Fall term 2010
KAIST EE209 Programming Structures for EE

Mid-term exam

Thursday Oct 21, 2010

Student's name: _____

Student ID: _____

The exam is closed book and notes. Read the questions carefully and focus your answers on what has been asked. You are allowed to ask the instructor/TAs for help only in understanding the questions, in case you find them not completely clear. Be concise and precise in your answers and state clearly any assumption you may have made. All your answers must be included in the attached sheets. You have 70 minutes to complete your exam. Be wise in managing your time.
Please do not fill in the "Score" fields below. Self-grading is not allowed. Good luck!

Scores

Question 1	_____/10___
Question 2	_____/10___
Question 3	_____/15___
Question 4	_____/20___
Question 5	_____/15___
Total	_____/70___

1. Number Systems (10 pts)

   a) 5 < 10 < 20;
   Is this C expression true, false or undefined? (2pts)

   True.
   5<10 <20 ⇔ (5<10) < 20 ⇔ (1) < 20 ⇔ 1 (True)

   b) 1 && 2 && 3 || 0;
   Is this C expression true, false or undefined? (2pts)

   True.

   c) i = j = 2;
   Is this C expression true, false, or undefined? (2pts)

   True. The expression evaluates to 2 (True).

   d) char ch = -5;
   Write ch as an 8-bit binary number. (4 pts)

   11111011

2. (String functions) Please fill out the function bodies below. (10pts)

int mystrlen(const char *s) returns the length of the string s, not including the terminating '\0' character. That is, the behavior is exactly the same as strlen(const char* s) in C run-time library (except the return value: strlen() returns size_t but we use int) (5pts)

```
int mystrlen(const char *s)
{
    int n = 0;

    while (*s++)
        n++;

    return n;
}
```

-2: if not handle a string with zero length (s="\0")
(It's OK to assume s != NULL)

char *mystrdup(const char *s) returns a pointer to a new string which is a duplicate of the string s. Please allocate the memory for the new string with malloc(). The behavior is the same is strdup(const char *s) in C run-time library. You can **use mystrlen()** but **cannot use strcpy()** in the body of mystrdup(). (5pts)

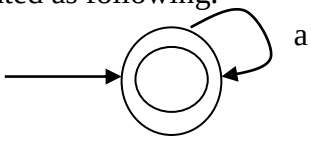
```
char *mystrdrup(const char *s)
{
    int n = mystrlen(s);
    char *p, *q;

    p = (char *)malloc(n + 1); /* it's ok not to cast the type to (char *) */
    assert(p);                 /* it's ok not to have assert() here */
    if (p == NULL)             /* check if we're out of memory */
      return NULL;

    q = p;
    while ((*q++ = *s++) != '\0')      /* copy *s until *s become a null char */
        ;
    return p;
}
```

-1: if not checking (p == NULL)
-2: if not null terminate the string
-1: for each function call other than mystrlen()

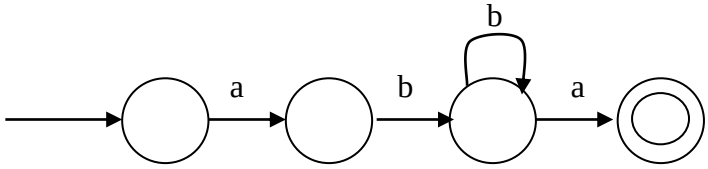
3. (DFA) It is known that every regular expression can be described by a deterministic finite automata (DFA). Assuming we have characters ‘a’, and ‘b’ for input, ^a*$ can be represented as following.



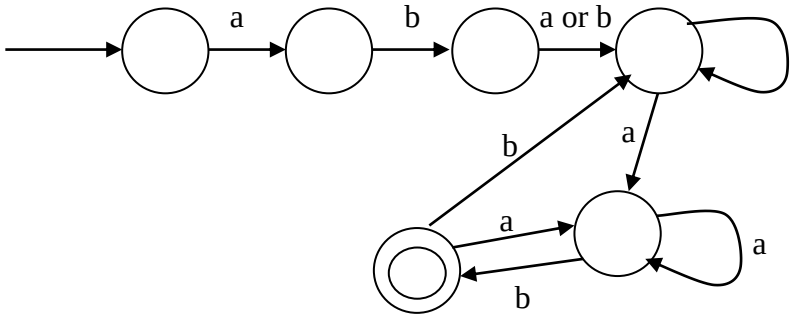
Please draw your DFA diagrams with a start state (marked with an incoming arrow) and success states(circled twice). If you do not specify a particular input character on a state, we assume the DFA automatically fails on that input character (ex. The above DFA fails when it receives ‘b’). As in the assignment # 2, * matches zero or more occurrence of the previous character while + matches one or more occurrences of the previous character. . matches either ‘a’ or ‘b’. ^ is the start of the character string while $ is the end of character string.

-1: for each incorrect transition
-1: for each required transition to correct DFA

3a) ^ab*ba$ (7pts)



3b) ^ab.+ab$ (8pts)



4. (Type and Size) What is the output of this program on a lab machine? If some output is undefined, say undefined. (20pts)

```
#include <stdio.h>
#include <string.h>

struct Table {
    int key;
    int value;
};

int main(void)
{
    double i, *p, **pp;
    struct Table t, *pt;
    char strA[] = “abc”;
    char strB[] = “a\0bc”;
    char strC[] = {‘a’, ‘b’, ‘c’};

    p = &i;
    pp = &p;
    pt = &t;

    printf(“(1) = %d\n”, sizeof(i));
    printf(“(2) = %d\n”, sizeof(p));
    printf(“(3) = %d\n”, sizeof(*p));
    printf(“(4) = %d\n”, sizeof(*pp));
    printf(“(5) = %d\n”, sizeof(t));
    printf(“(6) = %d\n”, sizeof(*pt));
    printf(“(7) = %d\n”, strlen(strA));
    printf(“(8) = %d\n”, strlen(strB));
    printf(“(9) = %d\n”, sizeof(strB));
    printf(“(10) = %d\n”, strlen(strC));
    return 0;
}
```

(1): 8
(2): 4
(3): 8
(4): 4
(5): 8
(6): 8
(7): 3
(8): 1
(9): 5
(10): undefined

5. Simple Programs (15pts)

a) What is the output of f(x)? (5pts)

Print the reverse string of x. The output is edcba.

```
void f(char *s)
{
    char *p = s;
    char *q = s;
    while (*s)
        s++;
    for (s--; s > p; s--, p++) {
        char c = *s;
        *s = *p;
        *p = c;
    }
    printf("%s", q);
}

…
char x[] = "abcde";
f(x);
```

b) What does f(798) produce? (2pts) What does the function do? (3pts)

f(798) = 897 ended by \n (2pts).
Print each digit in the number in the reverse direction. (3pts)

```
void f(unsigned int n)
{
    do {
      putchar ('0' + (n % 10));
    } while (n /= 10);
    putchar ('\n');
}
```

c) We may rewrite (b)'s code like following. Does this produce the identical output? If not, when is the answer different? (5pts)

It is identical except the case of n = 0.(3pts) The function below prints \n if n=0, whereas the function in (b) prints out 0\n. (2pts)

```
void f(unsigned int n)
{
      for ( ; n; n /= 10)
          putchar('0' + (n % 10));
      putchar('\n');
}
```