Fall term 2012
KAIST EE209 Programming Structures for EE

Mid-term exam

Thursday Oct 25, 2012

Student's name: _____

Student ID: _____

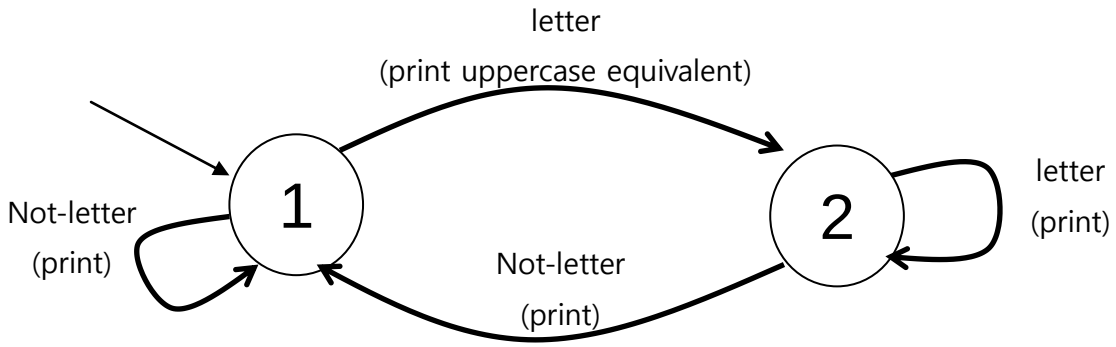
The exam is closed book and notes. Read the questions carefully and focus your answers on what has been asked. You are allowed to ask the instructor/TAs for help only in understanding the questions, in case you find them not completely clear. Be concise and precise in your answers and state clearly any assumption you may have made. All your answers must be included in the attached sheets. You have 120 minutes to complete your exam. Be wise in managing your time. Good luck!

Scores

Question 1	_____/20___
Question 2	_____/25___
Question 3	_____/20___
Question 4	_____/30___
Total	_____/90__

1. Deterministic finite state automaton (DFA). (20 points)

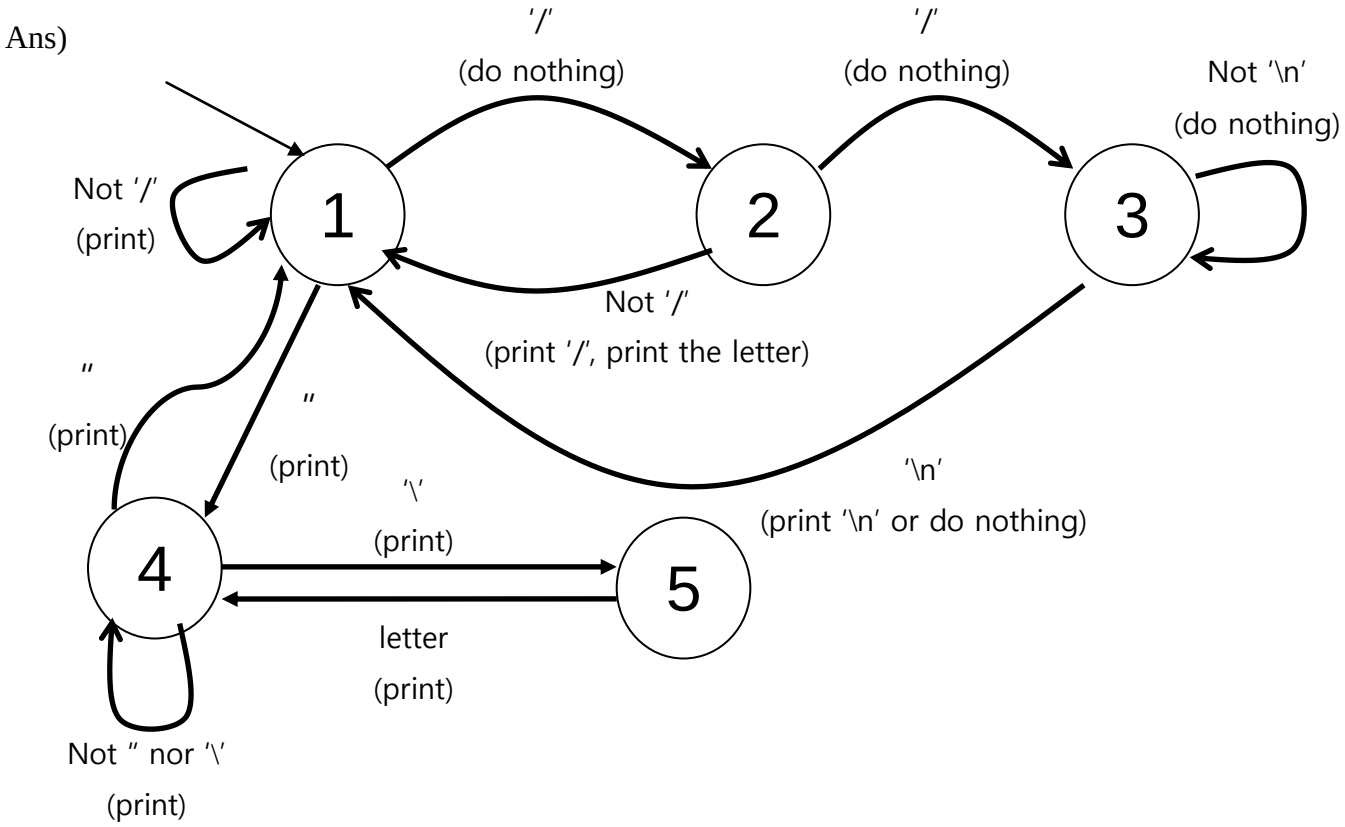
   a) What does the following DFA do? (3 pts)



