	Politecnico di Milano Scuola di Ingegneria Industriale e dell'Informazione **FONDAMENTI DI INFORMATICA** Appello 19 Febbraio 2018		COGNOME E NOME
	RIGA	COLONNA	CODICE PERSONA

Spazio riservato ai docenti

- Il presente plico contiene 5 esercizi e **deve essere debitamente compilato con cognome e nome, codice persona**.
- Il tempo a disposizione è di 1 ora 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.
- **Non 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 (8 punti)

Si implementi la seguente funzione:

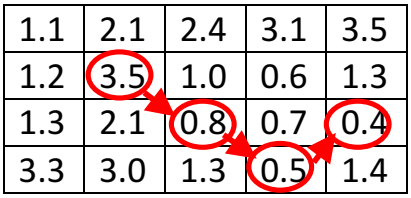
`float piccolo(float M[R][C], int i, int j)`

la funzione restituisce un elemento *piccolo* della matrice in ingresso `M[R][C]` (dove `R` e `C` sono il numero di righe e colonne della matrice `M` e sono due costanti definite precedentemente) calcolato come segue. La ricerca dell'elemento da restituire deve essere implementata come segue, a partire dalla posizione `<i,j>` della matrice `M` (dove `i` e `j` sono i parametri di ingresso della funzione). Ad ogni passo, la ricerca si sposta in una fra le posizioni adiacenti alla posizione corrente (muovendosi lungo le 8 direzioni possibili), scegliendo quella in cui `M` assume il valore minore. La ricerca non può proseguire in una posizione già visitata precedentemente o in cui il valore di `M` è maggiore di quello alla posizione corrente. Nel caso non vi sia, fra quelle adiacenti, una posizione che rispetta queste due condizioni, la ricerca termina e la funzione ritorna il valore della matrice `M` nella posizione corrente.

Note. Nel caso vi siano più posizioni adiacenti con lo stesso valore minimo di M, l'algoritmo di ricerca ne sceglie una arbitrariamente. Quando la posizione corrente si trova sul bordo della matrice, vengono analizzate solo le celle adiacenti in posizioni valide.

Esempio

Ipotizzando i=1 e j=1, e la seguente matrice M su cui effettuare la ricerca, la ricerca restituisce 0.4



Soluzione

```
float piccolo (float M[R][C], int i, int j)
{
  int V[R*C]={0},ii,jj,imin,jmin,stop=0;
  do {
    V[i*C+j] = 1;
    stop=1;
    for (ii=i-1; ii<=i+1; ii++)
      for (jj=j-1; jj<=j+1; jj++)
      {
        if (ii>=0 && ii<R && jj>=0 && jj<C && V[ii*C+jj] == 0 &&
              M[i][j]>=M[ii][jj] && (stop==1 || M[ii][jj]<M[imin][jmin]))
        {
          imin = ii;
          jmin = jj;
          stop=0;
        }
      }
    if (stop==0)
    {
      i = imin;
      j = jmin;
    }
  } while(stop==0);
  return M[i][j];
}
```

Esercizio 2 (8 punti)

Scrivere un programma C che apre un file testuale contenente una lista di codici utenti (uno per riga) e trascrive in un altro file testuale solo i codici utente che hanno un formato corretto (uno per riga). I nomi di entrambi i file devono essere acquisiti da tastiera (al più 50 caratteri). Un codice utente è valido se è una sequenza di 10 caratteri in cui i primi 6 caratteri sono cifre ed i successivi 4 caratteri sono lettere maiuscole.

Nota. Supporre che ogni riga del file contenente la lista dei codici utenti contenga al più di 12 caratteri (incluso il carattere '\n'). Supporre inoltre che le operazioni di apertura, lettura e scrittura dei file vadano a buon fine.

Soluzione

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

int valido(char str[]);
int lettera(char c);
int cifra(char c);

int main()
{
  char namein[51], nameout[51], riga[12];
  FILE *in, *out;
  scanf("%s",namein);
  scanf("%s",nameout);
  in = fopen(namein,"r");
  out = fopen(nameout,"w");
  while (feof(in)==0) {
    fgets(riga,12,in);
    if (valido(riga))
      fputs(riga,out);
  }
  fclose(in);
  fclose(out);
  return 0;
}

int valido(char str[])
{
  int i;
  if (strlen(str)<10)
    return 0;
  else {
    for (i=0; i<6; i++){
      if (cifra(str[i])==0)
        return 0;
    }
    for (; i<10; i++){
      if (lettera(str[i])==0)
        return 0;
    }
    if (str[10]!='\n') return 0;
    else return 1;
  }
}

