# IBM ILOG CPLEX

- SIMPLEX implementato in C.

# IBM ILOG CPLEX

- SIMPLEX implementato in C.
- Sono implementati anche altri metodi per trattare diversi problemi di ottimizzazione.

# IBM ILOG CPLEX

- SIMPLEX implementato in C.
- Sono implementati anche altri metodi per trattare diversi problemi di ottimizzazione.
- Licenza accademica “free”.

# IBM ILOG CPLEX

- SIMPLEX implementato in C.
- Sono implementati anche altri metodi per trattare diversi problemi di ottimizzazione.
- Licenza accademica “free”.
- Versione “Community Edition”: fino a 1000 variabili e 1000 vincoli.

# CPLEX Optimization Studio

- Ambiente integrato, costruito sulla base di Eclipse, per la risoluzione di problemi di ottimizzazione. Esso comprende:

# CPLEX Optimization Studio

- Ambiente integrato, costruito sulla base di Eclipse, per la risoluzione di problemi di ottimizzazione. Esso comprende:
  1. il linguaggio **OPL** (Optimization Programming Language).

# CPLEX Optimization Studio

- Ambiente integrato, costruito sulla base di Eclipse, per la risoluzione di problemi di ottimizzazione. Esso comprende:
  1. il linguaggio **OPL** (Optimization Programming Language).
  2. **CPLEX** per la risoluzione di problemi di ottimizzazione lineari, misti, quadratici, ecc.

# CPLEX Optimization Studio

- Ambiente integrato, costruito sulla base di Eclipse, per la risoluzione di problemi di ottimizzazione. Esso comprende:
  1. il linguaggio **OPL** (Optimization Programming Language).
  2. **CPLEX** per la risoluzione di problemi di ottimizzazione lineari, misti, quadratici, ecc.
  3. **CP** (Constraint Programming engine) per la risoluzione di problemi di scheduling e routing.

# Programmare in OPL

- Un programma in OPL consiste in una serie di istruzioni;

# Programmare in OPL

- Un programma in OPL consiste in una serie di istruzioni;
- ogni istruzione deve terminare con “;”

# Programmare in OPL

- Un programma in OPL consiste in una serie di istruzioni;
- ogni istruzione deve terminare con “;”
- è possibile inserire commenti: essi devono essere compresi fra “/*” e “*/”, oppure dopo “//” fino a fine linea;

# Programmare in OPL

- Un programma in OPL consiste in una serie di istruzioni;
- ogni istruzione deve terminare con “;”
- è possibile inserire commenti: essi devono essere compresi fra “/*” e “*/”, oppure dopo “//” fino a fine linea;
- colori del testo:
  - blu: parole-chiave
  - verde: commenti
  - nero: altro testo

# Programmare in OPL

- Un programma in OPL consiste in una serie di istruzioni;
- ogni istruzione deve terminare con “;”
- è possibile inserire commenti: essi devono essere compresi fra “/*” e “*/”, oppure dopo “//” fino a fine linea;
- colori del testo:
  - blu: parole-chiave
  - verde: commenti
  - nero: altro testo
- c’è differenza fra carattere minuscolo e maiuscolo.

# Programmare in OPL

- Un programma in OPL consiste in una serie di istruzioni;
- ogni istruzione deve terminare con “;”
- è possibile inserire commenti: essi devono essere compresi fra “/*” e “*/”, oppure dopo “//” fino a fine linea;
- colori del testo:
  - blu: parole-chiave
  - verde: commenti
  - nero: altro testo
- c’è differenza fra carattere minuscolo e maiuscolo.

# Programmare in OPL

- Un programma in OPL consiste in una serie di istruzioni;
- ogni istruzione deve terminare con “;”
- è possibile inserire commenti: essi devono essere compresi fra “/*” e “*/”, oppure dopo “//” fino a fine linea;
- colori del testo:
  - blu: parole-chiave
  - verde: commenti
  - nero: altro testo
- c’è differenza fra carattere minuscolo e maiuscolo.

# Programmare in OPL

- Un programma in OPL consiste in una serie di istruzioni;
- ogni istruzione deve terminare con “;”
- è possibile inserire commenti: essi devono essere compresi fra “/*” e “*/”, oppure dopo “//” fino a fine linea;
- colori del testo:
  - blu: parole-chiave
  - verde: commenti
  - nero: altro testo
- c’è differenza fra carattere minuscolo e maiuscolo.

