```
// $Id: maprefcount.cpp,v 1.2 2012-05-09 21:27:42-07 - - $

//
// Illustrate how to avoid leaks may for a map by wrapping each
// pointer in an auto_ptr.  Thus the map has no pointers itself.
// But C++11 deprecates auto_ptr in favor of unique_ptr or something
// else.  We use our own object_ptr and reference counting on the
// object itself.  Note that object_ptr properly overrides the
// default four members.  We also handle an object_ptr not having
// an object.
//

#include <iostream>
#include <map>

using namespace std;

int seqct = 0;
struct object {
   int ref;
   int seqnr;
   string value;
   explicit object (const string &val):
         ref(1), seqnr(++seqct), value(val) {}
};

struct object_ptr {
   object *obj;
   void incr() { if (obj) ++obj->ref; }
   void decr() { if (obj && --obj->ref == 0) delete obj; }
   explicit object_ptr (object *_obj): obj(_obj) {}

   // Following are the default four.
   object_ptr(): obj(0) {}
   object_ptr (const object_ptr &that): obj(that.obj) { incr(); }
   object_ptr &operator= (const object_ptr &that) {
      if (this != &that) { decr(); obj = that.obj; incr(); }
      return *this;
   }
   ~object_ptr() { decr(); }
};

typedef map <string, object_ptr> somap_t;
typedef somap_t::const_iterator somap_conitor;

int main (int argc, char **argv) {
   map <string, object_ptr> somap;

   // Push each element of argv into map as object.
   for (int index = 1; index < argc; ++index) {
      string arg = argv[index];
      somap[arg] = object_ptr (new object (arg));
   }

   // Iterate over the map, printing out the keys and values.
   for (somap_conitor itor = somap.begin();
        itor != somap.end(); ++itor) {
      cout << itor->first << " => (" << itor->second.obj->seqnr << ", "
           << itor->second.obj->value << ")" << endl;
   }

   return 0;
}

```

```
/*
//TEST// valgrind --leak-check=full --show-reachable=yes \
//TEST//       --log-file=maprefcount.out1.grind \
//TEST//       maprefcount these are some arguments to check on leak \
//TEST//       >maprefcount.out1 2>&1
//TEST// mkpspdf maprefcount.ps maprefcount.cpp* maprefcount.out*
*/

```

```
@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@ mkc: starting maprefcount.cpp
maprefcount.cpp: $Id: maprefcount.cpp,v 1.2 2012-05-09 21:27:42-07 - - $
g++ -g -O0 -Wall -Wextra maprefcount.cpp -o maprefcount -lm
rm -f maprefcount.o
@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@ mkc: finished maprefcount.cpp
```

```
1: are => (2, are)
2: arguments => (4, arguments)
3: check => (6, check)
4: leak => (8, leak)
5: on => (7, on)
6: some => (3, some)
7: these => (1, these)
8: to => (5, to)
```

```
==1501== Memcheck, a memory error detector
==1501== Copyright (C) 2002-2009, and GNU GPL’d, by Julian Seward et al.
==1501== Using Valgrind-3.5.0 and LibVEX; rerun with -h for copyright info
==1501== Command: maprefcount these are some arguments to check on leak
==1501== Parent PID: 1500
==1501==
==1501==
==1501== HEAP SUMMARY:
==1501==     in use at exit: 0 bytes in 0 blocks
==1501==   total heap usage: 24 allocs, 24 frees, 746 bytes allocated
==1501==
==1501== All heap blocks were freed -- no leaks are possible
==1501==
==1501== For counts of detected and suppressed errors, rerun with: -v
==1501== ERROR SUMMARY: 0 errors from 0 contexts (suppressed: 4 from 4)
```