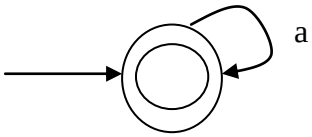
Ans) It converts the first character of each word to an uppercase character.

   b) Draw a DFA that removes all single-line comments that follow "//" in C source codes. (5 pts)

Ans)

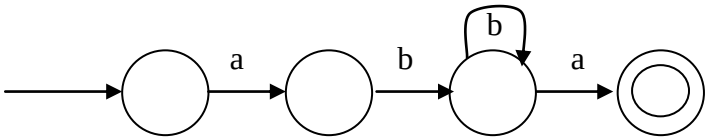


(c), (d) It is known that every regular expression can be described by a DFA. Assuming we have characters ‘a’, and ‘b’ as input, ^a*$ can be represented as following.



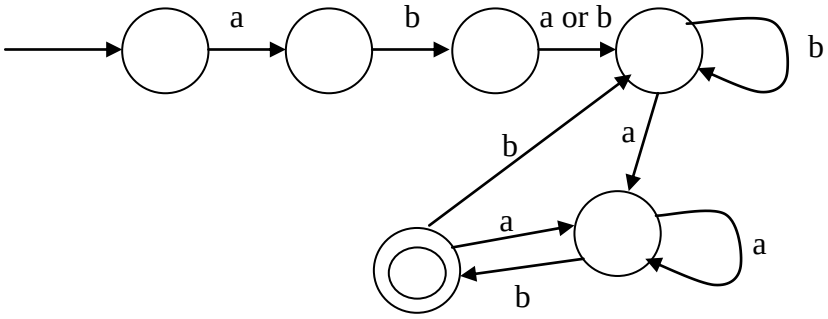
Please specify the regular expression that exactly matches each of the following DFAs. If a state does not specify a particular input character, we assume the DFA automatically fails on that input character (ex. The above DFA fails when it receives ‘b’). As in the assignment # 2, * matches zero or more 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.

c) (5pts)



Ans) ^ab+a$

d) (7pts)



Ans) ^ab.+ab$

2. Explain what the following does. If a program crashes, report the reason why. Assume we're on a 32-bit machine. (25pts, (a) to (g): 2pts each, (h): 4 pts (i): 7pts)

```
a) int *p = NULL;
   p = 1;
```

Ans) p has the value 1 (address 0x1).

```
b) int *p = NULL;
   *p = 1;
```

*Ans) it crashes (with a segmentation fault signal). This is because *p = 1 would attempt to write the value at an invalid memory location (0x0)*

```
c) double *k[25][25];
   printf("sizeof(k) = %d\n", sizeof(k));
   printf("sizeof(k[10][10]) = %d\n", sizeof(k[10][10]));
```

Ans)