int lettera(char c) {
  return (c>='A' && c<='Z');
}

int cifra(char c){
  return (c>='0' && c<='9');
}
```

Esercizio 3 (8 punti)

Definire un tipo di dato opportuno `clista` **per realizzare una lista dinamica di caratteri** (ogni elemento è un carattere) e serve per gestire sequenze di caratteri. Si definisce *inciso* una sequenza di caratteri compresa tra una parentesi tonda aperta ed una parentesi tonda chiusa. **Implementare una funzione che riceve in ingresso un puntatore alla testa di una lista che rappresenta una sequenza di caratteri e la modifica sostituendo tutti gli incisi con uno spazio.** Si supponga che la sequenza di caratteri nella lista in ingresso sia ben formata, cioè (i) ogni parentesi tonda aperta corrisponde ad una successiva parantesi tonda chiusa, (ii) non può esserci una parentesi tonda chiusa che non sia preceduta da una parentesi tonda aperta e (iii) non possono esserci altri incisi né parentesi all'interno di un inciso.

Esempi

La sequenza `<<Questa(ad esempio)è(una sequenza)ben formata>>` diventa `<<Questa è ben formata>>`.

Invece, `<<Questa ) ( no>>` e `<<Questa (ad (esempio)) neppure>>` non sono sequenze ben formate.

Soluzione

```
struct nodo
{
  char c;
  struct nodo *next;
};

typedef struct nodo *clista;

void pulisci(clista *l)
{
  clista cur = *l, pre = NULL, tmp = NULL;
  int inciso = 0; //0 fuori inciso, 1 dentro inciso
  while (cur!=NULL)
  {
    if (inciso == 0)
    {
      if (cur->c == '(')
      {
        inciso = 1;
        pre = cur;
        cur->c = ' ';
      }

      cur = cur->next;
    }
    else
    {
      if (cur->c == ')')
      {
        inciso = 0;
        pre->next = cur->next;
      }
      tmp = cur;
      cur = cur->next;
      free(tmp);
    }
  }
}
```

Esercizio 4 (4 punti)

Dati i due numeri A = −58 e B = +50, codificare entrambi in complemento a 2 (CPL2), utilizzando il numero minimo di bit necessari a rappresentare entrambi. Si effettuino quindi le operazioni A+B e A-B indicando esplicitamente se si verifica overflow o meno, e motivando la risposta. Mostrare i passaggi fatti.

Soluzione

Con 7 bit, in CPL2 possono essere rappresentati i numeri fra -64 e +63.

A = -58 → rappresento -58+128 = 70 = 64+4+2 → A = 1000110_{CPL2}
B = 50 = 32+16+2 = 0110010_{CPL2}

```
         11
A     1000110
+
B     0110010
----------------
      1111000
```

Non si verifica overflow, perché gli operandi hanno segno discorde.

-B → 1001101+1 → 1001110_{CPL2}

```
        111
A     1000110
-
B     1001110
----------------
  (1)0010100
```

Si verifica un overflow, perché il risultato è positivo e gli operandi negativi.

Esercizio 5 (4 punti)

Si consideri il seguente programma:

```c
#include <stdio.h>

void fun(int v[], int *p);

int main()
{
  int x[5]={1,2,3,4,5};

  printf("sizeof(int) = %d\n",(int) sizeof(int));
  printf("sizeof(int*) = %d\n",(int) sizeof(int*));
  printf("sizeof(x) = %d\n",(int) sizeof(x));
  fun(x,&x[2]); //***

  return 0;
}

void fun(int v[], int *p)
{
  printf("sizeof(v) = %d\n",(int) sizeof(v));
  printf("p-v = %d\n",(int) (p-v));
  printf("*p = %d\n",(int) *p);
}
```

A. Dire che cosa viene stampato dal precedente programma, ipotizzando che le prime due `printf` producano il seguente risultato:
```
sizeof(int) = 4
sizeof(int*) = 8
```

B. Cosa cambierebbe nel caso l'istruzione evidenziata da `//***` fosse modificata con `fun(&x[0],&x[2]);` ?

Giustificare le risposte date.

Soluzione

A.

Il programma stampa quanto segue:

```
sizeof(int) = 4
sizeof(int*) = 8
sizeof(x) = 20
sizeof(v) = 8
p-v = 2
*p = 3
```

Infatti,

- sizeof(x) restituisce la dimensione in byte dell'array x, cioè 5*4=20
- sizeof(v) restituisce 8 dal momento che v è a tutti gli effetti un puntatore int
- p-v è pari a 2, cioè la distanza fra l'indirizzo di p e v, calcolata come un multiplo della dimensione del tipo da loro puntato (int)
- *p restituisce invece il valore della variabile puntata da p, cioè il terzo elemento del vettore x, cioè 3

B.

Non cambia niente, dal momento che &x[0] è sempre l'indirizzo base dell'array x.