COS 217 Final Exam

Princeton University, Fall 2002

January 23, 2002

	Score	Possible
1		13
2		15
3		12
4		20
5		20
6		20
Total		100

Your name:

Unix login:

Precept number: 1 2 3

Write and sign the honor pledge:

1 Assembly language

Translate the following function from C to Sparc assembly language.

```
int f (int *a) {
  int i;
  int s=0;
  for (i=0; a[i]!=0; i++) {
    s += a[i];
  }
  return s;
}
```

2 Data structures for syntax trees

A. Write tagged union data structure definitions in C to represent abstract syntax trees for the following grammar. Assume that *func* is a function name (`char*`) and that *value* is a double-precision floating point value.

exp	::=	func(*exp*)
exp	::=	*exp*+*exp*
exp	::=	value

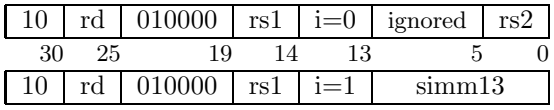
The first rule means that one kind of "exp" has two subcomponents, a function-name and another exp.

B. Write three allocation/initialization functions for your data structure. It is not necessary to make copies of strings. Do something to help defend against a brain-damaged client program.

3 Machine language

Write a function that examines a seqence of N Sparc machine instructions and returns 1 iff register d is the destination of any `addcc` instruction.

addcc




```
int anydest(unsigned int code[], int N, int d) {









}
```

4 Abstract Data Types

The following program reads matrix entries (i,j) from the standard input and adds them to a data structure. Then it calculates the sum of each row and of each column, and adds the squares of all those sums together.

```
#include <stdio.h>

#define N 100
int matrix[N][N];

int main(int argc, char **argv) {
  int i,j,sum,total=0;
  while (2==scanf("%d %d",&i,&j)) {
    matrix[i][j]+=1;
  }
  for(i=0;i<N;i++) {
    sum=0;
    for(j=0;j<N;j++)
         sum+=matrix[i][j];
    total += sum*sum;
  }
  for(j=0;j<N;j++) {
    sum=0;
    for(i=0;i<N;i++)
         sum+=matrix[i][j];
    total += sum*sum;
  }
  printf("%d\n", total);
  return 0;
}
```

A. Rewrite this program so that it's divided into three files: an ADT header, and ADT implementation, and a client implementation. The new client should not have nested loops. Other than your division into modules, preserve as much as possible of the structure and detail of the program.

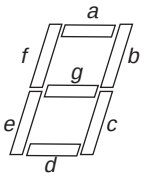
You may write a "first-class ADT" or a "second-class ADT" (what King calls an "abstract object"). You may assume N is a constant (defined by the ADT, not by the client). Do what you can to protect your ADT implementation against brain-damaged clients.

B. Provide a fourth file, an alternate ADT implementation, that's a lot more efficient but not a lot more complicated. Hints: Use incremental evaluation, and don't keep track of information that's not explicitly needed by the client.

ADT's, continued...

5 Circuits

You are familiar with 7-segment displays seen on calculators, etc. In circuit design, the segments are labeled a–g as at left:



For example, applying voltages (1,1,1,1,1,1,0) will light up a "0" symbol, as at right.

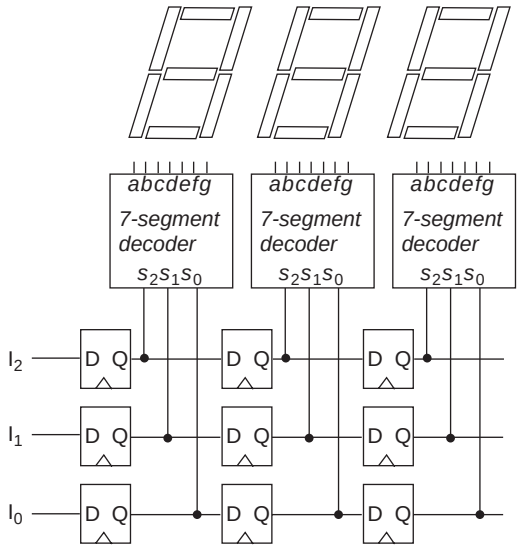
A. Construct a truth table for a 7-segment decoder circuit. It should take as input a 3-digit binary number $s_2 s_1 s_0$ representing a number from 0 to 7, and produce 7 bits of output representing the segments a–g.

B. Construct a sum-of-products circuit implementing *just the a,b,c outputs* of this truth table. Don't implement the e, f, g, h outputs.

