POLITECNICO
MILANO 1863

DIPARTIMENTO DI ELETTRONICA
INFORMAZIONE E BIOINGEGNERIA

CORSO DI LAUREA IN INGEGNERIA AEROSPAZIALE

Prof. Gerardo Pelosi

Informatica – a.a. 2020/2021– 2º Appello – 15 Luglio 2021

Cognome _____ **Matricola o Cod. Persona** _____

Nome_____ **Firma** _____

Istruzioni

- Non separate questi fogli. Scrivete la soluzione **solo sui fogli distribuiti**, utilizzando il retro delle pagine se necessario. **Cancellate le parti di brutta** (o ripudiate) con un tratto di **penna**.
- Ogni parte non cancellata a penna sarà considerata parte integrante della soluzione.
- **È possibile scrivere a matita** e non occorre ricalcare al momento della consegna.
- È **vietato** utilizzare **calcolatrici** e qualsiasi **dispositivo elettronico**. Chi tenti di farlo vedrà **annullata** la sua prova.
- È **vietato** consultare **libri o appunti**. Chi tenti di farlo vedrà **annullata** la sua prova.
- Qualsiasi **tentativo** di comunicare con altri studenti comporta **l'espulsione** dall'aula.
- Non è possibile lasciare l'aula conservando il tema della prova in corso.
- È possibile ritirarsi senza penalità.
- Tempo a disposizione: **2h:15**

Valore indicativo degli esercizi, voti parziali e voto finale:

Esercizio 1 (7 punti) _____

Esercizio 2 (10 punti) _____

Esercizio 3 (8 punti) _____

Esercizio 4 (8 punti) _____

Totale (33 punti) _____

Voto finale _____

Esercizio 1 Codifiche numeriche e algebra di Boole [7 punti]

1. Si stabilisca il minimo numero di bit sufficiente a rappresentare in complemento a due i valori numerici seguenti: A = **4E$_{hex}$** (numero positivo in notazione esadecimale), B = **321$_{OCT}$** (numero positivo in notazione ottale) e C = **-21$_{dec}$** (numero negativo in notazione decimale).
   (a) Si esibisca la conversione di A, B, C in *complemento a due*, con un numero di bit che sia quello minimo per rappresentarli tutti e tre.
   (b) Si esibisca il risultato della differenza (C-A) e della somma (B+A) svolte considerando gli operandi codificati in in *complemento a due*, e si indichi se nell'esecuzione di ciascuna operazione si genera un riporto oltre la posizione del bit di segno e se si verifica *overflow*.
   [5 punti]

   Si indichi se le espressioni condizionali dei seguenti costrutti di selezione (in linguaggio C) facenti uso delle variabili intere a, b sono equivalenti oppure no, giustificando la risposta **[2 punti]**

```
if ( a > 10 || ( b > 5 && a <= 10 ) ){
      ...
}

if ( !(a <= 10 && b <= 5) ){
      ...
}
```

Soluzione

A = **4E$_{hex}$** = 100 1110$_{bin}$ = 0100 1110$_{c2}$ ci vogliono almeno 8 bit
B = **321$_{oct}$** = 11 010 001$_{bin}$ = 011 010 001$_{c2}$ ci vogliono almeno 9 bit
C = **-21$_{dec}$** = 1 + not(010101$_{bin}$) = 101011 $_{c2}$ ci vogliono almeno 6 bit

Quindi con 9 bit si ha:
A = 001001110$_{c2}$ B = 011010001$_{c2}$ C = 111101011$_{c2}$

C-A = C+(-A) = (111101011$_{c2}$) +
(110110010 $_{c2}$) =
---
[1]110011101 SI bit di riporto oltre la cifra più significativa
NO *overflow*

B+A = B+A = (011010001$_{c2}$) +
(001001110$_{c2}$) =

100011111 NO bit di riporto oltre la cifra più significativa
SI *overflow*

Con la seguente sostituzione di variabili `A = (a > 0), B = (b>5)`, le espressioni booleane dei due costrutti **if**, si possono ri-leggere/ri-scrivere come:

```
A or (B and not(A))
not( not(A) and not(B) )
```

Esse sono equivalenti perché hanno la stessa tavola di verità oppure perché applicando la legge distributiva dell'`or` rispetto all'`and` alla prima espressione si ottiene: `(A or B)and(A or not(A))=(A or B);` applicando DeMorgan alla seconda espressione, si ottiene di nuovo: `A or B.`

Esercizio 2 (10 punti)

(a) Scrivere una funzione di prototipo:

```
int controlla_permutazione(int p[], int lung)
```

che abbia come parametri un *array* di interi e la sua lunghezza e che restituisca 1 se l'*array* contiene i valori di una permutazione valida, 0 altrimenti.
Un *array* di interi contiene una permutazione valida se i valori in ogni cella sono tutti e soli i numeri che possono rappresentare la posizione di una cella dell'*array* stesso.