*sizeof(k) = 2500 (25*25*4 (sizeof(double *))*
sizeof(k[10]k[10]) = 4 (each element is of the double type)*

```
d) int k = 1;
   int *p = &k;
   *p = 0;

   k = (3 < 2) ? (k == 1) : (k == 0);
   printf("k = %d\n", k);
   k = (*p = 0) ? (k == 1) : (k == 0);
   printf("k = %d\n", k);
```

Ans)

k = 1 (3<2 is false, so (k == 0) is stored at k, which is 1)
*k = 1 (*p = 0 stores 0 to k, and it evaluates to false (0), and k== 0 is executed which is 1)*

e)
```
int a[7] = {0, 1, 2, 3, 4, 5, 6};
int *p = &a[3];
p += 2;
*p += 2;
printf("x=%d\n", *p++);
printf("size=%d\n",sizeof(p)/sizeof(a[0]));
```

Ans) x = 7 (p points to a[5])
size = 1 (sizeof(p) = 4, sizeof(a[0]) = 4)

f)
```
struct L {
int val;
struct L* link;
} x, *y;
x.val = 20;
y = &x;
y->val = 30;
y->link = &x;

printf("x.val = %d\n", x.val);
printf("some val = %d\n",
        x.val + y->link->val + y->link->link->val);
```

Ans)
x.val = 30
some val = 90
(x.val(30), y->link->val(30),y->link->link->val(30))

g)
```
char *name = "kyoungsoo";
name[sizeof(name)-2] = 'Y';
printf("name=%s\n", name);
```

Ans) it crashes since "kyoungsoo" is stored at the read-only memory section and changing the value would violate memory access (segmentation fault).

*Note that char name[] = "kyoungsoo"; is different from char *name = "kyoungsoo"; where the former is simply initialization of the array, name[] which sits at the read-write memory section. The latter initializes the pointer, name, with the address of the first byte of "kyoungsoo" which sits at the read-only memory section. All string literals are stored at the read-only memory section.*

(h), (i)

```
char *itoa(int x)
{
  char buf[5];
  sprintf(buf, "%d", x);  /* see the comment below */
  return buf;
}
```

- **sprintf() is the same as printf() except that the output goes to the first argument instead of stdout. E.g. sprint(buf, "%d", 123); would be the same as strcpy(buf, "123");**
  /* buf[0] = '1', buf[1]='2', buf[2] = '3', buf[3] = 0; */

(h) Point out two serious bugs with the code. (4 pts)

Ans)

1. *If x is a large value (with more than four digits), buf would overflow, and the answer would be incorrect.*
2. *The returned value is a pointer to buf which sits at a stack and disappears after the function call. That is, the return value would point to the location which would have a garbage value after the function call.*

(i) Fix the bugs. (7 pts)

Ans)

```
char *itoa(int x)
{
        char buf[16];    /* enough to hold 2^32 */
        sprintf(buf, "%d", x);
        return strdup(buf);
}
```

Note: we will give you full points if you address the above two problems (e.g., you can call malloc() and strcpy() instead of strdup())

3. Playing with C strings. (20 pts) (If you need more space, please use the back of the sheet.)

(a) **`int hassubstr(const char *s, const char *p)`** returns 1 if s has p as a substring or 0 if not. For example, the function would return 1 if s="elephant" and p="leph" since p is a substring of s. Please fill out the function below to implement this. (10 pts) (Note: you should not use any C runtime library functions like strchr() or strstr(). Be careful about error processing (e.g., s and p could point to NULL). return 1 if *p is '\0'.)

```
int hassubstr(const char *s, const char *p)
{

  if (s == NULL || p == NULL) return 0;
  if (*p == '\0') return 1;

  for (; *s; s++) {

    if (*p == *s) {
        const char *pa = p+1;
        const char *sa = s+1;

        while (1) {
            if (*pa == '\0') return 1;
            if (*sa == '\0') break;
            if (*pa++ != *sa++)
                  break;
         }
    }
  }
  return 0;

}
```

(b) **`int is_anagram(const char *s, const char *p)`** returns 1 if s and p are anagrams or returns 0 otherwise. s and p are anagrams if they have the same length and s can be converted to p by rearranging the letters of s. For example, opts, post, stop, pots, tops are anagrams. (10 pts) (hint: count the number of the same letters in s and compare it with that of the same letters in p. You can use an integer array to remember the number of occurrence of each letter in s.)

```
int is_anagram(const char *s, const char *p)
{

  int cnt[256] = {0};
  int len = 0;

  for (; *s; s++, len++)
     cnt[*s]++;

  for (; *p; p++, len--) {
    if ((len == 0) || (--cnt[*p]) < 0)
       return 0;
  }

  return (len == 0);
}
```

