld\n", "Totals", calls, allocated);
    fprintf(file, "\nMaximum Memory Allocated: %ld bytes\n", MaxBrk - StartBrk);
    fclose(file);
}
```



```
/*
 * @DEC_COPYRIGHT@
 */
/*
 * HISTORY
 * $Log: malloc.inst.c,v $
 * Revision 1.1.2.2  1995/04/27  19:47:30  Greg_Lueck
 *      First revision of productized Atom and tools.
 *      [1995/04/26  21:20:13  Greg_Lueck]
 *
 * $EndLog$
 */
#pragma ident "@(#)$RCSfile: malloc.inst.c,v $ $Revision: 1.1.2.2 $ (DEC) $Date: 1995/04/27 19:47:30 $"

/*
**  malloc.inst.c -     Instrumentation routines for malloc recording tool.
**
**      This tool records each call to the routine malloc() and prints a
**      summary of the application's allocated memory.
*/

#include <stdio.h>
#include <cmplrs/atom.inst.h>


/*
**  Perform once-per-application initializations.
*/
void InstrumentInit(int iargc, char **iargv)
{
    unsigned long       BrkStart;
    Obj *               ObjMain;


    /*
     * Define prototypes for the analysis routines.
     */
    AddCallProto("Initialize(long)");
    AddCallProto("BeforeMalloc(REGV)");
    AddCallProto("AfterMalloc()");
    AddCallProto("BeforeBrk(REGV)");
    AddCallProto("BeforeSbrk(REGV)");
    AddCallProto("PrintResults()");

    /*
     * Calculate the start of this application's heap and pass it to the
     * initialization routine in the analysis code.  The application's
     * heap starts immediately after the main object's uninitialized
     * data area.
     */
    ObjMain = GetFirstObj();
    BrkStart = GetObjInfo(ObjMain, ObjUninitDataStartAddress) +
        GetObjInfo(ObjMain, ObjUninitDataSize);
    AddCallProgram(ProgramBefore, "Initialize", BrkStart);

    /*
     * Print out the results when the application finishes.
     */
    AddCallProgram(ProgramAfter, "PrintResults");
}


/*
 * Instrument each object in this application.
 */
Instrument(int iargc, char **iargv, Obj *obj)
{
    ProcRes     pres;
```

```
    /*
     * If this object defines malloc(), add an instrumentation point
     * to trace it.
     */
    ResolveNamedProc("malloc", &pres);
    if (pres.proc != NULL) {
        AddCallProc(pres.proc, ProcBefore, "BeforeMalloc", REG_ARG_1);
        AddCallProc(pres.proc, ProcAfter, "AfterMalloc");
    }

    /*
     * If this object defines brk(), add an instrumentation point.
     * This allows us to keep track of the end of the application's heap.
     */
    ResolveNamedProc("__brk", &pres);
    if (pres.proc != NULL) {
        AddCallProc(pres.proc, ProcBefore, "BeforeBrk", REG_ARG_1);
    }

    /*
     * If this object defines sbrk(), add an instrumentation point.
     * This allows us to keep track of the end of the application's heap.
     */
    ResolveNamedProc("__sbrk", &pres);
    if (pres.proc != NULL) {
        AddCallProc(pres.proc, ProcBefore, "BeforeSbrk", REG_ARG_1);
    }
}
```



```
#include <stdlib.h>
#include <stdio.h>
#include <assert.h>

long instrTotal = 0;
long *instrPerProc = 0;

FILE *file;

void OpenFile(int number) {
        instrPerProc = (long *) calloc(sizeof(long),number);
        assert(instrPerProc != NULL);
        file = fopen("prof.out","w");
        assert(file != NULL);
        fprintf(file,"%30s %15s %10s\n","Procedure","Instructions","%");
}

void ProcedureCount(int number, int instructions) {
        instrTotal += instructions;
        instrPerProc[number] += instructions;
}

void ProcedurePrint(int number, char *name) {
        if (!instrPerProc[number] > 0)
                return;
        fprintf(file,"%30s %15ld %9.3f\n", name, instrPerProc[number],
                        100.0 * instrPerProc[number] / instrTotal);
}

void CloseFile() {
        fprintf(file,"\n%30s %15ld\n", "Total", instrTotal);
        fclose(file);
}
```

```
#include <stdio.h>
#include <cmplrs/atom.inst.h>

static int number;
static int nprocs;

static const char *     pname(Proc *);

/*
 * Initialization: supply prototypes + clean up file I/O (may not
 * be flushed otherwise).
 */
void InstrumentInit(int argc, char **argv) {
        AddCallProto("OpenFile(int)");
        AddCallProto("ProcedureCount(int,int)");
        AddCallProto("ProcedurePrint(int,char *)");
        AddCallProto("CloseFile()");
        AddCallProgram(ProgramAfter, "CloseFile");
}

int Instrument(int argc, char **argv, Obj *obj) {
        Proc *p; Block *b;

        nprocs += GetObjInfo(obj, ObjNumberProcs);
        for (p = GetFirstObjProc(obj); p; p = GetNextProc(p)) {
                for (b = GetFirstBlock(p); b; b = GetNextBlock(b)) {
                        AddCallBlock(b,BlockBefore,"ProcedureCount",
                                number,GetBlockInfo(b,BlockNumberInsts));
                }
                AddCallObj(obj, ObjAfter, "ProcedurePrint",number,pname(p));
                number++;
        }
        return 0;
}

