Corso di Laurea in Ingegneria Biomedica
Corso di Laurea in Ingegneria Elettronica
"Informatica I", a.a. 2007-08

1 Esercitazione n^o 1: Preliminari

1.1 Linguaggio 'C'

1.1.1 Richiami

Il codice deve essere scritto in modo che faciliti la riusabilità e manutenzione. *La struttura di un programma è gerarchica.*

Alcuni consigli:

Stile di programmazione Utilizzare uno stile di programmazione chiaro e semplice. Se per esempio si volessero permutare ciclicamente i valori `j=(0,1,2)` rispettivamente in `k=(1,2,0)`, si potrebbe usare un codice del tipo

```
k=j+1;
if (k==3) k=0;
```

Quindi non utilizzare nè uno stile troppo "esteso", quale l'uso di uno `switch`, nè uno stile troppo conciso, come il seguente

```
k=(2-j)*(1+3*j)/2;
```

È conveniente *indentare* il codice per facilitarne la lettura.

È buona norma commentare il codice (utile in fase di debug o a colleghi). Inoltre si deve produrre la documentazione dei propri programmi (cfr. *Java: javadoc e /** ... */*).

I concetti di modularità e incapsulamento (cfr. *Java come OOP*) sono importanti: le varie unità di codice devono interagire tra loro attraverso interfacce ben definite.

Si deve seguire uno stile di programmazione ANSI C. Per esempio le dichiarazioni di funzioni devono essere di questo tipo:

```
int funct(int i, float x, float *y, int *it)
```

Costrutti utili Utilizzare costrutti adatti al compito

- `for` per iterazioni con un predeterminato numero di passi (e.g. indicizzare vettori)
- `while` per cicli terminati in un numero di passi non conosciuto a priori. `do-while` per testare la condizione alla fine di ogni iterazione
- `break` per terminare cicli con condizioni di terminazione multiple e in diverse posizioni
- `if` per operare scelte
- Le parentesi graffe sono necessarie solo se più di una istruzione segue il costrutto. Attenzione, però, a possibili errori:

  ```
  if (b>3)
     if (a>3) b+=1;
  else b-=1;
  ```

  l'ultimo `else` è riferito al secondo `if` e non al primo, come invece potrebbe sembrare dall'indentazione

- In generale la struttura del programma dipende dal particolare algoritmo che si sta implementando. Anche a livello di struture dati: per esempio, utilizzare i vettori (anche se allocati dinamicamente) solo quando sono strettamente necessari all'algoritmo. Nell'ambito del calcolo numerico molte volte il numero di iterazioni che si devono eseguire non è noto a priori (neanche quando il programma sta girando).

Vettori e puntatori Si ricorda che un vettore dichiarato con `float b[4];` ha come riferimenti validi `b[0], b[1], b[2], b[3]`, mentre `b[4]` *non* è valido.

Esiste una corrispondenza tra vettori e puntatori: il valore referenziato da un'espressione come `a[j]` è definito essere `*((a)+(j))`.

I puntatori sono utilizzati per permettere alle funzioni di modificare variabili dell'ambiente chiamante: in tal modo da una funzione si può ritornare più valori.

```
int func(float in1, float in2, float *out1, float *out2, float *out3);
...
int flag;
float x,y,res1,res2,res3;
...
flag=func(x,y,&res1,&res2,&res3);
```

Precisione dei calcoli È importante nel calcolo numerico sapere la precisione dell'aritmetica che si usa, pertanto se non diversamente specificato si considereranno i numeri reali come `float`.

Si ricorda che i `float` hanno una accuratezza di sei cifre decimali e un campo di rappresentazione pari a $\pm\{10^{-38}, 10^{38}\}$.

