This exam permits one sheet of notes: NO OTHER BOOKS, NOTES OR EQUIPMENT (besides manual writing instruments and non-data storage watches) are permitted. There are 5 parts and 7 pages. Make sure that you have them all before you begin. The duration is 1 hour.

PART 1 (5 points) State one fact that appears in Chapter 1 of Beck by writing one to three complete sentences in your own words. Both the context and the content must be clearly stated. For example, a sentence like “The condition code is set to indicate the result of this test.” is clearly not acceptable.

PART 2 (25 points)

1. (10 points) Write a clear, complete and correct program in C (or C++) that prints its argument list (i.e., its command line arguments), beginning with the name under which the program is run. **Clarity Counts: Obscure or unnecessarily complicated programs will be graded as wrong.** This goes for all the programming problems in the exam.
2. (10 points) Write a complete and correct program in C (or C++) for Unix that prints its environment strings in the order they appear in the environment, one per line, except that environment strings that are null should not be printed, i.e., a blank line should not appear for such strings. (A string of the form "NAME=" is not null.) Don’t worry about long environment strings.

3. (5 points) Explain clearly why, when we speak strictly, argument lists and environments belong to processes but they do not belong to programs.
PART 3 (30 points)

Read the documentation below edited from Linux for information needed to answer the following questions. Write your program(s) on the next page.

1. (25 points) Write a complete and correct program in C (or C++) opens the utmp file (whose pathname is /etc/utmp) and for each entry in that file, prints on one line, in the order below,

(a) the process id (pid) in decimal,
(b) the user name (Assume it is null-terminated!),
(c) the devicename of tty (a string), and
(d) the type of login process (print one of INIT, LOGIN, USER or DEAD).

Remember that fopen sets the file position indicator to the beginning, each file read operation leaves the position indicator at the byte right after the byte last read (or at the end of file position), and fread will return 0 if it called when the position indicator is at end of file.

NAME

utmp, wtmp - login records

SYNOPSIS

#include <utmp.h>

DESCRIPTION

The utmp file allows one to discover information about who is currently using the system. The file is a sequence of entries with the following structure declared in the include file:

/* utmp.h for Linux, by poe0daimi.aau.dk */
/* adapted for csi402 Spring 1997 midterm exam by sdc@cs.albany.edu */
typedef int pid_t;
typedef long int time_t;

struct utmp {
    short int ut_type;      /* type of login (see the four values below) */
    #define INIT_PROCESS 5
    #define LOGIN_PROCESS 6
    #define USER_PROCESS 7
    #define DEAD_PROCESS 8

    pid_t ut_pid;            /* pid of login-process */
    char ut_line[12];        /* devicename of tty -"/dev/", null-term */
    char ut_id[2];           /* abbrev. ttyname, as 01, s1 etc. */
    time_t ut_time;          /* logintime */
    char ut_user[8];         /* username, not null-term */
    char ut_host[16];        /* hostname for remote login[not null-term]*/
    long int ut_addr;        /* IP addr of remote host */
};
This structure gives the name of the special file associated with the user’s terminal, the user’s login name and the time of login in the form of time(2).

2. (5 points) Now, omitted documentation implies that if the user name is 8 characters then it is not null terminated; otherwise (when it is less than 8 characters in length) the string is null terminated. Show or explain how you would modify your program so that it would be correct for all lengths (0-8) of user names.
PART 4 (9 points) The header file elf.h makes use of type definitions Elf32_Word, Elf32_Off, and Elf32_Addr for various fields in ELF structures. For example, the section header structure is

```c
/*
 * Section header
 */
typedef struct {
    Elf32_Word    sh_name;    /* section name */
    Elf32_Word    sh_type;    /* SHT_... */
    Elf32_Word    sh_flags;   /* SHF_... */
    Elf32_Addr    sh_addr;    /* virtual address */
    Elf32_Off     sh_offset;  /* file offset */
    Elf32_Word    sh_size;    /* section size */
    Elf32_Word    sh_link;    /* misc info */
    Elf32_Word    sh_info;    /* misc info */
    Elf32_Word    sh_addralign; /* memory alignment */
    Elf32_Word    sh_entsize; /* entry size if table */
} Elf32_Shdr;
```

These types are defined in /usr/include/sys/elftypes.h by

```c
typedef unsigned long   Elf32_Addr;
typedef unsigned long   Elf32_Off;
typedef unsigned long   Elf32_Word;
```

Even though each of these three types is implemented by 32 bit binary data, they are used for different purposes in the ELF standard.

1. Elf32_Off signifies an offset. In what kind of thing is an offset meaningful and what does it mean?

2. Elf32.Addr signifies an address. In what kind of thing is an address meaningful and what does it mean?

3. Elf32_Word sometimes signifies an index. In what kind of thing is an index meaningful and what does it mean?
PART 5 (31 points)

1. (3 points) Write the arguments for a gcc command that will compile the C program in file module1.c to an object file module1.o that includes debugging information symbols.

   gcc

2. (3 points) Assume module2.c has already been compiled. Write the arguments for a gcc command that will link the object files module1.o (from item 1 above) and module2.o to produce an executable file whose name is exam402.

   gcc

3. (6 points) Assume the above was done on Solaris. Express the value of the 4th byte (at offset 3) in the file exam402 in three ways:
   
   (a) A printed character under the ASCII code;
   (b) A 2 digit hexadecimal numeral;
   (c) An unsigned 2 digit decimal numeral:

   Hint: The ASCII code for uppercase letter A, expressed as a binary numeral, is 0100 0001. The ASCII codes for the upper case letters A-Z are a sequence of consecutive integers.

4. (9 points) In order to bring some “unformatted” data, whose length cannot be predicted at compile time (such as the contents of the “.text” section in an ELF file), from somewhere in a file (not the beginning) into C program memory, show three C (or C++) library function calls that should be invoked. List them in a correct order. (Error checks should be omitted.) Assume that the file has been opened, pFile contains its FILE * pointer value, and both the location of the data (displacement in pData) in the file and its length (in lData, a quantity of bytes) have been determined. When your code has completed the reading, the value of pointer pData should point to the data. Space for the data has not been allocated.

5. (10 points) Suppose the Makefile stored in a directory consisted of the following four lines:

   abcd : abe charlie
     cat abe charlie > abcd

   abe :

   6
where there is an infamous “invisible” TAB character just preceding the cat. (The command cat abe charlie > abcd will concatenate the contents of files abe and charlie and store the resulting data in a file named abcd. If abcd already exists, then the original is replaced.) Running the command “gmake abcd” can have the four possible outcomes below. For each outcome, describe exact circumstances that will cause it to occur.

(a)  % gmake abcd  
cat abe charlie > abcd  
%

(b)  % gmake abcd  
(Some makes may display “target abcd up to date” here.)  
%

(c)  % gmake abcd  
make: *** No rule to make target `charlie`, needed by `abcd`. Stop.

(d)  % gmake abcd  
cat abe charlie > abcd  
cat: abe: No such file or directory  
make: *** [abcd] Error 1