

```
1: // $Id: numlist.c,v 1.1 2012-02-09 18:53:08-08 - - $
2:
3: //
4: // Demo of how to use malloc and free.
5: //
6:
7: #include <assert.h>
8: #include <libgen.h>
9: #include <stdio.h>
10: #include <stdlib.h>
11:
12: //
13: // Declare the type of the handle, or pointer, to the struct.
14: // In Java, the same name is used for both the handle and the
15: // struct.
16: //
17: // Declare the type of the node. This is much like Java, except
18: // that the word 'struct' is used. C does not allow functions
19: // to be declared inside structs, as does Java.
20: //
21: typedef struct node *node_ref;
22: struct node {
23:     double item;
24:     node_ref link;
25: };
26:
27: //
28: // The main program allocates some nodes, pushes them onto a list,
29: // prints them out, and then frees up the nodes.
30: //
31: int main (int argc, char **argv) {
32:     char *progrname = basename (argv[0]);
33:
34:     //
35:     // Declare and set the head of the list to NULL.
36:     //
37:
38:     node_ref head = NULL;
39:
40:     //
41:     // Loop, pushing some random numbers onto the list. Note that
42:     // '->' in C means '.' in Java. Malloc(3c) is used to allocate
43:     // storage, like 'new' in Java. Always check with 'assert' that
44:     // malloc has actually returned the address of new memory.
45:     // 'sizeof' returns the number of bytes necessary for its
46:     // argument.
47:     //
48:     int max = argc < 2 ? 10 : atoi (argv[1]);
49:     printf ("%s: looping %d times\n", progrname, max);
50:     for (int count = 0; count < max; ++count) {
51:         node_ref new = malloc (sizeof (struct node));
52:         assert (new != NULL);
53:         new->item = drand48() * 1e6;
54:         new->link = head;
55:         head = new;
56:     }
```

```
57:
58:  //
59:  // Loop down the list, printing out each entry in debug mode.
60:  //
61:  printf ("%p\n", (void*) &head);
62:  printf ("head= %p\n", (void*) head);
63:  for (node_ref curr = head; curr != NULL; curr = curr->link) {
64:      printf ("%p -> struct node {item= %.15g, link= %p}\n",
65:              (void*) curr, curr->item, (void*) curr->link);
66:  }
67:  printf ("NULL= %p\n", (void*) NULL);
68:
69:  //
70:  // Free up all of the nodes.
71:  //
72:  while (head != NULL) {
73:      node_ref old = head;
74:      head = head->link;
75:      free (old);
76:  }
77:
78:  //
79:  // Deliberately cause some memory leaks and throw away result.
80:  //
81:  for (int leaks = 0; leaks < 4; ++leaks) malloc (256);
82:  malloc (4096);
83:
84:  return EXIT_SUCCESS;
85: }
86:
87: /*
88: //TEST// valgrind --leak-check=full --log-file=numlist.lisval \
89: //TEST// ./numlist >numlist.lisout 2>&1
90: //TEST// mkpspdf numlist.ps numlist.c* numlist.lis*
91: */
92:
```

[illegible]

```
1: numlist: looping 10 times
2: &head= 0x7feffffcd8
3: head= 0x4c23310
4: 0x4c23310 -> struct node {item= 454433.423738244, link= 0x4c232c0}
5: 0x4c232c0 -> struct node {item= 526750.279762108, link= 0x4c23270}
6: 0x4c23270 -> struct node {item= 487217.223946828, link= 0x4c23220}
7: 0x4c23220 -> struct node {item= 92297.6476986754, link= 0x4c231d0}
8: 0x4c231d0 -> struct node {item= 91330.6121122943, link= 0x4c23180}
9: 0x4c23180 -> struct node {item= 364602.248390607, link= 0x4c23130}
10: 0x4c23130 -> struct node {item= 176642.642542916, link= 0x4c230e0}
11: 0x4c230e0 -> struct node {item= 41631.0015946131, link= 0x4c23090}
12: 0x4c23090 -> struct node {item= 985.394674650308, link= 0x4c23040}
13: 0x4c23040 -> struct node {item= 3.90798504668055e-08, link= (nil)}
14: NULL= (nil)
```

```
1: ==2560== Memcheck, a memory error detector
2: ==2560== Copyright (C) 2002-2010, and GNU GPL'd, by Julian Seward et al.
3: ==2560== Using Valgrind-3.6.0 and LibVEX; rerun with -h for copyright info
4: ==2560== Command: ./numlist
5: ==2560== Parent PID: 2559
6: ==2560==
7: ==2560==
8: ==2560== HEAP SUMMARY:
9: ==2560==     in use at exit: 5,120 bytes in 5 blocks
10: ==2560==   total heap usage: 15 allocs, 10 frees, 5,280 bytes allocated
11: ==2560==
12: ==2560== 1,024 bytes in 4 blocks are definitely lost in loss record 1 of 2
13: ==2560==    at 0x4A05FDE: malloc (vg_replace_malloc.c:236)
14: ==2560==    by 0x400869: main (numlist.c:81)
15: ==2560==
16: ==2560== 4,096 bytes in 1 blocks are definitely lost in loss record 2 of 2
17: ==2560==    at 0x4A05FDE: malloc (vg_replace_malloc.c:236)
18: ==2560==    by 0x40087D: main (numlist.c:82)
19: ==2560==
20: ==2560== LEAK SUMMARY:
21: ==2560==     definitely lost: 5,120 bytes in 5 blocks
22: ==2560==     indirectly lost: 0 bytes in 0 blocks
23: ==2560==     possibly lost: 0 bytes in 0 blocks
24: ==2560==     still reachable: 0 bytes in 0 blocks
25: ==2560==     suppressed: 0 bytes in 0 blocks
26: ==2560==
27: ==2560== For counts of detected and suppressed errors, rerun with: -v
28: ==2560== ERROR SUMMARY: 2 errors from 2 contexts (suppressed: 6 from 6)
```