Formattazione dell'output Quando si produce un'uscita è buona norma utilizzare una formattazione dei dati ordinata, chiara e approppriata.

```
float x=1.23,y=1.23e-10;
int i=3,j=200;

printf(''%f %f\n'',x,y);
printf(''%e %e\n'',x,y);
printf(''%d\n'',i);
printf(''%d\n'',j);
printf(''%3d\n'',i);
printf(''%3d\n'',j);
```

→ output →

```
1.230000 0.000000
1.230000e+00 1.230000e-10
3
200
  3
200
```

Espressioni compatte Sono molto utilizzati gli operatori di incremento e decremento `++` e `--`, sia in notazione prefissa che postfissa

```
int i=1,j=1,temp;

temp=i++;       /* temp vale 1 e i vale 2*/
temp=++j;       /* temp vale 2 e j vale 2*/
```

Inoltre si possono usare operatori di assegnamento "composti", quali `+= -= *= /= %=` per scrivere espressioni compatte

```
int i=1,j=1,temp,n=5;
float x,a=-1.2;

x=a;
for(i=2;i<=n;i++)
```

```
   x=x*a;
printf(''%f elevato a %d vale %f\n'',a,n,x);
```

è equivalente scrivere

```
for(x=a,i=2;i<=n;x*=a,i++);
printf(''%f elevato a %d vale %f\n'',a,n,x);
```

I/O dati È utile salvare i risultati del programma su file. Questo può essere fatto in due modi: redirezionare l'uscita a monitor su file (*nome_eseguibile* > *nome_file*) oppure scrivere direttamente su file.

Per utilizzare un file è necessario dichiarare un puntatore a FILE, aprire il file, scrivere sul file e infine chiudere il file. Il file può essere scritto elemento-a-elemento (formattato) o blocco-a-blocco (binario).

```
...
FILE *fp;
int i, vett[10];
...
fp=fopen(''dati.dat'',''w'');
for(i=0;i<10;i++)
   fprintf(fp,''%f\n'',vett[i]);
fclose(fp);
...
```

1.1.2 Macro con argomenti

La definizione ha la forma

```
#define nome testo_da_sostituire
```

ed è utile per descrivere le costanti di un programma. È anche possibile definire macro con argomenti, in modo che il testo da sostituire dipenda dai parametri delle diverse chiamate. Assomiglia ad una funzione ma ogni invocazione della macro viene espansa in codice in linea. Vediamo due esempi di definizione e di uso:

```
#define SQR(x) ((x)*(x))
#define MAX(A,B) ((A)>(B)?(A):(B))
...
float res, y=3, p=2, q=3;
res=SQR(y);
res=MAX(p+q,p-q);
```

Nella definizione di massimo è stata utilizzata un'*espressione condizionale*: la forma generale è `espr1 ? espr2 : espr3`. Viene dapprima valutata l'espressione `espr1` se è vera allora viene valutata `espr2` e il risultato ottenuto costituisce il valore dell'intera espressione, altrimenti si valuta `espr3`.

L'uso delle macro con argomenti può provocare errori inattesi. Per esempio: `MAX(i++,j++)` è sbagliato in quanto vi è una doppia valutazione degli argomenti; `#define SQR(x) x*x` produce un errore se invocato con espressioni come `SQR(i+1)`.

1.1.3 Puntatori a funzione

Il passaggio di funzioni ad altre funzioni viene fatta per mezzo dei puntatori a funzione. Tale tecnica è molto utilizzata nell'ambito del calcolo numerico e in generale quando si sviluppano librerie: si

implementa un certo algoritmo come funzione da fornire agli utenti, tale algoritmo utilizza valori di funzioni non note a priori, cioè fornite dall'utente (cfr. *programmazione Windows*: come si indica alla classe della finestra quale è la *funzione finestra*).

Il puntatore di una funzione è il nome stesso della funzione, la dichiarazione di un puntatore a funzione è del tipo:

```
tipo_ritornato (*nome_funzione)(argomenti_della_funzione)
```

È importante la presenza delle parentesi tonde:

```
int (*func)(float, float);
int *fun (float, float):
```

la prima indica un puntatore a funzione che ritorna un intero e ha due argomenti di tipo float, mentre la seconda indica una funzione che ritorna un puntatore a intero e ha due argomenti di tipo float.

