```c
// $Id: numlist.c,v 1.1 2012-02-09 18:53:08-08 - - $

//
// Demo of how to use malloc and free.
//

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

//
// Declare the type of the handle, or pointer, to the struct.
// In Java, the same name is used for both the handle and the
// struct.
//
// Declare the type of the node.  This is much like Java, except
// that the word ‘struct’ is used.  C does not allow functions
// to be declared inside structs, as does Java.
//
typedef struct node *node_ref;
struct node {
   double item;
   node_ref link;
};

//
// The main program allocates some nodes, pushes them onto a list,
// prints them out, and then frees up the nodes.
//
int main (int argc, char **argv) {
   char *progname = basename (argv[0]);

   //
   // Declare and set the head of the list to NULL.
   //

   node_ref head = NULL;

   //
   // Loop, pushing some random numbers onto the list.  Note that
   // ‘->’ in C means ‘.’ in Java.  Malloc(3c) is used to allocate
   // storage, like ‘new’ in Java.  Always check with ‘assert’ that
   // malloc has actually returned the address of new memory.
   // ‘sizeof’ returns the number of bytes necessary for its
   // argument.
   //
   int max = argc < 2 ? 10 : atoi (argv[1]);
   printf ("%s: looping %d times\n", progname, max);
   for (int count = 0; count < max; ++count) {
      node_ref new = malloc (sizeof (struct node));
      assert (new != NULL);
      new->item = drand48() * 1e6;
      new->link = head;
      head = new;
   }

   //
   // Loop down the list, printing out each entry in debug mode.
   //
   printf ("&head= %p\n", (void*) &head);
   printf ("head= %p\n", (void*) head);
   for (node_ref curr = head; curr != NULL; curr = curr->link) {
      printf ("%p -> struct node {item= %.15g, link= %p}\n",
              (void*) curr, curr->item, (void*) curr->link);
   }
   printf ("NULL= %p\n", (void*) NULL);

   //
   // Free up all of the nodes.
   //
   while (head != NULL) {
      node_ref old = head;
      head = head->link;
      free (old);
   }

   //
   // Deliberately cause some memory leaks and throw away result.
   //
   for (int leaks = 0; leaks < 4; ++leaks) malloc (256);
   malloc (4096);

   return EXIT_SUCCESS;
}

/*
//TEST// valgrind --leak-check=full --log-file=numlist.lisval \
//TEST//./numlist >numlist.lisout 2>&1
//TEST// mkpspdf numlist.ps numlist.c* numlist.lis*
*/

```

```
@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@ mkc: starting numlist.c
numlist.c: $Id: numlist.c,v 1.1 2012-02-09 18:53:08-08 - - $
gcc -g -O0 -Wall -Wextra -std=gnu99 numlist.c -o numlist -lm
lint -Xa -fd -m -u -x -errchk=%all numlist.c

function returns value which is always ignored
    printf

function returns value which is sometimes ignored
    malloc
rm -f numlist.o
@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@ mkc: finished numlist.c
```

```
numlist: looping 10 times
&head= 0x7fefffda8
head= 0x4c30310
0x4c30310 -> struct node {item= 454433.423738244, link= 0x4c302c0}
0x4c302c0 -> struct node {item= 526750.279762108, link= 0x4c30270}
0x4c30270 -> struct node {item= 487217.223946828, link= 0x4c30220}
0x4c30220 -> struct node {item= 92297.6476986754, link= 0x4c301d0}
0x4c301d0 -> struct node {item= 91330.6121122943, link= 0x4c30180}
0x4c30180 -> struct node {item= 364602.248390607, link= 0x4c30130}
0x4c30130 -> struct node {item= 176642.642542916, link= 0x4c300e0}
0x4c300e0 -> struct node {item= 41631.0015946131, link= 0x4c30090}
0x4c30090 -> struct node {item= 985.394674650308, link= 0x4c30040}
0x4c30040 -> struct node {item= 3.90798504668055e-08, link= (nil)}
NULL= (nil)
```

```
==14439== Memcheck, a memory error detector
==14439== Copyright (C) 2002-2009, and GNU GPL’d, by Julian Seward et al.
==14439== Using Valgrind-3.5.0 and LibVEX; rerun with -h for copyright info
==14439== Command: ./numlist
==14439== Parent PID: 14438
==14439==
==14439==
==14439== HEAP SUMMARY:
==14439==     in use at exit: 5,120 bytes in 5 blocks
==14439==   total heap usage: 15 allocs, 10 frees, 5,280 bytes allocated
==14439==
==14439== 1,024 bytes in 4 blocks are definitely lost in loss record 1 of 2
==14439==    at 0x4A05E1C: malloc (vg_replace_malloc.c:195)
==14439==    by 0x40082D: main (numlist.c:81)
==14439==
==14439== 4,096 bytes in 1 blocks are definitely lost in loss record 2 of 2
==14439==    at 0x4A05E1C: malloc (vg_replace_malloc.c:195)
==14439==    by 0x400841: main (numlist.c:82)
==14439==
==14439== LEAK SUMMARY:
==14439==    definitely lost: 5,120 bytes in 5 blocks
==14439==    indirectly lost: 0 bytes in 0 blocks
==14439==      possibly lost: 0 bytes in 0 blocks
==14439==    still reachable: 0 bytes in 0 blocks
==14439==         suppressed: 0 bytes in 0 blocks
==14439==
==14439== For counts of detected and suppressed errors, rerun with: -v
==14439== ERROR SUMMARY: 2 errors from 2 contexts (suppressed: 4 from 4)
```