# Le variabili (decisionali e non)

- I nomi di tutte le variabili devono cominciare sempre con un carattere alfabetico o underscore (seguito da caratteri alfabetici o numerici, o underscore);

# Le variabili (decisionali e non)

- I nomi di tutte le variabili devono cominciare sempre con un carattere alfabetico o underscore (seguito da caratteri alfabetici o numerici, o underscore);
- tutte le variabili devono essere dichiarate con il loro tipo;

# Le variabili (decisionali e non)

- I nomi di tutte le variabili devono cominciare sempre con un carattere alfabetico o underscore (seguito da caratteri alfabetici o numerici, o underscore);
- tutte le variabili devono essere dichiarate con il loro tipo;
- i tipi “base” per le variabili non decisionali sono: $int$, $float$ e $string$;

# Le variabili (decisionali e non)

- I nomi di tutte le variabili devono cominciare sempre con un carattere alfabetico o underscore (seguito da caratteri alfabetici o numerici, o underscore);
- tutte le variabili devono essere dichiarate con il loro tipo;
- i tipi “base” per le variabili non decisionali sono: $int$, $float$ e $string$;
- i tipi “base” per le variabili decisionali sono: $int$, $float$, $boolean$, $int+$, $float+$;

# Le variabili (decisionali e non)

- I nomi di tutte le variabili devono cominciare sempre con un carattere alfabetico o underscore (seguito da caratteri alfabetici o numerici, o underscore);
- tutte le variabili devono essere dichiarate con il loro tipo;
- i tipi "base" per le variabili non decisionali sono: $int$, $float$ e $string$;
- i tipi "base" per le variabili decisionali sono: $int$, $float$, $boolean$, $int+$, $float+$;
- una variabile decisionale viene dichiarata tale con $dvar$.

# La funzione obiettivo

- Massimizzare: $maximize$

# La funzione obiettivo

- Massimizzare: $maximize$
- minimizzare: $minimize$

# I vincoli

- *subject to { };*

# I vincoli

- *subject to { };*
- oppure *constraints { };*

# I vincoli

- *subject to { };*
- oppure *constraints { };*
- a un vincolo si può dare anche un nome: *nome_vincolo: vincolo*;

# I modelli

- Modello semplice;

# I modelli

- Modello semplice;
- modello complesso, in cui i dati sono strutturati usando insiemi, range, array e tuple, e le istruzioni *forall* e *sum*.

# Gli insiemi

- L’insieme è una collezione di elementi non indicizzati e non duplicabili;

# Gli insiemi

- L’insieme è una collezione di elementi non indicizzati e non duplicabili;
- Se *T* è un tipo, allora *{T}*, o in alternativa *setof(T)*, denota un insieme di tipo *T*.

# Gli insiemi

- L'insieme è una collezione di elementi non indicizzati e non duplicabili;
- Se *T* è un tipo, allora *{T}*, o in alternativa *setof(T)*, denota un insieme di tipo *T*.
- Esempio:
  - *{string} nomi = {"Antonio", "Marco", "Giovanni"};*
    *setof(string) nomi = {"Antonio", "Marco", "Giovanni"};*

# Gli insiemi

- L'insieme è una collezione di elementi non indicizzati e non duplicabili;
- Se *T* è un tipo, allora *{T}*, o in alternativa *setof(T)*, denota un insieme di tipo *T*.
- Esempio:
  - *{string} nomi = {"Antonio", "Marco", "Giovanni"};*
    *setof(string) nomi = {"Antonio", "Marco", "Giovanni"};*

# Operazioni sugli insiemi

- Esempi:
  - *{int} s1 = {1,2,3};*
    *{int} s2 = {1,4,5};*
    *{int} s3 = s1 inter s2;*
    *{int} s4 = {1,4,8,10} inter s2;*
    *{int} s5 = s1 union {5,7,9};*
    *{int} s6 = s1 diff s2;*
    *{int} s7 = s1 symdiff {1,4,5};*

# Operazioni sugli insiemi

- Esempi:
  - *{int} s1 = {1,2,3};*
    *{int} s2 = {1,4,5};*
    *{int} s3 = s1 inter s2;*
    *{int} s4 = {1,4,8,10} inter s2;*
    *{int} s5 = s1 union {5,7,9};*
    *{int} s6 = s1 diff s2;*
    *{int} s7 = s1 symdiff {1,4,5};*

# Range e range float

- Un range è utile per:
  - indicizzare un array;
  - specificare il campo di valori (lower bound e upper bound) delle variabili decisionali ;
  - indicizzare i vincoli quando si usa l'istruzione *forall*.

