CS240 Programming in C

Midterm Exam 2,
March 30, 2011

This exam is closed book, closed notes. Please do not open the exam until you are told to do so. There are 7 pages in this booklet. Please make sure that you have them all. Extra blank pages are provided in the back.

Name: _____

Section: _____

9:30 am - 11:20 am F	F930
11:30 am - 1:20 pm F	F1130
1:30 pm - 3:20 pm F	F130
3:30 pm - 5:20 pm F	F330
9:30 am - 11:20 am R	R930
11:30 am - 1:20 pm R	R1130
3:30 pm - 5:20 pm R	R330
11:30 am - 1:20 pm T	T1130

Question 1. (12 points)
Write the output of the following program.

```
#include <stdio.h>

int main()
{
   int array[10] = {10, 11, 12, 13, 14, 15, 16, 17, 18, 19};
   int *iptr = (int *)&array;
   void *vptr = &array;

   iptr = iptr + 8;
   vptr = ((char *)vptr) + 8;

   printf("%d\n", *iptr);
   printf("%d\n", *(int *)vptr);

   iptr = (int *) (((double *)iptr) - 2);
   vptr = ((int *)vptr) - 2;

   printf("%d\n", *iptr);
   printf("%d\n", *(int *)vptr);
}
```

Answer:

```
18
12
14
10
```

Question 2. (14 points)
Consider the following definitions:

```
struct intarray {
  int size;
  int *data;
};

struct intarray * create_intarray(int size) {
  struct intarray *a;
  a = malloc(sizeof(struct intarray));
  a->size = size;
  a->data = malloc(sizeof(int)*size);
  return a;
}
```

Rewrite the above function using just one call to malloc:

```
struct intarray * create_intarray(int size) {
  struct intarray *a;
  a = ____________________;
  a->size = size;
  a->data = ________________;
  return a;
}
```

Answer:

```
struct intarray *
create_intarray(int size)
{
struct intarray *a;
a = malloc(sizeof(struct intarray)+sizeof(int)*size);
a->size = size;
a->data = a+1;
// the following also works
// a->data = ((char *)a+sizeof(struct intarray));
return a;
}
```

Question 3. (16 points)
What gets printed by the following? Write '?' if it would print garbage.

```
#include <stdio.h>

int h1(int c) { return c + 1; }

int h2(int c) { return c - 1; }

int g1(int a, int b, int(*h)(int)) { return a + h(b); }

int g2(int a, int b, int(*h)(int)) { return a * h(b); }

void f1(int *arg1, int *arg2, int(*g)(int, int, int(*h)(int)))
{
  int (*h)(int);
  if (*arg1 % 2) { h = &h1; }
    else { h = &h2; }
  *arg1 += g(*arg1, *arg2, h);
}

void f2(int *arg1, int *arg2, int(*g)(int, int, int(*h)(int)))
{
  int (*h)(int);
  if (*arg1 % 2) { h = &h1; }
    else { h = &h2; }
  *arg1 *= g(*arg1, *arg2, h);
}

int main()
{
  int x, y;

  x = 5, y = 6;
  f1(&x, &y, &g1);
  printf("x = %d\n", x);

  x = 4, y = 5;
  f1(&x, &y, &g2);
  printf("x = %d\n", x);

  x = 3, y = 4;
  f2(&x, &y, &g1);
  printf("x = %d\n", x);

  x = 2, y = 3;
  f2(&x, &y, &g2);
  printf("x = %d\n", x);
}
```

Answer:

```
x = 17
x = 20
x = 24
x = 8
```

Question 4. (16 points)
The function pack shown below takes a linked list as an argument (whose nodes are potentially allocated at arbitrary points in heap memory), and returns an identical linked list where all list nodes are adjacent in heap memory. Please fill in the missing lines.

```
struct list {
  int data;
  struct list *next;
}

struct list *pack(struct list *x) {

  int size;
  struct list *y;

  for (size = 0, y = x; y; size++, y = y->next);

  // Allocate storage for the resulting list

  y = malloc(_______________________________________________);

  for (i = 0; i < size; i++, x = x->next){
    // copy data from the original

    y[i].data = ___________________________________________

    // set the pointer to the next element to point to
    // the next adjacent location

    y[i].next = ___________________________________________
  }

  // terminate the pointer chain

  y[_________________________].next = NULL;

  return y;
}
```

Answer:

```
struct list *
pack(struct list *x)
{
int size;
struct list *y;

for (size = 0, y = x; y; size++, y = y->next);
y = malloc(size*sizeof(struct list));
for (i = 0; i < size; i++, x = x->next){
y[i].data = x->data;
y[i].next = &y[i+1];
}
y[size-1].next = NULL;
return y;
}
```

Question 5. (12 points)
What is the output of the following program?

