LABORATORIO DI RICERCA OPERATIVA 2020 -2021

Laboratorio OPL - LEZIONE 1

Sommario

1. Il Software CPLEX OPTIMIZATION STUDIO
2. Il Linguaggio di Modellizzazione OPL
3. Esempi

Milestones

- 1826/1827 Fourier:rudimentory form of the simplex method in 3 dimensions.
- 1939 L. V. Kantorovitch:Foundations of linear programming (Nobel Prize 1975)
- 1930-1940:
  - LP models and solution approaches developed independently in the Soviet Union and the West for a variety of optimal resource allocation and planning applications.
  - The diet problem for US army's soldiers. 77 unknowns and 9 constraints. Stigler used a heuristic: $39.93/year
- 1947
  - G. B. Dantzig (1914-2005): simplex algorithm
  - Laderman used simplex for solving the Army's Diet Problem: $39.69/year (1939 prices). first "large-scale computation" took 120 man days on hand operated desk calculators (10 human "computers")

- 1953 First computer code by W. Orchad-Hyes (71 variables, 26 constraints, 8h running time, «a good portion of that time being spent manually feeding cards into the CPC, an ancient conglomeration of tabulating equipment, electro-mechanical stor-age devices, and an electronic calculator with tubes and relays»)

- 1960 LP solvers became commercially available
- ……
- 1988 First release of CPLEX (R.E. Bixby)

The key weakness in early optimization systems was not in their algorithms, however, but in their interaction with modelers

- ….. Gurobi, Mosek, FICO Xpress, Symphony….

Linguaggi di Modellizzazione per l'ottimizzazione

1990 R Fourer, *AMPL: A modeling language for mathematical programming*

Linguaggi general-purpose: supportano più software per l'ottimizzazione

- Paragon Decision Technology: AIMMS modeling language *An Optimisation Programming Language*
- GAMS Developement Corporation: GAMS modeling language
- Maximal Software Inc: MPL modeling language
- Free Software Foundation: GNU MathProg modelling language

Linguaggi special-purpose: supportano solver specifici

- LINDO SYSTEMS: LINGO
- IBM: OPL (P. Van Hentenryck OPL 1999: *An Optimisation Programming Language)*

Software per l'ottimizzazione

CPLEX Optimization Inc 1988: Cplex (Simplex in C)

- 1991 :solutore per problemi interi
- 1997: ILOG rileva Cplex;
- 2000-2001: ILOG Concert Technology (interfacce C++, Java, interfacce per la modellizzazione)
- 2009: IBM ILOG Cplex Optimization Studio

IBM CPLEX OPTIMIZATION STUDIO

Ambiente di Sviluppo Integrato (IDE) basato su ECLIPSE.
Esso comprende

- Il Linguaggio di Modellizzazione OPL (Optimization Programming Language ideato da Pascal Van Hentenryck, 1999 Massachusetts Institute of Technology)

- Due "motori" per l'ottimizzazione

  ① IBM CPLEX (Default)

  ② CONSTRAINT PROGRAMMING

CPLEX

E' software specifico per l'ottimizzazione

Componenti:

1. Ilog Cplex Interactive Optimizer: eseguibile che "legge" un problema da uno stream di input (terminale, file) e fornisce la soluzione su uno stream di output (terminale, file).

2. Ilog Cplex Callable Library: libreria C che consente di integrare CPLEX con tutti linguaggi di programmazione ad alto livello capaci di invocare una funzione C (C, VisualBasic, Fortran)

3. Ilog Concert Technology: insieme di librerie e classi (API) che consentono di integrare CPLEX all'interno di applicazioni C++, Java, .NET.

4. Python Api, Matlab Interface, Excel Solver

Ilog Cplex Interactive Optimizer

Aprire il prompt dei comandi (Windows) o il terminale (Unix like OS), spostarsi nella directory ……./IBM/ILOG/CPLEX_StudioXXX/cplex/bin/*piattaforma/*

Eseguire il programma cplex

```
x86-64_darwin9_gcc4.0 — cplex — 59×19
CPLEX> enter
Enter name for problem: PrimoEsempio
Enter new problem ['end' on a separate line terminates]:

maximize   3X1  +  2X2

subject to

C1:         X1  +   X2  <=  11

C2:        6X1  +  5X2  <=  60

C3:        8X1  +  3X2  <=  48

bounds
           X1 >=0
           X2 >=0
end
CPLEX>
```

comando

opzionale

keyword

Sintassi standard di un modello cplex:

1. Non è necessario dichiarare né il numero, né il nome delle variabili;
2. Per default, se non diversamente dichiarate, tutte le variabili di un modello sono continue e non negative.

