Name: Section: Rollno:

ESC101: Fundamentals of Computing (End Semester Exam)

Set A

Department of Computer Science & Engineering, IIT Kanpur

September 19, 2016

Total Number of Pages: 21 Total Points 200

Instructions

1. Read these instructions carefully.
2. Write you name, section and roll number on all the pages of the answer book, **including the ROUGH pages**. You will be penalised if you fail to write the name, roll number and correct section.
3. Write the answers cleanly in the space provided. Space is given for rough work in the answer book.
4. Using pens (blue/black ink) and not pencils. Do not use red pens for answering.
5. Do not exchange question books or change the seat after obtaining question paper.
6. Even if no answers are written, the answer book has to be returned back with name and roll number written.
7. Sign the attendance sheet.

Question	Points	Score
1	40	
2	40	
3	40	
4	40	
5	40	
Total:	200	

I PLEDGE MY HONOUR THAT DURING THE EXAMINATION I HAVE NEITHER GIVEN NOR RECEIVED ASSISTANCE.

.....

Signature

Question 1. This question consists of 10 parts. Each part is a small objective type question. **Write down your answers in the space provided below each question.**

(a) (4 points) Can you combine the following two statements into one?

```
1 char *p;
2 p = (char*) malloc (100);
```

a) `char p = *malloc (100);`
b) `char *p = (char) malloc (100);`
c) `char *p = (char*) malloc (100);`
d) `char *p = (char*) (malloc*) (100);`

Solution: c

(b) (4 points) If a variable is a pointer to a structure, then which of the following operator is used to access data members of the structure through the pointer variable?

a) `.`
b) `&`
c) `*`
d) `->`

Solution: d

(c) (4 points) What would be the equivalent pointer expression for: `a[i][j][k][l]` (`a` is declared as `int ****a;`)

a) `*((((a+i)+j)+k)+l)`
b) `*(*(*(*(a+i)+j)+k)+l)`
c) `((*(a+i)+j)+k+l)`
d) `*a+i+j+k+l;`

Solution: b

(d) (4 points) In C, if you pass an array as an argument to a function, what actually gets passed?

a) Values of the elements in the array
b) First element of the array
c) Address of the first element of the array
d) Address of the last element of the array

Solution: c

(e) (4 points) What is the output of the following code snippet?

```
#include <stdio.h>
int main() {
    int a[5] = {5, 1, 15, 20, 25};
    int i, j, m;
    i = 1+a[1]++;
    j = a[1]++;
    m = a[i++];
    printf("%d, %d, %d", i, j, m);
    return 0;
}
```

a) 2, 3, 15
b) 2, 2, 16
c) 3, 2, 15
d) 3, 3, 16

Solution: c

(f) (4 points) State whether each statement is True or False. (e.g. If you believe all are true, write a)True, b)True, c)True, d)True as answer)

a) To return control back to the calling function, we must use the return statement
b) A structure can be nested inside another structure
c) Can a structure have a pointer to itself?
d) `malloc()` returns a `float` pointer if memory is allocated for storing `float`s and a `double` pointer if memory is allocated for storing `double`s

Solution: False, True, True, False

(g) (4 points) What is the output of the following code?

```
#include <stdio.h>
void fun(int*, int*);
int main() {
    int i = 5, j = 2;
    fun(&i, &j);
    printf("%d, %d", i, j);
    return 0;
}
void fun(int *i, int *j) {
    *i = *i**i;
    *j = *j**j;
}
```

a) `5, 2`
b) `25, 4`
c) `10, 4`
d) `2, 5`

Solution: b

(h) (4 points) What would be the correct function declaration for the function fun?

```
#include <stdio.h>
int main() {
    int a[3][4];
    fun(a);
    return 0;
}
```

a) `void fun (int a[][]) { return; }`
b) `void fun (int a[3][]) { return; }`
c) `void fun (int a[][4]) { return; }`
d) `void fun (int *a[3][4]) { return; }`

