Princeton University
COS 217: Introduction to Programming Systems
Spring 2002 Midterm Exam 1 Answers

Question 1

```
a) 8
b) 4
c) 68
d) 68
e) 8
f) 16
g) 16
h) 8
i) 4
j) 1
k) 4
l) 8
```

Question 2

a) An opaque pointer allows an ADT to hide its data structures from clients.

b) C's **structure declarations** allow the ADT designer to inform the compiler of the existence of a structure without revealing the structure's definition (i.e. its fields). C's **typedef** construct allows the ADT designer to (more or less) hide from clients the fact that the data structure is implemented as a structure.

Question 3

a) A **declaration** gives the compiler **partial** information about a construct (a variable, a function, a structure, etc.). A **definition** gives the compiler **all** information about a construct. For a variable, a definition causes the compiler to allocate memory.

b) C has declarations to facilitate modularity by allowing a separation between interfaces and implementations. Using declarations, a module designer can hide implementation details from the module's clients.

Question 4

a) OK

b) Error

```
float a[] = { 1.0, 0.0, 1.0, 0.0 };
float *b = &a[2];
```

b points to memory that contains 1.0.

```
float *c = b-- + 1;
```

The expression b-- decrements b so it points to memory that contains 0.0. But it evaluates to the prior value of b, which points to memory that contains 1.0. The expression b-- + 1 thus evaluates to the address of memory that contains 0.0. That address is assigned to c.

```
float result = *b / *c;
```

*b evaluates to 0.0. *c evaluates to 0.0. Thus the expression is identical to (0.0 / 0.0). It evaluates to NaN (not a number).

c) Error

```
void *my_alloc(void *ptr, int size)
{
   if (ptr) return realloc(ptr, size);
   else return malloc(size);
}

void my_free(void *ptr)
{
   free(ptr);
   ptr = 0;
}

int main()
{
   void *ptr = my_alloc(0, 4);
```

Calls malloc to allocate 4 bytes of memory, and assigns the address of that memory to ptr.

```
   my_free(ptr);
```

Frees that memory, but does not change the value of ptr. ptr still points to the freed memory.

```
   ptr = my_alloc(ptr, 4);
```

Calls realloc to change the size of the block of memory to which ptr points. But that memory has been freed. The results are unpredictable.

```
   my_free(ptr);

   return 0;
}
```

Question 5

name	Global. Data section.
argc	Stack.
argv	Stack.
*argv	Unknown.
**argv	Unknown.

ext	Stack.
‘d’	Stack.
found	Stack.
“-name”	Global. Rodata section.
*name	Heap.
*ext	Stack.
sizeof(ext)	Evaluated at compiletime to 5. Will probably reside in a machine language instruction in the text section.

Question 6

a) Error

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

char *MakeFilename(char *name, char *ext)
{
   char buffer[BUFFERSIZE];   <- Assume this is big enough
   strcpy(buffer, name);
   strcat(buffer, ext);
   return buffer;
}

int main(int argc, char **argv)
{
   if (argc > 2) {
      char *filename = MakeFilename(*argv[1], *argv[2]);
```

I believe there is a small error in the test question: the actual parameters sent to MakeFilename should be argv[1] and argv[2], and not *argv[1] and *argv[2]. Assuming that is the case...

MakeFilename builds its result in buffer, and then returns buffer. But buffer is on the stack, and so ceases to exist after MakeFilename is finished executing. Thus filename becomes a dangling pointer.

```
      printf(“%s\n”, filename);
```

The string that is printed is unpredictable.

```
   }
}
```

b) Error

```
#include <stdio.h>

int main(int argc, char **argv)
{
   if (argc > 1) {
      int n = atoi(argv[1]);
      if (n = 0) printf(“Infinity\n”);
```

The programmer probably meant to write (n == 0). The expression (n = 0) assigns 0 to n, and then evaluates to 0. The "if" statement interprets 0 as FALSE. Thus the call to printf on the next line will always be executed.

```
        else printf("%f\n", 1.0 / n);
```

At this point the value of n is certainly 0. Thus the expression (1.0 / n) is identical to (1.0 / 0). That is a mixed-type expression, so the 0 will automatically be promoted to type double. Thus the expression is identical to (1.0 / 0.0) – a floating point division by 0. The result will be printed as Inf (infinity).

```
    }
}
```

c) OK. Prints 0.000000.

Question 7

```
int List_remove(List_T oList, void *pvData)
{
   struct Node *psNode;
   struct Node *psMarkerNode;

   assert(oList != NULL);
   assert(oList->iLength > 0);

   psMarkerNode = oList->psMarkerNode;
   psNode = psMarkerNode->psNextNode;

   while (psNode != psMarkerNode)
   {
      if ((*(oList->pfCompare))(psNode->pvItem, pvData) == 0)
      {
         psNode->psNextNode->psPrevNode = psNode->psPrevNode;
         psNode->psPrevNode->psNextNode = psNode->psNextNode;
         free(psNode);
         oList->iLength--;
         return 1;
      }
      psNode = psNode->psNextNode;
   }
   return 0;
}
```

Question 8

```
#define SPECIAL_VALUE '~'

char *String_malloc(int iSize)
{
   char *pc;
   assert(iSize > 0);

   pc = (char*)malloc(iSize + sizeof(char));
   *pc = SPECIAL_VALUE;
   return pc + sizeof(char);
}
```

```
void String_free(char *pcString)
{
   char *pc;

   if (pcString == NULL)
      return;

   pc = pcString - sizeof(char);
   assert(*pc == SPECIAL_VALUE);
   free(pc);
}
```

Question 9

Test incrementally.

> Test each function as it is created. A reasonable order is (Set_new), Set_put, Set_getLength, Set_getKey, Set_getValue, Set_remove, Set_clear, (Set_free).

Test boundary conditions.

> Test all functions when oSet is NULL.
> Test all functions when oSet contains no bindings.
> Test Set_put, Set_remove, Set_getKey, and Set_getValue when pvKey is NULL.
> Test Set_put when pvKey already exists in oSet.
> Test Set_remove, Set_getKey, and Set_getValue when pvKey does not exist in oSet.

Test as much code as possible.

> Examine each conditional statement (if, switch, for, while, do...while) in each function. Pass data to the function that executes each path through each conditional statement.

Produce output that is known to be right/wrong.

> Accumulate function calls into a driver program. Check the value returned by each function call against the expected result, perhaps using the assert macro.

Stress test.

> Test Set_put, Set_remove, Set_getKey, and Set_getValue when pvKey is a very long string.
> Test all functions when oSet contains many bindings.