C Homework

CS032 2008

Due 3/17 at 2:20PM

All homeworks are due at 3/17 at 2:20PM in the CS 32 bin on the second floor. No late homeworks are accepted.
Please include your login name on each piece of paper you hand in, and please staple your pages together before handing in.

Before attempting this homework you are highly encouraged to understand the material the basics of C. This mini course, blatantly stolen from cs 123, might be useful:
http://www.cs.brown.edu/courses/cs123/resources/c_mini_course.pdf

1 Header Files vs. C Files [10 pts]

a. [5 pts] When should you use header guards?

b. [5 pts] When should you put C function definitions in header files? A C function definition is the body of the function.

2 Referential Loop [10 pts]

Insert any necessary forward declarations into the code below and explain why you cannot simply include "type2.h" in the header file type1.h.

```
//1st header file
//type1.h
#ifndef TYPE1
#define TYPE1

typedef struct type1 {
        type2_t *myType2;
        int myInt;
} type1_t;

#endif
```

```
//2nd header file
//type2.h
#ifndef TYPE2
#define TYPE2

#include "type1.h"

typedef struct type2 {
        type1_t *myType1;
        int myInt;
} type2_t;

#endif
```

3 Pointer Basics [5 pts]

Fix the following code segment so that subsequently printing *x will print 3. Explain the error you fixed and how your change fixed it.

```
        int* x;
        int y = 10;
        x = y;
        y = 3;
```

4 Malloc and Free [15 pts]

a. [10 pts] Write C code to malloc ten integers using just one malloc statement. Set the integers to be 0 through 9. Next, print those values. Then free them using just one free statement. Warning: you should cast the void * that malloc produces.

b. [5 pts] Fix the following code segment so that subsequently printing *x will print 10. Explain how your change fixed it.

```
int* x;
int y = 10;
x = y;
y = 3;
```

5 Strings [20 pts]

In java, strings are immutable. When you set one to another or make any changes, you are creating a copy. In C a string is just an arrays of characters and you deal with a pointer to the start of the string. This makes working with them a little more difficult. The str(...) functions and the man pages are your friend. Try entering "man strcpy" in a terminal.

a. [5 pts] Given properly allocated char* string1, char* string2, edit the following block of code so will work like it would in Java. Don't think too hard.

```
//C                              //Java
string1 = string2;               //string1 = string2
printf("%s\n", string1);         //System.out.println(string1);
printf("%s\n", string2);         //System.out.println(string2);
```

b. [5 pts] What data can be modified and what data cannot be modified after the following declaration:

```
const char** temp;
```

c. [10 pts] Fix both of the bugs in the following listing. One of them is in the first line. The other might not be obvious when you run simple code like this, but it does exist!

```
char[5] string;
int i;
for(i = 0; i < 5; i++) { string[i] = 'a'; }
printf(string);
```

6 Structs [20 pts]

Given the struct:

```
typedef struct person {
        char name[5];
        int age;
} person_t;
```

a. [10 pts] Write code for two different ways of creating an array of 5 person_ts. Use the same name in both cases.

b. [10 pts] Write code for two different ways of using a for loop to print the age of each person_t in the array. Use the same name as above.

Editorial Break: Good Practices

It's often useful to think of structs as classes. Header files should contain a struct with its data and functions that act on that struct. Those functions should take a pointer to the struct as their first argument (which you can think of as an implicit "this"). Those functions should include constructors and destructors. Note that "constructors" should return a pointer to the struct and not take it as a "this" argument. Furthermore constructors should malloc and destructors should free.

7 Linked List [20 pts]

Given the following implementation of a singly linked list, write a function which reverses the list in place (do not copy the list).

```
//list.h
#ifndef LIST
#define LIST

typedef struct list {
        struct list *next;
        int myInt;
} list_t;

list_t *ctor(int firstitem);
list_t *add(list_t *list, int item);
void printlist(list_t *list);
list_t* reverse(list_t *list);

#endif
```

```c
//list.c
#include "list.h"
#include <stdio.h>
#include <stdlib.h>

//Takes as argument the first item of the list.
//Returns a pointer to the new list.
list_t *ctor(int firstitem)
{
        //Note: there is a much cooler way to do this
        list_t *list = (list_t*)malloc(sizeof(list_t));
        list->myInt = firstitem;
        list->next = NULL;
        return list;
}

//Takes as arguments a list_t* and the item to add.
//Returns a pointer to the new head of the list.
list_t* add(list_t *list, int item)
{
        list_t *newItem = (list_t*)malloc(sizeof(list_t));
        newItem->myInt = item;
        newItem->next = list;
        return newItem;
}

//Takes as argument the list to print.
void printlist(list_t *list)
{
        list_t *ph = list;
        while(ph != NULL) {
                printf("%d\n", ph->myInt);
                ph = ph->next;
        }
}

//Takes as argument the list to reverse.
//Returns the list in reverse order.
list_t* reverse(list_t *list)
{
        // your code goes here
}
```

8 Bonus! [Up to 10 pts Extra Credit]

Explain the meaning of this:

```c
void*(*func)(void*(*)(void*(*)(void*, void*(*)())), void*);
```