Solution: c

(i) (4 points) What is the output of the following code?

```
#include <stdio.h>
int main() {
    int a[10] = {20};
    printf("%d", 0[a]);
}
```

a) `0`
b) `20`
c) Garbage value
d) Error

Solution: b

ROUGH WORK

(j) (4 points) What is the output of the following code?

```
#include <stdio.h>
int main() {
    static int a[2][2] = {1, 2, 3, 4};
    int i, j;
    static int *p[] = {(int*)a, (int*)a+1, (int*)a+2};
    for(i=0; i<2; i++) {
        for(j=0; j<2; j++) {
            printf("%d, %d, %d, %d\n", *(*(p+i)+j), *(*(j+p)+i), *(*(i+
                p)+j), *(*(p+j)+i));
        }
    }
    return 0;
}
```

a)
```
1, 1, 1, 1
2, 3, 2, 3
3, 2, 3, 2
4, 4, 4, 4
```

b)
```
1, 2, 1, 2
2, 3, 2, 3
3, 4, 3, 4
4, 2, 4, 2
```

c)
```
1, 2, 3, 4
2, 3, 4, 1
3, 4, 1, 2
4, 1, 2, 3
```

d)
```
1, 1, 1, 1
2, 2, 2, 2
2, 2, 2, 2
3, 3, 3, 3
```

Solution: d

ROUGH WORK

Question 2. This question consists of 2 parts both of which require you to predict the output given the code and the input.

(a) (20 points) Predict the output of the following code.

```
#include <stdio.h>
#include <stdlib.h>
void r_a(int *arr, int n, int k) {
    if(n==k || k==0)
        return;
    if(n-k>=k) {
        int i;
        for(i=n-k;i<n;i++) {
            int t = arr[i];
            arr[i] = arr[i-k];
            arr[i-k] = t;
        }
        r_a(arr,n-k,k);
    }
    else {
        int i;
        for(i=0;i<n-k;i++) {
            int t = arr[i];
            arr[i] = arr[i+n-k];
            arr[i+n-k] = t;
        }
        r_a(arr+n-k,k,2*k-n);
    }
}

int main() {
    int n,k,i;
    scanf("%d%d",&n,&k);
    int *arr = (int*)malloc(n*sizeof(int));
    for(i=0;i<n;i++)
        scanf("%d",arr+i);
    r_a(arr,n,k);
    for(i=0;i<n;i++)
        printf("%d ",arr[i]);
    return 0;
}
```

Input	Output
4 2 5 2 3 4	
8 2 6 4 1 3 8 2 9 5	
11 4 2 6 0 5 3 4 1 9 7 8 11	

Solution:

Input	Output
4 2 5 2 3 4	3 4 5 2
8 2 6 4 1 3 8 2 9 5	9 5 6 4 1 3 8 2
11 4 2 6 0 5 3 4 1 9 7 8 11	9 7 8 11 2 6 0 5 3 4 1

ROUGH WORK

(b) (20 points) Predict the output of the following code.

```
#include <stdio.h>
int func(int a, int b) {
    if(b>a)
        return -1;
    if(b==0)
        return 1;
    return a * func(a-1,b-1) / b;
}
int main() {
    int a,b;
    scanf("%d%d",&a,&b);
    printf("%d",func(a,b));
}
```

Input	Output
4 2	
10 6	
62 59	

Solution:

Input	Output
4 2	6
10 6	210
62 59	37820

ROUGH WORK

Question 3. (40 points) You are given the following code snippet which possibly has a lot of errors and your job is to debug it.