# Range e range float

- Un range è utile per:
  - indicizzare un array;
  - specificare il campo di valori (lower bound e upper bound) delle variabili decisionali ;
  - indicizzare i vincoli quando si usa l'istruzione *forall*.

# Range e range float

- Un range è utile per:
  - indicizzare un array;
  - specificare il campo di valori (lower bound e upper bound) delle variabili decisionali ;
  - indicizzare i vincoli quando si usa l'istruzione *forall*.

# Range e range float

- Un range è utile per:
  - indicizzare un array;
  - specificare il campo di valori (lower bound e upper bound) delle variabili decisionali ;
  - indicizzare i vincoli quando si usa l'istruzione *forall*.

# Range e range float

- La sintassi dell'istruzione range è: *range nome_range = l..u.*

# Range e range float

- La sintassi dell'istruzione range è: *range nome_range = l..u.*
- Esempi:
  - *range R = 1..100; dvar int i in R;*
  - *range R = 1..100; forall (i in R) vincolo;*

# Range e range float

- La sintassi dell'istruzione range è: *range nome_range = l..u.*
- Esempi:
  - *range R = 1..100; dvar int i in R;*
  - *range R = 1..100; forall (i in R) vincolo;*
- La sintassi dell'istruzione range float è: *range float nome_range = l..u.*

# Range e range float

- La sintassi dell'istruzione range è: *range nome_range = l..u.*
- Esempi:
  - *range R = 1..100; dvar int i in R;*
  - *range R = 1..100; forall (i in R) vincolo;*
- La sintassi dell'istruzione range float è: *range float nome_range = l..u.*
- Esempio:
  - *range float X = 1.0..100.0; dvar float x in X;*

# Range e range float

- La sintassi dell'istruzione range è: *range nome_range = l..u.*
- Esempi:
  - *range R = 1..100; dvar int i in R;*
  - *range R = 1..100; forall (i in R) vincolo;*
- La sintassi dell'istruzione range float è: *range float nome_range = l..u.*
- Esempio:
  - *range float X = 1.0..100.0; dvar float x in X;*

# Range e range float

- La sintassi dell'istruzione range è: *range nome_range = l..u.*
- Esempi:
  - *range R = 1..100; dvar int i in R;*
  - *range R = 1..100; forall (i in R) vincolo;*
- La sintassi dell'istruzione range float è: *range float nome_range = l..u.*
- Esempio:
  - *range float X = 1.0..100.0; dvar float x in X;*

# Range e range float

- La sintassi dell'istruzione range è: *range nome_range = l..u.*
- Esempi:
  - *range R = 1..100; dvar int i in R;*
  - *range R = 1..100; forall (i in R) vincolo;*
- La sintassi dell'istruzione range float è: *range float nome_range = l..u.*
- Esempio:
  - *range float X = 1.0..100.0; dvar float x in X;*

# Array monodimensionali

- Esempi:
  - *int a[1..4] = [10, 20, 30, 40];*
  - *float f[1..4] = [1.2, 2.3, 3.4, 4.5];*
  - *string d[1..2] = [”Antonio”, ”Giovanni”];*
  - *range R = 1..100; int A[R];*
  - *range R = 1..100; dvar float+ z[R];*
  - NOTA: un array può anche essere indicizzato da stringhe o anche da tuple.

# Array monodimensionali

- Esempi:
  - *int a[1..4] = [10, 20, 30, 40];*
  - *float f[1..4] = [1.2, 2.3, 3.4, 4.5];*
  - *string d[1..2] = [”Antonio”, ”Giovanni”];*
  - *range R = 1..100; int A[R];*
  - *range R = 1..100; dvar float+ z[R];*
  - NOTA: un array può anche essere indicizzato da stringhe o anche da tuple.

# Array monodimensionali

- Esempi:
  - *int a[1..4] = [10, 20, 30, 40];*
  - *float f[1..4] = [1.2, 2.3, 3.4, 4.5];*
  - *string d[1..2] = [”Antonio”, ”Giovanni”];*
  - *range R = 1..100; int A[R];*
  - *range R = 1..100; dvar float+ z[R];*
  - NOTA: un array può anche essere indicizzato da stringhe o anche da tuple.