1. Optimize: per risolvere il problema
2. Display: per visualizzare la soluzione

Basic OPL Model

```
/*********************************************
 * OPL 12.4 Model
 * Author: Marcello
 * Creation Date: 08/nov/2017 at 9.15.00
 *********************************************/
 dvar float+ Pasta;
 dvar float+ Latte;
 dvar float+ Formaggio;
 dvar float+ Pesce;
 dvar float+ Verdura;
 dvar float+ Pane;
 dvar float+ Carne;
```

Prima differenza con cplex:
Le variabili devono essere dichiarate prima di essere usate

Seconda differenza con cplex:
I vincoli sono separati da ;

```
minimize  4*Pasta+ 4*Latte+ 15*Formaggio + 22.5*Pesce + 3*Verdura + Pane + 15*Carne;
subject to{
Contenuto_protei_LI:
11.5*Past+3.15*Latte+8*Formaggio+18.5*Pesce+2.1*Verdura+12*Pane+ 74*Carne >=25;
Contenuto_protei_LS:
11.5*Pasta + 3.15*Latte + 8*Formaggio + 18.5*Pesce + 2.1*Verdura + 12*Pane + 74*Carne <=35;

Contenuto_carboi_LI:
72.7*Pasta + 4.85*Latte + 3.8*Formaggio + 0.5*Pesce + 0*Verdura + 68*Pane + 1*Carne >=15;
Contenuto_carboi_LS:
72.7*Pasta + 4.85*Latte + 3.8*Formaggio + 0.5*Pesce + 0*Verdura + 68*Pane + 1*Carne <=25;

Contenuto_grassi_LI:
1.5*Pasta + 1.55*Latte + 11*Formaggio + 19*Pesce + 0.1*Verdura + 6*Pane + 9*Carne >=10;
Contenuto_grassi_LS:
1.5*Pasta + 1.55*Latte + 11*Formaggio + 19*Pesce + 0.1*Verdura + 6*Pane + 9*Carne <=20;
}
```

Terza differenza con cplex:
La presenza esplicita del simbolo * tra costante e variabile

