| ESC 101: Fundamentals of Computing | | | | **End-sem Exam** (25 Nov 2018) | | **100 marks** Page **1** of **10** |
|---|---|---|---|---|---|---|
| Name | | | | | | |
| Roll No | | Dept. | | Section | | |

1. This question paper contains 5 pages (10 fully printed sides of paper). Please verify.
2. Write your name, roll number, department and section **on every sheet** of this booklet.
3. Write your final answers neatly **with a blue/black pen**. Pencil marks may get smudged.

---

**Q1.** Write **T** or **F** for True/False (write **only in the box on the right hand side**) (**20x1=20marks**)

| | | |
|---|---|---|
| 1 | It is illegal in the C language for any function to invoke the `main()` function. It is also illegal for the `main()` function to recursively call itself. | **F** |
| 2 | Suppose $f(n) = (\log n)^3$ and $g(n) = n$ then $f(n) = \mathcal{O}\big(g(n)\big)$ (log is in base 2) | **T** |
| 3 | Let `int goof(int x){...}` be a function. Then invoking this function inside the `main()` function as `goof(3.414);` will not cause any compilation errors. | **T** |
| 4 | A C program can have two functions with the same name if their return types are different e.g. `int pym(void){...}` and `char* pym(void){...}` | **F** |
| 5 | If we are given an array with $n$ float numbers, then $\mathcal{O}(n)$ operations are sufficient to verify if the given array is already sorted in non-decreasing order or not. | **T** |
| 6 | If we have `float f = 4.0;` then `printf("%0.2f",f >> 1);` will print 2.00 since the right bitshift operator can be used to divide by powers of 2. | **F** |
| 7 | In a C program with no global variables, a function `int foo(void){...}` that takes no input and returns an int must return the same value each time it is called | **F** |
| 8 | If we declare two arrays as `int c[4]; int *d = (int*)malloc(16);` then the expressions `sizeof(c)` and `sizeof(d)` will give us the same value. | **F** |
| 9 | `char str[10] = "Hello";` Then the following if statement will print YES `if(str[5] = '\0') printf("YES"); else printf("NO");` | **F** |
| 10 | Writing a function without proper indentation will cause compilation errors. e.g. `int fac(int n){if(n == 0) return 1; else return n*fac(n-1);}` | **F** |
| 11 | 54 in octal, 2C in hexadecimal, 101100 in binary, all represent the same integer | **T** |
| 12 | All variables, including arrays (static as well as dynamic) that are declared inside a function, are destroyed when that function returns. | **F** |
| 13 | If `struct node{ int a, b, c; };` is a structure and `struct node *ptr;` is a pointer to a `node` variable. Then `sizeof(ptr)` will give us the value 12. | **F** |
| 14 | If we have two functions `int foo(int x){...}` and `int bar(int y){...}` then if we call `bar()` inside `foo()`, then we cannot call `foo()` inside `bar()` | **F** |
| 15 | Suppose $f(n) = 2^n$ and $g(n) = n^{(\log_2 n)}$ then we have $g(n) = \mathcal{O}\big(f(n)\big)$ | **T** |
| 16 | If we have a character array `char rtr[10] = "ByeESC101";` then the expressions `strlen(rtr)` and `sizeof(rtr)` will give us the same value. | **F** |
| 17 | The following loop will cause a compilation error if included in a C program. `for(float c = 0.0; c < 4.5; c = c + 1){printf("%0.2f",c);}` | **F** |
| 18 | A good way to ensure that quicksort always offers an $\mathcal{O}(n \log n)$ time complexity is to always choose the smallest or the largest element in the array as the pivot | **F** |
| 19 | Calling free with a pointer variable not assigned any memory, has no effect on the variable, nor does it ever generate any errors e.g. `int *qtr; free(qtr);` | **F** |
| 20 | The space needed to store an array of 100 integers is the same as that needed to store a singly linked list with 100 nodes, each node capable of storing an integer. | **F** |

**Q2.** Fill the circle (**don't tick**) next to the correct option (**only one choice correct**).(**6x2=12marks**)
**LOOK AT ALL OPTIONS CAREFULLY – YOU HAVE ENOUGH TIME IN THIS EXAM**
**Do not overwrite/scratch answers (will get a straight zero) – use a pencil if unsure.**

**2.1** Given a singly linked list with $n$ nodes and two pointers, one to the head node and the other to the tail node, which of the following operations can be executed in constant time i.e. time that doesn't depend on the length $n$ of the list?

