Princeton University
COS 217: Introduction to Programming Systems
Fall 2005 Midterm Exam Answers

The exam was a 50 minute open-book open-notes exam.

Question 1a

738

That is, the characters '7', '3', and '8' followed by a newline character.

Question 1b

The function prints to stdout the decimal digits that comprise n, in reverse order.

Question 1c

Given the unsigned integer 0, the first function prints the character '0' followed by a newline character. The second function prints only a newline character.

Question 2a

ykcul gnileef

That is, the string "feeling lucky" backwards, followed by a newline character.

Question 2b

If p and s were declared to be of type const char*, then they would be pointers to constants. As such, f()'s attempts to change the contents of memory referenced by p or s would yield compiletime errors.

Question 2c

The function reverses string s.

Question 3a

1, 2

That is, a '1', a comma, a space, a '2', and a newline character.

Question 3b

f is a pointer to an array of 25 floats.

Question 3c

foo is an array of 7 pointers to functions that take an integer pointer and return a float pointer.

Question 3d

The **heap** stores dynamically allocated memory, that is, memory allocated via calls to malloc() and similar functions.

The **stack** stores data that have temporary duration, that is, the values of non-static local variables and formal parameters. (It also stores function return addresses and other data, as we will see in the second half of the course.)

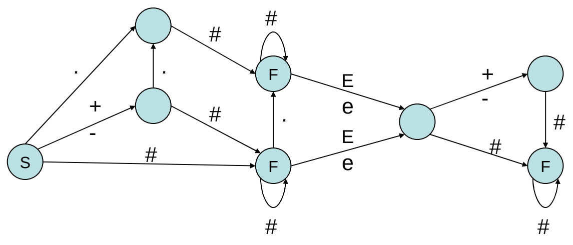
The **data section** stores programmer-initialized data that have process duration, that is, the values of global variables and static local variables that the programmer has explicitly initialized.

The **bss section** stores programmer-uninitialized data that have process duration, that is, the values of global variables and static local variables that the programmer has not explicitly initialized. The bss section is initialized to zeros.

Question 3e

If the operating system were to grow the heap and the stack in the same direction, then it would need to map them to distinct regions of memory. But using distinct regions of memory would incur the risk of the heap region becoming exhausted while the stack region still has unused space, or vice-versa. So, for maximum flexibility, the operating system maps the heap and the stack to the same region of memory, starts the heap at the top of the region and the stack at the bottom of the region, and grows them toward the common unused center.

Question 4



Question 5a

The item field is defined as type void* so a Queue object can store pointers to data of any type. A Queue object thus defined is generic.

Question 5b

Queue_t is defined in queue.c rather than in queue.h to enforce encapsulation. With encapsulation, clients cannot manipulate a Queue object's data directly. Instead, clients can manipulate a Queue object's data only by calling the functions, declared in the Queue interface, that encapsulate the data. Such encapsulation has several advantages; notably, it allows a programmer to change the Queue implementation without impacting clients.

Question 5c

```
int Queue_empty(Queue_T queue)
{
   assert(queue != NULL);
   return queue->head == NULL;
}
```

Question 5d

```
void Queue_add(Queue_T queue, void *item)
{
   struct list *newnode;

   assert(queue != NULL);

   newnode = (struct list*)malloc(sizeof(*newnode));
   assert(newnode != NULL);
   newnode->item = item;
   newnode->next = NULL;
   if (queue->tail == NULL)
      queue->head = newnode;
   else
      queue->tail->next = newnode;
   queue->tail = newnode;
}
```

Question 5e

```
void *Queue_remove(Queue_T queue)
{
   struct list *oldhead;
   void *olditem;

   assert(queue != NULL);
   assert(queue->head != NULL);

   oldhead = queue->head;
   olditem = oldhead->item;
   queue->head = oldhead->next;
   free(oldhead);
   if (queue->head == NULL)
      queue->tail = NULL;
   return olditem;
}
```

Alternative answer:

```
void *Queue_remove(Queue_T queue)
{
   struct list *oldhead;
   void *olditem;

   assert(queue != NULL);

   if (queue->head == NULL)
      return NULL;

   oldhead = queue->head;
   olditem = oldhead->item;
   queue->head = oldhead->next;
   free(oldhead);
   if (queue->head == NULL)
      queue->tail = NULL;
   return olditem;
}
```

Question 6

These are the three known answers:

```
int i, n=20;
for (i=0; i < n; n--)
   printf("-");

int i, n=20;
for (i=0; i + n; i--)
   printf("-");

int i, n=20;
for (i=0; -i < n; i--)
   printf("-");
```