Sommario

- I/O FILE: lettura e scrittura su file di testo
- Funzioni:
  - Overloading
  - Argomenti di default
- Puntatori a funzione
- Grafici in MATLAB

I/O FILE

- Per eseguire operazioni di I/O su file, si deve includere nel programma l'header `<fstream>`
- Tale header definisce numerosi classi, fra le quali:
  - `ifstream` per l'input
  - `ofstream` per l'output
  - `fstream` per input e output

I/O FILE

- E' possibile creare un oggetto ifstream (o ofstream) e poi invocare su di esso il metodo open()

```
ifstream input;
input.open("prova.txt")
```

- Oppure si puo' utilizzare un costruttore che apre automaticamente il file:

```
ifstream input("prova.txt");
```

- Per effettuare le operazioni di lettura o scrittura su *file di testo* si utilizzano gli operatori << e >> nello stesso modo visto per le operazioni di I/O su console.
- Invece che `cin` e `cout` si utilizza uno stream precedentemente collegato ad un file.

I/O FILE – LETTURA DA FILE DI TESTO

oggetto di tipo ifstream

lettura di un intero

```
int TestLettura(){
        ifstream infile("lettura.txt");
        if (!infile){
                cout<<"errore apertura file di input"<<endl;
                return -1;
        }
        int num;
        infile>>num;
        int* dati = new int[num];

        int i=0;
        while(infile>>dati[i] && i<num)
        {
                cout<<dati[i]<<endl;
                i++;
        }
        delete [] dati;
        infile.close();
        return 0;
}
```

I/O FILE – LETTURA DA FILE DI TESTO

lettura.txt

```
9
11
22
33
44
55
66
77
88
99
```

Una possibile uscita:

```
11
22
33
44
55
66
77
88
99
```


I/O FILE – SCRITTURA SU FILE DI TESTO

```
int TestScrittura()
{
	const int num=10;
	int seed=3;
	srand(seed);
	int dati[num];
	for (int i=0; i<num; i++)
		dati[i]=rand();
	ofstream outfile("scrittura.txt");
	if (!outfile){
		cout<<"errore apertura file di output"<<endl;
		return -1;
	}
	outfile<<"Numero di valori: "<<num<<endl;
	for (int i=0; i<num; i++)
		outfile<<dati[i]<<endl;
	outfile.close();
	return 0;
}
```

oggetto di tipo ofstream

scrittura

I/O FILE – SCRITTURA SU FILE DI TESTO

scrittura.txt

```
Numero di dati: 10
48
7196
9294
9091
7031
23577
17702
23503
27217
12168
```


Overloading

- L'*overloading* (sovraccaricamento) delle funzioni consente a più funzioni, che offrono un'operazione comune su tipi di parametri diversi, di condividere un nome comune.

- Se non esistesse l'overloading, ad esempio, si dovrebbe definire un insieme di funzioni `max` con nomi diversi in relazione al tipo dei parametri:

```
int i_max(int, int);
int v_max(int *);
int mat_max(Matrix *);
```

- Ma tali funzioni eseguono le *stessa azione generale*: restituire il valore massimo nell'insieme dei valori dei loro parametri.

Overloading

- Dal punto di vista dell'*utente*, esiste una sola operazione, quella di determinare il valore massimo, mentre i dettagli di implementazione sono di scarso interesse.
- Si può dare lo stesso nome a due o più funzioni se le loro *liste di parametri* sono *distinte* per numero o tipo di parametri.

```
void print(string s);
void print(int a);
void print(int a, int b);
```


Overloading

- Il solo tipo di ritorno non basta per distinguere due funzioni.

```
int   val(float x, float y);
float val(float x, float y);
```

Errore compile-time

- È necessario fornire una dichiarazione (e definizione) distinta per ogni funzione overloaded.

Overloading: esempio

```
#include <iostream>
using namespace std;
int max(int a, int b);
int max(int *v, int size);

int main(){
   int ia[]={-1,3,15,-20,1};
   int a1=3,a2=4,dim=5;

   cout<<max(a1,a2)<<endl;
   cout<<max(ia,dim)<<endl;
   return 0;
}
int max(int a, int b){
   if (a>b)  return a;
   else  return b;
}
int max(int *v, int size){
   int m = v[0];
   for(int i=1;i<size;i++)
        if (v[i]>m) m=v[i];
   return m;}
```

Scrivere una funzione che calcola il massimo tra due numeri e una il massimo valore contenuto in un vettore .

