Problem 1

You were to represent 244 in different ways. Here were the answers.

part	*version A*
a	244 = 128 + 64 + 32 + 16 + 4 = 11110100 base 2
b	Bit 7 is on, so it's a negative number. To find out which, we compute its negative by complementing the bits and adding 1: –n = 00001011 + 1 = 00001100 = 12, so n = -12
c	244 = 15 * 16 + 4 = F4 base 16.
d	244 = 243 + 1 = 100001 base 3

Problem 2

You were to implement a function named strchr that, given arguments char *s and char c, returns a pointer to the first occurrence of c in the string pointed to by s. If c does not occur in the string, strchr returns NULL. The terminating null character is considered part of the string; therefore if c is '\0', strchr locates the terminating '\0'. You were told at the exam not to use other functions declared in string.h. Here are two solutions.

```
char *strchr (char *s, char c) {
   char *p = s;
   while (*p) {
      if (*p == c) return p;
      p++;
   }
   if (*p == c) return p;
   return NULL;
}
```

```
char *strchr (char *s, char c) {
   char *p;                    // could just use s
   for (p=s; *p; p++)  if (*p == c) return p;
   if (*p == c) return p;
   return NULL;
}
```

Problem 3

Each part of this problem involved completing the replace function described below. Given an array of strings named table as argument along with an index k into the table and a new string s, replace essentially performs the assignment

```
table[k] = a copy of s;
```

Version B of this problem differed only in the parameter sequence (the second and third parameters were exchanged). Presented below are answers for version A.

Part a was to complete the replace function using array notation, i.e. square brackets, where possible.

```
void replace ( char *table [ ], int k, char s [ ] ) {
   table[k] = (char *) malloc (sizeof(char) * (strlen(s)+1));
   strcpy (table[k], s);
}
```

The reason that char table[ ] [ ] isn't allowed in the function header is that the declaration of any n-dimensional array must include the sizes of the last n–1 dimensions so that the algorithm to access elements can be determined. (We didn't expect you to know this.)

Part b was to complete the replace function, *without using* any array notation (square brackets).

```
void replace (char **table, int k, char *s) {
   *(table+k) = (char *) malloc (sizeof(char) * (strlen(s)+1));
   strcpy (*(table+k), s);
}
```

Problem 4

In part a, you were to provide the type declaration for a struct node that contains an int named val and pointer named ptr to a value of type struct node. Here's the declaration.

```
struct node {
   int val;
   struct node * ptr;
};
```

Part b was to write a function min that returns the minimum value in its argument list. You were allowed to assume that the argument list was not null and noncircular, and that the last node in the list stored a null ptr pointer. Here's an iterative solution:

```
int min (struct node *p) {
   int rtn;
   if  (p->ptr == NULL) {     /* 1-element list */
      return p->val;
   }
   for (rtn = p->val; p; p = p->ptr) {
      if (p->val < rtn) rtn = p->val;
   }
   return rtn;
}
```

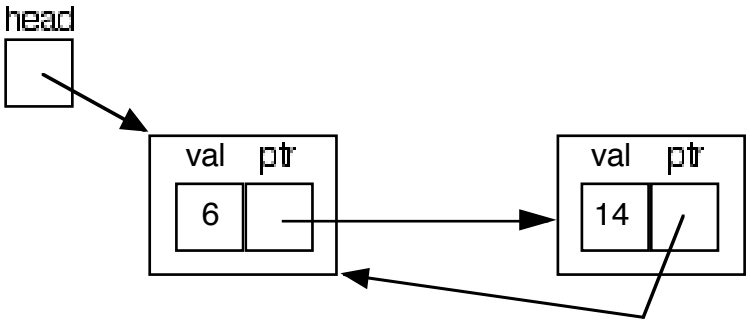
Here's a recursive solution.

```
int min (struct node *p) {
   return helper (p->ptr, p->val);
}

int helper (struct node *p, int soFar) {
   if (p == NULL) {
      return soFar;
   } else if (p->val < soFar) {
      return helper (p->ptr, p->val);
   } else {
      return helper (p->ptr, soFar);
```

```
   }
}
```

In part c, you were to use your declaration from part a to provide a program segment that dynamically allocates the structure represented in the diagram below.