Note: In the line number mentioned if there is no error then write "OK". If there is some error then write the correct code which should be a single statement. You are not allowed to add extra lines to the code. **Unnecessary edits will be penalized.**

For example, if you feel the following line is incorrect and a should be initialized with 0:
int a = 2;
You will correct it in the table as follows:
int a = 0;

```
1  struct node
2  {
3      int val;
4      struct node * next;
5  };
```

Given a linked list (created using the above structure *node*), you have to swap every two adjacent nodes and return the new head. In case, the length is odd, do not modify the last node.
For example, if you are given the following the linked list:
$1 -> 2 -> 3 -> 4 -> 5$
Your function will modify the linked list as follows and return its new head:
$2 -> 1 -> 4 -> 3 -> 5$
The function *swapPairs* takes the head of the input linked list, modifies it accordingly and returns the new head.

```
1  struct node* swapPairs(struct node* head) {
2      struct node* new_head;
3      struct node* prev = (struct node*)malloc(sizeof(struct node));
4      int flag = 0;
5      while(head!=NULL)
6      {
7          struct node* second = head->next;
8          head->next = head->next->next;
9          head->next->next = head;
10         if(flag==0){
11             new_head = head;
12             flag = 1;
13         }
14         if(prev!=NULL)
15             prev->next = second;
16         prev = head;
17         head = head->next->next;
18     }
19     return new_head;
20 }
```

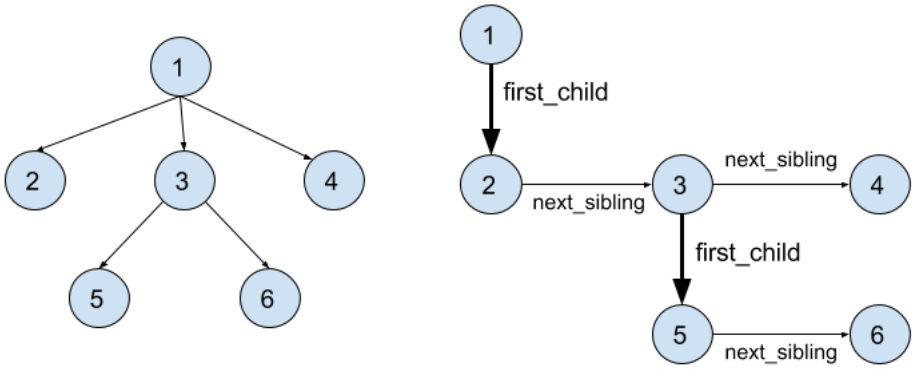
Suspicion in	**Your response**
`line 2`	
`line 3`	
`line 5`	
`line 9`	
`line 11`	
`line 17`	

Solution:

Suspicion in	**Your response**
`line 2`	struct node* new_head = head;
`line 3`	struct node* prev = NULL; OR struct node* prev;
`line 5`	while(head ! = NULL && head→next! =NULL)
`line 9`	second→next = head;
`line 11`	new_head = second;
`line 17`	head = head→next; OR head = second→next→next;

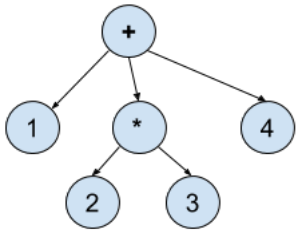
ROUGH WORK

Question 4. Trees are a fundamental structure ubiquitous in Computer Science. Some of you might know about Binary trees, in which every node has atmost two children i.e. nodes which it has access to. We now study the general tree structure, wherein every node may have one or more children. We define a *sibling* of a node as a node which is on the same *level*. For example, in the figure below, 3 and 4 are siblings of each other as well as 2. We can represent a tree using linked-lists as shown below. In principle, each tree node has a linked-list of its children, with a child pointer pointing to the head of this list. We will use this representation in our program.



(a) A General Tree Structure (b) Representation using linked lists

The figure above represents graphically, the structure of the tree. Trees are very useful for describing expressions. For example, the expression (1 + (2 * 3) + 4) can be described as a tree in the following manner.



All the *internal* nodes i.e. the nodes which have children are operations, while the *leaf* nodes, which have no children are numbers. The order of evaluation as prescribed by the parantheses in the expression is effectively captured in the tree structure. For computing the value of the expression represented by a tree, we need to work our way from the bottom i.e. the *leaf nodes*, right upto the top of the root. Let us try writing a program to evaluate an expression represented by a tree.

