Princeton University
COS 217: Introduction to Programming Systems
Spring 2004 Midterm Exam Answers

Question 1

(a) The sizeof operator is used to determine the size of a data type or an object.
(b) size_t
(c) The malloc function dynamically allocates memory from the heap.
(d) void*
(e) p = malloc(sizeof(int));

Question 2

```
void PrintPercent(int x, int y)
{
   double percent;
   assert(y != 0);
   percent = 100.0 * (double)x / (double)y;
   printf("The result of %d/%d in percentage is %.2f%%.\n", x, y, percent);
}
```

Question 3 (a)

```
#include <stdio.h>
int main(void)
{
   int iChar;
   int iBlankCount = 0;
   int iTabCount = 0;
   int iNewlineCount = 0;
   while ((iChar = getchar()) != EOF)
   {
      if (iChar == ' ')
         iBlankCount++;
      else if (iChar == '\t')
         iTabCount++;
      else if (iChar == '\n')
         iNewlineCount++;
   }
   printf("%d %d %d\n", iBlankCount, iTabCount, iNewlineCount);
   return 0;
}
```

Question 3 (b)

Execute the program multiple times, with stdin redirected to a file that contains:

- no blanks, no tabs, and no newlines.
- no blanks, no tabs, and one newline.
- no blanks, no tabs, and multiple newlines.
- no blanks, one tab, and no newlines.
- no blanks, multiple tabs, and no newlines.
- one blank, no tabs, and no newlines.
- multiple blanks, no tabs, and no newlines.
- multiple blanks, multiple tabs, and multiple newlines.
- very long lines for each case.
- random characters.

In each case, redirect stdout to a file. When possible, use the UNIX diff command to compare each of those files to manually created files that contain the correct counts.

Question 4 (a)

```
#ifndef LIST_INCLUDED
#define LIST_INCLUDED

typedef struct List *List_T;

List_T List_new(int (*pfCompare)(const void*, const void*));
void List_free(List_T oList);
int List_insert(List_T oList, const void *pvItem);
int List_remove(List_T oList, const void *pvItem);
int List_hasCycle(List_T oList);

#endif

Alternative:  Make pfCompare a parameter to both List_insert and List_remove.
```

Question 4 (b) Using a Length Field

```
#include "list.h"
#include <stdlib.h>
#include <assert.h>

#define FALSE 0
#define TRUE 1

struct ListNode
{
   const void *pvItem;
   struct ListNode *psNextNode;
};

struct List
{
   struct ListNode *psFirstNode;
   int (*pfCompare)(const void*, const void*);
   int iLength;
};

/* List_insert and List_remove maintain the iLength field. */

int List_hasCycle(List_T oList)
{
   struct ListNode *psNode;
   int i;

   assert(oList != NULL);

   psNode = oList->psFirstNode;
   for (i = 0; i < oList->iLength; i++)
      psNode = psNode->psNextNode;

   if (psNode == NULL)
      return FALSE;

   return TRUE;
}
```

Question 4 (b) Using a Marking Algorithm

```
#include "list.h"
#include <stdlib.h>
#include <assert.h>

#define FALSE 0
#define TRUE 1

struct ListNode
{
   const void *pvItem;
   int iIsMarked;
   struct ListNode *psNextNode;
};

struct List
{
   struct ListNode *psFirstNode;
   int (*pfCompare)(const void*, const void*);
};

int List_hasCycle(List_T oList)
{
   struct ListNode *psNode;
   int iHasCycle = FALSE;

   assert(oList != NULL);

   /* Mark all nodes, while checking for marks. */
   psNode = oList->psFirstNode;
   while (psNode != NULL)
   {
      if (psNode->iIsMarked)
      {
         iHasCycle = TRUE;
         break;
      }
      psNode->iIsMarked = TRUE;
      psNode = psNode->psNextNode;
   }

   /* Unmark all nodes. */
   psNode = oList->psFirstNode;
   while (psNode != NULL)
   {
      if (! psNode->iIsMarked)
         break;
      psNode->iIsMarked = FALSE;
      psNode = psNode->psNextNode;
   }
   return iHasCycle;
}
```

Question 4 (b) Using Floyd’s Algorithm

```
#include "list.h"
#include <stdlib.h>
#include <assert.h>

#define FALSE 0
#define TRUE 1

struct ListNode
{
   const void *pvItem;
   struct ListNode *psNextNode;
};

struct List
{
   struct ListNode *psFirstNode;
   int (*pfCompare)(const void*, const void*);
};

int List_hasCycle(List_T oList)
{
   struct ListNode *psSlowNode;  /* Traverses oList one node at a time. */
   struct ListNode *psFastNode;  /* Traverses oList two nodes at a time. */

   assert(oList != NULL);

   psSlowNode = oList->psFirstNode;
   psFastNode = oList->psFirstNode;
   while (TRUE)
   {
      if (psFastNode == NULL) return FALSE;
      psFastNode = psFastNode->psNextNode;
      if (psFastNode == NULL) return FALSE;
      if (psFastNode == psSlowNode) return TRUE;
      psSlowNode = psSlowNode->psNextNode;
      psFastNode = psFastNode->psNextNode;
   }
}
```

Question 4 (c)

Test boundary conditions.

- Call each function with an empty List object.
- Call each function with a List object that contains one item.

Test all logical paths.

- Call the List_insert function with an item that already exists in the List object.
- Call the List_remove function with an item that does not exist in the List object.

Produce output that is known to be right/wrong.

- Read data from a text file, and insert it into a List object.
- Remove the data from the List object, and write it to a second text file
- Use the UNIX diff command to compare the text files.

Stress test.

- Call each function with a List object that contains many items.

Test for cycles.

- Call the List_hasCycle function with a List object that contains one item in a cycle.
- Call the List_hasCycle function with a List object that contains several items in a cycle.
- Call the List_hasCycle function with a List object that contains a large number of items in a cycle.
- Call the List_hasCycle function for all three cases above without a cycle.

Question 4 (d)

```
#include "list.h"
#include <stdlib.h>
#include <assert.h>

int compareDouble(const void *pv1, const void *pv2)
{
   double *pd1 = (double*)pv1;
   double *pd2 = (double*)pv2;
   assert(pv1 != NULL);
   assert(pv2 != NULL);
   if (*pd1 < *pd2) return -1;
   if (*pd1 > *pd2) return 1;
   return 0;
}

int main(void)
{
   double d = 1.1;
   List_T oList;
   int i;

   oList = List_new(compareDouble);
   i = List_insert(oList, &d);
   i = List_remove(oList, &d);
   i = List_hasCycle(oList);
   return 0;
}
```

Alternative: Supply compareDouble an actual parameter to both List_insert and List_remove.