1. Insert a node just after the head node
2. Insert a node just before the head node
3. Insert a node just after the tail node
4. Insert a node just before the tail node

| | | |
|---|---|---|
| **A** | Only 1) can be done in const. time | ○ |
| **B** | Only 1), 2) can be done in const. time | ○ |
| **C** | Only 1), 2), 3) can be done in const. time | ● |
| **D** | 1), 2), 3), 4) can be done in const. time | ○ |

**2.2** The following function calculates the binomial. Suppose we call this function from `main()` with arguments `bin(n, n/2);` (`n` is even). How many times will `bin()` recursively call itself?

```
int bin(int n, int k){
    if (k < 0 || k > n) return 0;
    else if (k == 0 || n == k) return 1;
    else return bin(n-1,k) + bin(n-1,k-1);
}
```

| | | |
|---|---|---|
| **A** | $\mathcal{O}(\log n)$ recursive calls | ○ |
| **B** | $\mathcal{O}(n)$ recursive calls | ○ |
| **C** | $\mathcal{O}(n^2)$ recursive calls | ○ |
| **D** | $\mathcal{O}(2^n)$ recursive calls | ● |

**2.3** What is the output of the following program? Assume `stdio.h` has been included.

```
int main(){
    int arr[5] = {4, 13, 54, 12, 0};
    int *ptr1 = &arr[0], *ptr2 = ptr1 + 3;
    printf("%d %d", *ptr2 - *ptr1 , ptr2 - ptr1);
    return 0;
}
```

| | | |
|---|---|---|
| **A** | 0 3 | ○ |
| **B** | 0 12 | ○ |
| **C** | 8 3 | ● |
| **D** | 8 12 | ○ |

**2.4** Which of the following library functions, when invoked with appropriate arguments, returns nothing, i.e. has a void return type?

| | | |
|---|---|---|
| **A** | `strchr` from `string.h` | ○ |
| **B** | `realloc` from `stdlib.h` | ○ |
| **C** | `strcmp` from `string.h` | ○ |
| **D** | None of the above | ● |

**2.5** Consider the statement `printf("%d", x + ~x + 1);` where `x` is an int variable and `~` is the bitwise complement operator. We run this program on a new computer using 2's complement, as well as on a very old computer that uses 1's complement. On which of the computers will 0 get printed?

| | | |
|---|---|---|
| **A** | Only on old computer | ○ |
| **B** | Only on new computer | ○ |
| **C** | On both computers | ● |
| **D** | On neither computer | ○ |

**2.6** `struct Blah{ int val; char a[5]; };` I want an array of 100000 `Blah` elements and `int a = 100000;` Which of the following statements will allocate enough memory for this array?

| | | |
|---|---|---|
| **A** | Only 1) | ● |
| **B** | Only 1) and 2) | ○ |
| **C** | Only 1) 2) and 3) | ○ |
| **D** | 1) 2) 3) and 4) | ○ |

1. `struct Blah *btr = (struct Blah*) malloc(a*sizeof(struct Blah));`
2. `struct Blah *btr = (struct Blah*)malloc(a*9);`
3. `Blah btr[a];`
4. `struct Blah *btr = (struct Blah*)(a * malloc(sizeof(struct Blah)));`

**Q3.** Fill in the circles next to ALL CORRECT options (**multiple may be correct**).(**6x3=18marks**)
**LOOK AT ALL OPTIONS CAREFULLY – YOU HAVE ENOUGH TIME. NO OVERWRITING.**

**3.1** I have an unknown integer x and I write the statement `if(x & (x-1) == 0) printf("Y");` (& is the bitwise AND operator). Integers take 4 bytes and use the two's complement system. Which of the following values of x would cause Y to be printed?

| | | |
|---|---|---|
| A | `x = 0;` | ● |
| B | `x = 1;` | ● |
| C | `x = (1 << 20);` | ● |
| D | `x = (1 << 31);` | ● |

**3.2**
```
int **A = (int**)malloc(32);
for(int i = 0; i < 4; i++)
    A[i] = (int*)calloc(5,4);
```
Which expressions would always access the element at the 2nd row and 3rd column of this array of arrays?

| | | |
|---|---|---|
| A | `A[2][3]` | ○ |
| B | `A[1][2]` | ● |
| C | `**(A + 7)` | ○ |
| D | `*(A[1] + 2)` | ● |

**3.3** Consider the function $f(n) = n^2 \cdot (\log(n^2))^2 + 5n^2$. For which of the following definitions of the function $g(n)$ would we have $f(n) = \mathcal{O}(g(n))$? Assume that logs are to base 2.

