COS126: General Computer Science	Spring 1996

Third Midterm Exam

Date: Thursday, April 25, 1996

Instructions: *This exam is open-book. You may use your personal notes, any handouts and lecture notes from this course, the course readers, and "The C Programming Language" by Kernighan and Ritchie.* **No other materials are allowed.** *You have 90 minutes for this exam. There are four problems, some of which consist of more than one part. Write all your answers in an exam booklet. Show your work so that we can give you partial credit in case your final answer is wrong.*

Be sure that your name and precept number are on the front of the booklet and printed legibly. Write and sign the honor code pledge.

Problem 1, 25 points: Let `struct tree` be defined as in the lecture notes.

```
struct tree {int value; struct tree *left; struct tree *right;};
```

Write a function `counteven` with the declaration

```
int counteven(struct tree *t);
```

that counts the number of nodes in the tree that store an even number. (An *even* number is one that is divisible by 2.)

Thus if the tree contains the numbers $\{2, 7, 11, 15, 20\}$, then `counteven` returns 2, since only 2 and 20 are even.

Problem 2, 25 points: Write a C program that reads a file `myfile`, and outputs the number of times "`not`" appears in it, ignoring upper and lower case. You will lose 5 points if there are any circumstances that cause your program to core-dump.

Note that "`not`" could occur as part of another word. Thus for example if `myfile` contains the following text

```
     Note that "not" could occur as part of another word.
```

then the program should output 3.

Problem 3, 25 points:

Part a, 15 points: Suppose a paged virtual memory system uses 2^{16}-byte pages. An address is 32 bits long (addressing bytes), and there are 2^{28} bytes of physical memory. Suppose the following is a fragment of the page table (addresses in hex):

page no.	in memory	physical location
0A11	no	disk location
0A12	yes	231
0A13	no	disk location
0A14	yes	FFF
0A15	no	disk location
0A16	yes	0A7

Suppose the following sequence of memory accesses occurs:

```
0A11ABCD
0A12ABCD
0A110123
0A160000
0A14AA11
```

Also assume that if and when the first page fault occurs, virtual page 0A16 is replaced, using its physical location. On subsequent page faults, if any, the highest numbered virtual page among 0A11 through 0A16 that is in physical memory is replaced.

For each memory access in the sequence of memory accesses given above, say whether or not a page fault occurs. If a page fault does occur, give the changes to the page table.

Part b, 10 points: Suppose we quadruple the size of physical memory. Is the 32-bit virtual address adequate? Why? How many bits are needed to specify physical page numbers with this larger memory?

Problem 4, 25 points:

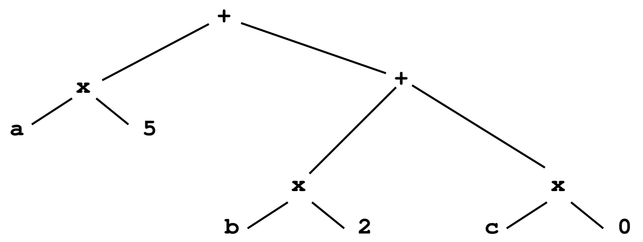
Part a, 5 points: Give the parse tree for the expression

```
((((a*b)+c)*d)+(e+f))
```

Part b, 20 points: We wish to parse expressions for polynomials. An example of the input form is

```
ax5+bx2+c
```

which represents polynomial $ax^5 + bx^2 + c$. We wish to build a parse tree of the form



Below is a code fragment to build the parse tree for an arbitrary polynomial expression with the following restrictions

1. A coefficient is present for each term and is restricted to a single lowercase letter among `'a'` through `'z'`
2. Exponents, if present, are between 2 and 9. An exponent is not present for a linear term, e.g. $ax = ax^1$, and `x` itself is not present for a constant term, e.g. $a = ax^0$.

The input is in a character string

```
char *inputstr;
```

defined as a global variable. The function `polyparse()` is incomplete. Complete this function. Note that the functions `error()`, `look()`, `get()` and `maketree()` are *exactly* the same as those used in lecture to parse fully parenthesized arithmetic expressions (Lecture 15 notes, pages 15 and 16 and Lecture 16 notes, page 18).

These are definitions you have seen in lecture, repeated here for your convenience.

```
char *inputstr;  /* entire input string */

int pos;  /*  current position within input string */

struct tree {char oper; struct tree *left, *right; };

 void error(char *message)
 { printf("%s, bye!\n",message);
   exit(1);
  }

 char look(void)
 { return inputstr[pos]; }


 char get(char *options)
 {int i;
  for(i=0; options[i] != '\0'; i++)
    if (inputstr[pos]==options[i])
       {pos++;
        return options[i];
       }
  error("syntax error");
 }

 struct tree *maketree(char operation,
               struct tree *leftchild, struct tree *rightchild)
 {
  struct tree *t;
  t = malloc(sizeof (struct tree));
  if (t==NULL) error("out of memory");
  (*t).oper=operation;
  (*t).left=leftchild;
  (*t).right=rightchild;
  return t;
 }
```

This is the new function that you need to complete.

```
 struct tree *polyparse(void)
 {
  char coef;
  char var;
  char exp;
  struct tree *l;
  struct tree *r;

  coef = get("abcdefghijklmnopqrstuvwxyz");
  l = maketree(coef, NULL, NULL);
  if (look() == '\0')
      { r = maketree('0', NULL, NULL);
        return maketree('x', l, r);
      }
  var = get("x+");
  if (var == '+')
      { r = maketree('0', NULL, NULL);
        l = maketree('x', l, r);
      }
  else
      {
        if (look() == '\0')
            { r = maketree ('1', NULL, NULL);
              return maketree('x', l, r);
            }

          /*  fill in rest of code here */

      }
  r = polyparse();
  return maketree('+', l, r );
}
```