Politecnico di Milano - Anno Accademico 2001-2002 - Informatica B
Recupero – 25 Febbraio 2002 – Elaborato C

NOME COGNOME	
Matricola	

REC2 /Punti	I / 2 punti	II / 2 punti	III / 2 punti	IV /3 punti	V /3 punti	REC1
Tot= / 12						

REC2 I – Calcolatore – **Tradurre** un semplice programma da C in Assembler che somma 3 numeri già contenuti in una tabella NUM e scrive il risultato. (2 punti)

Assembler			*Linguaggio C*
EXIT	EQU	2000	
DIM	EQU	3	#define DIM 3
Output	EQU	2	Void main()
INDEX	DW	?	int INDEX;
NUM	DW	2	int NUM[DIM]={2,5,8};
	DW	5	
	DW	8	
RISULT		DW ?	Int RISULT ;
START	LDA	#0	{INDEX = 0 ;
	STA	INDEX	
	STA	RISULT	RISULT = 0 ;
CALCOLA	LDA	INDEX	while (INDEX != DIM)
	SUB	#DIM	{
	JZ	SCRIVI	
	LDA	RISULT	RISULT = RISULT + NUM[INDEX];
	LDI	INDEX	
	ADD	NUM(I)	
	STA	RISULT	
	LDA	INDEX	INDEX=INDEX+1;
	ADD	#1	
	STA	INDEX	
	JMP	CALCOLA	} * end while
SCRIVI	LDA	RISULT	Scrivi RISULT ;
	OUT	Output	
	JMP	EXIT	} * end main

Politecnico di Milano - Anno Accademico 2001-2002 - Informatica B Recupero – 25 Febbraio 2002 – **Elaborato C**	
NOME COGNOME	
Matricola	

REC2 - II – cicli e funzioni (2 punti)

Data la seguente definizione di struttura

```
typedef struct {int a; char b[7];}DATI;
```

la funzione che segue serve a caricare una struttura di tipo DATI:

```
DATI caricaF(int c, char *b)
{
  DATI aux;
  aux.a=c;
  strcpy(aux.b,b);
  return(aux);}
```

Si chiede di

1. trasformare la **funzione** `DATI caricaF(int c, char *b)` nella **procedura** equivalente `void carica P(int c, char *b, ..?..completare.. )` e di codificarla.

```
void caricaP(int c, char *b, DATI *el)
  { el->a=c;
    strcpy(el->b,b);
  }
```

2. scrivere il frammento di codice che, richiamando la funzione `caricaF()`, permette di caricare i dieci elementi della variabile `DATI TAB[10]` leggendo i necessari dati da tastiera

```
DATI TAB[10];
int i, a;
char s[7];
for(i=0;i<10;i++)
  {scanf("%d",&a);
   gets(s);
   TAB[I] = caricaF(a,s);}
```

REC2 - III – Files (2 punti)

Il **file di testo** `inp.txt` contiene righe di testo (in quantità non nota a priori) del seguente tipo:

ciao 46 101.25
prova 15 99.41

ciascuna di tali righe contiene tre campi: una stringa (una parola priva di spazi e lunga al massimo 8 caratteri), un numero intero e un numero con parte frazionaria. I tre campi sono separati da spazi.

Il programma da completare, sotto riportato, deve leggere da inp.txt tali informazioni e generare in output il **file binario** `exit.dat` nel quale le stesse informazioni vengono scritte sotto forma di strutture conformi alla seguente definizione:

NOME COGNOME	
Matricola	

```
typedef struct {char s[9]; int a; double b;}EL;
```

Per esempio partendo da un file inp.txt contenente le due righe di esempio sopra riportate il programma dovrebbe generare in uscita un file exit.dat contenente due strutture riempite come segue:

s	**a**	**b**
ciao	46	101.25
prova	15	99.41

Potrà essere utile l'uso della funzione `fscanf()`.

```
main()
{FILE *fin, *fout;
 EL elem;

 /* apertura files – completare! */
 fin=fopen("inp.txt","r");
 fout=fopen("exit.dat","wb");

 if((fin!=NULL) && (fout!=NULL))
   { /* ciclo lettura-scrittura fino a esaurimento del file fin - completare! */
     fscanf(fin, "%s%d%lf", elem.s, &(elem.a), &(elem.b));
      while(!feof(fin))
           { fwrite(&elem,sizeof(elem),1,fout);
             fscanf(fin, "%s%d%lf", elem.s, &(elem.a), &(elem.b));}
      fclose(fin); fclose(fout);
     }
 else
   {  /* segnalazione errore – completare! */
      printf("errore apertura file!\n");
   }
 return(0);
}
```

REC2 - IV – liste (3 punti)

Data una lista, che potete supporre contenga almeno 4 elementi, basata sulla seguente definizione:

```
struct nodo_s { int valore; struct nodo_s * next; }
struct nodo_s * head;
```

scrivere un frammento di codice che assegna 75 al valore contenuto nel terzo nodo della lista solo se il valore contenuto nel primo nodo è maggiore del valore contenuto nel secondo nodo

```
if(head->valore > head->next->valore)
     head->next->next->valore=75;
```

NOME COGNOME	
Matricola	

scrivere le istruzioni necessarie per <u>eliminare il primo nodo</u> della lista

```
struct nodo_s *tmp=head;
head=head->next;
free(tmp);
```

REC2 - V – Programma (3 punti)

Un modo per approssimare i valori della funzione logaritmo naturale è dato dalla seguente serie:

$$\ln x = 2\left[a + \frac{1}{3}a^3 + \frac{1}{5}a^5 + \frac{1}{7}a^7 + \ldots\right] \qquad \text{dove a=} \frac{x-1}{x+1}$$

il valore del logaritmo così calcolato è tanto più preciso quanti più termini della serie si sommano; ci si può quindi limitare a prendere in considerazione tanti termini quanti bastano per ottenere la precisione desiderata.
Per esempio, ci si può fermare quando si riscontra che la differenza fra il valore ottenuto sommando K termini e il valore ottenuto sommando K+1 termini non supera, <u>in valore assoluto</u>, un valore desiderato (cioè la precisione richiesta).

Scrivere il programma che, facendo riferimento alla formula sopra indicata ed alle istruzioni fornite, <u>calcola il valore del logaritmo</u> per un x <u>inserito da tastiera</u>, approssimando il risultato alla seconda cifra decimale (ovvero in modo che la differenza fra due valori consecutivi della serie troncata sia minore in valore assoluto di 0,001).

Può essere utile usare la funzione `double fabs(double val)` che dato un valore ne restituisce il valore assoluto e, ma non è indispensabile, `double pow(double base, double esponente)` che restituisce il valore base^{esponente}. Saranno opportunamente valutate le soluzioni più efficienti.

```
double logx(double x)
  { int passo=1;
    double prec=0.0, att=0.0, rapp, pot=1.0;
    rapp=(x-1)/(x+1);
    att=rapp;
    while(fabs(2*att-2*prec)>0.001)
      { prec=att;
     pot=pot*rapp*rapp;
     att = att + pot/passo;
     passo=passo+2;
      }
    return(prec);
  }
```