/* Why is open file here? */
void InstrumentFini(void) {
        AddCallProgram(ProgramBefore, "OpenFile",nprocs);
}

static const char *pname(Proc *p) {
        const char *    name;
        static char     buf[128];

        if ((name = ProcName(p)))
                return(name);
        sprintf(buf, "proc_at_0x%lx", ProcPC(p));
        return(buf);
}
```



```
/*
 * @DEC_COPYRIGHT@
 */
/*
 * HISTORY
 * $Log: ptrace.anal.c,v $
 * Revision 1.1.2.2  1995/04/27  19:48:28  Greg_Lueck
 *      First revision of productized Atom and tools.
 *      [1995/04/26  21:21:35  Greg_Lueck]
 *
 * $EndLog$
 */
#pragma ident "@(#)$RCSfile: ptrace.anal.c,v $ $Revision: 1.1.2.2 $ (DEC) $Date: 1995/04/27 19:48:28 $"


#include <stdio.h>

void ProcTrace(char *name)
{
  fprintf(stderr,"%s\n",name);
}
```

```
/*
 * @DEC_COPYRIGHT@
 */
/*
 * HISTORY
 * $Log: ptrace.inst.c,v $
 * Revision 1.1.2.2  1995/04/27  19:48:40  Greg_Lueck
 *      First revision of productized Atom and tools.
 *      [1995/04/26  21:21:51  Greg_Lueck]
 *
 * $EndLog$
 */
#pragma ident "@(#)$RCSfile: ptrace.inst.c,v $ $Revision: 1.1.2.2 $ (DEC) $Date: 1995/04/27 19:48:40 $"


#include <stdio.h>
#include <cmplrs/atom.inst.h>

static const char *     SafeProcName(Proc *);


Instrument(int argc, char **argv, Obj *obj)
{
   Proc *p;

   AddCallProto("ProcTrace(char *)");
   for (p = GetFirstObjProc(obj); p != NULL; p = GetNextProc(p)) {
     AddCallProc(p,ProcBefore,"ProcTrace",SafeProcName(p));
   }
}

static const char *SafeProcName(Proc *p)
{
    const char *        name;
    static char         buf[128];

    name = ProcName(p);
    if (name)
        return(name);
    sprintf(buf, "proc_at_0x%lx", ProcPC(p));
    return(buf);
}
```



```
/*
 * @DEC_COPYRIGHT@
 */
/*
 * HISTORY
 * $Log: trace.anal.c,v $
 * Revision 1.1.2.2  1995/04/27  19:49:57  Greg_Lueck
 *      First revision of productized Atom and tools.
 *      [1995/04/26  21:23:20  Greg_Lueck]
 *
 * $EndLog$
 */
#pragma ident "@(#)$RCSfile: trace.anal.c,v $ $Revision: 1.1.2.2 $ (DEC) $Date: 1995/04/27 19:49:57 $"


#include <stdio.h>
#include <assert.h>
#define BUFSIZE 65536
char buf[BUFSIZE];

FILE *file;
void OpenFile()
{
  file = fopen("trace.out","w");
  assert(file != NULL);
  setbuffer(file,buf, BUFSIZE);
}

void InstReference(long pc)
{
  pc |= 1L << 63;
  fwrite(&pc,sizeof(pc),1,file);
}

void DataReference(long address)
{
  fwrite(&address,sizeof(address),1,file);
}

void CloseFile()
{
  fclose(file);
}
```

```
/*
 * @DEC_COPYRIGHT@
 */
/*
 * HISTORY
 * $Log: trace.inst.c,v $
 * Revision 1.1.2.2  1995/04/27  19:50:10  Greg_Lueck
 *      First revision of productized Atom and tools.
 *      [1995/04/26  21:23:34  Greg_Lueck]
 *
 * $EndLog$
 */
#pragma ident "@(#)$RCSfile: trace.inst.c,v $ $Revision: 1.1.2.2 $ (DEC) $Date: 1995/04/27 19:50:10 $"


#include <stdio.h>
#include <cmplrs/atom.inst.h>

void InstrumentInit(int p1, char **p2) {
   AddCallProto("OpenFile()");
   AddCallProto("InstReference(REGV)");
   AddCallProto("DataReference(VALUE)");
   AddCallProto("CloseFile()");

   AddCallProgram(ProgramBefore,"OpenFile");
   AddCallProgram(ProgramAfter,"CloseFile");
}

Instrument(int argc, char **argv, Obj *obj)
{
   Proc *p; Block *b; Inst *i;

   for (p = GetFirstObjProc(obj); p != NULL; p = GetNextProc(p)) {
     for (b = GetFirstBlock(p); b != NULL; b = GetNextBlock(b)) {
       AddCallBlock(b,BlockBefore,"InstReference",REG_PC);
       for (i = GetFirstInst(b); i != NULL; i = GetNextInst(i)) {
         if (IsInstType(i,InstTypeLoad) || IsInstType(i,InstTypeStore)) {
           AddCallInst(i,InstBefore,"DataReference",EffAddrValue);
         }
       }
     }
   }
}
```