C. On three successive clock cycles, the inputs to the following circuit are:

Time	I_2	I_1	I_0
1	1	1	1
2	0	0	1
3	0	1	0

Fill in the pattern on the 7-segment displays after the third clock tick. (Each box labeled "7-segment decoder" is the circuit you constructed in part B.)



6 Interprocess communication

The program below tests whether two trees have the same inorder traversal, using the following quaint method. First, it forks two processes; one child uses `putleaves` to spit the inorder traversal of tree `t1` into a pipe, and the other child uses `putleaves` to spit the inorder traversal of tree `t2` into a different pipe. The parent process reads from the pipes and uses `compare` to compare them.

Please fill in the implementation of the `sameinorder` function to do the pipes and forks as necessary. Here are some hints. You can get substantial partial credit even if you don't manage to use hints 6 and 7.

1. You will need `pipe` and `fork`, but not `exec`.
2. The `read` system call indicates end of file by returning 0 as the number of bytes read.
3. When a child process gets to the end of its tree, it should indicate end of file by closing its "writing end" of its pipe.
4. When reading from a pipe, end of file is indicated only if *no* process has the "writing end" of the pipe open; otherwise, the read will block waiting for input.
5. Your program should work no matter what (well formed) trees are used for t1 and t2.
6. If t1 is much longer than t2, or if the trees are both long but quickly discovered to be unequal, then one or both child processes will try to keep running even when the `compare` function is no longer interested in reading. Therefore, the parent should probably kill the children when it knows the answer.
7. After your children are dead, you should `wait` for them so that the process table does not fill up with zombies.

You may assume all system calls succeed (if given reasonable arguments) so you don't need to check for errors.

```
#include <stdio.h>
#include <stdlib.h>
#include <sys/types.h>
#include <signal.h>
#include <unistd.h>
#include <assert.h>

typedef struct tree *Tree;
struct tree {char key; Tree l, r;};

Tree tree (Tree l, char key, Tree r) {
  Tree t = malloc (sizeof *t);
  assert(t);
  t->l=l; t->key=key; t->r=r;
  return t;
}

void putleaves(int fd, Tree t) {
  if (t) {
    putleaves(fd, t->l);
    write(fd, &(t->key), 1);
    putleaves(fd, t->r);
  }
}
```

```
int sameinorder(Tree t1, Tree t2);

main() {
  Tree t1 = tree(NULL, 'a', tree(NULL, 'b', NULL));
  Tree t2 = tree(tree(NULL, 'a', NULL), 'b', NULL);
  if (sameinorder(t1,t2))
    printf("same inorders\n");
  else printf ("different inorders\n");
}

int compare(int fd1, int fd2) {
  int  n1, n2;
  char c1, c2;
  for(;;) {
    n1 = read(fd1,&c1,1);
    n2 = read(fd2,&c2,1);
    if (n1==0 && n2 == 0)
       return 1;
    else if (c1!=c2)
       return 0;
  }
}
```

```
int sameinorder(Tree t1, Tree t2) {

}
```

```
NAME
     pipe - create an interprocess channel

SYNOPSIS
     #include <unistd.h>

     int pipe(int fildes[2]);

DESCRIPTION
     The pipe() function creates an I/O mechanism called  a  pipe
     and  returns  two file descriptors, fildes[0] and fildes[1].
     The  files  associated  with  fildes[0]  and  fildes[1]  are
     streams and are both opened for reading and writing.
      The O_NDELAY and O_NONBLOCK flags are cleared.

     A read from fildes[0] accesses the data written to fildes[1]
     on  a  first-in-first-out  (FIFO)  basis  and  a  read  from
     fildes[1] accesses the data written to fildes[0] also  on  a
     FIFO basis. ...



NAME
     wait - wait for child process to stop or terminate

SYNOPSIS
     #include <sys/types.h>
     #include <sys/wait.h>

     pid_t wait(int *stat_loc);

DESCRIPTION
     The wait() function will suspend execution  of  the  calling
     thread  until  status  information for one of its terminated
     child processes is available ...

     If wait() returns because the status of a child  process  is
     available,  it  returns the process ID of the child process.
     If the  calling  process  specified  a  non-zero  value  for
     stat_loc,  the  status of the child process is stored in the
     location  pointed  to  by  stat_loc. ...



NAME
     kill - send a signal to a process or a group of processes

SYNOPSIS
     #include <sys/types.h>
     #include <signal.h>

     int kill(pid_t pid, int sig);

DESCRIPTION
     The kill() function sends a signal to a process ...
     The process ... to which the signal is to be sent is specified by pid.
     The signal that is to be sent is specified by sig ... [for example,
     SIGKILL].
```