| | | |
|---|---|---|
| A | $g(n) = n^2 \cdot (\log n)^2$ | ● |
| B | $g(n) = n^3$ | ● |
| C | $g(n) = n^2 \cdot \log(n^4)$ | ○ |
| D | $g(n) = n^2$ | ○ |

**3.4** Suppose `int a = x & y, b = x | y;` where `x` and `y` are two integers. For which of the given pairs of values of `x` and `y` would the following print Y?
`if(a < b) printf("Y");else printf("N");`
In the following, ~ is the bitwise complement operator.

| | | |
|---|---|---|
| A | `x = 31; y = (x >> 4);` | ● |
| B | `x = ~(1 << 30); y = x;` | ○ |
| C | `y = (1 << 31); x = ~y;` | ○ |
| D | `y=(1 << 9); x=(y >> 1);` | ● |

**3.5** `struct Node {int val; struct Node *next;};`
Using this structure, a singly linked list is created with 8 nodes. If given a pointer `struct Node *h;` to the head node of this list, which of the following expressions will correctly generate the value stored at the second node in the list?

| | | |
|---|---|---|
| A | `(h->next).val` | ○ |
| B | `(h.next)->val` | ○ |
| C | `(*(h->next)).val` | ● |
| D | `(h->next)->val` | ● |

**3.6** I wish to print the powers of 2 from 1 to n (no spaces) for a given value of n. For example, for n = 4, I wish to print "24816" (without quotes). Which of the following recursive functions will correctly do so for all values of n from 1 to 50? (**Fill in the circles below the boxes**).

```
long pow2(int n){
  if(n == 0)
    return 1;
  long f = pow2(n-1);
  printf("%ld", 2*f);
  return 2*f;
}
```
A ●

```
int pow2(int n){
  if(n == 1){
    printf("2");
    return 2;
  }
  int f = pow2(n-1);
  printf("%d", 2*f);
  return 2*f;
}
```
B ○

```
long pow2(int n){
  if(n == 1){
    printf("2");
    return 2;
  }
  long f = pow2(n-1);
  printf("%ld", 2*f);
  return 2*f;
}
```
C ●

**Q4** In the space provided, write down the output of the program when given the input indicated. Every line of output carries equal weightage. All test cases have equal weight. (**5x4=20marks**)

**4.1** Write down the output for the given input in the space provided. The output has four lines. Assume that `stdio.h` has been included. In order to indicate a space in your answer, leave a small gap (**don't write a dot like Prutor**).

| OUTPUT |
|---|
| **U V** |
| **A B** |
| **V W** |
| **V W** |

```
struct V{ char a, b; };
void foo(struct V v){
  v.a = 'U';
  v.b = 'V';
  printf("%c %c\n", v.a, v.b);
}
void bar(struct V *p){
  p->a = 'V';
  p->b = 'W';
  printf("%c %c\n", p->a, p->b);
}
int main(){
  struct V s;
  s.a = 'A'; s.b = 'B';
  foo(s);
  printf ("%c %c\n", s.a, s.b);
  bar(&s);
  printf ("%c %c", s.a, s.b);
  return 0;
}
```

**4.2** Write down, for each test case, the output in the space given. Your output should be a single integer. Assume `string.h stdio.h` are included.

| | |
|---|---|
| INPUT | abc<br>abc |
| OUTPUT | **0** |
| INPUT | abcd<br>abdc |
| OUTPUT | **2** |
| INPUT | mistake<br>mistoke |
| OUTPUT | **1** |
| INPUT | esc101rocks<br>rock |
| OUTPUT | **7** |

```
int hoo(int c1, int c2, int c3){
  int res = c1 < c2 ? c1 : c2;
  res = res < c3 ? res : c3;
  return res;
}
int foo(char *s1,char *s2,int m,int n){
  if (!m || !n)
    return m + n;
  if (s1[m - 1] == s2[n - 1])
    return foo(s1, s2, m - 1, n - 1);
  else{
    int c1 = 1 + foo(s1,s2,m - 1,n - 1);
    int c2 = 1 + foo(s1,s2,m - 1,n);
    int c3 = 1 + foo(s1,s2,m, n - 1);
    return hoo(c1, c2, c3);
  }
}
int main(){
  char s1[20], s2[20];
  scanf("%s\n%s",s1,s2);
  int m = strlen(s1), n = strlen(s2);
  printf("%d", foo(s1, s2, m, n));
  return 0;
}
```