(a) (4 points) First, we need to define C structures that can represent our data. As is mentioned in the comments, we only deal with addition and multiplication operations. Fill in the blanks appropriately. **Keep your answers as *simple* as possible. There will be penalty for overly complicated solutions**

```
struct node {
    /* val contains the value stored at the node (integer) */
    /* val is meaningful only for the leaf nodes */
    /* val is ignored for nodes that are operations (internal nodes) */
    /* IMP - The value of val can be ANYTHING for internal nodes */
    int val;

    /* The value of op can be '+', '*' or '\0' */
    /* The value '\0' signifies that the node is not an operation */
    char op;

    /* The next node in the child linked-list of the parent node */
    ____________________ next_sibling;
    /* The head of the child linked-list for this node */
    ____________________ first_child;
};
```

(b) (16 points) Now, you need to implement the *eval_expr* function which takes the root of a tree as the input, and the returns an integer equal to the value of the expression represented by the tree. Fill in the blanks appropriately. ***Think simple.*** **You may NOT use ternary operators. Only simple expressions (single assignment statements) are allowed as answers. No single-line while loops etc. are permitted. You are also NOT allowed to modify the tree structure i.e. you CANNOT modify any of the node's fields.** (Hint : Use recursion)

```
int eval_expr(struct node * root) {
    if (_______________________________) { /* It is an op */
        int base_val;
        if (_______________________________) base_val = ____;
        else if (root->op == '+') base_val = 0;

        struct node *cur_child = root->first_child;
        while (cur_child != NULL) {
            if (root->op == '*') {
                ___________________________________________ ;
            } else if (root->op == '+') {
                ___________________________________________ ;
            }

            cur_child = ____________________________________ ;
        }

        return _________________________ ;

    } else {
        return _________________________ ;
    }
}
```

(c) (20 points) You also need to implement a debugging functionality, namely a function to print the expression tree into human-readable form. For example, for the expression tree displayed before, the output should be (1 + (2 * 3) + 4). **Adhere strictly to this output format**. Note that the parantheses are important. Again, note that we are only dealing with addition and multiplication operators. Fill in the blanks appropriately. ***Think simple.*** **You may NOT use ternary operators. Only simple expressions (single assignment statements) are allowed as answers. No single-line while loops etc. are permitted. You are also NOT allowed to modify the tree structure i.e. you CANNOT modify any of the node's fields.** (Hint : Use recursion)

```
void print_expr(struct node *root)
{
    if (root->_____________________  == NULL) {
        /* This is a leaf node i.e. a value*/
        printf(_________________________________);

    } else {
        printf("(");
        struct node *cur_child = root->first_child;
        while (cur_child != NULL) {
            ___________________________________________;
            cur_child = _______________________________________;
            if (cur_child != NULL) {
                printf(_________________________________);
            }
        }
        printf(")");
    }
}
```

Solution: (a)

```
struct node {
    /* val contains the value stored at the node (integer) */
    /* val is meaningful only for the leaf nodes */
    /* val is ignored for nodes that are operations (internal nodes)
       */
    /* IMP - The value of val can be ANYTHING for internal nodes */
    int val;

    /* The value of op can be '+', '*' or '\0' */
    /* The value '\0' signifies that the node is not an operation */
    char op;

    /* The next node in the child linked-list of the parent node */
    struct node * next_sibling;
    /* The head of the child linked-list for this node */
    struct node * first_child;
};
```

(b)

```
int eval_expr(struct node * root) {
```

```
    if (root->op != '\0') { /* It is an op */
        int base_val;
        if (root->op == '*') base_val = 1;
        else if (root->op == '+') base_val = 0;

        struct node *cur_child = root->first_child;
        while (cur_child != NULL) {
            if (root->op == '*') {
                base_val *= eval_expr(cur_child) ;
            } else if (root->op == '+') {
                base_val += eval_expr(cur_child) ;
            }

            cur_child = cur_child->next_sibling ;
        }

        return base_val ;

    } else {
        return root->val ;
    }
}
```

(c)

```
void print_expr(struct node *root)
{
    if (root->first_child  == NULL) {
        /* This is a leaf node i.e. a value*/
        printf("%d", root->val);

    } else {
        printf("(");
        struct node *cur_child = root->first_child;
        while (cur_child != NULL) {
            print_expr(cur_child);
            cur_child = cur_child->next_sibling;
            if (cur_child != NULL) {
                printf(" %c ", root->op);
            }
        }
        printf(")");
    }
}
```

ROUGH WORK

Question 5. This question consists of **3** parts. Each part is a short question ranging from output finding, multiple choice to option matching. Exact details are provided in the question description.
Write down your answers in the space provided below each question.