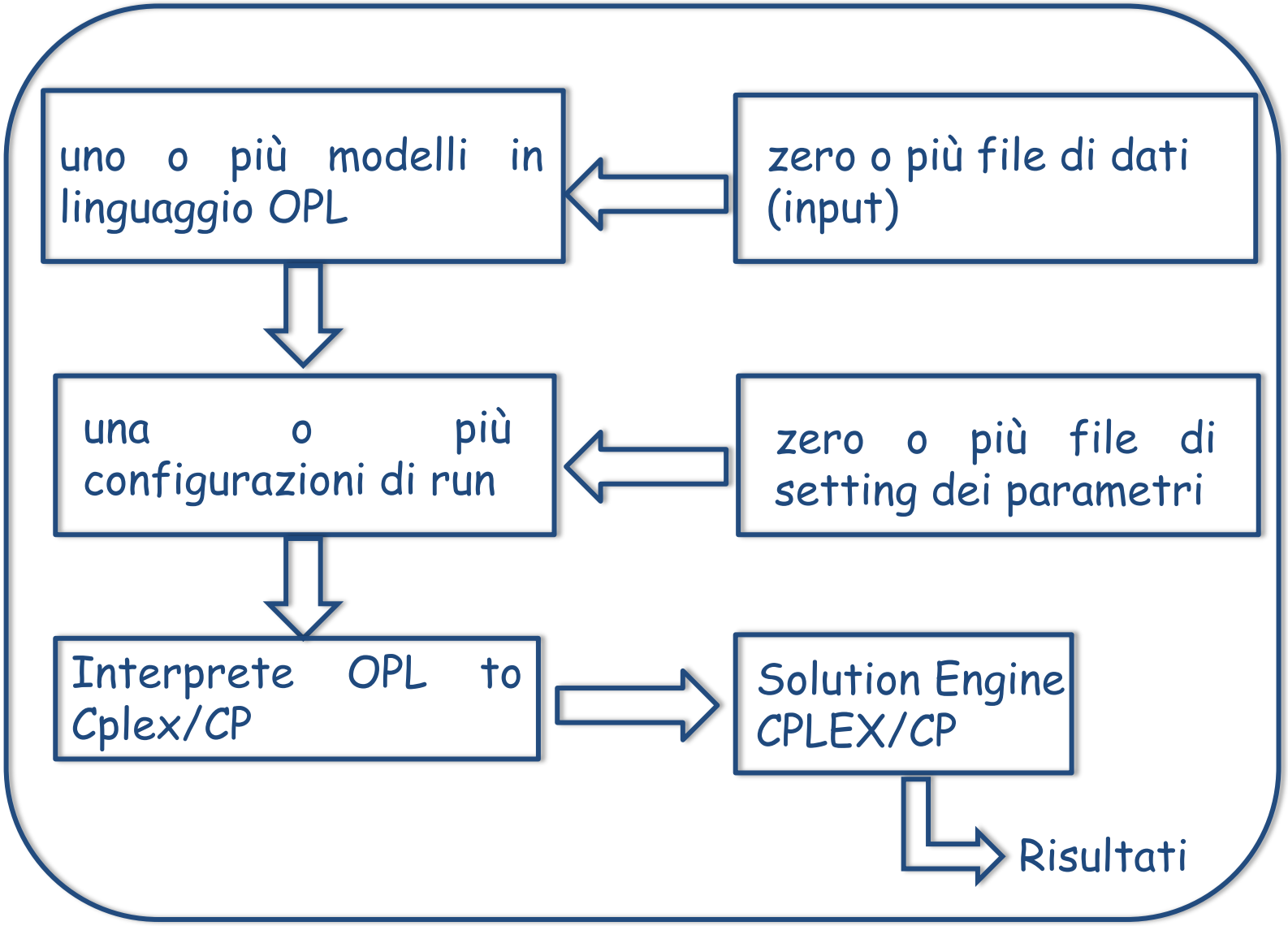
Advanced OPL Model

```
/*********************************************
 * OPL 12.6 Model
 * Author: Marcello
 * Creation Date: 8/nov/2017 at 9.30.16
 *********************************************/
int numAlimenti = ...;
int numNutrienti = ...;

range Vincoli = 1..numNutrienti;
range Variabili = 1..numAlimenti;
range LimInf = 1..numNutrienti;
range LimSup = 1..numNutrienti;


float C[Variabili]=...;
float A[Vincoli][Variabili] = ...;
float UB[LimSup] = ...;
float LB[LimInf]= ...;

dvar float+ X[Variabili];

minimize sum(j in Variabili) C[j]*X[j];
subject to{
  forall(i in Vincoli)
  vincoliSup: sum(j in Variabili) A[i][j]*X[j] <= UB[i];

  forall(i in Vincoli)
  vincoliInf: sum(j in Variabili) A[i][j]*X[j] >= LB[i];
  };
```