**4.3** Using the structure `struct Node{ int data; struct Node *next; };` we create a singly linked list. The lists are given to you schematically in the input. For example, if the list given is 1→2→3→4→NULL, this means the head node stores 1 and the tail node stores 4. What will the given code print, when invoked with a pointer to the head node of the given linked lists?

```
void f1(struct Node* head){
  if(head == NULL) return;
  printf("%d", head->data);
  // No spaces in printf
  if(head->next != NULL)
    f1((head->next)->next);
  printf("%d", head->data);
  // No spaces in printf
}
```

| INPUT | 1→2→3→NULL |
|---|---|
| OUTPUT | **1331** |
| INPUT | 1→2→3→4→5→6→7→8→NULL |
| OUTPUT | **13577531** |

**4.4** Write down, for each test case, the output for the given input. The output has two lines. Assume that `stdio.h` has been included.

| INPUT | 5 10 |
|---|---|
| OUTPUT | **43** |
| | **80** |
| INPUT | 12 21 |
| OUTPUT | **68** |
| | **119** |

```
int funcg(int x){
  static int y = 10;
  return x + (++y);
}
int funcf(int x){
  return funcg(x);
}
int main(){
  int x, y, c;
  scanf("%d %d", &x, &y);
  for(c = 1; c <= 2; ++c){
    y += funcf(x);
    y += funcg(x);
    printf("%d", y);
    if(c == 1) printf("\n");
  }
  return 0;
}
```

**4.5** This question will use a linked list using the structure given on the right. On the next page, three functions are defined, in addition to the `main()` function. In order to present all functions to you in one page, the functions are presented in different boxes but they belong to the same C program.

```
typedef struct Node{
  int data;
  struct Node *next;
}Node;
```

The function `print()` prints the linked list with some nice formatting, `add()` adds a new node to the linked list at an appropriate location and `doOps()` does something nice to the linked list. The functions `add()` and `print()` and `doOps()` are all defined before the `main()` function in the C program. Write the output of the program in the space provided in the next page. Indicate space using a small gap **-** (**don't write a dot like Prutor**). `stdio.h` and `stdlib.h` are included

```
void add(Node **head, int x){
  Node *new =
(Node*)malloc(sizeof(Node));
  new->data = x;
  new->next = (*head);
  *head = new;
}
```

```
void print(Node *head, int nl){
  while(head != NULL){
    printf("%d", head->data);
    head = head->next;
    if(head != NULL) printf(" ");
  } // No trailing spaces
  if(nl) printf("\n");
}
```

```
void doOps(Node **head){
  Node *first, *second, *third, *fourth, *front;
  first = second = third = fourth = front = *head;
  if(*head != NULL){
    while(second != NULL && second->next != NULL){
      third = second->next;
      second = second->next->next;
      fourth = first;
      first = first->next;
    }
    printf("%d\n", first->data);
    if(second == NULL){
      *head = first;
      fourth-> next = front;
      Node *temp = first->next;
      first->next = front->next;
      front-> next = temp;
    }else{
      fourth->next = second;
      third->next = first;
      Node * temp = second->next;
      second->next = first->next;
      first->next = temp;
    }
  }
}
```

```
int main(){
  Node *head = NULL;
  add(&head, 4);
  add(&head, 3);
  add(&head, 22);
  add(&head, 5);
  doOps(&head);
  print(head, 1);
  add(&head, 6);
  doOps(&head);
  print(head, 0);
  return 0;
}
```

| OUTPUT |
| --- |
| **3** |
| **3 22 5 4** |
| **22** |
| **6 3 4 5 22** |

**Q5** In the following questions, you will be given either incomplete code or else incorrect code. Use the space provided to complete/correct the code, as the case may be. If you need to indicate a space, leave a small gap (**don't write a dot like Prutor**). (**8+8+8+6=30marks**)

**5.1** Complete the following program so that it creates a singly linked list 5→1→4→2→3→NULL, and then prints it reverse i.e. as "3 2 4 1 5" (without quotes). Do not worry about spaces at the end of the output line. Assume that `stdio.h` and `stdlib.h` are included. **The definition of the Node structure is given in question 4.5**.

```
Node* makeNode(int a){  // Create a new node to store a
    Node *temp = (Node*)malloc(sizeof(Node));
    temp->data = a;
    temp->next = NULL;  // Initialize
    return temp;
}
void print(Node *head){  // Print list recursively
    if(head == NULL)
        return;
    print(head->next);
    printf("%d ",head->data);
}
int main(){
    int a[5] = {5, 1, 4, 2, 3};
    Node *head = makeNode(a[0]);
    Node *runner = head;
    for(int i = 1; i < 5; i++){
        Node* temp = makeNode(a[i]);
        runner->next = temp;
        runner = temp;
        // runner = runner->next; // also correct
    }
    print(head);
    return 0;
}
```