Esempio:

indice: `0 1 2 3 4 5`
array p: `[ 1, 3, 2, 4, 5, 0 ]`

(b) Scrivere una funzione

```
int applica_permutazione(char str[], int p[], int lung)
```

che abbia come parametri una stringa, un array di interi e un intero indicante la lunghezza dell'array di interi. Se *l'array* di interi rappresenta una permutazione valida e ha la stessa lunghezza della stringa, la funzione applica la permutazione ai caratteri della stringa indicata come parametro e restituisce 1, altrimenti 0.

Esempio:

`str: "AIRONE"`
array p: `[ 1, 3, 2, 4, 5, 0 ]`
`lung: 6`

dopo l'esecuzione della funzione `str: "AIRONE"` diventa
`str: "EARION"`

N.B: è consentito/raccomandato far uso delle funzioni di libreria in `<string.h>` e `...malloc(...)`

Se si continua sul retro di qualche foglio, indicare quale

Soluzione (a)

```
int controlla_permutazione(int p[], int lung) {

     if (lung <= 0) return 0;
     for (int idx = 0; idx < lung; idx++) {

        if (p[idx] < 0 || p[idx] > lung-1) return 0;

        for (int i = 0; i < lung; i++)
            if (i != idx && p[idx] == p[i]) return 0;
     }
     return 1;

} // end controlla_permutazione
```

Soluzione (b)

```
int applica_permutazione(char str[], int p[], int lung) {

     if (lung != strlen(str) || controlla_permutazione(p,lung) == 0)
       return 0;

     char* temp = (char*) malloc(sizeof(char)*(lung+1));
     if (temp == NULL) return 0;

     strcpy(temp, str);
     for (int i = 0; i < lung; i++) {
        str[p[i]] = temp[i];
     }

     free(temp);

     return 1;

} // end applica_permutazione
```

Se si continua sul retro di qualche foglio, indicare quale

Esercizio 3 (8 punti)

Si considerino le seguenti definizioni di costanti e tipi di dato:

```
#define MAX_RIGHE    30
#define MAX_COLONNE  50

typedef struct {
                  int r;
                  int c;
        } cella_t;
```

Esibire il prototipo e il codice di un sottoprogramma

```
...trova_ciclo(...)
```

che prenda come primo parametro un *array* bidimensionale in cui ogni cella sia di tipo `cella_t`, come secondo parametro il numero di righe effettivo della matrice indicata dal primo argomento (dovrà essere un valore <= `MAX_RIGHE`) e come terzo parametro il numero di colonne effettivo della matrice (dovrà essere un valore <= `MAX_COLONNE`).

Il sottoprogramma deve acquisire da tastiera una coppia di coordinate della matrice (controllando che siano valide) e restituire 1 se a partire dalla cella con le coordinate indicate è possibile seguire sulla matrice un percorso ciclico, 0 altrimenti.

Nota: le coordinate nella prima cella individueranno una seconda cella, che a sua volta conterrà delle coordinate. Le coordinate in quest'ultima individueranno una terza cella e così via.
Il sottoprogramma deve stabilire se procedendo in tal modo è possibile ritornare sulla cella di partenza (restituendo 1) oppure no (restituendo 0).

Soluzione

```
int trova_ciclo( cella_t mat[][MAX_COLONNE], int nr, int nc ) {

     if ( nr <= 0 || nr > MAX_RIGHE || nc <= 0 || nc > MAX_COLONNE )
       return 0;

     int cond, x, y;

     do {
       printf("\n Inserisci coordinate x, y di una cella della matrice\n");
       printf("\n 0 <= x < %d, 0 <= y < %d", nr, nc);
       printf("\n x = "); scanf("%d", &x);
       printf("\n y = "); scanf("%d", &y);

       cond = ((x >= 0 && x < nr) && (y >=0 && y < nc));

     } while (cond == 0);


     int trovato = 0; // var. booleana per indicare se il ciclo è stato
                      // individuato (== 1) o meno (== 0).

     int cont = 0;    // contatore del numero di celle
                      // visitate sulla matrice (non può essere > nr*nc)

     int currX = x, currY = y;

     while (cont < nr*nc && trovato == 0) {

         nextX = mat[currX][currY].r;
         nextY = mat[currX][currY].c;
         if ( !(nextX >= 0 && nextX < nr && nextY >=0 && nextY < nc) )
           return 0;

         cont += 1;

         trovato = (nextX == x && nextY == y);
         currX = nextX;
         curry = nextY;

     } // end while

     if (trovato == 1 ) return 1;
     return 0;

} // end trova_ciclo
```

Esercizio 4 (8 punti)

Si desidera analizzare la statistica dei consumi di *toner* di un'azienda per ottimizzare gli acquisti futuri, facendo uso delle costanti e del tipo di dato di seguito specificati.

```
#define MAX_STR 30
#define MAX_NUMERO 6
typedef struct {
            char nomeDip[MAX_STR+1];
            int  occorrenze[MAX_NUMERO];
        } stat;
```

La quantità di cartucce di *toner* prelevate dal magazzino è riportata all'interno di un *file* di testo.
Il *file* contiene una riga per ogni ordine fatto da ciascun dipartimento. Ogni riga contiene in sequenza:

- il nome del dipartimento che ha prelevato il/i *toner* (una stringa lunga `MAX_STR` caratteri);
- un numero intero (valore minimo 1 e massimo `MAX_NUMERO`) che indica la quantità di cartucce di *toner* prelevate da quel dipartimento.