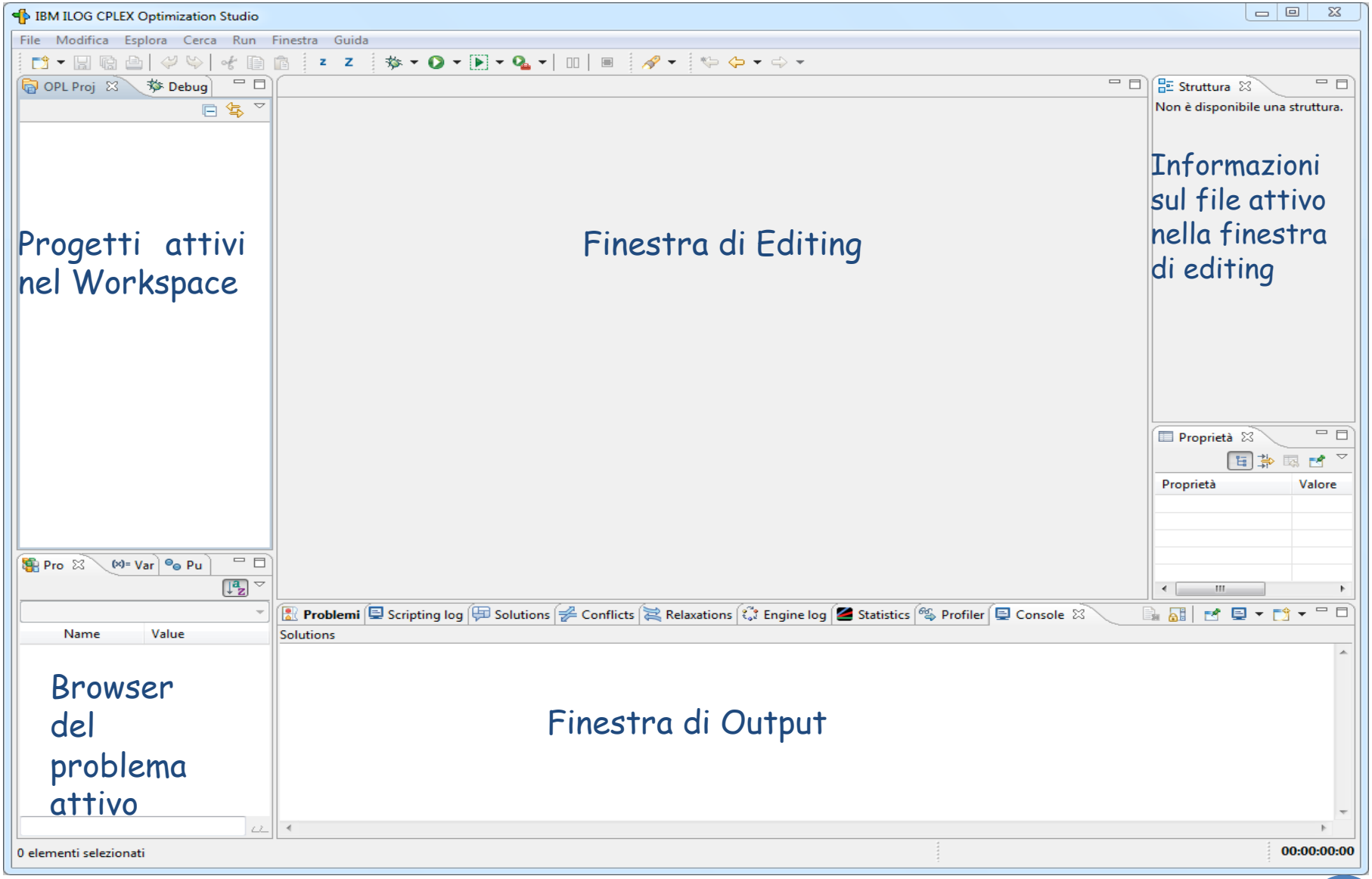
```
/*********************************************
 * OPL 12.6 Data
 * Author: Marcello
 * Creation Date: 8/nov/2017 at 9.35.19
 *********************************************/
numAlimenti = 7;
numNutrienti = 3;
C = [4 4 15 22.5 3 1 15];
UB = [35 25 20];
LB = [25 15 10];
A = [[11.5 3.15  8    18.5 2.1 12  74]
     [72.7 4.85  3.8   0.5 0   68   1]
     [ 1.5 1.55 11    19   0.1  6   9]];
```

Quarta differenza con cplex:
Possibilità di tenere separati i dati dal modello

I PROGETTI DI IBM CPLEX OPTIMIZATION STUDIO



IL WORKSPACE DI IBM CPLEX OPTIMIZATION STUDIO



Il Linguaggio OPL -1

$$
\begin{array}{lll}
maximize(minimize) & c_1x_1 + \cdots + c_nx_n & \\
subject\ \ to & & \\
[con\_1] & a_{11}x_1 + \cdots + a_{1n}x_n \ <= & b_1 \\
\cdots \quad \cdots & \cdots \quad \cdots \quad \cdots & \cdots \\
[con\_i] & a_{i1}x_1 + \cdots + a_{in}x_n \ <= & b_i \\
\cdots \quad \cdots & \cdots \quad \cdots \quad \cdots & \cdots \\
[con\_m] & a_{m1}x_1 + \cdots + a_{mn}x_n \ <= & b_m \\
 & x_1 \geq 0 & \\
 & x_2 \geq 0 & \\
 & x_n \geq 0 &
\end{array}
$$

