	Politecnico di Milano Facoltà di Ingegneria Industriale **FONDAMENTI DI INFORMATICA** Prima prova in itinere - 27 Novembre 2015		COGNOME E NOME
	RIGA	COLONNA	MATRICOLA
			Spazio riservato ai docenti

- Il presente plico contiene 5 esercizi e **deve essere debitamente compilato con cognome e nome, numero di matricola**.
- Il tempo a disposizione è di 1 ore e 45 minuti.
- Non separate questi fogli. Scrivete la soluzione solo sui fogli distribuiti, utilizzando il retro delle pagine in caso di necessità. Cancellate le parti di brutta 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 telefoni o pc**. Chi tenti di farlo vedrà annullata la sua prova.
- È ammessa la consultazione di libri e appunti.
- **Qualsiasi tentativo di comunicare con altri studenti comporta l'espulsione dall'aula**.
- È possibile ritirarsi senza penalità.
- **Non è possibile lasciare l'aula conservando il tema della prova in corso**.

Esercizio 1 (4 punti)

Si implementi una funzione ***conta*** che:

- riceve in ingresso due stringhe, ***str1*** e ***str2***
- restituisce il numero di volte che la stringa ***str1*** compare nella stringa ***str2***

Esempio: La chiamata `conta("va","provaprovava")` restituisce 3 perché la sottostringa `"va"` comprare tre volte in `"pro**va**pro**vava**"`

Note: *Si implementi la funzione conta, ipotizzando che le stringhe str1 ed str2 contengano il carattere terminatore.*

Soluzione

```
#include<stdio.h>
#include<string.h>

int conta(char str1[], char str2[]);

int main()
{
  char s1[]="va";
  char s2[]="provaprovava";


  printf ("La sottostringa %s compare %d volte in %s\n",s1,conta(s1,s2),s2);

  return 0;
}

int conta(char str1[], char str2[])
{
  int i,j,count=0;
  for (i=0; i<=strlen(str2)-strlen(str1); i++)
  {
    int trovato=1;
    for (j=0; j<strlen(str1) && trovato==1; j++)
      if (str2[i+j] != str1[j])
        trovato = 0;
    count = count + trovato;
  }
  return count;
}
```

Esercizio 2 (3 punti)

Si consideri il seguente programma C

```
#include <stdio.h>
int fun(int a[], int *b, int c);

int main()
{
 int i,b;
 int a[5]={4,5,3,7,3}; //<---

 b = fun(a,a+2,5);

 for (i=0; i<5; i++)
      if (i==2 && b<=0)
            printf ("%d ",a[i]);
      else
            printf ("+%d ",a[i]);

 return 0;
}

int fun(int a[], int *b, int c)
{
 int i;
 *b=0;
 for(i=0; i<b-a; i++)
      *b=*b-a[i];
 for(i=b-a+1;i<c;i++)
      *b=*b+a[i];
  return *b;
}
```

a) Cosa viene stampato a video dal programma?

b) Come cambia la risposta precedente, nel caso in cui la riga di codice indicata con una freccia sia modificata in:
`int a[5]={4,5,3,5,3};`

c) Spiegare sinteticamente il significato dei parametri in ingresso della funzione `fun` e che cosa fa la funzione.

Soluzione

a) +4 +5 +1 +7 +3
b) +4 +5 -1 +5 +3
c) La funzione fun riceve tre parametri: un vettore a, un puntatore b che punta ad una posizione di a, e un intero c che contiene la dimensione del vettore passato come primo parametro. La funzione azzera l'elemento puntato da b, gli sottrae tutti gli elementi di a precedenti alla posizione puntata da b e gli somma tutti gli elementi di a successivi alla posizione puntat a b; infine ritorna l'elemento puntato da b.

Esercizio 3 (2 punti)

a) Codificare +16 e -3, utilizzando la codifica in complemento a 2 e usando il minore numero possibile di bit necessario a rappresentare entrambi gli operandi.
b) Eseguire la somma fra +16 e -3 in complemento a 2. Il risultato è corretto o si verifica un overflow? (Giustificare la risposta).