Non è noto il numero di righe presenti nel *file*.

Esempio di formato del *file*:

```
ACQUISTI 1
VENDITE 2
ACQUISTI 1
ACQUISTI 5
```

(a) Si esibisca il codice del sottoprogramma

```
void riempi_vett(char nomefile[], stat vett[], int lungVett)
```

che richiede come primo parametro il nome del *file* di testo contenete le informazioni sugli ordini delle cartucce di *toner*, come secondo parametro un *array* le cui celle sono di tipo `stat` e come terzo parametro la lunghezza dell'*array* stesso.
Il sottoprogramma assume che una stringa nell'attributo `nomeDip` in ciascuna cella dell'*array* `vett[]` sia già presente e che sia diversa da quelle nelle altre celle, inoltre il valore di tutti gli interi nell'attributo `occorrenze` di ogni cella è assunto essere inizializzato a 0.
Il sottoprogramma modifica l'attributo `occorrenze` di ciascuna cella, memorizzando in `occorrenze[0]` quante righe del *file* contengono 1 come quantità di *toner* richiesta dal dipartimento indicato nell'attributo `nomeDip`, `in occorrenze[1]` quante righe del *file* contengono 2 come quantità di *toner* richiesta dal dipartimento indicato nell'attributo `nomeDip` e così via.

Soluzione

```
void riempi_array(char nomefile[], stat vett[], int lungVett) {
    FILE* fpr = fopen(nomefile, "r");
    if (fpr == NULL) {
      fprintf(stderr, "\n Errore in apertura del file %s", nomefile);
      exit(-1);
    }
    char nomeDip[MAX_STR+1];
    int qta;
    while ( feof(fpr)  == 0) {
      fscanf(fpr, "%s %d", nomeDip, &qta);
      for (int i = 0; i < lungVett; i++) {
         if ( strcmp( nomedip, dipart ) == 0 ) vett[i].occorrenze[qta-1] += 1;
      } // end for
    } // end while

    fclose(fpr);
} // end riempi_array
```

Se si continua sul retro di qualche foglio, indicare quale

(b) Si esibisca il codice del sottoprogramma

```
void toner_statistic(stat vett[], int lungVett)
```

Il sottoprogramma assume come primo parametro un vettore di statistiche creato dal sottoprogramma al punto (a) . Esso crea un nuovo *file* chiamato `"statistic.txt"` con tante righe quante sono le celle dell'*array* `vett` e, su ogni rigo, riporta: il nome del dipartimento, la minima, la massima e la media quantità di *toner* che è stata richiesta negli ordini emessi dal dipartimento, il valore della quantità di *toner* richiesta più spesso (la moda) dal dipartimento e il numero di volte che tale valore si è ripetuto - in caso ci sia più di un valore moda, è sufficiente riportarne solo uno.
Se un dipartimento non ha effettuato ordini di toner, il sottoprogramma riporterà sul rigo del *file*: `"Il Dip. non ha ordinato toner!"`

Esempio: se l'*array* `vett` avesse lunghezza 2 e contenesse i seguenti dati
nella cella[0]: {`"ACQUISTI"`, { 2, 1, 0, 0, 1, 0 } }
nella cella[1]: {`"VENDITE"`, { 0, 1, 0, 0, 0, 0 } }
Il *file* `"statistic.txt"` creato dal sottoprogramma sarebbe:

```
ACQUISTI 1, 5, 2.25, (1, 2)
VENDITE 2, 2, 2.00, (2, 1)
```

Soluzione

```
void toner_statistic(stat vett[], int lungVett) {

    FILE* fpw = fopen(nomefile, "w");
    if (fpw == NULL) {
       fprintf(stderr, "\n Errore in creazione del file %s", "statistics.txt");
       exit(-1);
    }

    for (int i = 0; i < lungVett; i++) {

        int min = 0, max, modaVal = -1, maxOccorrenze = 0, cont = 0;
        float media;
        for (int j = 0; j< MAX_NUMERO; j++) {
           if (min == 0 && vett[i].occorrenze[j] > 0) min = j+1;

           if (vett[i].occorrenze[j] != 0)  max = j+1;

           media +=  vett[i].occorrenze[j] * (j+1);
           cont + = vett[i].occorrenze[j];

           if (maxOccorrenze < vett[i].occorrenze[j]) {
             maxOccorrenze = vett[i].occorrenze[j];
             modaVal = j+1;
          } // end for
        } // end for
        if (cont != 0)
          fprintf(fpw,"%s %d %d %f (%d, %d)\n", vett[i].nomeDip,
                        min, max, media/cont, modaVal, MaxOccorrenze);
        else
          fprintf(fpw,"Il Dip. %s non ha ordinato toner!\n", vett[i].nomeDip);

     } // end for

     fclose(fpw);

     return;
} // end toner_statistic
```

Se si continua sul retro di qualche foglio, indicare quale