CPLEX (Interactive Optimizer)
Low-Level Modelling Language

$$
\begin{aligned}
\max\ (\min)\quad & \sum_{j=1}^{n} c_j x_j & \\
& \sum_{j=1}^{n} a_{ij} x_j \leq b_i & \forall i = 1, \ldots, m \\
& x_j \geq 0 & \forall j = 1, \ldots, n
\end{aligned}
$$

OPL
Low-Level Modelling Language
High-Level Modelling Language

Perché OPL è un Linguaggio di Modellizzazione ad Alto Livello?

Un Modello OPL consiste di

1. Un insieme di dichiarazioni (variabili decisionali, costanti usate dal modello)
2. Un insieme (opzionale) di istruzioni di preprocessing
3. La definizione del modello matematico (funzione obiettivo e vincoli)
4. Un insieme (opzionale) di istruzioni di postprocessing
5. Un insieme (opzionale) di istruzioni di controllo del flusso di esecuzione (main Block), utile per progettare algoritmi di ottimizzazione "special purpose".

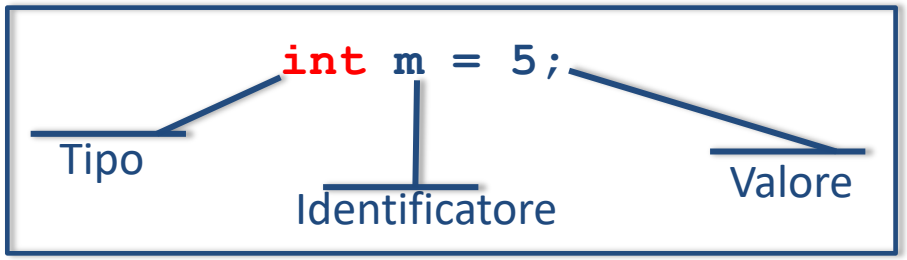
SINTASSI: Nomi validi in OPL

1) Sono nomi validi di variabili e costanti tutte le stringhe alfanumeriche di al più 255 caratteri (a-z, A-Z, 0-9, ! " # $ % & ( ) , . ; ? @ _ { } ~).
   a. Non è consentito iniziare il nome di una variabile o costante con un numero o con .
   b. Non è consentito l'uso del carattere e (o E) seguito da una cifra o da un altro carattere e (sequenza riservata alla notazione scientifica dei numeri decimali).
   c. Non è consentito, ovviamente, l'utilizzo delle keywords proprie del linguaggio

Tipi di dati fondamentali in OPL: `int, float, string`

Dichiarazione di costanti:

```
tipo nomecostante [= valore];
```



dichiara una costante di tipo intero inizializzata al valore 5

```
float n = 2.5;
```

dichiara una costante reale inizializzata

```
float square_n = n*n;
```

dichiara una costante reale come risultato di un'espressione

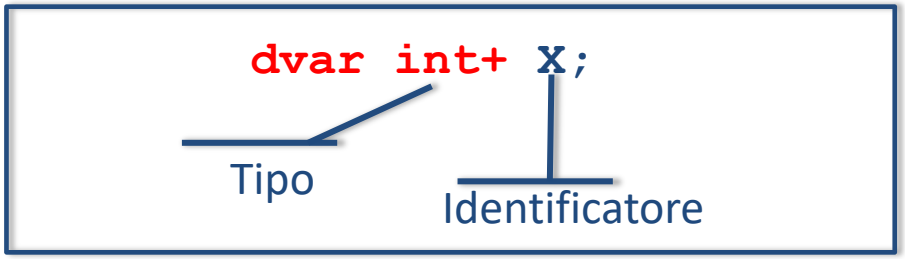
```
string s = "CPLEX";
int laInizializzoAltrove;
```

Sono Tipi di Dati Semplici

Tipi di dati fondamentali in OPL: `int, float, string`

Dichiarazione di variabili decisionali:

```
dvar tipo[+] nomevariabile;
```



dichiara una **variabile decisionale** di tipo intero non negativa

```
dvar float  f;
```
variabile continua

```
dvar float+ t;
```
variabile continua non negativa

Attenzione

Non confondere il concetto di **variabile decisionale** con quello tipico dei linguaggi di programmazione (locazione di memoria). Le variabili decisionali sono il **risultato** di un calcolo e non vanno mai inizializzate o manipolate dall'utente!