**5.2** Complete the following program to create a binary search tree with a root storing the value 0 and then inserting five nodes into the tree. Recall that in a BST, all left descendants of a node must store values smaller than values stored at that node and all right descendants must store values larger than value at the node. Assume values being stored in the tree are unique. Also assume that **`stdio.h`** and **`stdlib.h`** have been included.

```
typedef struct node{
  int key;
  struct node* left;
  struct node* right;
}node;
node* makeNode(int a){  // Create a new node to store a
  node* temp = (node*)malloc(sizeof(node));
  temp->key = a;
  temp->left = temp->right = NULL;  // Initialize
  return temp;
}
void insert(int val, node* root){ // Recursive insert
  if(val < root -> key){  // Should val go left?
    if(root->left == NULL){  // Root does not have a left child
      node *temp = makeNode(val); // Now make it left child of root
      root->left = temp;
    }else{ insert(val, root->left); }
  }else{  // val should go into the right subtree
    if(root->right == NULL){
      node *temp = makeNode(val);// Make it right child of root
      root->right = temp;
    }else{ insert(val, root->right); }
  }
}
int main(){
  node* root = makeNode(0);
  int d;
  for(int i = 0; i < 5; i++){
    scanf("%d", &d);
    insert(d, root);  // Insert d into the tree
  }
  return 0;
}
```

**5.3** The following program uses binary search to search for a given element in an array which has elements sorted in increasing order such that no element repeats. If the element is present, its index is printed else -1 is printed. However, the code has logical errors. First tell us what output this incorrect program will give on the inputs given below. Then correct all errors.

```
1  #include <stdio.h>
2  int main(){
3    int a[10] = {1, 2, 3, 4, 5, 6, 7, 8, 9, 10};
4    int tofind, start = 0, end = 9, index = 0;
5    scanf("%d", &tofind);
6    while(start < end){
7      int mid = start + end / 2;
8      if(a[mid] == tofind){
9        index = mid;
10       break;
11     }else if(a[mid] > tofind){
12       start = mid + 1;
13     }else{
14       end = mid + 1;
15     }
16   }
17   printf("%d\n", index);
18   return 0;
19 }
```

| INPUT | -1 | 0 | 2 | 5 |
|---|---|---|---|---|
| OUTPUT | **0** | **0** | **0** | **4** |

| Line No | Corrected Code |
|---|---|
| **4** | **int tofind, start = 0, end = 9, index = -1; // error message is -1** |
| **6** | **while(start <= end){ // otherwise input = 1 will not work** |
| **7** | **int mid = (start + end) / 2; // otherwise segfault risk** |
| **11** | **}else if(a[mid] < tofind){** |
| **14** | **end = mid - 1;** |
| | **//Other combinations possible too** |
| DO NOT SUGGEST CORRECTIONS TO MORE THAN 6 LINES | |

**5.4** Recall that the quicksort algorithm requires a *partitioning* step which rearranges the array such that all elements smaller than or equal to the pivot element are on the left side of the pivot and all elements greater than the pivot are to the right of the pivot. In class we saw an algorithm that uses a separate array to solve this. Complete the following program that does this in an *in-place* manner i.e. does not use any additional arrays. It only uses a constant number of variables that does not depend on the length of the array being sorted. Assume `stdio.h` and `stdlib.h` are included.

**Hint**: the invariant used by this algorithm is the following. Suppose in the first q elements of the array, there are r elements that are smaller than or equal to the pivot. Then the algorithm ensures that after it has looked at these first q elements from left to right, those r elements are placed in the first r locations of the array (although not necessarily in any particular order).

```
void swap(int *a, int *b){  // Swap two integers
   int temp = *a;
   *a = *b;
   *b = temp;
}
// Partition arr of length len. The index of pivot is idx
// Return the new index of the pivot element after done
int partition(int *arr, int len, int idx){
   int pivot = arr[idx];  // Store value of pivot for easy comparison
   // First, swap pivot element with last element of the array for ease
   swap(&arr[idx], &arr[len-1]);
   // Invariant: Make sure that at all time, the array elements till index i
   // (including index i) are less than or equal to the pivot value.
   int i = -1;  // No small elements seen till now
   for(int j = 0; j < len - 1; j++){
      // If current element is smaller than or equal to pivot
      if(arr[j] <= pivot){  // If so, left region must be expanded
         i++;
         // Swap elements to bring the smaller element into the left region
         swap(&arr[i], &arr[j]);
      }
   }
   // Put the pivot element at its correct location
   swap(&arr[i + 1], &arr[len - 1]);
   return i+1;  // Return correct index of pivot
}
```