(a) (16 points) **Array Indexing:** Please select **ALL** line numbers which will give any error. There can be **MULTIPLE** such lines. And the error could be either compiler or run time error.
Marking Scheme: Each incorrect option will fetch **-2** (but total marks will not be less than zero).

```
1  #include <stdio.h>
2
3  int main() {
4      int arr[] = {1,2,3,5,7};
5      printf("%d\n", *(arr+2));
6      printf("%d\n", (*arr)[2]);
7      printf("%d\n", *(&*(arr+2) - 1));
8      printf("%d\n", *(*(&(arr+2)) - 1));
9      printf("%d\n", *(2+arr));
10     printf("%d\n", 2[arr]);
11     printf("%d\n", (arr+1)[3]);
12     arr = arr + 2;
13     printf("%d\n", arr[1][3]);
14     return 0;
15 }
```

Choose all the options which represent **INCORRECT** lines and mention them in the box provided below:
Clarification: If you feel Line 5 and Line 10 are incorrect and the rest are correct, then just mark a) and f).

a) `Line 5`
b) `Line 6`
c) `Line 7`
d) `Line 8`
e) `Line 9`
f) `Line 10`
g) `Line 11`
h) `Line 12`
i) `Line 13`

Solution: b, d, h, i

ROUGH WORK

(b) (8 points) **Switch Case:** This is your favorite output prediction problem. Please write the outputs for the following cases.
Marking Scheme: +2 for each correct output, No negative marks.

```
#include <stdio.h>

int main() {
    int i, j;
    scanf("%d", &j);
    switch (i = j + 1) {
        case 0:
            printf("LINE A\n");
        case 1:
            printf("LINE B\n");
        case 2:
            printf("LINE C\n");
        default:
            printf("LINE D\n");
    }
    return 0;
}
```

Input	**Predicted Output**
0	
1	
-1	
5	

Solution:

Input	**Predicted Output**
0	Line B Line C Line D
1	Line C Line D
-1	Line A Line B Line C Line D
5	Line D

ROUGH WORK

(c) (16 points) **Pointers:** This is a option matching type question on your favorite topic: pointers.

```
1 int i = 5, j = 10;
2 int *tmpPtr, *jPtr = &j;
3 *jPtr = (j + 10);
4 j = (*jPtr + 5);
5 tmpPtr = jPtr + 8;
6 *tmpPtr = (2*i);
```

The memory location of variable **i** begins at address **00043248442** in base 10 (decimal) and the memory location of variable **j** begins at address **00043248410** (Again in base 10). For each expression in the left column, select the correct value in the right column (after execution of all statements).
Note: Each integer quantity occupies 4 bytes of memory.
Marking Scheme: 4 marks allotted for each expression. No Negative marks.

Expressions	**Values**
1) `tmpPtr`	A) 25
2) `*jPtr`	B) 00043248418
3) `i`	C) 40
4) `*tmpPtr`	D) 5
	E) 35
	F) 00043248442
	G) 10

You have to match the values provided in the left column to the values in the right column in the table provided above. Mention the options matched with each of the quantities in boxes below.

Expressions	**Values**
1) `tmpPtr`	
2) `*jPtr`	
3) `i`	
4) `*tmpPtr`	

Solution:

Expressions	**Values**
1) `tmpPtr`	F)
2) `*jPtr`	E)
3) `i`	G)
4) `*tmpPtr`	G)

ROUGH WORK