Primo Esempio (Esercizio 2 dell'esercitazione di ieri)

1. Eseguire OPL
2. Menu File -> Nuovo -> OPL Project (Specificare il nome del progetto e la cartella di lavoro)
3. Attivare i flags: **Add a default Run Configuration** e **Create Model**
5. Nella finestra Model digitare il codice seguente

```
dvar float+ x1;
dvar float+ x2;
dvar float+ x3;
dvar float+ x4;
dvar float+ x5;
dvar float+ x6;

minimize  x1+2*x2+x3+x4+x5+x6;
      subject to{
                    vincolo1: x1+2*x2+3*x3+x4 ==3;
                    vincolo2: 2*x1-x2-*x3+x5 == 2;
                    vincolo3: x1+2*x2-*x3+x6 == 2;
                };
```

6. Click con il tasto destro su Nome progetto -> Run -> Default Configuration

Prototipazione di un modello

Cosa succede se dobbiamo risolvere un modello simile al precedente ma con 100 variabili e 50 vincoli?

1. Dichiarare (pedantemente!) tutte le variabili e scrivere tutti i vincoli (alto rischio di commettere errori).

2. Scrivere un modello parametrico nelle dimensioni e nei dati per pervenire al Prototipo di un Modello.

$$\begin{aligned} \min \quad & c^T x \\ & Ax \geq b \\ & x \geq 0 \end{aligned}$$

Tipi di Dati Strutturati 1: Range

Un Range è un intervallo di valori, specificato dai suoi estremi e viene utilizzato per definire array, oppure come intervallo in cui fare variare gli indici di sommatorie (∑) e quantificatori (∀), o come dominio per le variabili.

Sintassi: `range nomeRange = EstremoInf .. EstremoSup;`

```
range Range1 = 1..5;
```
dichiara un intervallo discreto di 5 valori da 1 a 5

```
int k=10;
range Range2 = k+1..2*k*k;
```
dichiara un intervallo di valori da 11 a 200

```
range DomI = 0..10;
dvar int+  X in DomI;
```
dichiara una variabile intera a valori tra 0 e 10

```
range DomC = 0.0..1.0;
dvar float+  Y in DomC;
```
dichiara un intervallo continuo di valori tra 0 e 1

dichiara una variabile continua a valori tra 0 e 1

Attenzione: I range continui servono solo a definire il dominio delle variabili!

Tipi di Dati Strutturati 2: Array

Array: insieme di oggetti omogenei ciascuno dei quali è accessibile direttamente tramite indice

Sintassi: `tipo nomeArray[Range] = [listaValori];`

```
int d[1..5] = [10,4,5,6,8];
```
dichiara un vettore di 5 interi

```
range R = 1..3;
float c[R] = [1.5, 4.0, -2.9];

string days[1..4]=["Lun", "Mart", "Mer", "Giov"];
dvar float+ D[R];
```
dichiara un vettore di 3 variabili decisionali continue >=0

Array multidimensionali

Sintassi:

```
tipo nomeArray[Range1][Range2]= [Riga1, Riga2, …, Rigam];
```

ogni riga è un Array con `Range2` elementi

```
range righe = 1..3
range colonne = 1..4
int mat[righe][colonne]=[[1,5,6,2],[6,2,3,4],[1,3,4,6]];
```

Tipi di Dati: Inizializzazione Interna (nel file .mod)

```
int d[1..5] = [10,4,5,6,8];
```
elencando i valori

```
range R=1..4;

int a[R];
execute{
    for (var i in R){
       a[i]=i+1;
    }
}
```

chiamata a routine di sistema nella fase di pre-processing

```
range righe = 1..3
range colonne = 1..4
int mat[righe][colonne]=[[1,5,6,2],[6,2,3,4],[1,3,4,6]];

int mat1[righe][righe];
execute{
    for (var i in righe)
       for(var j in righe{
          mat1[i][j]=i+j;
    }
}
```

