

FINAL EXAMINATION:

Systems Programming in C
GU (INN060) and CTH (TDA200)

DAY: 7 March 2001
TIME: 08.45 – 12.45
ROOM: V

Instructor: David Modjeska (tel. 031-772 5403)
Department of Computer Science
Results: Announced by 23 March 2001
Materials permitted: All written/printed materials, e.g., books, notes, printouts.
No computers, e.g., notebooks, handhelds
Marking ranges GU: G – 28 points, VG – 48 points
Marking ranges CTH: 3 – 24 points, 4 – 36 points, 5 – 48 points

Please note:

- Write clearly (illegible = incorrect)
- Number each page, and begin each new problem on a new page.
- Write your personal number on each page, including the cover page
- Points will be deducted for unnecessarily complicated or unstructured answers
- It is permitted to write in either Swedish or English

Good luck!

Problem 1 (10 points)

Consider the following code. What will be printed when the program executes?

```
#include <stdio.h>

#define THREE 2
#define ALPHA(x) ((--x) - THREE)

static int z = 1;
int y = 2001;

int swapInt1(int x, int y)
{
    int iTemp = x; x = y; y = iTemp;
    return iTemp;
}

int swapInt2(int *x, int *y)
{
    int iTemp = *x; *x = *y; *y = iTemp;
    return iTemp;
}

int BETA(int* x)
{
    int z = 0;
    static int y = 0;

    (*x)++;
    z++;
    y = ALPHA(y);
    return y;
}

int main()
{
    static int iTemp = 0;

    printf("y = %d\n",y);
    printf("z = %d\n",z);

    iTemp = BETA(&z);

    printf("iTemp = %d\n",iTemp);
    iTemp = swapInt1(y, z);
    printf("y = %d\n",y);
    printf("z = %d\n",z);

    iTemp = BETA(&z);

    printf("iTemp = %d\n",iTemp);
    iTemp = swapInt2(&y, &z);
    printf("y = %d\n",y);
    printf("z = %d\n",z);

    exit(0);
}
```

Problem 2 (15 points)

The task is to write a short program to sort a list of words that the user will enter at the keyboard.

- 1) The program should first prompt the user to enter a list of words, and then read one word per line, until the user types Ctrl+D (to end input). (6 points)
- 2) Then, the program will then call the C library function `qsort()` to sort the words. For this purpose, it will be necessary to write a string-comparison function, as required by `qsort()`. A “man” page for `qsort()` is attached to this exam. (6 points)
- 3) Finally, the program will print out a list of sorted words, one per line. (3 points)

The program can define a reasonable line length, and ignore “words” longer than this length. Similarly, the program can define a reasonable list length, and ignore words entered after this length has been reached, proceeding instead with sorting and printing. In no case should the program crash.

Problem 3 (10 points)

The problem is to write a function that takes an unsigned integer as a parameter, and returns an unsigned integer with the parameter’s bits in reversed order. That is, you should write a function with the following prototype:

```
unsigned ReverseBits(unsigned x);
```

For example, if the unsigned integer whose bit pattern 1000100010001000 is given as a parameter to the function, the result should be 0001000100010001. Do not assume anything about the byte size of an unsigned integer for a particular machine architecture.

(continued on next page)

Problem 4 (25 points)

Your task is to write a function that will multiply two matrices and return a pointer to the product matrix. In case it is needed, a short review of matrix multiplication is attached to this exam.

- 1) The first step is to define a structure type `Matrix`. The structure should show the number of rows and columns in a particular matrix, as well as the matrix elements themselves. (3 points)
- 2) The second step is to define a macro `ELEM(pMatrix, row, col)` for accessing a specific matrix element. This macro is needed because the dimensions of a matrix are not known until run-time, but C arrays normally require specifying the dimensions at compile-time. The accessed element will be in the row `row` and column `col` of the matrix indicated by the pointer `pMatrix`. (8 points)
- 3) The third step is to write the function `MultiplyMatrix()`, which has the following prototype:

```
Matrix *MultiplyMatrix(Matrix *pMatrix1,  
                       Matrix *pMatrix2);
```

- a) First, the function must validate its arguments. If the number of columns in the first matrix does not equal the number of rows in the second matrix, the function should return `NULL`. (3 points)
- b) Second, the function must create a new matrix, which will be the product of the two arguments. The new matrix must be initialized with its correct dimensions, as well as memory for the elements that will be calculated in step 3c below. (If initialization is impossible, `MultiplyMatrix()` should return `NULL`.) The number of rows in the new matrix must equal the number of rows in the first matrix argument; the number of columns in the new matrix must equal the number of columns in the second matrix argument. (3 points)
- c) Third, the elements of the new matrix must be calculated. This should be done in the traditional way (which is reviewed in the attachment to this exam). (8 points)