BIRLA INSTITUTE OF TECHNOLOGY & SCIENCE, PILANI (RAJ.)
Second Semester 2017-2018

Date: 11-May-2018	**CS F111 Computer Programming**	Max. Marks: 75 (37.5%)
Time: 3 PM	**COMPREHENSIVE EXAMINATION SOLUTIONS**	Duration: 3 hours

Duration: 2 hours	**PART-B (Open-book)**	Max. Marks: 50 (25%)

1. $(-11.375)_{10} = (-1011.011)_2$
Normalized form = -1.011011×2^3
Biased exponent = 3 + 1023 = 1026 **[2]**

(i) 1 10000000010 011011000… (46 zeros) **[4]**

(ii) C026 C000 0000 0000 **[1]**

2. *Translation question*:

(i)
```
scanf("%[AUCGaucg]", seq);
```
[2]

(ii)
```
void convertSequence(char *ptr);
{
  for ( ; *ptr ; ptr++)      /* till the end of string is reached... */
    switch (*ptr)
    {
        case 'A': case 'a': *ptr = '0';  /* substitute 'A' or 'a' with '0' */
                            break;
        case 'U': case 'u': *ptr = '1';
                            break;
        case 'C': case 'c': *ptr = '2';
                            break;
        case 'G': case 'g': *ptr = '3';
                            break;
    }
  return;
}
```
Alternatively, an array-based implementation can also be used. Instead of `switch`, one can also use nested `if...else` construct.
Mark distribution: Use of loop construct – 1 mark, checking both lower and upper cases – 1 mark, changing the array element to a numeric constant – 1 mark, correctness of `switch`/`if` construct – 1 mark. **[4]**

(iii)
```
void translateSequence(char seq[], char table[][4][4] )
{
  char *locn;
  short int i1, i2, i3;   /* to store array indexes */
  locn = strstr(seq,"013");
  if (!locn) {
    printf("No ORF found.\n");
    return;
   }
  do
    {
      if (*loc) i1 = *loc++ - '0'; else break;
      if (*loc) i2 = *loc++ - '0'; else break;
      if (*loc) i3 = *loc++ - '0'; else break;
      printf(" %c ", table[i1][i2][i3]);
      if (table[i1][i2][i3] == ' ') break;   // encountered stop codon
    } while (*loc);
  return;
}
```
[1] **[2]** **[3]** **[1]** **[1]** **[1]** **[1]**

(iv) One can use a 4d array that can store strings: `char table[4][4][4][MAX_NAME_LEN];`
Or, a 3d array of character pointers (that can each point to a string): `char * table[4][4][4];`

Mark distribution: 1 mark for correct declaration + 1 mark for brief explanation **[2]**

3.

```
#include <stdio.h>
#include <string.h>
typedef enum {FALSE, TRUE} BOOL;                                          [1]
BOOL isPalinRecur(char *start, char *end)
{
  if (start >= end)  /* first half of the string was compared with the second
                        half and all characters matched, or it is a single-
                        character string ... */                           [2]
       return TRUE;   /* hence it is a palindrome */
if (*start != *end)   /* a mismatch found */                              [1]
       return FALSE;
return (isPalinRecur(start+1, end-1));                                    [2]
}

int main()
{
  char str[100];
  int len;
  scanf("%[^\n]",str);
  if (isPalinRecur(str,&str[strlen(str)-1]) == TRUE)                      [1]
    printf("%s is a palindrome.\n", str);
  else
    printf("%s is not a palindrome.\n", str);
  return 0;
}
```

4.

(i) Corrections are provided in red lettering:

```
 1 void addStudent(char *name, char *cc, float marks)
 2 {
 3   STUDENT *new;
 4   new = malloc(sizeof(STUDENT *));      // sizeof(STUDENT)              [1]
 5   new->next = NULL;
 6   new->subj_next = NULL;
 7   new->name = name;          // strcpy(new->name, name);               [½]
 8   new->course_code = cc;     // strcpy(new->course_code, cc);          [½]
 9   new->marks= marks;
10   new->next = list;
11   new = list;                // list = new;                            [1]
12   if(!strcmp(new->course_code, "EG1"))
13   {
14     eg_list = new;                 // new->subj_next = eg_list;        [2]
15     new->subj_next = eg_list;      // eg_list = new;
16   }
17   else
18   if(!strcmp(new->course_code, "WP1"))
19   {
20     wp_list = new;                 // new->subj_next = wp_list;        [1]
21     new->subj_next = wp_list;      // wp_list = new;
22   }
23 }
```

Note: If any other piece of code was changed to an erroneous one, marks will be deducted.

(ii)
```
void printList(STUDENT *ls, int list_num)
{
  STUDENT *tmp;
  if(list==NULL)                                                    [1]
  {
    printf("List is empty");
    return;
  }
  tmp = ls;
  while (tmp)                                          //loop       [1]
  {
    printf("\n%s %s %f ", tmp->name, tmp->course_code, tmp->marks);  [1]
    switch(list_num)                                   //checking   [1]
    {
     case 0:      tmp = tmp->next;                     // entire list  [1]
                  break;
     case 1:      tmp = tmp->subj_next;        // subject-specific list  [1]
                  break;
    }
  }
  printf("\n");
}
```

(iii)
```
void makeCircularList()
{
  STUDENT *tmp1 = eg_list;
  STUDENT *tmp2 = wp_list;
  if (tmp1 == NULL && tmp2 == NULL)       // both sub-lists are empty   [1]
    return;
  if (tmp1 != NULL && tmp2 == NULL)       // only EG1 sub-list is present
  {
    while(tmp1->subj_next != NULL)        // traverse till the end      [1]
      tmp1 = tmp1->subj_next;
    tmp1->subj_next = eg_list;            // make last node point to first  [1]
  }
  else
  if (tmp1 == NULL && tmp2 != NULL)       // only WP1 sub-list is present  [1]
  {
    while(tmp2->subj_next != NULL)
      tmp2 = tmp2->subj_next;
    tmp2->subj_next = wp_list;
  }
  else
  {
    while(tmp1->subj_next != NULL)
      tmp1 = tmp1->subj_next;
    tmp1->subj_next = wp_list;            // making the last node of EG1 sub-list
                                          // point to WP1 sub-list      [1]
    while(tmp2->subj_next != NULL)
      tmp2 = tmp2->subj_next;             // WP1 sub-list leads to EG1  [1]
    tmp2->subj_next = eg_list;
  }
}
```