Here's the code.

```
struct node *p1;
struct node *head;
p1 = (struct node *) malloc (sizeof (struct node));
head = (struct node *) malloc (sizeof (struct node));
p1->ptr = head;
p1->val = 14;
head->ptr = p1;
head->val = 6;
```

Finally, you were to explain in part d what would happen if head in the structure in part c was passed as argument to the function you wrote for part b. An infinite loop would result, since none of the ptr fields are null.

Problem 5

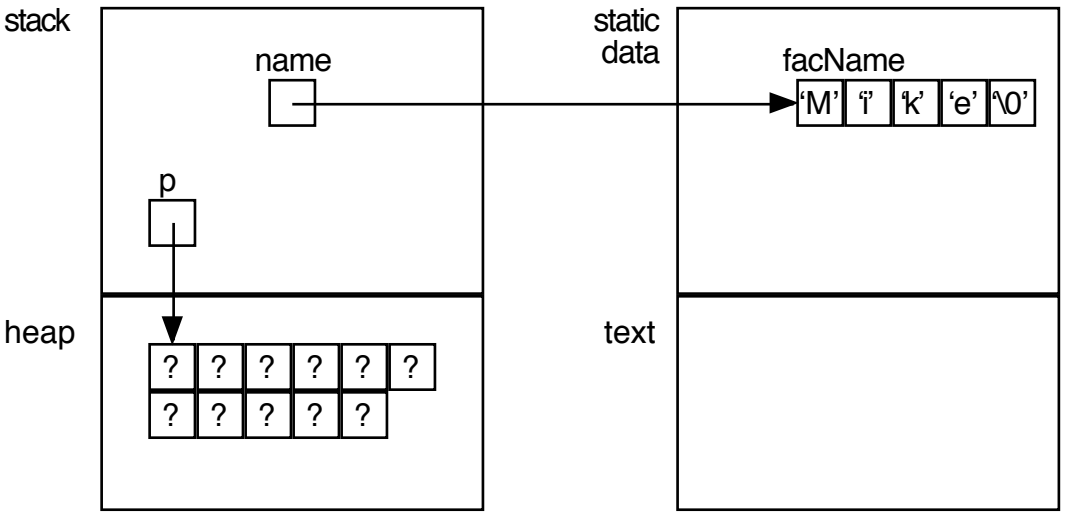
Both parts of this problem involved the following code.

```
char facName [ ] = "Mike";

char *response (char *name) {
   int k, helloLen;
   char *p;
   helloLen = strlen ("Hello, ");
   p = (char *) malloc (sizeof(char) * (helloLen + strlen (name)));
   p = "Hello, ";
   for (k=0; k<=strlen (name); k++) {
      p[k+helloLen] = name[k];
   }
}

int main ( ) {
   printf ("%s\n", response (facName));
   return 0;
}
```

In part a, you were to indicate (using box-and-pointer diagrams) the location and contents of the variables facName, name, and p. Your answer for p was to reflect its state immediately after the call to malloc. If the contents of a variable could not be determined, you were to put a question mark in the corresponding box. Here's a solution.



Part b involved debugging the response function and explaining your fixes. The code contains three bugs:

- It doesn't return anything; the last statement should be

```
return p;
```

- Space is allocated for the characters in the two strings but not for the terminating null; the `malloc` statement should be

```
p = malloc (1 + helloLen + strlen(name));
```

- Constant string space is overwritten by the `for` loop; in fact, the second assignment statement to `p` completely undoes the `malloc`, orphaning the allocated storage. It should be rewritten to use `strcpy`:

```
strcpy (p, "Hello, ");
```