Soluzione

a) L'operando che richiede più bit per essere codificato è +16 → sono necessari 6 bit
$+16_{10}$ → 010000_{CPL2}
-3_{10} → equivalente a codificare 2^6-3 come intero positivo → 111101_{CPL2}

b)

```
  (1)
   010000
   111101
 ----------
(1)001101        (13_10)
```

Non si può verificare un overflow, dato che gli operandi hanno segno discorde

Esercizio 4 (4 punti)

Si supponga di dover rappresentare in C un archivio contenente informazioni sulle stelle. Per ogni stella, occore rappresentare: nome della stella (massimo 30 caratteri), massa della stella (espressa come rapporto rispetto alla massa solare), distanza dal sistema solare (espresso in anni luce) ed il numero di pianeti che appartengono al suo sistema (massimo 50); inoltre, per ciascun pianeta, occorre anche memorizzare il nome del pianeta (massimo 30 caratteri), la gravità del pianeta e il periodo di rivoluzione (espresso in giorni).

a) Si dichiari un tipo di dato adatto a rappresentare tutte le informazioni necessarie a rappresentare una stella
b) Si dichiari una variabile *archivio* in grado di contenere le informazioni di 1000 stelle
c) Si scriva un frammento di codice C che individui, fra tutti i pianeti *abitabili* (cioè con gravità minore di 30 e con periodo di rivoluzione compreso fra 100 e 900), quello che si trova nel sistema più vicino al sistema solare; quindi stampi a video il nome di questo pianeta ed il nome della stella attorno a cui gravita.

*Note: non è necessario inizializzare o leggere i dati della variabile archivio, ma si può ipotizzare che essa contenga **già** i dati relativi alle 1000 stelle su cui bisogna effettuare la ricerca descritta al punto C. Se ci fosse più di un pianeta abitabile a distanza minima dal sistema solare, è sufficiente stampare i dati di **uno soltanto** di questi pianeti (e della sua stella). Se non si trova alcun pianeta abitabile, occorre invece comunicarlo con un messaggio.*

Soluzione

```
typedef struct
{
  char nome[30];
  float gravita;
  float rivoluzione;
} pianeta;

typedef struct
{
  char nome[30];
  float massa;
  float distanza;
  int np;
  pianeta pianeti[50];
} stella;


stella archivio[1000];
float dist_min;
int i,j,imin,jmin,trovato=0;

for (i=0; i<1000; i++)
  for (j=0;j<archivio[i].np; j++)
  {
    pianeta p = archivio[i].pianeti[j];
    if (p.rivoluzione >= 100 && p.rivoluzione <= 900 && p.gravita < 30
              && (trovato==0 || archivio[i].distanza < dist_min) )
    {
        dist_min = archivio[i].distanza;
        imin = i;
        jmin = j;
        trovato = 1;
    }
  }
if (trovato==1)
{
  printf("Il pianeta abitabile più vicino è %s ", archivio[imin].pianeti[jmin].nome);
  printf(" nella stella %s\n",archivio[imin].nome);
}
else
    printf("Non esiste alcun pianeta abiabile in archivio");
```

Esercizio 5 (3 punti)

Si consideri la seguente espressione booleana:

NOT (A AND NOT B) AND (NOT B OR C)

Si compili la seguente tabella della verità (in cui 0 rappresenta il valore logico FALSO, 1 il valore VERO):

A	*B*	*C*	*NOT B*	*A AND NOT B*	*NOT (A AND NOT B)*	*NOT B OR C*	*NOT (A AND NOT B) AND (NOT B OR C)*
0	0	0					
0	0	1					
0	1	0					
0	1	1					
1	0	0					
1	0	1					
1	1	0					
1	1	1					

Si consideri ora la condizione, scritta in linguaggio C, in cui x e y siano due variabili int:

!((y>5) && !(x>2)) && (!(x>2) || (x<3))

ottenuta dalla prima formula sostituendo la variabile A con y>5, la variabile B con x>2, la variabile C con x<3
Si risponda alle seguenti domande:

a. L'espressione e' vera o falsa quando x=1 e y=7? (giustificare la risposta)
b. Se y>5, per quali valori di x l'espressione e' vera? (giustificare la risposta)

Soluzione

A	*B*	*C*	*NOT B*	*A AND NOT B*	*NOT (A AND NOT B)*	*NOT B OR C*	*NOT (A AND NOT B) AND (NOT */B OR C)*
0	0	0	1	0	1	1	1
0	0	1	1	0	1	1	1
0	1	0	0	0	1	0	0
0	1	1	0	0	1	1	1
1	0	0	1	1	0	1	0
1	0	1	1	1	0	1	0
1	1	0	0	0	1	0	0
1	1	1	0	0	1	1	1

a) Per x=1 e y=7, abbiamo A=1, B=0, C=1 per cui, dalla tabella della verità l'espressione è falsa
b) Se y>5 allora A=1. In questo caso, l'espressione è vera quando B e C sono vere, quindi per 2<x<3