Tipi di Dati: Inizializzazione Esterna (nel file .dat)

OPL consente di tenere "separati" i dati dal modello, cioè consente dichiarazioni (costanti, array) non istanziate.

```
int d = 10;
```
costante intera che vale 5 ogni volta che il modello che la utilizza viene risolto.

```
float z = ...;
```
costante float, dichiarata ma non inizializzata: il valore che essa assumerà potrà essere diverso in risoluzioni diverse del modello

I punti di sospensione `...` avvisano l'interprete che i "dati" sono elencati "altrove" (nel file dat) e non dove la costante è dichiarata.

```
z =2.5;
```
inizializzazione nel file dat

Con l'inizializzazione esterna gli array si inizializzano sempre elencandone i valori,

```
int h[1..3] = ...;
```
dichiarazione nel file mod

```
h = [10, 15, -7];
```
inizializzazione nel file dat

oppure elencando le coppie <indice,valore>

```
h = #[3:-7 , 1:10 , 2:15];#
```
inizializzazione nel file dat

Inizializzazione Esterna: il file .dat

E' un file testuale in cui sono elencati i valori che le costanti non inizializzate del modello devono assumere.

ERRORE COMUNE: Le variabili decisionali non vanno inizializzate nel file .dat (nè si potrebbe fare): esse sono l'output del processo di ottimizzazione.

```
int cento = ...;
```
(nel file mod cento è una costante intera non inizializzata)

```
cento = 25;
```
(nel file dat cento è inizializzata al valore 25)

```
int h[1..3] = ...;
```
(un vettore di 3 interi non inizializzato)

```
int k[1..2][1..3] = ...;
```
(matrice di 6 interi non inizializzato)

I vettori multidimensionali vengono inizializzati come vettori di vettori

```
k = [[ 4, 5, 8],
     [-3, 2, 7]];
```

```
k = #[2:[-3, 2, 7],
      1:[ 4, 5, 8]] #;
```

Primi passi verso la Prototipizzazione

Ogni modello di programmazione lineare è caratterizzato dalla sua "dimensione": numero di variabili e numero di vincoli

```
int numVincoli = ...;
int numVariabili = ...;

range Vincoli   = 1..numVincoli;
range Variabili = 1..numVariabili;

float C[Variabili]=...;
float A[Vincoli][Variabili] = ...;
float B[Vincoli] = ...;

dvar float+ X[Variabili];
```

$$\min \quad c^T x$$

$$Ax \geq b$$

$$x \geq 0$$

L'insieme delle precedenti dichiarazioni sono generalmente sempre presenti in ogni modello OPL

Funzione Obiettivo e Vincoli: gli operatori `sum` e `forall`

Sintassi:

```
sum(Indice in Range) exprAlg;

forall(Indice in Range) exprAlg opRel exprAlg;

opRel: == | <= | >=
```

```
minimize sum(j in Variabili)C[j]*X[j];

subject to{
  forall(i in Vincoli)
    sum(j in Variabili) A[i][j]*X[j] >= (==, <=) B[i];
  }
```

Il file mod

```
int numVincoli = ...;
int numVariabili = ...;

range Vincoli   = 1..numVincoli;
range Variabili = 1..numVariabili;

float C[Variabili]=...;
float A[Vincoli][Variabili] = ...;
float B[Vincoli] = ...;

dvar float+ X[Variabili];

minimize sum(j in Variabili)C[j]*X[j];

subject to{
  forall(i in Vincoli)
   vincoli: sum(j in Variabili) A[i][j]*X[j] >= ( ==, <= ) B[i];
  }
```