Vediamo un esempio:

```
...
float func1(float ,int);
float func2(float (*f)(float,int),int);
...
main(){
int j=5;
float res;
...
res=func2(func1,j);
...
}
float func2(float (*f)(float,int),int a){
float temp,x=2;
...
temp=(*f)(x,a);
return temp;
}
```

1.2 MATLAB: visualizzazione dati

Se viene creato (con un qualsiasi editor o con un programma 'C') un file di testo (e.g. dati.txt) con una struttura a "colonne", per esempio:

```
1.0 2.0
1.5 3.0
2.0 5.5
```

Lo posso importare in MATLAB:

```
>> load -ascii dati.txt
>> dati
dati =
1.0000 2.0000
1.5000 3.0000
2.0000 5.5000
```

Per visualizzare i dati importati si utilizza la funzione `plot()`

```
>> plot(dati(:,1),dati(:,2))
```

1.3 Sviluppo e organizzazione di programmi

Lo sviluppo di una routine per il calcolo numerico, in generale di qualsiasi routine, può seguire tre fasi principali: (1) studiare l'algoritmo e scrivere un semplice codice che lo implementi; (2) testare il codice in un programma, cioè inserirlo in un main(); (3) riscrivere il codice come funzione di libreria. È naturale fare un debug ad ogni fase per poter proseguire nello sviluppo del programma completo.

1. In questa fase va studiato l'algoritmo da implementare e si deve prendere in considerazione la struttura generale del programma. In particolare si deve scrivere il codice base che implementa l'algoritmo e provare con semplici esempi la funzionalità delle varie parti. È buona norma commentare il codice.

2. A questo punto si deve scrivere un programma per testare il codice sviluppato. Per prima cosa si deve verificare che il programma esegua le operazioni attese: si deve fare una specie di "debug concettuale", si verifica la funzionalità dell'algoritmo con dati che forniscono uscite note. A questo livello è bene visualizzare anche i dati di operazioni intermedie per meglio capire cosa fa il programma. Poi si applica a dati di utilizzo reale (si possono ancora confrontare i risultati con quelli ottenuti con altri software, per esempio MATLAB). Infine si possono migliorare le prestazioni del programma, ottimizzandone il codice e verificandone nuovamente la piena funzionalità.

3. Quando il programma è stato completamente testato, si può sviluppare una corrispondente routine di calcolo numerico e inserirla in una libreria, che potrà essere utilizzata in altri programmi o fornita a utenti finali . A questo livello si deve produrre la documentazione delle proprie funzioni. Per creare la libreria si trasforma il programma in funzione, la si inserisce in un file sorgente con estensione .c e si crea il corrispondente file header con estensione .h.

1.4 Valutazione di funzioni

Per valutare funzioni di una variabile ($y = f(x)$) è necessario calcolare i valori della funzione in un dato intervallo della variabile indipendente. Matematicamente si considera un insieme infinito di valori compresi tra due estremi: a causa della precisione finita dei numeri in un calcolatore, si considera un insieme finito e quindi discreto di valori compresi in un intervallo, cioè tra due estremi (e.g. $x \in [-2, 1]$ con incrementi/passi di 0.1: quindi l'insieme di numeri -2.0, -1.9, -1.8 $\cdots$ 0.8, 0.9, 1.0). Quindi si considerano insiemi numerici: insiemi di numeri in cui l'informazione contenuta nel valore iniziale, nel valore finale e nel passo di "campionamento" usato (il valore della differenza tra due numeri contigui dell'insieme considerato).