# Array monodimensionali

- Esempi:
  - *int a[1..4] = [10, 20, 30, 40];*
  - *float f[1..4] = [1.2, 2.3, 3.4, 4.5];*
  - *string d[1..2] = [”Antonio”, ”Giovanni”];*
  - *range R = 1..100; int A[R];*
  - *range R = 1..100; dvar float+ z[R];*
  - NOTA: un array può anche essere indicizzato da stringhe o anche da tuple.

# Array monodimensionali

- Esempi:
  - *int a[1..4] = [10, 20, 30, 40];*
  - *float f[1..4] = [1.2, 2.3, 3.4, 4.5];*
  - *string d[1..2] = [”Antonio”, ”Giovanni”];*
  - *range R = 1..100; int A[R];*
  - *range R = 1..100; dvar float+ z[R];*
  - NOTA: un array può anche essere indicizzato da stringhe o anche da tuple.

# Array multidimensionali

- Esempio:
  - *float cc[1..4][1..4] = [[0, 150, 400, 300], [150, 0, 250, 200], [400, 250, 0, 200], [300, 200, 200, 0]];*

# Array multidimensionali

- Esempio:
  - *float cc[1..4][1..4] = [[0, 150, 400, 300], [150, 0, 250, 200], [400, 250, 0, 200], [300, 200, 200, 0]];*

# Inizializzazione di array

- Esempi:
  - modalità ordinaria:
    *int a[1..2][1..3] = [ [10, 20, 30], [40, 50, 60] ];*
  - modalità coppia indice-valore (ordine arbitrario, solo con file .dat):
    *\/* .mod file */*
    *{string} giorni = {lun, mar,mer,gio,ven,sab,dom};*
    *int a[giorni] = ...;*
    *\/* .dat file */*
    *a = #[ mar: 2, lun: 1, mer: 3, gio: 4, ven: 5, sab: 6, dom: 7 ] #;*

# Inizializzazione di array

- Esempi:
  - modalità ordinaria:
    *int a[1..2][1..3] = [ [10, 20, 30], [40, 50, 60] ];*
  - modalità coppia indice-valore (ordine arbitrario, solo con file .dat):
    *\/* .mod file */*
    *{string} giorni = {lun, mar,mer,gio,ven,sab,dom};*
    *int a[giorni] = ...;*
    *\/* .dat file */*
    *a = #[ mar: 2, lun: 1, mer: 3, gio: 4, ven: 5, sab: 6, dom: 7 ] #;*

# Inizializzazione di array

- Esempi:
  - modalità ordinaria:
    *int a[1..2][1..3] = [ [10, 20, 30], [40, 50, 60] ];*
  - modalità coppia indice-valore (ordine arbitrario, solo con file .dat):
    *\/* .mod file */*
    *{string} giorni = {lun, mar,mer,gio,ven,sab,dom};*
    *int a[giorni] = ...;*
    *\/* .dat file */*
    *a = #[ mar: 2, lun: 1, mer: 3, gio: 4, ven: 5, sab: 6, dom: 7 ] #;*

# Inizializzazione di array

- Esempi:
  - modalità combinata:
    *\/* .mod file *\/*
    *int a[1..2][1..3] = ...;*
    *\/* .dat file *\/*
    *a = #[ 2: [40, 50, 60], 1: [10, 20, 30] ]#;*

# Inizializzazione di array

- Esempi:
  - modalità combinata:
    *\/* .mod file *\/*
    *int a[1..2][1..3] = ...;*
    *\/* .dat file *\/*
    *a = #[ 2: [40, 50, 60], 1: [10, 20, 30] ]#;*

# Array generici indicizzati

- Esempi:
  - *int a[i in 1..10] = i+1;*
  - *int m[i in 0..10][j in 0..10] = 10*i + j;*
  - */* t = trasposta di m */*
    *int m[Dim1][Dim2] = ...;*
    *int t[j in Dim2][i in Dim1] = m[i][j];*

# Array generici indicizzati

- Esempi:
  - *int a[i in 1..10] = i+1;*
  - *int m[i in 0..10][j in 0..10] = 10*i + j;*
  - */* t = trasposta di m */*
    *int m[Dim1][Dim2] = ...;*
    *int t[j in Dim2][i in Dim1] = m[i][j];*

# Array generici indicizzati

- Esempi:
  - *int a[i in 1..10] = i+1;*
  - *int m[i in 0..10][j in 0..10] = 10*i + j;*
  - */* t = trasposta di m */*
    *int m[Dim1][Dim2] = ...;*
    *int t[j in Dim2][i in Dim1] = m[i][j];*