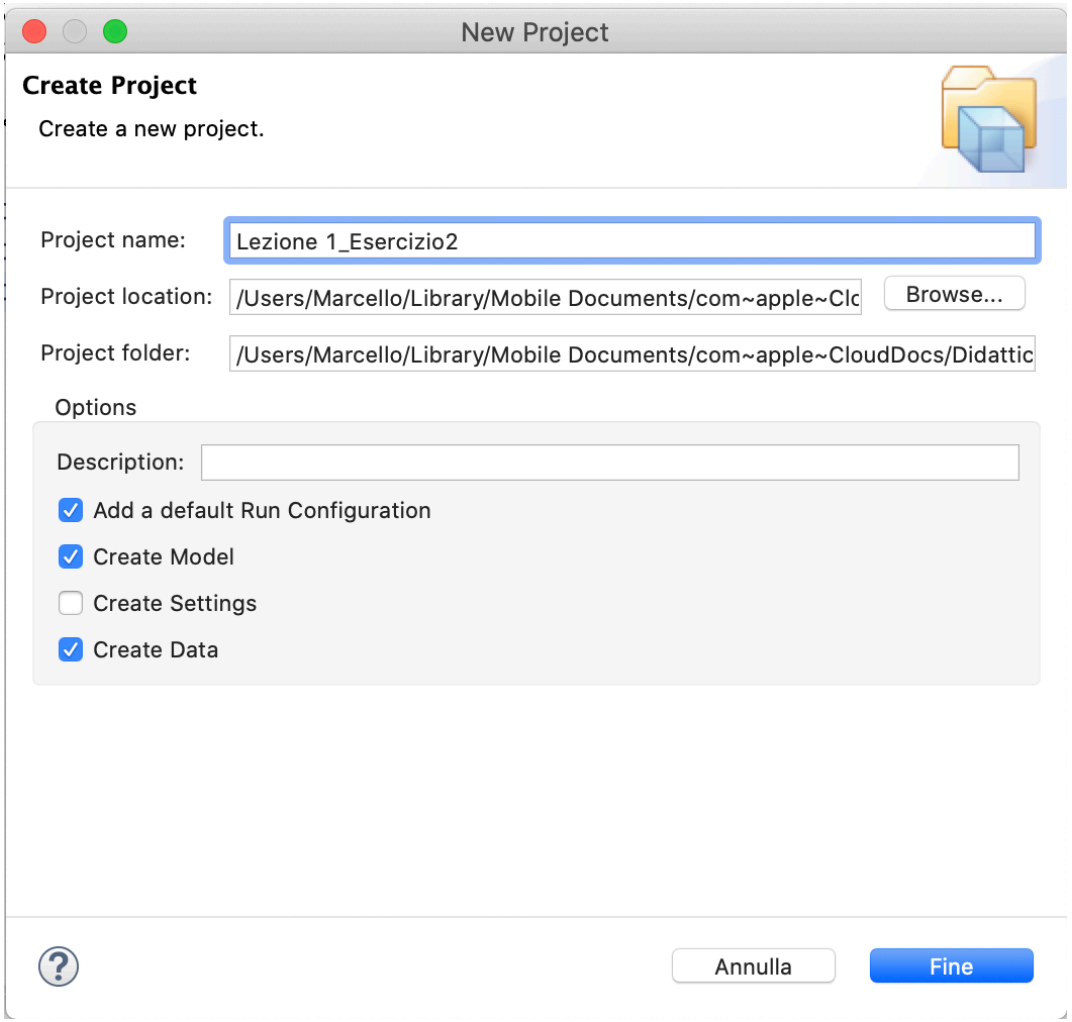
Primo Esempio – Esercizio 2 di ieri

```
numVariabili = 6;
numVincoli = 3;
C=[1, 2, 1, 1, 1, 1];
b=[3, 2, 1];


A=[
   [1 , 2,  3, 1, 0, 0],
   [2, -1, -5, 0, 1, 0],
   [1,  2, -1, 0, 0, 1]
   ];
```

Primo Esempio – Il problema della dieta

File -> Nuovo -> OPL Project

Flag su questi campi



Secondo Esempio - Esercizio 2 - Il File Mod

```
/*********************************************
 * OPL 12.6 Model
 * Author: Marcello
 * Creation Date: 8/nov/2017 at 12.45.21
 *********************************************/
int numVariabili = ...;
int numVincoli = ...;

range Vincoli = 1..numVincoli;
range Variabili = 1..numVariabili;


float C[Variabili]=...;
float A[Vincoli][Variabili] = ...;
float b[Vincoli] = ...;

dvar float+ X[Variabili];

minimize sum(j in Variabili) C[j]*X[j];
subject to{
  forall(i in Vincoli)
  vincoli: sum(j in Variabili) A[i][j]*X[j] ==
b[i];

};
```