4. (key, value) table management (30pts)

We implement a simple Table data structure that maintains a list of (key, value) pairs, where both key and value point to non-NULL strings. In **`table.h`**, we have

```
#ifndef TABLE_INCLUDED
#define TABLE_INCLUDED
  typedef struct Table_t *Table_T;
  extern Table_T Table_new(void);
  extern void Table_free(Table_T t);
  extern int Table_empty(Table_T t);
  extern int Table_add(Table_T t, const char *key, const char *val);
  extern char* Table_search(Table_T t, const char *key);
  extern void Table_del(Table_T t, const char *key);
#endif
```

In **`table.c`**,

```
#include <stdlib.h>
#include <assert.h>
#include "table.h"

struct item {
  char *key;
  char *value;
  struct item *link;
};

struct Table_t {
  struct item *head;
};

Table_T Table_new(void) {
 Table_T table = calloc(1, sizeof(Table_t));
 assert(table != NULL);
 return table;
}

void Table_free(Table_T t) {
  struct item *p, *next;
   assert(t != NULL);

   for (p = t->head; p; p = next) {
       next = p->next;
       free(p->key);
       free(p->value);
       free(p);
   }
   free(t);
 }
```

(a) Why is **`struct Table_t`** defined in **`table.c`** instead of **`table.h`**? (2pts)

Ans) It hides the fields of struct Table_t from the user of the module. That is, the user should not be able to directly access the fields of struct Table_t. This improves the abstraction of the type, Table_T.

(b) Write the code for **`int Table_empty(Table_T t)`** that returns 1 if the table is empty, 0 if not. (3pts)

Ans)

```
int Table_empty(Table_T t)
{
    return (t->head == NULL);
}
```

(c) Write code **`int Table_add(Table_T t, const char *key, const char *val)`** that stores an item with (key, value) at the start of the list in Table t. Allocate the memory for key and value and copy the content from the input so that the table operations won't be disrupted by client's operations. Return 0 in case of an error: (1) if either key or val is NULL, (2) if the table already has an item with the same key, or (3) if it bumps into any other errors. Return 1 if the operation is successful. You can call any functions declared in table.h. (10 pts)

Ans)

```
int Table_add(Table_T t, const char* key, const char* val)
{
  struct item *p;

  if (key == NULL || val == NULL) /* handle error cases */
        return 0;

  if (Table_search(key)) /* if key exists, return 0 */
        return 0;

  /* allocate the memory for the item */
  p = malloc(sizeof(struct item));
  if (p == NULL)
        return 0;

  /* copy the key: own the key string */
  p->key = strdup(key);
  if (p->key == NULL) {
        free(p);
        return 0;
  }

  /* copy the value: own the value string */
  p->val = strdup(val);
  if (p->val == NULL) {
        free(p->key);
        free(p);
        return 0;
  }
```

```
 /* add to the start of the list */
  p->next = t->head;
  t->head = p;

 return 1;
}
```

(d) Write the code for **`char* Table_search(Table_T t, const char *key)`** that returns the value of the item whose key matches "key". Return NULL if such an item is not found. (7 pts)

Ans)

```
char* Table_search(Table_T t, const char *key)
{
 struct item *p;

 if (key == NULL)
     return NULL;

 for (p = t->head; p; p = p->next) {
        if (strcmp(p->key, key) == 0)
               return p->value;
 }
 return NULL;
}
```

(e) Write the code for **`void Table_del(Table_T t, const char *key)`** that removes an item whose key matches "key". It does nothing when such an item does not exist. (8pts)

Ans)

```
void Table_del(Table_T t, const char *key)
{
   struct item *p, *pre = NULL;

  if (key == NULL)
      return;

   for (p = t->head; p; pre = p, p = p->next) {
          if (strcmp(p->key, key) == 0) {

            if (pre)
               pre->next = p->next;
            else
               t->head = p->next;

            free(p->key);
            free(p->value);
            free(p);
            return;
          }
    }
}
```