Princeton University
COS 217: Introduction to Programming Systems
Spring 2003 Midterm Exam Answers

Question 1 Part (a)

```
struct employee
{
   char name[128];
   int *access_codes
   struct department *dept;
};
```

Question 1 Part (b)

```
struct department
{
   char name[128];
   struct employee **employees;
   int size;
};
```

Question 1 Part (c)

```
#ifndef DEPARTMENT_H
#define DEPARTMENT_H

#include "employee.h"  /* optional */

typedef struct department *Department;

Department Department_new(void);
/* Return a new Department. */

void Department_free(Department dept);
/* Free dept. */

void Department_addEmployee(Department dept, struct Employee *emp);
/* Add emp to dept.  It is a checked runtime error for dept or emp
   to be NULL. */

void Department_removeEmployee(Department dept, struct Employee *emp);
/* Remove emp from dept.  It is a checked runtime error for dept or
   emp to be NULL. */

void Department_map(Department dept,
   void (*func)(struct Employee *emp, void *data),
   void *data);
/* Call (*func)(emp, data) for each Employee emp in dept.  It is a
   checked runtime error for dept or func to be NULL. */

#endif
```

Question 1 Part (d)

```
#include department.h”

void Department_map(Department dept,
   void (*func)(struct Employee *emp, void *data),
   void *data)

/* Call (*func)(emp, data) for each Employee emp in dept.  It is a
   checked runtime error for dept or func to be NULL. */

{
   int i;

   assert(dept != NULL);
   assert(func != NULL);

   for (i = 0; i < dept->size; i++)
   {
      struct employee *emp = dept->employees[i];
      assert(emp != NULL);
      (*func)(emp, data);
   }
}
```

Question 2

```
   === main.c ===

      int a = 3;  /* file, external, process */
      int b = 4;  /* file, external, process */

      extern void f(int b);

      int main()
      {
        int a;   /* block, internal, temporary */
        for (a = 0; a < 4; a++)
          f(a);
      }

   === f.c ===

      #include <stdio.h>

      extern int a;
      static int b;  /* file, internal, process */

      void f(int b)  /* block, internal, temporary */
      {
        int c = 0;  /* block, internal, temporary */
        static int d = 0;  /* block, internal, process */

        printf("%d %d %d %d\n", a, b, c, d);
```

```
        a++;
        b++;
        c++;
        d++;
      }
```

Program output:

```
3 0 0 0
4 1 0 1
5 2 0 2
6 3 0 3
```

Question 3 Part (a)

```
#define MAX_BLOCKS 1000

char *pcFirstBlock = NULL;
char *pcFirstFreeBlock = NULL;
int iBlocksAllocatedCount = 0;

void *malloc16(void)

/* Allocate a block of 16 bytes of memory, and return the address of
   that memory block. */

{
   char *pc;

   /* If the memory pool has not been allocated, then allocate it. */

   if (pcFirstBlock == NULL)
      pcFirstBlock = (char*)malloc(16 * MAX_BLOCKS);

   /* If the free list contains some memory blocks, then remove the
      first one and return its address. */

   if (pcFirstFreeBlock != NULL)
   {
      pc = pcFirstFreeBlock;
      pcFirstFreeBlock = *(char**)pcFirstFreeBlock;
      return (void*)pc;
   }

   /* If some blocks have never been allocated, then return the address
      of the first one. */

   if (iBlocksAllocatedCount < MAX_BLOCKS)
   {
      pc = pcFirstBlock + (16 * iBlocksAllocatedCount);
      iBlocksAllocatedCount++;
      return pc;
   }

   /* Return NULL to indicate failure to satisfy the client's request
```

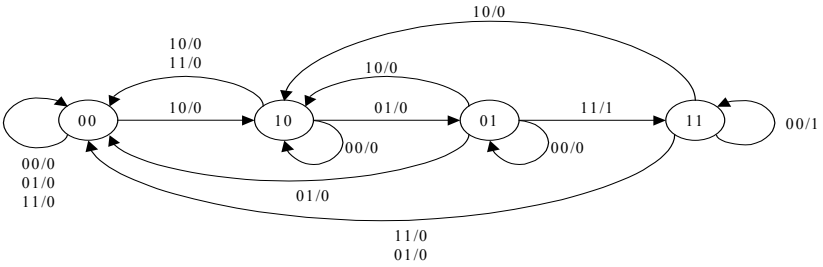
```
      for a memory block. */

   return NULL;
}

void free16(void *ptr)

/* Free the block of 16 bytes of memory whose address is ptr. */

{
   char **ppcNewFirstFreeBlock;
   assert(ptr != NULL);

   /* Insert the given memory block onto the front of the free list. */

   ppcNewFirstFreeBlock = (char**)ptr;
   *ppcNewFirstFreeBlock = pcFirstFreeBlock;
   pcFirstFreeBlock = (char*)ptr;
}
```

Question 3 Part (b)

Speed. Whereas malloc must search for a free block of the right size, malloc16 can simply return a pointer from the front of its list.

Fragmentation. malloc allocates blocks of all sizes and can have holes of unused bytes. malloc16 avoids this problem by allocating blocks of the same size and never breaking them up.

Question 4 Part (a)



Question 4 Part (b)

i0	i1	Q0	Q1		next Q0	next Q1	output
0	0	0	0		0	0	0
0	0	0	1		0	1	0
0	0	1	0		1	0	0
0	0	1	1		1	1	1
0	1	0	0		0	0	0
0	1	0	1		0	0	0
0	1	1	0		1	1	0
0	1	1	1		0	0	0
1	0	0	0		1	0	0
1	0	0	1		1	0	0
1	0	1	0		1	0	0
1	0	1	1		1	0	0
1	1	0	0		0	0	0
1	1	0	1		1	1	1
1	1	1	0		0	0	0
1	1	1	1		0	0	0

Question 4 Part (c)

(See separate document spr03exam1ansq4c.pdf, scanned from hardcopy.)

Question 5 Part (a)

Preprocessing time
Compile time
Link time
Run-time programming
Run-time user
Run-time exception

Question 5 Part (b)

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

void PrintFile(void)
{
   FILE *fp = NULL;
   char filename[16];
   char *buffer;
   size_t uiFilenameLength;

   /* Use fgets instead of gets to eliminate the possibility
```

```
      of corrupting memory beyond the end of the filename array. */

   if (fgets(filename, 16, stdin) == NULL)
   {
      /* Runtime user error */
      fprintf(stderr, "Unable to read stdin\n");
      exit(1);
   }

   uiFilenameLength = strlen(filename);
   if (filename[uiFilenameLength - 1] != '\n')
   {
      /* Runtime user error */
      fprintf(stderr, "File name too long\n");
      exit(1);
   }

   /* Remove the newline character from the end of filename. */
   filename[uiFilenameLength - 1] = '\0';

   if (strlen(filename) == 0)
   {
      /* Runtime user error */
      fprintf(stderr, "Empty filename\n");
      exit(1);
   }

   fp = fopen(filename, "r");
   if (fp == NULL)
   {
      /* Runtime user error */
      fprintf(stderr, "Unable to open file: %s\n", filename);
      exit(1);
   }

   buffer = (char*)malloc(16);

   /* Runtime programming error */
   assert(buffer != NULL);

   /* Use fgets instead of gets to eliminate the possibility
      of corrupting memory beyond the end of the buffer array. */

   while (fgets(buffer, 16, fp) != NULL)
      printf(buffer);

   /* Should free buffer only once. */
   free(buffer);

   fclose(fp);
}
```