Inizializzazione array

Una possibile uscita

```
4
15
```


Argomenti di default

- Un argomento di default è un *valore* che, sebbene non applicabile in tutte le circostanze, viene giudicato valore appropriato di un parametro nella maggior parte dei casi.

- Una funzione può specificare un argomento di default per uno o più parametri usando la *sintassi di inizializzazione* nella lista dei parametri:

```
int val(int v, int cf=0);
```

Argomenti di default

- Tale funzione può essere chiamata con o senza argomento per tale parametro:

```
val(3);
val(3,3);
```

- I parametri *non inizializzati* devono essere quelli più a *sinistra*. Non si può mescolare parametri inizializzati e parametri non inizializzati:

```
int val2(int v=33, int cf);
```

Errore compile-time

Argomenti di default

- Gli argomenti di default sono usati soltanto per sostituire gli *ultimi argomenti* mancanti di una chiamata.

- L'argomento di default è specificato nella dichiarazione contenuta in un file header pubblico (che descrive l'interfaccia) e non nella definizione della funzione.

Argomenti di default

```
#include <iostream>
using namespace std;
void print(int=1, char='a' );
int main(){
  print();
  print(3);
  print(3,'b');
  //print(,'c');
  return 0;
}
void print(int n, char ch){
  for(int i=0; i<n;i++)
      cout<<ch;
  cout<<endl;
}
```

Scrivere una funzione che: chiamata senza argomenti stampa 'a', con un argomento intero n *stampa* n *volte 'a' e con un secondo argomento char* ch *stampa* n *volte* ch.

Notare la dichiarazione

Errore compile-time

Una possibile uscita

```
a
aaa
bbb
```


Argomenti di default e overloading

- In generale quale scelta operare dipende da quale implementazione risulta più chiara:
  - Se il corpo della funzione comprende diverse parti alternative eseguite in relazione al valore dell'argomento di default, allora è più opportuno fare un overloading e lasciare la scelta di quale codice eseguire al compilatore.
  - Se si devono eseguire solo delle inizializzazioni diverse, ma il codice è unico, allora è meglio usare gli argomenti di default.

Argomenti di default e overloading

- Si deve prestare attenzione a non creare situazioni *ambigue*: un overloading che coincide con una chiamata con argomento di default, in tal caso il compilatore genera un errore.

```
void print(int=1, char='a' );
void print(int );

int main(){
  print(3);
}
```

Errore compile-time

Puntatori a funzione

- Il passaggio di funzioni ad altre funzioni viene fatta per mezzo dei puntatori a funzione.

- Tale tecnica è molto utilizzata 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 (e.g. calcolare il valore dell'integrale di una funzione fornita dall'utente).

Puntatori a funzione

- Il puntatore di una funzione è il nome stesso della funzione. Dichiarazione di un puntatore a funzione:

```
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.

Puntatori a funzione

fp.h

```
#ifndef    FP_H
#define    FP_H

void Stampa(float (*f)(float), float x);

#endif
```

fp.cpp

```
#include <iostream>
#include "fp.h"

using namespace std;

void Stampa(float (*f)(float), float x){
       float y;
       y=(*f)(x);
       cout<<"x="<<x<<" y="<<y<<endl;
}
```

Puntatori a funzione

Test_fp.cpp

```
#include <iostream>
#include <cmath>
#include "fp.h"

using namespace std;

float f1(float);
float f2(float);

int main(){

        Stampa(f1,3);
        Stampa(f2,-2);

        return 0;
}
```

```
float f1(float x){
        return x*x;
}

float f2(float x){
        return exp(x);
}
```

Una possibile uscita

```
x=3  y=9
x=-2 y=0.135335
```


Grafici in MATLAB

- MATLAB (*MATrix LABoratory*) è un ambiente di sviluppo interattivo per il calcolo scientifico
- Originariamente sviluppato come interfaccia per il software matriciale LINPACK e EISPACK

→ L'elemento base è la **matrice**, che non richiede dimensionamento

→ Ideale per risolvere problemi con formulazione matriciale o vettoriale

Grafici in MATLAB

- Un file di testo (e.g. `dati.txt`) con una struttura a "colonne", per esempio ascisse e ordinate.
- Può essere importato in MATLAB con le istruzioni
  `load -ascii dati.txt`
- In tal modo viene creata una matrice con lo stesso nome e dimensione del file .
- Per visualizzare i dati importati si utilizza la funzione *plot(ascisse,ordinate)*:
  `plot(dati(:,1),dati(:,2))`

Dati.txt

1.0	2.0
2.0	4.0
3.0	9.0