# Array generici indicizzati

- Esempi:
  - *int a[i in 1..10] = i+1;*
  - *int m[i in 0..10][j in 0..10] = 10*i + j;*
  - */* t = trasposta di m */*
    *int m[Dim1][Dim2] = ...;*
    *int t[j in Dim2][i in Dim1] = m[i][j];*

# Array generici indicizzati

- Esempi:
  - *int a[i in 1..10] = i+1;*
  - *int m[i in 0..10][j in 0..10] = 10*i + j;*
  - */* t = trasposta di m */*
    *int m[Dim1][Dim2] = ...;*
    *int t[j in Dim2][i in Dim1] = m[i][j];*

# sum

- Effettua una sommatoria;

# sum

- Effettua una sommatoria;
- Sintassi: *sum* (qualificatori) espressione;

# sum

- Effettua una sommatoria;
- Sintassi: *sum* (qualificatori) espressione;
- Esempio:

```
int n = 10;
range Range = 0..n-1;
dvar int s[Range];
subject to
{
sum(i in Range) s[i] == n;
sum(i in Range) s[i]*i == n;
}
```

# forall

- Consente di scrivere un gruppo di vincoli;

# forall

- Consente di scrivere un gruppo di vincoli;
- Sintassi: *forall* (qualificatori) vincolo;

# forall

- Consente di scrivere un gruppo di vincoli;
- Sintassi: *forall* (qualificatori) vincolo;
- Esempio:

```
int n=8;
dvar int a[1..n][1..n];
subject to
{
forall(i in 1..8)
forall(j in 1..8: i < j)
a[i][j] >= 0;
}
```

# Assert

- Consente di verificare alcune assunzioni.

# Assert

- Consente di verificare alcune assunzioni.
- Sintassi: *assert* espressione;

# Le tuple

- Una tupla accomuna dati fra loro correlati.
  Ciascun dato della tupla corrisponde a un campo.
  Dichiarando una tupla, si definisce un nuovo tipo.

# Le tuple

- Una tupla accomuna dati fra loro correlati.
  Ciascun dato della tupla corrisponde a un campo.
  Dichiarando una tupla, si definisce un nuovo tipo.
- Esempio:
  - *tuple Punto {*
    *int x;*
    *int y;*
    *}*
    *Punto p = <2,3>;*
    *assert p.x==2;*
    *Punto punto[1..3] = [<1,2>, <2,3>, <3,4>];*
    *Punto punti = {<1,2>, <2,3>};*
    *tuple Rettangolo {*
    *Punto ll;*
    *Punto ur;*

# Le tuple

- Una tupla accomuna dati fra loro correlati.
  Ciascun dato della tupla corrisponde a un campo.
  Dichiarando una tupla, si definisce un nuovo tipo.
- Esempio:
  - *tuple Punto {*
    *int x;*
    *int y;*
    *}*
    *Punto p = <2,3>;*
    *assert p.x==2;*
    *Punto punto[1..3] = [<1,2>, <2,3>, <3,4>];*
    *Punto punti = {<1,2>, <2,3>};*
    *tuple Rettangolo {*
    *Punto ll;*
    *Punto ur;*

# Inizializzazione di tuple

- Esempi:
  - modalità ordinaria:
    *tuple rettangolo {*
    *int id;*
    *int x[1..2];*
    *int y[1..2];*
    *}*
    *rettangolo r = <1, [0,10], [0,10]>;*

# Inizializzazione di tuple

- Esempi:
  - modalità ordinaria:
    *tuple rettangolo {*
    *int id;*
    *int x[1..2];*
    *int y[1..2];*
    *}*
    *rettangolo r = <1, [0,10], [0,10]>;*

# Inizializzazione di tuple

- Esempi:
  - modalità coppia campo-valore (ordine arbitrario, solo con file .dat):

```
/* .mod file */
tuple point
{
int x;
int y;
}
point p1=...;
point p2=...;
/* .dat file */
p1=#<y:1,x:2>#;
p2=<2,1>;
```

# Inizializzazione di tuple

- Esempi:
  - modalità coppia campo-valore (ordine arbitrario, solo con file .dat):

```
/* .mod file */
tuple point
{
int x;
int y;
}
point p1=...;
point p2=...;
/* .dat file */
p1=#<y:1,x:2>#;
p2=<2,1>;
```