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

struct sample{
  char name1[21];
  char name2[21];
  char name3[21];
  char name4[21];
};

int main(){
  struct sample * p = malloc(sizeof(struct sample)*2);
  struct sample *q;

  strcpy(p->name1,"c");
  strcpy(p->name2,"language");
  strcpy(p->name3,"is");
  strcpy(p->name4,"wonderful");


  strcpy((p+1)->name1,"no other");
  strcpy((p+1)->name2,"language is");
  strcpy((p+1)->name3,"like");
  strcpy((p+1)->name4,"c");

  q = (p+(strlen(p->name2)+1)%3+1);
  printf("%s %s %s\n",q->name1, q->name2, p->name4);

  return 0;
}
```

Answer:

```
no other language is wonderful
```

Question 6. (14 points)
Please complete the lookup function. It should print all recipes containing the ingredient passed to the function.

```
/* unit_type of an ingredient */
#define TBSPS 1
#define CUPS 2

typedef union {
  int tbsps;
  float cups;
} amount;

typedef struct {
  char name[30];
  int unit_type;
  amount how_much;
} ingredient;

typedef struct {
  char name[30];
  int numIngreds;
  ingredient ingreds[10];
} recipe;

void lookup(recipe * book, int numRecipes, char* targetIngredient)
{
  int i, j, k;
  for(i = 0; i < numRecipes; i++) {
   for(j=0; j < book[i].numIngreds; j++) {

     // compare the target ingredient with the name of the current ingredient

     if(_______________________________) {

       printf("%s recipe:\n", book[i].name);
       for(k=0; k < book[i].numIngreds; k++) {

         // print the appropriate amount based on unit_type

         if(_______________________________)

           printf("%d tbsps %s\n", ____________________________________)

         else

           printf("%f cups %s\n",  ____________________________________)
       }
       printf("\n");
    }
   }
 }
}
```

Answer:

```
void lookup(recipe * book, int numRecipes, char* targetIngredient)
{
```

```
    int i, j, k;
    for(i = 0; i < numRecipes; i++)
    {
      for(j=0; j < book[i].numIngreds; j++)
      {
        if(strcmp(book[i].ingreds[j].name, targetIngredient) == 0)
        {
           printf("%s recipe:\n", book[i].name);
           for(k=0; k < book[i].numIngreds; k++)
           {
             if(book[i].ingreds[k].unit_type == TBSPS)
               printf("%d tbsps %s\n", book[i].ingreds[k].how_much.tbsps,
                                        book[i].ingreds[k].name);
             else
               printf("%f cups %s\n", book[i].ingreds[k].how_much.cups,
                                       book[i].ingreds[k].name);
           }
     printf("\n");
         }
      }
   }
}
```

Question 7. (16 points)
A *palindrome* is a word that can be read the same way forwards or backwards (e.g., `eye`, `racecar`, `deed`, and `level`) without consideration of spaces, special characters, or case sensitivity. Fill in the missing portions of the following program that determines whether its input string is a palindrome.

```
#include <stdio.h>

int main(int argc, char **argv) {

 char * str;
 int i, j;

 if (argc!=2) {
     printf("Usage: ispalindrome string\n");
     exit(1);
 }

 str = argv[1];

 // Convert all characters to lower case
 for (i=0; i<strlen(str); i++) {
   if (str[i]>='A' && str[i] <= 'Z') {
      // It is an upper case character. Convert it to lower case

      ------------------------------------------------
   }
 }

 // Verify if it is a palindrome
 i = 0;
 j = strlen(str)-1;

 // Check that the characters at the begining and the end match until i and j meet
 while (_________________________________) {

   // Advance i, skipping spaces and special characters
   while (_______________________________________) {
      i++;
    }

   // Advance j, skipping spaces and special characters
   while (_______________________________________) {
      j--;
    }

   // Check that the characters at i and j match.
   if (___________________________________) {

     // Characters do not match
     printf("No\n");  exit(1);
    }

   // advance i and j
    ------------------------;
    ------------------------;
 }

 // Characters match
 printf("Yes\n"); exit(0);
}
```

Answer:

```
#include <stdio.h>

int main(int argc, char **argv) {

char * str;
int i, j;

if (argc!=2) {
printf("Usage: ispalindrome string\n");
exit(1);
}

str = argv[1];

// Convert all characters to lower case
for (i=0; i<strlen(str); i++) {
if (str[i]>='A' && str[i] <= 'Z') {
// It is a upper case character. Convert it to upper case
str[i]=str[i]-'A'+'a';
}
}

// Verify if it is a palindrome
i = 0;
j = strlen(str)-1;

// Check the characters at the begining and the end match until i and j meet
while (i<j) {
// Advance i skipping spaces and special characters
while (i<j && (str[i] < 'a' || str[i] > 'z')) {
i++;
}

// Advance j skipping spaces and special characters
while (i<j && (str[j] < 'a' || str[j] > 'z') ) {
j--;
}

// Check that character at i and j match.
if (str[i]!=str[j]) {
// Characters do not match
printf("No\n");
exit(1);
}
i++;
j--;
}

// Characters match
printf("Yes\n");
exit(0);
}
```