Computer Programming — Quiz 2

Instructor: Girish Varma • Course Code: CS0.101 • IIIT Hyderabad

Problems are in comments, highlighted.

1 Enumerating Arrangements of k items from n objects.

Tips for solving: Look at the simplest cases and try to work out the pseudo code for a recursive algorithm. Some of the functions defined can give clues as to what is to be done.

```
/* Consider all arrangements of k items
   from n objects. For $n = 3, k = 2$, they
   are 12,21,13,31,23,32. The number
   of such arrangements is
   $^nP_k = n(n-1)\cdots(n-k+2)(n-k+1)$.
   Bellow is a program which when given n,k
   as input, prints all arragements of k
   items from n objects.                 */
#include <stdio.h>
#include <stdlib.h>
#include <stdbool.h>
typedef int** PermList;

int count_arrangements(int n, int k) {
  // Problem 1 a.) write a recursive
  // function logic here in one line to
  // compute the number of all arragements
  // of k items from n objects. (2 marks)
}

PermList create_perm_list(int n,
                                  int k) {
  int fn = count_arrangements(n, k);
  PermList pl=malloc(fn*sizeof(int *));
  for(int i =0; i < fn ;i++) {
    pl[i] = malloc(n*sizeof(int));
  }
  return pl;
}
void destroy_perm_list(PermList pl,
                       int n, int k) {
  int fn = count_arrangements(n, k);
  for (int i =0; i < fn ;i++) {
    free(pl[i]);
  }
  free(pl);
}

// given a `small_row` of size `size`
// copies it to `big_row` which has size
// `size+1`. Also sets the last position
// in `big_row` to `e`
void insert_and_copy(int* small_row, int size,
                     int e, int* big_row) {
  for (int i = 0; i < size; i++) {
    big_row[i] = small_row[i];
  }
  big_row[size] = e;
}

// checks if `e` is in the `row` of size `size`
bool find(int e, int* row, int size) {
  for(int i = 0; i < size; i++) {
    if (row[i] == e) {
      return true;
    }
  }
  return false;
}
// find the numbers from {1,...,n}
// that are not in `row` which is of size `k`
// and puts them in `elements`
void find_elements_not_in_row(int* row, int n,
                              int k,
                              int* elements) {
  int c = 0;
  for (int i = 0; i < n; i++) {
    if (find(i+1, row, k) == false) {
      elements[c++] = i+1;
    }
  }
}

PermList enumerate_arrangements(
                          int n, int k) {
  PermList B = create_perm_list(n,k);
  if (k == 1) {
    // Problem 1 b.) write code here for base
    // case of recursively building lst `B`
    // of all arrangements of $k = 1$ items
    // from {1,..,n}. (3 marks)
  } else {
    // Problem 1 c.) write code here for
    // recursively building list `B` of all
    //  arrangements of k items from
    // {1,..,n}. (5 marks)
  }
  return B;
}
void print_perm_list(PermList pl,
                     int n, int k) {
  int fn = count_arrangements(n, k);
  for(int i = 0; i < fn;i++) {
    for (int j =0; j <k; j++) {
      printf("%d ", pl[i][j]);
    }
    printf("\n");
  }
}

int main() {
  int n = 10;
  int k = 5;
  print_perm_list(
     enumerate_arrangements(n, k),n,k);
  return 0;
}
```

2 Banking on Structs

```c
/* Build a program for managing a bank.
   There should be a database of bank
   accounts and transactions. We should
   be able to add new accounts,
   new transactions (credit/debit) and
   compute the balance of a account      */
#include <stdio.h>
#include <string.h>

typedef enum AccountType {
  Savings,
  Current
} AccountType;

typedef enum TransactionType {
  Credit,
  Debit
} TransactionType;

typedef struct Transaction {
  TransactionType type;
  struct BankAccount* account;
  int amount;
} Transaction;

typedef struct BankAccount {
  char name[100];
  int pin;
  AccountType type;
  // passbook is an array of transactions
  // pointers to avoid taking too much memory
  struct Transaction* passbook[1000];
  int transactions_count;
} BankAccount;

typedef struct BankDatabase {
  BankAccount accounts[1000];
  Transaction transactions[10000];
  int accounts_count;
  int transactions_count;
} BankDatabase;

// compute the total amount of money
// with the bank amoung all the accounts
int compute_money_with_bank(
                       BankDatabase* db) {
  int sum = 0;
  for(int i = 0;
        i < db->transactions_count; i++) {
    switch(db->transactions[i].type) {
      case Credit:
        sum += db->transactions[i].amount;
        break;
      case Debit:
        sum -= db->transactions[i].amount;
        break;
    }
  }
  return sum;
}


int compute_balance(BankAccount* acc) {
  // Problem 2 a.) fill in the code to
  // find the balance of the account
  // 'acc'.(3 marks)
}

BankAccount* add_bank_account(char* name,
                   int pin, AccountType type,
                   BankDatabase* db) {
  // Problem 2 b.) fill in the code to add
  // a new account 'acc' to the bank
  // database 'db'. The function should
  // also return a pointer to the bank
  // account created in 'db'. (3 marks)
}

Transaction* add_transaction(
            TransactionType type,
            BankAccount *account,
            int amount, BankDatabase* db) {
  // Problem 2 c.) Fill in the code for
  // adding a transaction to the system.
  // The logic should be written such
  // that the all the other functions in
  //  this program continue to work
  // correctly. (6 marks)
}

int main() {

  BankDatabase db;
  db.accounts_count = db.transactions_count = 0;
  BankAccount acc = { .pin = 1234,
                      .transactions_count = 0};
  strcpy(acc.name, "Ivan");
  BankAccount* acc_ptr = add_bank_account(
                 acc.name,acc.pin,acc.type, &db);
  add_transaction(Credit, acc_ptr, 10000, &db);
  add_transaction(Debit, acc_ptr, 2000, &db);
  add_transaction(Credit, acc_ptr, 5000, &db);

  // should print 13000
  printf("Account balance is %d\n",
         compute_balance(acc_ptr));

  BankAccount acc2 = { .pin = 6897,
                       .transactions_count = 0};
  strcpy(acc2.name, "Jake");
  BankAccount* acc_ptr2 = add_bank_account(
             acc2.name, acc2.pin,acc2.type, &db);
  add_transaction(Credit, acc_ptr2, 100000, &db);
  add_transaction(Debit, acc_ptr2, 20000, &db);
  add_transaction(Credit, acc_ptr2, 50000, &db);

  // should print 130000
  printf("Account balance is %d\n",
         compute_balance(acc_ptr2));

  // should print 143000
  printf("Total Money with bank is %d\n",
         compute_money_with_bank(&db));
}
```