

MIDTERM EXAM

CS 217
October 28, 1999

Name:

Precept:

Honor Code:

Score:

| Problem | Score | Max |
|---------|-------|-----|
| 1 | | 15 |
| 2 | | 5 |
| 3 | | 10 |
| 4 | | 15 |
| 5 | | 5 |
| 6 | | 10 |
| 7 | | 10 |
| Total | | 70 |

1. Number Systems

- (a) Translate the following decimal numbers to binary, and binary numbers to decimal. Assume a sign magnitude representation for binary numbers. Assume all binary numbers are 8 bits.

$$125_{10} = \underline{\hspace{2cm}}_2$$

$$-75_{10} = \underline{\hspace{2cm}}_2$$

$$11010101_2 = \underline{\hspace{2cm}}_{10}$$

$$00101010_2 = \underline{\hspace{2cm}}_{10}$$

- (b) Translate the following decimal numbers to 2's complement form, complete the arithmetic operation, and translate the result back into decimal. Indicate if overflow occurs, assuming 8-bit words. Show your work.

i. $10 + 45$

ii. $-40 - 20$

iii. $-40 - 90$

2. Suppose the following three variables have been defined. Assume 32-bit integers.

```
unsigned int a = 0x30cf0034;  
unsigned int b = 0x4a98ff72;  
unsigned int c;
```

Write the 2-3 C instructions needed to assign the high-order 20 bits of a, followed by the high-order 12 bits of b, to variable c. That is, after the instructions, c should contain 0x30cf04a9. (Note: you may not use the constant 0x30cf04a9 in your answer.)

3. Show the output for each of the following two programs.

(a) *counter.h*

```
int get_counter(void);
```

counter.c

```
static counter;
int get_counter(void) {
    int counter = 5;
    return ++counter;
}
```

main.c

```
#include <stdio.h>
#include "counter.h"
static counter;
int main(void) {
    printf("%d\n", counter);
    counter = get_counter();
    printf("%d\n", counter);
    counter++;
    printf("%d\n", counter);
    return 0;
}
```

(b) *main.c*

```
#include <stdio.h>

#define mymul(a, b) { \
    int z = (a * b); \
    printf("%d\n", z); \
    return z; \
}

int main(void) {
    int x = 5, y = 6;
    mymul(x + 2, y);
    printf("%d %d\n", x, y);
    return 0;
}
```

4. Give the declaration/definition for each of the following structures, types, and variables. You can use types and structures defined in early sub-problems to declare variables in later sub-problems.
- (a) Give the definition of structure *day* that has two components: *date* (a pointer to an array of 3 integers), and *events* (a pointer to a pointer to characters).

 - (b) Give the declaration of variable *month*, which is a pointer to structure *day_element*. Also give the definition of structure *day_element*, which consists of two components: *today* (of type structure *day*), and *tomorrow* (a pointer to structure *day_element*).

 - (c) Declare variable *year* as an array of 12 pointers to structure *day_element*.

 - (d) For the following definition (`struct day_element *(* year[12]);`), describe in words what *year+1* points to. Limit your answer to the space provided.

 - (e) Define variable *apply* that points to such a function that takes two arguments: *x* (an array of 30 pointers, each pointing to any type of object), and *y* (a pointer to a function that has two arguments, one is a pointer to any type of object and the other is a pointer to pointer to any type of object). Both of the functions that *y* and *apply* point to return nothing.

 - (f) Suppose that we have already defined functions *f1*, *f2*, *f3* and *f4* earlier in the program. Each of these four functions takes a pointer to any type of object as an argument and returns an integer. Declare a variable *funcs* to be an array of four pointers to these functions.

 - (g) Show how the third function in array *funcs* is invoked on integer variable *a*.

5. The following program implements a simplified link list. It is syntactically correct (i.e., it successfully compiles), but it contains a serious flaw that may cause it to behave in an unpredictable way. Identify the flaw.

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

typedef struct node {
    void *val;
    struct node *next;
} node;
typedef node * pnode;

int getFoo (int i)
{
    /* some random computation */
    int x = 1;
    int y = 2;
    int z = 3;
    return (i + x + y + z);
}

pnode insert1 (pnode p, int level)
{
    char one[4];
    pnode pnew;

    strcpy(one, "one");
    pnew = malloc (sizeof (node));
    pnew->val = one;
    pnew->next = p;
    if (level > 1)
        pnew = insert1 (pnew, level-1);
    return pnew;
}

void main()
{
    pnode s, q;
    char *zero = "zero";
    int foo;

    s = malloc (sizeof (node));
    s->val = zero;
    s->next = NULL;
    s = insert1 (s, 1);
    foo = getFoo(1);
    q = s;
    while (q != NULL)
    {
        printf ("%s\n", (char *)q->val);
        q = q->next;
    }
    return;
}
```

6. The following program uses a binary tree to store integer values. It will compile, but core dump when executed.

- (a) Identify the problem and suggest a fix.
- (b) Show what the corrected program outputs.

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

typedef struct tree
{
    int val;
    struct tree *left, *right;
} tree;

typedef tree * ptree;

ptree insert (ptree p, int nval)
{
    ptree pnew;

    if (p == NULL)
    {
        pnew = malloc (sizeof (ptree));
        pnew->val = nval;
        pnew->left = pnew->right = NULL;
        return pnew;
    }
    else
        if (nval < p->val)
            p->left = insert (p->left, nval);
        else
            p->right = insert (p->right, nval);
    return p;
}

void traverse (ptree p)
{
    if (p != NULL)
    {
        traverse (p->left);
        printf ("%d\n", p->val);
        traverse (p->right);
    }
}

void main()
{
    ptree p = NULL;

    p = insert (p, 3);
    p = insert (p, 4);
    p = insert (p, 2);
    traverse (p);
}
```

7. Suppose you are given the following Makefile.

```
CFLAGS=-A
CC=lcc

main: main.o stat.o freq.o store.o list.o table.o
$(CC) $(CFLAGS) -o main main.o stat.o freq.o store.o list.o table.o

main.o: main.c
$(CC) $(CFLAGS) -c main.c

stat.o: stat.c freq.h
$(CC) $(CFLAGS) -c stat.c

freq.o: freq.c list.h
$(CC) $(CFLAGS) -c freq.c

store.o: store.c table.h list.h
$(CC) $(CFLAGS) -c store.c

list.o: list.c list.h
$(CC) $(CFLAGS) -c list.c

table.o: table.c table.h
$(CC) $(CFLAGS) -c table.c
```

(a) Draw a diagram showing the dependencies between the modules.

(b) What is the output of the following commands within the Unix shell? Assume the commands are invoked in the directory where all compiled modules and source files are located, and all programs compile without error or warning messages.

```
touch main.c
make -n
make
touch freq.h
rm -f main.o
make -n
rm -f list.o
make -n
rm -f *.o
make -n
```

(c) What impact does adding a new local function to `list.c` have on other modules?