Lösningsförslag

Systemnära Programmering, 5p
28 oktober 1994

Uppgift 1 (5 p)

Följande skrivs ut:

```
rast
eller inte
```

Uppgift 2 (2 + 3 + 1 p)

(a)
```
#define BETWEEN(a, b, c) ((a) >= (b) && (a) <= (c))
```

(b)
```
#define LSAME(c1, c2) \
    (islower((c1)) ? \
        islower((c2)) ? (c1) == (c2) : (c1) - 'a' == (c2) - 'A' : \
        isupper((c2)) ? (c1) == (c2) : (c1) - 'A' == (c2) - 'a')
```

(c) Ja, t ex vid följande anrop:

```
BETWEEN(++i, j, k)
LSAME(++c, cc)
```

då kommer ++–variabeln att öka med minst två.

Uppgift 3 (6 p)

```
int NoWord(char *s1, char *s2)
{
    int i = 0, nr = 0;

    if (strlen(s1) == 0)
        return 0;
    if (strlen(s2) == 0)
        return 1;

    while (s1[i] != '\0') {
        while (s1[i] != '\0' && strchr(s2, s1[i]))    /* Hoppa over alla */
            i++;                                      /* ordeseparatorer */
        if (s1[i] != '\0') {
            nr++;
            while (s1[i] != '\0' && !strchr(s2, s1[i])) /* Las till nasta */
                i++;                                     /* ordeseparator  */
        }
    }
    return nr;
}
```

Uppgift 4 (9 p)

```
#include    <stdio.h>

#define DELETE  -2
void usage(char *name);

/*
   Usage: change [-char -newchar] [file]
   Changes all characters "char" to "newchar" in "file" and prints the
   results on standard output. If no file is given then standard input
   is used. If no character is given everything is copied.
*/
```

```
main(int argc, char *argv[])
{
    FILE *InFile;
    int OldChar = -2, NewChar = DELETE, c;

    if (argc > 4)                         /* Too many arguments     */
        usage(argv[0]);
    else if (argc == 1)                   /* No argument, use stdin */
        InFile = stdin;
    else if (argv[1][0] == '-') {         /* Check for arguments    */
        if (argc < 3 || strlen(argv[1]) != 2 || argv[2][0] != '-')
            usage(argv[0]);
        OldChar = argv[1][1];
        if (argv[2][1] != '\0')
            NewChar = argv[2][1];
        if (argc == 4) {                  /* file-argument given or not */
            if ((InFile = fopen(argv[3], "r")) == NULL) {
                perror(argv[0]);
                usage(argv[0]);
            }
        } else
            InFile = stdin;
    } else {
        if (argc != 2)
            usage(argv[0]);
        if ((InFile = fopen(argv[1], "r")) == NULL) {
            perror(argv[0]);
            usage(argv[0]);
        }
    }

    while ((c = getc(InFile)) != EOF)
        if (c != OldChar)
            putchar(c);
        else if (NewChar != DELETE)
            putchar(NewChar);
}

void usage(char *name)
{
    fprintf(stderr, "\nChanges all characters char to newchar in file\n");
    fprintf(stderr, "and prints the results on standard output. If no\n");
    fprintf(stderr, "file is given then standard input is used.\n");
    fprintf(stderr, "If no character is given everything is copied.\n");
    fprintf(stderr, "\nUsage: %s [-char newchar] [file]\n\n", name);
    exit(1);
}
```

Uppgift 5 (2 + 4 p)

(a) Scriptet kollar först så att anropet är korrekt, med ett argument. Om anropet är felaktigt så skrivs ett felmeddelande ut som talar om hur scriptet ska användas.

Scriptet numrerar och skriver ut sina argument. Ex

```
% myscript arg1 arg2 arg3
1: arg1
2: arg2
3: arg3
%
```

Scriptet sätter också status-variabeln till 1 vid fel och 0 när allt gick bra.

(b)
```
#!/bin/csh -f
# Usage: map command directory
# Maps "command" on every file in "directory", calls map
# recursively on all subdirectories

if ($#argv != 2) then                    # Check nr of arguments
    echo Wrong number of arguments
    echo Usage: $0 command directory
    exit 1
endif

if (! -d $argv[2]) then                  # Check if argument is directory
    echo $0\: $argv[2] not a directory
    echo Usage: $0 command directory
    exit 2
endif

cd $argv[2]
foreach file (*)
    if (-d $file) then
        $0 "$argv[1]" $file              # Recursive call
    else
        $argv[1] $file                   # Apply command on file
    endif
end
exit 0
```

Uppgift 6 (4 p)

Skriver ut ledtexten på en ny rad, utan att byta rad efter texten. try_me anropas. try_me läser in ett tecken, i den lokala variabeln c, och om det inte är newline så anropas try_me rekursivt, skulle tecknet vara newline så skrivs c ut och funtionen returnerar till "sin anropare" som i sin tur skriver ut sitt c osv. Sist skrivs det ut två newline av main.

Skriver alltså ut den inskrivna texten, avslutad med newline, baklänges.

Uppgift 7 (3 + 3 p)

(a) **RUNNING**: exekverar instruktioner på processorn
BLOCKED: väntar på någon yttre händelse, t ex I/O
READY: klar att exekvera, men väntar i en kö eftersom processorn är upptagen

(b) **READY** → **RUNNING**: Processorn ledig och scheduleraren har valt en process som får exekvera.
READY ← **RUNNING**: Scheduleraren avbryter en process som exekverar, t ex eftersom den har exekverat tillräckligt länge.
RUNNING → **BLOCKED**: Processen har exekverat en instruktion som innebär att den måste vänta på något, t ex I/O.
BLOCKED → **READY**: Det en process har väntat på är klart, dvs processen är redo att exekvera så snart som processorn blir ledig.

Uppgift 8 (2 p)

C-shell skapar en ny process för att exekvera ett externt kommando. Använder fork för att skapa en kopia av sig själv och väntar (wait) sedan på att barnprocessen ska exekvera klart. Barnprocessen anropar exec för att ersätta sig självt med koden för kommandot som ska exekveras. När barnet exekverat klart väcks moderprocessen upp och sätter status-variabeln till det värde som barnet indikerat. Sedan skriver moderprocessen ut en ny prompt och fortsätter ta emot kommandon på vanligt sätt.