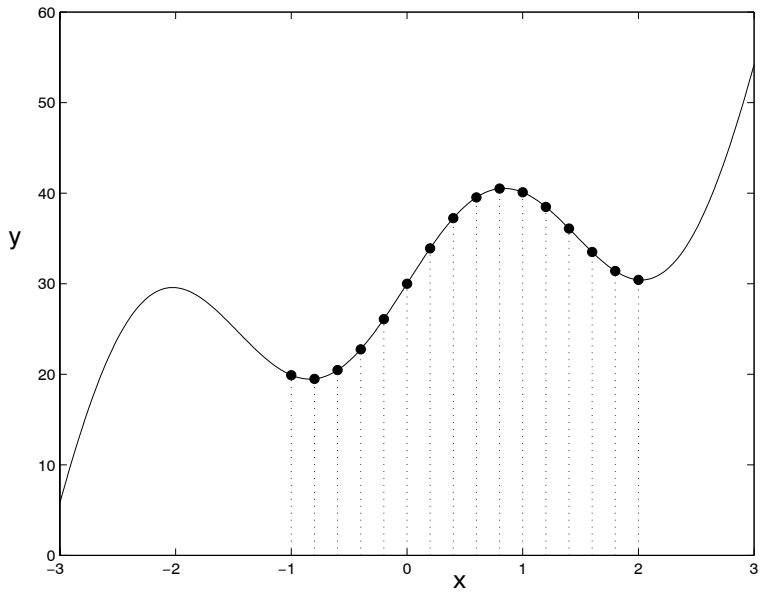
Per creare tabelle di valori di funzioni (dominio e codominio) in un dato intervallo $[a, b]$ con un numero di campioni N, si può costruire l'insieme discreto di numeri relativi alla variabile indipendente utilizzando un passo pari a

$$step = \frac{(b-a)}{(N-1)}$$

Quindi si considera l'insieme di numeri

$$x_i = a + i\ step \qquad \text{con } i = 0, 1, \cdots, N-1$$

da cui si ricava il corrispondente insieme di campioni $y_i = f(x_i)$ della funzione data.



$f(x) = x^3 + 10\sin(2x) + 30$
con $x \in [-1, 2]$ e $N = 16$

x_i	$f(x_i)$
-1.0000	19.9070
-0.8000	19.4923
-0.6000	20.4636
-0.4000	22.7624
...	
1.6000	33.5123
1.8000	31.4068
2.0000	30.4320

1.5 Esercizi proposti

Sviluppare un programma per la valutazione di una funzione in un dato intervallo. Si consideri di fornire la funzione, l'intervallo del dominio e il numero di campioni da considerare. Salvare su file i valori del dominio e codominio della funzione, formattandoli su due colonne.
N.B. Non è necessario usare vettori o allocazioni dinamiche.
Tracciare i grafici dei dati ottenuti con MATLAB.
Infine si trasformi il programma nella prima routine della libreria di calcolo numerico.

Valutare le seguenti funzioni:

$$
\begin{aligned}
f(x) &= x^2 + 0.1x - 1 && \text{con } x \in [-3,3] \text{ e } N = 61 \\
f(x) &= \frac{1}{(x-0.3)^2 + 0.01} + \frac{1}{(x-0.9)^2 + 0.04} - 6 && \text{con } x \in [-1,2] \text{ e } N = 41 \\
f(x) &= \sin(2\pi f x) && \text{con } x \in [-2,2],\ N = 41,\ f = 0.5 \text{ e } f = 5 \\
f(x) &= \frac{x - \sin(x)}{\tan(x)} && \text{con } x \in [-2,2],\ N = 17 \text{ e } N = 19 \\
f(x) &= x - \sqrt{x-1} && \text{con } x \in [1,5] \text{ e } N = 81
\end{aligned}
$$

Alcuni output di esempio:

$f(x) = x^2 + 0.1x - 1$
con $x \in [-2,2]$ e $N = 15$

x_i	$f(x_i)$
-2.0000	2.8000
-1.7143	1.7673
-1.4286	0.8980
-1.1429	0.1918
-0.8571	-0.3510
-0.5714	-0.7306
-0.2857	-0.9469
-0.0000	-1.0000
0.2857	-0.8898
0.5714	-0.6163
0.8571	-0.1796
1.1429	0.4204
1.4286	1.1837
1.7143	2.1102
2.0000	3.2000

$f(x) = (x - \sin(x))/\tan(x)$
con $x \in [-1,2]$ e $N = 16$

x_i	$f(x_i)$
-1.0000	0.1018
-0.8000	0.0803
-0.6000	0.0517
-0.4000	0.0250
-0.2000	0.0066
-0.0000	0.0000
0.2000	0.0066
0.4000	0.0250
0.6000	0.0517
0.8000	0.0803
1.0000	0.1018
1.2000	0.1042
1.4000	0.0715
1.6000	-0.0175
1.8000	-0.1927
2.0000	-0.4992